Institut für integrierte
Naturwissenschaften
Abteilung Physik
Universität Koblenz-Landau
Universitätsstraße 1
56070 Koblenz

Studienarbeit

Taktstraße

Betreuer:
Dr. Merten Joost

Arbeitsgruppe:	
Andreas Schmidt	203110082
Norman Böttcher	203110032

Inhalte des Projekts:

Ansteuerung einer Taktstraße mit Hilfe des Mikrocontrollers ATMega16. Entwicklung der dazugehörigen Platinen. Bedienung und Kommunikation durch einen weiteren ATMega16 über den TWI-Bus. Nutzen einer entwickelten LCD-Bibliothek zur Ausgabe von benutzerorientierten Informationen.

Erklärung

Wir versichern, dass wir die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek sind wir einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimmen wir zu.	☐	☐

. . . .

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

1 Vorwort **1**
1.1 Zielsetzung . . . 1
1.2 Motivation . . . 1

2 Grundlagen **3**
2.1 Die Taktstraße . . . 3
2.1.1 Beschreibung der Taktstraße . . . 4
2.2 Software . . . 7
2.2.1 WinAVR . . . 7
2.2.2 Flash . . . 7
2.2.3 AVRStudio4 . . . 8
2.2.4 AVRTerminal . . . 9
2.2.5 AVR-Dude . . . 9
2.2.6 Eagle . . . 10
2.3 Das Herstellen von Platinen . . . 11
2.3.1 Erstellung eines Schaltplans . . . 11
2.3.2 Entwerfen eines Layouts . . . 12
2.3.3 Fertigung einer Platine . . . 14
2.4 Grundlegende Gedanken zur Implementation . . . 16
2.4.1 Grundlagen zum AT-Mega16 . . . 17
2.4.2 Grundlagen zum USART . . . 18
2.4.3 Grundlagen zum AD-Wandler . . . 20
2.4.4 Grundlagen zur Display-Ansteuerung . . . 22
2.4.5 Grundlagen zum Two-Wire Serial Interface . . . 25

3 Entwicklung **33**
3.1 Analyse der Taktstraße . . . 33
3.2 Belegung des AT-Mega16 . . . 34
3.3 Entwicklung der Hardware . . . 35
3.3.1 Schaltpläne . . . 35
3.3.2 Platinen entwerfen und herstellen . . . 41
3.3.3 Verkabeln des Displays . . . 44
3.4 Entwicklung der Software . . . 46
3.4.1 USART - Universal Synchronous and Asynchronous serial Receiver and Transmitter . . . 47
3.4.2 AD-Wandler . . . 48

3.4.3 Display 49
3.4.4 TWI - Kommunikationsspezialisierung 58
3.4.5 TWI - Two-Wire Interface (Slave) 59
3.4.6 TWI - Two-Wire Interface (Master) 64
3.4.7 Hauptprogramm und Funktionen 66
3.5 Ansteuerung der Taktstraße 69

4 Fazit **77**
4.1 Erreichte Ziele 77
4.2 Einige Gedanken 77
4.3 Ein Ausblick 78

5 Literaturverzeichnis **79**

Index **81**

A Anhang **83**
A.1 Platinen 83
A.1.1 Prozessorplatine 83
A.1.2 Anschlussplatine 85
A.1.3 Kontrollplatine 87
A.1.4 Finale Ein-Platinen-Lösung 89
A.2 HWP-Versuch 10 91
A.3 Sourcecode 98
A.3.1 TWI-Slave (Taktstraße) 98
A.3.2 TWI-Master (HWP-Board) 114

Abbildungsverzeichnis

2.1 Modellabbildung der Taktstraße 51 664A 3
2.2 Schieber mit Motor der Taktstraße 5
2.3 Aufsicht auf das Zuführband, Motoren, Schieber, Fototransistoren 5
2.4 Motor am Auslagerband 5
2.5 Linke Bearbeitungsstation mit Motor 6
2.6 Rechte Bearbeitungsstation mit Motor 6
2.7 Flash-Tool 8
2.8 AVR-Studio 4 8
2.9 AVR-Terminal 9
2.10 AVR-Dude 10
2.11 Eagle 10
2.12 Beispielausschnitt aus dem Platinenschaltplan 11
2.13 Ansicht Layout ohne geroutete Leiterbahnen 12
2.14 Ansicht Layout mit gerouteten Leiterbahnen 13
2.15 Fertiges Layout der Platine 13
2.16 Eingesprühte Platine mit Layoutausdruck 14
2.17 Entwicklung einer Platine im Natriumbad 15
2.18 Eintauchen der Platine in ein Ätzbad 15
2.19 Platine im Ätzbad ändert ihre Farbe 16
2.20 Pinbelegung AT-Mega 16 17
2.21 USART: Register UDR 19
2.22 USART: Register **UBRR** 19
2.23 USART: Register **UCSRB** 19
2.24 USART: Register **UCSRC** 19
2.25 USART: Register *UCSRA* 20
2.26 Multiplexer-Auswahl-Register **ADMUX** 20
2.27 Steuer- und Statusregister **ADCSRA** 20
2.28 Daten-Register **ADCL** und **ADCH** 21
2.29 Ablauf einer Wandlung 22
2.30 Belegung LCD 22
2.31 Kontrast LCD-Display 23
2.32 Befehlsübersicht zur Ansteuerung des LCD-Displays 24
2.33 Initialisierungsequenz des LCD-Displays 24
2.34 TWI Bit-Rate Register 27
2.35 Formel zur Errechnung der Takt-Frequenz 28

2.36 TWI (Slave) Adress-Register 28
2.37 TWI Status-Register 28
2.38 TWI Control-Register 29
2.39 TWI Data-Register 30
2.40 Ablauf einer Kommunikation 31
2.41 Ablauf eines Sendevorganges 31
2.42 Flussdiagramm einer Kommunikation 32

3.1 Prozessorplatine 36
3.2 Kontrollplatine 38
3.3 Anschlussplatine 40
3.4 Prozessorplatine 42
3.5 Platine zwischen Taktstraße und Prozessor 43
3.6 Kontrollplatine mit allen Anschlüssen 43
3.7 Verbindung aller drei Platinen 44
3.8 Frontansicht LCD-Display 45
3.9 Rückansicht LCD-Display 45
3.10 Rückansicht nah LCD-Display 46
3.11 TWI-Kommunikationsablauf: Senden 58
3.12 TWI-Kommunikationsablauf: Empfangen 59
3.13 Reaktion des Slaves auf ein Datum 61
3.14 Flußdiagramm der Taktstraße 69

A.1 Schaltplan der Prozessorplatine 83
A.2 Boardansicht der Prozessorplatine 84
A.3 Schaltplan der Anschlussplatine 85
A.4 Boardansicht der Anschlussplatine 86
A.5 Schaltplan der Kontrollplatine 87
A.6 Boardansicht der Kontrollplatine 88
A.7 Schaltplan der finalen Ein-Platinen-Lösung 89
A.8 Boardansicht der finalen Ein-Platinen-Lösung 90
A.9 Modellabbildung der Taktstraße 51 664A 91
A.10 TWI-Kommunikationsablauf: Senden 93
A.11 TWI-Kommunikationsablauf: Empfangen 94
A.12 Reaktion des Slaves auf ein Datum 96

1 Vorwort

Im Rahmen des Informatikstudiums an der Universität Koblenz wird im Verlauf des Hauptstudiums von den Studierenden eine Studienarbeit erarbeitet. Diese gilt als Voraussetzung zum Erlangen des Diploms und wird von den Studierenden selbstständig unter Betreuung angefertigt.

1.1 Zielsetzung

Ziel unserer Studienarbeit war es eine Ansteuerung für eine vorgegebene Taktstraße zu entwickeln. Eine Taktstraße ermöglicht eine automatisierte Verarbeitung eines Werkstücks mit Hilfe von Förderbändern, Lichtschranken, Schiebern und Bearbeitungsstationen. [1]

Eine Vorgabe für die Realisierung dieses Projektes war die Verwendung des Mikrocontrollers ATMega16 von Atmel, der die Ansteuerung der Taktstraße verwalten sollte. Ein externer Mikrocontroller muss in der Lage sein können, Steuerbefehle über den TWI-Bus[2] an den mit der Taktstraße verbundenen Controller zu senden, um diese sinnvoll agieren und reagieren zu lassen. Um die Taktstraße bedienbar zu machen, sollte eine geeignete Platine enwtworfen werden, die fest mit ihr verbunden ist.

Die Lösung sollte alle steuerbaren Elemente und Zustände der Taktstraße geeignet ansprechen und auslesen. Ein LCD-Display dient dabei als Informationsmedium der Taktstraße über den internen Zustand und des Weiteren als Möglichkeit Nachrichten des Benutzers auszugeben.

1.2 Motivation

Da in unserem bisherigen Bildungsweg die hardwarenahe Praxis oft im Hintergrund stand, jedoch bei uns auf reges Interesse stieß, bemühten wir uns um eine Studienarbeit, die uns neben einem theoretischen Teil die Möglichkeit eröffnen würde, praxisbezogen zu arbeiten.

[1] Eine beispielhafte Abbildung findet sich in den Grundlagen.
[2] Ermöglicht eine Buskommunikation über eine Takt- und eine Datenleitung.

Dieses Projekt soll nach Abschluss den Studenten der Informatik zur Lehre im Hardwarepraktikum zur Verfügung gestellt werden, in dem die Studenten praxisnahen Umgang mit der Digitalelektronik erlernen.

Diese Aufgabenstellung deckt alle für ein Projekt benötigten Entwicklungsstadien ab. Darunter fällt die Projektplanung, Aneignung von spezifichem Grundlagenwissen, Software- und Hardwareentwicklung, sowie mehrere ausführliche Entwicklungs- und Testphasen.

2 Grundlagen

In diesem Kapitel soll im Folgenden ein kurzer Überblick über benötigte Grundlagen im Bereich Softwareerstellung, Platinenentwicklung und Bus-Kommunikation gegeben werden. Es werden also kurz die Werkzeuge und natürlich evtl. Arbeitsschritte erläutert [Sch06].

2.1 Die Taktstraße

Die von uns angesteuerte Taktstraße ist ein Modell der Firma **Fischer Technik** und ermöglicht über eine Ansteuerung die Modellverarbeitung eines Werkstücks in mehreren Arbeitsschritten. Unter Verwendung von Motoren und Lichtschranken ist bei korrekter Ansteuerung ein kompletter Durchlauf des Werkstücks durch die Verarbeitungsstraße möglich. Die Motoren steuern unter Anderem Bänder an, die in der Lage sein müssen ein Werkstück vorwärts und rückwärts zu befördern.

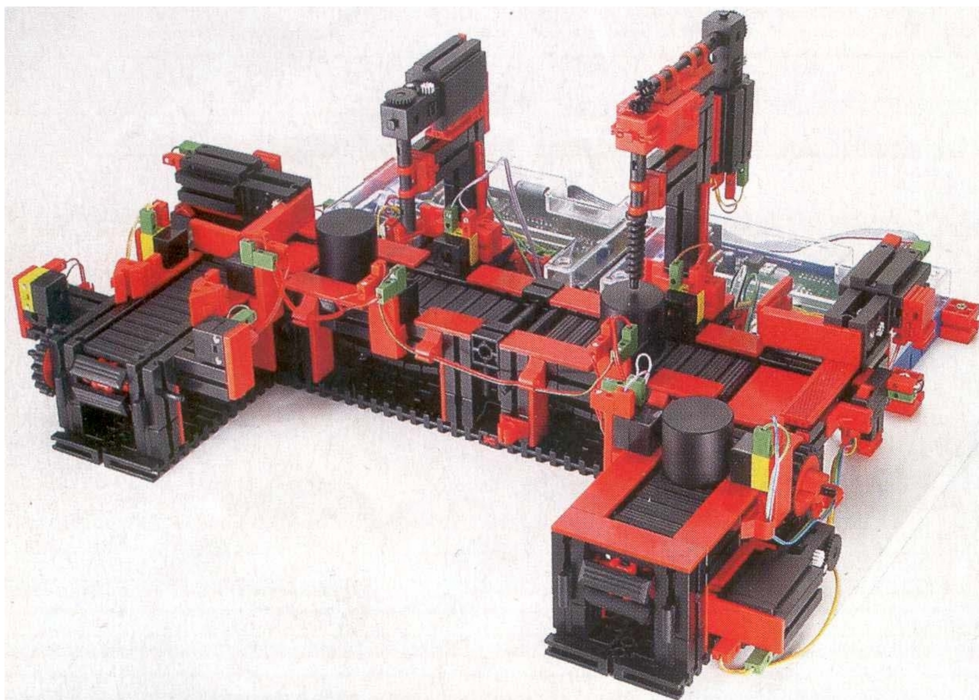


Abbildung 2.1: Modell 51 664A, [Mül]

2.1.1 Beschreibung der Taktstraße

Die von uns verwendete Taktstraße ist auf einer Grundplatte mit den Maßen 45 x 41 cm angebracht und wird mit einer Spannung von 9V betrieben.

Weitere Eigenschaften der Taktstraße:

- 2 Bearbeitungsstationen
  - Fräser
  - Bohrer
- 4 Transportbänder (U-förmig angeordnet)
  - Zuführband
  - Transportband Fräser
  - Transportband Bohrer
  - Auslagerband
- 8 Motoren
  - 4 Bändermotoren
  - 2 Motoren der Bearbeitungsstationen
  - 2 Schiebermotoren
- 4 Taster
  - 2 Taster beim Schieber vor den Bearbeitungsstationen
  - 2 Taster beim Schieber nach den Bearbeitungsstationen
- 5 Lichtschranken (mit Fototransistor und Linsenlampe)
  - Start und Ende Zuführband
  - Bearbeitung Fräser und Bohrer
  - Ende Auslagerband

Nachfolgend einige Detailbilder der Taktstraße. Das erste Bild zeigt einen Schieber, das Zweite den Bereich des Zuführbands, das Dritte den Motor am Auslagerband und das Vierte und Fünfte die Bearbeitungsstationen.

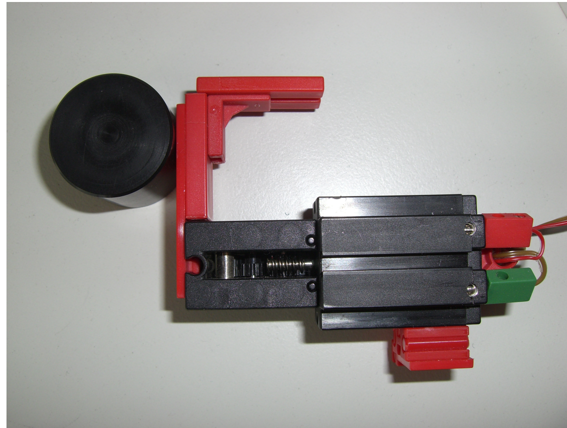


Abbildung 2.2: Schieber mit Motor der Taktstraße

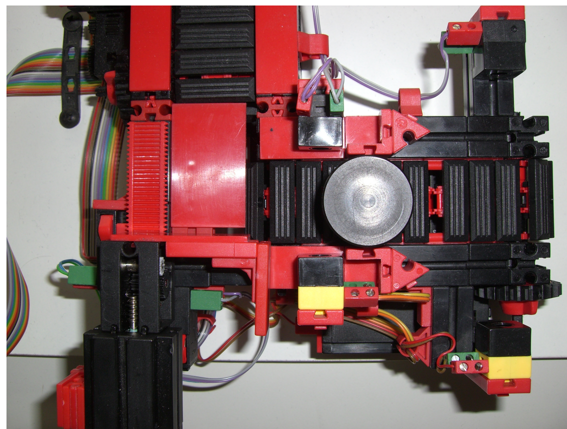


Abbildung 2.3: Aufsicht auf das Zuführband, Motoren, Schieber, Fototransistoren

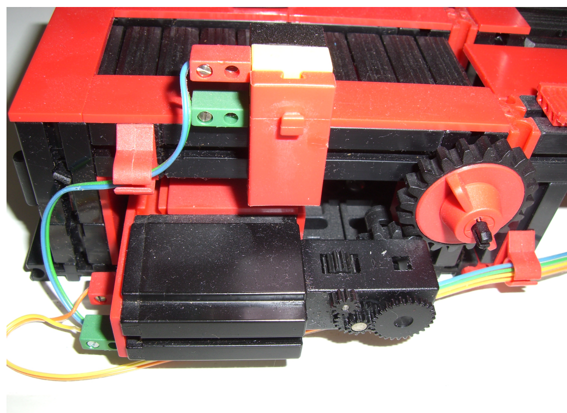


Abbildung 2.4: Motor am Auslagerband

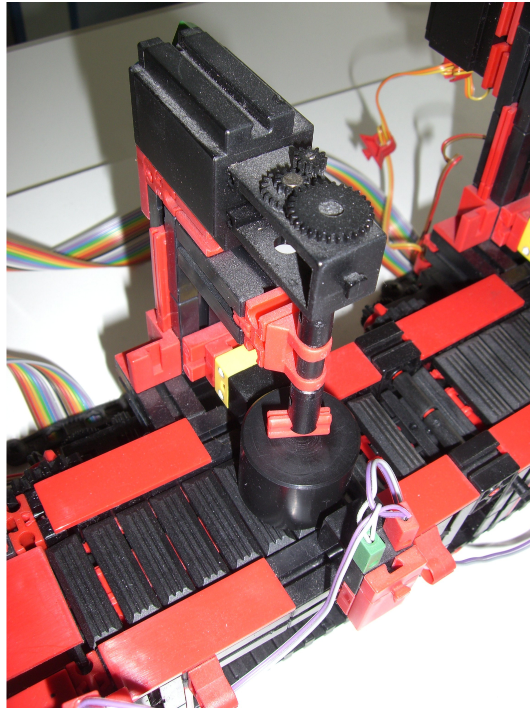


Abbildung 2.5: Linke Bearbeitungsstation mit Motor

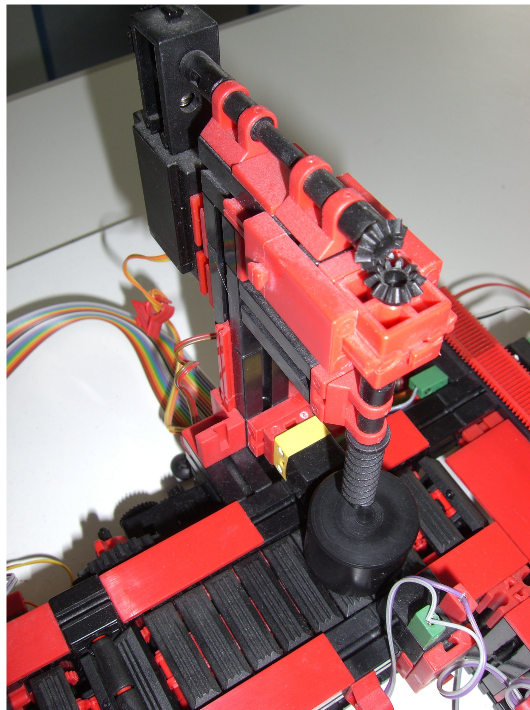


Abbildung 2.6: Rechte Bearbeitungsstation mit Motor

2.2 Software

Zur Realisierung des Projektes wurde folgende Software verwendet:

- WinAVR
- Flash
- AVRStudio4
- AVRTerminal
- AVRDude
- Eagle

Auf die Funktionalitäten der einzelnen Werkzeuge wird nun kurz eingegangen.

2.2.1 WinAVR

WinAVR[1] enthält die Windows C und C++ Compiler der Compiler-Collection AVR-GCC für AVR-Mikrocontroller. Die Distribution enthält außerdem:

- Die C-Standardbibliothek avr-libc
- AVRDude
- Programmer's Notepad.

2.2.2 Flash

Flash [2] wird benötigt um einen ATMega16 in einem speziellen HWP-Board[3] zu beschreiben. Diese Mikrocontroller besitzen einen eigens implementierten Bootloader, der mit dem hier vorgestellten Programm beschrieben wird. Man sendet wie bei Avr-Dude einen kompilierten Quellcode (.hex-Datei) an den Mikrocontroller. Die Daten werden dabei über einen seriellen Port übertragen.

[1] http://winavr.sourceforge.net/
[2] http://www.uni-koblenz.de/~physik/informatik/hwp/flash/
[3] Entwicklung der Universität

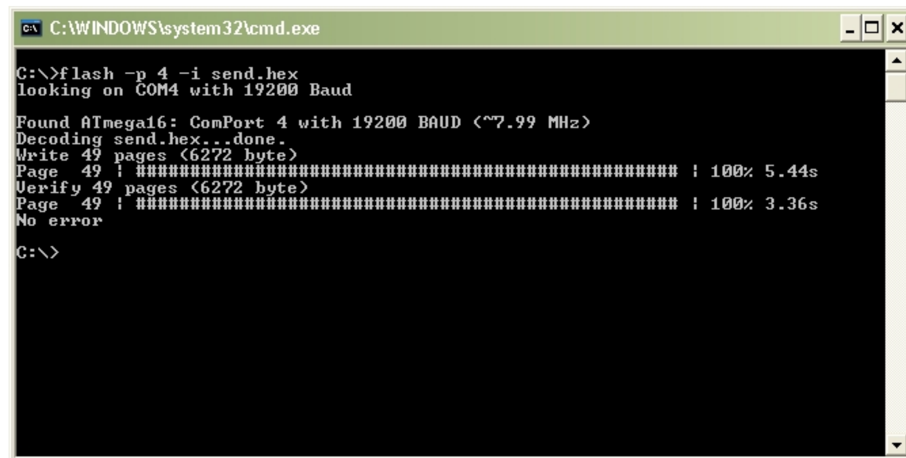


Abbildung 2.7: Flash-Tool

2.2.3 AVRStudio4

AVRStudio4[4] ist eine Entwicklungsumgebung um Software mit dem beiliegenden Editor zu schreiben, zu kompilieren, debuggen und zu simulieren. Man benötigt wie bei anderen IDE's keine Kommandozeile mehr um das Projekt zu kompilieren, sondern benutzt für alle relevanten Operationen die eingebauten Schaltflächen. Als Compiler wird der des AVR-GCC-Paketes verwendet, welcher beispielsweise im oben angesprochenen Paket WinAVR bereit gestellt wird.

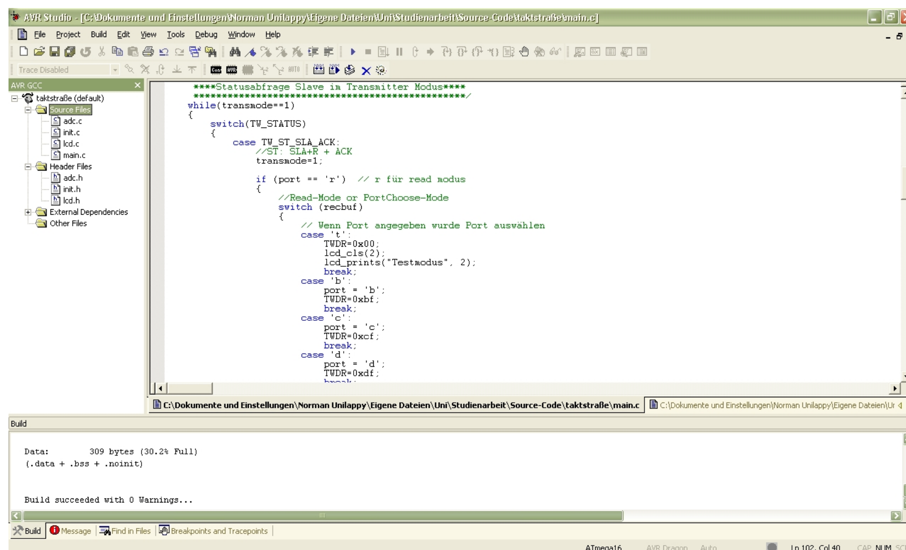


Abbildung 2.8: AVR-Studio 4

[4]http://www.atmel.com/dyn/Products/tools_card.asp?tool_id=2725

2.2.4 AVRTerminal

AVRTerminal[5] ist ein einfaches Programm um über eine serielle Schnittstelle Daten zu senden und zu empfangen. Dieses Programm wurde von uns im Wesentlichen dazu genutzt, Debugausgaben über die USART des ATMega 16 zu senden um eine direkte Rückmeldung des Mikrocontrollers zu bekommen.

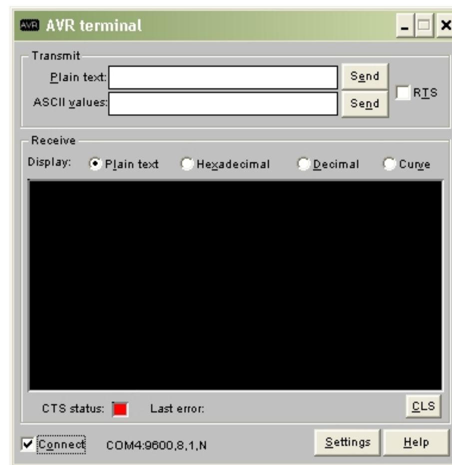


Abbildung 2.9: AVR-Terminal

2.2.5 AVR-Dude

AVRDude[6] ist ein bekanntes Werkzeug um einen Mikrocontroller wie den von uns verwendeten ATMega16 zu beschreiben, bzw. zu programmieren. Zur Verwendung benutzen wir einen parallelen Port am Computer.

Wichtigste Funktionen im Überblick [Mik]:

- Beschreiben des Mikrocontrollers mit einer kompilierten .hex-Datei
- Auslesen und setzen der Fuse- und Lockbits
- Schreiben und Lesen des EEPROMS
- Ungeschützten Code aus dem Flash auslesen

[5] http://bray.velenje.cx/avr/terminal
[6] http://www.nongnu.org/avrdude/

```
C:\WINDOWS\system32\cmd.exe - avrdude -p m16 -c sp12 -P lpt1 -U flash:w:taktstraße.he...
C:\Dokumente und Einstellungen\Norman Unilappy\Eigene Dateien\Uni\Studienarbeit\
Source-Code\taktstraße\default>avrdude -p m16 -c sp12 -P lpt1 -U flash:w:taktstr
aße.hex:i -E noreset

avrdude: AVR device initialized and ready to accept instructions

Reading | ################################################## | 100% 0.01s

avrdude: Device signature = 0x1e9403
avrdude: NOTE: FLASH memory has been specified, an erase cycle will be performed
         To disable this feature, specify the -D option.
avrdude: erasing chip
avrdude: reading input file "taktstra▀e.hex"
avrdude: writing flash (7396 bytes):

Writing | ####                                               | 8% 0.38s
```

Abbildung 2.10: AVR-Dude

2.2.6 Eagle

Eagle[7][8] ist eine Software, um Schaltpläne und zugehörige Platinen schnell und einfach zu entwickeln. Hat man einen Schaltplan erstellt, generiert Eagle automatisch eine Platine, deren Layout man selbst festlegen kann. Eagle bietet desweiteren die Möglichkeit, Leiterbahnen mit Hilfe eines Autorouters zu verlegen - sowohl einseitig, als auch zweiseitig mit vielen Einstellmöglichkeiten. In der Freeware-Version ist die Größe der Platine auf 100 x 80 mm beschränkt und es können nur zwei Signal-Lagen verwendet werden.

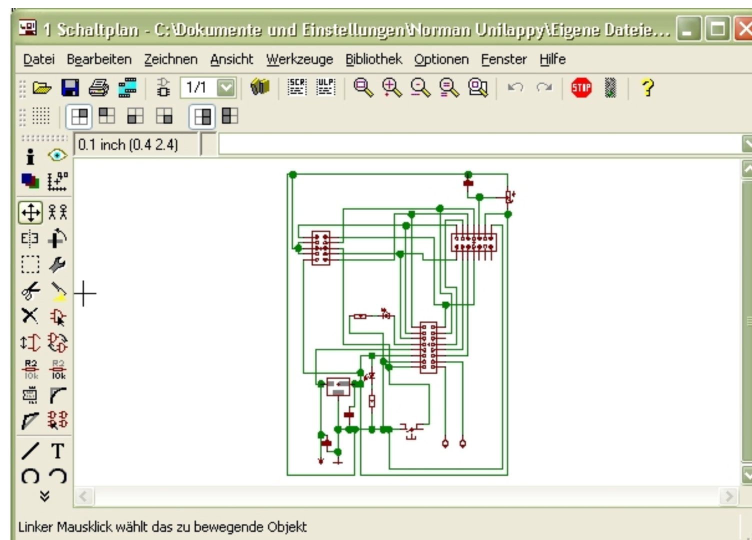


Abbildung 2.11: Eagle

[7] http://http://www.cadsoft.de/
[8] **E**infach **A**nzuwendender **G**rafischer **L**ayout **E**ditor

2.3 Das Herstellen von Platinen

Damit unsere Softwarelösung auch in Verbindung mit der Taktstraße zum Einsatz kommen konnte, bestand ein Teil unserer Aufgabe darin, eine geeignete Platine zu erstellen. Dies geschah in drei Schritten [Wil06]. Zuerst wurde mit dem Programm EAGLE ein Schaltplan entworfen. Anschließend wurden mit Hilfe eines Layout Managers die Bauteile und Leiterbahnen angeordnet. Der fertige Platinenentwurf musste dann nur noch entwickelt werden. Hier die Arbeitsschritte nochmal im einzelnen:

2.3.1 Erstellung eines Schaltplans

Zum Gestalten eines Schaltplans wählten wir das Programm EAGLE. Hier konnten wir komfortabel unsere gewünschte Platine erarbeiten. Die Anordnung der Leiterbahnen muss hier noch nicht berücksichtigt werden. Dies geschieht erst beim Entwerfen des Layouts. Zu diesem Arbeitsschritt gehört die Auswahl an Bauteilen, die korrekte Vernetzung aller Bauteile und die Einbindung der Erdung und Versorgungsspannung in das Schaltnetz.

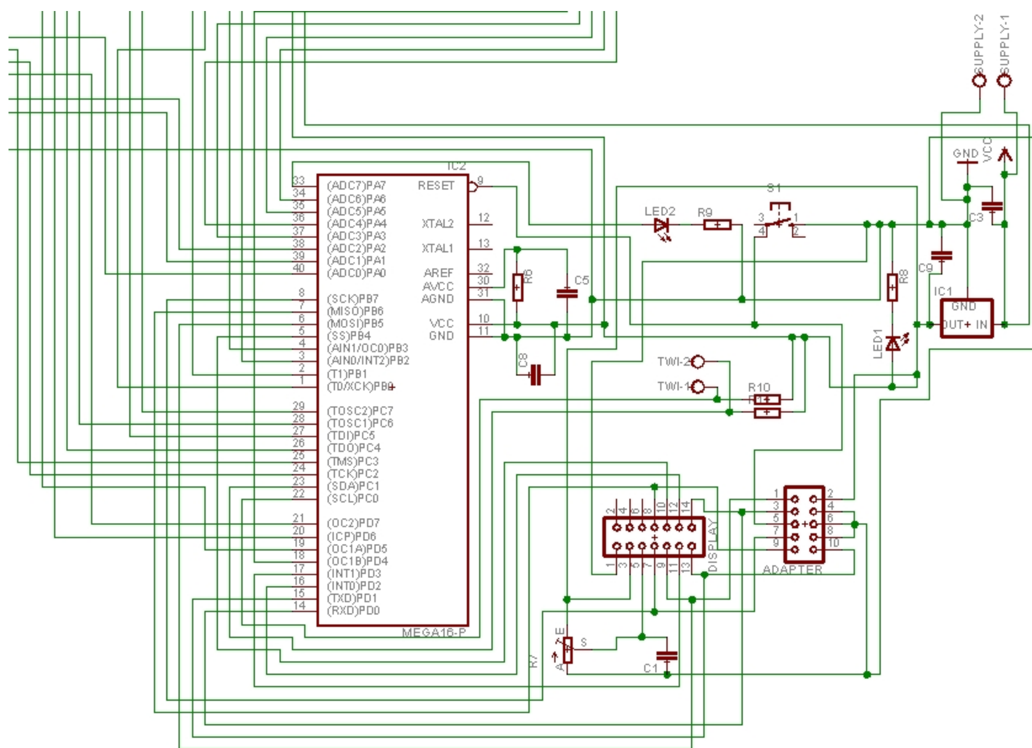


Abbildung 2.12: Beispielausschnitt aus dem Platinenschaltplan

2.3.2 Entwerfen eines Layouts

- **Anordnen der Bauteile:**

  Mit EAGLE kann man, nachdem man das Schaltnetz erstellt hat, zur Erstellung des Layouts wechseln. Die erste Aufgabe sieht vor, die Bauteile auf der Platine sinnvoll anzuordnen. Dabei ist zu beachten, dass Kondensatoren möglichst nahe an den zugehörigen Bauteilen platziert werden.

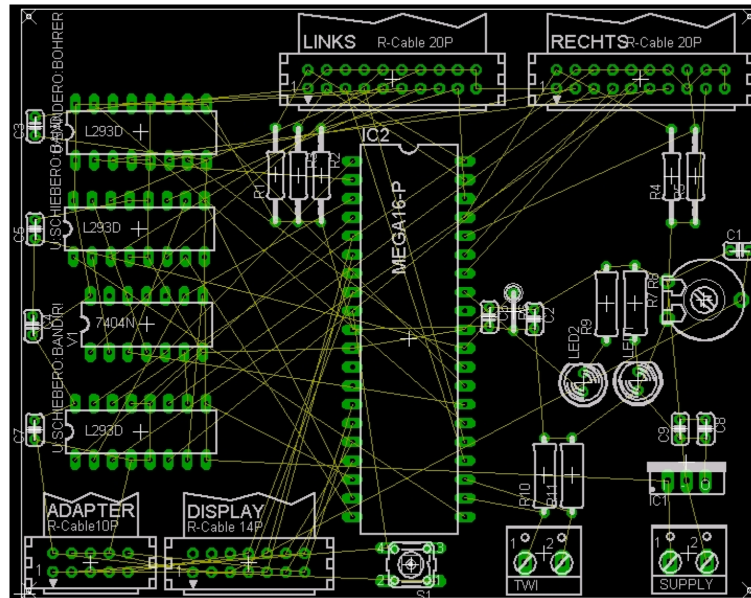


Abbildung 2.13: Ansicht Layout ohne geroutete Leiterbahnen

- **Leiterbahnen routen:**

  Sind die Bauteile alle auf der neuen Platine angeordnet, besteht der nächste Schritt darin, die Leiterbahnen zu verlegen. Das Optimum wäre dann erreicht, wenn alle Leiterbahnen sich auf einer Seite der Platine legen ließen. Dies ist jedoch nicht immer möglich. Auch bei unserer Platine kamen wir aufgrund der vielen Bauteile nicht um die Erstellung einer zweiseitigen Platine herum. Eagle bietet eine Möglichkeit mit der Funktion *Autoroute* die Leiterbahnen gemäß gewählter Einstellungen automatisch zu routen.

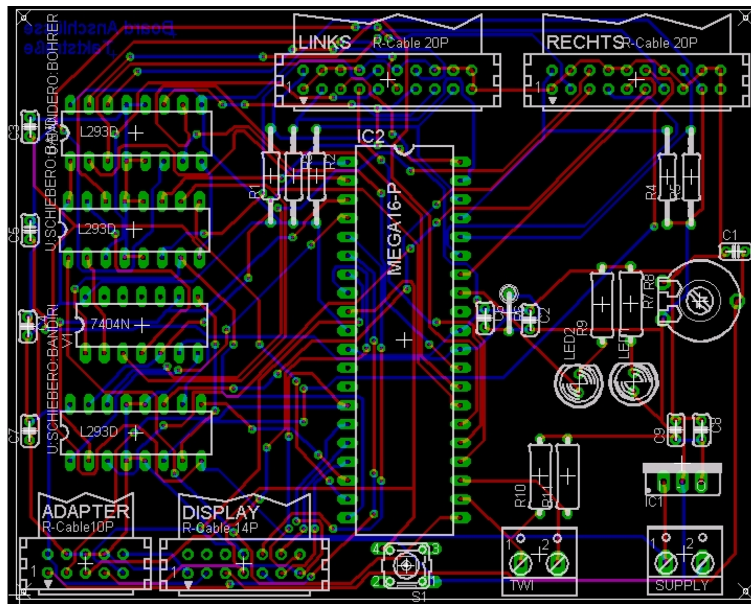


Abbildung 2.14: Ansicht Layout mit gerouteten Leiterbahnen

- **Freiflächen bedecken:**

  Schließlich sollten noch die Freiflächen mit Hilfe der dafür vorgesehene Funktion *Ratsnest*auf der Platine bedeckt werden, da alle nichtbedeckten Flächen entwickelt und in den kommenden Arbeitsschritten weggeätzt werden. Diese Maßnahme spart das unnötige Entfernen von Kupfer ein. Dadurch wird die Verwendbarkeit der Ätzlösung verlängert.

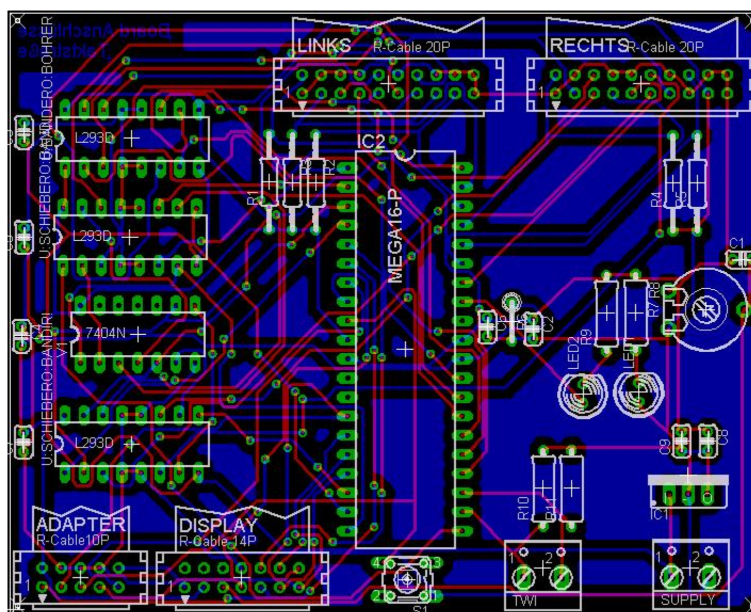


Abbildung 2.15: Fertiges Layout der Platine

2.3.3 Fertigung einer Platine

Die Fertigung einer Platine ist in einer universitätsintern erstellten Anleitung sehr anschaulich erklärt [Wil06]. Dennnoch wird im Folgenden kurz auf die wichtigsten Schritte im Herstellungsprozess eingegangen:

1. Zuallererst nimmt man sich eine Platine und entfernt auf den Seiten, auf denen man später ätzen will, die Schutzfolie.
2. Die zu fertigende Platine wird dann auf den Seiten, an denen sie später geätzt werden soll, gründlich mit Pausklar, einem Transparentspray, eingesprüht.
3. Der Papierausdruck des Platinenlayouts wird anschließend auf die besprühte Seite gelegt.

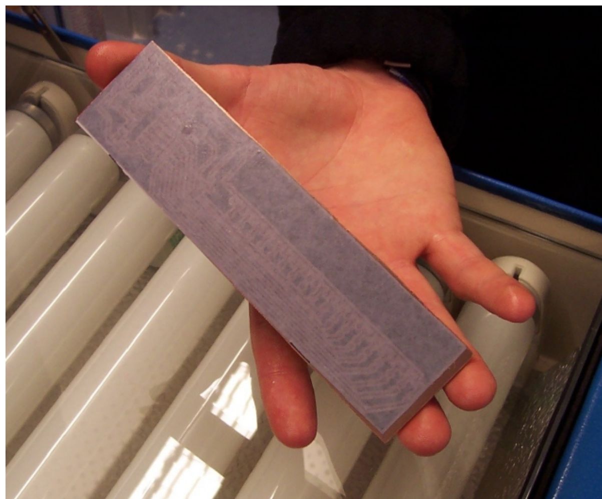


Abbildung 2.16: Eingesprühte Platine mit Layoutausdruck [Wil06]

4. In einer Belichtungsmaschine wird die Platine belichtet.
5. Nach bereits wenigen Minuten kann man die Platine wieder herausnehmen und sorgfältig alle Reste des Pausklars mit Seife abspülen.
6. Die gereinigte Platine wird nun in ein Entwicklerbad gelegt. In diesem befindet sich eine Natriumhydroxid-Lösung.

Abbildung 2.17: Entwicklung einer Platine im Natriumbad [Wil06]

7. Nachdem sich die Leiterbahnen sichtbar abgebildet haben wird die Platine aus dem Entwicklerbad genommen und klar abgespült.
8. Nun wird die belichtete Platine in ein Ätzbad eingetaucht.

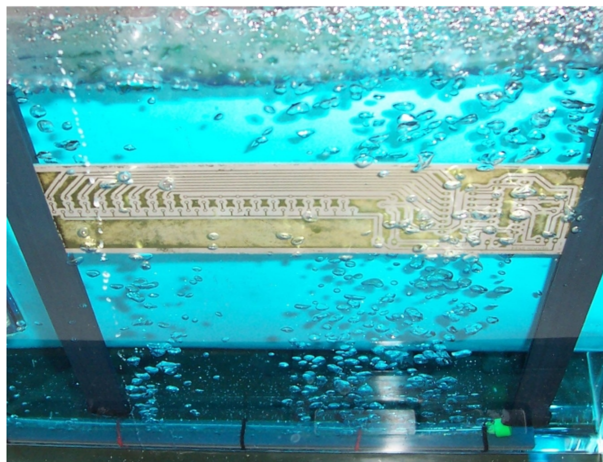


Abbildung 2.18: Eintauchen der Platine in ein Ätzbad [Wil06]

Dieses Ätzbad besteht aus einer Natriumpersulfat-Lösung.

An den belichteten Flächen wird dadurch das Kupfer weggeätzt. Die Platine ändert nun auch sichtbar ihre Farbe.

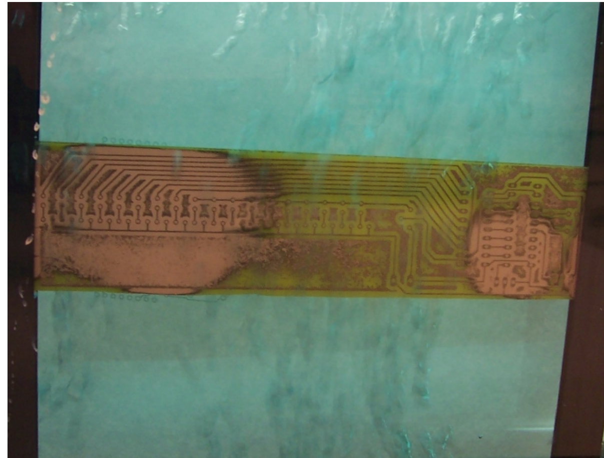


Abbildung 2.19: Platine im Ätzbad ändert ihre Farbe [Wil06]

9. Ist der Vorgang abgeschlossen, entnimmt man die fertige Platine aus dem Ätzbad und spühlt sie nochmal gründlich ab. Sind noch Unreinheiten vorhanden, so kann man diese mit Hilfe von Spiritus sehr leicht entfernen.
10. Als vorletzten Schritt werden Bohrungen an den Stellen vorgenommen, an denen später Leiterpins der verwendeten Bauteile eingesetzt werden.
11. Zum Schluss werden nochmal alle Leiterbahnen geprüft und gegebenenfalls nachgebessert. Anschließend wird die Platine mit Bauteilen bestückt.

2.4 Grundlegende Gedanken zur Implementation

Um unsere Software zu entwickeln griffen wir unter Anderem auf vorimplementierte Dienste des Mikrocontrollers zurück. Nach einigem Abwägen einigten wir uns auf folgende Funktionalitäten, die unser Projekt beinhalten sollte[9].

USART Um eine serielle Verbindung mit dem AT-Mega 16 aufzunehmen, und sei es nur zu Debug-Zwecken, benötigten wir die vom Mikrocontroller zur Verfügung gestellte USART[10]. Diese ermöglicht eine einfache Implementation der benötigten Schnittstelle zur Kontrolle und Information.

AD-Wandler Um die fünf Lichtschranken auslesen zu können, benötigt man eine Möglichkeit, analog gemessene Werte in digitaler Form weiterzuverarbeiten. Der AT-Mega 16 bietet diese Möglichkeit, so dass man keinen externen Analog-Digital-Wandler mehr benötigt. Auch dies spart in der Testphase etwas Arbeit, da man

[9] Eine ausführliche umfassende Dokumentation findet sich im Handbuch, bzw. Datenblatt zum ATMega16 [Atm06]. Eine komplette Einführung würde den Rahmen dieser Arbeit sprengen. Daher sind unsere Grundlagen viel mehr als eine Kurzeinführung zur schnellen Bedienung und zur richtigen Benutzung der jeweiligen Dienste zu sehen

[10] Universal Synchronous and Asynchronous serial Receiver and Transmitter

noch nicht zwingend eine eigene Platine benötigte, um diese Lösung zu implementieren.

Display Eine weitere Idee war der Anschluss eines vierzeiligen LCD-Displays, welches in diesem Projekt in erster Linie dazu dienen sollte, dem Benutzer eine Rückmeldung über den momentanen Status der Taktstraße und eventuelle Fehler und weitere interessante Informationen zu geben. Diese Möglichkeit wurde allerdings vom AT-Mega 16 nicht vorimplementiert, so dass die Entwicklung einer eigenen Bibliothek nötig wurde, die allerdings so allgemein konzipiert werden sollte, dass sie auch in weiteren Projekten verwendet werden kann.

TWI - I^2C Um zwischen zwei Mikrocontrollern zu kommunizieren bot sich das zur Verfügung gestellte TWI[11] an, welches in der Lage ist in einem System auf einem Bus zwischen Teilnehmern zu kommunizieren. Da die Kommunikation als einfache Zwei-Draht-Verbindung[12] realisierbar ist, ist sie dementsprechend platzsparend in ersten Tests. Sie besteht lediglich aus einer Daten- und einer Taktsignalleitung. Alles was benötigt wird ist ein TWI-Master sowie ein TWI-Slave, zum Beispiel zwei HWP-Boards aus der Lehre, um eine erste Verbindung aufzunehmen.

2.4.1 Grundlagen zum AT-Mega16

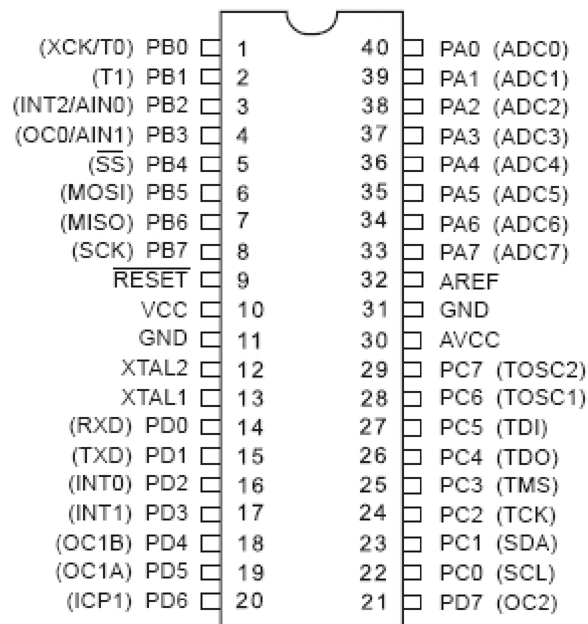


Abbildung 2.20: Pinbelegung AT-Mega 16

[11] Two-Wire Interface, vgl. I^2C
[12] eigentl. 3-Draht, rechnet man die Masse mit

Im Folgenden geben wir einen kurzen Überblick über den Mikrocontroller[13].

Der AT-Mega 16 ist ein 8-Bit Mikrocontroller mit einer erweiterten RISC-Architektur.

Er besitzt

- 131 Befehle,
- 32*8 Register,
- eine Taktfrequenz von bis zu 16 MHZ[14],
- nichtflüchtige Program-/ und Datenspeicher,
- ein JTag-Interface,
- zwei 8-Bit-Timer,
- einen 16-Bit-Timer,
- vier PWM-Kanäle,
- acht AD-Wandler-Kanäle,
- eine USART-Schnittstelle,
- eine TWI-Bus-Schnittstelle,
- ein SPI,
- einen Watchdog,
- einen Analog Comparator und
- spezielle Eigenschaften wie zB sechs verschiedene Sleep-Modi.

Diese Eigenschaften bieten eine Vielzahl von Einsatzmöglichkeiten des Mikrocontrollers auf kleinem Raum. Aufgrund dieser Tatsache und dem kleinen Anschaffungspreis bietet dieser Mikrocontroller einen guten Einstieg in dieses Themengebiet.

2.4.2 Grundlagen zum USART

Die USART-Schnittstelle erweitert die UART-Schnittstelle um weitere asynchrone Funktionen und um den Synchronbetrieb. Wir verwenden jedoch den asynchronen Betrieb.

Der Datenaustausch erfolgt über das Register **UDR**. Dieses kann beschrieben und ausgelesen werden.

[13] vgl. Spezifikation bei ATMEL
[14] ATMega16L: 0..8 MHZ

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDR (Read)
	TXB[7:0]								UDR (Write)
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 2.21: USART: Register **UDR** [Atm06]

Da zuerst einmal die USART initialisiert werden muss, sind folgende Schritte notwendig:

- Die Baudrate muss berechnet und dann im Baudratenregister **UBRR** gesetzt werden. Es ergibt sich folgende Berechnung für den zu schreibenden Wert:

$$UBRR = \frac{CLOCK}{(16 * BAUD)} - 1$$

  Das Low-Byte der 12 bit langen Baudrate wird in das Low-Register **UBRRL**, und das High-Byte in das High-Register **UBRRH** geschrieben.

Bit	15	14	13	12	11	10	9	8	
	URSEL	–	–	–	UBRR[11:8]				UBRRH
	UBRR[7:0]								UBRRL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R	R	R	R/W	R/W	R/W	R/W	
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Abbildung 2.22: USART: Register UBRR [Atm06]

- Der Receiver und Transmitter müssen aktiviert werden. Dies geschieht durch Setzen der Bits `TXEN` und `RXEN` im Register **UCSRB**.

Bit	7	6	5	4	3	2	1	0	
	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	UCSRB
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 2.23: USART: Register UCSRB [Atm06]

- Durch Löschen des Bits `UMSEL` im Register **UCSRC** wird der asynchrone Operationsmodus ausgewählt und durch Setzen der Bits `UCSZ0` und `UCSZ1` wird die Zeichengröße auf acht Bit gesetzt.

Bit	7	6	5	4	3	2	1	0	
	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	UCSRC
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	0	0	0	0	1	1	0	

Abbildung 2.24: USART: Register UCSRC [Atm06]

Um die USART zum Senden zu nutzen würde es nun eigentlich ausreichen, einen Wert in das Register **UDR** zu schreiben. Es wird zunächst noch gewartet, bis das das Bit `UDRE` im Register `UCSRA` gesetzt ist. Dies zeigt an, dass der Sendepuffer für die Übertragung neuer Daten bereit ist.

Bit	7	6	5	4	3	2	1	0	
	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	UCSRA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

Abbildung 2.25: USART: Register UCSRA [Atm06]

Genauso einfach wie das Senden ist das Empfangen. Man muss nur das Register **UDR** auslesen. Dies ist nach Anzeige eines angekommenen Zeichens mit Hilfe des Bits `RXC` im Register `UCSRA` möglich.

2.4.3 Grundlagen zum AD-Wandler

Der AT-Mega 16 besitzt bereits einen an einen Acht-Kanal-Multiplexer angeschlossenen Zehn-Bit-Analog-Digital-Wandler. Der Wandler unterstützt Dauer- und Einzelwandlungen. Dauerwandlungen werden im Folgenden aufgrund der gewählten Implementation nicht erläutert.

Die folgenden Register sind für eine Analog-Digital-Wandlung verantwortlich:

Bit	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 2.26: Multiplexer-Auswahl-Register ADMUX [Atm06]

Bit	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 2.27: Steuer- und Statusregister ADCSRA [Atm06]

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	–	–	ADC9	ADC8	ADCH
	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Bit	15	14	13	12	11	10	9	8	
	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	
Read/Write	R	R	R	R	R	R	R	R	
	R	R	R	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	
	0	0	0	0	0	0	0	0	

Abbildung 2.28: Daten-Register ADCL und ADCH [Atm06]

Mit der Zuweisung eines Kanals an die letzten drei Bits des Registers **ADMUX** wird der umzuwandelnde Kanal ausgewählt und durch Setzen der Bits `REFS0` und `REFS1` wird als Referenzspannung eine interne Spannung von 2.56V angelegt.

Im Steuer- und Status-Register **ADCSRA** wird der Wandler durch Setzen des Bits `ADEN` aktiviert und der Wandlungstaktteiler durch Setzen der Bits `ADPS0`, `ADPS1` und `ADPS2` festgelegt. Mit Hilfe dieser Einstellung kann man den Umwandlungstakt sinnvoll zwischen 50 und 200kHZ festlegen. Dies wird durch eine Auflösung in 10 Bits realisiert. [15].

Durch Einschreiben einer `1` in das Register **ADCSRA** an die Bitposition `ADSC` wird die Umwandlung gestartet. Bei einer Einzelwandlung wird dieses Bit nach einer Wandlung wieder auf den Wert `0` gesetzt, so dass nur gewartet werden muss, bis das Bit wieder `0` ist.

Nun steht das Ergebnis in den Datenregistern **ADC** zur Verfügung. Möchte man nur den Low- oder den High-Teil der Wandlung abholen, so stehen diese in den **ADC**-Registern **ADCL** und **ADCH** zur Verfügung. Man muss zuerst das Register **ADCL** und dann das Register **ADCH** auslesen um sicherzugehen, dass die Ergebnisse zur selben Wandlung gehören.

Der Wandler wieder durch Einschreiben einer `0` an die Bitposition `ADEN` des Registers `ADCSRA` wieder deaktiviert.

Das folgende Diagramm zeigt einen zeitlichen Ablauf einer Wandlung[16]:

[15] Benötigt man eine geringere Bitauflösung, so kann man den Takt auch höher als 200kHz wählen, um eine höhere Sample-Rate zu bekommen

[16] In diesem Fall die allererste Wandlung, der Wandler wird durch setzen des Bits `ADEN` eingeschaltet

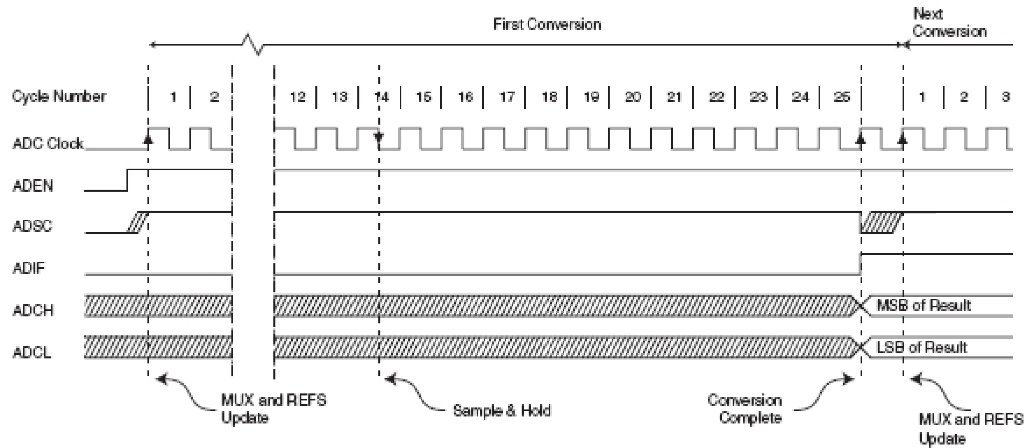


Abbildung 2.29: Ablauf einer Wandlung [Atm06]

2.4.4 Grundlagen zur Display-Ansteuerung

Das in diesem Projekt verwendete Display besitzt vier Zeilen mit jeweils 27 Zeichen, die von zwei Controllern angesteuert werden. Der erste Controller steuert die oberen beiden und der zweite die unteren beiden Zeilen.

Da in diesem Projekt aufgrund weniger offener Pins nur die Möglichkeit bestand ein Display im 4-Bit-Modus anzusprechen, wird hier der ebenfalls mögliche 8-Bit-Modus nicht benutzt[17] [Hee02].

Die folgende Grafik veranschaulicht kurz die Anschlussbelegung im Standard 8-Bit-Betrieb, in welchem die Datenleitungen an den Pins 8 bis 15 liegen[18]:

Pin	Funktion
1	GND
2	V_{DD} +5V ± 5%
3	V_O (ca. 0…4V Kontrasteinstellung)
4	RS
5	R/W
6	E1 (Controller obere Displayhälfte)
7	E2 (Controller untere Displayhälfte)
8	D0

Pin	Funktion
9	D1
10	D2
11	D3
12	D4
13	D5
14	D6
15	D7
16–21	Verbunden mit Kontaktflächen für acht Gummimatten-Taster

Abbildung 2.30: Belegung LCD [Hee02]

[17] Ausführliche Informationen finden sich jedoch im Datenblatt.

[18] Die Pinnummern sind deutlich an der Displayunterseite gekennzeichnet Liegt der Anschlussstreifen nach oben, so ist `PIN 1` links und `PIN 21` rechts.

Bei einer 4-Bit-Ansteuerung, wie wir sie verwenden, liegen die Datenleitungen an den Pins `D4` bis `D7`.

Die Kommunikation verläuft nun nur noch über die vier Datenbus-Leitungen, sowie über die Kontrolleitungen `RS`, `R/W`, `E1` und `E2`.

Diese Signale haben verschiedene Auswirkungen:

RS wählt bei einer eingeschriebenen `0` das Befehlsregister, und bei einer eingeschriebenen `1` das Datenregister aus.

R/W gibt bei einer `1` an, dass gelesen, und bei einer `0` ob geschrieben werden soll.

E1 und E2 sind im Ruhezustand `0`. Sie dienen dazu, bei einem Lesezugriff mit einer `1` anzuzeigen, ob zu lesende Daten anstehen, und bei einem Schreibzugriff werden anliegende Daten bei einer fallenden Flanke übernommen. `E1` spricht den oberen Displaycontroller an, während `E2` mit dem unteren kommuniziert.

Das Display verfügt bei schlechter Lesbarkeit noch über die Möglichkeit den Kontrast über den `PIN 3` zu regeln. Angeschlossen wird die Kontraststeuerung nach folgendem Plan:

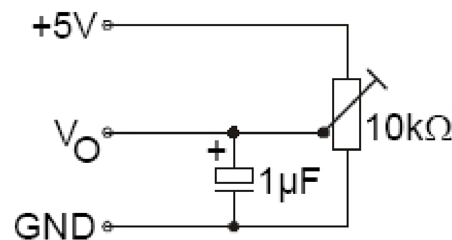


Abbildung 2.31: Kontrast LCD-Display

Um das Display zu verwenden empfiehlt es sich dieses zu initialisieren, da dies eine reibungslose Kommunikation garantiert. Die Initialisierung und die spätere Bedienung setzen einige Befehle voraus, die in der nachfolgenden Kurzübersicht aufgeführt sind.

Befehl	RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Funktion
Clear display	0	0	0	0	0	0	0	0	0	1	Anzeige löschen
Cursor home	0	0	0	0	0	0	0	0	1	*	Platziert den Cursor an DD-RAM-Adresse 0
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	I/D=1: vorwärts/inkrementieren/rechts I/D=0: rückwärts/dekrementieren/links S=1: Die Anzeige wird nach dem Schreiben eines Zeichens jeweils um eine Stelle entsprechend I/D verschoben S=0: Der Cursor wird nach dem Schreiben eines Zeichens jeweils um eine Stelle entsprechend I/D verschoben
Display on/off	0	0	0	0	0	0	1	D	C	B	D=1/0: Display ein/aus C=1/0: Unterstrich-Cursor ein/aus B=1/0: Blinkender Cursor ein/aus
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	*	*	Verschiebt die Anzeige (S/C=1) oder den Cursor (S/C=0) um eine Stelle nach rechts (R/L=1) oder nach links (R/L=0)
System set	0	0	0	0	1	DL	N	F	G	*	DL=0: 4-Bit Ansteuerung über D4…D7 DL=1: 8-Bit Ansteuerung N=1: 2 oder 4 Displayzeilen N=0: 1 Displayzeile F=1: 5×10-Zeichenbox oder 4-Zeilen-Display F=0: 5×7-Zeichenbox G=1: Spannungsinverter ein (Philips Contrl.)
Set CG-RAM address	0	0	0	1	A5	A4	A3	A2	A1	A0	Stellt die Schreibadresse (0…63) ins Zeichengenerator(CG)-RAM ein. Die nachfolgenden Zugriffe auf das Datenregister greifen auf das CG-RAM zu.
Set DD-RAM address	0	0	1	A6	A5	A4	A3	A2	A1	A0	Stellt die Schreibadresse (0…39, 64…103) ins Display(DD)-RAM ein. Die nachfolgenden Zugriffe auf das Datenregister greifen auf das DD-RAM zu.
Read busy flag/address counter	0	1	BF	A6	A5	A4	A3	A2	A1	A0	BF=1: das Display ist beschäftigt/kein Schreib-/Lesezugriff möglich BF=0: Display bereit/Zugriff möglich A6…A0: aktuelle Adresse im CG- oder DD-RAM
Write Data	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Schreibt Daten in CG- oder DD-RAM
Read Data	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Liest Daten aus dem CG- oder DD-RAM

Abbildung 2.32: Befehlsübersicht zur Ansteuerung des LCD-Displays [Hee02]

Die Initialisierung sollte folgendermaßen durchgeführt werden:

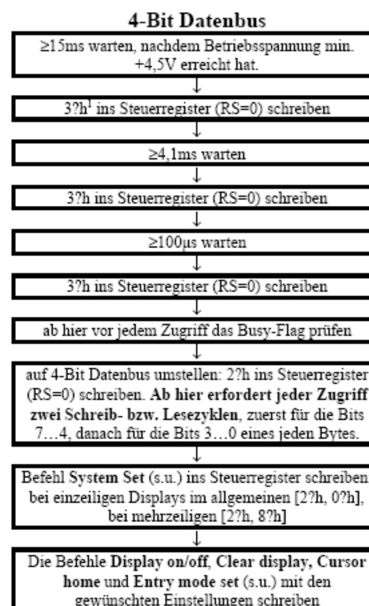


Abbildung 2.33: Initialisierungsequenz des LCD-Displays [Hee02]

Das Display ist danach einsatzbereit und sollte ein klares und unbeschriebenes Textbild zeigen.

Nun gibt es mehrere Möglichkeiten, das Display zu bedienen. Da wir im 4-Bit-Modus arbeiten, werden die Daten immer in zwei Blöcken à `4` Bit behandelt. Man benötigt zum Lesen und Schreiben nur Informationen darüber, welchen Teil des Displays man auslesen möchte, und dementsprechend über die Zuordnung der Enable-Leitung des zugehörigen Controller.

- Auslesen des Displayzustands[19] oder von Daten.

  Zuerst wird `R/W` gesetzt. Um nun den Zustand des Displays auszulesen muss `RS` `low` sein, sollen Daten gelesen werden, so muss `RS` auf `high` gesetzt sein. Nun wird für jedes Nibble die entprechende Enable-Leitung des gewählten Controllers auf `high` gesetzt, denn nur dann gibt das Display Daten zurück. Nun kann man die oberen vier Bit auslesen und dann den Pegel auf `low` setzen, um die unteren vier Bit mit dem gleichen Verfahren abzuholen.

- Senden eines Befehls oder von Daten

  Man sollte zuerst das Busy-Flag abfragen um sicherzugehen, dass das Display bereit ist, Daten zu empfangen. Dann muss `R/W` auf `low` gesetzt werden und `RS` auf `low`, möchte man ein Kommando senden, oder auf `high`, wenn Daten folgen sollen. Zuerst schreibt man nun die oberen vier Bit, welche mit einem kurzen *Toggle*[20] des Enable-Bits des jeweiligen Controllers vom Display übernommen werden. Mit dem gleichen Verfahren werden nun noch die unteren vier Bits gesendet.

2.4.5 Grundlagen zum Two-Wire Serial Interface

Two-**W**ire Serial **I**nterface (TWI) ist eine von Atmel eingeführte Bezeichnung für das auf vielen AVR Megas vorhandene I^2C-Bus-Interface[21]. Dies wird meist zur Kommunikation von Busteilnehmern innerhalb einer Schaltung verwendet.

Die Vorteile des TWI:

- Eingeteilt in Master- und Slavemodus
- Arbeiten als Transmitter oder Receiver möglich
- 2 Busleitungen, eine Datenleitung(SDA) und eine Taktleitung(SCL)
- 7-Bit Adressierung (ermöglicht bis zu 112 Geräte an einem Bus)
- erreichbare Übertragungsrate von maximal 400kBit/s

---

[19] Zum Beispiel zum Erfahren des momentanen Statusses des Displays anhand des Busy-Flags
[20] Wechsel auf `high` und zurück auf `low`.
[21] **I**nter-**I**ntegrated-**C**ircuit wurde von Philips entwickelt und als eingetragenes Markenzeichen geschützt.

Eine TWI-Verbindung wird vom Master aufgebaut und beendet. Dieser generiert den Takt und initiiert eine Kommunikation durch Legen eines *START*-Befehls auf den Bus und beendet sie durch einen *STOP*-Befehl. Zwischen diesen beiden Befehlen ist der Bus beschäftigt (busy) und ein oder mehrere Slaves können durch den Master adressiert werden. Eine besondere Situation entsteht, sobald nun einen weiteren *START*-Befehl gesendet wird. Diesen nennt man einen *REPEATED-START*-Befehl, welcher benutzt wird um einen neuen Transfer zu initiieren ohne die Kontrolle über den Bus zu verlieren.

Das TWI benötigt auf beiden Leitungen einen High-Pegel. Informationen auf der Datenleitung werden nur dann übernommen, wenn die Taktleitung auf `low`-Pegel gezogen wird. Die einzigen Ausnahmen sind die oben angesprochenen *START*-, *REPEATED-START*- und *STOP*-Befehle, bei denen auf der Datenleitungen Pegeländerungen akzeptiert werden, obwohl auf der Taktleitung ein High-Pegel anliegt.

Master und Slaves nehmen in einer Kommunikation jeweils eine der beiden möglichen Rollen, Transmitter beziehungsweise Receiver an. Der Transmitter legt die Daten auf den Bus, die der Receiver lesen soll. Somit befindet sich die Kommunikation immer in einem von vier Übertragungszuständen. Ein Master spricht einen Slave mit seiner Adresse und der Information, ob gesendet (*SLA+W*) oder gelesen werden soll (*SLA+R*), an. Dann werden die Daten übertragen. Jeder Befehl wird mit einem *ACK*[22] oder einem *NACK* quittiert. Des Weiteren kann ein *GENERAL CALL*, gefolgt von einem Write-Bit, vom Master benutzt werden um mehrere Slaves gleichzeitig anzusprechen und zu beschreiben. Ein Lesen mehrerer Slaves ist nicht möglich, da durch gleichzeitiges Senden von verschiedenen Daten ein Konkurrenzverhalten verursacht würde.

In jedem der vier TWI-Zustände sind mehrere Statusse möglich[23]. Diese werden im Folgenden vorgestellt:

- Master Transmitter Modus

Statuscode	**Beschreibung**
$08	*START*-Befehl übertragen
$10	*REPEATED-START*-Befehl übertragen
$18	*SLA+W* übertragen, *ACK* erhalten
$20	*SLA+W* übertragen, *NACK* erhalten
$28	Datum übertragen, *ACK* erhalten
$30	Datum übertragen, *NACK* erhalten
$38	Verbindung verloren

[22] Acknowledge
[23] vgl. TWSR

- Master Receiver Modus

Statuscode	Beschreibung
$08	*START-Befehl* übertragen
$10	*REPEATED-START-Befehl* übertragen
$38	Verbindung verloren
$40	*SLA+R* übertragen, *ACK* erhalten
$48	*SLA+R* übertragen, *NACK* erhalten
$50	Datum erhalten, *ACK* übertragen
$58	Datum erhalten, *NACK* übertragen

- Slave Transmitter Modus

Statuscode	Beschreibung
$A8	*SLA+R* erhalten, *ACK* übertragen
$B0	Verbindung verloren, *SLA+R* erhalten, *ACK* übertragen
$B8	Datum übertragen in **TWDR**, *ACK* erhalten
$C0	Datum übertragen in **TWDR**, *NACK* erhalten
$C8	Letztes Datum übertragen, *ACK* erhalten

- Slave Receiver Modus

Statuscode	Beschreibung
$60	*SLA+W* erhalten, *ACK* übertragen
$68	Verbindung verloren, *SLA+W* erhalten, *ACK* übertragen
$70	*GENERAL CALL* erhalten, *ACK* übertragen
$78	Verbindung verloren, *GENERAL CALL* erhalten, *ACK* übertragen
$80	Adressiert mit *SLA+W*, Datum erhalten, *ACK* übertragen
$88	Adressiert mit *SLA+W*, Datum erhalten, *NACK* übertragen
$90	Adressiert mit *GENERAL CALL*, Datum erhalten, *ACK* übertragen
$98	Adressiert mit *GENERAL CALL*, Datum erhalten, *NACK* übertragen
$A0	*STOP* oder *REPEATED-START* als Slave erhalten

Um den TWI Betrieb erfolgreich nutzen zu können, benötigt man folgende Register:

- **TWBR**

Bit	7	6	5	4	3	2	1	0	
	TWBR7	TWBR6	TWBR5	TWBR4	TWBR3	TWBR2	TWBR1	TWBR0	TWBR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 2.34: TWI Bit-Rate Register [Atm06]

Dies ist das Bit-Rate-Register. Es dient zum Einstellen des Bustaktes für den Master. Mit unten stehender Rechnung lässt sich die dafür benötigte Takt-Frequenz berechnen.

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot 4^{TWPS}}$$

Abbildung 2.35: Formel zur Errechnung der Taktfrequenz [Atm06]

- **TWAR**

Bit	7	6	5	4	3	2	1	0	
	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	TWAR
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	1	1	1	1	1	1	1	0	

Abbildung 2.36: TWI Adress-Register [Atm06]

Wird der AVR nicht im Master-Mode, sondern als Slave betrieben, so wird hier die Adresse festgelegt, mit der dieser erreichbar ist. Die ersten 7-Bit adressieren den Slave. Mit dem 8. Bit kann man einen *GENERAL CALL* aktivieren, mit Hilfe dessen ein Master mehrere Slaves erreichen kann.

- **TWSR**

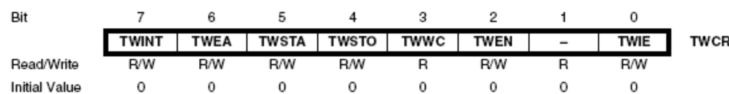
Bit	7	6	5	4	3	2	1	0	
	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	TWSR
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	1	1	1	1	1	0	0	0	

Abbildung 2.37: TWI Status-Register [Atm06]

Die beiden niederwertigsten Bits des Registers kontrollieren den Bit Rate Prescaler. Die ersten fünf Bits dieses Registers geben an, in welchem logischen Zustand sich der TWI Bus gerade befindet. Das heißt, sie zeigen an, ob der Bus als Master beziehungsweise Slave, oder als Receiver beziehungsweise Transmitter arbeitet. Zusätzlich kann hier abgelesen werden, welche Aktion auf dem Bus als letztes durchgeführt wurde und ob ein *ACK* oder *NACK* empfangen wurde.

In jedem Arbeitsschritt mit dem TWI Bus überprüft man zuerst das Statusregister. Anhand dieser Informationen kann man durch Lesen oder Schreiben aus dem Daten-Register **TWDR**, beziehungsweise durch Schreiben auf das Kontroll-Register **TWCR** passende Reaktionen auf den Bus geben.

- **TWCR**



Bit	7	6	5	4	3	2	1	0	
	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	TWCR
Read/Write	R/W	R/W	R/W	R/W	R	R/W	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Abbildung 2.38: TWI Control-Register [Atm06]

Das **TWCR** ist das Kontroll-Register. Hier werden alle Operationen des TWI durchgeführt. Das Control Register steuert die komplette TWI-Verbindung. Zusätzlich wird durch die Interrupts verhindert, dass Kollisionen auftreten indem während einer Übertragung ein weiterer Master auf den Bus zugreift.

Hier die 8 Bits im einzelnen:

- **Bit7 (TWINT)**
  Das TWI Interrupt-Flag. Nach Abarbeitung jeder Anweisung wird dieses Flag gesetzt und dadurch ein Interrupt ausgelöst, falls `TWIE` (Interrupt Enable Flag) gesetzt ist. Anschließend sollten die Aktionen im Status- beziehungsweise im Datenregister abgearbeitet werden. Um das Flag zu löschen muss es wieder gesetzt werden. Mit dem Zurücksetzen von `TWINT` wird die nächste Aktion ausgelöst. Es müssen deshalb vorher alle notwendigen Flags und Register (**TWDR**, **TWAR** oder **TWSR**) entsprechend gesetzt werden.
- **Bit6 (TWEA)**
  `TWEA` steht für TWI Enable-Acknowledge-Bit. Dieses kontrolliert das Senden eines *ACK*s nach einer Übertragung wenn:
  * als Slave die eigene Slaveadresse (im **TWAR**) erkannt wurde
  * ein *GENERAL CALL* erkannt wurde
  * ein Byte im Slave oder Master empfangen wurde
- **Bit5 (TWSTA)**
  `TWSTA` steht für das TWI Start-Bit. Wenn der Bus frei ist, kann ein *START*-Befehl gesendet werden. Ist der Bus nicht frei (busy), so wartet der Sender auf ein Stop-Bit und sendet dann den *START*-Befehl erneut. Wenn der *START*-Befehl erfolgt ist, muss das Flag wieder gelöscht werden.
- **Bit4 (TWSTO)**
  Mit Hilfe des Bits `TWSTO` wird das TWI Stop-Bit gesetzt. Als Slave kann dieses Flag zum Zurücksetzen nach einem Fehler verwendet werden. Es wird keine Stopsequenz ausgegeben, aber das Modul beeinflusst anschliessend nicht mehr die Leitungen SCL oder SDA. Das TWI-Modul befindet sich anschließend in einem definierten unadressierten Zustand.
- **Bit3 (TWWC)**
  Dies ist das TWI Write-Collision-Flag. Dieses Flag zeigt eine Kollision an, wenn man in das Register **TWDR** schreiben will, das Interupt Flag jedoch

nicht gesetzt ist. Ist das Interupt-Flag gesetzt, so kann auf das Datenregister geschrieben werden.

- **Bit2 (TWEN)**

  `TWEN` steht für TWI Enable. Dieses Flag benötigt man, um den TWI Modus zu aktivieren. Das Modul übernimmt dann die Steuerung von SDA (Datentransferleitung) und SCL (Taktleitung).

- **Bit1 (-)**

  Dieses Bit ist reserviert und wird stets auf `0` gesetzt.

- **Bit0 (TWIE)**

  `TWIE` steht für TWI Interrupt Enable. Die Abfrage durch Interrupts wird erst durch dieses Flag ermöglicht. Ist dieses Bit auf `1` gesetzt, ist ein Interrupt solange aktiv, bis das Interrupt Flag wieder auf `low` gezogen wird.

Nachfolgend die Maskierung der Bits der drei Befehle zum Aufnehmen oder Beenden einer Kommunikation:

Befehl	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**
START	1	X	1	0	X	1	0	X
REP START	1	X	1	0	X	1	0	X
STOP	1	X	0	1	X	1	0	X

- **TWDR**

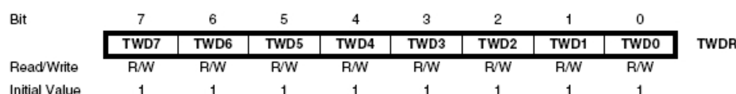


Abbildung 2.39: TWI Data-Register [Atm06]

TWDR bezeichnet das Data Register. Hier werden zur Übertragung der Daten alle acht Bit benötigt. Befindet sich das Device im Transmit-Mode (Sender), dann steht im Register **TWDR** das nächste zu übertragende Byte. Befindet sich das Device im Receive-Mode (Empfänger), so steht im hier das letzte Byte, dass empfangen wurde. Solange das `TWINT` Flag gesetzt ist, bleiben die Daten im **TWDR** gültig.

Ein Adresspaket besteht aus neun übertragenen Bits. Es beinhaltet sieben Adressbits, ein Read/Write-Bit und ein *ACK*-Bit. Ist das Read/Write-Bit gesetzt, wird ein Lesevorgang, andernfalls ein Schreibvorgang durchgeführt. Sobald ein Slave erkennt, dass er adressiert wurde, bestätigt dieser durch Ziehen der Datenleitung SDA auf `low` im neunten übertragenen Bit den Vorgang. Er sendet dadurch ein *ACK*. Passiert dies nicht und bleibt die Datenleitung unverändert, so kann der Master davon ausgehen, dass der Slave zur Zeit nicht erreichbar[24] ist.

[24] busy

Ein Datenpaket besteht ebenfalls aus neun Bits. Diese setzen sich zusammen aus einem Datenbyte und einem Acknowledge-Bit. Zur Bestätigung wird vom Empfänger die Datenleitung wird auf `low` gezogen (*ACK*). Reagiert der Empfänger nicht, interpretiert der Sender ein *NACK*. Der Empfänger sendet ein *NACK*, nachdem das letzte Byte gesendet wurde, beziehungsweise wenn der Empfänger keine weiteren Bytes mehr empfangen kann.

Eine kombinierte Verwendung aus Adress- und Datenpaket zeigt die folgende Grafik:

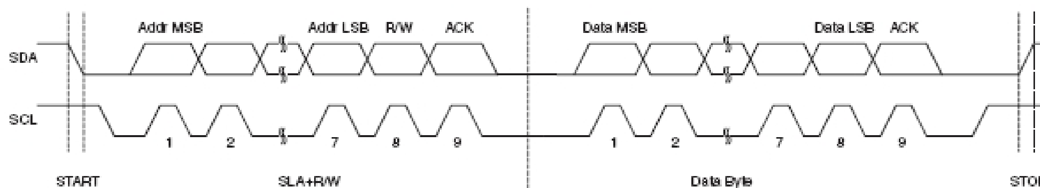


Abbildung 2.40: Ablauf einer Kommunikation [Atm06]

Einen beispielhaften Ablauf für einen Sendevorgang von Master zu Slave zeigt die der Atmel-Dokumentation entnommene Grafik:

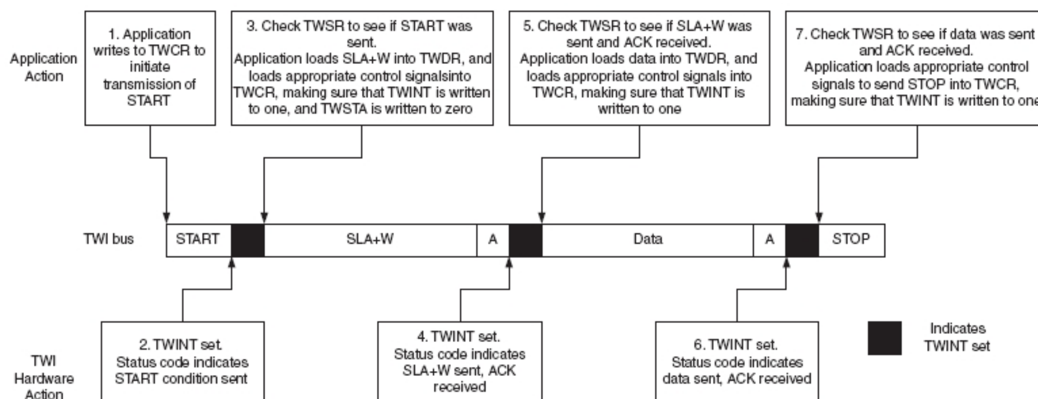


Abbildung 2.41: Ablauf eines Sendevorganges [Atm06]

Wir unterscheiden zwei Kommunikationsteile. Im ersten Teil wollen wir mit dem Master Daten an den Slave senden. Im zweiten Teil empfangen wir Daten vom Slave.

Die Kommunikation beginnt mit einem *START*-Befehl des Masters. Der Befehl *SLA+W* leitet die Sendephase des Masters ein. Der Master begibt sich in den Master-Transmitter Modus, der Slave in den Slave-Receiver Modus. Der Slave wartet nun auf Datensätze des Masters. Nach jedem empfangenen Datensatz sendet der Slave wiederum ein *ACK* an den Master, um ihm zu zeigen, dass er das Datum erfolgreich empfangen hat. Dieser Vorgang kann beliebig oft wiederholt werden. Sendet der Master nun eineb *Stop-Befehl*, ist die Verbindung zwischen beiden beendet und der Master verliert die Kontrolle über den

Bus. Mit einem *REPEATED-START*-Befehl anstelle dieser ist es möglich den gleichen oder einen beliebigen verfügbaren Slave zu adressieren.

Im zweiten Teil der Kommunikation empfangen wir Daten vom Slave. Mit einem *START*-Befehl bzw. einem *REPEATED-START*-Befehl und dem Befehl *SLA+R* begibt sich der Master in den Lesemodus. Dieser befindet sich danach im Master-Receiver Modus, der Slave im Slave-Transmitter Modus. Der Slave beginnt nun mit dem Senden des ersten Datums und wartet auf ein *ACK* des Masters. Hat dieser ein ACK gesendet, so sendet der Slave weiter seine Daten. Dies macht er solange, bis der Master ein *NACK* sendet und dem Slave somit signalisiert, dass er keine weiteren Daten mehr benötigt. Für den Slave ist die Verbindung dadurch beendet. Der Master kommt durch einen *Stop-Befehl* (verliert den Bus) oder *REPEATED-START*-Befehl (ermöglicht neue Adressierung) in den Urzustand zurück.

Diese Kommunikation wird in folgendem Flussdiagramm deutlich:

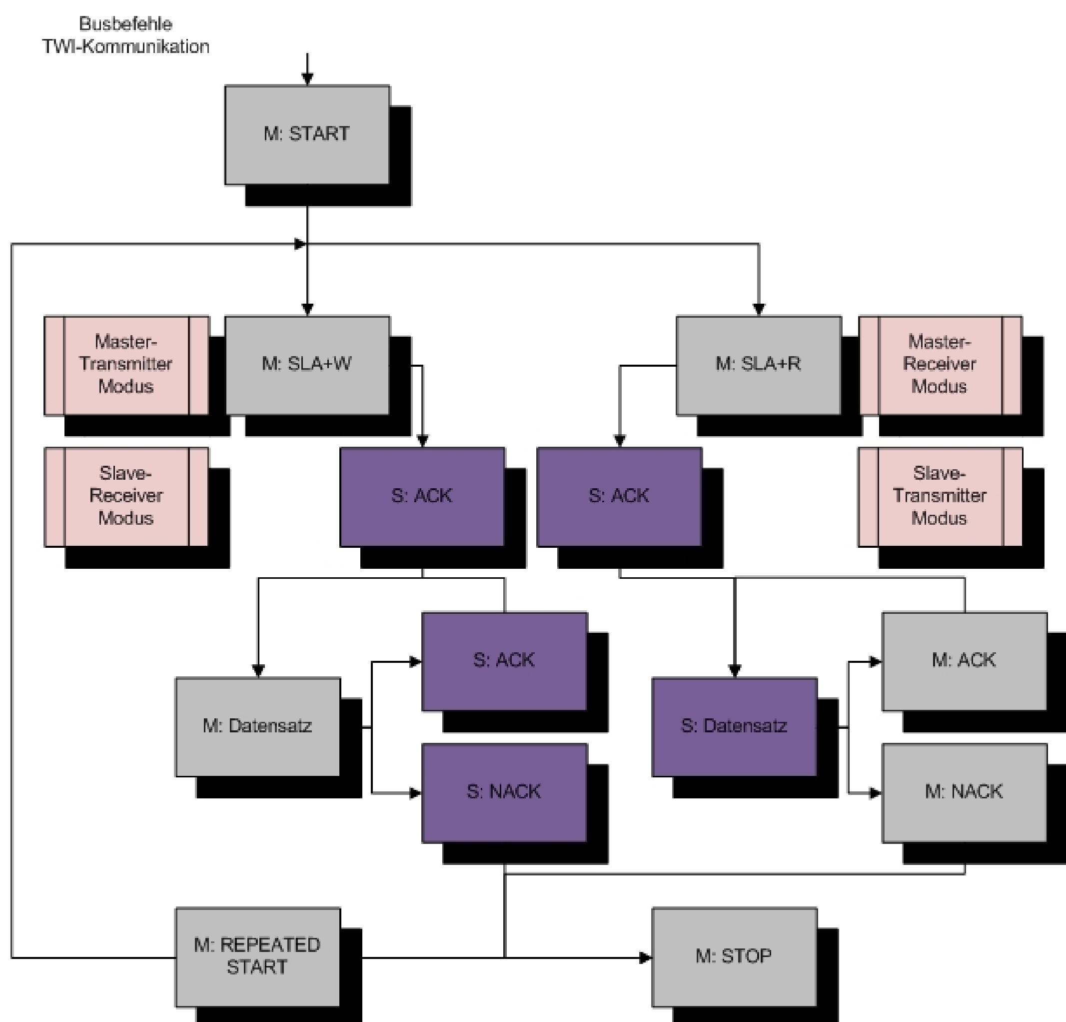


Abbildung 2.42: Flussdiagramm einer Kommunikation

3 Entwicklung

In diesem Kapitel soll Schritt für Schritt die Entwicklung und Entstehung des Projektes dokumentiert werden.

3.1 Analyse der Taktstraße

Der erste Schritt war also eine Sammlung der Aufgaben, die unsere spätere Lösung einmal bewältigen sollte. Um einen Überblick zu erhalten und erste Tests zu absolvieren, analysierten wir die Taktstraße und kamen zu folgendem Ergebnis.

Wir benötigten:

- eine serielle Schnittstelle zur Kontrolle und Benutzerkommunikation,
- eine Programmierschnittstelle,
- ein externes LCD-Display um direkt mit dem Benutzer zu kommunizieren sowie
- eine Möglichkeit den TWI-Bus zu benutzen,
- den internen Analog-Digital-Wandler um die Lichtschranken korrekt handhaben zu können,
- Vorwiderstände um die Fototransistoren[1] nutzen zu können,
- interne Pullups um Taster abzufragen,
- eine geeignete Ansteuerung für:
  - fünf Fototransistoren,
  - vier Förderbänder (die Bearbeitungsbänder sollen in beide Richtungen steuerbar sein),
  - vier Taster an den Schiebern,
  - zwei Schieber(in beide Richtungen steuerbar), Fräser und Bohrer.

An den Tastern und Fototransistoren sollen durch Pullup-Widerstände beziehungsweise Vorwiderstände High-Pegel anliegen. Diese schließen somit auf `0`.

Aufgrund dieser Anforderungen bot sich die Nutzung des AT-Mega16 als Mikroprozessor, der mit allen Aufgaben bedacht beinahe alle Pins belegt haben sollte, an.

[1] vgl. Pullup-Widerstände bei digitalen Signalen

3.2 Belegung des AT-Mega16

Die nachfolgende Pinbelegung des Mikrocontrollers steuert die Taktstraße an:

Port A	**Beschreibung**
PA0(ADC0)	Fototransistor an der Einlegestation
PA1(ADC1)	Fototransistor am linken Schieber
PA2(ADC2)	Fototransistor an der Fräsmaschine
PA3(ADC3)	Fototransistor am Bohrer
PA4(ADC4)	Fototransistor hinter dem Auslagerband
PA5(ADC5)	Taster des Schiebers rechte Seite (hinten)
PA6(ADC6)	Taster des Schiebers rechte Seite (vorne)
PA7(ADC7)	Rote LED (einstellbare Kontrolllampe)

Port B	**Beschreibung**
PB0(XCK/T0)	Band Bohrmaschine (Richtung)
PB1(T1)	Auslagerband (an/aus)
PB2(INT2/AIN0)	Motor Schieber rechts (Richtung)
PB3(OC0/AIN1)	Motor Schieber rechts (an/aus)
PB4(SS)	LCD(Kontrolleitung RS)
PB5(MOSI)	LCD(Kontrolleitung R/W), Programmieradapter(Pin 1)
PB6(MISO)	LCD(Kontrolleitung E1), Programmieradapter(Pin 9)
PB7(SCK)	LCD(Kontrolleitung E2), Programmieradapter(Pin 7)

Port C	**Beschreibung**
PC0(SCL)	TWI-Taktleitung
PC1(SDA)	TWI-Datenleitung
PC2(TCK)	Taster des Schiebers linke Seite (vorne)
PC3(TMS)	Taster des Schiebers linke Seite (hinten)
PC4(TDO)	Zuführband (an/aus)
PC5(TDI)	Band Fräse (Richtung)
PC6(TOSC1)	Motor Fräse (an/aus)
PC7(TOSC2)	Motor Bohrer (an/aus)

Port D	**Beschreibung**
PD0(RXD)	LCD(Datenleitung D4), Programmieradapter(Pin 3)
PD1(TXD)	LCD(Datenleitung D5), Programmieradapter(Pin 10)
PD2(INT0)	LCD(Datenleitung D6)
PD3(INT1)	LCD(Datenleitung D7)
PD4(OC1B)	Band Bohrmaschine (an/aus)
PD5(OC1A)	Band Fräse (an/aus)
PD6(ICP1)	Motor des Schiebers linke Seite (Richtung)
PD7(OC2)	Motor des Schiebers linke Seite (an/aus)

Die Änderung der Richtung wird nach folgendem Schema gehandhabt:

Wird ein Pin zur Änderung der Richtung gesetzt, bewegt sich ein Schieber oder ein Förderband vorwärts. Wird dieser Pin auf Null gezogen, bewegt sich ein Schieber oder ein Förderband rückwärts.

Einige Erläuterungen zu den gewählten Belegungen:

- 4 Anschlüsse zu den insgesamt vier Tastern.
- 5 Anschlüsse für die fünf Fototransistoren.
- 5 Anschlüsse zum Programmieradapter. Diese haben wir identisch gewählt mit den Anschlüssen zum Display, da die Ansteuerung des Programmieradapters nur in der Entwicklungsphase auftritt und in der finalen Nutzung entfällt.
- 6 Anschlüsse zu den Förderbändern. Jeweils ein Anschluss zu den beiden Außenbändern, welche in eine Richtung steuerbar sind und jeweils zwei Anschlüsse zu den beiden Innenbändern, welche in zwei Richtungen steuerbar sind.
- 6 Anschlüsse zu Motoren. Jeweils ein Anschluss zu den beiden Verarbeitungsgeräten (Bohrer, Fräse) und jeweils zwei Anschlüsse zu den Schiebern für die Bewegungen vorwärts und rückwärts.
- 8 Anschlüsse zum Display

3.3 Entwicklung der Hardware

Die Entwicklung der Hardware besteht aus dem Entwerfen und Erstellen der benötigten Platinen, sowie der Verkabelung des Displays.

3.3.1 Schaltpläne

Um die Taktstraße ansteuern zu können, benötigen wir neben einer Softwarelösung auch Platinen. Um den Aufwand in einem gewissen Rahmen zu halten, entschieden wir uns insgesammt 3 Platinen logisch unterteilt zu entwerfen[2].

Auf der Prozessorplatine befindet sich der AT Mega16 inklusive eines R/C-Gliedes, um die Spannungsversorgung des Analog-Digital-Wandlers sicher zu stellen. Diese Platine ist über Wannenbuchsen und Flachbandkabel mit den beiden anderen, der Anschlussplatine und der Kontrollplatine, verbunden.[3]

[2] Zum damaligen Zeitpunkt war das zweiseitige Routen zu umständlich. Daher versuchten wir möglichst alles einseitig zu routen. Wie im Fazit auch zu lesen ist, kann man zweiseitige Platinen ganz unkompliziert professionell routen lassen, was wir im zweiten Anlauf auch getan haben. Leider konnte die neue Platine bisher nicht getestet werden.

[3] Vergrößerte Darstellung im Anhang A1-Platinen

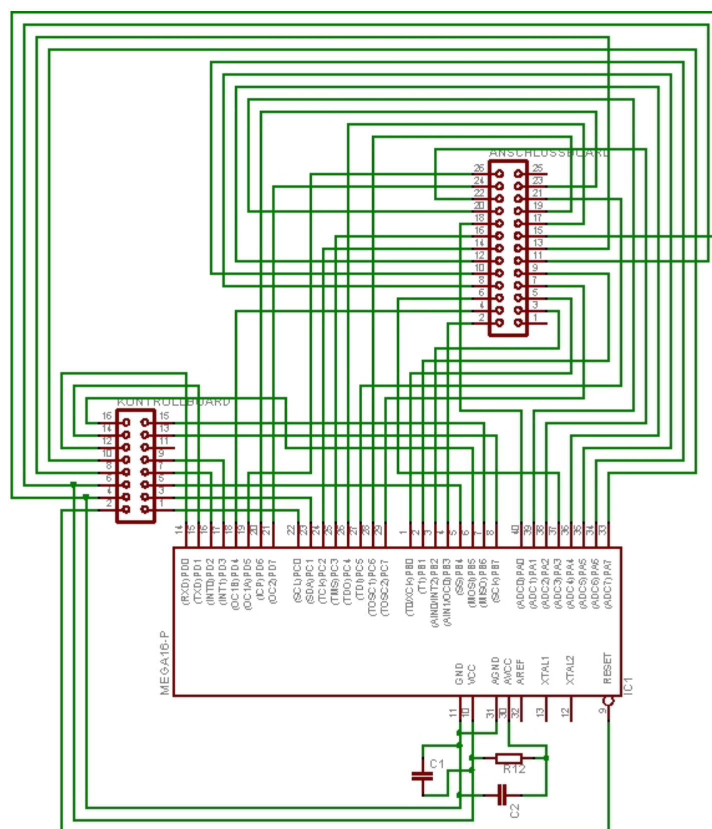


Abbildung 3.1: Prozessorplatine

Bauteile der Prozessorplatine:

- ein ATMega16
- eine Wannenbuchse 16-polig
- eine Wannenbuchse 26-polig
- zwei Keramikkondensatoren à 100 nF
- ein Widerstand à 100 Ω

Der Widerstand und einer der beiden Kondensatoren bilden das R/C-Glied zum Glätten der Versorgungsspannung auf der Platine, der andere Kondensator sichert die Spannung des ATs.

Anschlussbelegung der Prozessorplatine zur Anschlussplatine:

Pin	Beschreibung	AT-Pin
1	-	-
2	Motor des Schiebers rechte Seite (an/aus)	PB3(OC0/AIN1)
3	Motor des Schiebers rechte Seite (Richtung)	PB2(INT2/AIN0)
4	Band Bohrmaschine (an/aus)	PD4(OC1B)

5	Band Bohrmaschine (Richtung)	PB0(XCK/T0)
6	Fototransistor am Bohrer	PA3(ADC3)
7	Motor Bohrer (an/aus)	PC7(TOSC2)
8	Taster des Schiebers rechte Seite (vorne)	PA6(ADC6)
9	Auslagerband (an/aus)	PB1(T1)
10	Taster des Schiebers rechte Seite (hinten)	PA5(ADC5)
11	*VCC(5V)*	-
12	Fototransistor hinter dem Auslagerband	PA4(ADC4)
13	*VCC(9V)*	-
14	Taster des Schiebers linke Seite (vorne)	PC2(TCK)
15	*GND*	-
16	Taster des Schiebers linke Seite (hinten)	PC3(TMS)
17	Zuführband (an/aus)	PC4(TDO)
18	Fototransistor an der Einlegestation	PA0(ADC0)
19	Motor Fräse (an/aus)	PC6(TOSC1)
20	Fototransistor am linken Schieber	PA1(ADC1)
21	Band Fräse (Richtung)	PC5(TDI)
22	Fototransistor an der Fräsmaschine	PA2(ADC2)
23	Motor des Schiebers linke Seite (Richtung)	PD6(ICP1)
24	Motor des Schiebers linke Seite (an/aus)	PD7(OC2)
25	-	-
26	Band Fräse (an/aus)	PD5(OC1A)

Anschlussbelegung der Prozessorplatine zur Kontrollplatine:

Pin	**Beschreibung**	**AT-Pin**
1	TWI-Taktleitung	PC0(SCL)
2	Programmieradapter Pin 5	RESET
3	TWI-Datenleitung	PC1(SDA)
4	*GND*	-
5	Display(RS)	PB4(SS)
6	*VCC (5V)*	-
7	Display(D6)	PD2(INT0)
8	*VCC (9V)*	-
9	Display(D7)	PD3(INT1)
10	LED	PA7(ADC7)
11	-	-
12	Display(D4), Programmieradapter(Pin 3)	PD0(RXD)
13	Display(E2), Programmieradapter(Pin 7)	PB7(SCK)
14	Display(D5), Programmieradapter(Pin 10)	PD1(TXD)
15	Display(E1), Programmieradapter(Pin 9)	PB6(MISO)
16	Display(R/W), Programmieradapter(Pin 1)	PB5(MOSI)

Auf der Kontrollplatine sind die Anschlüsse vom Programmieradapter, Display, Potentiometer, Resetknopf und die Stromversorgung untergebracht.[4]

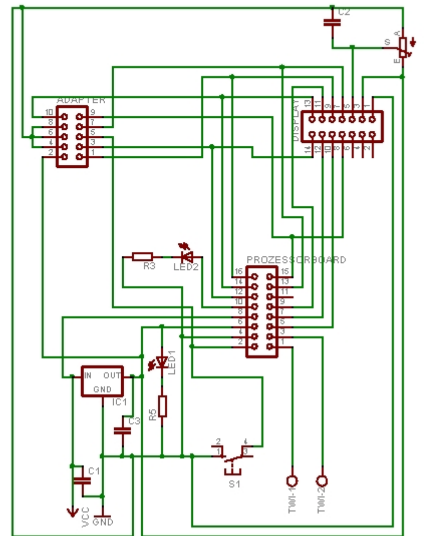


Abbildung 3.2: Kontrollplatine

Bauteile der Kontrollplatine:

- ein Potentiometer[5] à 100 KΩ (Display)
- ein Taster
- jeweils ein GND- und VCC-Plug
- zwei TWI Klemmen
- ein Spannungswandler von 9V auf 5V
- zwei LEDs inklusive Vorwiderstand à 154 Ω
- eine Wannenbuchse 10-polig
- eine Wannenbuchse 14-polig
- eine Wannenbuchse 16-polig
- ein Elektrolytkondensator[6] à 1 μF (Display)
- zwei Elektrolytkondensatoren à 10 μF

Die beiden Kondensatoren mit 10 μF sichern die Stromversorgung, der Kleinere die Kontrastregelung des Displays.

[4] Vergrößerte Darstellung im Anhang A1-Platinen
[5] hier auch: Trimmer
[6] Elektrolytkondensatoren(kurz: Elko) sind langsamer als die schnelleren Keramikkondensatoren(kurz: Kerko), besitzen jedoch eine größere Kapazität.

Anschlussbelegung der Kontrollplatine zum Programmieradapter:

Pin	**AT-Pin**
1	PB5(MOSI)
2	*VCC*
3	PD0(RXD)
4	*GND*
5	RESET
6	*GND*
7	PB7(SCK)
8	*GND*
9	PB6(MISO)
10	PD1(TXD)

Anschlussbelegung der Kontrollplatine zum Display[7]:

Pin	**Beschreibung**	**AT-Pin**
1	*GND*	-
2	-	-
3	*VCC*	-
4	-	-
5	*Kontrast*	-
6	-	-
7	Kontrolleitung E2	PB7(SCK)
8	Kontrolleitung E1	PB6(MISO)
9	Kontrolleitung R/W	PB5(MOSI)
10	Kontrolleitung RS	PB4(SS)
11	Datenleitung D7	PD3(INT1)
12	Datenleitung D6	PD2(INT0)
13	Datenleitung D5	PD1(TXD)
14	Datenleitung D4	PD0(RXD)

Auf der dritten Platine, der Anschlussplatine sind zur logischen Verknüpfung mehrere L293D und ein 7404 verbaut. Der Baustein 7404 negiert Eingangssignale. Er wird dazu verwendet mit Hilfe einer H-Brückenschaltung Polaritäten zu ändern. In diesem Projekt wird dadurch ermöglicht, zwei Transportbandmotoren sowie die beiden Schiebemotoren in beide Richtungen laufen zu lassen.

Zusätzlich befinden sich auf dieser Platine die Anschlüsse zur linken und rechten Seite der Taktstraße.[8]

[7]Anschlussinformation des Displays ist bei den Grundlagen in der Abbildung 2.32 zu finden.
[8]Vergrößerte Darstellung im Anhang A1-Platinen

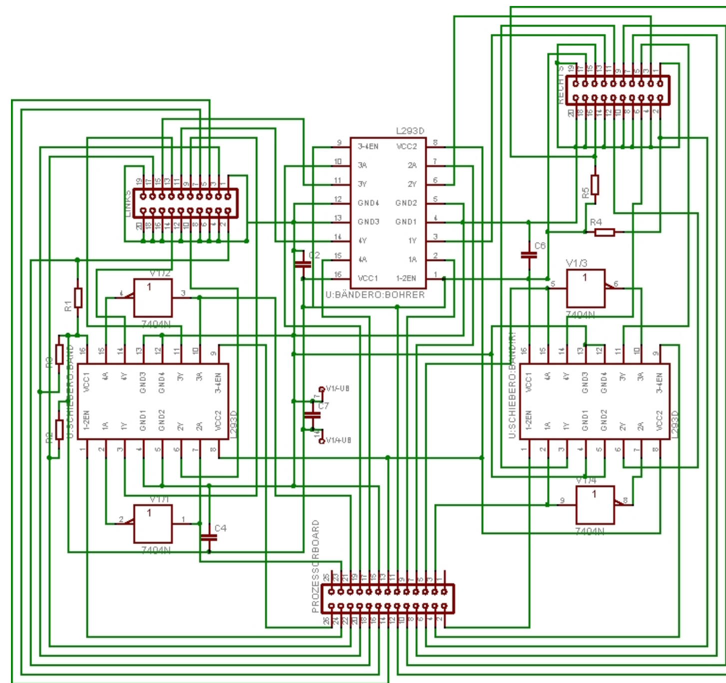


Abbildung 3.3: Anschlussplatine

Bauteile der Anschlussplatine:

- ein 7404 Baustein
- drei L293D Bausteine
- zwei Wannenbuchsen 20-polig
- eine Wannenbuchse 26-polig
- vier Keramikkondensatoren à 100 nF
- fünf Vorwiderstände vor den Fototransistoren à 2,2 KΩ

Jeder der vier Kondensatoren sichert die Spannungsversorgung der vier ICs[9].

Anschlussbelegung der Anschlussplatine zur Taktstraße (linker Connector):

Pin	Beschreibung	AT-Pin
1	Fototransistor an der Einlegestation *GND*	-
2	Fototransistor an der Einlegestation	PA0(ADC0)
3	Fototransistor am linken Schieber	PA1(ADC1)
4	Fototransistor am linken Schieber *GND*	-
5	Taster des Schiebers linke Seite (vorne)	PC2(TCK)
6	Taster des Schiebers linke Seite (vorne) *GND*	-
7	Taster des Schiebers linke Seite (hinten)	PC3(TMS)

[9](engl. **I**ntegrated **C**ircuit) Integrierter Schaltkreis

8	Taster des Schiebers linke Seite (hinten) *GND*	-
9	Motor des Schiebers linke Seite	-
10	Motor des Schiebers linke Seite	-
11	Zuführband (an/aus)	PC4(TD0)
12	Zuführband (an/aus) *GND*	-
13	Band Fräse	-
14	Band Fräse	-
15	Motor Fräse (an/aus)	PC6(TOSC1)
16	Motor Fräse (an/aus) *GND*	-
17	Fototransistor an der Fräsmaschine	PA2(ADC2)
18	Fototransistor an der Fräsmaschine *GND*	-
19	*GND*	-
20	*GND*	-

Anschlussbelegung der Anschlussplatine zur Taktstraße (rechter Connector):

Pin	**Beschreibung**	**AT-Pin**
1	Fototransistor am Bohrer *GND*	-
2	Fototransistor am Bohrer	PA3(ADC3)
3	Motor Bohrer (an/aus)	PC7(TOSC2)
4	Motor Bohrer (an/aus) *GND*	-
5	Band Bohrmaschine	-
6	Band Bohrmaschine	-
7	Taster des Schiebers rechte Seite (vorne)	PA6(ADC6)
8	Taster des Schiebers rechte Seite (vorne) *GND*	-
9	Taster des Schiebers rechte Seite (hinten)	PA5(ADC5)
10	Taster des Schiebers rechte Seite (hinten) *GND*	-
11	Motor Schieber rechts	-
12	Motor Schieber rechts	-
13	Auslagerband (an/aus)	PB1(T1)
14	Auslagerband (an/aus) *GND*	-
15	Fototransistor hinter dem Auslagerband *GND*	-
16	Fototransistor hinter dem Auslagerband	PA4(ADC4)
17	Lampen *VCC(5V)*	-
18	Lampen *GND*	-
19	*GND*	-
20	*GND*	-

3.3.2 Platinen entwerfen und herstellen

In unserem ersten Ansatz entschieden wir uns für eine Aufteilung der Platine in drei kleinere Platinen. Grund dazu war einerseits die Limitierung der Platinengröße durch

die Nutzung von EAGLE als Freewareprogramm, andererseits versuchten wir diese Platinen einseitig zu routen. Im einer späteren Entwicklungsphase haben wir einen Entwurf zweiseitig geroutet, welcher alle Komponenten auf nur einer Platine beinhaltete. Einige Bilder zu dieser Platine sind in der Einführung zu finden (Abbildungen 2.8, 2.9 und 2.10 zeigen die Erstellung des Layouts für diese Platine). Leider konnten wir mit dieser neuen Platine unsere drei logisch aufgeteilten Platinen noch nicht ersetzen, da uns diese noch nicht zur Verfügung steht.

Im Folgenden wird die Drei-Platinen Lösung vorgestellt.

Auf unserer ersten Platine, der Prozessorplatine, befindet sich ein ATMega16, zwei Wannenbuchsen, zwei Kondensatoren und ein Widerstand. Von dieser Platine aus laufen Verbindungen zur Anschlussplatine der Taktstraße sowie Verbindungen zur Platine der Kontrollanschlüsse.

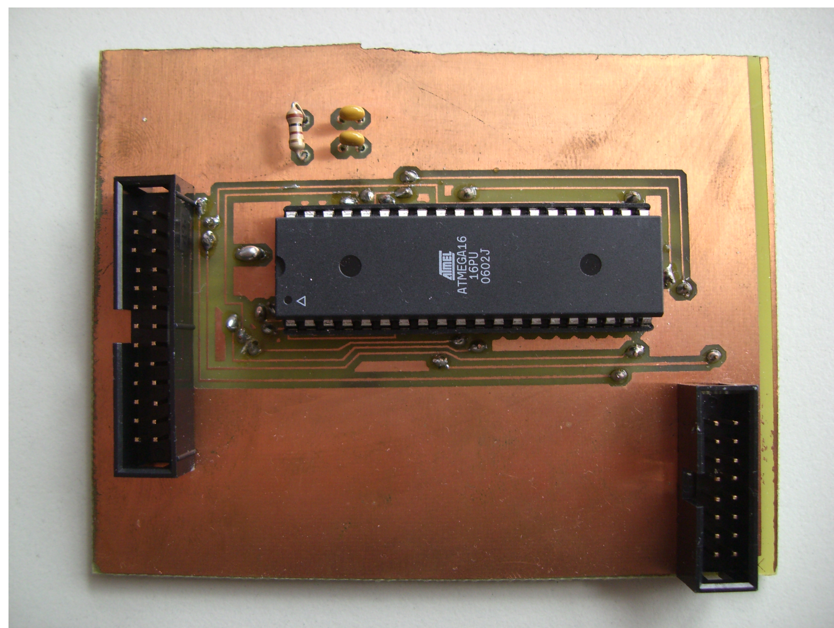


Abbildung 3.4: Prozessorplatine

Auf der Anschlussplatine zur Taktstraße sind drei L293D Bausteine, ein 7404 Baustein, fünf Widerstände, vier Kondensatoren und die Verbindungsbuchsen zur linken, beziehungsweise zur rechten Seite der Taktstraße sowie zum Prozessorplatine angebracht.

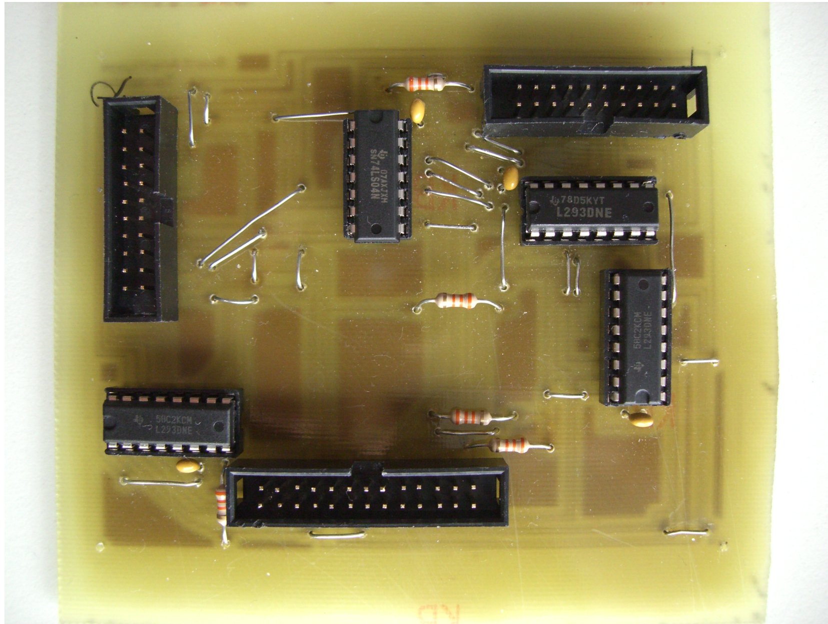


Abbildung 3.5: Platine zwischen Taktstraße und Prozessor

Auf der Kontrollplatine befinden sich die Stromversorgung, die Verbindungsbuchse zum Programmieradapter, die beiden TWI Anschlüsse, eine Verbindungsbuchse zum Display mit Kontrasteinstellung, ein Resetknopf, eine Verbindungsbuchse zur Prozessorplatine, drei Kondensatoren und zwei Widerstände sowie ein Kontrolllämpchen für die Betriebsanzeige(grün) und ein programmierbares Alarmlämpchen (rot).

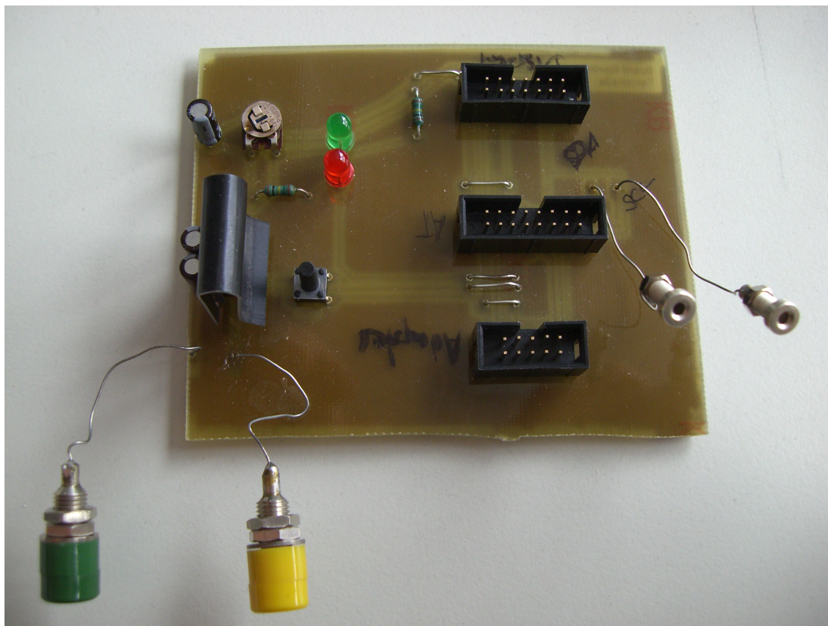


Abbildung 3.6: Kontrollplatine mit allen Anschlüssen

Miteinander verbunden ergeben die drei Platinen folgendes Bild:

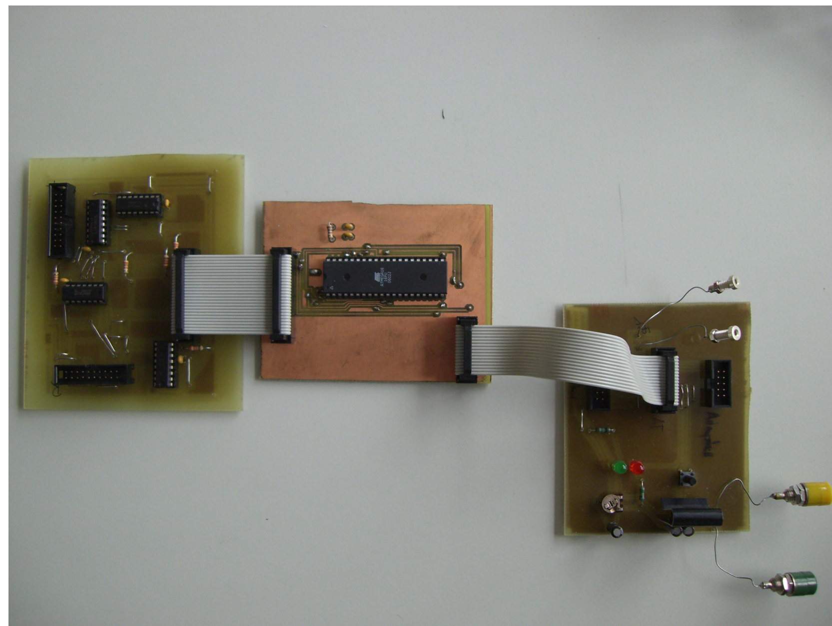


Abbildung 3.7: Verbindung aller drei Platinen

3.3.3 Verkabeln des Displays

Da das Display wahlweise in einem 4-Bit- oder einem 8-Bit-Modus betrieben werden kann, wird das Display mit einem angelöteten Folienleiter ausgeliefert.Dieser wurde nicht benötigt und daher abgelötet, um das verwendete Flachbandkabel nach folgendem Anschlussplan direkt auf die Kontaktpads zu löten:

Pin	Beschreibung	AT-Pin
1	*GND*	-
2	*VCC*	-
3	Kontrast	-
4	Kontrolleitung RS	PB4(SS)
5	Kontrolleitung R/W	PB5(MOSI)
6	Kontrolleitung E1	PB6(MISO)
7	Kontrolleitung E2	PB7(SCK)
8	frei	-
9	frei	-
10	frei	-
11	frei	-
12	Datenleitung D4	PD0(RXD)
13	Datenleitung D5	PD1(TXD)
14	Datenleitung D6	PD2(INT0)

15	Datenleitung D7	PD3(INT1)
16	frei	-
17	frei	-
18	frei	-
19	frei	-
20	frei	-
21	frei	-

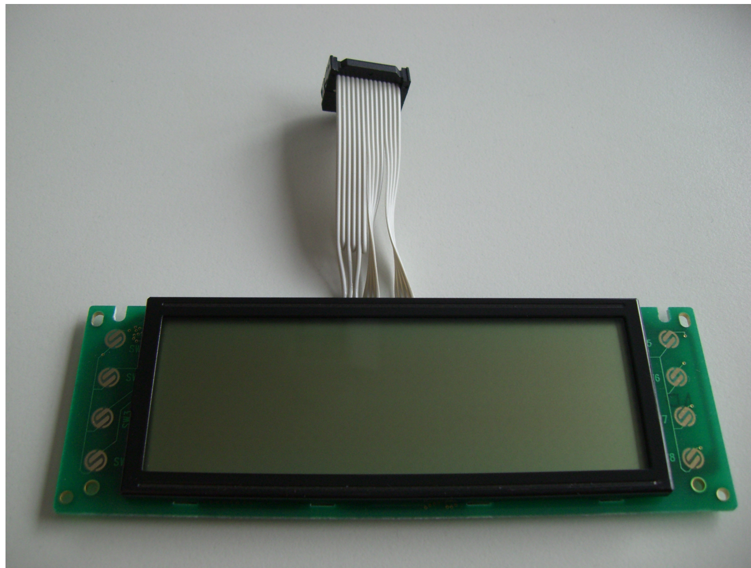


Abbildung 3.8: Frontansicht LCD-Display

Abbildung 3.9: Rückansicht LCD-Display

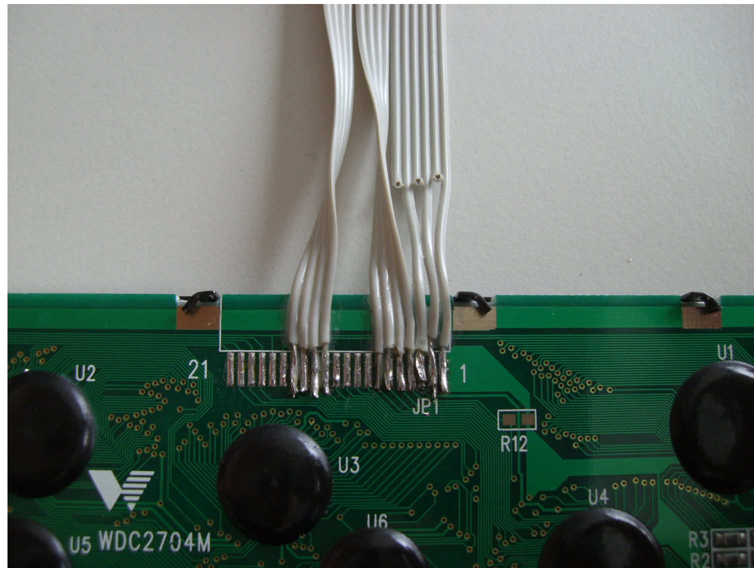


Abbildung 3.10: Rückansicht nah LCD-Display

Der Trimmer[10] zur Kontrastregelung befindet sich auf der Kontrollplatine.

3.4 Entwicklung der Software

In diesem Kapitel werden die Implementierungen der Teillösungen, die für unser Projekt relevant sind, besprochen.

Bei den Teillösungen[11] handelt es sich um Möglichkeiten zur Kommunikation zwischen Anwender und Hardware, zum Verwerten von analog eingelesenen Werten, sowie zur Teilnahme an einem TWI-Bus-System, welches für unser Projekt allerdings spezialisiert wurde. Die TWI-Kommunikation wird aus naheliegenden und praxisbezogenen Gründen in der Reihenfolge Slave/Master erläutert.

Im Folgenden wird auf diese Funktionen eingegangen:

USART Die serielle USART-Schnittstelle ermöglicht es, textuelle Rückgaben an den Benutzer zu machen. Dies ist zur Ausgabe des Infostatus oder als Debuginformation gedacht[12].

AD-Wandler Diese Implementierung wird benötigt, um die analog gelesenen Werte der Fototransistoren weiterverarbeiten zu können. Diese Funktion ist essentiell, da die Fototransistoren an wichtigen Punkten der Taktstraße platziert sind.

[10]vgl. Potentiometer
[11]Der komplette Quellcode befindet sich im Anhang.
[12]Das Hauptaugenmerk für die Benutzung der Schnittstelle lag auf der Verwendung als Ausgabemedium in der Entwicklungsphase

LCD-Display In der Realisierung des Projektes kam uns die Idee, dass es sinnvoll sei, dem Benutzer Rückmeldung über eventuell auftretende Fehler, sowie eine Möglichkeit der Ausgabe einer Zeichenkette zu geben, ohne dafür zu verlangen, dass dieser die serielle Schnittstelle zur Verwendung der USART auf seinem System implementiert hat; er wird ohne weitere nötige Hardware über den Status der Taktstraße informiert.

TWI Wie bereits beschrieben, soll das TWI-Protokoll benutzt werden, um die Möglichkeit zu geben, zwei Mikrocontroller untereinander Daten austauschen zu lassen, beziehungsweise die komplette Taktstraße steuern zu können. Das Protokoll wurde in dem Sinne angepasst, dass zur Steuerung nur ein Byte hin und zurück übertragen wird.

3.4.1 USART - Universal Synchronous and Asynchronous serial Receiver and Transmitter

Im Folgenden wird nur auf den Transmit-Modus der USART-Schnittstelle eingegangen; der Receive-Modus wurde nicht implementiert.

Um die USART zu verwenden definierten wir zwei Konstanten, um die Schnittstelle unabhängig von unserem System initialisieren zu können.

```
1 #define CLOCK 8000000UL
2 #define BAUD 9600UL
```

Wie bereits bei den Grundlagen zur USART angesprochen, ist eine Initialisierung der USART nötig und wird gemäß der bereits vorgestellten Bedingungen realisiert:

```
1  void usart_init(void)
2  {
3    unsigned char x;
4
5    // Baudrate setzen
6    UBRRH = (unsigned char)(((CLOCK/(16UL * BAUD)) - 1)>>8);
7    UBRRL = (unsigned char)((CLOCK/(16UL * BAUD)) - 1);
8
9    // Transmitter und Receiver einschalten
10   UCSRB |= (1 << TXEN ) | (1 << RXEN );
11   //8 Datenbits
12   UCSRC |= (1 << URSEL ) | (1 << UCSZ1 ) | (1 << UCSZ0 );
13
14   // Hilfsvariable leert den Empfänger
15   x = UDR;
16 }
```

Um nun Daten schreiben zu können, wird die oben angeführte Sendefunktion implementiert. In unserer Lösung nannten wir die Funktion *usart_write()*, und fügten noch eine Erkennung des Zeichens `\n` ein, um saubere Zeilenübergänge zu erhalten.

```
1 static int usart_write(char x, FILE *stream)
2 {
3   if (x == '\n')
4     usart_write('\r', stream);
5   loop_until_bit_is_set(UCSRA, UDRE);
6   UDR = x;
7   return 0;
8 }
```

Da wir die Funktion *printf()* überschreiben wollten, werden folgende zwei Zeilen benötigt, die die Zielfunktion für *printf()* auf die von uns geschriebene Funktion *usart_write()* umleiten.

```
1 static int usart_write(char x, FILE *stream);
2 static FILE mystdout = FDEV_SETUP_STREAM(usart_write, NULL,
      _FDEV_SETUP_WRITE);
```

Im Hauptprogramm sieht die Benutzung folgendermaßen aus:

```
1 int main(void)
2 {
3   ...
4   stdout = &mystdout;
5
6   ...
7   initusart();
8   printf("USART-Test \r\n");
9
10  ...
11  return 0;
12 }
```

3.4.2 AD-Wandler

Wir steuern den AD-Wandler wie in den Grundlagen beschrieben an, was zu folgender Implementierung führt.

Zuerst wird der Wandler aktiviert und der Wandlungsteiler festgesetzt:

```
5   ADCSRA |= (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1);
```

Dann weisen wir den übergebenen Kanal zu und legen die interne Referenzspannung von 2,56 V an:

```
7   ADMUX = channel;
8   ADMUX |= (1 << REFS1) | (1 << REFS0);
```

Nun aktivieren wir die Umwandlung und warten auf das Ende der Ausführung:

```
10  ADCSRA |= (1 << ADSC);
11  while(ADCSRA & (1 << ADSC));
```

und holen das Ergebnis aus dem Register **ADC**:

```
13  result = ADC;
```

Zum Schluss deaktivieren wir den Wandler folgedermaßen wieder:

```
15  ADCSRA |= (0<<ADEN);
```

Man erhält als Rückgabewert der Funktion den umgewandelten Analogwert des angegebenen Kanals.

3.4.3 Display

Da wir ein Display an die Platine anschließen wollten, entschieden wir uns für ein vierzeiliges Display mit jeweils 27 Zeichen[13]. Dies genügt völlig, um dem Benutzer eine sinnvolle Ausgabe zu geben. Dieses Display wird wie die meisten nach einem immer gleichen Standard angesprochen. Wie das Display angeschlossen wird ist in den Grundlagen beschrieben. Daher gehen wir im Folgenden nur auf die Funktionalitäten der Bibliothek ein. [14]

Das Display kann in einem 8-Bit-Modus und in einem 4-Bit-Modus betrieben werden. Da wir am Mikrocontroller jedoch keine Pins für acht Datenleitungen mehr zur Verfügung hatten, entschieden wir uns für den etwas aufwändigeren 4-Bit-Modus, in dem das Display mit den 4 Steuerleitungen und 4 Datenleitungen angesprochen wird.

Zuerst muss das Display initialisiert werden. Diese Funktion wird von uns bereit gestellt. Da wir die Bibliothek allgemein verwendbar geschrieben haben, sollte man jedoch alle benötigten Konstanten in der Header-Datei entsprechend belegen. Zunächst werden die Ports und die Datenrichtungen vereinbart. Mit diesen Konstanten kann man nun den einzelnen Pins des Displays die verlangten Werte zuweisen. Im Anschluss daran werden noch einfache Befehle vereinbart, die unter diesen Konstanten einen sinnvollen Namen erhalten. Die Belegung für dieses Projektes sieht folgendermaßen aus:

```
1 #define LCD_DATA_DDR      DDRD
2 #define LCD_DATA_PORT     PORTD
3 #define LCD_DATA_PIN      PIND
4
5 #define LCD_CTRL_DDR      DDRB
6 #define LCD_CTRL_PORT     PORTB
7 #define LCD_CTRL_PIN      PINB
```

[13] WINTEK WD-C2704M-1HNN, bestellbar unter http://www.pollin.de

[14] Zur weiteren Orientierung diente als weitere Quelle eine bekannte Implementierung einer Displayansteuerung [Fle].

```
8
9  #define LCD_DATA_D7_DDR   LCD_DATA_DDR
10 #define LCD_DATA_D7_PORT  LCD_DATA_PORT
11 #define LCD_DATA_D7_PIN   LCD_DATA_PIN
12 #define LCD_DATA_D7_P     3
13
14 #define LCD_DATA_D6_DDR   LCD_DATA_DDR
15 #define LCD_DATA_D6_PORT  LCD_DATA_PORT
16 #define LCD_DATA_D6_PIN   LCD_DATA_PIN
17 #define LCD_DATA_D6_P     2
18
19 #define LCD_DATA_D5_DDR   LCD_DATA_DDR
20 #define LCD_DATA_D5_PORT  LCD_DATA_PORT
21 #define LCD_DATA_D5_PIN   LCD_DATA_PIN
22 #define LCD_DATA_D5_P     1
23
24 #define LCD_DATA_D4_DDR   LCD_DATA_DDR
25 #define LCD_DATA_D4_PORT  LCD_DATA_PORT
26 #define LCD_DATA_D4_PIN   LCD_DATA_PIN
27 #define LCD_DATA_D4_P     0
28
29 #define LCD_CTRL_RW_DDR   LCD_CTRL_DDR
30 #define LCD_CTRL_RW_PORT  LCD_CTRL_PORT
31 #define LCD_CTRL_RW_PIN   LCD_CTRL_PIN
32 #define LCD_CTRL_RW_P     5
33
34 #define LCD_CTRL_RS_DDR   LCD_CTRL_DDR
35 #define LCD_CTRL_RS_PORT  LCD_CTRL_PORT
36 #define LCD_CTRL_RS_PIN   LCD_CTRL_PIN
37 #define LCD_CTRL_RS_P     4
38
39 #define LCD_CTRL_E1_DDR   LCD_CTRL_DDR
40 #define LCD_CTRL_E1_PORT  LCD_CTRL_PORT
41 #define LCD_CTRL_E1_PIN   LCD_CTRL_PIN
42 #define LCD_CTRL_E1_P     6
43
44 #define LCD_CTRL_E2_DDR   LCD_CTRL_DDR
45 #define LCD_CTRL_E2_PORT  LCD_CTRL_PORT
46 #define LCD_CTRL_E2_PIN   LCD_CTRL_PIN
47 #define LCD_CTRL_E2_P     7
48
49 #define LCD_CLR_DISPLAY              0x01
50 #define LCD_CUR_HOME                 0x02
51 #define LCD_ENTRY_MODE_SET_INC_V     0x07
52 #define LCD_ENTRY_MODE_SET_INC_C     0x06
53 #define LCD_ENTRY_MODE_SET_DEC_V     0x05
54 #define LCD_ENTRY_MODE_SET_DEC_C     0x04
55 #define LCD_DISPLAY_ON_U_B           0x0F
56 #define LCD_DISPLAY_ON_U             0x0E
57 #define LCD_DISPLAY_ON_B             0x0D
58 #define LCD_DISPLAY_ON               0x0C
59 #define LCD_DISPLAY_OFF              0x08
```

```
60 //NUR 4-BIT-ANSTEUERUNG!!!
61 #define LCD_SYS_SET_4BIT_24R_57_IO  0x28
```

Nun kann man die Initialisierungsfunktion einfach aufrufen. Sie funktioniert genau nach dem in den Grundlagen angegebenen Schema zur Initialisierung für den 4-Bit-Betrieb. Wir benötigen jedoch noch einige Funktionen.

Da dieses Display jedoch wie beschrieben 4-zeilig ist, und auch zwei Controller enthält, muss für jeden dieser Controller eine *apply*-Funktion zur Verfügung stehen, die die momentan anliegenden Werte an den Datenleitungen übernimmt, um so das Display zu steuern. Man zieht dazu die entsrechende enable-Leitung auf den High-Pegel und wieder auf den Low-Pegel zurück[15]. Dazu werden entsprechende Makros definiert, die diese Aufgaben bewältigen:

```
1 #define e1_set()  LCD_CTRL_E1_PORT |= (1<<LCD_CTRL_E1_P)
2 #define e1_clr()  LCD_CTRL_E1_PORT &= ~(1<<LCD_CTRL_E1_P)
3 #define e2_set()  LCD_CTRL_E2_PORT |= (1<<LCD_CTRL_E2_P)
4 #define e2_clr()  LCD_CTRL_E2_PORT &= ~(1<<LCD_CTRL_E2_P)
```

Nun kann der anliegende Wert einfach nurch Angabe des Controllers übernommen werden, wobei eine `0` bewirkt, dass beide Controller den anliegenden Wert übernehmen:

```
1 void lcd_apply(int controller)
2 {
3   //Controller auswählen, Pegel hoch, Pegel runter
4   if (controller==1)
5   {
6     e1_set();
7     e1_clr();
8   }
9   else
10    if (controller==2)
11    {
12      e2_set();
13      e2_clr();
14    }
15    else
16      if (controller==0)
17      {
18        e1_set();
19        e2_set();
20        e1_clr();
21        e2_clr();
22      }
23 }
```

Man benötigt des Weiteren eine Funktion, die den Zustand des Displays ausliest. Unter anderem benötigt man eine solche Funktion um das Busyflag auszulesen, welches an-

[15]hier auch: toggle

zeigt, ob das Display bereit ist Daten zu empfangen. Dies geschieht durch eine Funktion `lcd_get(int rs, int controller)`.

Man benötigt auch hier einige Makros, die die Zuweisungen an die Kontrollleitungen übernehmen:

```
5 #define rs_set()  LCD_CTRL_RS_PORT |= (1<<LCD_CTRL_RS_P)
6 #define rs_clr()  LCD_CTRL_RS_PORT &= ~(1<<LCD_CTRL_RS_P)
7 #define rw_set()  LCD_CTRL_RW_PORT |= (1<<LCD_CTRL_RW_P)
8 #define rw_clr()  LCD_CTRL_RW_PORT &= ~(1<<LCD_CTRL_RW_P)
```

Zunächst wird also der Lesemodus ausgewählt:

```
7   rw_set();
```

Nun wird anhand der übergebenen Parameter festgelegt, ob Daten oder der Zustand (inklusive Busy-Flag) ausgelesen werden sollen:

```
9   //BF/AC lesen
10  if (rs==0)
11    //Kontrolle
12    rs_clr();
13  //Daten lesen
14  else if (rs==1)
15    //Daten
16    rs_set();
```

Nun werden die Pins auf den Ports in den Lesemodus gebracht und die Leitungen sicherheitshalber explizit auf *low* gezogen:

```
18  //Lesemodus für die jeweiligen Pins an Ports
19  LCD_DATA_D7_DDR &= ~(1<<LCD_DATA_D7_P);
20  LCD_DATA_D6_DDR &= ~(1<<LCD_DATA_D6_P);
21  LCD_DATA_D5_DDR &= ~(1<<LCD_DATA_D5_P);
22  LCD_DATA_D4_DDR &= ~(1<<LCD_DATA_D4_P);
23
24  //Leitungen auf 0 ziehen (Zur Sicherheit)
25  LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
26  LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
27  LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
28  LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
```

Da nur vier Leitungen verfügbar sind, wird das Byte in zwei Schritten[16] abgeholt. Zunächst muss die gewählte Enable-Leitung des Displays auf *high* gezogen. Nur dann gibt das Display Daten zurück.

```
32  if (controller==1)
33    e1_set();
34  else if (controller==2)
35    e2_set();
```

[16]vgl. Nibble

Dann kann das Nibble gelesen und gespeichert werden:

```
37  if ((LCD_DATA_D7_PIN & (1<<LCD_DATA_D7_P))!=0x00) tmp |= 0x80;
38  if ((LCD_DATA_D6_PIN & (1<<LCD_DATA_D6_P))!=0x00) tmp |= 0x40;
39  if ((LCD_DATA_D5_PIN & (1<<LCD_DATA_D5_P))!=0x00) tmp |= 0x20;
40  if ((LCD_DATA_D4_PIN & (1<<LCD_DATA_D4_P))!=0x00) tmp |= 0x10;
```

Um das zweite Nibble abzuholen wird die Enable-Leitung wieder auf *low* gezogen:

```
43  if (controller==1)
44    e1_clr();
45  else if (controller==2)
46    e2_clr();
```

Um das zweite Nibble abzuholen geht man analog vor. Zuerst muss die Enable-Leitung auf *high* gezogen werden, nur dann kann das Display Daten zurückgeben, welche gelesen und gespeichert werden können. Anschließend wird die Enable-Leitung wieder auf *low* gezogen, und das Byte ist komplett.

```
50  if (controller==1)
51    e1_set();
52  else if (controller==2)
53    e2_set();
54
55  if ((LCD_DATA_D7_PIN & (1<<LCD_DATA_D7_P))!=0x00) tmp |= 0x08;
56  if ((LCD_DATA_D6_PIN & (1<<LCD_DATA_D6_P))!=0x00) tmp |= 0x04;
57  if ((LCD_DATA_D5_PIN & (1<<LCD_DATA_D5_P))!=0x00) tmp |= 0x02;
58  if ((LCD_DATA_D4_PIN & (1<<LCD_DATA_D4_P))!=0x00) tmp |= 0x01;
59
60  //Enable-Leitung auf Low-Pegel setzen
61  if (controller==1)
62    e1_clr();
63  else if (controller==2)
64    e2_clr();
```

Nun werden die entsprechenden Pins wieder auf Ausgang gesetzt und die Leitungen auf `0` gezogen:

```
66  //Port auf Ausgang stellen
67  LCD_DATA_D7_DDR |= (1<<LCD_DATA_D7_P);
68  LCD_DATA_D6_DDR |= (1<<LCD_DATA_D6_P);
69  LCD_DATA_D5_DDR |= (1<<LCD_DATA_D5_P);
70  LCD_DATA_D4_DDR |= (1<<LCD_DATA_D4_P);
71
72  //Leitungen auf 0 ziehen
73  LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
74  LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
75  LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
76  LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
```

Die Lesemethode ist damit vollständig, und kann benutzt werden, um unter anderem das Busyflag auszulesen. Dies geschieht in der dafür implementierten Prozedur `lcd_loopBusy(int controller)`:

```
81 void lcd_loopBusy(int controller)
82 {
83   while(1)
84   {
85     if ((lcd_get(0, controller) & 0x80)==0x00)
86       break;
87   }
88 }
```

Eine weitere Prozedur ist die Sendeprozedur, die einen Wert an einen Controller des Displays sendet. Man kann diese Prozedur *lcd_send(int rs, char c, int controller)* benutzen, anstatt die Pins wie am Anfang manuell zu belegen, und eine entsprechende Zeit zu warten.

Zunächst wird an dem angegeben Controller das Busy-Flag überprüft um sicherzugehen, dass das Display bereit ist. Es besteht die Möglichkeit die Daten an beiden Controllern des Displays durch Angabe einer `0` anzulegen.

```
3   if (controller==0)
4   {
5     lcd_loopBusy(1);
6     lcd_loopBusy(2);
7   }
8   else
9     if ((controller==1)|(controller==2))
10      lcd_loopBusy(controller);
```

Daraufhin wird der Schreibmodus aktiviert und gewählt, ob man einen Wert oder ein Kommando senden möchte. Möchte man einen Zeilenumbruch mit `\n` ausgeben, so wird diese Prozedur rekursiv mit dem Wert `(0x40|0x80)` aufgerufen. Danach wird sie verlassen.

```
12  rw_clr(); //Schreibe-Modus
13  if (rs==0) //Kommando senden
14    rs_clr();//Kontrollregister
15  else if (rs==1) //Daten senden
16  {
17    rs_set(); //Datenregister
18    if (c=='\n')
19    {
20      //New Line
21      lcd_send(0,(0x40|0x80),controller); //DD-Adresse 64 (Zeile2) (und
          D7 muss an sein) ->set_DDRam_Adress
22      return;
23    }
24  }
```

Analog zum Lesevorgang wird der zu übertragene Wert in zwei Nibbles aufgeteilt. Um das erste zu übertragen werden die Pins auf Ausgang gestellt und die entsprechenden Leitungen auf *0* gezogen.

```
26   //Port auf Ausgang stellen
27   LCD_DATA_D7_DDR |= (1<<LCD_DATA_D7_P);
28   LCD_DATA_D6_DDR |= (1<<LCD_DATA_D6_P);
29   LCD_DATA_D5_DDR |= (1<<LCD_DATA_D5_P);
30   LCD_DATA_D4_DDR |= (1<<LCD_DATA_D4_P);
31
32   //Leitungen auf 0 ziehen
33   LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
34   LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
35   LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
36   LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
```

Nun wird das erste Nibble mit den oberen 4 Bits angelegt und der angegebene Controller übernimmt die Daten:

```
42   if ((0x80 & c)==0x80) LCD_DATA_D7_PORT |= (1<<LCD_DATA_D7_P);
43   if ((0x40 & c)==0x40) LCD_DATA_D6_PORT |= (1<<LCD_DATA_D6_P);
44   if ((0x20 & c)==0x20) LCD_DATA_D5_PORT |= (1<<LCD_DATA_D5_P);
45   if ((0x10 & c)==0x10) LCD_DATA_D4_PORT |= (1<<LCD_DATA_D4_P);
46
47   if (controller==1)
48   {
49     lcd_apply(1);
50   }
51   else
52     if (controller==2)
53     {
54       lcd_apply(2);
55     }
56     else
57       if (controller==0)
58       {
59         lcd_apply(0);
60       }
```

Um das zweite Nibble zu übertragen werden die Leitungen erneut auf `0` gezogen.

```
63   LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
64   LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
65   LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
66   LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
```

Analog werden nun die unteren vier Bits übertragen. Die Daten werden zugewiesen und die Controller übernehmen die Daten:

```
70   if ((0x08 & c)==0x08) LCD_DATA_D7_PORT |= (1<<LCD_DATA_D7_P);
71   if ((0x04 & c)==0x04) LCD_DATA_D6_PORT |= (1<<LCD_DATA_D6_P);
72   if ((0x02 & c)==0x02) LCD_DATA_D5_PORT |= (1<<LCD_DATA_D5_P);
```

```
73    if ((0x01 & c)==0x01) LCD_DATA_D4_PORT |= (1<<LCD_DATA_D4_P);
74
75    if (controller==1)
76    {
77      lcd_apply(1);
78    }
79    else
80      if (controller==2)
81      {
82        lcd_apply(2);
83      }
84      else
85        if (controller==0)
86        {
87          lcd_apply(0);
88        }
```

Zum Schluss werden die Datenleitungen erneut auf `0` gezogen:

```
91    LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
92    LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
93    LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
94    LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
```

Das Display kann nun durch die implementierten Methoden anhand der in den Grundlagen beschriebenen Sequenz initialisiert werden:

```
1  void lcd_init(int controller)
2  {
3    //ACHTUNG: BUSYFLAG KANN AM ANFANG NOCH NICHT ABGEFRAGT WERDEN,
4    //DAHER EINFACH ENTSPRECHEND LANGE WARTEN -> DELAY-FUNKTION
5
6    //Port (Kontrolle) auf Ausgang
7    LCD_CTRL_RW_DDR |= (1<<LCD_CTRL_RW_P);
8    LCD_CTRL_RS_DDR |= (1<<LCD_CTRL_RS_P);
9    LCD_CTRL_E1_DDR |= (1<<LCD_CTRL_E1_P);
10   LCD_CTRL_E2_DDR |= (1<<LCD_CTRL_E2_P);
11   //Port (Daten) auf Ausgang
12   LCD_DATA_D7_DDR |= (1<<LCD_DATA_D7_P);
13   LCD_DATA_D6_DDR |= (1<<LCD_DATA_D6_P);
14   LCD_DATA_D5_DDR |= (1<<LCD_DATA_D5_P);
15   LCD_DATA_D4_DDR |= (1<<LCD_DATA_D4_P);
16
17   //Leitungen auf 0 ziehen
18   //Kontrolle
19   LCD_CTRL_RW_PORT &= ~(1<<LCD_CTRL_RW_P);
20   LCD_CTRL_RS_PORT &= ~(1<<LCD_CTRL_RS_P);
21   LCD_CTRL_E1_PORT &= ~(1<<LCD_CTRL_E1_P);
22   LCD_CTRL_E2_PORT &= ~(1<<LCD_CTRL_E2_P);
23   //Daten
24   LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
25   LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
```

```
26    LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
27    LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
28
29    //Init-Sequenz
30    delay_ms(15);
31    //3h ins Steuerregister
32    LCD_DATA_D5_PORT |= (1<<LCD_DATA_D5_P);
33    LCD_DATA_D4_PORT |= (1<<LCD_DATA_D4_P);
34    lcd_apply(controller);
35    delay_ms(5);
36    //nochmal 3h ins Steuerregister
37    lcd_apply(controller);
38    delay_ns(100);
39    //ein drittes Mal 3h ins Steuerregister
40    lcd_apply(controller);
41
42    //4-Bit Datenbus
43    //2h schreiben
44    LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
45    lcd_apply(controller);
46
47    //AB HIER WERDEN DIE BYTES IN 4-BIT-BLÖCKEN VERARBEITET!!!
48    //BUSY-FLAG KANN AB JETZT ABGEFRAGT WERDEN!!!
49
50    //System set
51    lcd_send(0, LCD_SYS_SET_4BIT_24R_57_IO, controller);
52    //Display on
53    lcd_send(0, LCD_DISPLAY_ON, controller);
54    //Clear Display
55    lcd_send(0, LCD_CLR_DISPLAY, controller);
56    //Cursor home
57    lcd_send(0, LCD_CUR_HOME, controller);
58    //Entrymode set
59    lcd_send(0, LCD_ENTRY_MODE_SET_INC_C, controller);
60  }
```

Ist das Display initialisiert, kann man mit folgender Funktion Werte darauf ausgeben. Man gibt als Argument an die Funktion nur den auszugebenden String, sowie den Controller an, auf dem der Text ausgegeben werden soll:

```
1  void lcd_prints(char *s, int controller)
2  {
3    //For-Schleife für jeden char einzeln
4    int len = strlen(s);
5    for (int i = 0; i<len;i++)
6    {
7      lcd_send(1, s[i], controller);
8    }
9  }
```

3.4.4 TWI - Kommunikationsspezialisierung

Die von uns entwickelte TWI-Kommunikation ist eine leichte Abwandlung des in den Grundlagen beschriebenen Protokolls. Sie beschränkt sich in einer Kommunikation auf das Austauschen von zwei Bytes.

Nach der Aufnahme der Verbindung mit dem Senden eines START-Befehls und der daraufhin folgenden Information des Schreibmodus durch das Senden des SLA+W-Befehls, welchen der Slave mit einem ACK beantwortet, befindet sich der Master im *Master-Transmitter-Modus* und der Slave im *Slave-Receiver-Modus*. Nun wird ein Byte vom Master zum Slave gesendet, welches dieser mit einem NACK beantwortet. Damit ist der Sendevorgang beendet.

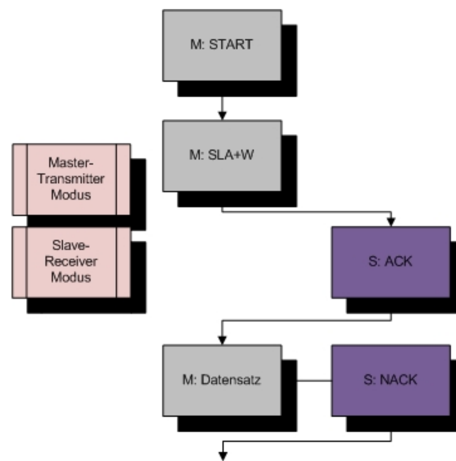


Abbildung 3.11: TWI-Kommunikationsablauf: Senden

Durch einen daraufhin zu sendenden REPEATED-START-Befehl behält der Master den Bus und adressiert den selben Slave mit dem Befehl SLA+R erneut. Dieser antwortet mit einem ACK und begibt sich in den *Slave-Transmitter-Modus*, während der Master in den *Master-Receiver-Modus* wechselt. Nach dem Erhalt eines Datensatzes des Slaves, welches vom Master mit einem NACK beantwortet werden muss, da ein Byte als Antwort ausreicht, ist auch dieser Sendevorgang beendet und der Master beendet die Verbindung durch das Senden eines STOP-Befehls.

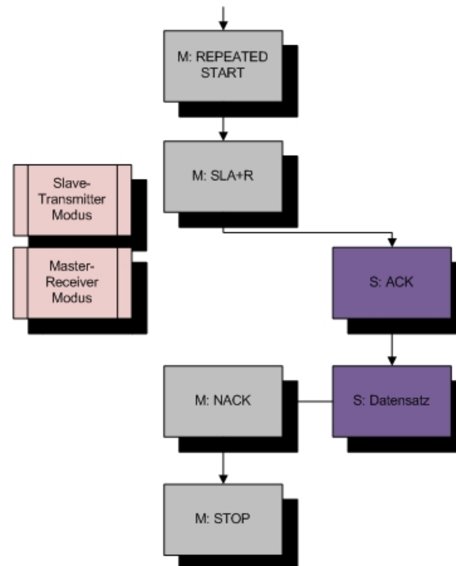


Abbildung 3.12: TWI-Kommunikationsablauf: Empfangen

Die Zwei-Byte-Kommunikation über den TWI-Bus ist abgeschlossen.

3.4.5 TWI - Two-Wire Interface (Slave)

Im Folgenden wird der TWI-Service der Taktstraße angesprochen, dies entspricht dem Slave[17].

Um eine wie weiter oben beschriebene TWI-Kommunikation zu implementieren, wird das TWI nur initialisiert, und kann dann per *ISR*[18] und dem Argument `TWI_vect` benutzt werden. Diese Funktion triggert, sobald ein entsprechender Interrupt augelöst wird, das heißt, sobald über den TWI-Bus zu verarbeitende Daten anliegen. Dieser Interrupt-Service wird mit der Funktion *sei();*[19] aktiviert und mit der Funktion *cli();* deaktiviert.

Eine Initialisierung sieht folgendermaßen aus:

```
1 void twi_init(void)
2 {
3   // Initialisierung TWI-Register
4   TWCR = 0x00;
5   TWSR = 0x00;
6   PORTC |= 0x03;
7
```

[17]Der angepasste TWI-Master wird im Hardwarepraktikum der Universität Koblenz als Lehraufgabe entwickelt, siehe auch Kapitel 3.5.

[18]Interrupt Service Routine

[19]In der *main*-Funktion muss diese Funktion aufgerufen werden, um die ISR überhaupt benutzen zu können.

```
8    // Slave-Receiver-Modus initialisieren
9    TWAR = TAKTSTRASSE;
10   TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWIE) | (1 << TWINT);
11 }
```

Die ISR kann nun modelliert werden. Zunächst wartet der Slave bis dieser die eigene Adresse mitsamt der Information zur Durchführung eines Schreibvorgangs empfängt.

```
8    loop_until_bit_is_set(TWCR, TWINT);
```

Der Slave befindet sich im Normalfall nun in einem Status, in dem er dem Master auf seine eigene Slave-Adresse ein ACK zurückgesendet hat. Er nimmt nun nun die Rolle des Slave-Receivers ein, während der Master die des Transmitters übernimmt.

In dem folgenden Codestück wird der momentane Status ausgewertet und ein Datum empfangen, auf das ein NACK des Slaves folgt.

```
12   if (TW_STATUS == TW_SR_SLA_ACK)
13   {
14     // Byte wird empfangen und ein NACK zurückgesendet
15     TWCR = (1 << TWINT) | (1 << TWEN);
16     loop_until_bit_is_set(TWCR, TWINT);
17   }
18   else
19   {
20     // Kein Slave-Receiver-Modus
21     // Setze Slave in Ausgangslage (empfangsbereit)
22     lcd_cls(2);
23     lcd_prints("FEHLER:\nKEIN SR-MODUS",2);
24     twi_init();
25     PORTA |= 0x80;
26     return;
27   }
```

Nun wird der Status ausgewertet und der Slave gibt den Befehl zum Wechsel in den Slave-Transmitter-Modus.

```
32   switch(TW_STATUS)
33   {
34     // Daten wurden empfangen, NACK wurde gesendet
35     case TW_SR_DATA_NACK:
36       recbuf = TWDR;
37       // Setze slave in not-adressed-slave-mode
38       TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN);
39       transmode = 1;
40       break;
41     default:
42       // Kein Byte empfangen
43       // Setze Slave in Ausgangslage (empfangsbereit)
44       lcd_cls(2);
45       lcd_prints("FEHLER:\nKEIN DATA+NACK",2);
```

```
46
47        twi_init();
48        PORTA |= 0x80;
49        return;
50    }
```

Der Slave wartet auf die eigene Adressierung mit der Information, dass dieser ein Datum zurücksenden soll.

```
54    loop_until_bit_is_set(TWCR, TWINT);
```

Der Slave befindet sich nun im Slave-Transmitter-Modus, während der Master in den Master-Receiver-Modus wechselt.

Die Reaktion des Slaves auf das zuvor empfangene Byte ist variabel:

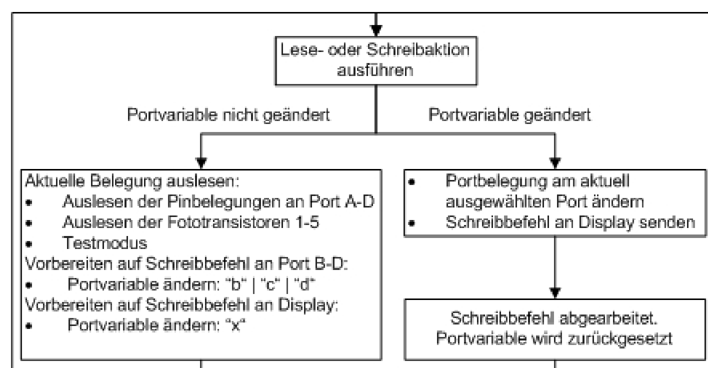


Abbildung 3.13: Reaktion des Slaves auf ein Datum

Soll vom Slave gelesen werden, so gelten die folgenden Vereinbarungen:

- Wird ein **t** gesendet, so wird der Testwert `0x00` zurückgegeben.
- Wird ein **x** gesendet, so wird das Display gelöscht und der Kontrollwert `0x11` zurückgegeben.
- Wird ein **d** gesendet, so wird der Kontrollwert `0x10` zurückgegeben und die Portvariable wird auf auf den speziellen Wert `x` geändert.
- Wird ein Port
  - **0xbe**
  - **0xce**
  - **0xde**

  angegeben, so startet die Portauswahl. Es wird ein Kontrollwert `0xbf`, `0xcf` oder `0xdf` zurückgegeben und die Portvariable auf `b`, `c` oder `d` geändert.
- Einer der folgenden Werte sorgt dafür, dass die jeweilige momentan anliegende Portbelegung zurückgegeben wird:

0xaa Port A
0xbb Port B
0xcc Port C
0xdd Port D

- Wird einer folgenden Werte empfangen, wird der Wert des angeforderten Analog-Digital-Wandlers zurückgegeben:

  0xd0 DA-Wandler 1
  0xd1 DA-Wandler 2
  0xd2 DA-Wandler 3
  0xd3 DA-Wandler 4
  0xd4 DA-Wandler 5

Ist bereits eine Portauswahl zum Schreiben der Ports geschehen, so wird das vom Slave empfangene Byte dem zuvor gespeichertem Port zugewiesen.

Wurde die Portvariable bereits mit `x` belegt, so wird das nächste Byte auf dem Display ausgegeben.

```
66          if (port == 'r')
67          {
68            // Lesemodus oder Portauswahlmodus
69            switch (recbuf)
70            {
71              // Testfall
72              case 't':
73                TWDR = 0x00;
74                break;
75              // Displayausgabe vorbereiten
76              case 'd':
77                port = 'x';
78                TWDR = 0x10;
79                break;
80              // Display löschen
81              case 'x':
82                lcd_cls(2);
83                TWDR = 0x11;
84                break;
85              // Wenn Port angegeben wurde Port auswählen
86              case 0xbe:
87                port = 'b';
88                TWDR = 0xbf;
89                break;
90              case 0xce:
91                port = 'c';
92                TWDR = 0xcf;
93                break;
94              case 0xde:
95                port = 'd';
96                TWDR = 0xdf;
```

```
97              break;
98            // Pins auslesen
99            case 0xaa:
100             DDRA = 0x00;
101             TWDR = PINA;
102             DDRA = 0x80;
103             break;
104           case 0xbb:
105             DDRB = 0x00;
106             TWDR = PINB;
107             DDRB = 0xff;
108             break;
109           case 0xcc:
110             DDRC = 0x00;
111             TWDR = PINC;
112             DDRC = 0xf0;
113             break;
114           case 0xdd:
115             DDRD = 0x00;
116             TWDR = PIND;
117             DDRD = 0xff;
118             break;
119           // Fototransistoren abfragen
120           case 0xd0:
121             TWDR = (char)(readADC(0x00) >> 2);
122             break;
123           case 0xd1:
124             TWDR = (char)(readADC(0x01) >> 2);
125             break;
126           case 0xd2:
127             TWDR = (char)(readADC(0x02) >> 2);
128             break;
129           case 0xd3:
130             TWDR = (char)(readADC(0x03) >> 2);
131             break;
132           case 0xd4:
133             TWDR = (char)(readADC(0x04) >> 2);
134             break;
135           default:
136             //Unbekannter KontrollCode!
137             lcd_cls(2);
138             lcd_prints("FEHLER:\nUNBEKANNTER KONTROLLCODE",2);
139             TWDR = recbuf;
140             twi_init();
141             PORTA |= 0x80;
142             return;
143         }
144       }
145       else
146       {
147         switch (port)
148         {
```

```
149             case 'b':
150               PORTB = recbuf;
151               TWDR = 0xbb;
152               port = 'r';
153               break;
154             case 'c':
155               PORTC = recbuf;
156               TWDR = 0xcc;
157               port = 'r';
158               break;
159             case 'd':
160               PORTD = recbuf;
161               TWDR = 0xdd;
162               port = 'r';
163               break;
164             case 'x':
165               lcd_send(1, recbuf, 2);
166               TWDR = 0x12;
167               port = 'r';
168               break;
169           }
170         }
```

Der Slave sendet das nun im Register TWDR abgelegte Byte zurück.

```
173         TWCR = (1 << TWEN) | (1 << TWINT);
174         loop_until_bit_is_set(TWCR, TWINT);
```

Auf das übertragene Byte sendet der Master ein Nack zurück und der Slave nimmt sich selbst vom Bus[20]. Die Übertragung ist damit beendet, und der Slave kann erneut durch das Senden seiner eigenen Adresse und der Information, dass der Master schreiben will, angesprochen werden.

```
180         TWCR = (1 << TWEA) | (1 << TWEN) | (1 << TWINT) | (1 << TWIE);
```

3.4.6 TWI - Two-Wire Interface (Master)

In diesem Kapitel wird die Senderoutine des TWI-Masters vorgestellt. Er gibt seine Befehle an die Taktstraße (den Slave) weitert.

Eine notwendige Initialisierung des Masters sieht folgendermaßen aus:

```
1 void twi_init(void)
2 {
3   // Initialisieren des ATMega16 zur
4   // Verwendung des TWI inkl. interner
```

[20] Eventuelle Fehler werden in der Kommunikation abgefangen und der Slave wechselt in den nicht-adressierten Modus. Zur Kontrolle leuchtet die rote Status-LED auf der Kontrollplatine auf.

```
5   // Pullup-Widerstände
6   DDRC = 0x00;
7   PORTC = 0x03;
8
9   //Initialisierung des TWI
10  TWAR = 0x00;
11  //Einstellen des langsamsten Takts
12  TWBR = 0xff;
13  TWCR = 0x00;
14  TWDR = 0x00;
15  TWSR = 0x00;
16 }
```

Die Funktion, in der der Sender dem Empfänger seine Daten schickt heißt *twi_ send(unsigned char adres, unsigned char daten)* und wird immer aufgerufen mit der Adresse des Slaves und dem dazugehörigen Datum (z.B. `twi_send(AUSAD,0x00)`).

Um eine Kommunikation aufzubauen legt der Master einen Start-Befehl auf den Bus und wartet darauf, dass das `TWINT`-Flag von der Hardware gesetzt wird[21].

```
7   TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
8   loop_until_bit_is_set(TWCR, TWINT);
```

Nun wird der Slave durch das Senden seiner eigenen Adresse und der Information, dass der Master Daten senden möchte, angesprochen. Diese Information wird in das Register `TWDR` geladen. Nachdem das `INT`-Flag gesetzt wurde befindet sich der Master nun im Master-Transmitter-Modus und der Slave geht in den Slave-Receiver-Modus über.

```
11  TWDR = (adres) | (TW_WRITE);
12  TWCR = (1 << TWINT) | (1 << TWEN);
13  loop_until_bit_is_set(TWCR, TWINT);
```

Im Folgenden werden die Daten in das Register `TWDR` geladen und daraufhin gesendet:

```
16  TWDR = daten;
17  TWCR = (1 << TWINT) | (1 << TWEN);
18  loop_until_bit_is_set(TWCR, TWINT);
```

Durch einen erneuten Start-Befehl wird nun derselbe Slave erneut adressiert:

```
21  TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
22  loop_until_bit_is_set(TWCR, TWINT);
```

Durch die nun folgende Adressierung mit der Information, dass der Master Daten lesen möchte, bewirkt einen Wechsel des Masters in den Master-Receiver-Modus und des Slave in den Slave-Transmitter-Modus:

[21] Entgegen der normalen Funktionalität bewirkt das Schreiben einer `1` in das Bit `TWINT` ein Löschen, das Schreiben einer `0` entspricht dem Setzen des Bits.

```
31          TWDR = (adres) | (TW_READ);
32          TWCR = (1 << TWINT) | (1 << TWEN);
33          loop_until_bit_is_set(TWCR, TWINT);
```

Hat der Slave auf seine Adresse mit einem `ACK` reagiert, wird der Empfang der Daten vorbereitet:

```
40          TWCR = (1<<TWEN) | (1<<TWINT);
41          loop_until_bit_is_set(TWCR, TWINT);
```

Nachdem das Byte vom Master mit einem `NACK` beantwortet wurde, wird das empfangene Byte gesichert und ein STOP-Befehl auf den Bus gelegt. Die Verbindung ist damit beendet[22].

```
47          databuf = TWDR;
```

```
49          TWCR = (1 << TWEN)|(1 << TWSTO)|(1<<TWINT);
50          loop_until_bit_is_set(TWCR,TWSTO);
```

3.4.7 Hauptprogramm und Funktionen

Zusammengefasst ergeben diese Funktionalitäten folgende Funktionen.

Unsere *main*-Funktion sieht also folgendermaßen aus:

```
1  int main(void)
2  {
3    // Globales Interrupt-Flag
4    sei();
5
6    // Initialisierungen
7    atm16_init();
8    twi_init();
9
10   lcd_init(0);
11   lcd_prints("XXXXX Taktstrasse TWI XXXXX\nXXXXX Ansteuerung TWI XXXXX"
         , 1);
12
13   while(1)
14   {
15   }
16
17   cli();
18   return 0;
19 }
```

[22]Fehler in der Kommunikation werden abgefangen und der Master sendet in diesem Fall einen STOP-Befehl.

Zuerst wird das globale Interrupt-Flag gesetzt, und der Mikrocontroller sowie das TWI initialisiert. Die zugehörige Initialisierungsfunktion sieht folgendermaßen aus:

```
1 void atm16_init(void)
2 {
3   // Vereinbaren der Portrichtung
4   // 0 = Eingang
5   // 1 = Ausgang
6   DDRA = 0x80;
7   DDRB = 0xff;
8   DDRC = 0xf0;
9   DDRD = 0xff;
10
11  // Pullups für Taster und TWI
12  PORTA = 0x60;
13  PORTB = 0x00;
14  PORTC = 0x0f;
15  PORTD = 0x00;
16 }
```

Die Ports werden gemäß ihren Funktionen initialisiert.

```
3   // Globales Interrupt-Flag
4   sei();
5
6   // Initialisierungen
7   atm16_init();
8   twi_init();
```

Daraufhin wird das LCD-Display initialisiert und eine Nachricht auf dem ersten (oberen) Controller ausgegeben. Das Programm läuft nun in einer Endlosschleife und wartet auf die Abarbeitung von Daten, die über den TWI-Bus gesendet werden.

```
10  lcd_init(0);
11  lcd_prints("XXXXX Taktstrasse TWI XXXXX\nXXXXX Ansteuerung TWI XXXXX"
        , 1);
12
13  while(1)
14  {
15  }
16
17  cli();
18  return 0;
```

Um das Display von einem TWI-Master aus anzusprechen, wird im Folgenden noch eine beispielhafte Methode vorgestellt, die diese Aufgabe übernimmt. Zuerst wird das Display gelöscht und dann jedes einzelne Zeichen der Zeichenkette zum TWI-Slave übertragen.

```
1 void twi_display(char* str)
2 {
3   int j;
4
```

```
5      // Display löschen
6      twi_send(TAKTSTRASSE, 'x');
7
8      for (j=0; j<strlen(str); j++)
9      {
10       // Display mit einem Zeichen beschreiben
11       twi_send(TAKTSTRASSE, 'd');
12       twi_send(TAKTSTRASSE, str[j]);
13     }
14   }
```

Nun folgt abschließend ein kurzer Überblick über alle implementierten Funktionen, geordnet nach Themengebiet:

USART Serielle Schnittstelle (Nicht im Slave enthalten.)

- *void usart_init(void)*
- *static int usart_write(char x, FILE *stream)*

TWI Two-Wire Interface

- *void twi_init(void)*
- *ISR(TWI_vect)*
- *unsigned char twi_send(unsigned char adres, unsigned char daten)*
- *void twi_display(char* str)*

AD-Wandler Analog-Digital-Wandler

- *int readADC(char channel)*

LCD-Display Display

- *void delay_ns(unsigned int ns)*
- *void delay_ms(unsigned int ms)*
- *void lcd_apply(int controller)*
- *void lcd_cls(int controller)*
- *void lcd_prints(char *s, int controller)*
- *void lcd_send(int rs, char c, int controller)*
- *char lcd_get(int rs, int controller)*
- *void lcd_loopBusy(int controller)*
- *void lcd_init(int controller)*

Hauptprogramm Hauptprogramm

- *void atm16_init(void)*
- *int main(void)*

3.5 Ansteuerung der Taktstraße

Die Ansteuerung der Taktstraße soll nach folgendem Diagramm realisiert werden:

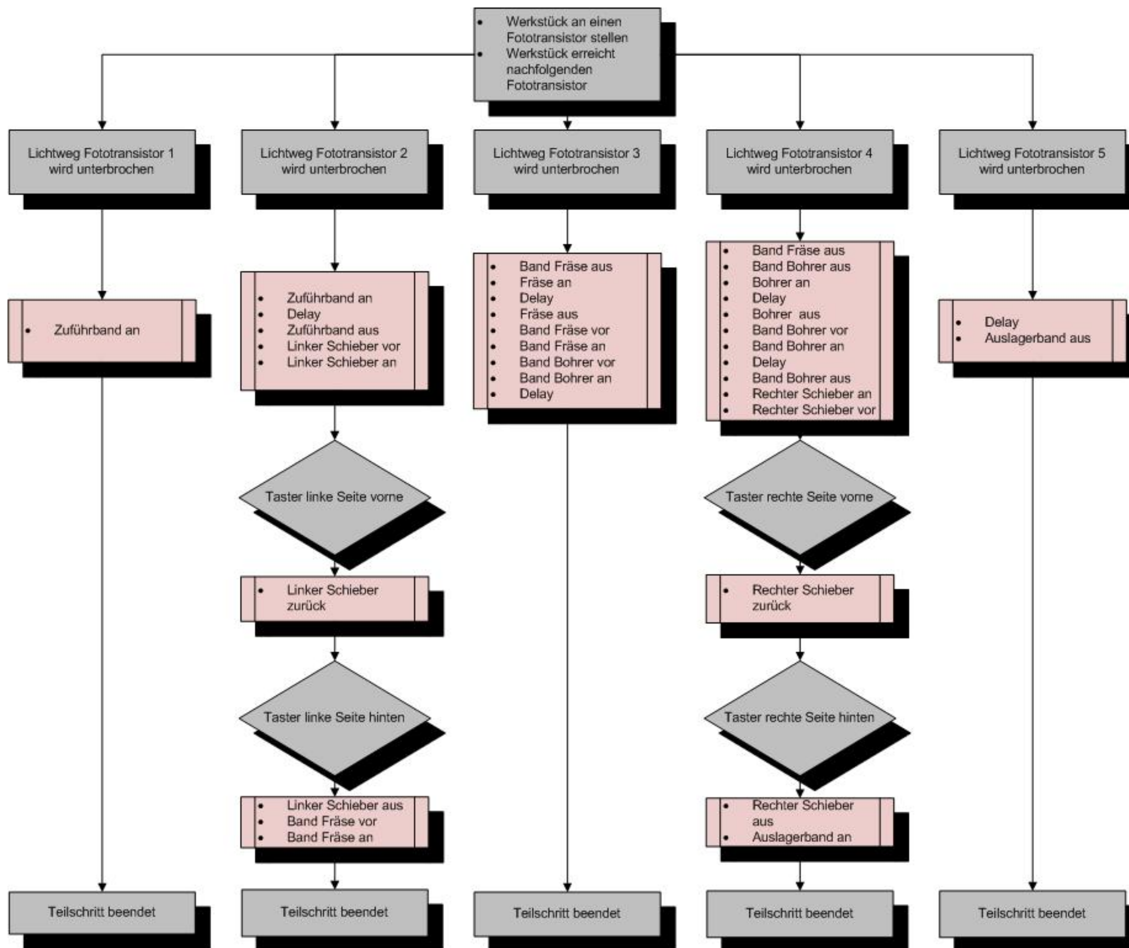


Abbildung 3.14: Flußdiagramm der Taktstraße

Vor jeder Reaktion der Taktstraße muss der Lichtweg von einem der fünf Fototransistoren von einem Werkstück unterbrochen werden. Ist dies geschehen, so soll die Taktstraße so reagieren, wie das obenstehende Diagramm aufzeigt. Die Taktstraße kann auf einen von neun möglichen äußeren Einflüssen reagieren. Entweder schließt einer der vier Taster oder es wird bei einem der fünf Lichtschranken der Lichtweg unterbrochen. Die Reaktionen der Taster und Motoren richten sich dabei nach den Teilschritten, die durch die Lichtunterbrechung bei den Fototransistoren aufgerufen werden.

Die Taktstraße durchläuft bei einem Durchlauf mit einem Werkstück die oben im Diagramm beschriebenen Teilabschnitte wie folgt:

- Abschnitt Einlegestation(Lichtschranke 1)

- Abschnitt linker Schieber(Lichtschranke 2)
- Abschnitt Fräsmaschine(Lichtschranke 3)
- Abschnitt Bohrer(Lichtschranke 4)
- Abschnitt Auslagerband(Lichtschranke 5)

Die Befehle, die zur Kommunikation mit der Taktstraße führen, werden in der Hauptfunktion abgehandelt. Für das Senden jedes einzelnen Befehls wird die Funktion *unsigned char twi_send(unsigned char adres, unsigned char daten)* benutzt. In unserer Lösung wird für jeden Befehl mit zugehöriger Antwort eine eigene Verbindung aufgebaut. Prinzipiell könnte man auch mit dem TWI eine Verbindung öffnen und anschließend einen ganzen Satz Daten verschicken. Unsere Lösung sieht vor, zu jedem Datum eine Verbindung aufzubauen. Für unsere Taktstraße haben wir unsere Befehle in neun Bereiche unterteilt, die Reaktionen der vier Taster und die Reaktionen der fünf Lichtschranken.

Nachfolgend wird auf eine mögliche Musterlösung der Taktstrasse eingegangen:

Damit die jeweilige Portbelegung auf den Ports B,C und D geändert werden kann, benötigt man die aktuell anliegende Portbelegung. Mit Hilfe dieser ist es möglich, immer genau einen Pin zu verändern. Zu Beginn der Kommunikation lesen wir einmalig alle Ports aus und hinterlegen deren aktuell anliegende Belegung auf den Zwischenspeichern `byteB`, `byteC` und `byteD`.

```
1 unsigned char byteB;
2 unsigned char byteC;
3 unsigned char byteD;
```

```
15   byteB = twi_send(TAKTSTRASSE, 0xbb);
16   byteC = twi_send(TAKTSTRASSE, 0xcc);
17   byteD = twi_send(TAKTSTRASSE, 0xdd);
```

Im Anschluss betritt der Master die Hauptschleife, in der dieser auf das Auslösen einer der fünf Fototransistoren wartet.

Fototransistor an der Einlegestation wird ausgelöst

Der Ablauf, der durchgeführt wird, sobald der Lichtweg des Fototransistors an der Einlegestation durchbrochen wurde, sieht folgendermaßen aus:

```
22     if (twi_send(TAKTSTRASSE, 0xd0) >= 0xaa)
23     {
```

(1) Zuführband wird angeschaltet

```
28         twi_send(TAKTSTRASSE, 0xce);
29         byteC |= 0x10;
30         twi_send(TAKTSTRASSE, byteC);
31     }
```

Fototransistor am linken Schieber wird ausgelöst

Der Ablauf, der durchgeführt wird, sobald der Lichtweg des Fototransistors am linken Schieber durchbrochen wurde, sieht folgendermaßen aus:

```
34    if (twi_send(TAKTSTRASSE, 0xd1) >= 0xaa)
35    {
```

(1) Zuführband wird angeschaltet

```
50        twi_send(TAKTSTRASSE, 0xce);
51        byteC |= 0x10;
52        twi_send(TAKTSTRASSE, byteC);
```

(2) Verzögerungsschleife um das Werkstück bis zum Ende des Laufbandes zu transportieren

```
55        for (i=1; i<45; i++)
56          _delay_ms(32);
```

(3) Zuführband wird ausgeschaltet

```
59        twi_send(TAKTSTRASSE, 0xce);
60        byteC &= ~(0x10);
61        twi_send(TAKTSTRASSE, byteC);
```

(4) Linker Schieber auf Richtung *vorwärts* einstellen

```
64        twi_send(TAKTSTRASSE, 0xde);
65        byteD |= (0x40);
66        twi_send(TAKTSTRASSE, byteD);
```

(5) Linker Schieber wird angeschaltet

```
69        twi_send(TAKTSTRASSE, 0xde);
70        byteD |= (0x80);
71        twi_send(TAKTSTRASSE, byteD);
```

(6) Betreten einer Warteschleife, bis der vordere Taster des linken Schiebers aktiviert wird

```
74        while((twi_send(TAKTSTRASSE, 0xcc) & 0x04) != 0x00)
75        {
76        }
```

(7) Linker Schieber auf Richtung *rückwärts* einstellen

```
79        twi_send(TAKTSTRASSE, 0xde);
80        byteD &= ~(0x40);
81        twi_send(TAKTSTRASSE, byteD);
```

(8) Betreten einer Warteschleife, bis der hintere Taster des linken Schiebers aktiviert wird

```
84        while((twi_send(TAKTSTRASSE, 0xcc) & 0x08) != 0x00)
85        {
86        }
```

(9) Linker Schieber wird ausgeschaltet

```
89        twi_send(TAKTSTRASSE, 0xde);
90        byteD &= ~(0x80);
91        twi_send(TAKTSTRASSE, byteD);
```

(10) Laufband der Fräse auf Richtung *vorwärts* einstellen

```
94        twi_send(TAKTSTRASSE, 0xce);
95        byteC |= (0x20);
96        twi_send(TAKTSTRASSE, byteC);
```

(11) Laufband der Fräse wird angeschaltet

```
99        twi_send(TAKTSTRASSE, 0xde);
100       byteD |= (0x20);
101       twi_send(TAKTSTRASSE, byteD);
102     }
```

Fototransistor an der Fräsmaschine wird ausgelöst

Der Ablauf, der durchgeführt wird, sobald der Lichtweg des Fototransistors an der Fräsmaschine durchbrochen wurde, sieht folgendermaßen aus:

```
105     if (twi_send(TAKTSTRASSE, 0xd2) >= 0xaa)
106     {
```

(1) Laufband der Fräse wird ausgeschaltet

```
119       twi_send(TAKTSTRASSE, 0xde);
120       byteD &= ~(0x20);
121       twi_send(TAKTSTRASSE, byteD);
```

(2) Motor der Fräse wird angeschaltet

```
124       twi_send(TAKTSTRASSE, 0xce);
125       byteC |= (0x40);
126       twi_send(TAKTSTRASSE, byteC);
```

(3) Verzögerungsschleife um das Werkstück von der Fräse bearbeiten zu lassen

```
129       for (i=1; i<45; i++)
130         _delay_ms(32);
```

(4) Motor der Fräse wird ausgeschaltet

```
133        twi_send(TAKTSTRASSE, 0xce);
134        byteC &= ~(0x40);
135        twi_send(TAKTSTRASSE, byteC);
```

(5) Laufband der Fräse auf Richtung *vorwärts* einstellen

```
138        twi_send(TAKTSTRASSE, 0xce);
139        byteC |= (0x20);
140        twi_send(TAKTSTRASSE, byteC);
```

(6) Laufband der Fräse wird angeschaltet

```
143        twi_send(TAKTSTRASSE, 0xde);
144        byteD |= (0x20);
145        twi_send(TAKTSTRASSE, byteD);
```

(7) Laufband des Bohrers auf Richtung *vorwärts* einstellen

```
148        twi_send(TAKTSTRASSE, 0xbe);
149        byteB |= (0x01);
150        twi_send(TAKTSTRASSE, byteB);
```

(8) Laufband des Bohrers wird angeschaltet

```
153        twi_send(TAKTSTRASSE, 0xde);
154        byteD |= (0x10);
155        twi_send(TAKTSTRASSE, byteD);
```

(9) Verzögerungsschleife um den Fototransistor der Fräse zu verlassen

```
158        for (i=1; i<45; i++)
159          _delay_ms(32);
160      }
```

Fototransistor am Bohrer wird ausgelöst

Der Ablauf, der durchgeführt wird, sobald der Lichtweg des Fototransistors am Bohrer durchbrochen wurde, sieht folgendermaßen aus:

```
163    if (twi_send(TAKTSTRASSE, 0xd3) >= 0xaa)
164    {
```

(1) Laufband der Fräse wird ausgeschaltet

```
184        twi_send(TAKTSTRASSE, 0xde);
185        byteD &= ~(0x20);
186        twi_send(TAKTSTRASSE, byteD);
```

(2) Laufband des Bohrers wird ausgeschaltet

```
189        twi_send(TAKTSTRASSE, 0xde);
190        byteD &= ~(0x10);
191        twi_send(TAKTSTRASSE, byteD);
```

(3) Motor des Bohrers wird angeschaltet

```
194        twi_send(TAKTSTRASSE, 0xce);
195        byteC |= (0x80);
196        twi_send(TAKTSTRASSE, byteC);
```

(4) Verzögerungsschleife um das Werkstück vom Bohrer bearbeiten zu lassen

```
199        for (i=1; i<45; i++)
200          _delay_ms(32);
```

(5) Motor des Bohrers wird ausgeschaltet

```
203        twi_send(TAKTSTRASSE, 0xce);
204        byteC &= ~(0x80);
205        twi_send(TAKTSTRASSE, byteC);
```

(6) Laufband des Bohrers auf Richtung *vorwärts* einstellen

```
208        twi_send(TAKTSTRASSE, 0xbe);
209        byteB |= (0x01);
210        twi_send(TAKTSTRASSE, byteB);
```

(7) Laufband des Bohrers wird angeschaltet

```
213        twi_send(TAKTSTRASSE, 0xde);
214        byteD |= (0x10);
215        twi_send(TAKTSTRASSE, byteD);
```

(8) Verzögerungsschleife um das Laufband des Bohrers zu verlassen

```
218        for (i=1; i<45; i++)
219          _delay_ms(32);
```

(9) Laufband des Bohrers wird ausgeschaltet

```
222        twi_send(TAKTSTRASSE, 0xde);
223        byteD &= ~(0x10);
224        twi_send(TAKTSTRASSE, byteD);
```

(10) Rechter Schieber auf Richtung *vorwärts* einstellen

```
227        twi_send(TAKTSTRASSE, 0xbe);
228        byteB |= (0x04);
229        twi_send(TAKTSTRASSE, byteB);
```

(11) Rechter Schieber wird angeschaltet

```
232         twi_send(TAKTSTRASSE, 0xbe);
233         byteB |= (0x08);
234         twi_send(TAKTSTRASSE, byteB);
```

(12) Betreten einer Warteschleife, bis der vordere Taster des rechten Schiebers aktiviert wird

```
237         while((twi_send(TAKTSTRASSE, 0xaa) & 0x40) != 0x00)
238         {
239         }
```

(13) Rechter Schieber auf Richtung *rückwärts* einstellen

```
242         twi_send(TAKTSTRASSE, 0xbe);
243         byteB &= ~(0x04);
244         twi_send(TAKTSTRASSE, byteB);
```

(14) Betreten einer Warteschleife, bis der hintere Taster des rechten Schiebers aktiviert wird

```
247         while((twi_send(TAKTSTRASSE, 0xaa) & 0x20) != 0x00)
248         {
249         }
```

(15) Rechter Schieber wird ausgeschaltet

```
252         twi_send(TAKTSTRASSE, 0xbe);
253         byteB &= ~(0x08);
254         twi_send(TAKTSTRASSE, byteB);
```

(16) Auslagerband wird angeschaltet

```
257         twi_send(TAKTSTRASSE, 0xbe);
258         byteB |= (0x02);
259         twi_send(TAKTSTRASSE, byteB);
260     }
```

Fototransistor hinter dem Auslagerband wird ausgelöst

Der Ablauf, der durchgeführt wird, sobald der Lichtweg des Fototransistors hinter dem Auslagerband durchbrochen wurde, sieht folgendermaßen aus:

```
263     if (twi_send(TAKTSTRASSE, 0xd4) >= 0xaa)
264     {
```

(1) Auslagerband wird angeschaltet

```
271         twi_send(TAKTSTRASSE, 0xbe);
272         byteB |= (0x02);
273         twi_send(TAKTSTRASSE, byteB);
```

(2) Verzögerungsschleife um den Fototransistor des Auslagerbands zu verlassen

```
276        for (i=1; i<45; i++)
277          _delay_ms(32);
```

(3) Auslagerband wird ausgeschaltet

```
280        twi_send(TAKTSTRASSE, 0xbe);
281        byteB &= ~(0x02);
282        twi_send(TAKTSTRASSE, byteB);
283      }
```

4 Fazit

4.1 Erreichte Ziele

Mit Fertigstellen dieser Arbeit liegt eine fertig entwickelte Platine und zugehörige, vollständige Implementation der Anforderungen vor. Erreicht wurde eine Ansteuerung einer Takt- und Verarbeitungsstraße über das TWI-Busprotokoll, eine Entwicklung einer seriellen Kommunikationsmöglichkeit eines PCs mit der vorliegenden Hardware, die Implementierung der angesprochenen Funktionen, zum Beispiel zur Analog-Digital-Wandlung, sowie die Entwicklung und Implementierung einer Bibliothek zur anspruchsvolleren 4-Bit-Ansteuerung eines vierzeiligen LCD-Display mit zwei Controllern[1]. Diese Bibliothek[2] ist nicht fest an das Projekt gebunden und kann nach Anpassung des Header-Files an jedem Pin, an jedem beliebigen Port eines jeden AT-Mega16 betrieben werden.

Weiterhin stellen wir eine Musterlösung zur Verfügung, die die Taktstraße komplett anspricht und einen Single-Processing-Durchlauf garantiert.

Eventuelle Fehler in der Kommunikation, sowie vom Benutzer über das TWI gesendete Meldungen, werden über das Display ausgegeben.

4.2 Einige Gedanken

Da in unserem Hauptstudium der praktische Einsatz von Digitalelektronik etwas zu kurz kommt, freuten wir uns natürlich eine Möglichkeit geboten zu bekommen, sich mit Mikrocontrollern, Platinen etc. zu beschäftigen. Für den persönlichen Nutzen war diese Studienarbeit eine große Bereicherung.

Dazu kommt, dass die entwickelte Platine in der Lehre eingesetzt wird. Dieser praktische Nutzen ist eine interessante Beigabe. Die Arbeit konnte in der veranschlagten Zeit gut gelöst werden, die Zusammenarbeit mit dem betreuenden Dozent Dr. Merten Joost war immer freundlich und funktionierte reibungslos. Da dieses Projekt von zwei Studenten realisiert wurde, war die Arbeit vom Umfang und der Schwierigkeit gut lösbar.

---

[1] Im Vergleich zu einer 8-Bit-Ansteuerung benötigt man eine Kommunikation über zwei Nibble.

[2] Sie befindet sich auf der beiliegenden CD und im Anhang.

4.3 Ein Ausblick

Da wir in unserer Studienarbeit eine Platine für den Einsatz in der Lehre entwickelten kam der Gedanke auf, aus den Entwicklerversionen der drei Boards, wie am Anfang geplant, eine Platine fertigen zu lassen. Die Gründe sind dafür sind ein professionelles Aussehen und kein Problem der Durchkontaktierungen bei der Benutzung mehrerer Layers[3]. Dies wurde bisher aus Kosten- und Zeitgründen noch nicht realisiert, jedoch liegt diese Lösung bereits vor. Die Platine müsste nur noch mit den benötigten Bausteinen bestückt werden.

Ein weiterer Entwicklungsanstoß wäre gewiß die Überprüfung auf Realisierbarkeit eines Multiprocessing-Systems, welches bedeutete, dass mehrere Werkstücke zeitgleich durch die Anlage fahren. Da unser Hauptaugenmerk auf der Lehre und dem Erlernen einer Buskommunikation und ersten Schritten mit einem Mikrocontroller lag, wurde wie bereits angesprochen eine Single-Processing-Lösung entwickelt. Zwar funktioniert auch diese Lösung mit mehreren Werkstücken, sobald der Master geeignet programmiert wird, jedoch kann es hier und da zu Engpässen, beziehungsweise Stauungen und demzufolge zu einem Fehlverhalten in der Taktstraße kommen.

Zusätzlich werden die beiden Bandmotoren der Bearbeitungsstationen, sowie die Motoren der Schieber über die OC-Pins des AT-Mega16 angesteuert. Dies könnte dazu genutzt werden, diese Motoren mit variabler Geschwindigkeit betreiben zu können[4].

Da diese Platine und die TWI-Kommunikation in der Lehre eingesetzt werden, passiert es häufig, dass eine Entwicklungsversion der Software die Buskommunikation behindert. In diesem Fall bleibt die Kommunikation stehen und die zu diesem Zweck eingesetzte Möglichkeit des Hardware-Resets muss beansprucht werden. Die Software hängt meist in der TWI-Routine fest und wartet in Schleifen auf Daten, die nicht gesendet werden. Eventuell könnte man eine Methodik entwickeln, die diesem Problem vorbeugt und so die Anwendungs- und Entwicklungssicherheit anhebt[5].

Zudem wäre eine nicht allzu komplizierte Weiterentwicklung denkbar, die es ermöglicht, in einer Kommunikation mehrere Daten auszutauschen. Aufgrund der Anforderung an die Lehre wurde diese, wie bereits angesprochen, von uns nicht implementiert.

[3] Der zugehörige, fertig entwickelte und geroutete Schaltplan findet sich im Anhang und auf der beiliegenden CD.

[4] vgl. PWM: Pulsweitenmodulation

[5] Eventuell über Timer

5 Literaturverzeichnis

[Atm06] ATMEL: *Datenblatt des Herstellers Atmel zum Microcontroller AT-Mega 16*, 2006. http://www.atmel.com/dyn/resources/prod_documents/doc2466.pdf.

[Fle] FLEURY, PETER. http://jump.to/fleury.

[Hee02] HEEG, TASSILO: *Datenblatt zur Ansteuerung eines vierzeiligen LCD-Displays*, 2002. http://www.pollin.de/shop/downloads/D120232S.ZIP.

[Mik] MIKROCONTROLLER.NET. http://www.mikrocontroller.net/articles/AVRDUDE#Parallelport-Programmer_an_aktuellen_PCs.

[Mül] MÜLLER, ULRICH. http://www.ulrich-mueller.de/taktstrasse.htm.

[Sch06] SCHMITT, GÜNTER: *Mikrocomputertechnik mit Controllern der Atmel AVR-RISC-Familie*. Oldenbourg Verlag München Wien, 2006.

[Wil06] WILBERT/FOGEL: *Workshop: Platinenentwicklung mit EAGLE*, 2006. http://www.uni-koblenz.de/~physik/informatik/workshop_ss2006/index.html.

Index

A
AD-Wandler **20**, **48**
Ansteuerung **69**
AT-Mega 16 .... 18, **34**

H
Hauptprogramm **66**

I
I²C .... *siehe auch* TWI

L
LCD **22**, **49**
  Anschluss 44

P
Platine 11, **41**
  Layout 12
  Ratsnest 13
  Routen 12
  Schaltnetz 12
  Schaltplan 11, **35**

S
Software 7
  AVR-Dude 9
  AVRStudio 8
  AVRTerminal 9
  Eagle 10
  Flash 7
  WinAVR 7

T
Taktstraße 3, 4, **33**
TWI **25**
  Spezialisierung **58**
  TWI-Master **64**
  TWI-Slave **59**

U
USART **18**, **47**

A Anhang

A.1 Platinen

A.1.1 Prozessorplatine

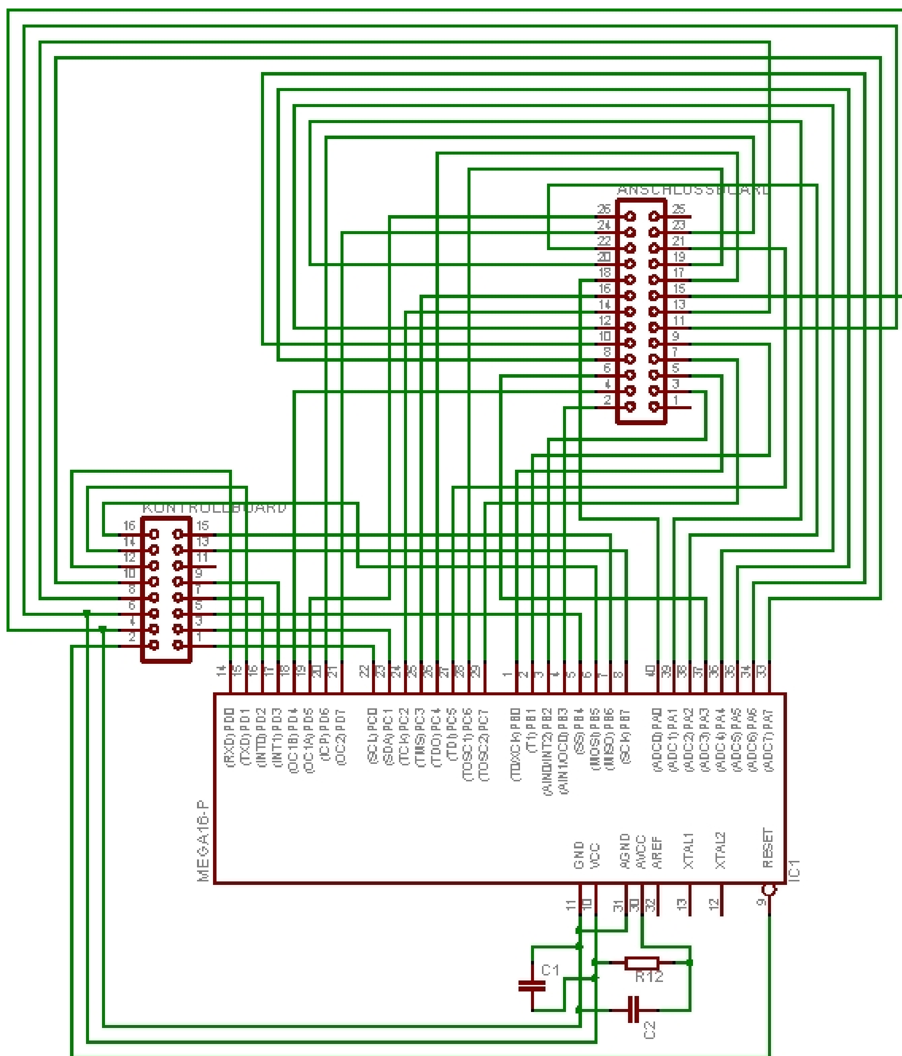


Abbildung A.1: Schaltplan der Prozessorplatine

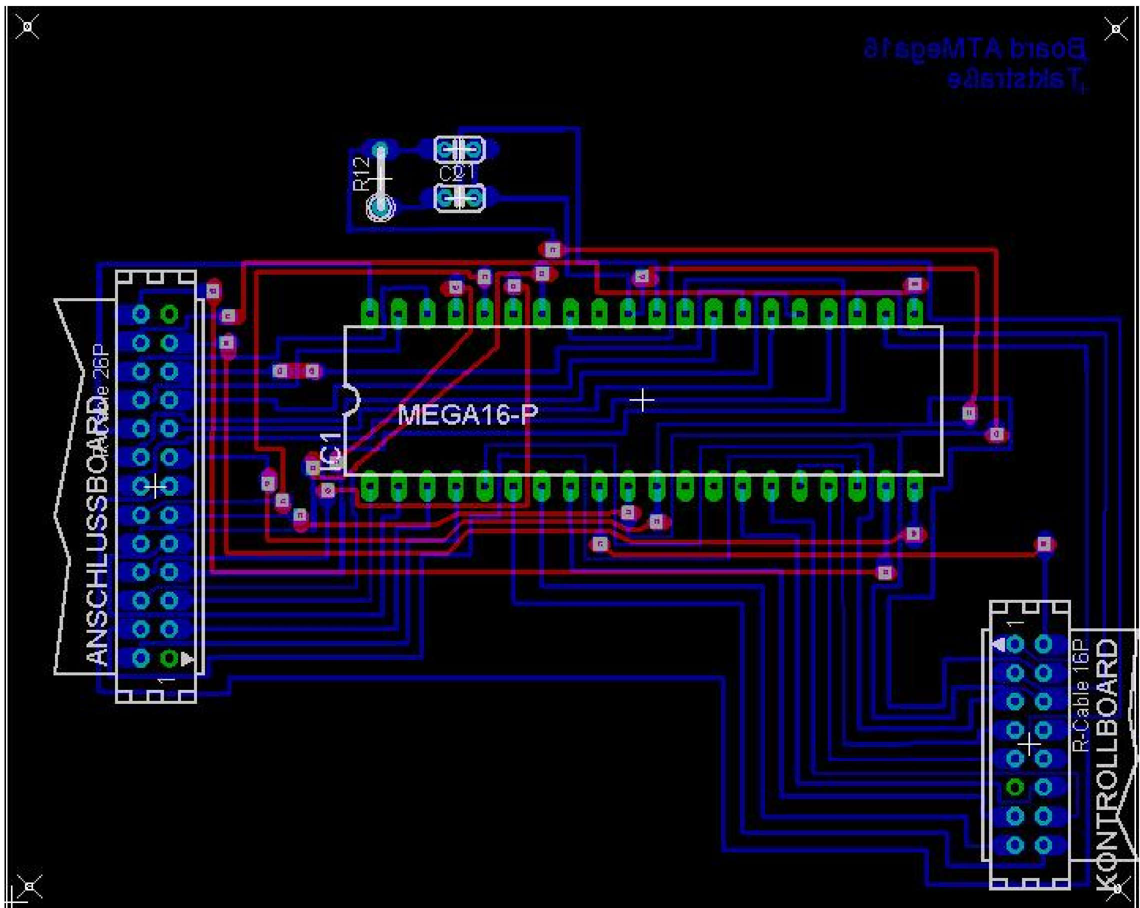


Abbildung A.2: Boardansicht der Prozessorplatine

A.1.2 Anschlussplatine

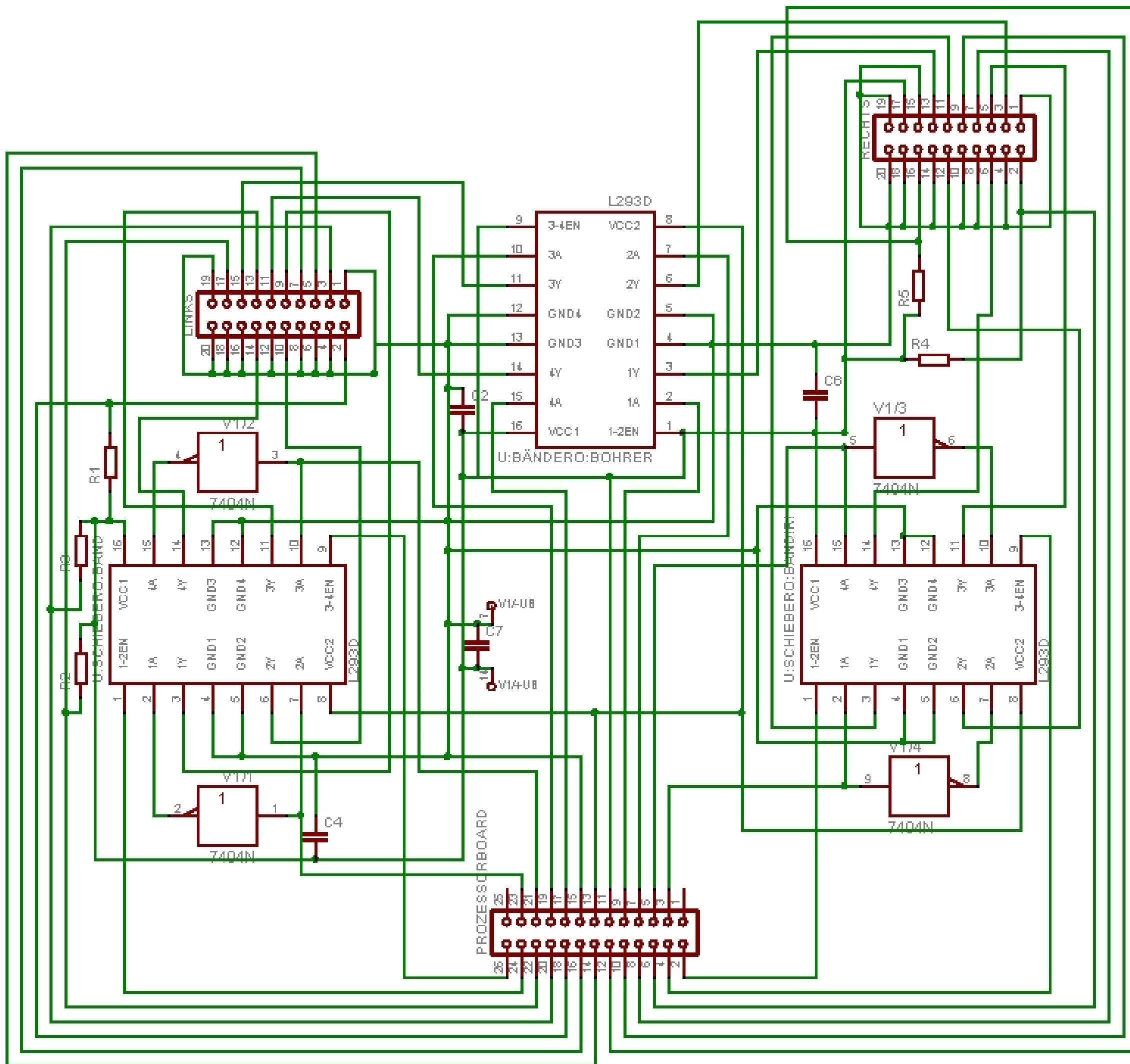


Abbildung A.3: Schaltplan der Anschlussplatine

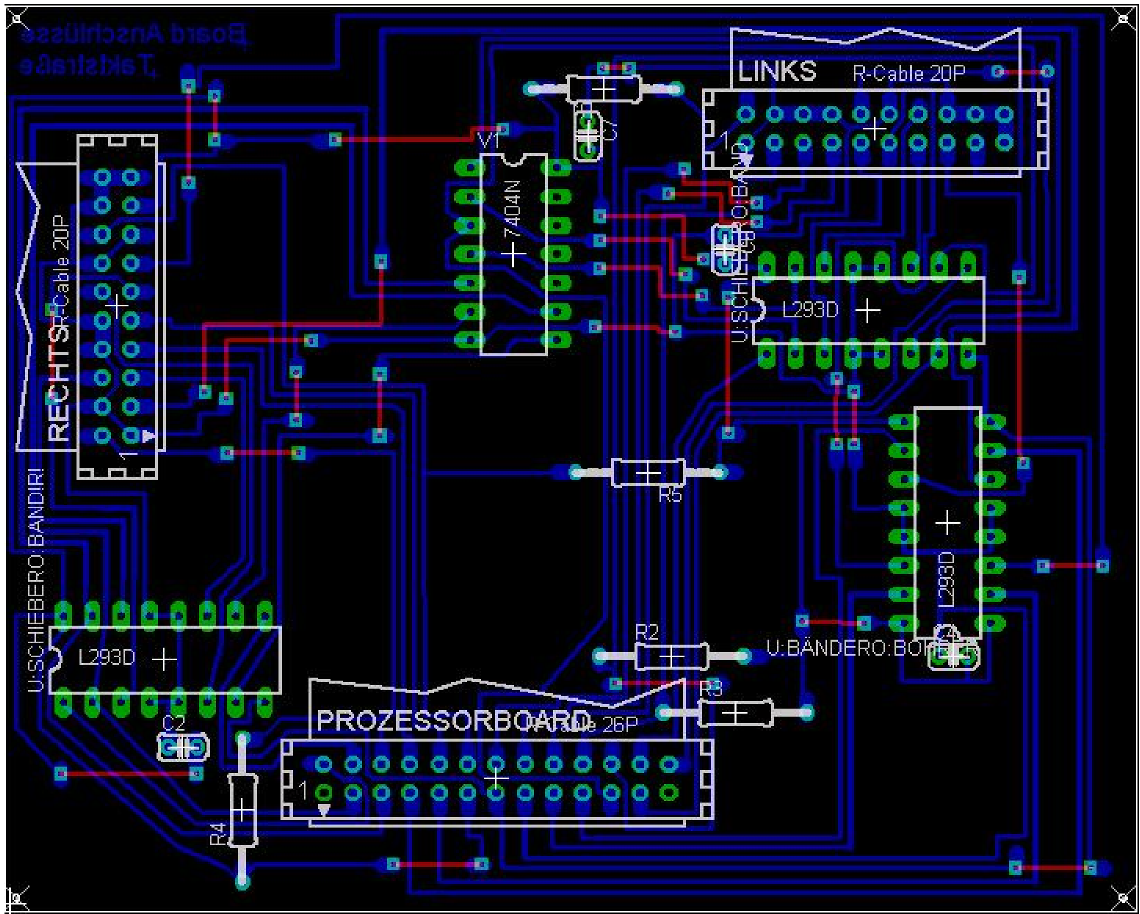


Abbildung A.4: Boardansicht der Anschlussplatine

A.1.3 Kontrollplatine

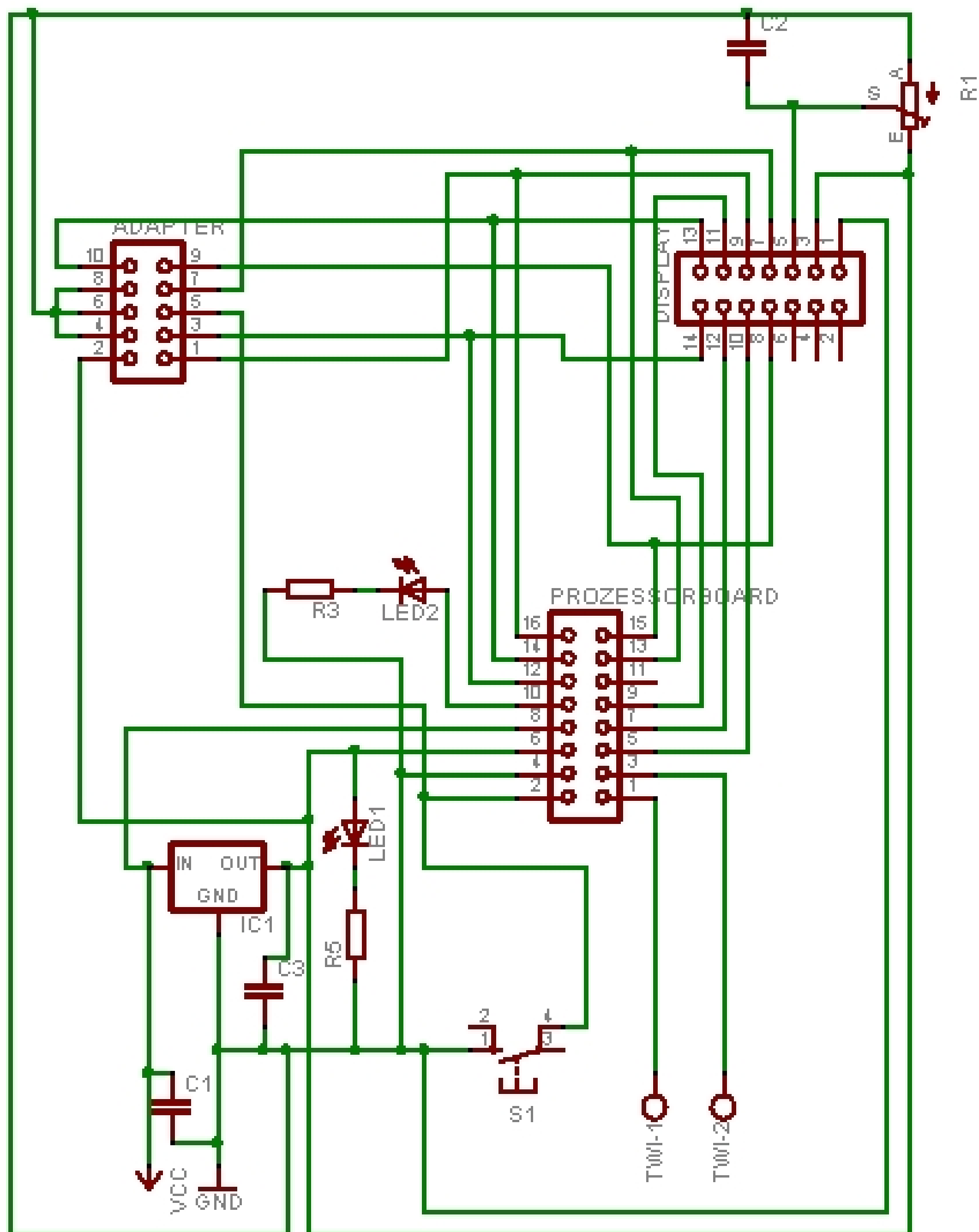


Abbildung A.5: Schaltplan der Kontrollplatine

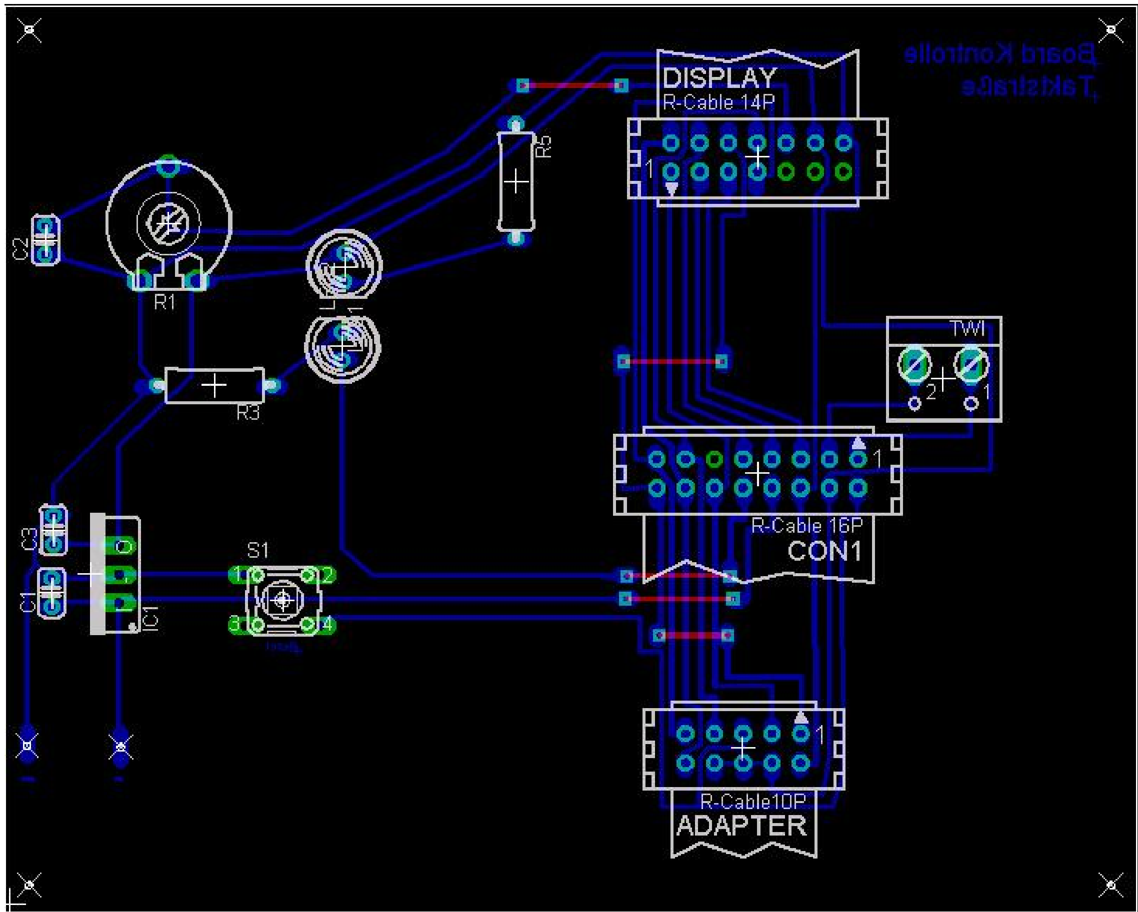


Abbildung A.6: Boardansicht der Kontrollplatine

A.1.4 Finale Ein-Platinen-Lösung

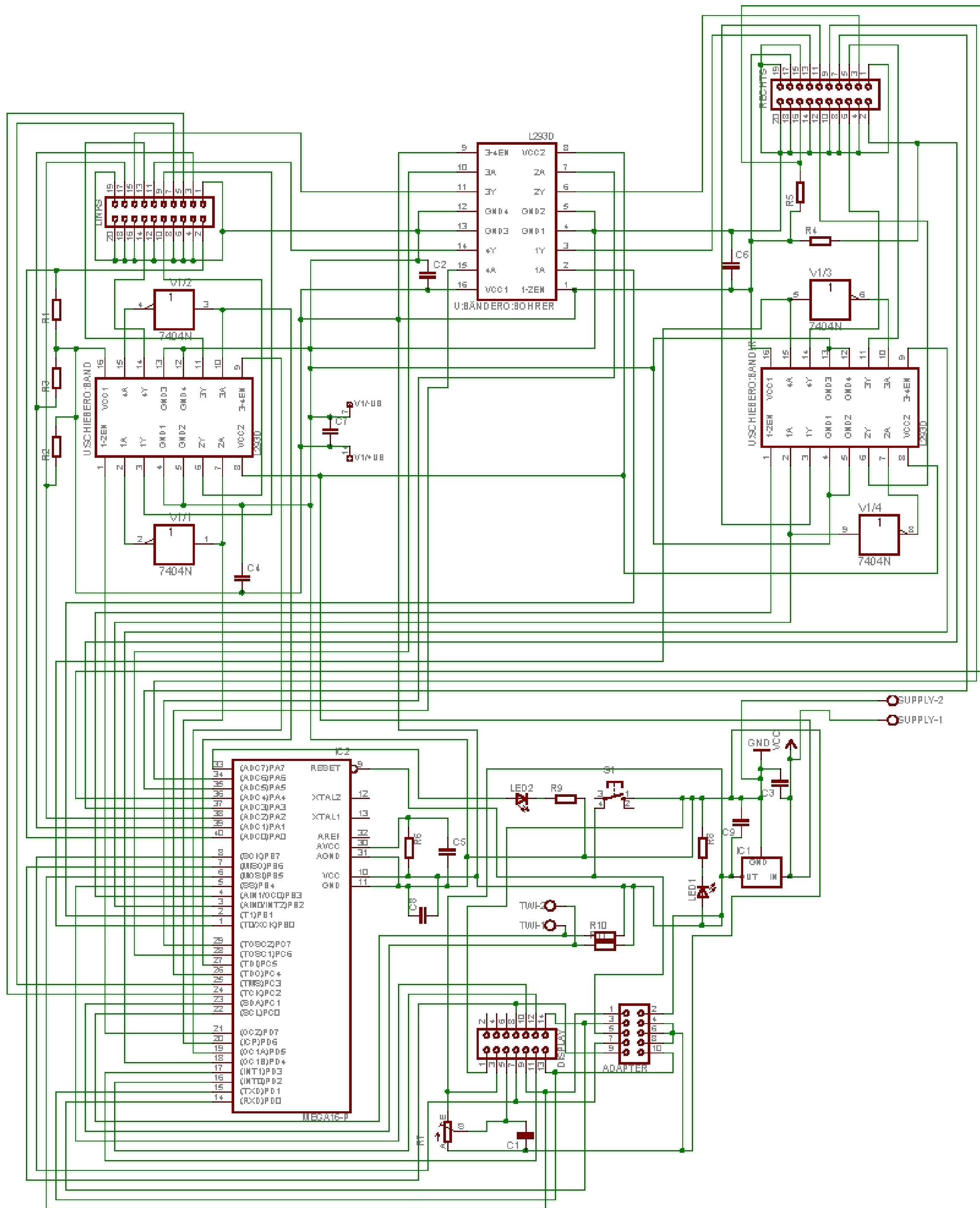


Abbildung A.7: Schaltplan der finalen Ein-Platinen-Lösung

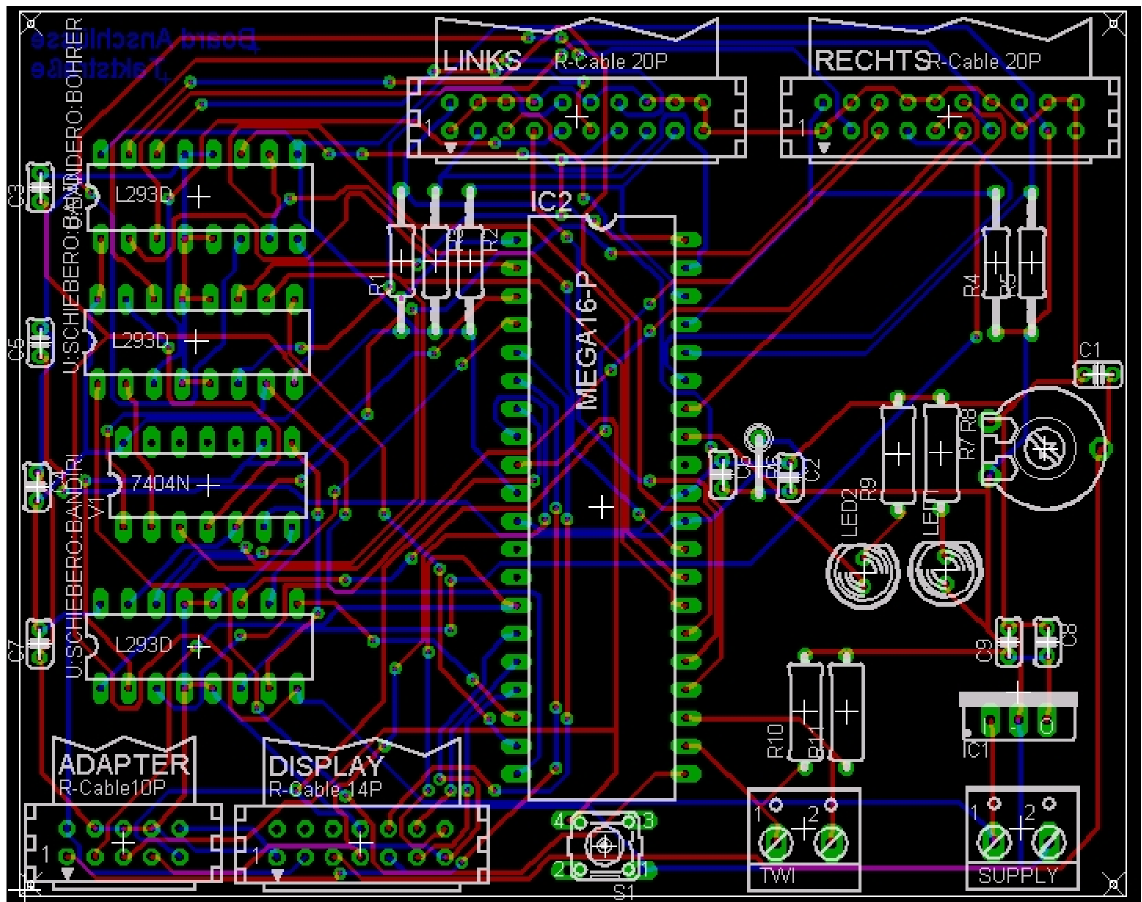


Abbildung A.8: Boardansicht der finalen Ein-Platinen-Lösung

A.2 HWP-Versuch 10

Ansteuerung einer Taktstraße mit TWI (I^2C)

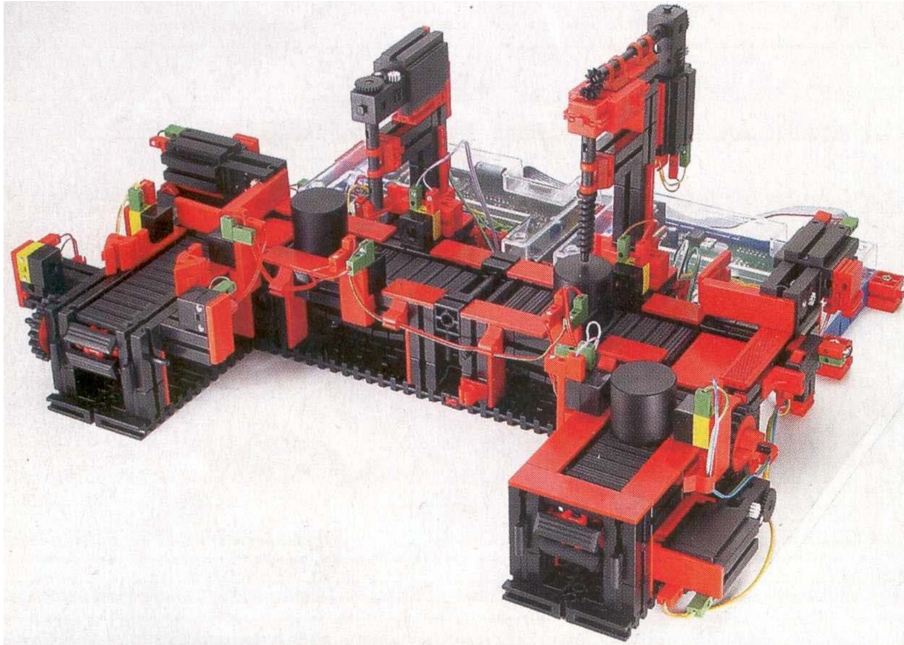


Abbildung A.9: Modell 51 664A, [Mül]

Die Taktstraße besteht aus folgenden Komponenten:

- 2 Bearbeitungsstationen
  - Fräser und Bohrer
- 4 Transportbänder (U-förmig angeordnet)
  - Zuführband
  - Band Fräser
  - Band Bohrer
  - Auslagerband
- 8 Motoren
  - 4 Bändermotoren
  - 2 Motoren der Bearbeitungsstationen
  - 2 Schiebermotoren
- 4 Taster
  - 2 Taster bei Schieber vor Bearbeitungsstation

  - 2 Taster bei Schieber nach Bearbeitungsstation
- 5 Lichtschranken (mit Fototransistor und Linsenlampe)
  - Start Zuführband
  - Ende Zuführband
  - Bearbeitung Fräser
  - Bearbeitung Bohrer
  - Ende Auslagerband

Da die Vielzahl der vorhanden Aktoren und Sensoren es nicht mehr erlaubt, die Taktstraße mit einem ECB16 Board zu betreiben, wurde ein anderes Konzept gewählt: Die Taktstraße bezitzt einen eigenen Mikrocontroller, der von aussen über eine serielle Schnittstelle (TWI) angesprochen werden kann. Der Controller der Taktstraße nimmt also über das TWI Kommandos entgegen und führt diese entsprechend aus.

Der Controller (ebenfalls ein ATMega16) ist wie folgt mit den Aktorem und Sensoren der Taktstraße verbunden:

Port A	**Beschreibung**
PA0(ADC0)	Fototransistor an der Einlegestation
PA1(ADC1)	Fototransistor am linken Schieber
PA2(ADC2)	Fototransistor an der Fräsmaschine
PA3(ADC3)	Fototransistor am Bohrer
PA4(ADC4)	Fototransistor hinter dem Auslagerband
PA5(ADC5)	Taster des Schiebers rechte Seite (hinten)
PA6(ADC6)	Taster des Schiebers rechte Seite (vorne)
PA7(ADC7)	Rote LED (einstellbare Kontrolllampe)

Port B	**Beschreibung**
PB0(XCK/T0)	Band Bohrmaschine (Richtung)
PB1(T1)	Auslagerband (an/aus)
PB2(INT2/AIN0)	Motor Schieber rechts (Richtung)
PB3(OC0/AIN1)	Motor Schieber rechts (an/aus)
PB4(SS)	LCD(Kontrolleitung RS)
PB5(MOSI)	LCD(Kontrolleitung R/W), Programmieradapter(Pin 1)
PB6(MISO)	LCD(Kontrolleitung E1), Programmieradapter(Pin 9)
PB7(SCK)	LCD(Kontrolleitung E2), Programmieradapter(Pin 7)

Port C	**Beschreibung**
PC0(SCL)	TWI-Taktleitung
PC1(SDA)	TWI-Datenleitung
PC2(TCK)	Taster des Schiebers linke Seite (vorne)
PC3(TMS)	Taster des Schiebers linke Seite (hinten)
PC4(TDO)	Zuführband (an/aus)
PC5(TDI)	Band Fräse (Richtung)

PC6(TOSC1)	Motor Fräse (an/aus)
PC7(TOSC2)	Motor Bohrer (an/aus)

Port D	**Beschreibung**
PD0(RXD)	LCD(Datenleitung D4), Programmieradapter(Pin 3)
PD1(TXD)	LCD(Datenleitung D5), Programmieradapter(Pin 10)
PD2(INT0)	LCD(Datenleitung D6)
PD3(INT1)	LCD(Datenleitung D7)
PD4(OC1B)	Band Bohrmaschine (an/aus)
PD5(OC1A)	Band Fräse (an/aus)
PD6(ICP1)	Motor des Schiebers linke Seite (Richtung)
PD7(OC2)	Motor des Schiebers linke Seite (an/aus)

TWI Protokoll

TWI - Kommunikationsspezialisierung

Der vorliegende TWI-Slave erwartet einen besonderes, für diesen Zweck optimiertes Protokoll, welches sich in einer Kommunikation auf das Austauschen von zwei Bytes beschränkt.

Nach der Aufnahme der Verbindung mit dem Senden eines START-Befehls und der daraufhin folgenden Information des Schreibmodus durch das Senden des SLA+W-Befehls, welchen der Slave mit einem ACK beantwortet, befindet sich der Master im *Master-Transmitter-Modus* und der Slave im *Slave-Receiver-Modus*. Nun wird ein Byte vom Master zum Slave gesendet, welches dieser mit einem NACK beantwortet. Damit ist der Sendevorgang beendet.

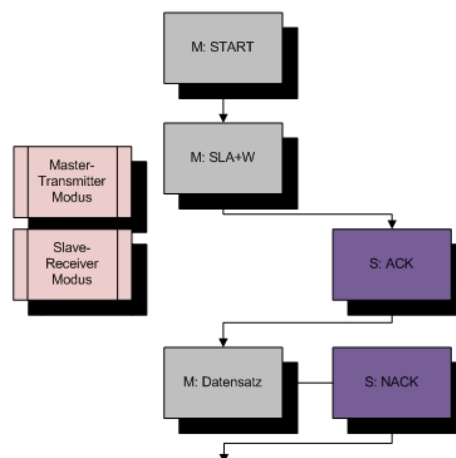


Abbildung A.10: TWI-Kommunikationsablauf: Senden

Durch einen daraufhin zu sendenden REPEATED-START-Befehl behält der Master den Bus und adressiert den selben Slave mit dem Befehl SLA+R erneut. Dieser antwortet mit einem ACK und begibt sich in den *Slave-Transmitter-Modus*, während der Master in den *Master-Receiver-Modus* wechselt. Nach dem Erhalt eines Datensatzes des Slaves, welches vom Master mit einem NACK beantwortet werden muss, da ein Byte als Antwort ausreicht, ist auch dieser Sendevorgang beendet und der Master beendet die Verbindung durch das Senden eines STOP-Befehls.

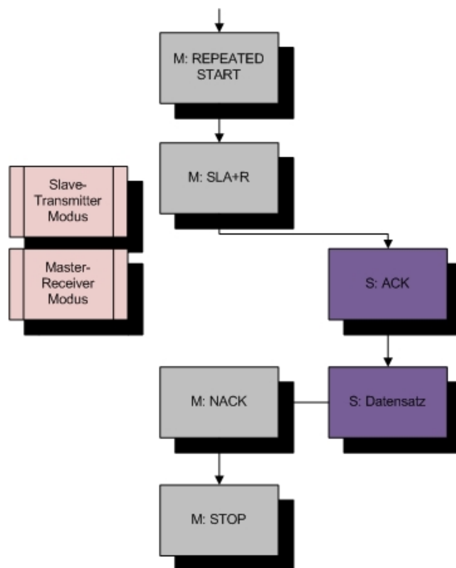


Abbildung A.11: TWI-Kommunikationsablauf: Empfangen

Die Zwei-Byte-Kommunikation über den TWI-Bus ist abgeschlossen.

Beschreibung TWI-Befehle (TWI-Slave)

Über das TWI Protokoll hat man die Möglichkeit, über geeignete Befehle die Fototransistoren auszulesen, die jeweilige Portbelegung auszulesen, sowie einzelne Ports zu beschreiben. Der Controller der Taktstraße kann zwischen 2 Modi auswählen (R/W) und befindet sich standardmäßig im Lesemodus.

Im Lesemodus erwartet der Controller eine von folgenden Anweisungen:

Testbefehl TWI Verbindung

t Senden eines Testwertes.

Portauswahl zum Wechsel in den Schreibmodus

0xbe Auswahl von Port B für den nächsten Schreibbefehl

0xce Auswahl von Port C für den nächsten Schreibbefehl

0xde Auswahl von Port D für den nächsten Schreibbefehl

- Auf den Datenbus wird je nach Portauswahl zur Kontrolle ein '0xbf', '0xcf' oder '0xdf' geschrieben.

Spezielle Portauswahl zum Beschreiben des Displays

d Auswahl des Displays für den nächsten Schreibbefehl.

- Es wird ein '0x10' zurückgegeben.

Löschen des Displays

x Löschen des Displays.

- Es wird ein '0x11' zurückgegeben.

aktuelle Portbelegung auslesen

0xaa PORTA auslesen

0xbb PORTB auslesen

0xcc PORTC auslesen

0xdd PORTD auslesen

Fototransistoren auslesen

0xd0 Fototransistor 1 (Start Zuführband) auslesen

0xd1 Fototransistor 2 (Ende Zuführband) auslesen

0xd2 Fototransistor 3 (Verarbeitungsstation Fräser) auslesen

0xd3 Fototransistor 4 (Verarbeitungsstation Bohrer) auslesen

0xd4 Fototransistor 5 (Ende Auslagerband) auslesen

Schreibmodus

Wenn vorher die Portauswahl getroffen wurde, wechselt der Controller der Taktstraße in den Schreibmodus. Der übermittelte Hexwert wird dann auf den vorher ausgewählten Port geschrieben und der Schreibmodus wird wieder verlassen. Auf den Datenbus wird je nach Portauswahl zur Kontrolle ein '0xbb', '0xcc' oder '0xdd' geschrieben.

Wurde die spezielle Portauswahl zum Beschreiben des Displays getroffen, so wird der übermittelte Wert dem Display zugewiesen und der Schreibmodus wieder verlassen. Zur Kontrolle wird ein '0x12' zurückgegeben.

Die Reaktion der Taktstraße auf ein gesendetes Byte ist zusammenfassend in folgender Grafik dargestellt:

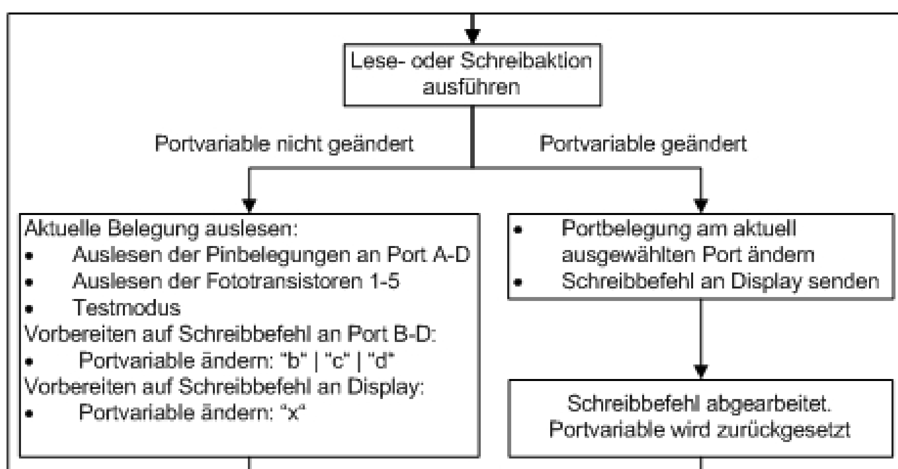


Abbildung A.12: Reaktion des Slaves auf ein Datum

Aufgabe 1: TWI Kommunikation

Implementieren Sie mit Hilfe des ECB16 einen TWI Master. Der Master soll im Master-Transmit, sowie daraufhin im Master-Receiver Modus betrieben werden. Ziel ist es, dem Controller der Taktstraße ein 't' über den TWI Bus zu senden. Die Taktstraße besitzt die Adresse 112. Eine Antwort kann beispielsweise mit dem Programm AVR-Term über die USART ausgelesen werden.

Antwort der Taktstraße: _______

Aufgabe 2: Fototransistor

Messen Sie unter Verwendung des Masters aus Versuch 1 die Werte der fünf Fototransistoren der Taktstraße. Unterbrechen Sie bei Ihren Versuchen auch den Lichtweg.

Tragen Sie hier die tiefsten Werte ein:

Fototransistor 1 _______

Fototransistor 2 _______

Fototransistor 3 _______

Fototransistor 4 _______

Fototransistor 5 _______

Tragen Sie hier die höchsten Werte ein:

Fototransistor 1 _______

Fototransistor 2 _______

Fototransistor 3 _______

Fototransistor 4 _______

Fototransistor 5 _______

Berechnen Sie nun einen Grenzwert nach folgender Rechenvorschrift:

$$Grenzwert = \frac{min(hoechsteWerte) + max(kleinsteWerte)}{2}$$

Grenzwert: _______

Aufgabe 3: Zuführband

Benutzen Sie die in Versuch 2 gemessenen Werte um das Zuführband der Taktstraße zu steuern. Liegt also ein Werkstück bei Fototransistor 1, so soll sich das Zuführband bewegen. Erreicht das Werkstück Fototransistor 2, so soll das Band wieder stehen bleiben.

Aufgabe 4: Schieber

Lassen Sie den Schieber auf der linken Seite der Taktstraße zwischen seinen beiden Bewegungstastern hin und her fahren. Sobald ein Taster schließt soll ein Richtungswechsel stattfinden.

Hinweis: Taster werden anders als Fototransistoren abgefragt (Bit-Maskierung).

Aufgabe 5: Taktstraße

Implementieren Sie einen Durchlauf mit einem Werkstück durch die komplette Taktstraße. (single processing)

A.3 Sourcecode

A.3.1 TWI-Slave (Taktstraße)

Dateien: init

- init.h
- init.c

Listing A.1: 'include\init.h'

```
#ifndef INIT_H
#define INIT_H

#include <stdio.h>
#define TAKTSTRASSE 112

/*!
\brief Initializes all relevant registers of the Twisted Wire Interface
    . The adress is taken from a define named SLAVEAD.

*/
void twi_init(void);

/*!
\brief Initializes the microcontrollers ATMega16, defines its ports as
    input or output and sets Pullups

*/
void atm16_init(void);

#endif
```

Listing A.2: 'include\init.c'

```
#include "init.h"
#include <avr/io.h>
#include <stdio.h>
#include <compat/twi.h>

void atm16_init(void)
{
  // Vereinbaren der Portrichtung
  // 0 = Eingang
  // 1 = Ausgang
  DDRA = 0x80;
  DDRB = 0xff;
  DDRC = 0xf0;
  DDRD = 0xff;

  // Pullups für Taster und TWI
```

```
17   PORTA = 0x60;
18   PORTB = 0x00;
19   PORTC = 0x0f;
20   PORTD = 0x00;
21 }
22
23
24 void twi_init(void)
25 {
26   // Initialisierung TWI-Register
27   TWCR = 0x00;
28   TWSR = 0x00;
29   PORTC |= 0x03;
30
31   // Slave-Receiver-Modus initialisieren
32   TWAR = TAKTSTRASSE;
33   TWCR = (1 << TWEN) | (1 << TWEA) | (1 << TWIE) | (1 << TWINT);
34 }
```

Dateien: adc

- adc.h
- adc.c

Listing A.3: 'include\adc.h'

```
1 #ifndef ADC_H
2 #define ADC_H
3
4 /*!
5 \brief Converts an analog value on a specified channel into a digital
      value with the help of the built-in ADC of the ATMega16.
6
7 \param char channel to be chosen for the next convertion.
8 */
9 int readADC(char channel);
10
11 #endif
```

Listing A.4: 'include\adc.c'

```
1 #include "adc.h"
2 #include <avr/io.h>
3
4 int readADC(char channel)
5 {
6   unsigned int result;
7
8   ADCSRA |= (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1);
9
```

```
10	// Kanal des Multiplexers waehlen
11	// Anlegen der internen Spannung von 2.5 V
12	ADMUX = channel;
13	ADMUX |= (1 << REFS1) | (1 << REFS0);
14
15	// Eine Wandlung START
16	ADCSRA |= (1 << ADSC);
17
18	// Auf Ergebnis warten
19	while(ADCSRA & (1 << ADSC));
20
21	result = ADC;
22
23	// ADC wieder deaktivieren
24	ADCSRA |= (0 << ADEN);
25
26	return result;
27 }
```

Dateien: lcd

- lcd.h
- lcd.c

Listing A.5: 'include\lcd.h'

```
1 #ifndef LCD_H
2 #define LCD_H
3
4 ////////////////////////
5 ////////DEFINES//////////
6 ////////////////////////
7 #ifndef F_CPU
8 #define F_CPU 8000000UL
9 #endif
10
11 //VORAUSSETZUNG: 4-BIT-MODUS
12 //KONTROLLLEITUNGEN: R/W RS E1 E2
13 //DATENLEITUNGEN: D7-D4
14 #define LCD_DATA_DDR       DDRD
15 #define LCD_DATA_PORT      PORTD
16 #define LCD_DATA_PIN       PIND
17
18 #define LCD_CTRL_DDR       DDRB
19 #define LCD_CTRL_PORT      PORTB
20 #define LCD_CTRL_PIN       PINB
21
22 #define LCD_DATA_D7_DDR    LCD_DATA_DDR
23 #define LCD_DATA_D7_PORT   LCD_DATA_PORT
24 #define LCD_DATA_D7_PIN    LCD_DATA_PIN
```

```
25 #define LCD_DATA_D7_P      3
26
27 #define LCD_DATA_D6_DDR    LCD_DATA_DDR
28 #define LCD_DATA_D6_PORT   LCD_DATA_PORT
29 #define LCD_DATA_D6_PIN    LCD_DATA_PIN
30 #define LCD_DATA_D6_P      2
31
32 #define LCD_DATA_D5_DDR    LCD_DATA_DDR
33 #define LCD_DATA_D5_PORT   LCD_DATA_PORT
34 #define LCD_DATA_D5_PIN    LCD_DATA_PIN
35 #define LCD_DATA_D5_P      1
36
37 #define LCD_DATA_D4_DDR    LCD_DATA_DDR
38 #define LCD_DATA_D4_PORT   LCD_DATA_PORT
39 #define LCD_DATA_D4_PIN    LCD_DATA_PIN
40 #define LCD_DATA_D4_P      0
41
42 #define LCD_CTRL_RW_DDR    LCD_CTRL_DDR
43 #define LCD_CTRL_RW_PORT   LCD_CTRL_PORT
44 #define LCD_CTRL_RW_PIN    LCD_CTRL_PIN
45 #define LCD_CTRL_RW_P      5
46
47 #define LCD_CTRL_RS_DDR    LCD_CTRL_DDR
48 #define LCD_CTRL_RS_PORT   LCD_CTRL_PORT
49 #define LCD_CTRL_RS_PIN    LCD_CTRL_PIN
50 #define LCD_CTRL_RS_P      4
51
52 #define LCD_CTRL_E1_DDR    LCD_CTRL_DDR
53 #define LCD_CTRL_E1_PORT   LCD_CTRL_PORT
54 #define LCD_CTRL_E1_PIN    LCD_CTRL_PIN
55 #define LCD_CTRL_E1_P      6
56
57 #define LCD_CTRL_E2_DDR    LCD_CTRL_DDR
58 #define LCD_CTRL_E2_PORT   LCD_CTRL_PORT
59 #define LCD_CTRL_E2_PIN    LCD_CTRL_PIN
60 #define LCD_CTRL_E2_P      7
61
62
63 #define LCD_CLR_DISPLAY             0x01
64 #define LCD_CUR_HOME                0x02
65 #define LCD_ENTRY_MODE_SET_INC_V    0x07
66 #define LCD_ENTRY_MODE_SET_INC_C    0x06
67 #define LCD_ENTRY_MODE_SET_DEC_V    0x05
68 #define LCD_ENTRY_MODE_SET_DEC_C    0x04
69 #define LCD_DISPLAY_ON_U_B          0x0F
70 #define LCD_DISPLAY_ON_U            0x0E
71 #define LCD_DISPLAY_ON_B            0x0D
72 #define LCD_DISPLAY_ON              0x0C
73 #define LCD_DISPLAY_OFF             0x08
74 //NUR 4-BIT-ANSTEUERUNG!!!
75 #define LCD_SYS_SET_4BIT_24R_57_IO  0x28
76
```

```
77 /////////////////////////
78 ////////FUNKTIONEN////////
79 /////////////////////////
80
81 /*!
82 \brief Initializes one of the two given controllers in the LCD-Display.
83
84 \param int controller specifies which controller in the display is to
       be initialized.
85 */
86 void lcd_init(int controller);
87
88 /*!
89 \brief Clears the part of the display the given controller is connected
       to.
90
91 \param int controller specifies the part of the display which should be
       cleared.
92 */
93 void lcd_cls(int controller);
94
95 /*!
96 \brief Prints a string on the Part of the Display the given Controller
       is connected to.
97
98 \param char* c is the string that should be printed.
99 \param int controller specifies the part of the display where the
       string is expected to be printed.
100 */
101 void lcd_prints(char *c, int controller);
102
103
104
105 //Sendet(R/W=0) ein Kommando(RS=0) oder Daten(RS=1) an das LCD
106 /*!
107 \brief Sends data or a commando to the given controller.
108
109 \param int rs regulates the way of writing the data. RS=0 means
       commando; RS=1 means Data.
110 \param char c is the data (char) which is to be transmitted.
111 \param int controller specifies the controller to which the data will
       be send.
112 */
113 void lcd_send(int rs, char c, int controller);
114
115 //Liest(R/W=1) BF/AdressCounter(RS=0) oder Daten aus dem Register(RS=1)
116 /*!
117 \brief Gets data or the Busyflag/AdressCounter from the given
       controller.
118
119 \param int rs regulates the way of reading the data. RS=0 means BF/AC;
       RS=1 means Data.
```

```
120 \param int controller specifies the controller from which the data will
      be read.
121 */
122 char lcd_get(int rs, int controller);
123
124
125 //Fragt den Status des Busyflags ab und wartet bis dieses gecleared ist
126 /*!
127 \brief reads the busyflag and loops until it is cleared.
128
129 \param int controller specifies the controller whose busyflag will be
     checked.
130 */
131 void lcd_loopBusy(int controller);
132
133 //Übernimmt anliegende Daten in das LCD, der Controller-Pin wird auf
     High
134 //und anschließen auf Low gezogen. Das LCD übernimmt Daten bei
     fallender Flanke.
135 /*!
136 \brief applies the data given at the LCD-Wires. The Data is taken when
     the signal falls from 1 to 0.
137
138 \param int controller specifies the controller which is going to work
     with the data.
139 */
140 void lcd_apply(int controller);
141
142
143 #endif
```

Listing A.6: 'include\lcd.c'

```
1 /*
2 Bibliothek zur Ansteuerung eines LCD-Displays
3 mit 2 Controllern (Wintek-Pollin)
4
5 init(0); initialisiert beide controller
6 init(1); oberer Controller
7 init(2); unterer Controller
8
9 prints(c, controller); druckt einen String (inkl. \n) auf den
     angegebenen Controller, auch auf beide
10 */
11
12 #include <avr/io.h>
13 #include <string.h>
14 #include "lcd.h" //da F_CPU vor include delay.h bekannt sein muss
15 #include <util/delay.h>
16
17 #define e1_set()  LCD_CTRL_E1_PORT |= (1<<LCD_CTRL_E1_P)
18 #define e1_clr()  LCD_CTRL_E1_PORT &= ~(1<<LCD_CTRL_E1_P)
19 #define e2_set()  LCD_CTRL_E2_PORT |= (1<<LCD_CTRL_E2_P)
```

```
20 #define e2_clr()  LCD_CTRL_E2_PORT &= ~(1<<LCD_CTRL_E2_P)
21 #define rs_set()  LCD_CTRL_RS_PORT |= (1<<LCD_CTRL_RS_P)
22 #define rs_clr()  LCD_CTRL_RS_PORT &= ~(1<<LCD_CTRL_RS_P)
23 #define rw_set()  LCD_CTRL_RW_PORT |= (1<<LCD_CTRL_RW_P)
24 #define rw_clr()  LCD_CTRL_RW_PORT &= ~(1<<LCD_CTRL_RW_P)
25
26 //Delay-Funktion (ns). Hängt ab von F_CPU -> 768 us / F_CPU in MHz ist
       Maximum eines Delays
27 //->daher in Schleife (96 wäre bei 8MHZ das Maximum)
28 void delay_ns(unsigned int ns)
29 {
30   for (unsigned int i=0;i<ns;i++)
31     _delay_us(1);
32 }
33
34 //Delay-Funktion (ms). Hängt ab von F_CPU -> 262.14 ms / F_CPU in MHz
       ist Maximum eines Delays
35 //->daher in Schleife (32,7675 wäre bei 8MHZ das Maximum)
36 void delay_ms(unsigned int ms)
37 {
38   for (unsigned int i=0;i<ms;i++)
39     _delay_ms(1);
40 }
41
42 void lcd_apply(int controller)
43 {
44   //Controller auswählen, Pegel hoch, Pegel runter
45   if (controller==1)
46   {
47     e1_set();
48     e1_clr();
49   }
50   else
51     if (controller==2)
52     {
53       e2_set();
54       e2_clr();
55     }
56     else
57       if (controller==0)
58       {
59         e1_set();
60         e2_set();
61         e1_clr();
62         e2_clr();
63       }
64 }
65
66
67 void lcd_cls(int controller)
68 {
69   lcd_send(0, LCD_CLR_DISPLAY, controller); //Display clear
```

```
70   lcd_send(0, LCD_CUR_HOME, controller); //Cursor home DD-Ram Adresse 0
71 }
72
73 void lcd_prints(char *s, int controller)
74 {
75   //For-Schleife für jeden char einzeln
76   int len = strlen(s);
77   for (int i = 0; i<len;i++)
78   {
79     lcd_send(1, s[i], controller);
80   }
81 }
82
83 //Sendet ein Kommando (RS=0) oder Daten (RS=1) an das LCD
84 void lcd_send(int rs, char c, int controller)
85 {
86   if (controller==0)
87   {
88     lcd_loopBusy(1);
89     lcd_loopBusy(2);
90   }
91   else
92     if ((controller==1)|(controller==2))
93       lcd_loopBusy(controller);
94
95   rw_clr(); //Schreibe-Modus
96   if (rs==0) //Kommando senden
97     rs_clr();//Kontrollregister
98   else if (rs==1) //Daten senden
99   {
100    rs_set(); //Datenregister
101    if (c=='\n')
102    {
103      //New Line
104      lcd_send(0,(0x40|0x80),controller); //DD-Adresse 64 (Zeile2) (und
           D7 muss an sein) ->set_DDRam_Adress
105      return;
106    }
107  }
108
109  //Port auf Ausgang stellen (Sicher is Sicher ;))
110  LCD_DATA_D7_DDR |= (1<<LCD_DATA_D7_P);
111  LCD_DATA_D6_DDR |= (1<<LCD_DATA_D6_P);
112  LCD_DATA_D5_DDR |= (1<<LCD_DATA_D5_P);
113  LCD_DATA_D4_DDR |= (1<<LCD_DATA_D4_P);
114
115  //Leitungen auf 0 ziehen
116  LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
117  LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
118  LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
119  LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
120
```

```
121   //Datenzuweisen
122   //Zuerst die oberen 4 Bit, danach die unteren
123   //Obere 4
124
125   if ((0x80 & c)==0x80) LCD_DATA_D7_PORT |= (1<<LCD_DATA_D7_P);
126   if ((0x40 & c)==0x40) LCD_DATA_D6_PORT |= (1<<LCD_DATA_D6_P);
127   if ((0x20 & c)==0x20) LCD_DATA_D5_PORT |= (1<<LCD_DATA_D5_P);
128   if ((0x10 & c)==0x10) LCD_DATA_D4_PORT |= (1<<LCD_DATA_D4_P);
129
130   if (controller==1)
131   {
132     lcd_apply(1);
133   }
134   else
135     if (controller==2)
136     {
137      lcd_apply(2);
138     }
139     else
140       if (controller==0)
141       {
142         lcd_apply(0);
143       }
144
145   //Leitungen auf 0 ziehen
146   LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
147   LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
148   LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
149   LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
150
151   //Datenzuweisen
152   //Untere 4
153   if ((0x08 & c)==0x08) LCD_DATA_D7_PORT |= (1<<LCD_DATA_D7_P);
154   if ((0x04 & c)==0x04) LCD_DATA_D6_PORT |= (1<<LCD_DATA_D6_P);
155   if ((0x02 & c)==0x02) LCD_DATA_D5_PORT |= (1<<LCD_DATA_D5_P);
156   if ((0x01 & c)==0x01) LCD_DATA_D4_PORT |= (1<<LCD_DATA_D4_P);
157
158   if (controller==1)
159   {
160     lcd_apply(1);
161   }
162   else
163     if (controller==2)
164     {
165       lcd_apply(2);
166     }
167     else
168       if (controller==0)
169       {
170         lcd_apply(0);
171       }
172
```

```
173   //Leitungen auf 0 ziehen
174   LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
175   LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
176   LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
177   LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
178 
179 }
180 
181 char lcd_get(int rs, int controller)
182 {
183   char tmp=0x00; //Variable zum Zwischenspeichern des Wertes, den das
        Display zurückgibt
184 
185   rw_set(); //(RW auf 1) -> Lesemodus
186 
187   if (rs==0) //BF/AC lesen
188     rs_clr();//Kontrolle
189   else if (rs==1) //Daten lesen
190     rs_set(); //Daten
191 
192   //Lesemodus für die jeweiligen Pins an Ports
193   LCD_DATA_D7_DDR &= ~(1<<LCD_DATA_D7_P);
194   LCD_DATA_D6_DDR &= ~(1<<LCD_DATA_D6_P);
195   LCD_DATA_D5_DDR &= ~(1<<LCD_DATA_D5_P);
196   LCD_DATA_D4_DDR &= ~(1<<LCD_DATA_D4_P);
197 
198   //Leitungen auf 0 ziehen (Zur Sicherheit)
199   LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
200   LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
201   LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
202   LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
203 
204 
205   //1. Nibble
206   //Enable-Leitung auf High-Pegel setzen, da nur dann das Display Daten
         zurückgibt
207   if (controller==1)
208     e1_set();
209   else if (controller==2)
210     e2_set();
211 
212   if ((LCD_DATA_D7_PIN & (1<<LCD_DATA_D7_P))!=0x00) tmp |= 0x80;
213   if ((LCD_DATA_D6_PIN & (1<<LCD_DATA_D6_P))!=0x00) tmp |= 0x40;
214   if ((LCD_DATA_D5_PIN & (1<<LCD_DATA_D5_P))!=0x00) tmp |= 0x20;
215   if ((LCD_DATA_D4_PIN & (1<<LCD_DATA_D4_P))!=0x00) tmp |= 0x10;
216 
217   //Enable-Leitung auf Low-Pegel setzen, um zweites Nibble abzuholen
218   if (controller==1)
219     e1_clr();
220   else if (controller==2)
221     e2_clr();
222 
```

```
223   //2. Nibble
224   //Enable-Leitung auf High-Pegel setzen
225   if (controller==1)
226     e1_set();
227   else if (controller==2)
228     e2_set();
229
230   if ((LCD_DATA_D7_PIN & (1<<LCD_DATA_D7_P))!=0x00) tmp |= 0x08;
231   if ((LCD_DATA_D6_PIN & (1<<LCD_DATA_D6_P))!=0x00) tmp |= 0x04;
232   if ((LCD_DATA_D5_PIN & (1<<LCD_DATA_D5_P))!=0x00) tmp |= 0x02;
233   if ((LCD_DATA_D4_PIN & (1<<LCD_DATA_D4_P))!=0x00) tmp |= 0x01;
234
235
236   //Enable-Leitung auf Low-Pegel setzen
237   if (controller==1)
238     e1_clr();
239   else if (controller==2)
240     e2_clr();
241
242   //Port auf Ausgang stellen
243   LCD_DATA_D7_DDR |= (1<<LCD_DATA_D7_P);
244   LCD_DATA_D6_DDR |= (1<<LCD_DATA_D6_P);
245   LCD_DATA_D5_DDR |= (1<<LCD_DATA_D5_P);
246   LCD_DATA_D4_DDR |= (1<<LCD_DATA_D4_P);
247
248   //Leitungen auf 0 ziehen
249   LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
250   LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
251   LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
252   LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
253
254   return tmp;
255 }
256
257 void lcd_loopBusy(int controller)
258 {
259   while(1)
260   {
261     if ((lcd_get(0, controller) & 0x80)==0x00)
262       break;
263   }
264 }
265
266
267 void lcd_init(int controller)
268 {
269   //ACHTUNG: BUSYFLAG KANN AM ANFANG NOCH NICHT ABGEFRAGT WERDEN,
270   //DAHER EINFACH ENTSPRECHEND LANGE WARTEN -> DELAY-FUNKTION
271
272   //Port (Kontrolle) auf Ausgang
273   LCD_CTRL_RW_DDR |= (1<<LCD_CTRL_RW_P);
274   LCD_CTRL_RS_DDR |= (1<<LCD_CTRL_RS_P);
```

```
275   LCD_CTRL_E1_DDR |= (1<<LCD_CTRL_E1_P);
276   LCD_CTRL_E2_DDR |= (1<<LCD_CTRL_E2_P);
277   //Port (Daten) auf Ausgang
278   LCD_DATA_D7_DDR |= (1<<LCD_DATA_D7_P);
279   LCD_DATA_D6_DDR |= (1<<LCD_DATA_D6_P);
280   LCD_DATA_D5_DDR |= (1<<LCD_DATA_D5_P);
281   LCD_DATA_D4_DDR |= (1<<LCD_DATA_D4_P);
282
283   //Leitungen auf 0 ziehen
284   //Kontrolle
285   LCD_CTRL_RW_PORT &= ~(1<<LCD_CTRL_RW_P);
286   LCD_CTRL_RS_PORT &= ~(1<<LCD_CTRL_RS_P);
287   LCD_CTRL_E1_PORT &= ~(1<<LCD_CTRL_E1_P);
288   LCD_CTRL_E2_PORT &= ~(1<<LCD_CTRL_E2_P);
289   //Daten
290   LCD_DATA_D7_PORT &= ~(1<<LCD_DATA_D7_P);
291   LCD_DATA_D6_PORT &= ~(1<<LCD_DATA_D6_P);
292   LCD_DATA_D5_PORT &= ~(1<<LCD_DATA_D5_P);
293   LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
294
295   //Init-Sequenz
296   delay_ms(15);
297   //3h ins Steuerregister             #1
298   LCD_DATA_D5_PORT |= (1<<LCD_DATA_D5_P);
299   LCD_DATA_D4_PORT |= (1<<LCD_DATA_D4_P);
300   lcd_apply(controller);
301   delay_ms(5);
302   //nochmal 3h ins Steuerregister       #2
303   lcd_apply(controller);
304   delay_ns(100);
305   //ein drittes Mal 3h ins Steuerregister   #3
306   lcd_apply(controller);
307
308   //4-Bit Datenbus
309   //2h schreiben
310   LCD_DATA_D4_PORT &= ~(1<<LCD_DATA_D4_P);
311   lcd_apply(controller);
312
313   //AB HIER WERDEN DIE BYTES IN 4-BIT-BLÖCKEN VERARBEITET!!!
314   //BUSY-FLAG KANN AB JETZT ABGEFRAGT WERDEN!!!
315
316   //System set
317   lcd_send(0, LCD_SYS_SET_4BIT_24R_57_IO, controller);
318   //Display on
319   lcd_send(0, LCD_DISPLAY_ON, controller);
320   //Clear Display
321   lcd_send(0, LCD_CLR_DISPLAY, controller);
322   //Cursor home
323   lcd_send(0, LCD_CUR_HOME, controller);
324   //Entrymode set
325   lcd_send(0, LCD_ENTRY_MODE_SET_INC_C, controller);
326 }
```

Datei: main

- main.c

Listing A.7: 'main.c'

```
#include <stdio.h>
#include <stdlib.h>
#include <avr/io.h>
#include <compat/twi.h>
#include <avr/interrupt.h>
#include <util/delay.h>

#include "include/init.h"
#include "include/adc.h"
#include "include/lcd.h"

unsigned char recbuf = 0x00;
unsigned char port = 'r';

// Variable für Übertragungsmodus
short transmode = 0;

/*********************************
 *****Interrupt Service Routine*****
 *********************************/
ISR(TWI_vect)
{
  // Wartet bis selbst adressiert (SLA+W wurde gesendet)
  //   -> TWINT-Flag wird gesetzt
  loop_until_bit_is_set(TWCR, TWINT);

  // Status auswerten
  // TW_SR_SLA_ACK bedeutet: SLA+W wurde empfangen, ACK wurde gesendet
  if (TW_STATUS == TW_SR_SLA_ACK)
  {
    // Byte wird empfangen und ein NACK zurückgesendet
    TWCR = (1 << TWINT) | (1 << TWEN);
    loop_until_bit_is_set(TWCR, TWINT);
  }
  else
  {
    // Kein Slave-Receiver-Modus
    // Setze Slave in Ausgangslage (empfangsbereit)
    lcd_cls(2);
    lcd_prints("FEHLER:\nKEIN SR-MODUS",2);
    twi_init();
    PORTA |= 0x80;
    return;
  }

  /****************************************
   ****Statusabfrage Slave im Reciever Modus****
```

```
48     *******************************************/
49    switch(TW_STATUS)
50    {
51      // Daten wurden empfangen, NACK wurde gesendet
52      case TW_SR_DATA_NACK:
53        recbuf = TWDR;
54        // Setze slave in not-adressed-slave-mode
55        TWCR = (1 << TWINT) | (1 << TWEA) | (1 << TWEN);
56        transmode = 1;
57        break;
58      default:
59        // Kein Byte empfangen
60        // Setze Slave in Ausgangslage (empfangsbereit)
61        lcd_cls(2);
62        lcd_prints("FEHLER:\nKEIN DATA+NACK",2);
63
64        twi_init();
65        PORTA |= 0x80;
66        return;
67    }
68
69    // Wartet bis selbst adressiert (SLA+R wurde gesendet)
70    //   -> TWINT-Flag wird gesetzt
71    loop_until_bit_is_set(TWCR, TWINT);
72
73    /***********************************************
74     ****Statusabfrage Slave im Transmitter Modus****
75     ***********************************************/
76    while(transmode == 1)
77    {
78      switch(TW_STATUS)
79      {
80        case TW_ST_SLA_ACK:
81          // Im Status TW_ST_SLA_ACK: SLA+R empfangen+ ACK zurückgesendet
82          // Lesemodus
83          if (port == 'r')
84          {
85            // Lesemodus oder Portauswahlmodus
86            switch (recbuf)
87            {
88              // Testfall
89              case 't':
90                TWDR = 0x00;
91                break;
92              // Displayausgabe vorbereiten
93              case 'd':
94                port = 'x';
95                TWDR = 0x10;
96                break;
97              // Display löschen
98              case 'x':
99                lcd_cls(2);
```

```
100               TWDR = 0x11;
101               break;
102             // Wenn Port angegeben wurde Port auswählen
103             case 0xbe:
104               port = 'b';
105               TWDR = 0xbf;
106               break;
107             case 0xce:
108               port = 'c';
109               TWDR = 0xcf;
110               break;
111             case 0xde:
112               port = 'd';
113               TWDR = 0xdf;
114               break;
115             // Pins auslesen
116             case 0xaa:
117               DDRA = 0x00;
118               TWDR = PINA;
119               DDRA = 0x80;
120               break;
121             case 0xbb:
122               DDRB = 0x00;
123               TWDR = PINB;
124               DDRB = 0xff;
125               break;
126             case 0xcc:
127               DDRC = 0x00;
128               TWDR = PINC;
129               DDRC = 0xf0;
130               break;
131             case 0xdd:
132               DDRD = 0x00;
133               TWDR = PIND;
134               DDRD = 0xff;
135               break;
136             // Fototransistoren abfragen
137             case 0xd0:
138               TWDR = (char)(readADC(0x00) >> 2);
139               break;
140             case 0xd1:
141               TWDR = (char)(readADC(0x01) >> 2);
142               break;
143             case 0xd2:
144               TWDR = (char)(readADC(0x02) >> 2);
145               break;
146             case 0xd3:
147               TWDR = (char)(readADC(0x03) >> 2);
148               break;
149             case 0xd4:
150               TWDR = (char)(readADC(0x04) >> 2);
151               break;
```

```
152             default:
153               //Unbekannter KontrollCode!
154               lcd_cls(2);
155               lcd_prints("FEHLER:\nUNBEKANNTER KONTROLLCODE",2);
156               TWDR = recbuf;
157               twi_init();
158               PORTA |= 0x80;
159               return;
160           }
161         }
162         else
163         {
164           switch (port)
165           {
166             case 'b':
167               PORTB = recbuf;
168               TWDR = 0xbb;
169               port = 'r';
170               break;
171             case 'c':
172               PORTC = recbuf;
173               TWDR = 0xcc;
174               port = 'r';
175               break;
176             case 'd':
177               PORTD = recbuf;
178               TWDR = 0xdd;
179               port = 'r';
180               break;
181             case 'x':
182               lcd_send(1, recbuf, 2);
183               TWDR = 0x12;
184               port = 'r';
185               break;
186           }
187         }
188
189         // In TWDR angelegte Daten werden übertragen, NACK wird
                empfangen
190         TWCR = (1 << TWEN) | (1 << TWINT);
191         loop_until_bit_is_set(TWCR, TWINT);
192         break;
193       case TW_ST_DATA_NACK:
194         // Status TW_ST_DATA_NACK: Data übertragen + NACK erhalten
195         transmode = 0;
196         // TWI in Ausgangszustand und vom Bus
197         TWCR = (1 << TWEA) | (1 << TWEN) | (1 << TWINT) | (1 << TWIE);
198         break;
199       case TW_NO_INFO:
200         lcd_cls(2);
201         lcd_prints("FEHLER:\nNO INFO",2);
202
```

```
203         // ST: Fehler. No Info
204         transmode = 0;
205         // TWI in Ausgangszustand (reset)
206         twi_init();
207         PORTA |= 0x80;
208         return;
209       default:
210         // Sonstige Fehlercodes (TW_STATUS)
211         lcd_cls(2);
212         lcd_prints("FEHLER:\nFALSCHER STATUS",2);
213 
214         transmode = 0;
215         // TWI in Ausgangszustand (reset)
216         twi_init();
217         PORTA |= 0x80;
218         return;
219     }
220   }
221 }
222 
223 int main(void)
224 {
225   // Globales Interrupt-Flag
226   sei();
227 
228   // Initialisierungen
229   atm16_init();
230   twi_init();
231 
232   lcd_init(0);
233   lcd_prints("XXXXX Taktstrasse TWI XXXXX\nXXXXX Ansteuerung TWI XXXXX"
        , 1);
234 
235   while(1)
236   {
237   }
238 
239   cli();
240   return 0;
241 }
```

A.3.2 TWI-Master (HWP-Board)

Datei: main

- main.c

Listing A.8: 'main.c'

```
1 #include <avr/io.h>
```

```
2 #include <stdio.h>
3 #include <compat/twi.h>
4 #include <string.h>
5
6 #define F_CPU 8000000UL
7 #include <util/delay.h>
8
9 #define TAKTSTRASSE 112
10
11 // Defines für USART
12 #define CLOCK 8000000UL
13 #define BAUD 9600UL
14
15 unsigned char databuf;
16 unsigned char byteB;
17 unsigned char byteC;
18 unsigned char byteD;
19
20 static int usart_write(char x, FILE *stream);
21 static FILE mystdout = FDEV_SETUP_STREAM(usart_write, NULL,
      _FDEV_SETUP_WRITE);
22
23 void twi_init(void)
24 {
25   // Initialisieren des ATMega16 zur
26   // Verwendung des TWI inkl. interner
27   // Pullup-Widerstände
28   DDRC = 0x00;
29   PORTC = 0x03;
30
31   //Initialisierung des TWI
32   TWAR = 0x00;
33   //Einstellen des langsamsten Takts
34   TWBR = 0xff;
35   TWCR = 0x00;
36   TWDR = 0x00;
37   TWSR = 0x00;
38 }
39
40 void usart_init(void)
41 {
42   unsigned char x;
43
44   // Baudrate setzen
45   UBRRH = (unsigned char)(((CLOCK/(16UL * BAUD)) - 1)>>8);
46   UBRRL = (unsigned char)((CLOCK/(16UL * BAUD)) - 1);
47
48   // Transmitter und Receiver einschalten
49   UCSRB |= (1 << TXEN ) | (1 << RXEN );
50   //8 Datenbits
51   UCSRC |= (1 << URSEL ) | (1 << UCSZ1 ) | (1 << UCSZ0 );
52
```

```
53   // Hilfsvariable leert den Empfänger
54   x = UDR;
55 }
56
57 static int usart_write(char x, FILE * stream)
58 {
59   if (x == '\n')
60     usart_write('\r', stream );
61   loop_until_bit_is_set(UCSRA, UDRE);
62   UDR = x;
63   return 0;
64 }
65
66 unsigned char twi_send(unsigned char adres, unsigned char daten)
67 {
68   // Schleifenvariable erstellen
69   int moveon=1;
70
71   // Start condition
72   TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
73   loop_until_bit_is_set(TWCR, TWINT);
74
75   // Adresse + Write-Bit senden
76   TWDR = (adres) | (TW_WRITE);
77   TWCR = (1 << TWINT) | (1 << TWEN);
78   loop_until_bit_is_set(TWCR, TWINT);
79
80   // Daten senden
81   TWDR = daten;
82   TWCR = (1 << TWINT) | (1 << TWEN);
83   loop_until_bit_is_set(TWCR, TWINT);
84
85   // Neue Start Condition -> Master Receiver Mode
86   TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
87   loop_until_bit_is_set(TWCR, TWINT);
88
89   while(moveon==1)
90   {
91     switch(TW_STATUS)
92     {
93       case TW_REP_START:
94         // Status REPEATED START betreten
95         // SLA + Read(bit 1), Slave adressieren
96         TWDR = (adres) | (TW_READ);
97         TWCR = (1 << TWINT) | (1 << TWEN);
98         loop_until_bit_is_set(TWCR, TWINT);
99         moveon=1;
100        break;
101      case TW_MR_SLA_ACK:
102        // Slave hat mit ACK auf eigene Adresse reagiert
103        // Status TW_MR_SLA_ACK betreten
104        // Empfangen der Daten vorbereiten inkl. NACK
```

```
105         TWCR = (1<<TWEN) | (1<<TWINT);
106         loop_until_bit_is_set(TWCR, TWINT);
107         moveon=1;
108         break;
109       case TW_MR_DATA_NACK:
110         // Daten erhalten, NACK gesendet
111         // Status TW_MR_DATA_NACK betreten
112         databuf = TWDR;
113         // Stop-Condition
114         TWCR = (1 << TWEN)|(1 << TWSTO)|(1<<TWINT);
115         loop_until_bit_is_set(TWCR,TWSTO);
116         moveon=0;
117         break;
118       default:
119         // Stop-Condition
120         TWCR = (1 << TWEN)|(1 << TWSTO)|(1<<TWINT);
121         loop_until_bit_is_set(TWCR,TWSTO);
122         return -1;
123     }
124   }
125   // Daten zurückgeben
126   return databuf;
127 }
128 
129 void twi_display(char* str)
130 {
131   int j;
132 
133   // Display löschen
134   twi_send(TAKTSTRASSE, 'x');
135 
136   for (j=0; j<strlen(str); j++)
137   {
138     // Display mit einem Zeichen beschreiben
139     twi_send(TAKTSTRASSE, 'd');
140     twi_send(TAKTSTRASSE, str[j]);
141   }
142 }
143 
144 int main(void)
145 {
146   int i=1;
147   databuf = 0x00;
148   stdout = &mystdout;
149 
150   twi_init();
151   usart_init();
152 
153   // Aktuelle Portbelegungen auslesen
154   byteB = twi_send(TAKTSTRASSE, 0xbb);
155   byteC = twi_send(TAKTSTRASSE, 0xcc);
156   byteD = twi_send(TAKTSTRASSE, 0xdd);
```

```
157
158  while(1)
159  {
160    // Fototransistor an der Einlegestation wird ausgelöst
161    if (twi_send(TAKTSTRASSE, 0xd0) >= 0xaa)
162    {
163      // Ablauf:
164      // (1) Zuführband an
165
166      twi_display("Teilabschnitt Einlegestation\nbetreten");
167
168      // (1)
169      twi_send(TAKTSTRASSE, 0xce);
170      byteC |= 0x10;
171      twi_send(TAKTSTRASSE, byteC);
172    }
173
174    // Fototransistor am linken Schieber wird ausgelöst
175    if (twi_send(TAKTSTRASSE, 0xd1) >= 0xaa)
176    {
177      // Ablauf:
178      // (1) Zuführband an
179      // (2) Delay
180      // (3) Zuführband aus
181      // (4) Linker Schieber Richtung auf "VOR"
182      // (5) Linker Schieber an
183      // (6) Auf Aktivierung des Tasters Linke Seite vorne warten
184      // (7) Richtung des Schiebers linke Seite auf "ZURUECK" setzen
185      // (8) Auf Aktivierung des Tasters Linke Seite hinten warten
186      // (9) Linker Schieber aus
187      // (10) Band Fräse Richtung auf "VOR"
188      // (11) Band Fräse an
189
190      twi_display("Teilabschnitt Schieber L\nbetreten");
191
192      // (1)
193      twi_send(TAKTSTRASSE, 0xce);
194      byteC |= 0x10;
195      twi_send(TAKTSTRASSE, byteC);
196
197      // (2)
198      for (i=1; i<45; i++)
199        _delay_ms(32);
200
201      // (3)
202      twi_send(TAKTSTRASSE, 0xce);
203      byteC &= ~(0x10);
204      twi_send(TAKTSTRASSE, byteC);
205
206      // (4)
207      twi_send(TAKTSTRASSE, 0xde);
208      byteD |= (0x40);
```

```
209         twi_send(TAKTSTRASSE, byteD);
210 
211         // (5)
212         twi_send(TAKTSTRASSE, 0xde);
213         byteD |= (0x80);
214         twi_send(TAKTSTRASSE, byteD);
215 
216         // (6)
217         while((twi_send(TAKTSTRASSE, 0xcc) & 0x04) != 0x00)
218         {
219         }
220 
221         // (7)
222         twi_send(TAKTSTRASSE, 0xde);
223         byteD &= ~(0x40);
224         twi_send(TAKTSTRASSE, byteD);
225 
226         // (8)
227         while((twi_send(TAKTSTRASSE, 0xcc) & 0x08) != 0x00)
228         {
229         }
230 
231         // (9)
232         twi_send(TAKTSTRASSE, 0xde);
233         byteD &= ~(0x80);
234         twi_send(TAKTSTRASSE, byteD);
235 
236         // (10)
237         twi_send(TAKTSTRASSE, 0xce);
238         byteC |= (0x20);
239         twi_send(TAKTSTRASSE, byteC);
240 
241         // (11)
242         twi_send(TAKTSTRASSE, 0xde);
243         byteD |= (0x20);
244         twi_send(TAKTSTRASSE, byteD);
245       }
246 
247       // Fototransistor an der Fräsmaschine wird ausgelöst
248       if (twi_send(TAKTSTRASSE, 0xd2) >= 0xaa)
249       {
250         // Ablauf:
251         // (1) Band Fräse aus
252         // (2) Fräse an
253         // (3) Delay
254         // (4) Fräse aus
255         // (5) Band Fräse Richtung auf "VOR"
256         // (6) Band Fräse an
257         // (7) Band Bohrer Richtung auf "VOR"
258         // (8) Band Bohrer an
259         // (9) Delay um Fräse zu verlassen
260 
```

```
261       twi_display("Teilabschnitt Fraese\nbetreten");
262
263       // (1)
264       twi_send(TAKTSTRASSE, 0xde);
265       byteD &= ~(0x20);
266       twi_send(TAKTSTRASSE, byteD);
267
268       // (2)
269       twi_send(TAKTSTRASSE, 0xce);
270       byteC |= (0x40);
271       twi_send(TAKTSTRASSE, byteC);
272
273       // (3)
274       for (i=1; i<45; i++)
275         _delay_ms(32);
276
277       // (4)
278       twi_send(TAKTSTRASSE, 0xce);
279       byteC &= ~(0x40);
280       twi_send(TAKTSTRASSE, byteC);
281
282       // (5)
283       twi_send(TAKTSTRASSE, 0xce);
284       byteC |= (0x20);
285       twi_send(TAKTSTRASSE, byteC);
286
287       // (6)
288       twi_send(TAKTSTRASSE, 0xde);
289       byteD |= (0x20);
290       twi_send(TAKTSTRASSE, byteD);
291
292       // (7)
293       twi_send(TAKTSTRASSE, 0xbe);
294       byteB |= (0x01);
295       twi_send(TAKTSTRASSE, byteB);
296
297       // (8)
298       twi_send(TAKTSTRASSE, 0xde);
299       byteD |= (0x10);
300       twi_send(TAKTSTRASSE, byteD);
301
302       // (9)
303       for (i=1; i<45; i++)
304         _delay_ms(32);
305     }
306
307     // Fototransistor am Bohrer wird ausgelöst
308     if (twi_send(TAKTSTRASSE, 0xd3) >= 0xaa)
309     {
310       // Ablauf:
311       // (1) Band Fräse aus
312       // (2) Band Bohrer aus
```

```
313     // (3) Bohrer an
314     // (4) Delay
315     // (5) Bohrer aus
316     // (6) Richtung Band Bohrer auf "VOR"
317     // (7) Band Bohrer an
318     // (8) Delay
319     // (9) Band Bohrer aus
320     // (10) Richtung des rechten Schiebers auf "VOR"
321     // (11) Rechter Schieber an
322     // (12) Auf Aktivierung des Tasters rechte Seite vorne warten
323     // (13) Richtung des Schiebers rechte Seite auf "ZURUECK" setzen
324     // (14) Auf Aktivierung des Tasters rechte Seite hinten warten
325     // (15) Schieber rechte Seite aus
326     // (16) Auslagerband an
327
328     twi_display("Teilabschnitt Bohrer\nbetreten");
329
330     // (1)
331     twi_send(TAKTSTRASSE, 0xde);
332     byteD &= ~(0x20);
333     twi_send(TAKTSTRASSE, byteD);
334
335     // (2)
336     twi_send(TAKTSTRASSE, 0xde);
337     byteD &= ~(0x10);
338     twi_send(TAKTSTRASSE, byteD);
339
340     // (3)
341     twi_send(TAKTSTRASSE, 0xce);
342     byteC |= (0x80);
343     twi_send(TAKTSTRASSE, byteC);
344
345     // (4)
346     for (i=1; i<45; i++)
347       _delay_ms(32);
348
349     // (5)
350     twi_send(TAKTSTRASSE, 0xce);
351     byteC &= ~(0x80);
352     twi_send(TAKTSTRASSE, byteC);
353
354     // (6)
355     twi_send(TAKTSTRASSE, 0xbe);
356     byteB |= (0x01);
357     twi_send(TAKTSTRASSE, byteB);
358
359     // (7)
360     twi_send(TAKTSTRASSE, 0xde);
361     byteD |= (0x10);
362     twi_send(TAKTSTRASSE, byteD);
363
364     // (8)
```

```
365       for (i=1; i<45; i++)
366         _delay_ms(32);
367 
368       // (9)
369       twi_send(TAKTSTRASSE, 0xde);
370       byteD &= ~(0x10);
371       twi_send(TAKTSTRASSE, byteD);
372 
373       // (10)
374       twi_send(TAKTSTRASSE, 0xbe);
375       byteB |= (0x04);
376       twi_send(TAKTSTRASSE, byteB);
377 
378       // (11)
379       twi_send(TAKTSTRASSE, 0xbe);
380       byteB |= (0x08);
381       twi_send(TAKTSTRASSE, byteB);
382 
383       // (12)
384       while((twi_send(TAKTSTRASSE, 0xaa) & 0x40) != 0x00)
385       {
386       }
387 
388       // (13)
389       twi_send(TAKTSTRASSE, 0xbe);
390       byteB &= ~(0x04);
391       twi_send(TAKTSTRASSE, byteB);
392 
393       // (14)
394       while((twi_send(TAKTSTRASSE, 0xaa) & 0x20) != 0x00)
395       {
396       }
397 
398       // (15)
399       twi_send(TAKTSTRASSE, 0xbe);
400       byteB &= ~(0x08);
401       twi_send(TAKTSTRASSE, byteB);
402 
403       // (16)
404       twi_send(TAKTSTRASSE, 0xbe);
405       byteB |= (0x02);
406       twi_send(TAKTSTRASSE, byteB);
407     }
408 
409     // Fototransistor hinter dem Auslagerband wird ausgelöst
410     if (twi_send(TAKTSTRASSE, 0xd4) >= 0xaa)
411     {
412       // Ablauf:
413       // (1) Auslagerband an
414       // (2) Delay
415       // (3) Auslagerband aus
416 
```

```
417         twi_display("Teilabschnitt Auslagerband\nbetreten");
418
419         // (1)
420         twi_send(TAKTSTRASSE, 0xbe);
421         byteB |= (0x02);
422         twi_send(TAKTSTRASSE, byteB);
423
424         // (2)
425         for (i=1; i<45; i++)
426           _delay_ms(32);
427
428         // (3)
429         twi_send(TAKTSTRASSE, 0xbe);
430         byteB &= ~(0x02);
431         twi_send(TAKTSTRASSE, byteB);
432
433         twi_display("Ende der Fertigung\nerreicht");
434       }
435     }
436     return 0;
437 }
```