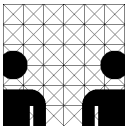
Bachelorarbeit
im Studiengang Informatik

Konzept und Prototyp für ein taktiles Sensorarray[1]

am Arbeitsbereich für
Technische Aspekte Multimodaler Systeme,
Universität Hamburg

vorgelegt von
Nikolas Slottke
Hendrik Udo Linne

April 2011



betreut von
Prof. Dr. Jianwei Zhang
Dr. Norman Hendrich

Zusammenfassung

Diese Bachelorarbeit behandelt den Aufbau eines Konzeptes und eines Prototypen für ein taktiles Sensorarray im Rahmen des "Handle-Project". Dabei geht es darum die Positionsbestimmung einer mechanischen Hand auf einem zu greifenden Objekt durchzuführen. Diese gewonnen Daten sollen einem Rechner übermittelt werden, der entsprechend die Steuerung der Roboter-Hand verändern kann.

Die Arbeit gliedert sich, neben der Vorstellung des angedachten Konzeptes, in zwei Hauptteile. Der erste Teil befasst sich mit der Software. Für dieses Teilgebiet wird wiederum ein eigenes Konzept erstellt und umgesetzt. Der zweite Teil behandelt die Hardware des Systems. Die Erstellung entsprechender Schaltungen dient der späteren Realisierung des Prototypen. Dabei werden zunächst Versuchsaufbauten verwendet, bevor der Prototyp realisiert wird. Dabei werden auch andere Möglichkeiten der Konstruktion in Betracht gezogen. Zum Schluss wird noch ein Ausblick gegeben, indem auf die Eigenschaften des Prototypen eingegangen wird, um mögliche Entwicklungen oder Erweiterungen aufzuzeigen.

Damit liegt das Hauptaugenmerk dieser Bachelorarbeit auf der Erstellung eines Konzeptes und der Konstruktion eines Prototypen.

Abstract

This bachelor thesis deals with the building of a concept and a prototype of a tactile sensor array, in the context of the "Handle-Project". It's about locating the position of a mechanical hand on an object that is gripped. These collected data shall be transmitted to a computer, which can change -corresponding to these data- the control of the robot hand.

This paper is divided, besides the presentation of the envisaged concept, into two main parts. The first part deals with the software. For this branch is created and implemented its own concept. The second part deals with the hardware of the system. In order to construct a prototype later, the creation of circuit layouts is necessary. Before a prototype can be realized, experimental setups are used first. Other possibilities for a construction are also contemplated. Finally an outlook is given, in which the characteristics of the prototype are named in order to demonstrate possible developments or extensions.

So the main focus of this bachelor thesis is set on the development of the concept and the construction of a prototype.

[1]Zweite und überarbeitete Version der eingereichten Bachelorarbeit.

Inhaltsverzeichnis

Abbildungsverzeichnis

Einleitung

1

1.1 Motivation

In der Robotik sind die Robotersysteme, wie auch der Mensch, nur über Sensorik mit der Umwelt verbunden. Ein Weltbild kann so geschaffen werden. Dies ist aber nur so gut, wie auch die Sensoren auflösend sind. Existieren für bestimmte Bereiche der Welt keine Sensoren, so kann dieser Weltausschnitt auch nicht mit in das Weltbild des Robotersystems übernommen werden.

Seit Beginn der Forschung in dem Bereich der Robotik ist es das Bestreben, die Sensorik weiter auszubauen, neue Systeme zu entwickeln und bestehende Systeme zu verfeinern und zu optimieren, um die Leistungsfähigkeit des einzelnen Robotersystems zu erhöhen und zu verbessern. Denn je leistungsfähiger ein Roboter ist, umso effektiver kann er mit seiner Umwelt interagieren.

Eine Herausforderung in der Welt der Robotik ist die Manipulation von Objekten mit Hilfe eines oder mehrerer Robotersystemen. Dabei ist es wichtig, dass das Robotersystem ein möglichst genaues Bild davon erhält, was im Bezug auf seine Aufgabe zu manipulieren ist. Das Weltbild und somit auch die Sensorik spielen also eine große Rolle.

Ein gutes Vorbild ist die Interaktion des Menschen mit seiner Umwelt. Diese nachzuempfinden und zu verstehen, ermöglicht es Robotersysteme zu konstruieren, die dem Menschen ähnliche Interaktionsformen aufweisen. Die Handlungsweise des Menschen ist beobachtbar und somit auch in vielen Bereichen auf ein Robotersystem übertragbar. Eine solche Übertragung von "technischem Equipment" ist die Konstruktion von mechanischen Händen, die es einem Robotersystem ermöglichen sollen, die Umwelt nach dem Vorbild des Menschen zu manipulieren.

Das "Handle-Project", gestartet am 2.Februar 2009 und befristet bis 1.Februar 2013, ist ein europäisches Projekt mehrerer Universitäten mit dem Ziel die menschliche Benutzung der Hand zur Manipulation von Objekten in der Umwelt zu analysieren und nachzuvollziehen, um diese Erkenntnisse auf eine künstliche Hand zu übertragen [hpr07, vgl.].

Einem Robotersystem einer mechanischen Hand nun zu ermöglichen, seine Umwelt nach menschlichem Vorbild, eventuell sogar autonom, zu manipulieren, bedarf es detaillierter Informationen über die zu manipulierende Umwelt. Auch hier kann sich wieder dem menschlichen Vorbild bedient werden. Der Mensch ist in der Lage, ein mit der Hand gegriffenes Objekt

zu erfühlen. Besser noch zu ertasten. Die Sensorik muss nun eine Möglichkeit aufbieten das menschliche Tastempfinden zu replizieren.

1.2 Zielsetzung

Wie lässt sich ein Sensor konstruieren, der in der Lage ist eine möglichst genaue Positionsbestimmung zwischen Hand und Objekt zu gewährleisten und einen Aufbau bietet der einen flexiblen Einsatz ermöglicht?

Im Rahmen dieser Bachelorarbeit soll nun ein Konzept für eine prototypische Sensormatrix erarbeitet und der Prototyp des taktilen Sensorarrays aufgebaut werden. Es soll ermöglicht werden eine Positionsbestimmung zwischen einer mechanischen Hand und einem zu greifenden Objekt herzustellen. Dabei ist zu beachten, dass je mehr Sensorflächen auf der mechanischen Hand oder dem Objekt selbst untergebracht werden können, umso auflösender sind die Daten, die zur Verarbeitung zur Verfügung stehen. Die Möglichkeit der Miniaturisierung spielt also eine gewichtige Rolle, aber auch die Maximierung der Verarbeitung der erhaltenen Daten, welche mit der Miniaturisierung zunehmen, darf nicht unbeachtet bleiben. Angestrebt wird hierbei ein modularer Aufbau der erweiterbar ist. Deshalb muss eine Grundlage geschaffen werden mehrere solcher Sensorfelder zu verschalten. Dadurch wird eine Kommunikation erforderlich, die zwischen den Mikrocontroller gesteuerten Sensorfeldern und seiner Kontrolleinheit sicher zu stellen ist.

1.3 Überblick

Neben der mechanischen Konstruktion eines Sensorfeldes wird in der schriftlichen Arbeit die Theorie und das Konzept für das prototypische taktiles Sensorarray dargestellt.

Zunächst soll das erarbeitete Konzept im Vergleich zu aktuellen bereits bestehenden Systemen vorgestellt werden. Es dient somit als Einstieg in die Arbeit und soll es ermöglichen die Arbeit von den Grundgedanken her zu verstehen und eine Einordnung in die bestehenden wissenschaftlichen Arbeiten ermöglichen.

Als nächstes wird in dieser schriftlichen Ausarbeitung die entwickelte Software vorgestellt. Die Darstellung der Software ist in zwei Teile gegliedert, wobei zunächst die Software, welche auf den Hardwaremodulen Einsatz findet, beschrieben wird. Dazu wird auf jeden Teil der entwickelten Module separat eingegangen. Als weiteres wird die entwickelte Anbindungssoftware auf der Computerseite dargestellt.

Der dritte Teil der schriftlichen Ausarbeitung befasst sich mit der Hardware die entwickelt und umgesetzt wurde. Hier wird zunächst ein Überblick über die eingesetzten Komponenten und Bussysteme gegeben. Danach folgt die Erläuterung der aufgebauten Module anhand ihrer elektronischen Schaltungen. Auch wird in diesem Abschnitt erläutert wie das Konzept der Kontaktmatrizen erarbeitet und umgesetzt wurde.

Als Abschluss der Arbeit folgt ein Ausblick über Erweiterungen des Systems, die, für die spätere Weiterentwicklung des Systems, möglich und denkbar sein könnten. Auch werden hier verschiedene Verbesserungen vorgestellt, die bei der Entwicklung und Test des Systems auffällig waren und auch bei einer Weiterentwicklung des Systems berücksichtigt werden sollten.

Konzept

2

2.1 State of the Art

2.1.1 Funktion der menschlichen Hand

Das Manipulieren von Objekten mit der menschlichen Hand besteht aus einer Reihe von Aktionen, die jeweils eine untergeordnete Aufgabe erfüllen. Jede dieser Aufgaben hängt von verschiedenen Parametern ab, wie z.B. Größe, Gewicht, Form und Oberfläche des Objektes. Der Tastsinn der menschlichen Hand basiert auf taktilen Signalen, welche von Mechanorezeptoren unter der Haut wahrgenommen werden und die äußeren Schichten der Haut entsprechend anregen um beispielsweise ein Objekt festzuhalten.

2.1.2 Existierende Messverfahren

Ohmsche Sensoren: Ohmsche Sensoren messen eine Belastung durch die Veränderung eines Widerstandes, wenn der Sensor mit einer Kraft belastet wird. Typische ohmsche Sensoren sind lange Schlangenförmige Strukturen, die bei Deformation in ihrem Querschnitt verkürzt und ihrem Leitvermögen vergrößert werden. Hierbei ist allerdings in erster Linie die Veränderung der mechanischen Form relevant im Gegensatz zur Veränderung des elektrischen Widerstandes.

Belastungssensoren aus Mikromaterial haben den Vorteil, dass sie direkt mit ihrer Messelektronik und anderen Mikroelekromechanischen Systemen (MEMS) integriert werden können. Sie sind klein, haben eine hohe Sensitivität und räumliche Auflösung und können mit fest etablierten Herstellungstechniken produziert werden, beispielsweise auf einem flexiblem PCB (printed circuit board).

Kapazitive Sensoren: Kapazitive Sensoren sind die sensitivste Technik um kleine Biegungen von Material temperaturunabhängig festzustellen. Der Sensor besteht aus zwei vergoldeten runden Platten mit einer zwischen den Platten liegenden dielektrischen Schicht Parylene. Die auf den Sensor wirkende Kraft, kann durch Veränderung der Kapazität des Sensors gemessen werden.

Piezoelektrische Sensoren: Piezoelektrische Sensoren wandeln eine auf sie wirkende Kraft als elektrische Spannung um. Die piezoelektrischen Elemente brauchen hierbei keine eigene Span-

nungsversorgung. Die Sensoren reagieren sehr sensibel mit hohen Spannungen selbst auf nur kleine Deformation. Allerdings nimmt die Spannung nach einer gewissen Zeit ab, daher sind die Sensoren nur tauglich für dynamische Veränderung von Kräften und nicht für einen längeren Zeitraum konstant wirkende Kräfte.

Optische Sensoren: Optische Sensoren versuchen das Probleme der Verkabelung bei vielen taktilen Sensoren zu lösen. Hierbei werden Glasfaserkabel benutzt um Signale zu übertragen. Die Glasfaserkabel sind in einem Silikonelastomer eingelassen. Die Glasfaserkabel verbinden dabei je eine LED-Lichtquelle mit einem CCD-Sensor. Diese Sensoren können eine Veränderung der Lichtintensität messen, wenn eine Kraft auf das Silikonelastomer mit den Glasfaserkabeln einwirkt.

Organic field-effect transistors (OFETs): Organische Feldeffekttransistoren (OFETs) können als Sensoren zur Messung von direkt einwirkendem Druck benutzt werden. Als Prototyp wurden mehrere OFETs auf eine Mylar-Folie angebracht, die sowohl als dielektrische Schicht und als Übertragungsmedium des Drucks fungiert. Der Druck kann wie bei den kapazitiven Sensoren über die Veränderung der Kapazität der OFETs gemessen werden.

2.2 Konzeptdarstellung

Unser Konzept ist es, die Verbindung von Objekt und Handschuh durch Verbindung mit elektrischen Leitungen zu realisieren. Dabei sollen Kontaktflächen jeweils auf Objekt und Handschuh aufgebracht werden, die beim Berühren miteinander verbunden werden sollen. Jede dieser Kontaktflächen soll dabei an einen Eingang/Ausgang eines Mikrocontrollers angeschlossen sein. Der Mikrocontroller des Handschuhs soll permanent immer wieder einen Kontakt unter Strom setzen, bis er alle Kontakte abgearbeitet hat. Dann soll er wieder von vorne anfangen. Die Frequenz mit der alle Kontakte jeweils nacheinander unter Strom gesetzt werden, soll mindestens 10 Hz betragen, um alles unter Realbedingungen auch in Echtzeit auswerten zu können. Bei Verbindung soll also ein Strom zu dem Eingang auf dem Objekt fließen. Der Mikrocontroller des Objektes soll dabei alle Eingänge permanent auf Verbindung abfragen. Koordiniert werden sollen dabei die Mikrocontroller von einem weiteren Mikrocontroller, einem Kommunikationscontroller. Dieser soll Befehle an den Controller des Objektes senden welcher Kontakt unter Strom gesetzt werden soll und soll daraufhin den Controller des Objektes nach dem Zustand der Kontaktflächen auf dem Objekt fragen. Sollte also eine Verbindung bestehen, kann rekonstruiert werden welcher Kontakt des Handschuhs mit welchem Kontakt des Objektes verbunden wäre. Diese Informationen sollen dann über eine serielle Schnittstelle vom Kommunikationscontroller an einen PC gesendet und dort ausgewertet werden.

Software

3

3.1 Atmel

Die Atmel® Mikrocontroller werden mit C programmiert. Dazu wird die Eclipse C++ Entwicklungsumgebung mit dem AVR-Eclipse Plugin[avr11a, vgl.], der Atmel® AVR® Toolchain sowie dem avr-gcc Compiler benutzt[avr11c, vgl.]. Die Programme werden dann mit dem Atmel® STK500 Entwicklungsboard auf die Mikrocontroller geflasht.

3.1.1 Kommunikationscontroller (Comcontroller)

Der Controller ist ein Atmel® ATmega 644-20 PU, welcher mit einem internen Takt von 8MHz läuft. Er verfügt über 4 Ports, die je 8 Pins haben, nutzbar als Ein- oder Ausgang[Atm10a, vgl.]. Da der Comcontroller nur die Kommunikation mit den anderen Controllern und dem PC steuert, werden nur die ersten beiden Pins von Port C für den TWI-Bus und die ersten beiden Pins von Port D für den seriellen Anschluss benötigt.

Zuerst wird im Quellcode die Frequenz der CPU des Controllers festgelegt, falls sie nicht schon in der Entwicklungsumgebung gesetzt wurde. Beim ATmega 644-20 PU sind das 8.000.000 Hz.

```
1 /*
2  * Define CPU Frequency
3  */
4 #ifndef F_CPU
5 #define F_CPU 8000000
6 #endif
```

Außerdem werden natürlich alle benötigten Headerdateien eingebunden inklusive ein paar TWI Funktionen[twi08, vgl.].

```
1 /*
2  * Includes
3  */
4 #include <avr/io.h>
```

```
5  #include <avr/interrupt.h>
6  #include <avr/pgmspace.h>
7  #include <util/delay.h>
8  #include <util/twi.h>
9  #include <stdlib.h>
10 #include "TWI_Master.h"
```

Die main-Methode beinhaltet den Programmablauf der vom Mikrocontroller ausgeführt wird. In dieser Methode wird zuerst der USART mit einer Baudrate von 38400bps initialisiert.

```
1  USART_Init(((F_CPU / 16L / 38400L) - 1));
```

Außerdem werden alle benötigten Variablen initialisiert.

```
1  uint8_t i;
2  uint8_t HandData = 0;
3  uint8_t ObjectDataA = 0;
4  uint8_t ObjectDataB = 0;
5  uint8_t ObjectDataC = 0;
6  uint8_t ObjectDataD = 0;
7  uint8_t HandAddress = 15;
8  uint8_t ObjectAddress = 16;
9  // Send cycle if 1
10 int sendcycle = 0;
11 // Cycle count
12 int x = 0;
```

Danach werden alle Interrupts gelöscht und es wird 1,5 Sekunden auf die anderen Mikrocontroller gewartet, dann wird der TWI-Bus initialisiert.

```
1  // Clear any interrupt
2  cli ();
3  // Wait for other AVR to be ready
4  _delay_ms(1500);
5  USART_TransmitString("\n Communication Controller ready!\n");
6  //Initiate TWI Master Interface with bitrate of 100000 Hz
7  if (!TWIM_Init(100000)) {
8          USART_TransmitString(" Error initiating TWI interface
              \n");
9          while (1)
10                 ;
11 }
```

Darauf folgend beginnt ein sich immer wiederholender Programmabschnitt. In diesem wird der Handcontroller angewiesen einen Pin auf High (1) zu setzen und Rückmeldung darüber zu geben. Dann wird der Status der einzelnen Ports des Objektcontrollers abgefragt und in den Variablen vorübergehend gespeichert. Falls nun mindestens eine Verbindung zwischen Hand und Objekt besteht, wird der entsprechende Pin der Hand und der oder eventuell mehrere

verbundene Pins des Objektes an den PC gesendet. Dieses wird 30 Mal für alle Pins des Handcontrollers wiederholt und am Schluss wird der dazugehörige Programmzyklus an den PC gesendet. Daraufhin beginnt die Schleife mit dem nächsten Zyklus erneut.

```
1  while (1) {
2          x++;
3          for (i = 1; i <= 30; i++) {
4                  /*
5                   * Send to Hand Controller
6                   */
7                  if (!TWIM_Start(HandAddress, TWIM_WRITE)) {
8                          TWIM_Stop();
9                          USART_TransmitString(" Could not
                              start TWI for HANDWRITE\n");
10                 } else {
11                         TWIM_Write(i);
12                         TWIM_Stop();
13                         _delay_ms(0.01);
14                 }
15
16                 /*
17                  * Receive from Hand Controller
18                  */
19                 if (!TWIM_Start(HandAddress, TWIM_READ)) {
20                         TWIM_Stop();
21                         USART_TransmitString(" Could not
                              start TWI for HANDREAD\n");
22                 } else {
23                         HandData = TWIM_ReadNack();
24                         if (HandData != i) {
25                                 USART_TransmitString(" Hand
                                      Controller Error.");
26                         }
27                         TWIM_Stop();
28                 }
29
30                 /*
31                  * Send to Object Controller
32                  */
33                 if (!TWIM_Start(ObjectAddress, TWIM_WRITE)) {
34                         TWIM_Stop();
35                         USART_TransmitString(" Could not
                              start TWI for OBJECTWRITE\n");
36                 } else {
37                         TWIM_Write(100);
38                         TWIM_Stop();
```

```
39                          _delay_ms(0.01);
40                  }
41
42                  /*
43                   * Receive from Object Controller
44                   */
45                  if (!TWIM_Start(ObjectAddress, TWIM_READ)) {
46                          TWIM_Stop();
47                          USART_TransmitString(" Could not
                               start TWI for OBJECTREAD\n");
48                  } else {
49                          ObjectDataA = TWIM_ReadAck();
50                          ObjectDataB = TWIM_ReadAck();
51                          ObjectDataC = TWIM_ReadAck();
52                          ObjectDataD = TWIM_ReadNack();
53                          if ((ObjectDataA != 0) || (
                               ObjectDataB != 0) || (ObjectDataC
                               != 0) || (ObjectDataD != 0)) {
54                                  sendcycle = 1;
55                                  USART_TransmitString("S");
56                                  USART_TransmitInt(i);
57                                  USART_TransmitString("\n");
58                                  if (ObjectDataA != 0) {
59                                          USART_TransmitString(
                                               "A");
60                                          USART_TransmitHex(
                                               ObjectDataA);
61                                          USART_TransmitString(
                                               "\n");
62                                  }
63                                  if (ObjectDataB != 0) {
64                                          USART_TransmitString(
                                               "B");
65                                          USART_TransmitHex(
                                               ObjectDataB);
66                                          USART_TransmitString(
                                               "\n");
67                                  }
68                                  if (ObjectDataC != 0) {
69                                          USART_TransmitString(
                                               "C");
70                                          USART_TransmitHex(
                                               ObjectDataC);
71                                          USART_TransmitString(
                                               "\n");
```

```
72                                      }
73                                      if (ObjectDataD != 0) {
74                                              USART_TransmitString(
                                                   "D");
75                                              USART_TransmitHex(
                                                   ObjectDataD);
76                                              USART_TransmitString(
                                                   "\n");
77                                      }
78                              }
79                              TWIM_Stop();
80                      }
81          }
82
83          /*
84           * Send cycle count via USART
85           */
86          if (sendcycle == 1) {
87                  USART_TransmitString("N");
88                  USART_TransmitInt(x);
89                  USART_TransmitString("\n\n");
90                  sendcycle = 0;
91          }
92  }
```

Der Comcontroller hat außerdem noch diverse Methoden um Daten über den USART zu senden. Zum Beispiel die Methode USART_Transmit mit der ein einzelnes Zeichen gesendet werden kann. Die Methode sieht wie folgt aus.

```
1  /*
2   * Transmit char via USART
3   */
4  void USART_Transmit(unsigned char data) {
5          /* Wait for empty transmit buffer */
6          while (!(UCSR0A & (1 << UDRE0)))
7                  ;
8          /* Put data into buffer, sends the data */
9          UDR0 = data;
10 }
```

Das Zeichen, welches gesendet werden soll wird der Methode als Parameter übergeben, hier unsigned char data. Die Methode wartet erst bis der Transmit Buffer leer ist, d.h. bis der USART wieder bereit ist zu senden, und schreibt dann erst das Zeichen in den Register UDR0, wodurch dieses gesendet wird.

Außerdem gibt es noch die Methode USART_TransmitString, mit der ein char-Array gesendet werden kann. Diese Methode iteriert durch das char-Array und sendet die einzelnen Zeichen des char-Arrays mit der oben genannten Methode USART_Transmit.

```
1  /*
2   * Transmit String via USART
3   */
4  void USART_TransmitString(char* s) {
5          for (; *s; s++) {
6                  USART_Transmit(*s);
7          }
8  }
```

Dann gibt es noch die Methode USART_TransmitBin mit der ein 8bit Binärwort (uint8_t) gesendet werden kann. Als Präfix wird vor dem Wort '0b' gesendet.

```
1  /*
2   * Transmit uint8_t via USART
3   */
4  void USART_TransmitBin(uint8_t u) {
5          USART_TransmitString("0b");
6          for (int i = 0; i < 8; i++) {
7                  char c = (u & 0x80) ? '1' : '0';
8                  USART_Transmit(c);
9                  u = u << 1;
10         }
11 }
```

Die Methode USART_TransmitInt sendet einen Integer als eine bis zu 5-stellige Dezimalzahl.

```
1  /*
2   * Transmit int via USART
3   */
4  void USART_TransmitInt(int i) {
5          char buf[5];
6          itoa(i, buf, 10);
7          USART_TransmitString(buf);
8  }
```

Und die letzte Methode USART_TransmitHex sendet einen uint8_t als Hexadezimalwert. Diese Methode wird gebraucht, um die Portzustände zu senden und dabei möglichst wenig Zeichen zu senden. Es wird ein zweistelliger Hexadezimalwert gesendet mit dem die $256 = 2^8$ Zustände eines Ports darstellbar sind. Die Methode DecToHex wandelt dabei immer 4 Bit in ein Hexadezimalzeichen um, z.B. das Binärwort '1111' in 'F'.

```
1  /*
2   * Transmit uint8_t as hex value via USART
3   */
4  void USART_TransmitHex(uint8_t u) {
5          int a = ((u & 0xF0) >> 4);
6          int b = (u & 0x0F);
7          //USART_TransmitString("0x");
```

```
 8           USART_Transmit(DecToHex(a));
 9           USART_Transmit(DecToHex(b));
10  }
```

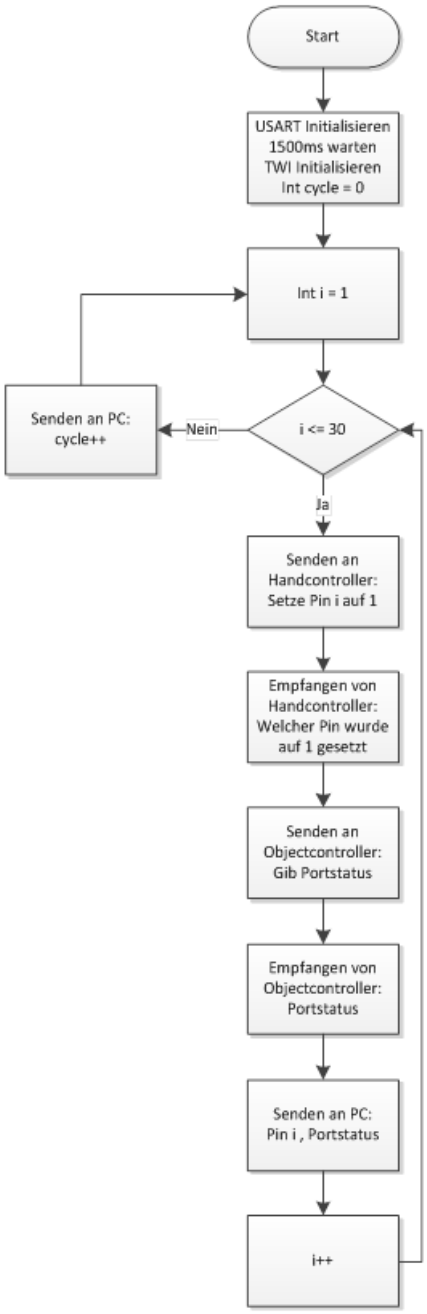


Abbildung 3.1: Programmablaufplan des Comcontrollers

3.1.2 Objektcontroller

Die Software des Objektcontrollers ist recht einfach aufgebaut. Es stehen insgesamt 30 Pins als Eingänge zur Verfügung von Port A, B, C und D, wobei die ersten beiden Pins (PINC0 und PINC1) von Port C für den TWI-Bus reserviert sind[Atm10a, vgl.].

Das Programm läuft folgendermaßen ab. Zuerst werden die benutzten Ports als Eingang definiert. Es werden alle Interrupts gelöscht und 500ms gewartet. Nun wird der TWI-Bus mit der Slave Adresse des Objektcontrollers initialisiert und Variablen für den Status der einzelnen Ports initialisiert.

```
1  // Set Port A, B, C and D to input
2  DDRA = 0x00;
3  DDRB = 0x00;
4  DDRC &= ~((1 << DDC2) | (1 << DDC3) | (1 << DDC4) | (1 <<
       DDC5) | (1 << DDC6) | (1 << DDC7));
5  DDRD = 0x00;
6  PORTA = 0b00000000;
7  PORTB = 0b00000000;
8  PORTC = 0b00000000;
9  PORTD = 0b00000000;
10
11 // Clear any interrupt
12 cli();
13
14 // Wait
15 _delay_ms(500);
16
17 // Start TWI Slave with address 16 and bitrate of 100000 Hz
18 TWIS_Init(16, 100000);
19
20 uint8_t TWIS_ResponseType;
21 uint8_t ObjectDataA = 0;
22 uint8_t ObjectDataB = 0;
23 uint8_t ObjectDataC = 0;
24 uint8_t ObjectDataD = 0;
```

Daraufhin folgt eine sich immer wiederholende while-Schleife in der der Objektcontroller auf Anweisungen vom Comcontroller wartet. Der Comcontroller gibt dem Objektcontroller erst die Anweisung den Status seiner Ports auszulesen und in den Variablen zu speichern, danach wird die Anweisung gegeben diese über den TWI-Bus an den Comcontroller zu senden.

```
1          while (1) {
```

```
2                   if (TWIS_ResponseRequired(&TWIS_ResponseType)
                        ) {
3                           switch (TWIS_ResponseType) {
4                           case TWIS_ReadBytes:
5                                   if (TWIS_ReadNack() == 100) {
6                                           ObjectDataA =
                                                read_portA();
7                                           ObjectDataB =
                                                read_portB();
8                                           ObjectDataC =
                                                read_portC();
9                                           ObjectDataD =
                                                read_portD();
10                                  }
11                                  TWIS_Stop();
12                                  break;
13                          case TWIS_WriteBytes:
14                                  TWIS_Write(ObjectDataA);
15                                  TWIS_Write(ObjectDataB);
16                                  TWIS_Write(ObjectDataC);
17                                  TWIS_Write(ObjectDataD);
18                                  TWIS_Stop();
19                                  break;
20                          }
21                  }
22          }
```

Die Methoden read_port lesen nur den Status der Ports aus und geben diesen als uint8_t zurück. Beim Status von Port C müssen allerdings noch die ersten beiden Pins gelöscht werden, da dort der TWI-Bus kommuniziert und es sonst zur Verfälschung der Daten kommen würde. Die Methoden sehen daher wie folgt aus.

```
1  /*
2   * Read status of Port A
3   */
4  uint8_t read_portA() {
5          return PINA;
6  }
7
8  /*
9   * Read status of Port B
10  */
11 uint8_t read_portB() {
12         return PINB;
13 }
14
```

```
15 /*
16  * Read status of Port C
17  */
18 uint8_t read_portC() {
19         uint8_t u = PINC;
20         u = (u & 0xFC);
21         return u;
22 }
23
24 /*
25  * Read status of Port D
26  */
27 uint8_t read_portD() {
28         return PIND;
29 }
```

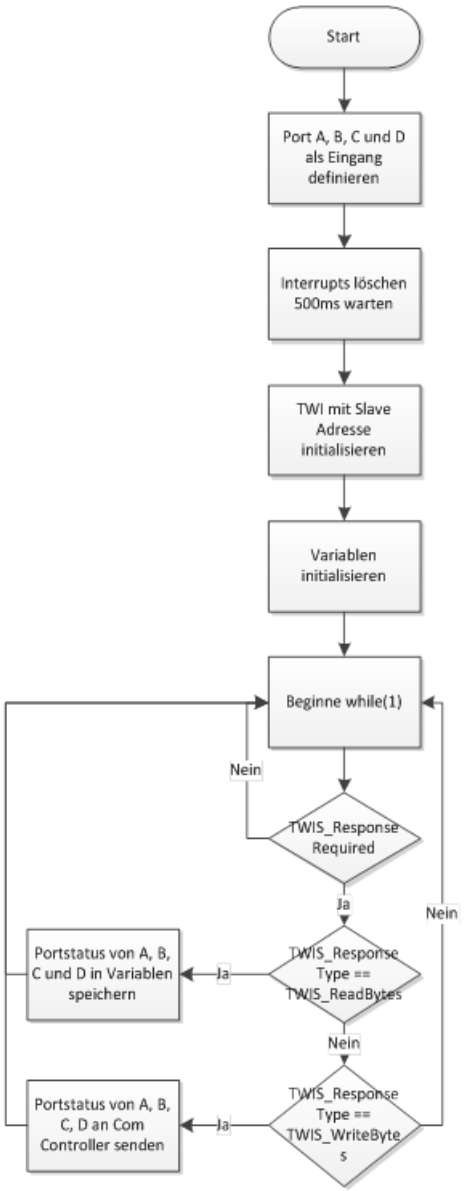


Abbildung 3.2: Programmablaufplan des Objektcontrollers

3.1.3 Handcontroller

Aufgabe des Handcontrollers ist es, möglichst schnell alle Pins der Hand nacheinander einzeln auf High zu setzen, um die Verbindung zum Objekt zu rekonstruieren. Die Anweisung, welcher Pin gesetzt werden soll, bekommt der Handcontroller vom Comcontroller über den TWI-Bus. Vom Comcontroller wird dabei ein Integer-Wert im Bereich von 1 bis 30 gesendet, je nachdem welcher der 30 Pins des Handcontrollers auf High gesetzt werden soll.

Der Programmablauf ist wie folgt. Zuerst werden alle Ports mit Ausnahme der beiden Pins für den TWI-Bus als Ausgang definiert. Es werden alle Interrupts gelöscht und 500ms gewartet. Dann wird der TWI-Bus mit der Slave Adresse des Handcontrollers initialisiert, sowie eine Variable für den zu setzenden Pin und die Antwort des TWI-Busses initialisiert.

```
1  /*
2   * Main
3   */
4  int main(void) {
5          // Set Port A, B, C and D to output
6          DDRA = 0xff;
7          DDRB = 0xff;
8          DDRC = (1 << DDC2) | (1 << DDC3) | (1 << DDC4) | (1
               << DDC5) | (1 << DDC6)       | (1 << DDC7);
9          DDRD = 0xff;
10
11         // Clear any interrupt
12         cli();
13
14         // Wait
15         _delay_ms(500);
16
17         //Start TWI Slave with address 15 and bitrate of
               100000 Hz
18         TWIS_Init(15, 100000);
19
20         uint8_t TWIS_ResponseType;
21         uint8_t pin_to_set;
22
23         while (1) {
24                 if (TWIS_ResponseRequired(&TWIS_ResponseType)
                       ) {
25                         switch (TWIS_ResponseType) {
26                         case TWIS_ReadBytes:
```

```
27                                      pin_to_set = TWIS_ReadNack();
28                                      set_pin(pin_to_set);
29                                      TWIS_Stop();
30                                      break;
31                                  case TWIS_WriteBytes:
32                                      TWIS_Write(read_portstatus())
                                            ;
33                                      TWIS_Stop();
34                                      break;
35                                  }
36                          }
37                  }
38                  return 0;
39  }
```

Die Methode set_pin bekommt als Parameter einen Wert von 1 bis 30 als uint8_t übergeben. 1 bis 8 sind dabei die 8 Pins von Port A, 9 bis 16 sind die 8 Pins von Port B, 17 bis 22 sind die 6 Pins von Port C und 23 bis 30 sind die letzten 8 Pins von Port D. Je nach dem übergebenem Wert wird dann der entsprechende Pin eines Ports auf High (1) gesetzt. Falls ein Wert außerhalb von 1 bis 30 übergeben werden sollte, werden alle Ports auf Low (0) geschaltet.

Des Weiteren gibt es noch die Methode read_portstatus die den Status der Ports als Wert zwischen 1 und 30 zurückgibt. Dieser Wert wird in der main-Methode auch nach dem setzen eines Pins an den Comcontroller gesendet und sollte immer der gleiche Wert sein, der vorher empfangen wurde. Falls dieser Wert sich von dem vom Comcontroller gesendeten Wert unterscheidet, wird vom Comcontroller eine Fehlermeldung an den PC gesendet.

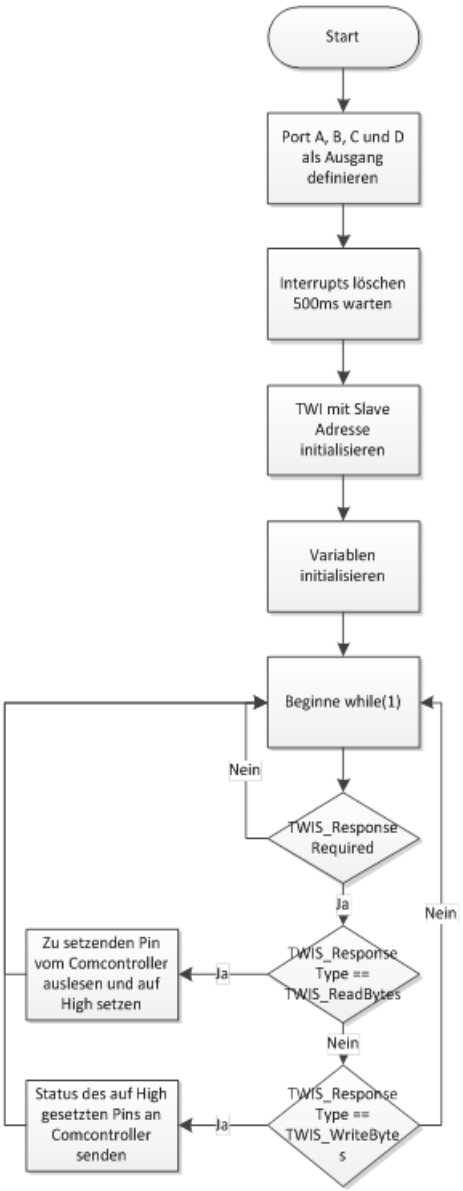


Abbildung 3.3: Programmablaufplan des Handcontrollers

3.1.4 TWI-Bus

Der TWI-Bus (auch I^2C genannt, von Inter-Integrated Circuit) ist ein serieller zwei-drätiger Bus (Two-Wire Interface), bestehend aus einer Daten- (SDA) und Taktleitung (SCL) für die Kommunikation von mehreren Mikrocontrollern untereinander. Jeder der Mikrocontroller hat dabei seine eigene Addresse. Es können dabei theoretisch bis zu 128 Controller an einem TWI Bus zur Kommunikation angeschlossen werden[Atm10b, vgl. Seite 1].

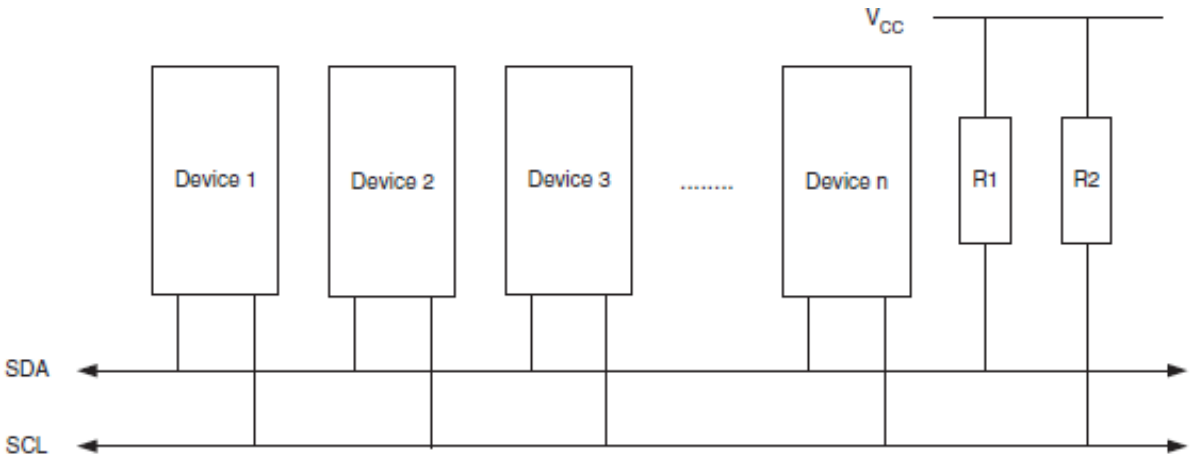


Abbildung 3.4: Aufbau des TWI-Busses[Atm10b, Seite 2, Figure 1]

Der TWI-Bus ist ein multi-master Bus an dem ein oder mehr Controller die Kontrolle über den Bus haben. Datenübertragungen werden immer vom Master ausgelöst. Slaves ist es nur erlaubt Daten auf Anfrage über die Datenleitung (SDA) zu senden. Eine Datenübertragung läuft wie folgt ab:

Zuerst wird von Master ein START Signal gesendet, gefolgt von der 7-bit Slave Adresse und einem Data Direction Bit. Daraufhin muss der angesprochene Slave mit ACK bestätigen, ansonsten wird der Transfer abgebrochen. Wenn der Transfer bestätigt wurde, wird nun abhängig vom Data Direction Bit vom Master oder Slave 8-bit Daten auf der Datenleitung gesendet. Der empfangende Controller bestätigt dann den Empfang der Daten mit ACK. Mit STOP wird dann der Transfer vom Master abgeschlossen. Es können auch mehrere Bytes an Daten in eine Richtung gesendet werden, bevor ein erneuter Transfer mit START oder STOP vom Master aus nötig ist.

Falls ein Slave nicht bereit ist weitere Daten zu verarbeiten, da er beschäftigt ist, kann er den Master in einen warten-Status bringen. Alle Datenpakete sind 9-bit lang, bestehend aus 8-bit Daten und einem Acknowledge (ACK) Bit. Signale wie START, STOP und ACK werden durch den Zustand von SDA und SCL definiert. Ein Übergang von High zu Low auf der Datenleitung

während die Taktleitung High ist, wird beispielsweise als START Signal interpretiert[Atm10b, vgl. Seite 2 und 3].

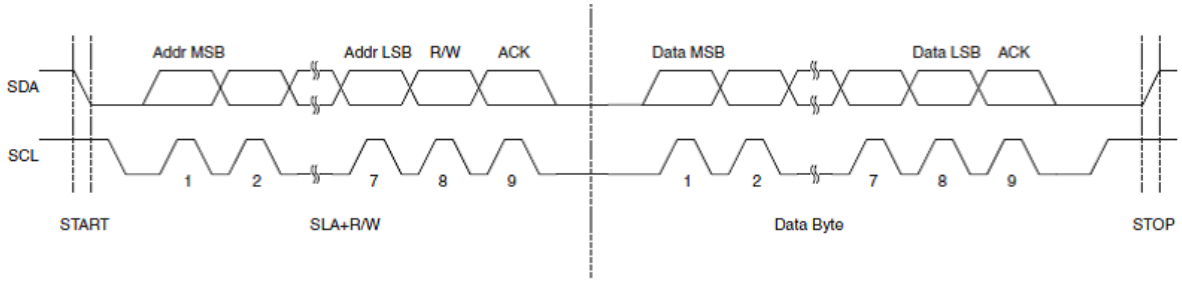


Abbildung 3.5: Datenübertragung beim TWI-Bus[Atm10b, Seite 2, Figure 2]

Das TWI Modul des Atmel® AVR® Mikrocontrollers kann als Master oder Slave arbeiten. Der Mikrocontroller hat einen TWI Statusregister (TWSR) und einen TWI Control Register (TWCR) durch die der Status des TWI mit Hilfe von Statuscodes bestimmt werden kann. Es wird bei den Statuscodes zwischen Master und Slave unterschieden. Das TWI Modul arbeitet mit diesen Zuständen und wird durch Events gesteuert. Falls z.B. ein START Signal gesendet wird und die darauf folgende Adresse mit dem Adressregister (TWAR) des Slaves übereinstimmt, wird das TWINT Flag gesetzt, und ein entsprechendes Interrupt wird ausgeführt. Die Firmware des Slaves liest daraufhin den Statuscode im TWSR und reagiert entsprechend. Alle TWI Events setzen das Interrupt Flag TWINT und die Firmware muss auf Basis des Statusregisters (TWSR) reagieren[Atm10b, vgl. Seite 4].

3.1.5 Serielle Kommunikation (USART)

USART steht für Universal Synchronous/Asynchronous Receiver Transmitter. Der USART ist ein serielles Kommunikationselement des ATmega 644-20 PU[Atm10a, vgl. Seite 164].

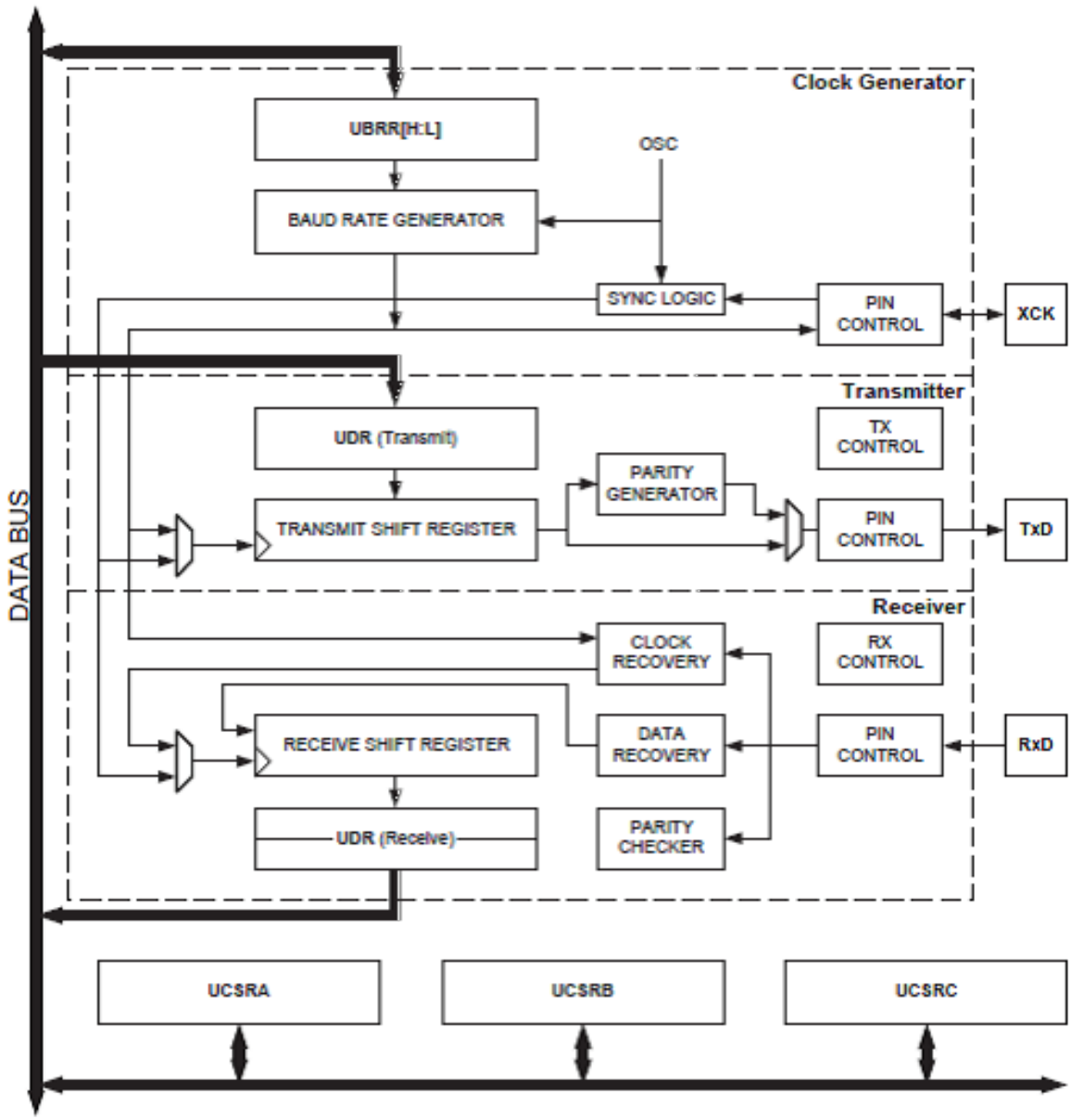


Abbildung 3.6: Aufbau des USART[Atm10a, Seite 165, Figure 17-1]

Die Abbildung 3.6 zeigt die drei Hauptbestandteile des USART: Den Taktgenerator, den Transmitter (Sender) und den Receiver (Empfänger). Alle Bestandteile können auf die Kontrollregister (UCSRA, UCSRB, UCSRC) zugreifen. Der Taktgenerator besteht aus einer Synchronisations-Logikeinheit für externen Takt und einem Baudratengenerator. Der Transfer Clock Pin (XCK) wird nur im synchronen Transfermodus gebraucht. Der Transmitter besteht aus einem einfach gepuffertem Schreibpuffer, einem seriellen Shift Register, Paritätsgenerator und Kontroll-Logikeinheit um unterschiedliche serielle Paketformate zu verarbeiten. Der Schreibpuffer erlaubt kontinuierliches unterbrechungsfreies Senden von Daten. Der Receiver besteht aus Paritätsprüfer Kontroll-Logikeinheit, einem Shift Register und einem Zwei-Level Puffer. Außerdem enthält er noch Recovery-Einheiten für den asynchronen Datenempfang. Der Receiver unterstützt die selben Paketformate wie der Transmitter und ist in der Lage Fehler zu erkennen[Atm10a, vgl. Seite 165].

Baudratenberechnung:

Der USART Baud Rate Register (UBBRn) ist mit einem Down-Counter (Herunterzähler), der mit dem Systemtakt (8MHz) läuft, verbunden. Der Down-Counter lädt den Wert aus UBRRn, wenn er Null erreicht. Zu diesem Zeitpunkt ist genau ein Taktzyklus der Baudrate generiert. Der Transmitter teilt diesen Baudraten-Takt je nach Modus durch 2, 8 oder 16. Die Baudrate wird in bit per second (bps) angegeben[Atm10a, vgl. Seite 166].

Operating Mode	Equation for Calculating Baud Rate[1]	Equation for Calculating UBRR Value
Asynchronous Normal mode (U2Xn = 0)	$BAUD = \frac{f_{OSC}}{16(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{16BAUD} - 1$
Asynchronous Double Speed mode (U2Xn = 1)	$BAUD = \frac{f_{OSC}}{8(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{8BAUD} - 1$
Synchronous Master mode	$BAUD = \frac{f_{OSC}}{2(UBRRn+1)}$	$UBRRn = \frac{f_{OSC}}{2BAUD} - 1$

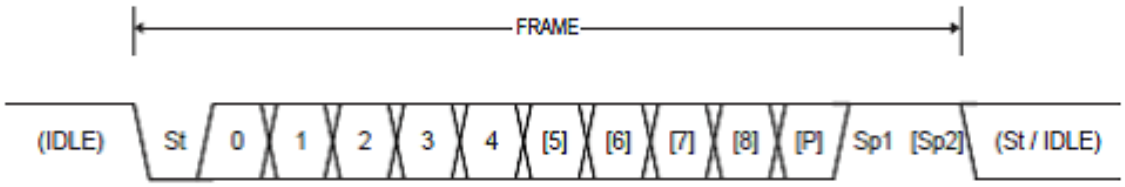
BAUD	Baud rate (in bits per second, bps)
f_{OSC}	System Oscillator clock frequency
UBRRn	Contents of the UBRRHn and UBRRLn Registers, (0-4095)

Abbildung 3.7: Baudratenberechnung[Atm10a, Seite 167, Table 17-1]

Paketformate:

Ein Paket ist definiert als ein Zeichen mit Datenbits und Synchronisationsbits (Start- und Stopbits). Es ist auch noch möglich ein Paritätsbit mit dem Paket zu senden, um Fehler zu erkennen[Atm10a, vgl. Seite 168]. Der USART erkennt Pakete mit:

- 1 Startbit
- 5, 6, 7, 8 oder 9 Datenbits
- Kein, gleiches oder ungleiches Paritätsbit
- 1 oder 2 Stopbits



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.

Abbildung 3.8: Paketformate[Atm10a, Seite 169, Figure 17-4]

USART Initialisierung:

Die Initialisierung des USART besteht aus dem setzen der Baudrate, dem Paketformat und dem aktivieren des Empfängers und Senders. Diese Initialisierung muss durchgeführt werden bevor der USART zur Kommunikation benutzt werden kann. Nachfolgend ein Beispiel zur Initialisierung in C mit 8 Datenbits, 2 Stopbits und keinem Paritätsbit[Atm10a, vgl. Seite 170].

C Code Example(1)

```
void USART_Init( unsigned int baud )
{
  /* Set baud rate */
  UBRRHn = (unsigned char)(baud>>8);
  UBRRLn = (unsigned char)baud;
  /* Enable receiver and transmitter */
  UCSRnB = (1<<RXENn)|(1<<TXENn);
  /* Set frame format: 8data, 2stop bit */
  UCSRnC = (1<<USBSn)|(3<<UCSZn0);
}
```

Abbildung 3.9: C-Beispiel zur USART Initialsierung[Atm10a, Seite 170]

Datenübertragung:

Die Datenübertragung via USART erfolgt durch das Setzen des Transmit Enable (TXEN) Bits im UCSRnB Register. Ein Datentransfer wird durch das Laden der Daten in den Transmit Buffer initialisiert. Die Daten werden von dort dann in den Shift Register geladen, wenn dieser bereit ist zu senden. Nachfolgend ein Beispiel in C zum Senden eines Zeichens für Pakete mit 5 bis 8 Datenbits[Atm10a, vgl. Seite 171].

C Code Example(1)

```
void USART_Transmit( unsigned char data )
{
  /* Wait for empty transmit buffer */
  while ( !( UCSRnA & (1<<UDREn)) )
       ;
  /* Put data into buffer, sends the data */
  UDRn = data;
}
```

Abbildung 3.10: C-Beispiel zur USART Datenübertragung[Atm10a, Seite 171]

3.2 PC-Programm

3.2.1 Funktion

Die Sensor Array Application Software wurde mit C# und Microsoft® Visual Studio 2010 erstellt und soll die Daten die vom Comcontroller zum PC gesendet werden empfangen, auswerten und Verbindungen von Handschuh und Objekt übersichtlich darstellen.

Die Sensor Array Application ist eine Windows Forms Anwendung. Der Benutzer kann verschiedene COM-Ports in einer ComboBox auswählen und sich mit dem Controller verbinden. Die Baudrate und das Paketformat sind schon im Programm festgelegt. Die Baudrate beträgt 38400bps und ist die höchstmögliche Baudrate die vom ATmega 644-20 PU mit dem internen 8Mhz Takt fehlerfrei funktioniert. Das Paketformat besteht aus 8 Datenbits, 2 Stopbits und keinem Paritätsbit. In der Anwendung gibt es eine RichTextBox im linken Teil in der alle vom Comcontroller via RS232 gesendeten Daten angezeigt werden sollen. Des Weiteren soll mit jeweils einer CheckedListBox der Status der Kontakte des Handcontrollers und des Objektcontroller dargestellt werden. Außerdem soll in einer RichTextBox im rechts unten gelegenen Teil der Anwendung eine direkte Verbindung von 2 Kontakten als Text angezeigt werden, da bei mehr als einem Kontakt die Verbindung in den CheckedListBoxes nicht mehr ersichtlich wäre. Die Anzahl der Zyklen (Cycles), die der Controller schon durchlaufen hat und der Puffer (Buffer) der COM-Schnittstelle werden auch von dem Programm in zwei Textfeldern angezeigt.

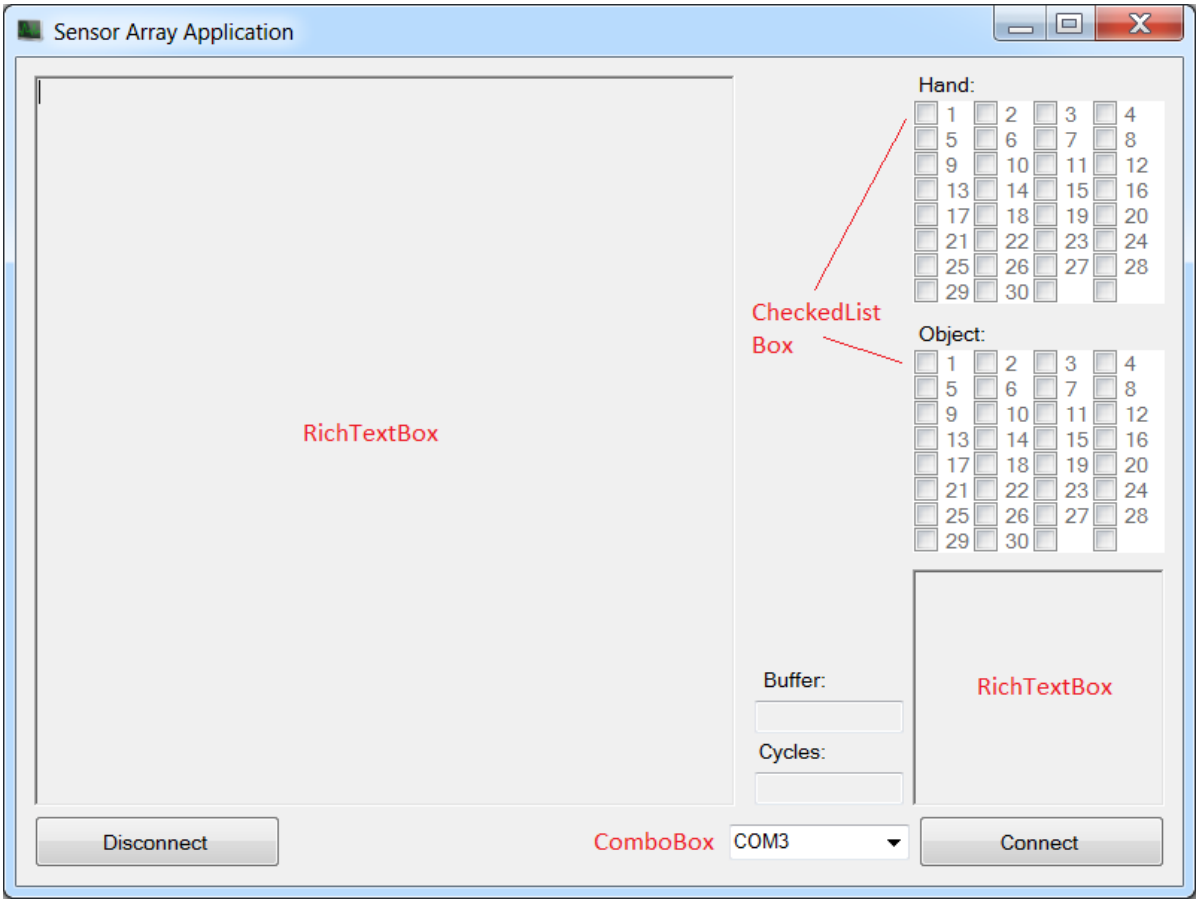


Abbildung 3.11: Screenshot der Sensor Array Application

3.2.2 Quellcode

Das Program startet mit dem Element Program.cs. Diese beinhaltet allerdings nur das Aufrufen der Fensterform in der das Programm läuft.

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Windows.Forms;
5
6  namespace WindowsFormsApplication2
7  {
8      static class Program
9      {
10         static void Main()
11         {
12             Application.EnableVisualStyles();
13             Application.SetCompatibleTextRenderingDefault(
                   false);
14             Application.Run(new Form1());
15         }
16     }
17 }
```

Das Element Form.cs enhält alle Informationen zum Aussehen der Fensteranwendung und alle Methoden für ihre Funktionalität. Der Konstruktor der Windows-Form ruft die Methode InitializeComponent auf in der alle Elemente der Windows-Form, wie z.B. Buttons, Textboxen, Textfelder, Labels und Checkboxen hinzugefügt werden.

```
1          public Form1()
2          {
3              InitializeComponent();
4          }
```

Es wird sowohl das Aussehen der Elemente, deren Position im Fenster, teilweise auch deren Inhalt und alles andere was mit dem Layout zu tun hat in dieser Methode festgelegt. Außerdem werden dort auch je nach Bedarf Button-Events zugewiesen, die aufgerufen werden, falls diese z.B. gedrückt werden. Dem Connect-Button wird die Methode connect_Click zugewiesen, welche beim Drücken des Buttons aufgerufen wird. In dieser Methode wird die Methode serialInit aufgerufen welche eine Instanz der Klasse SerialPort erstellt und dieser Klasse eine COM-Port Nummer, Baudrate, Parität, Datenbits, Stopbits, Handshake und Timeouts zuweist. Die COM-Port Nummer wird durch den Inhalt der ComboBox in der Anwendung bestimmt. Es gibt eine

Auswahl von COM1 bis COM4. Außerdem kann auch ein String manuell eingegeben werden, falls noch weitere COM-Ports mit einer höheren Nummer gebraucht werden. Über die Instanz der Klasse SerialPort kann später mit dem COM-Port kommuniziert werden[ser11, vgl.]. In der Methode connect_Click wird dann das COM-Port geöffnet und der Text „Connected“ in der RichTextBox ausgegeben.

```
1          public void serialInit()
2          {
3              _serialPort = new SerialPort();
4
5              // Settings hardcoded
6              _serialPort.PortName = this.comboBox1.
                   SelectedItem.ToString();
7              _serialPort.BaudRate = 38400;
8              _serialPort.Parity = Parity.None;
9              _serialPort.DataBits = 8;
10             _serialPort.StopBits = StopBits.Two;
11             _serialPort.Handshake = Handshake.None;
12
13             // Set the read/write timeouts
14             _serialPort.ReadTimeout = 500;
15             _serialPort.WriteTimeout = 500;
16         }
```

Es wird dem Boolean _continue der Wert „true“ zugewiesen. Dieser ist für den Ablauf des BackgroundWorkers wichtig, welcher daraufhin gestartet wird. Der BackgroundWorker führt die Methode backgroundWorker1_DoWork in einem eigenen Thread aus damit die Benutzeroberfläche ungehindert weiterlaufen kann, wenn dieser ausgelastet ist. In der Methode backgroundWorker1_DoWork wird alles was mit den Nachrichten des COM-Ports zu tun hat verwaltet[bac11, vgl.].

```
1          private void connect_Click(object sender, System.
               EventArgs e)
2          {
3              try
4              {
5                  serialInit();
6                  _serialPort.Open();
7                  richTextBox1.AppendText("Connected\n\n");
8              }
9              catch { }
10             _continue = true;
11             try
12             {
13                 this.backgroundWorker1.RunWorkerAsync();
14             }
15             catch { }
```

```
16          }
```

In der Methode backgroundWorker1_DoWork werden zuerst zwei Strings (text und pin_text) initialisiert. Dann wird eine while-Schleife mit dem Parameter _continue, welchem vorher der Wert „true“ zugewiesen wurde, gestartet. In der while-Schleife wird am Anfang dem String text ein leerer String zugewiesen. Daraufhin wird in einem try-Block geprüft, ob der Buffer des COM-Ports größer als 128 Byte ist und gelöscht, falls dies der Fall ist. Es wird dann eine Zeile des COM-Ports in den String text geschrieben. Danach wird das erste Zeichen des Strings text auf unterschiedliche Fälle geprüft.

Falls der String mit „S“ beginnt, soll in mit der Methode SetHandCheckBox der entsprechende Pin, welcher vom Handcontroller auf High (1) gesetzt wurde in der CheckedListBox des Handcontrollers markiert werden. Nach dem „S“ steht ein zweistelliger Integerwert, welcher den auf High (1) gesetzten Pin bezeichnet. In den String pin_text wird „HandPinXX→ObjectPin”geschrieben, wobei XX für den entsprechenden Pin steht.

Falls der String mit „A“ beginnt, was für den Status von Port A des Objektcontrollers steht, wird ein neuer String s erstellt. Nach dem „A“ steht ein zweistelliger Hexadezimalwert, der den Portstatus beschreibt. Die Methode hexToBin wandelt den vom Objektcontroller gesendeten zweistelligen Hexadezimalwert in ein 8bit Binärwort als String um. Dieses Binärwort wird dann im String s gespeichert und als zweiter Parameter einmal an die Methode SetObjectCheckBox(1, s) weitergegeben, um die jeweiligen Pins in der CheckedListBox zu markieren, welche einen High Status (1) haben. Der String s wird dann ein weiteres Mal übergeben an die Methode BinToPins(1, s), welche das Binärwort in die dazugehörigen Pin-Nummern umwandelt die den Status High (1) haben. Diese Pin-Nummern werden dann an den String pin_text angehängt.

Es folgen dann drei weitere Fälle bei denen der String entweder mit „B“, „C“ oder „D“ beginnt. Diese sind mit derselben Struktur aufgebaut, wie der vorherige Fall mit „A“. Der einzige Unterschied ist, dass je nach dem ersten Zeichen, welches für den Port steht, die Methoden SetObjectCheckBox und BinToPins mit einem anderen ersten Parameter aufgerufen werden. Beim Fall „A“ wurde eine 1 übergeben, bei „B“ wird nun eine 2, bei „C“ eine 3 und bei „D“ eine 4 übergeben, um zwischen den Ports unterscheiden zu können. Für den Fall „B“ würden die Methoden dann als SetObjectCheckBox(2, s) und BinToPins(2, s) aufgerufen werden.

Nun folgt noch ein weiterer Fall, falls der String text mit „N“ beginnt. „N“ steht für die Zyklen und gibt an, das ein kompletter Zyklus durchlaufen wurde. Nach dem „N“ wird dann die Nummer des Zyklus gesendet z.B. „N123“ für den 123ten Zyklus. Zuerst wird der Zyklus in der Anwendung mit der Methode SetCycle in ein Textfeld gesetzt. Dann werden die beiden CheckedListBoxen und die RichTextBox für die Pinverbindungen zurückgesetzt und die Größe des Buffers in ein Textfeld geschrieben. Das ist notwendig, da sonst noch Verbindungen angezeigt werden würde, auch wenn diese in der realen Umgebung nicht mehr bestehen. Eine Verbindung wird daher immer genau einen Zyklus lang angezeigt, es sei denn sie besteht noch im nächsten Zyklus.

In der while-Schleife wird nun noch der String text in die RichTextBox geschrieben, falls er nicht leer ist und der String pin_text in die RichTextBox für die Pinverbindungen, falls diese vorhanden ist. Das ist der Fall, falls der String pin_text länger als 20 Zeichen ist.

Daraufhin beginnt die while-Schleife wieder von vorne, vorausgesetzt der Boolean _continue hat noch den Wert „true“.

```
1          /*
2           * Background Worker fuer COM-Nachrichten
3           * */
4          private void backgroundWorker1_DoWork(object sender,
               DoWorkEventArgs e)
5          {
6              String text;
7              String pin_text = "";
8
9              while (_continue)
10             {
11                 text = "";
12                 //COM-Schnittstelle: Zeile auslesen und evtl
                       Buffer loeschen
13                 try
14                 {
15                     if (_serialPort.BytesToRead > 128)
16                     {
17                         _serialPort.DiscardInBuffer();
18                     }
19                     text = _serialPort.ReadLine();
20                 }
21                 catch
22                 {
23                 }
24                     //High Pins
25                     if (text.StartsWith("S"))
26                     {
27                         SetHandCheckBox(Convert.ToInt32(text.
                               Substring(1)));
28                         pin_text = "HandPin" + text.Substring
                               (1) + "->ObjectPin";
29                     }
30                     //Port A
31                     if (text.StartsWith("A"))
32                     {
33                         String s = hexToBin(text.Substring(1)
                               );
34                         SetObjectCheckBox(1, s);
35                         pin_text += BinToPins(1, s);
36                     }
37                     //Port B
38                     if (text.StartsWith("B"))
```

```
39                      {
40                          String s = hexToBin(text.Substring(1)
                                );
41                          SetObjectCheckBox(2, s);
42                          pin_text += BinToPins(2, s);
43                      }
44                      //Port C
45                      if (text.StartsWith("C"))
46                      {
47                          String s = hexToBin(text.Substring(1)
                                );
48                          SetObjectCheckBox(3, s);
49                          pin_text += BinToPins(3, s);
50                      }
51                      //Port D
52                      if (text.StartsWith("D"))
53                      {
54                          String s = hexToBin(text.Substring(1)
                                );
55                          SetObjectCheckBox(4, s);
56                          pin_text += BinToPins(4, s);
57                      }
58                      //Cycles und Checkbox reset bis zum
                            naechsten Cycle
59                      if (text.StartsWith("N"))
60                      {
61                          SetCycle(text.Substring(1));
62                          SetObjectCheckBox(0, "");
63                          SetHandCheckBox(0);
64                          SetPinText("clear");
65                          try
66                          {
67                              SetBuffer(_serialPort.BytesToRead
                                    .ToString());
68                          }
69                          catch
70                          {
71                          }
72                      }
73                      //Text setzen
74                      if (text.Length > 0)
75                      {
76                          SetText(text);
77                      }
78                      //Verbunden Pins Text setzen
```

```
79                      if (pin_text.Length > 20)
80                      {
81                          SetPinText(pin_text);
82                      }
83
84              }
85          }
```

Die Methode SetHandCheckBox ist notwendig, um Verbindungen der Kontakte des Handcontrollers darzustellen. Als Parameter wird ein Integer zwischen 1 und 30 übergeben. In der Methode wird zuerst geprüft ob ein Invoke notwendig ist. Da der BackgroundWorker1 in einem anderen Thread abläuft, als das Windows-Forms Steuerelement (in unserem Fall die CheckedListBox), werden hierdurch sichere threadübergreifende Zugriffe gewährleistet[thr11, vgl.]. Wenn das nun der Fall ist, wird im else-Block der eigentliche Teil der Methode ausgeführt. Falls 0 als Parameter übergeben wurde, soll die komplette CheckedListBox zurückgesetzt werden. Ansonsten wird durch alle Checkboxen iteriert und es soll die Checkbox mit der gleichen Bezeichung, wie dem übergebenem Integer, markiert werden. Wenn also beispielsweise eine 1 als Integer übergeben wird, soll die Checkbox mit der Bezeichnung '1' markiert werden.

```
1           delegate void SetHandCheckBoxCallback(int i);
2
3           private void SetHandCheckBox(int pin)
4           {
5               if (this.hand_checkbox.InvokeRequired)
6               {
7                   SetHandCheckBoxCallback d = new
                        SetHandCheckBoxCallback(SetHandCheckBox);
8                   this.Invoke(d, new object[] { pin });
9               }
10              else
11              {
12                  if (pin == 0)
13                  {
14                      for (int j = 0; j < hand_checkbox.Items.
                            Count; j++)
15                      {
16                          hand_checkbox.SetItemChecked(j, false
                                );
17                      }
18                      return;
19                  }
20                  for (int i = 0; i < hand_checkbox.Items.Count
                        ; i++)
21                  {
22                      if (hand_checkbox.Items[i].ToString().
                            Equals(pin + ""))
```

```
23                      {
24                          hand_checkbox.SetItemChecked(i, true)
                                ;
25                      }
26                  }
27              }
28          }
```

Die Methode SetObjectCheckBox ist für das Darstellen der Verbindungen mit den Kontakten des Objektes notwendig. Es werden der Methode zwei Parameter übergeben, ein Integer port, welcher den Port bezeichnet und ein String status, welcher den Status des Ports als 8bit Binärwort darstellt. 1 wird beispielsweise für Port A, 2 für Port B etc. übergeben. Im else-Block findet wieder die eigentliche Aufgabe der Methode statt. Falls 0 als Wert für den Port übergeben wird, soll die CheckedListBox zurückgesetzt werden. Andernfalls wird je nach Port der String status einer ausgelagerten Methode übergeben. In dieser werden je nach Status, alle Checkboxen die einen Pin darstellen, der den Status High (1) hat, markiert.

```
1           delegate void SetObjectCheckBoxCallback(int port,
               String status);
2
3           private void SetObjectCheckBox(int port, String
               status)
4           {
5               if (this.object_checkbox.InvokeRequired)
6               {
7                   SetObjectCheckBoxCallback d = new
                       SetObjectCheckBoxCallback(
                       SetObjectCheckBox);
8                   this.Invoke(d, new object[] { port, status })
                       ;
9               }
10              else
11              {
12                  if (port == 0)
13                  {
14                      for (int i = 0; i < object_checkbox.Items
                           .Count; i++)
15                      {
16                          object_checkbox.SetItemChecked(i,
                               false);
17                      }
18                  }
19
20                  if (port == 1)
21                  {
22                      CheckBoxPortA(status);
```

```
23                  }
24                  if (port == 2)
25                  {
26                      CheckBoxPortB(status);
27                  }
28                  if (port == 3)
29                  {
30                      CheckBoxPortC(status);
31                  }
32                  if (port == 4)
33                  {
34                      CheckBoxPortD(status);
35                  }
36              }
37          }
```

Weitere Methoden, die vom BackgorundWorker aufgerufen werden, sind SetCycle und SetBuffer. In diesen wird der Zyklus bzw. die größe des COM-Port Buffers in ein Textfeld geschrieben. Beide Methoden bekommen einen String als Parameter übergeben, welcher den zu setzenden Text enthält.

```
1           delegate void SetCycleCallback(String s);
2
3           private void SetCycle(String s)
4           {
5               if (this.cycle_textbox.InvokeRequired)
6               {
7                   SetCycleCallback d = new SetCycleCallback(
                        SetCycle);
8                   this.Invoke(d, new object[] { s });
9               }
10              else
11              {
12                  this.cycle_textbox.Text = s;
13              }
14          }
15
16          delegate void SetBufferCallback(String s);
17
18          private void SetBuffer(String s)
19          {
20              if (this.buffer_textbox.InvokeRequired)
21              {
22                  SetBufferCallback d = new SetBufferCallback(
                        SetBuffer);
23                  this.Invoke(d, new object[] { s });
```

```
24              }
25              else
26              {
27                  this.buffer_textbox.Text = s;
28              }
29          }
```

Die Methoden SetText und SetPinText sind für das Setzen des Textes in ihren zugehörigen RichTextBoxen zuständig. Beide Methoden bekommen einen String als Parameter übergeben. Die Methode SetText prüft vor dem Setzen des Textes, ob die RichTextBox mehr als zwei Millarden Zeichen enthält und löscht diese, wenn das der Fall ist. Danach wird der übergebene String mit einem Zeilenumbruch der RichTextBox angehängt. Die Methode SetPinText prüft vor dem Setzen des Textes, ob der zu setzende Text schon vorhanden ist. Das macht natürlich Sinn, da hier die Verbindungen von zwei Kontakten angezeigt werden sollen und eine Verbindung nur einmal angezeigt werden soll. Außerdem kann der Text gelöscht werden, falls man der Methode SetPinText den String 'clear' als Parameter übergibt.

```
1           delegate void SetTextCallback(string text);
2
3           private void SetText(string text)
4           {
5               if (this.richTextBox1.InvokeRequired)
6               {
7                   SetTextCallback d = new SetTextCallback(
                        SetText);
8                   this.Invoke(d, new object[] { text });
9               }
10              else
11              {
12                  if (this.richTextBox1.TextLength >
                        2000000000)
13                  {
14                      this.richTextBox1.Clear();
15                  }
16                  this.richTextBox1.AppendText(text);
17                  this.richTextBox1.AppendText("\n");
18              }
19          }
20
21          delegate void SetPinTextCallback(string text);
22
23          private void SetPinText(string text)
24          {
25              if (this.pin_textfield.InvokeRequired)
26              {
```

```
27                  SetPinTextCallback d = new SetPinTextCallback
                        (SetPinText);
28                  this.Invoke(d, new object[] { text });
29              }
30              else
31              {
32                  if (text.Equals("clear"))
33                  {
34                      this.pin_textfield.Clear();
35                      return;
36                  }
37                  bool a = true;
38                  foreach (String s in pin_textfield.Lines)
39                  {
40                      if (s.Equals(text))
41                      {
42                          a = false;
43                      }
44                  }
45
46                  if (a)
47                  {
48                      this.pin_textfield.AppendText(text);
49                      this.pin_textfield.AppendText("\n");
50                  }
51              }
52          }
```

Hardware

4

Dieses Kapitel behandelt den technischen Teil des taktilen Sensorarrays. Nach einem kurzen Überblick der wichtigsten verwendeten Komponenten, soll eine Einführung in die verwendeten Schnittstellen gegeben werden. Der dritte Teil des Kapitels befasst sich mit den elektrischen Schaltungen, welche die Grundlage des hardwaremäßigen Systems darstellen. Es wird dabei auch darauf eingegangen, wie die Umsetzung der Schaltung realisiert werden kann. Der letzte Teil befasst sich mit den Möglichkeiten eine Kontaktmatrix, die Sensorfläche des taktilen Sensorarrays, zu konstruieren. Zum Schluss folgt noch ein technischer Überblick des entwickelten Prototypen.

4.1 Komponenten

Die Auswahl der Komponenten für einen prototypischen Schaltungsaufbau zur Realisierung des taktilen Sensorfeldes, ist unter verschiedenen Gesichtspunkten zu treffen. Zunächst sollten die Komponenten eine möglichst hohe Flexibilität aufweisen, so dass es möglich ist, das Konzept zu erweitern oder zu verändern, ohne alle Bauteile des bisherigen Prototypen verwerfen zu müssen, zudem kann mit solchen Komponenten auch auf der Basis des Prototypen ggf. weitere Entwicklungen gemacht werden. Des weiteren müssen Komponenten zur Entwicklung eines Prototypen natürlich die Bedingungen des gesetzten Konzeptes erfüllen.

4.1.1 AVR® STK500

Das Entwicklungboard AVR® STK500 von Atmel® bietet die Grundlage für die ersten Versuchsaufbauten. Der Haupteinsatzgrund für das AVR® STK500 ist die Möglichkeit die geschriebenen Programme auf eine Vielzahl unterschiedlicher Mikrocontroller zu schreiben oder zu brennen. Es besteht die Möglichkeit die Ein- und Ausgänge der Mikrocontroller zu belegen und somit flexibel zu nutzen. Die weiteren Vorteile liegen in der schon stabilisierten 5V-Versorgungsspannung, der Möglichkeit externe Quarze zur Taktung der Mikrocontroller zu benutzen, um einen präziseren Takt zu erhalten und die im Entwicklungsboard integrierte RS-232 Schnittstelle zur Kommunikation mit einem Rechnersystem [Atm00, vgl.].

4.1.2 Mikrocontroller

Der Mikrocontroller bildet das Kernstück des gesamten prototypischen Sensorsystems. In diesem Sensorsystem findet der ATmega 644-20 PU aus der AVR® RISC-Familie von Atmel® seinen Einsatz. Für die ersten Versuchsaufbauten ist die Größe des Mikrocontrollers, DIL-40, von Vorteil. Er besitzt 32 Ein- bzw. Ausgänge in 4 Ports angeordnet, die nicht nur je Port sondern auch einzeln als Ein- oder Ausgang definiert genutzt werden können. Der interne Quarz mit einem Takt von 8Mhz macht den Einsatz von externen Quarzen zur Taktung überflüssig und erleichtert die Versuchsaufbauten [Atm10a, vgl.]. Der ATmega 644-20 PU kommt mit einer integrierten USART-Schnittstelle, die sehr flexibel auf Übertragungseigenschaften eingestellt werden kann [Atm10a, vgl. Seite 164 ff.]. Des weiteren ist der 2-wire Serial Interface (TWI oder auch als I^2C bekannt) in der Schaltung des Mikrocontrollers integriert, so dass eine Kommunikation zwischen Mikrocontrollern ohne Zusatzmodule möglich ist [Atm10a, vgl. Seite 200].

In Ausblick auf einen Ausbau der Schaltung, d.h. dem Wegfall der Entwicklungsumgebung für die Schaltung und dem daraus resultierenden autonomen Sensorsystems, führt der ATmega 644-20 PU auch schon wichtige Eigenschaften mit sich. Der interne Quarz zur Taktung von 8Mhz ermöglicht eine hohe Datenübertragungsrate für die USART-Schnittstelle [Atm10a, vgl. Seite 166 ff.], ohne einen externen Quarz für die Taktung verwenden zu müssen. Zudem ist der ATmega 644-20 PU auch als ATmega 644-20 AU erhältlich, der in seiner Funktionalität keinen Unterschied aufweist, jedoch in einem TQFP (Thin Quad Flat Package) angeboten wird und mit seinen Ausmaßen von ca. 1cm für hochintegrierte Schaltungen genutzt werden kann [Atm10a, vgl. Seite 362].

4.1.3 MAX232

Der MAX232 Baustein von Texas Instruments ist ein Treiber für die serielle RS-232 Kommunikation. Die RS-232 Bezeichnung ist die im europäischen Raum übliche, während der aktuelle amerikanische Standard von 1997 als (ANSI EIA/) TIA-232-F bezeichnet wird [wik11, vgl.]. Mit einer Baugröße von DIL-16 eignet sich auch dieser Baustein für die ersten Versuchsaufbauten. Er arbeitet mit einer Spannungsversorgung von 5V (stabilisiert) und kann die TTL/CMOS-Signale von 5V mit Hilfe von 1μF Kondensatoren nach dem Spannungspumpenprinzip auf ein für TIA/EIA-232 benötigtes Spannungsniveau von $\pm$30V anheben. Als Empfänger eingesetzt wird das Spannungsniveau der TIA/EIA-232 Eingangssignale von $\pm$30V auf ein TTL/CMOS spezifischen Wert von 5V abgesenkt. Dabei unterstützt der Baustein eine Datenrate von bis 120kBd [Tex04, vgl.].

Wie auch bei der Wahl des Mikrocontroller ist der Max232 Baustein in kleinerer Form erhältlich, so dass sich auch hier, in Ausblick auf einen Ausbau der Schaltung, hochintegrierte Schaltungen mit dem MAX232 Baustein noch realisieren lassen.

4.2 Schnittstellen

Das Konzept für das Sensorsystem sieht einen modularen Aufbau des Systems vor. Es soll ermöglicht werden die Sensorfläche zu vergrößern in dem zusätzliche Sensorflächen zu den gegeben Sensorflächen zugeschaltet werden können. Das allerdings erfordert nicht nur den modularen Aufbau sondern auch eine geeignete Kommunikation zwischen den einzelnen Modulen des System. Letztendlich soll das System die von den Sensoren erfassten Daten auch an ein Rechnersystem weiterleiten.

Jede Kommunikation, ob nun zwischen den Mikrocontrollern selbst oder mit einem Rechnersystem, erfordert eine geeignete Schnittstelle zur Übertragung der Daten. Diese Schnittstellen sind bestenfalls genormt und können so mit genormten Bauteilen in einer Schaltung umgesetzt werden oder die Schnittstellen müssen noch konstruiert werden um den Bedürfnissen der Schaltung zu genügen.

4.2.1 TWI/PWR

2-wire Serial Interface. Der TWI-Bus, bereitgestellt durch den verwendeten Mikrocontroller ATmega 644-20 PU, arbeitet auf einem Bussystem, das eine High-Level Spannung führt. Diese Spannung wird durch 2,4kΩ Pull-Up Widerstände erzeugt. Die Treiber aller TWI-Bus anhängenden Komponenten sind mit open-drain bzw. open-collector konzipiert. Das ermöglicht auf dem Bus das Arbeiten mit logischen UND Verknüpfungen [Atm10a, vgl. Seite 201].

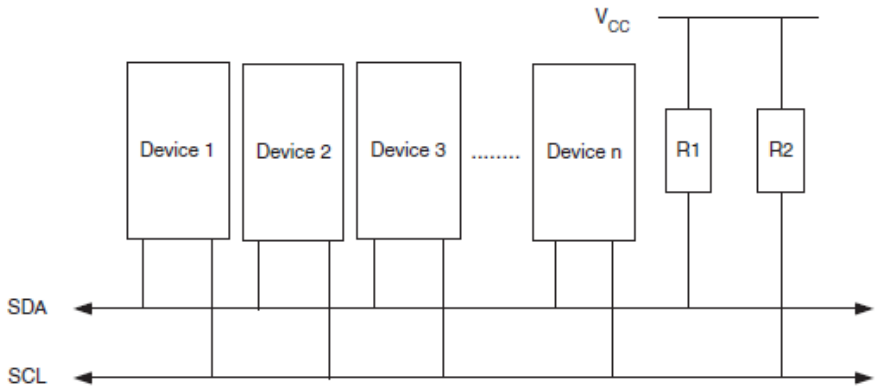


Abbildung 4.1: Schematischer Aufbau des TWI-Bus mit allen benötigten Komponenten [Atm10a, Seite 200].

Die Abbildung 4.1 macht deutlich, dass nicht nur die Datenleitung (Sda) des Bussystems, sondern auch die Taktleitung (Scl) auf ein High-Pegel Spannungsniveau gezogen wird. Dies erfordert natürlich, dass alle Mikrocontroller, die an den TWI-Bus angeschlossen werden, mit einer Spannung versorgt werden, um die Ausgänge zu den Busleitungen auf ein High-Pegel zu

bringen. Sobald ein Mikrocontroller dazu nicht in der Lage ist, würde die Kommunikation auf dem TWI-Bussystem gestört werden. Der fehlerhafte Mikrocontroller kann jedoch, dank des modularen Aufbaus des Sensorsystem, ausfindig gemacht werden.

Das TWI-Bussystem erlaubt eine Anzahl von bis zu 128 Kommunikationsteilnehmern und bietet somit eine ausreichende Erweiterungsmöglichkeit. Für das Sensorsystem bedeutet dies 126 mögliche Sensorflächen, da eine TWI-Busadresse für den Comcontroller benötigt wird und die Adresse 000 als Broadcast-Adresse reserviert ist [Atm10a, vgl. Seite 202].

Stromversorgung. Jedes der einzelnen Sensormodule muss auch mit Strom versorgt werden, daher werden zwei weitere Leitungen für +Vcc und Gnd benötigt. Jedes einzelne Sensormodul mit einer eigenen Stromversorgung zu belegen, scheint übertrieben, da somit für jedes einzelne Sensormodul eine eigene Spannungsstabilisierung benötigen würde. Auch die Möglichkeit für jedes einzelne Sensormodul eine einzelne Stromversorgung von der Hauptplatine des Comcontrollers zum jeweiligen Sensormodul zu legen, scheint zu aufwendig.

Schnittstellenbeschreibung. Aus diesem Grund und auch um die Modularität des Systems aufrecht zu erhalten, ist ein “neuer” Bus konstruiert worden. Dieser Bus ist mit vier Leitungen ausgestattet.

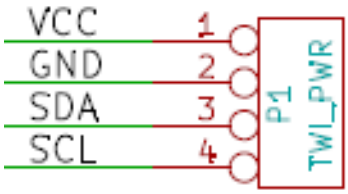


Abbildung 4.2: Schematischen Schaltplandarstellung der TWI/PWR-Schnittstelle.

Die Leitungen des TWI-Bus, also Sda und Scl, sind enthalten, aber auch die Stromversorgung, +Vcc und Gnd. Die Abbildung 4.2 zeichnet die Schnittstelle und beschreibt die Belegung des Bus. Es lässt auch schnell erkennen, dass der Vorteil dieser Konstruktion darin besteht, auf kein Verpolschutz achten zu müssen. Sollte der Stecker vertauscht (gedreht) werden, dann liegt +Vcc am Scl Eingang und Gnd auf Sda. Da aber der TWI-Bus standardmäßig eine High-Level Spannung auf dem Bus führt, gibt es keine Auswirkung für das +Vcc auf Scl und auch der Gnd auf Sda verursacht keinen Kurzschluss, sondern einfach nur ein Signal, das als nicht sinnvoll interpretiert wird.

Dieser TWI/PWR-Bus stellt zudem sicher, dass die Modularität des Gesamtsystems erhalten bleibt. Der TWI/PWR-Bus kann von Sensormodul zu Sensormodul weiter geschleift werden, da der TWI-Bus keine Terminatoren am Leitungsende des Bus benötigt und so mit offenen Leitungen am Ende gearbeitet werden kann. Jedes Sensormodul ist mit zwei TWI/PWR-Bus Schnittstellen ausgestattet und ermöglicht das Hintereinanderschalten von theoretisch bis zu 126 Sensormodule.

4.2.2 RS-232

Die RS-232 Schnittstelle soll zur Kommunikation mit einem Rechnersystem dienen. Der Vorteil in der Benutzung dieser Schnittstelle liegt darin begründet, dass die Kommunikation bereits von dem Mikrocontroller ATmega 644-20 PU unterstützt wird [Atm10a, vgl. Seite 164 ff.] und auch die Schnittstelle selbst seit langen Jahren genormt und etabliert ist.

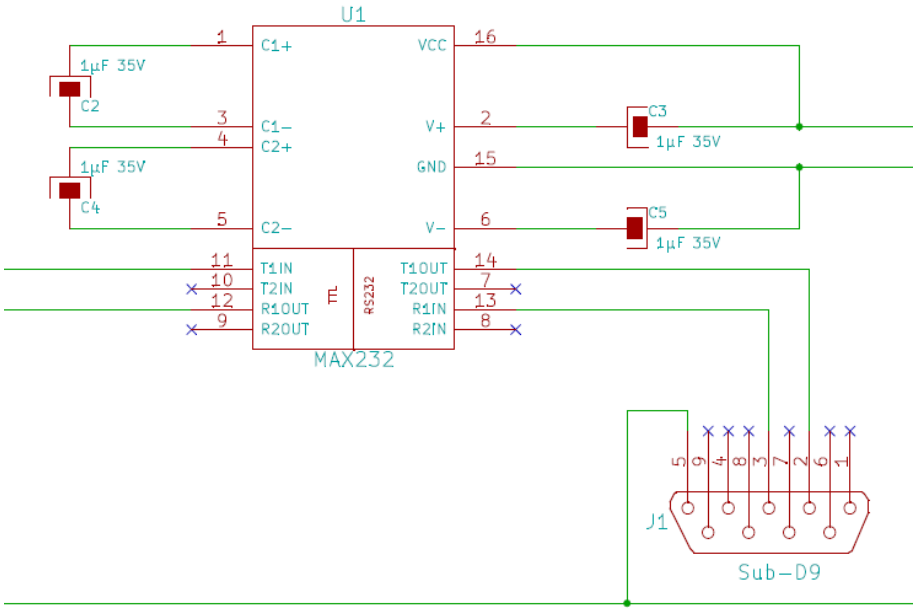


Abbildung 4.3: Schematische Schaltplandarstellung der RS-232 Schnittstelle mit Treiberbaustein MAX232 von Texas Instruments.

Jedoch liefert der Mikrocontroller nur ein TTL/CMOS-Signal von 0V bis 5V, während die RS-232 Spannungen von -15V bis -3V und von +3V bis +15V benötigt. Um die Spannungsniveaus anzugleichen, wird der Treiberbaustein MAX232 von Texas Instruments eingesetzt (siehe dazu Kapitel 4.1.3). Die RS-232 Schnittstelle sieht mehr als reine Datenübertragung vor und enthält mehrere Steuerleitungen, um z.B. einen Hardware-Handshake durchzuführen [wik11, vgl.]. Diese Steuerleitungen sind für das Sensorsystem nicht notwendig, da die Daten nur in eine Richtung vom Mikrocontroller zum Rechnersystem übertragen werden sollen, daher werden sie nicht berücksichtigt. In Abbildung 4.3 ist der realisierte Aufbau als Schaltplanausschnitt zu sehen. Der MAX232 Baustein wird mit vier 1μF Kondensatoren betrieben. Die Leitung TxD (Pin PD1 am ATmega 644-20 PU) liegt an T1IN (Pin 11 am MAX232). Die Leitung RxD (Pin PD0 am ATmega 644-20 PU) liegt an R1OUT (Pin 12 am MAX232). Entsprechend der Beschaltung auf der TTL/CMOS-Seite, ist die Beschaltung auf der TIA/EIA-232-Seite an Pin 13 und Pin 14. Die Leitungen werden an ein Steckverbinder vom Typ D-Sub 9 female angeschlossen, welche die hardwaremäßge Schnittstelle des RS-232 bildet.

4.3 Elektrische Schaltung

Die Gesamtschaltung des Systems ist das Resultat einer Schritt für Schritt Entwicklung. Zunächst ist der Versuchsaufbau mit Hilfe des AVR®STK500 Entwicklungsboardes (siehe dazu Kapitel 4.1.1) und einer Erweiterungs-Steckplatine mit einem Rastermaß von 2,54mm realisiert worden. Danach folgte die Entnahme der Bausteine aus den Entwicklungsboards um eine autonome und modulare Schaltung auf separaten Platinen zu realisieren. Die Gesamtschaltung des Sensorsystems besteht aus drei einzelnen Schaltungen. Die Schaltung für den Comcontroller, für den Handcontroller und für den Objektcontroller. Die kompletten elektrischen Schaltpläne sind im Anhang einzusehen.

4.3.1 Comcontroller-Schaltung

Die Comcontroller-Schaltung ist das Mainboard des Gesamtsystems und der Ankerpunkt für die Verteilung der Hand- bzw. der Objektcontroller. Die erste Aufgabe, die die Comcontroller-Schaltung zu erfüllen hat ist, die stabilisierte Spannungsversorgung von 5V für das Gesamtsystem sicher zu stellen.

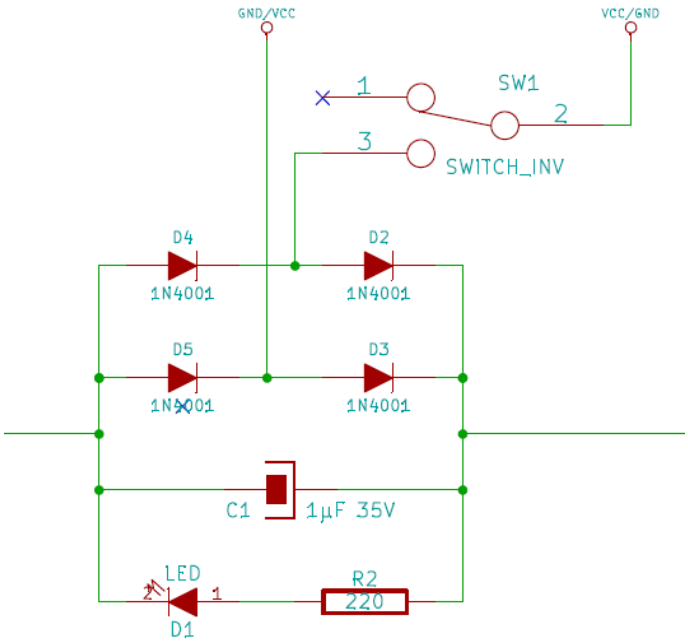


Abbildung 4.4: Stabilisierung und Glättung der Versorgungsspannung.

Um einen Verpolschutz sicher zu stellen wird, wie in Abbildung 4.4 gezeigt, die Versorgungsspannung über eine Gleichrichterbrückenschaltung geführt. Damit wird die Polung des Anschlusssteckers der Spannungsversorgung irrelevant. Die Stabilisierung und Glättung erfolgt

über einen 1μF Kondensator der parallel zur Versorgungsspannung geschaltet ist und somit etwaige Spannungsspitzen aus dem System filtern kann. Für die Brückenschaltung wurden die Siliziumdioden 1N4001 verwendet. Diese haben eine Arbeitsspannung bis zu 50V und können bis zu einem Strom von 1A arbeiten [Dio11, vgl. Seite 2]. Zusätzlich ist ein Schalter zum ein- und ausschalten des Systems und eine Diode zur Betriebsanzeige eingebaut.

Das Herz der Comcontroller-Schalung ist der ATmega 644-20 PU. Seine Aufgabe besteht unter anderem darin den TWI-Bus zu kontrollieren. Er ist als Master des TWI-Bus eingerichtet. Somit kann er die Daten der Hand- und Ojektcontroller auf Anfrage erhalten und in geeigneter Weise umsetzten. Der TWI-Bus ist an den Pins 22 (Scl) und 23 (Sda) angeschlossen. Diese Pins sind für den TWI-Bus reserviert. Die aufbereiteten Daten (siehe dazu Kapitel 3.1.1) werden über die USART-Schnittstelle für die RS-232 Kommunikation ausgegeben. Die Pins 14 und 15 sind für RxD und TxD des USART reserviert. Die RS-232 Schnittstelle mit dem Treiberbaustein MAX232 von Texas Instruments befindet sich ebenfalls in der Comcontroller-Schaltung und somit auf dem Mainboard des Systems. Die detaillierte Beschreibung der Funktionsweise erfolgte bereits in Kapitel 4.2.2.

4.3.2 Handcontroller-Schaltung

Der Handcontroller ist die sendende Einheit des Systems. Von ihr werden die aktiven Signale an die Kontaktmatrix gegeben. Kern dieser Schaltung ist ein ATmega 644-20 PU, dessen Aufgabe darin besteht einzelne Ausgänge auf 5V (High-Pegel) zu setzten. Bei Anfrage des Comcontroller sind die angefragten Daten über den TWI-Bus zu senden.

Aus diesem Aufgabenbereich der Schaltung ergeben sich die Schnittstellen. Zum einen ist der TWI/PWR-Bus mit zwei Schnittstellen in der Schaltung vertreten, damit ein weiterer Handcontroller oder auch Objektcontroller zum System hinzugefügt werden kann. Die zweite Schnittstelle der Schaltung ist eine 30-polige Steckverbindung, die die Schnittstelle zu einer passenden Kontaktmatrix herstellt.

Die Besonderheit dieser Schaltung des Handcontrollers sind die 1N4148 Siliziumdioden. Sie dienen vor jedem Ein-/Ausgangs-Pin des ATmega 644-20 PU als Schutz gegen Eingangssignale. Mit einer größtmöglichen Sperrspannung von maximal 75V und einem Arbeitsstrom von 150mA [Dio02, vgl. Seite 2] bieten diese Universaldioden ausreichend Schutz. Ein unerwünschtes Eingangssignal kann entstehen wenn auf der Kontaktmatrix zwei Kontakte, darunter der High-Pegel führende, durch die Kontaktmatrix des Objektcontrollers überbrückt wird. Tatsächlich ergibt sich auf Grund der Programmierung keine Problematik durch eine unerwünschtes Signal an einem Ein-/Ausgabe-Pin. Die Dioden dienen zum physischen Schutz des Mikrocontrollers vor Kurzschlüssen und Überspannungen.

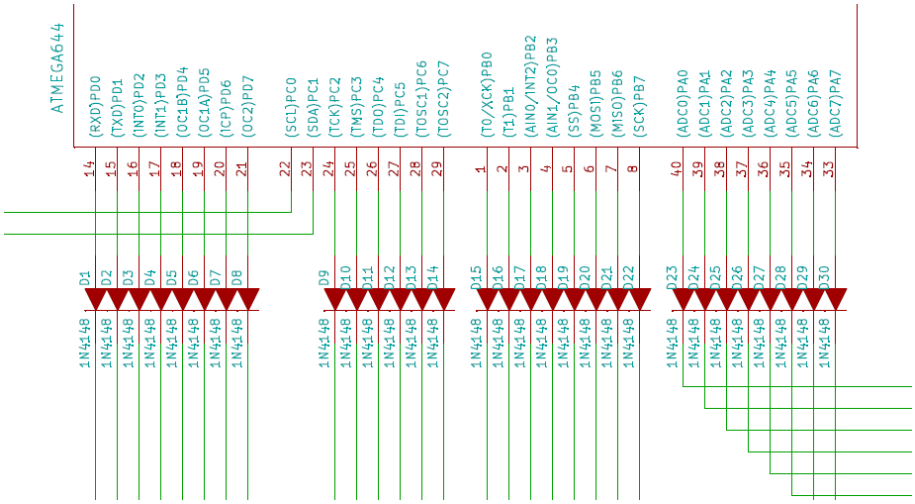


Abbildung 4.5: Zum Schutz vor Eingangssignalen und Kurzschlüssen dient die 1N4148.

4.3.3 Objektcontroller-Schaltung

Der Objektcontoller ist die empfangende Einheit des Systems. Von der Kontaktmatrix werden entsprechend die High-Pegel führenden Leitungen abgegriffen. Kern der Schaltung ist ein ATmega 644-20 PU, dessen Aufgabe darin besteht die High-Pegel an allen seinen Eingängen abzufragen. Bei Anfrage des Comcontrollers sind die angefragten Daten über den TWI-Bus zu senden.

Wie auch schon bei dem Handcontroller ergeben sich aus dem Aufgabengebiet der Schaltung die Schnittstellen. Zum einen sind wieder zwei TWI/PWR-Schnittstellen Teil der Schaltung, um die Erweiterbarkeit von zusätzlichen Objektcontrollern oder auch Handcontrollern zu gewährleisten. Eine 30-polige Steckverbindung dient als Schnittstelle zu einer passenden Kontaktmatrix.

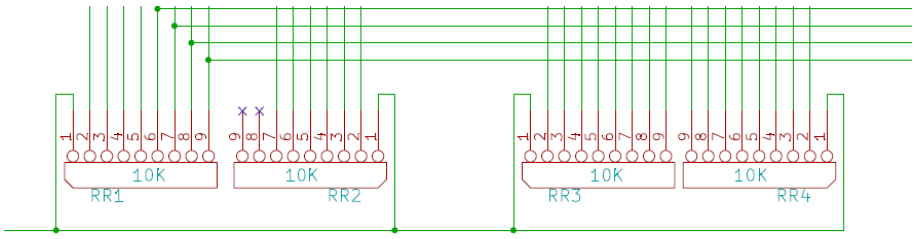


Abbildung 4.6: 10kΩ Pull-Down Widerstände gegen unerwünschte Eingangssignale.

Um sicher zu stellen, dass sich kein unerwünschtes Eingangssignal durch Spannungsspitzen auf den Leitungen einstellt, werden die Eingänge des Mikrocontrollers mit einem hochohmigen Widerstand, einem Pull-Down-Widerstand von 10kΩ, auf Masse gezogen. Bei dem ATmega 644-20 PU sind die Ein-/Ausgänge als open-drain bzw. open-collector konzipiert und sind daher anfällig für Störsignale. Der ATmega 644-20 PU besitzt intern Pull-Up-Widerstände, um einen

Ein-/Ausgang auf einen definierten High-Pegel setzen zu können [Atm10a, vgl. Seite 4 ff.]. Das Konzept des Objektcontroller sieht jedoch vor einen High-Pegel an einer Eingangsleitung auch als eine logische eins zu betrachten. Daher ist es notwendig die Pegel mit Hilfe eines Pull-Down-Widerstandes auf Masse zu ziehen, um einen definierten Low-Pegel an den Ein-/Ausgängen des Mikrocontrollers zu haben. Sollten die internen Pull-Up-Widerstände des Mikrocontrollers nicht deaktiviert worden sein, ist das kein Problem für die Schaltung an sich. Es entsteht kein Kurzschluss auf den Leitungen, da beide Widerstände sehr hochohmig sind. Es fließt jedoch ein Strom [Atm10a, vgl. Seite 4 ff.].

4.3.4 Versuchsaufbau

Das AVR® STK500 dient als erstes Entwicklungsboard für die Schaltung. Auf ihm untergebracht ist der Mikrocontroller für den Comcontroller. Damit ist das AVR® STK500 das, was als Mainboard des Gesamtsystems zu bezeichnen ist. Die Ein-/Ausgänge des Mikrocontrollers werden auf die im unteren Teil liegenden Pfostensteckverbinder geführt. Von dort aus wird die weitere Beschaltung vorgenommen.

Das in der Abbildung 4.7 gezeigte rot/schwarze Kabel ist die RxD/TxD-Leitung zur RS-232 Kommunikationsschnittstelle des Entwicklungsboards. Das blau/weiße Kabel ist die Verbindung vom Scl/Sda des Mikrocontrollers des TWI-Bus an die Erweiterungssteckplatine. Auf der Erweiterungssteckplatine sind die ersten Schaltungen für den Handcontroller und den Objektcontroller realisiert.

Auf der linken Seite in der Abbildung 4.8 ist die Handcontroller-Schaltung zu sehen. Für den Versuchsaufbau ist nur ein Port (Port A) des Mikrocontrollers mit den universellen Siliziumdioden 1N4148 bestückt. Die von da aus abgehenden blauen Kabel simulieren die Verbindung von zwei Kontaktmatrizen. Auf der rechten Seite der Abbildung 4.8 ist die Objektcontroller-Schaltung zu sehen. Die Pull-Down-Widerstände der Ein-/Ausgänge des Mikrocontrollers sind diesem Schaltungsaufbau mit 10kΩ Widerstandsarrays in Sternschaltung realisiert (in rot neben dem Mikrocontroller). Der TWI-Bus führt unterhalb der beiden Controller entlang und wird auf der rechten Seite mit den Pull-Up-Widerständen auf ein High-Pegel gezogen. Die in der Abbildung 4.8 erkennbaren Kabel in der Farbe orange, sind die Verbindungskabel des Reset-Anschlusses (Pin 9) des ATmega 644-20 PU über einen Pull-Up-Widerstand an +Vcc, da der Eingang am Mikrocontroller negiert ist.

4.3.5 Modularer Aufbau

Der Einsatz des taktilen Sensorarrays gibt vor, einen möglichst flexiblen Einsatz zu gewährleisten. Es ist sinnvoll die Controller in der Nähe der Sensormatrizen zu haben, damit möglichst wenige Kabel von der Hand oder dem Objekt zum Main-Board geführt werden müssen. Eine zentrale Positionierung der Controller beim Comcontroller würde bedeuten, dass

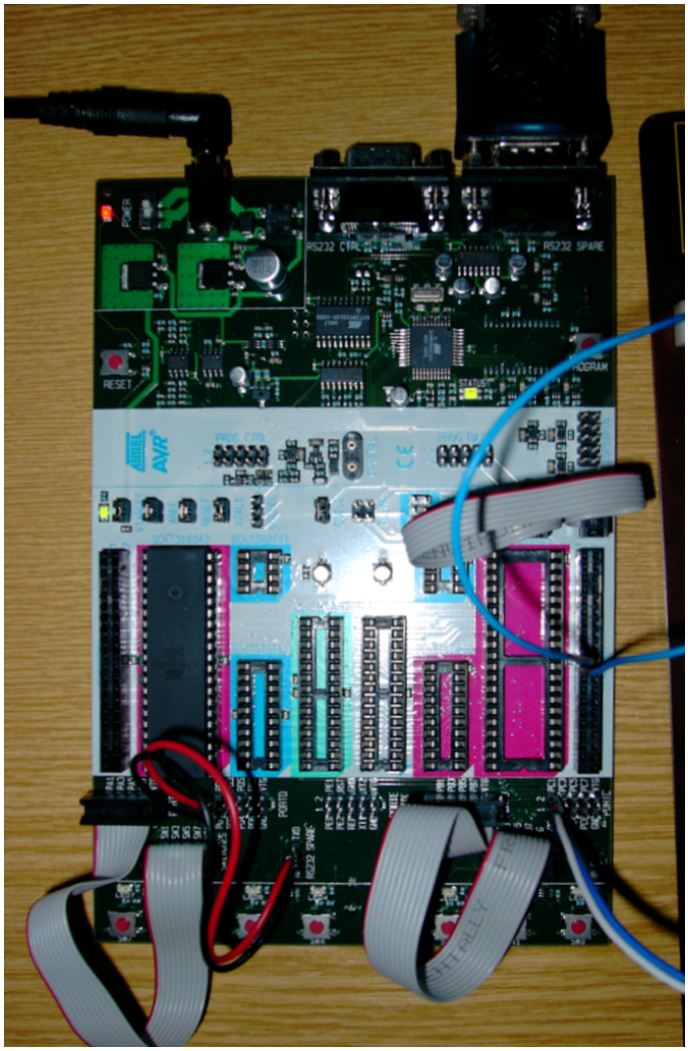


Abbildung 4.7: Das AVR® STK500 Entwicklungsboard mit aufgebauter Comcontroller-Schaltung.

mehrere Tausend Leitungen von der Hand oder dem Objekt, beim Einsatz von theoretisch bis zu 126 Controllern, zu den zentral positionierten Controllern geführt werden müssten. Der TWI/PWR-Bus schafft genau in diesem Punkt Abhilfe. Ein einzelnes 4-poliges Kabel reicht aus, um alle notwendigen Daten zu übertragen. Das bedeutet aber auch, dass jeder Controller auf einer eigenen Platine realisiert werden muss, um an den entsprechend günstige Positionen eingesetzt werden zu können.

Die Realisierung der einzelnen Platinen erfordert zunächst ein geeignetes Leiterbahnen-Layout zur Konstruktion. Die Platinengröße wird durch die Größe der Bauteile bestimmt. Da für den Prototypen die gleichen Bauteile Verwendung finden wie in den Versuchsaufbauten, werden Platinen mit einem Rastermaß von 2,54mm verwendet.

Die in der Abbildung 4.9 gezeigten Leiterbahnen-Layouts für den Hand-, Objekt- und Comcontroller sind spiegelverkehrt zur Bestückungsseite, da es sich um eine Verdrahtung auf der

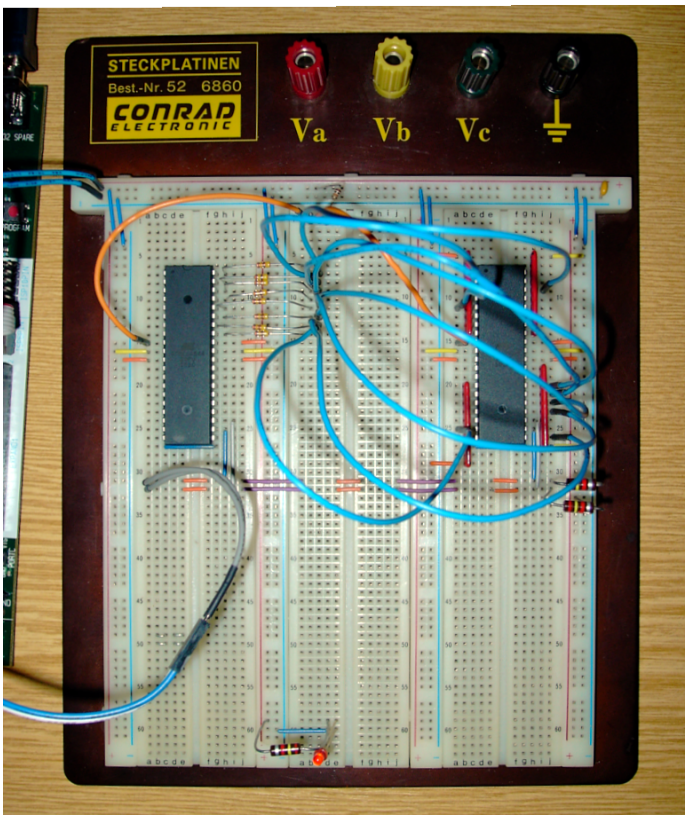


Abbildung 4.8: Die Erweiterungssteckplatine mit aufgebauter Handcontroller- und Objektcontroller-Schaltung.

Rückseite der entsprechenden Platine handelt.

Das Platinenlayout für den Comcontroller ist ein wenig umfangreicher als aus den Versuchsaufbauten bekannt. Es muss nun eine verpolsichere stabilisierte und geglättete Spannungsversorgung sicher gestellt werden, mit der Möglichkeit diese bei Bedarf an und auch wieder ab zu schalten. Des weiteren muss nun auf der Platine des Comcontroller die RS-232 Schnittstelle mit untergebracht werden, um die Kommunikation mit einem Rechnersystem sicher zu stellen.

Die Schnittstellen des Comcontrollers befinden sich an den äußeren Seiten, damit sich die Verbindungskabel besser einstecken lassen. Neben der Spannungsversorgungsklemme befindet sich der Schalter zum an- und abschalten des Gesamtsystems. Eine rote 3mm Leuchtdiode dient hierzu als Betriebsanzeige. Links neben der Leuchtdiode befindet sich der 1μF Kondensator zu Stabilisierung und Glättung der Spannung. Die daneben liegende Brückenschaltung aus vier 1N4001 Siliziumdioden dient zum Verpolschutz. Der kleine 16-Pin IC links neben der RS-232 D-Sub 9 Schnittstelle ist der MAX232 von Texas Instruments mit den dazugehörigen Kondensatoren für die Kommunikation mit einem Rechnersystem. Der große 40-Pin IC ist der ATmega 644-20 PU. Der eigentliche Comcontroller. Auf der rechten Seite in der Abbildung 4.10 befinden sich die zwei Printstecker für den TWI/PWR-Bus zum Anschluss von Hand- und Objektcontrollern über 4-polige Leitungen.

Der Handcontroller entspricht im Platinenlayout so ziemlich dem der Erweiterungssteckplatine

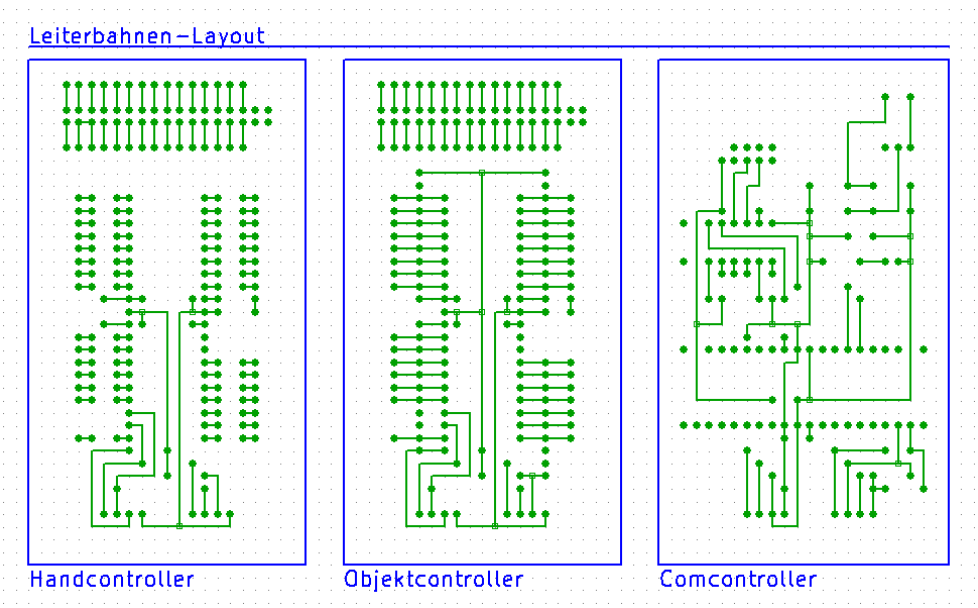


Abbildung 4.9: Das Leiterbahnen-Layout für den Hand-, Objekt- und Comcontroller-Aufbau.

im Versuchsaufbau. Der Mikrocontroller ATmega 644-20 PU ist der Mittelpunkt der Schaltung. Die Siliziumdioden 1N4148 zum Schutz vor unerwünschten Eingangssignalen, an den Ein-/Ausgangs-Pins des Mikrocontrollers, sind direkt neben dem Mikrocontroller in der Abbildung 4.11 erkennbar. An der rechten Seite befinden sich, wie auf der Comcontrollerplatine, die zwei 4-poligen Printstecker für den TWI/PWR-Bus. Ein Printstecker dient als Anschluss der Platine an das System, der zweite ist die Erweiterung für einen weiteren Controller. Auf der linken äußeren Seite der Platine befindet sich der 34-polige Wannensteckverbinder. Zwar ist das System nur für 30 Kontakte ausgelegt, dient die 34-polige Steckverbindung dennoch der Benutzung von genormten Bauteilen. Die Verbindungsleitungen von dem Wannenstecker zu den Ein-/Ausgängen des Mikrocontrollers sind als einzelne Drähte realisiert. So ist es möglich das Platinenlayout einfach zu halten. Eine Verdrahtung als Leiterbahn hätte die Größe der Platine erheblich beeinflusst.

Entwurf und Platinenlayout des Objektcontrollers sind ähnlich dem des Handcontrollers und entspricht in weiten Teilen des Layouts des Versuchsaufbaus. Den Mittelpunkt der Schaltung bildet der Mikrocontroller ATmega 644-20 PU dessen Ein-/Ausgänge mit Pull-Down-Widerständen auf Masse gezogen werden müssen. Dies geschieht mit 10kΩ Widerstandsarrays in Sternschaltung, welche direkt neben dem Mikrocontroller angebracht sind. Das restliche Layout ist mit dem des Handcontrollers identisch. Auf der rechten Seite die zwei 4-poligen Printstecker für den TWI/PWR-Bus und auf der linken Seite der 34-polige Wannenstecker als Steckverbinder für eine Kontaktmatrix, dessen Anschluss am Mikrocontroller über einzelne Drähte realisiert ist.

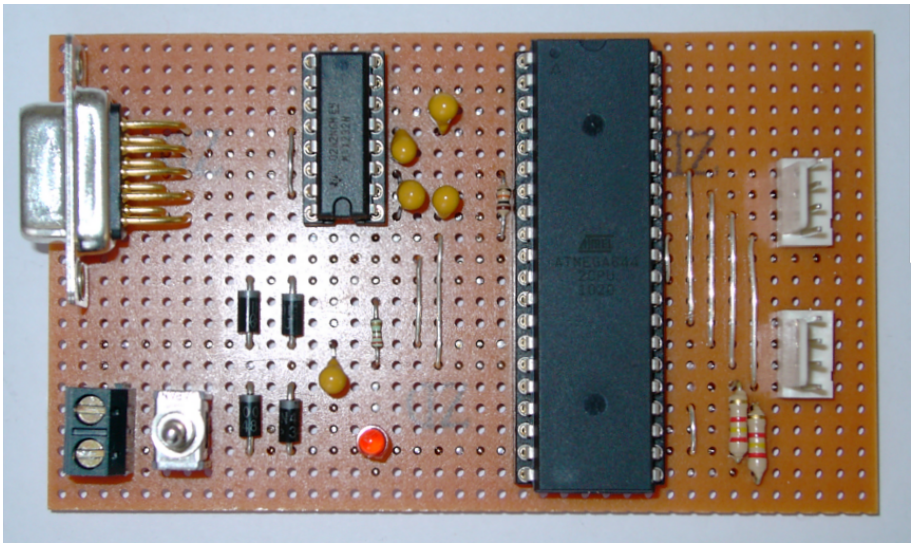


Abbildung 4.10: Das Platinen-Layout des Comcontrollers von der Vorderseite.

4.4 Kontaktmatrix

Eine Kontaktmatrix ist die Sensorfläche des Hand- oder Objektcontrollers. Es sind die Kontaktflächen, welche auf der Hand oder dem Objekt aufgebracht werden sollen und die Kontaktierung für die Positionsbestimmung herzustellen.

Im Folgenden werden zwei unterschiedliche Konzepte der Umsetzung gezeigt. Die erste und günstig zu realisierende Möglichkeit sind einfache Kontaktflächen, eine Fläche für jeden Ein-/Ausgang des Mikrocontrollers. Die zweite Möglichkeit ist, ein Gewebe aus Drähten zu erstellen, wobei eine Kontaktierung von zwei oder mehr Drähten Aufschluss über die Koordinate geben kann.

4.4.1 Der erste Aufbau einer Matrix

Kontaktmatrix des Handcontrollers. Der erste Matrixaufbau stellt noch ein simples System dar. Auf einer Lochrasterplatine mit Rastermaß von 2,54mm sind Kabel aufgelötet. Mit einem Abstand von je drei Löchern und einer Anordnung von vier mal vier, ergeben die Lötpunkte auf der Lötseite der Platine 16 Kontaktpunkte. Dies stellt die Kontaktmatrix für den Handcontroller dar und gehört somit zu der sendenden Einheit.

Die Kabel selbst, auf der anderen Seite der Platine herausgeführt, sind auf zwei Steckerleisten zusammengefasst. Diese Steckerleisten sind ebenfalls im Rastermaß 2,54mm, um auf der Erweiterungssteckplatine aufgesteckt werden zu können. An den Schaltungsaufbau angepasst, sind die Kabel in zwei Stecker mit jeweils acht Verbindungen gebündelt, so dass ein Stecker genau einem Port des ATmega 644-20 PU entspricht.

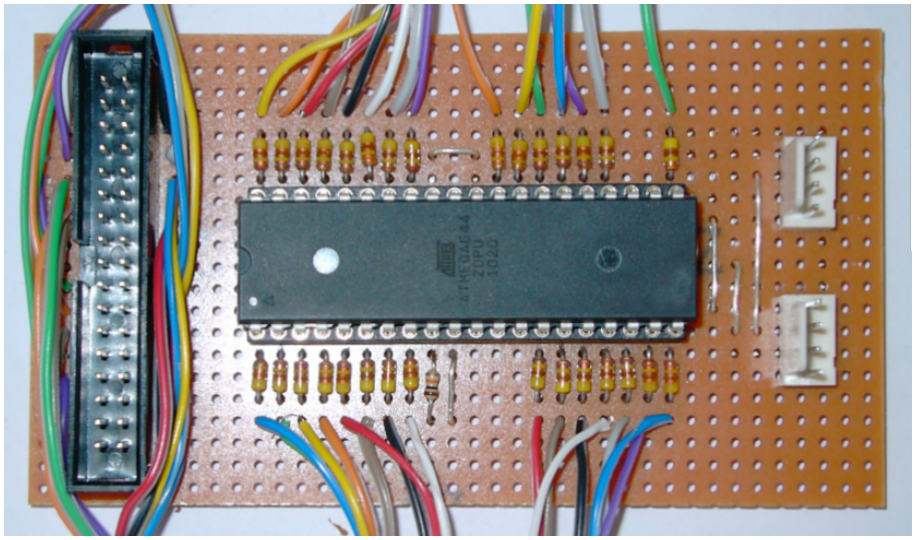


Abbildung 4.11: Das Platinen-Layout des Handcontrollers von der Vorderseite.

Kontaktmatrix des Objektcontrollers. Das Gegenstück von der Kontaktmatrix des Handcontrollers stellt die Kontaktmatrix des Objektcontrollers dar. Auch hier ist noch ein simples System zur Anwendung gekommen. Ähnlich der Kontaktmatrix des Handcontrollers sind Kabel auf einer Lochrasterplatine im Rastermaß 2,54mm aufgelötet. Dabei ist der selbe Lochabstand von drei im Mittelpunkt eingehalten worden. Mit einer Anordnung von vier mal vier ergeben sich auch hier 16 Kontaktpunkte.

Zum Objektcontroller gehörend ist die Kontaktmatrix Teil der empfangenden Einheit. Im Gegensatz zur Kontaktmatrix des Handcontrollers ist die Kontaktmatrix des Objektcontrollers nicht mit einfachen Lötpunkten ausgeführt. Die Lötstellen führen um einen Mittelpunkt herum und bilden so eine kleine Senke (siehe Abbildung 4.13 (rechts)). Dies soll einen besseren Kontakt zwischen den Kontaktmatrizen des Handcontrollers und des Objektcontrollers gewährleisten, da Lötpunkte immer eine unterschiedliche Höhe aufweisen. Die Kabel sind auch bei dieser Kontaktmatrix in zwei Steckerleisten mit einem Rastermaß von 2,54mm zu je acht Verbindungen zusammengefasst, so dass sie genau einem Port des ATmega 644-20 PU entsprechen.

4.4.2 Aufbau einer Gewebematrix

Der zweite Aufbau einer Kontaktmatrix verfolgt eine andere Idee der Gestaltung. Anstelle von einfachen Kontaktflächen wie bei dem ersten Aufbau, ist die Idee für den zweiten Aufbau eine Art Gewebe aus leitfähigen Drähten herzustellen.

Sobald zwei Drähte über eine Kontaktfläche mit der Kontaktmatrix des Handcontrollers in Berührung kommen, kann die Koordinate der Kontaktierung festgestellt werden, da zwei leitende Drähte eine High-Pegel Spannung führen. Ein weiterer Vorteil stellt sich bei der Gewebestruktur ein, wie die Abbildung 4.14 (links) deutlich macht. Mit 12 Drähten ergibt sich

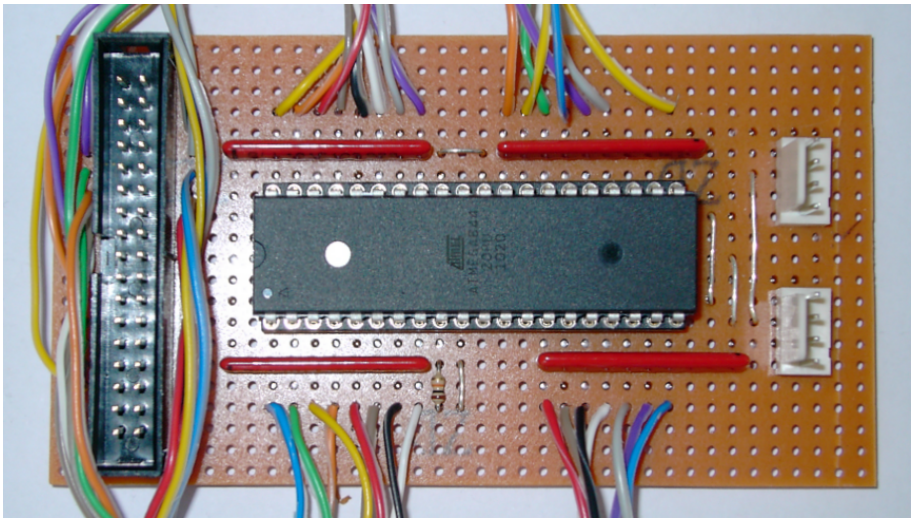


Abbildung 4.12: Das Platinen-Layout des Objektcontrollers von der Vorderseite.

eine Matrix aus 36 einzelnen Kontaktpunkten. Die Dichte erhöht sich und somit wird auch die Auflösung des Sensors erheblich verbessert.

Auf einer Lochrasterplatine ist dieses Prinzip der Gewebematrix mit angeschliffenen Kupferlackdraht umgesetzt. Mit acht Drähten ergeben sich, wie in Abbildung 4.14 (rechts) gezeigt, 16 Kontaktpunkte. Das sind genauso viele Kontaktpunkte wie im ersten Aufbau einer Matrix (siehe dazu Kapitel 4.4.1), jedoch mit dem entscheidenden Vorteil, dass hier nur ein Port des ATmega 644-20 PU benutzt wurde und somit die Ressourcen besser genutzt werden können.

Allerdings ist dieses Prinzip nicht uneingeschränkt einsetzbar. Als Kontaktmatrix des Objektcontrollers in der empfangenden Einheit erfüllt es durchaus seinen Zweck. Die Kontaktflächen der Kontaktmatrix des Handcontrollers sind deutlich größer als die dünnen Drähte und können dadurch mehrere Drähte gleichzeitig überbrücken. Auch die Anordnung auf einem Rastermaß von 2,54mm ist der Dichte nicht zuträglich. Das hat zur Folge, dass der Einsatz als Kontaktmatrix des Handcontrollers in der sendenden Einheit keine gute Kontaktierung ergibt. Jedoch muss hier, wie schon erwähnt, die Dichte berücksichtigt werden und auch die Tatsache, dass das Gewebe auf einer starren Platine aufgebracht ist, die nicht gebogen werden kann, um einzelne Drähte zur Kontaktierung mit der anderen Kontaktmatrix zu bringen. Je höher die Dichte und umso flexibler der Untergrund, desto besser sollte sich das Prinzip anwenden lassen.

4.4.3 Handschuh und Gewebematrixtuch

Der dritte Aufbau bringt nun die in den ersten beiden Aufbauten getesteten Möglichkeiten einer Kontaktmatrix näher an das geforderte Konzept heran und führt auch gleichzeitig näher an das HandleProject. Umgesetzt ist hier einerseits die Kontaktmatrix als reine Kontaktflächen für den Handcontroller in der sendenden Einheit, als auch die Gewebematrix für den Objektcontroller in der empfangenden Einheit.

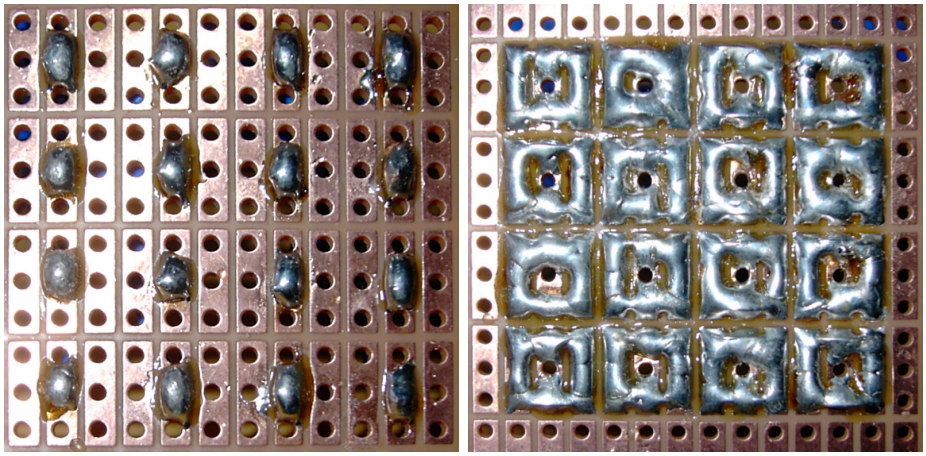


Abbildung 4.13: Für den Handcontroller (links) mit einfachen Lötpunkten und dem Objektcontroller (rechts) mit Lötflächen realisiert.

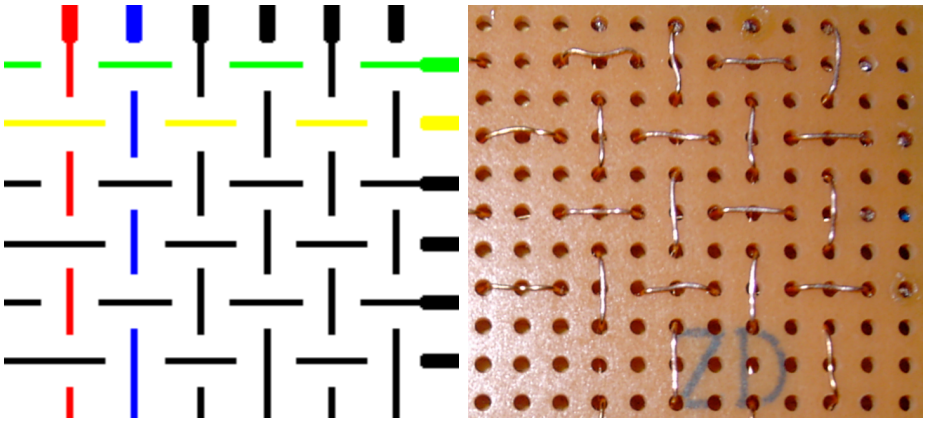


Abbildung 4.14: Die schematische Darstellung (links) und die Realisierung (rechts).

Der Handschuh. Für die Kontaktmatrix des Handcontrollers mit reinen Kontaktflächen ist ein Handschuh exemplarisch am Zeigefinger der rechten Hand mit sechs Kontaktflächen ausgestattet worden. Die Anordnung sind jeweils zwei Kontakte pro Fingerglied.

Die einzelnen Kontaktflächen sind im Handschuh mit dünnem Kupferlackdraht verlötet und zum Schutz vor mechanischen Beanspruchungen zusätzlich verklebt. Diese werden zwischen dem Zeigefinger und Daumen wieder hinaus geführt. Der dünne Kupferlackdraht ist flexibel genug, um den Bewegungen einer Hand zu folgen. Eine entsprechende Fixierung der Kupferlackleitungen und eine Bewegungsreserve, schützen vor zu starken Belastungen. Im Handschuh sind die Kupferlackleitungen zusätzlich mit Isolierband fixiert, was einerseits vor mechanischen Beanspruchungen schützen soll, zum anderen aber auch eine versehentliche Kontaktierung durch den entsprechenden Finger vermeiden soll. In der Abbildung 4.15 wird auch gezeigt, dass die Verbindung vom Kupferlackdraht auf das 34-polige Flachbandkabel überführt wird. Das ist notwendig, da das Flachbandkabel stabiler und nicht so anfällig auf äußere Beanspruchung reagiert. Ein Kontaktschluss zwischen zwei oder mehreren Drähten wegen einer beschädigten

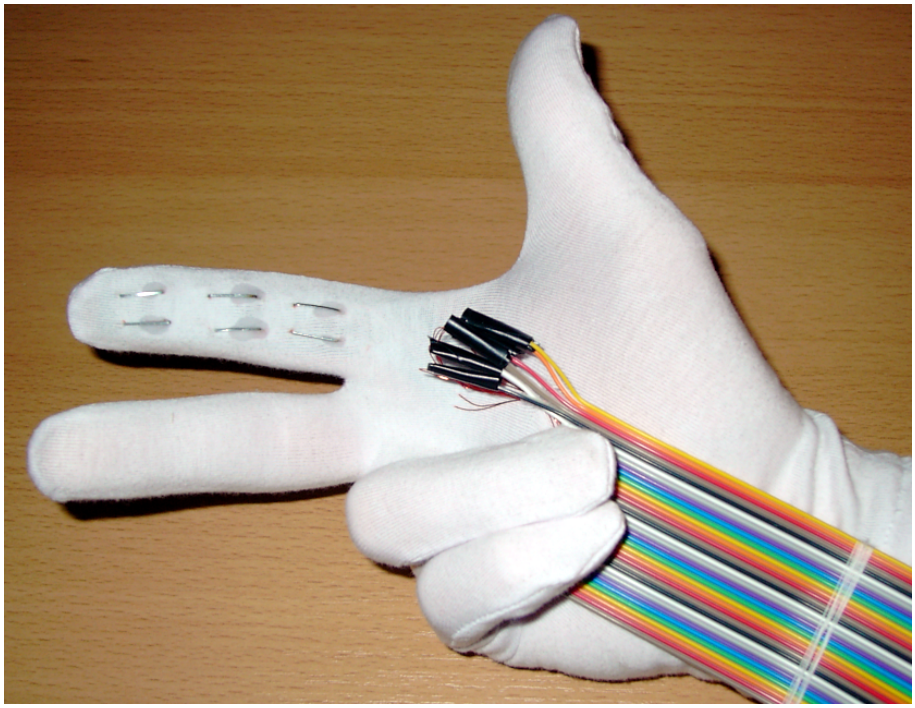


Abbildung 4.15: Für den Handcontroller ist der Handschuh mit einer Kontaktmatrix belegt.

Lackschicht kann somit verhindert werden. Eine Isolierung um die Lötstellen des Kupferlackdrahtes und des Flachbandkabels verhindern ebenfalls einen ungewollten Kontaktschluss. Das Flachbandkabel dient zusätzlich zur besseren Verbindung zum Handcontroller, da im modularen Aufbau der Schaltung (siehe dazu Kapitel 4.3.5) 34-polige Steckverbinder als Schnittstelle zur Kontaktmatrix eingesetzt werden.

Das Gewebematrixtuch. Das zu greifende Objekt, an dem die Koordination zur Hand bestimmt werden soll, kann sehr unterschiedlich sein. Ein Tuch, in welches eine Gewebematrix eingearbeitet ist, dient als erster Ansatz für eine möglichst flexible Nutzung. Das Gewebematrixtuch kann, je nach Größe des Objektes, um das Objekt herum gelegt und befestigt werden. Wenn das Objekt gewechselt wird kann das Tuch auch weiterhin für das nächste Objekt benutzt werden. Ein zu großer Aufwand wäre es alle Objekte mit einer eigenen Kontaktmatrix auszustatten.

Das Gewebematrixtuch ist mit 15 Kupferlackdrähten vertikal und 15 Kupferlackdrähten horizontal durchwebt. Auf einer Fläche von ca. 16cm×16cm ergibt sich ein Abstand der Drähte untereinander von 1cm und damit eine Auflösung von 225 Kontaktpunkten. Um einen flexibleren Umgang mit dem Tuch zu gewährleisten, wird genau wie bei dem Handschuh ein 34-poliges Flachbandkabel zur Verbindung zum Objektcontroller benutzt. Die Kupferlackdrähte sollten, wegen ihrer Anfälligkeit bei mechanischer Beanspruchung, nicht als Verbindungskabel zum Controller genutzt werden. Die Lötstellen zwischen dem Flachbandkabel und dem Kupferlackdraht sind mit einer Isolierung umschlossen, damit kein Kontaktschluss entstehen kann. Aus dem selben Grund sind auch die Enden der einzelnen Drähte mit einem Isolierband überzogen.

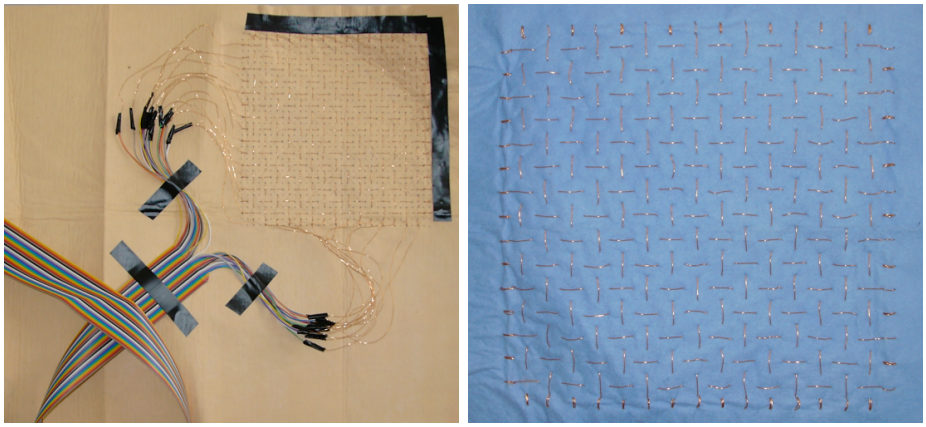


Abbildung 4.16: Für den Objektcontroller ist eine Gewebematrix aus Kupferlackdraht in das Tuch eingearbeitet worden.

Da die Kupferlackdrähte mit einem Lack zur Isolierung überzogen sind, muss auf der zur Kontaktierung genutzten Seite des Gewebematrixtuches, hier in blau, der Lack vorsichtig entfernt werden. Trotz der Drähte bleibt das Tuch flexibel um sich unterschiedlichen Gegebenheiten anzupassen.

4.5 Technische Daten

Die technischen Daten geben die Eckdaten des erstellten Prototypen. Sie dienen als Referenz für den Einsatz. Bei der Spannungsversorgung braucht auf keine korrekte Polung geachtet werden.

Comcontroller	
Abmessungen:	57mm × 100mm × 20mm
Spannungsversorgung:	5V DC
Stromaufnahme (ohne RS-232):	ca. 15,8mA
Stromaufnahme (mit RS-232):	ca. 17,3mA
max. zulässiger Strom:	300mA
Schnittstellen:	1x Stromversorgung - Klemmenanschluss
	1x RS-232 - D-Sub 9 (female)
	2x TWI/PWR - Printstecker 4-polig
Handcontroller	
Abmessungen:	54mm × 100mm × 14mm
Spannungsversorgung:	5V DC (stabilisiert, über TWI/PWR)
Stromaufnahme:	ca. 5,8mA
Schnittstellen:	2x TWI/PWR - Printstecker 4-polig
	1x Kontaktmatrix - Wannenstecker 34-polig
Objektcontroller	
Abmessungen:	54mm × 100mm × 14mm
Spannungsversorgung:	5V DC (stabilisiert, über TWI/PWR)
Stromaufnahme:	ca. 3,0mA
Schnittstellen:	2x TWI/PWR - Printstecker 4-polig
	1x Kontaktmatrix - Wannenstecker 34-polig

Ausblick

5

Als Prototyp realisiert, bieten sich noch eine Vielzahl von Erweiterungen oder Verbesserungen des taktilen Sensorarrays an. Nicht alles konnte bei der Konstruktion des Prototypen berücksichtigt werden.

Zunächst gibt es die Möglichkeit die hier eingesetzte Software zu erweitern, um so eine Verbesserung des Systems zu erreichen. Aktuell werden bei der Anbindungssoftware auf einem Computersystem die Kontaktschlüsse als Momentanzeige in einem entsprechenden Anzeigefeld visualisiert. Ein verbessertes System könnte hier die angezeigten Kontaktschlüsse eine Weile zwischenspeichern, sodass auf dem Anzeigefeld eine verblassende Spur dargestellt werden kann. Eine weitere Erweiterung könnte die Darstellung des Anzeigefeldes selber sein. Hier ist es natürlich einerseits denkbar die Hand oder vielleicht auch das Objekt selber als Schema darzustellen. Eine andere, aber komplexere und aufwändigere, Darstellungsmöglichkeit ist eine dreidimensionale Darstellung der Hand und des Objektes. Dort könnte dann die Greifbeziehung konkret simuliert dargestellt werden.

Denkbare Verbesserungen seitens der Software für das Gesamtsystem sind nicht nur in der Darstellung der Ergebnisse zu sehen. Mittels der eingesetzten Software wäre es zudem möglich, Mehrfachkontakte zwischen Hand und Objekt auf einen zentralen Punkt zu reduzieren, wenn dies sinnvoll erscheint. Des Weiteren ist es theoretisch möglich, dass bei der Kontaktierung von z.B. zwei Fingern auf der empfangenden Kontaktmatrix die Kontakte des einen Fingers eine Überbrückung herstellen, die ein Signal vom anderen Finger weiterleiten. Dies würde dazu führen, dass zu einem Kontakt der Hand zwei Kontakte auf dem Objekt registriert würden. Dieser Effekt ließe sich mittels Software bereinigen, wenn z.B. eine Art von Positionswahrscheinlichkeit der Hand über bereits schon gemessene Kontaktschlüsse errechnet würde.

Zur Zeit werden dem einzelnen Modul, des Hand- bzw. Objektcontrollers, des prototypischen Systems manuell in der Software des jeweiligen Controllers eine Adresse für das eingesetzte Bussystem zugewiesen. Für den dynamischen Einsatz ist diese Lösung undenkbar. Hier muss eine Möglichkeit der dynamischen Adressierung geschaffen werden, sodass Module im laufenden Betrieb des Systems hinzugefügt, abgekoppelt oder ausgetauscht werden können.

Eine sehr entscheidende Verbesserung des Systems für den künftigen, vielleicht sogar dauerhaften, Einsatz ist die Miniaturisierung. Wie auch schon in manchen Kapiteln angeklungen, sind die verwendeten technischen Komponenten in einer Größe gewählt, sodass Versuchsaufbauten bequem zu realisieren sind. Ausnahmslos alle verwendeten Bauteile sind aber auch in einer kleinen Version für hochintegrierte SMD-Schaltungen verfügbar. Von den Dioden, über die Widerstände, Kondensatoren, der Schnittstellen bis hin zum RS-232 Treiberbaustein und

natürlich dem Mikrocontroller ATmega 644-20 PU als AU Variante [Atm10a, vgl. Seite 362]. Es ermöglicht die Dimension der Schaltungen auf wenige Zentimeter zu verkleinern, ohne den Verlust an Leistungsfähigkeit an irgendeiner Stelle des Systems.

Die elektrischen Schaltpläne des taktilen Sensorarrays sind mit Hilfe der KiCad suite erstellt worden. Die KiCad suite ist eine Sammlung von Programmen die es ermöglichen den Schaltplan zu erstellen und auf Korrektheit zu prüfen, die in den Schaltpan verwendeten Bauteile mit physisch existierenden zu assoziieren, ein Platinenlayout zu entwickeln und dann als Gerber-File zur Vorlage von Platinen Ätzungen auszugeben [kic10, vgl.]. Das ermöglicht das weiterarbeiten mit den bereits erstellten Schaltplänen. Das Assoziieren der Bauteile kann für die Miniaturisierung also auf entsprechend kleinere Bauteile geschehen. Zudem kann auch eine Platinenvorlage erstellt werden, sodass keine Rasterplatinen Verwendung finden müssen. Auch das verkleinert eine Schaltung enorm.

Der TWI-Bus, bereitgestellt vom ATmega 644-20 PU, arbeitet mit einer Geschwindigkeit von bis zu 400kHz [Atm10a, vgl. Seite 200 ff.]. Das ermöglicht eine Übertragungsrate von bis zu 400kBit/s [i2c11, vgl.]. Sollte es dennoch nötig sein eine höhere Datenübertragungsrate zu benutzen, können derzeit sogar schon aktuellere Varianten 3,4MBit/s übertragen, ohne weitere Kabel zu benutzen. Es bleibt bei zwei Kabeln für den Bus [i2c11, vgl.]. Des Weiteren gibt die neue 10Bit-Adressierung die Möglichkeit bis zu 1024 Geräte an den Bus zu koppeln. Die Abwärtskompatibilität zu der 7Bit-Adressierung ist gegeben, sodass bestehende Komponenten deswegen nicht ausgetauscht werden müssen [i2c11, vgl.].

Die RS-232 Schnittstelle bietet auch Ausblick auf eine Verbesserung. Die Leistungsfähigkeit des gesamten taktilen Sensors hängt zur Zeit an der Schnittstelle. Je mehr Daten übertragen werden müssen, umso mehr bremst die Schnittstelle das Gesamtsystem. Das bedeutet, je hochauflösender die Kontaktmatrizen, je mehr Kontaktierungen, desto mehr Daten müssen über die Schnittstelle an das Rechnersystem übertragen werden. Es gäbe einmal die Möglichkeit eine Filterfunktion in der Software zu implementieren, die mehrere Kontaktpunkte auf eine bestimmte Koordinate rechnen würde. Eine andere Möglichkeit ist die Übertragung via USB. Ohne konkrete Datenraten nennen zu brauchen ist USB als Nachfolger der, mittlerweile veralteten, RS-232 Schnittstelle schneller.

Nicht zuletzt kann auch die Kontaktmatrix weiterentwickelt werden. Der Prototyp des taktilen Sensorarrays ist schon mit zwei Versionen realisiert. Die einfachen Kontaktflächen ermöglichen eine gute Kontaktierung, während die Gewebematrix sehr hochauflösend realisiert werden kann. Da ist in Bezug auf Dichte der Kontakte und dem verwendeten Material noch Potenzial zur Erweiterung und zur Verbesserung.

Fazit

6

Diese Bachelorarbeit diente dem Zweck, ein taktiles Sensorarray als Konzept und als Prototyp umzusetzen.

Durch die Zielsetzung bestimmte Vorgaben wurden schon bei der Entwicklung des Konzeptes berücksichtigt. So ist eine der Haupteigenschaften des Systems eine hohe Flexibilität der Konstruktion in Bezug auf Erweiterung. Das entwickelte Konzept für den Prototypen sieht vor, ein modulares System aufzubauen, welches sich aus den verschiedenen Aufgaben zusammensetzt. So ist ein Handcontroller für die Sensorflächen der Hand, ein Objektcontroller für die Sensorflächen am Objekt und ein Comcontroller der alle Controller miteinander verbindet angedacht worden. Das entwickelte Konzept sieht auch den Einsatz in realen Systemen vor. Daraus ergeben sich auch dementsprechend enge Vorgaben, was zeitliche Kommunikation anbelangt.

Die Entwicklung des Prototypen teilte sich in zwei Bereiche auf, die Software und die Hardware. Während die Software die Funktionsweise und auch die Kommunikation steuert, bildet die Hardware die Grundlage dafür.

Es ist gelungen das erdachte Konzept, welches den Vorgaben der Zielsetzung entspricht, in einem Prototypen umzusetzen der alle Anforderung erfüllt. Die Software ist so flexibel umgesetzt, dass es möglich ist, unterschiedliche Ausprägungen der Sensorflächen zu benutzen, ohne die Software dabei verändern zu müssen. Auch den zeitlichen Anforderungen werden softwaretechnisch genüge getan, trotz der Benutzung von älteren Kommunikationsschnittstellen, die nicht die Unterstützung der heute gängigen Übertragungsraten in MBit-Bereichen haben. Die hardwareseitige Realisierung bietet, zusammen mit der Software, eine große Modularität auf die es ermöglicht mit nur einem Kabel zum zentralen Comcontoller theoretisch bis zu 126 Controller zu betreiben.

Der Prototyp des taktilen Sensorarray gibt guten Grund zur Annahme, ein solches Konzept weiterzuentwickeln. Es ist schon mit dem Prototyp möglich, klare Ergebnisse bei der Umsetzung einer Positionsbestimmung zu erzielen.

Danksagung

An dieser Stelle möchten wir uns bei den Menschen bedanken, die es uns ermöglicht haben diese Bachelorarbeit zu erstellen.

Herrn Dr. N. Hendrich möchten wir als erstes danken, dass er uns die Möglichkeit gegeben hat diese Arbeit schreiben zu können. Auch für die Unterstützung und Leitung während des Arbeitens und die Bereitstellung aller erforderlichen Mittel möchten wir uns bedanken.

Herrn Prof. Dr. J. Zhang gilt unser Dank für die Übernahme des Erstgutachtens, da sonst die Arbeit gar nicht möglich gewesen wäre.

Herrn D. Klimentjew möchten wir danken, dass er uns überhaupt auf die Idee gebracht hat im Arbeitsbereich TAMS unsere Bachelorarbeit zu schreiben und dass er bei auftretenden Fragen für uns immer Ansprechpartner war.

Allen weiteren Mitarbeitern des Arbeitsbereiches TAMS für die Antworten auf die häufig gestellten Fragen zwischen Tür und Angel und das unzählige aufschließen der Werkstatt, damit wir arbeiten konnten, möchten wir herzlich danken.

Hendrik Udo Linne, persönliche Danksagung.
Ich möchte die Gelegenheit nutzen um meiner gesamten Familie für die Unterstützung während meiner Bachelorstudienzeit zu danken. Besonderen Dank gilt hierbei meinen Eltern Udo und Margot Linne, die mich in Allem unterstützt haben und auch weiterhin unterstützen wollen. Weiterer Dank gilt meiner Schwester Mareike Linne, die mich so manches Mal aus dem Alltag heraus geholt hat, und auch bei meiner Oma Agnes Apel möchte ich mich bedanken, die immer ein offenes Ohr für mich hat. Nicht zuletzt möchte ich hier meinen Freunden für die gegebene Motivation zum Studium und für die vielen Gespräche und Diskussionen danken.

Literaturverzeichnis

[Atm00] Atmel Corporation (Hrsg.): *AVR STK500.* : Atmel Corporation, 2000. – Available online at `www.atmel.com/atmel/acrobat/doc1925.pdf`, visited on March 18th 2011.

[Atm09] Atmel Corporation (Hrsg.): *AVR311: Using the TWI module as I2C slave.* : Atmel Corporation, 2009. – Available online at `http://www.atmel.com/dyn/resources/prod_documents/doc2565.pdf`, visited on April 9th 2011.

[Atm10a] Atmel Corporation (Hrsg.): *8-bit AVR Microcontroller with 64K Bytes In-System Programmable Flash, ATmega644/V.* : Atmel Corporation, 2010. – Available online at `www.atmel.com/dyn/resources/prod_documents/doc2593.pdf`, visited on March 16th 2011.

[Atm10b] Atmel Corporation (Hrsg.): *AVR315: Using the TWI module as I2C master.* : Atmel Corporation, 2010. – Available online at `http://www.atmel.com/dyn/resources/prod_documents/doc2564.pdf`, visited on April 9th 2011.

[avr11a] *AVR Eclipse.* Website, 2011. – Available online at `http://www.mikrocontroller.net/articles/AVR_Eclipse`, visited on April 9th 2011.

[avr11b] *AVR-Einstieg leicht gemacht.* Website, 2011. – Available online at `http://www.rn-wissen.de/index.php/AVR-Einstieg_leicht_gemacht`, visited on April 6th 2011.

[avr11c] *AVR-GCC-Tutorial.* Website, 2011. – Available online at `http://www.mikrocontroller.net/articles/AVR-GCC-Tutorial`, visited on April 9th 2011.

[bac11] *C# BackgroundWorker Tutorial.* Website, 2011. – Available online at `http://www.dotnetperls.com/backgroundworker`, visited on April 9th 2011.

[BDH+02] Brechmann, Gerhard ; Dzieia, Werner ; Hörnemann, Ernst ; Hübscher, Heinrich ; Jagla, Dieter ; Klaue, Jürgen: *Elektrotechnik Tabellen Energieelektronik Industrielektronik.* Braunschweig : Westermann Schulbuchverlag GmbH, 2002. – 401 S. – ISBN 3–14–225035–2

[BU07] Brinkschulte, Uwe ; Ungerer, Theo: *Mikrocontroller und Mikroprozessoren.* Berlin Heidelberg New York : Springer-Verlag, 2007. – 453 S. – ISBN 978–3–540–46801–1

[Dio02] Diotec Semiconductor (Hrsg.): *Silizium-Planar-Dioden.* : Diotec Semiconductor, 2002. – Available online at `http://www.conrad.de/ce/de/product/162280/DIODE-1N4148-500MW`, visited on April 6th 2011.

[Dio11] Diodes incorporated (Hrsg.): *1N4001 - 1N4007.* : Diodes incorporated, 2011. – Available online at `http://www.diodes.com/datasheets/ds28002.pdf`, visited on April 6th 2011.

[Gre11] Grebe, Wolfgang: *LED-Vorwiderstandsrechner*. Website, 2011. – Available online at `http://www.elektronik-kompendium.de/sites/bau/1109111.htm`, visited on April 6th 2011.

[Hau07] Haun, Matthias: *Handbuch Robotik*. Berlin Heidelberg New York : Springer-Verlag, 2007. – 531 S. – ISBN 978–3–540–25508–6

[hpr07] *Handle Project Website*. Website, 2007. – Available online at `http://www.handle-project.eu/`, visited on March 31th 2011.

[i2c11] *I2C*. Website, 2011. – Available online at `http://www.rn-wissen.de/index.php/I2C`, visited on April 7th 2011.

[kic10] *KiCad - GPL PCB suite*. Website, 2010. – Available online at `http://iut-tice.ujf-grenoble.fr/kicad/`, visited on March 10th 2011.

[Sch10] Schmitt, Günter: *Mikrocomputertechnik mit Controllern der Atmel AVR-Risc-Familie*. München : Oldenbourg Wissenschaftsverlag GmbH, 2010. – 600 S. – ISBN 978–3–486–58988–7

[ser11] *SerialPort-Klasse*. Website, 2011. – Available online at `http://msdn.microsoft.com/de-de/library/system.io.ports.serialport.aspx`, visited on April 9th 2011.

[SK08] Siciliano, Bruno ; Khatib, Oussama: *Handbook of Robotics*. Berlin Heidelberg New York : Springer-Verlag, 2008. – 1611 S. – ISBN 978–3–540–23957–4

[Tex04] Texas Instruments (Hrsg.): *MAX232, MAX232I - DUAL EIA 232 DRIVERS/RECEIVERS*. : Texas Instruments, 2004. – Available online at `focus.ti.com/lit/ds/symlink/max232.pdf`, visited on April 3th 2011.

[thr11] *Gewusst wie: Threadsicheres Aufrufen von Windows Forms-Steuerelementen*. Website, 2011. – Available online at `http://msdn.microsoft.com/de-de/library/ms171728%28v=VS.100%29.aspx`, visited on April 9th 2011.

[twi08] *TWI Master-Slave C-Functions*. Website, 2008. – Available online at `http://www.mikrocontroller.net/attachment/29844/TWI_Master-Slave_C_Functions.zip`, visited on April 9th 2011.

[wik11] *RS-232*. Website, 2011. – Available online at `http://de.wikipedia.org/wiki/EIA-232`, visited on March 21th 2011.

[YBA] Yousef, Hanna ; Boukallel, Mehdi ; Althoefer, Kaspar:

Anhang

Auf den folgenden Seiten befinden sich die elektrischen Schaltpläne des Prototypen des taktilen Sensorarrays. Die elektrischen Schaltpläne sind erstellt worden mit der Softwaresuite KiCad [kic10].

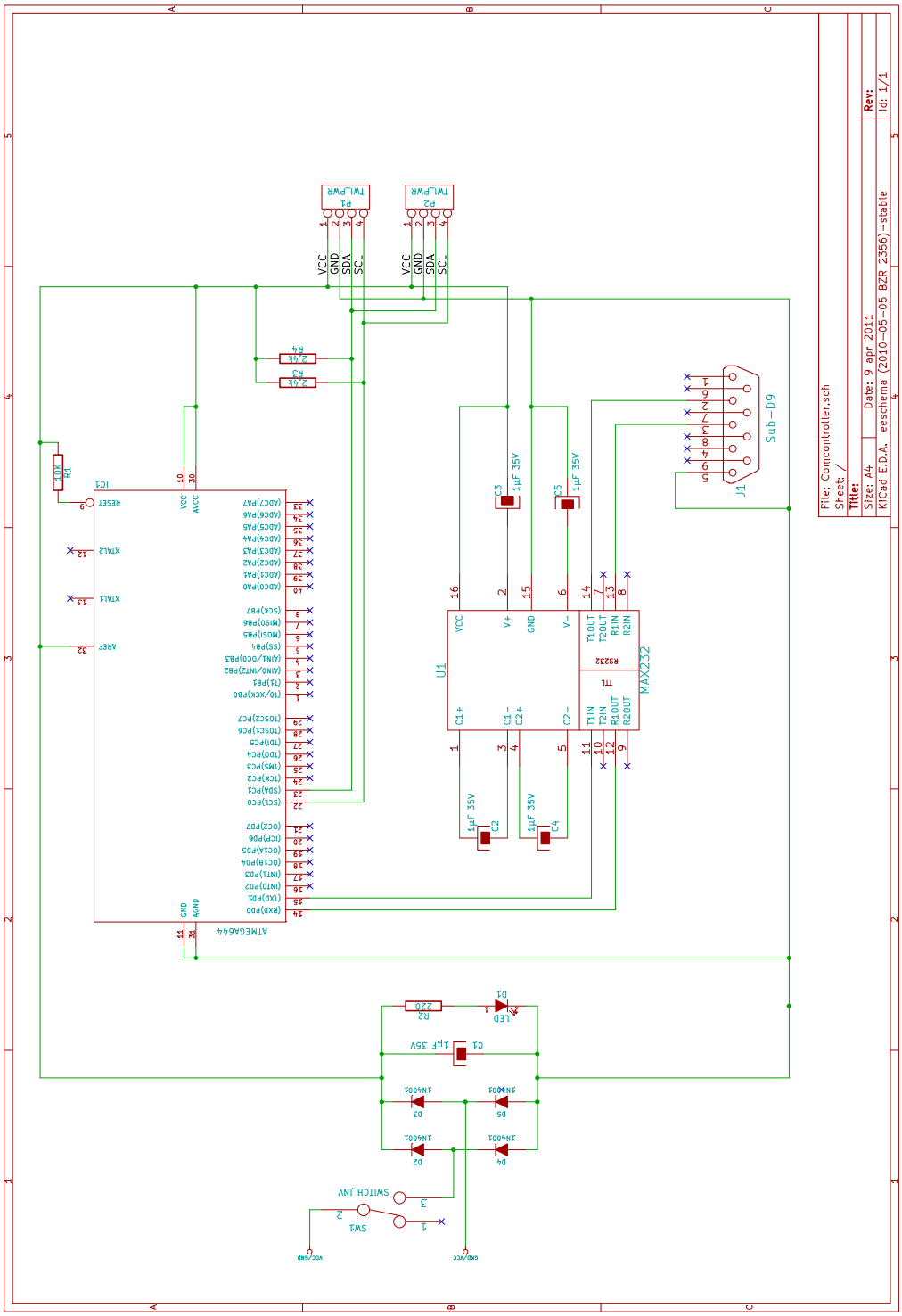


Abbildung 1: Elektrischer Schaltplan des Comcontroller.

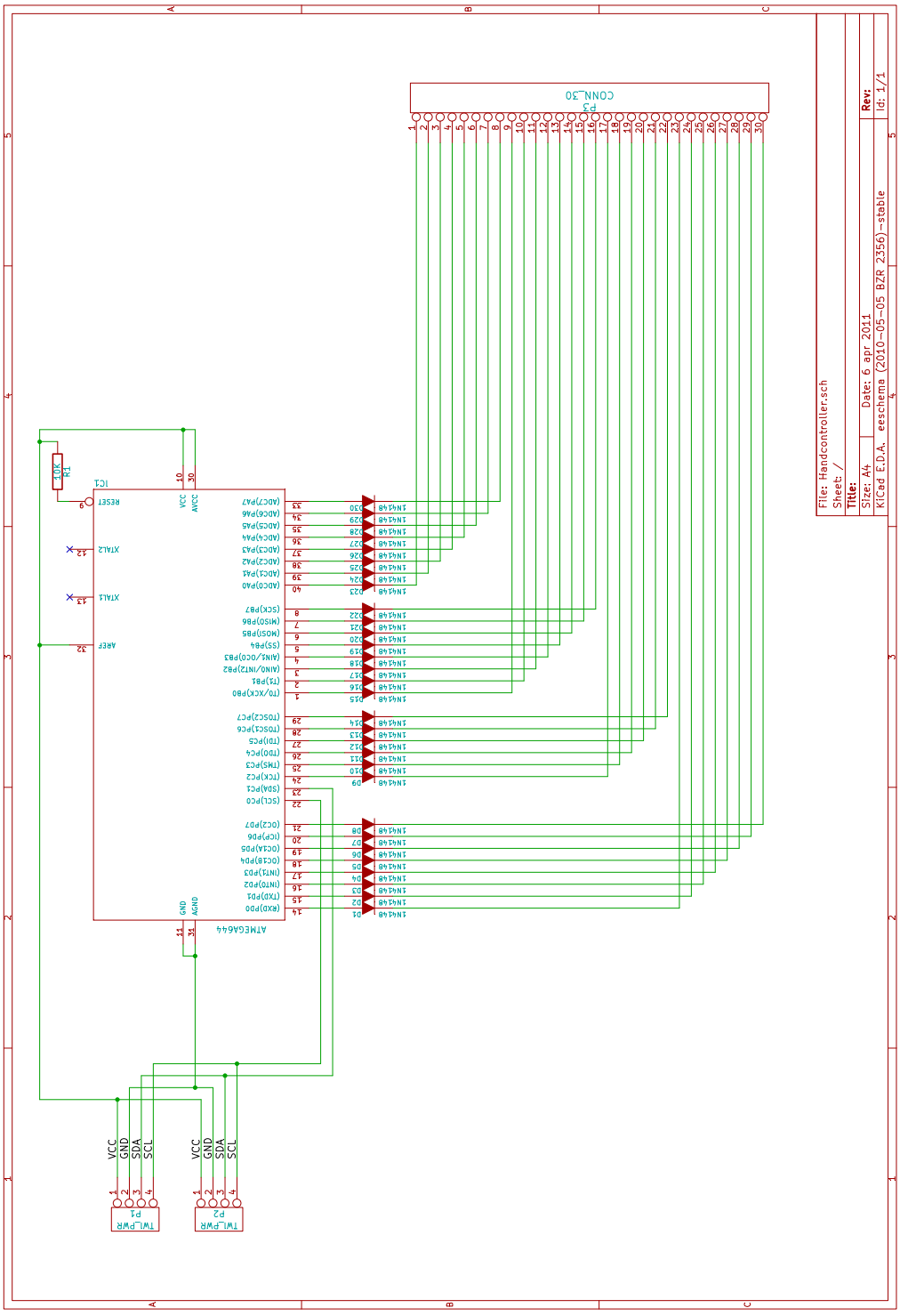


Abbildung 2: Elektrischer Schaltplan des Handcontroller.

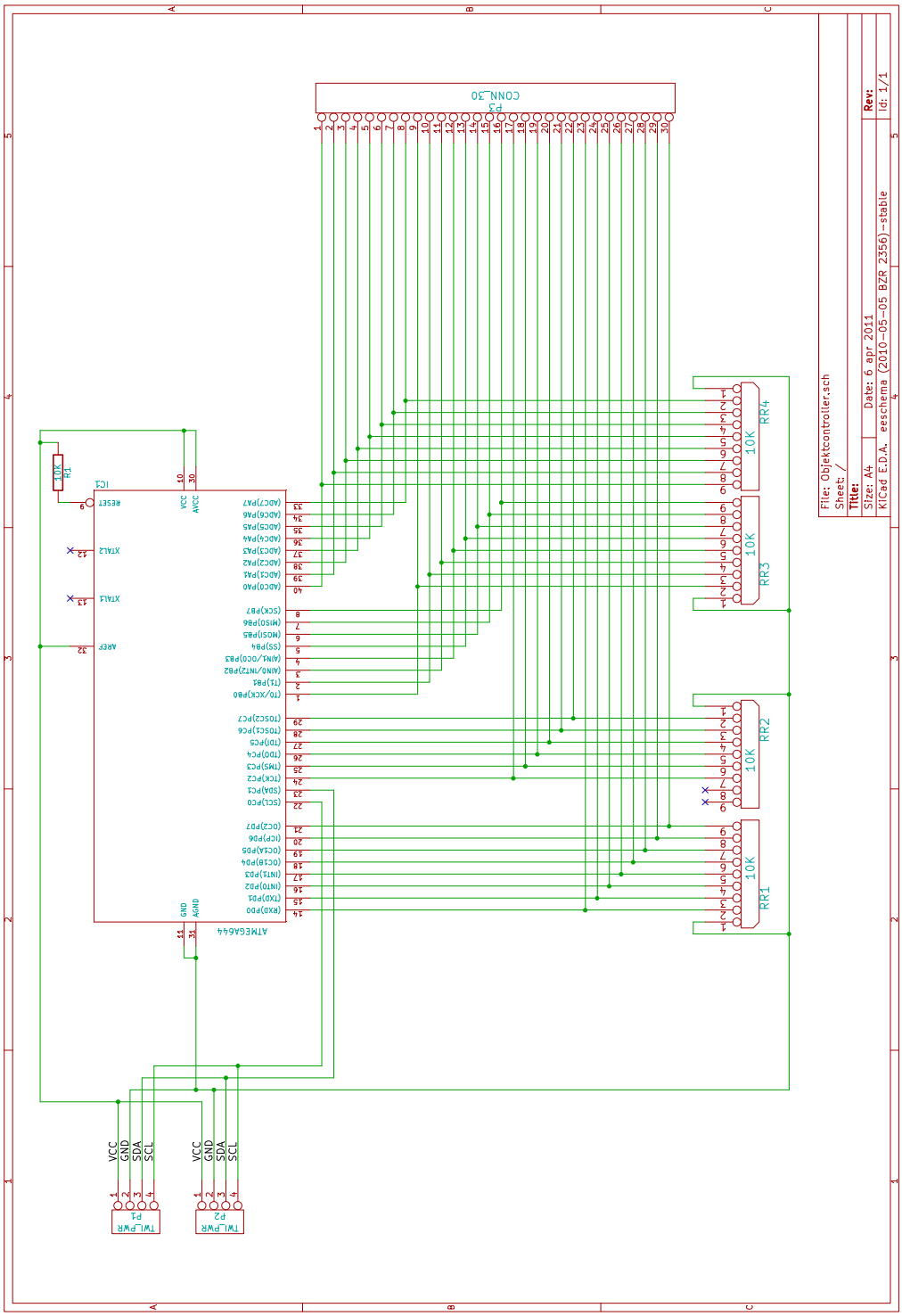


Abbildung 3: Elektrischer Schaltplan des Objektcontroller.

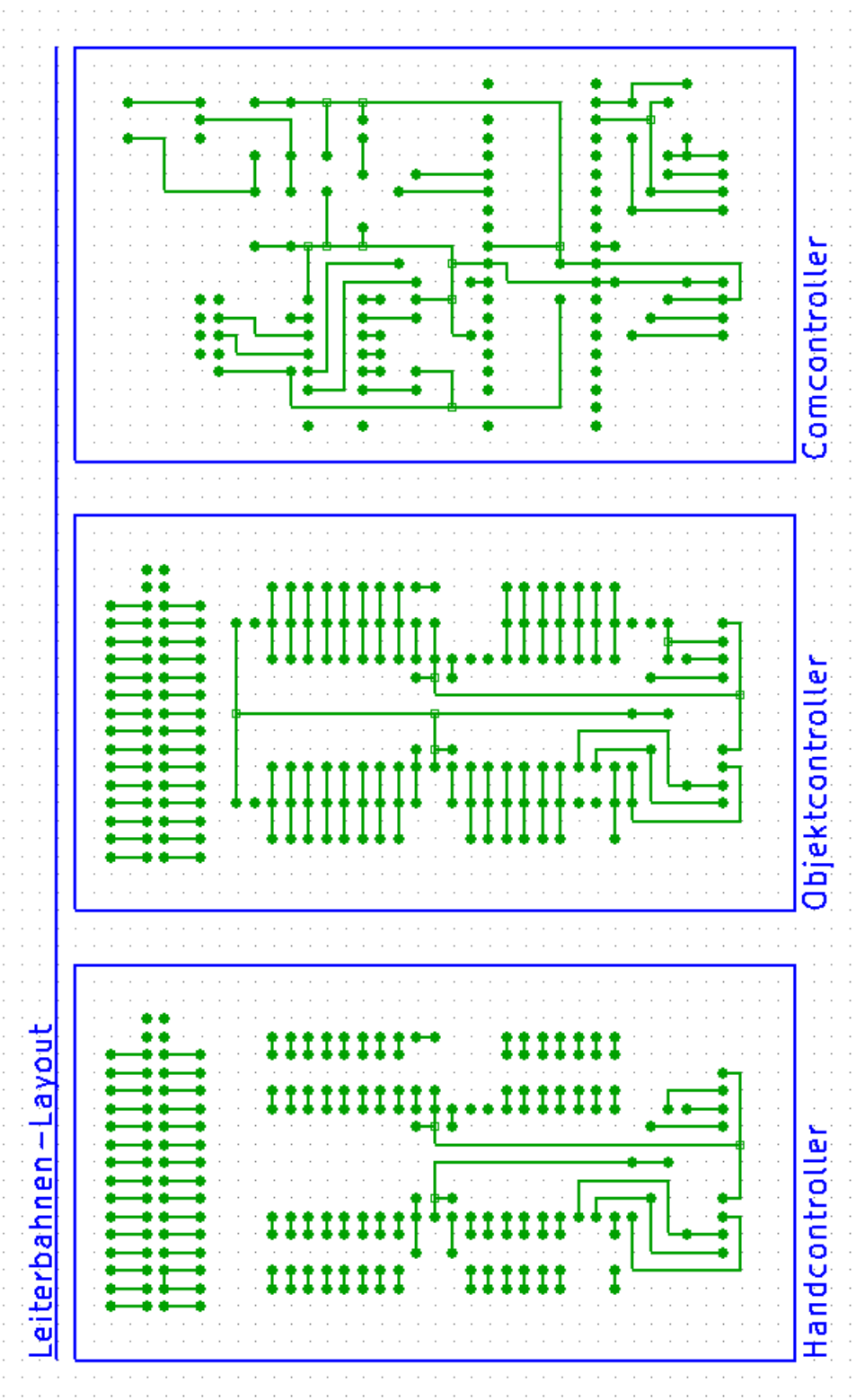


Abbildung 4: Layout der Leiterbahnen für die einzelnen Platinen.

Aufteilung der Gruppenarbeit

Diese Bachelorarbeit wurde gemeinsam von Nikolas Slottke und Hendrik Udo Linne erstellt. Aus diesem Grund sollen im Folgenden die einzelnen Kapitel dem jeweiligen Autor zugeordnet werden.

Nikolas Slottke (Matr.-Nr.: 5945015) schrieb die Kapitel: 2 und 3

Hendrik Udo Linne (Matr.-Nr.: 5944442) schrieb die Kapitel: 1, 4, 5 und 6

Die Entwicklung des Konzeptes und des Prototypen eines taktilen Sensorarrays erfolgte in gemeinsamer Zusammenarbeit.

Erklärung zur eigenständigen Arbeit

Ich, Nikolas Slottke (Matr.-Nr.: 5945015), versichere hiermit die Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen - benutzt zu haben, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht zu haben und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

(Ort, Datum) (Unterschrift)

Ich, Hendrik Udo Linne (Matr.-Nr.: 5944442), versichere hiermit die Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel - insbesondere keine im Quellenverzeichnis nicht benannten Internet-Quellen - benutzt zu haben, die Arbeit vorher nicht in einem anderen Prüfungsverfahren eingereicht zu haben und die eingereichte schriftliche Fassung der auf dem elektronischen Speichermedium entspricht.

Ich bin mit einer Einstellung in den Bestand der Bibliothek des Fachbereiches einverstanden.

(Ort, Datum) (Unterschrift)