Höhere Lehranstalt für Informatik
Machlandstraße 48
A-4320 Perg

Diplomarbeit
Legonator

Programmierung einer interaktiven Benutzeroberfläche für einen Roboter, der mit LEGO® baut.

Eingereicht von:

Christina Franziska Astleithner

Irene Stöger

Betreut durch:

DI Gerhard Zweimüller

In Kooperation mit:

Wolfgang Schinnerl

RK Automatisierungstechnik GmbH

eingereicht an der HTL-Perg für Informatik am 13. Mai 2015

1 Inhaltsverzeichnis

Inhalt

2 Eidesstaatliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Diplomarbeit selbständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer andern Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht. Sie wurde eigens im Rahmen der Diplomarbeit der HTL-Perg während des Schuljahres 2014/2015 erstellt.

Ich bin mir bewusst, dass eine falsche Erklärung rechtliche Folgen haben wird.

Astleithner Christina	*Stöger Irene*

Ort, Datum

3 Einleitung

3.1 Kurzbeschreibung

Das Projekt Legonator entstand aus dem Wunsch, eine Diplomarbeit im Rahmen der Matura an der HTL Perg für Informatik abzuwickeln. Dabei stellte sich das Unternehmen RK Automatisierungstechnik GmbH als Auftraggeber an unsere Seite und kam mit dem Auftrag eines Messestandes, um diesen dann zu repräsentativen Zwecken verwenden zu können.

Der Roboterarm, der für Messen mitgenommen werden soll, sollte dann in der Lage sein, den Schriftzug der Firma aus Legosteinen zu legen. Da ergibt sich aber das Problem, dass Roboter in der Regel unhandlich sind, da sie meist viel Gewicht und Sperrigkeit mit sich bringen. Außerdem benötigen sie viele zusätzliche Komponenten, um sie betriebsbereit zu machen. Nun kann es aber auch vorkommen, dass nur ein Repräsentant auf eine Messe fährt und somit nicht in der Lage ist, einen riesigen Roboter mitzunehmen.

Daher soll anhand eines kleineren Roboterarms, der in einer Zelle verbaut wird, das ganze Konstrukt überschaubarer gestaltet werden. Hier kommt unsere Diplomarbeit ins Spiel.

Unsere Aufgabe war es, sowohl den Mikrocontroller, als auch das Display zu programmieren und somit den Roboter von einer kleineren Bedienoberfläche aus steuern zu können. Weiters sollte der Display in der Lage sein, auf Signale der Sensoren zu achten, wenn beispielsweise die Legosteine zur Neige gehen. Der Mikrocontroller soll dann in weiterer Folge Signale an den Roboter senden, um seine Tätigkeit zu unterbrechen.

Also kurz gesagt, unsere Diplomarbeit befasst sich mit der Programmierung eines Mikrocontrollers und dieser Vorgang wird auf den folgenden Seiten noch detaillierter ausgeführt.

3.2 Abstract

The Legonator project emerged from our wish, to do a diploma thesis in order to graduate from the HTL for computer science in Perg, Austria. To achieve this goal, the company RK Automatisierungstechnik GmbH supported us as the client. They charged us with contributing to the creation of a fair stand which should be used to represent their company.

The robotic arm that the company plans to use on company fairs, should later be able to reproduce the company's logo with Lego bricks. Unfortunately such robots are usually unwieldy because of their weight and bulkiness. Aside from that they need additional components to work properly. And sometimes only one company agent takes part in the fair and one person alone does not easily take such a big, unwieldy robot plus a lots of additional components with him.

The solution to this problem is building a mobile cell containing both the robot and all the components needed. We contribute to this goal with our diploma project.

Our task was to program a user interface on a small touch display to interact with the robot. Therefore we use a microcontroller. Additionally the display should be able to react to signals it gets from external sensors. These sensors measure the state of the robot's environment. For example there's a sensor which counts the amount of on-hand Lego bricks. The microcontroller should also be able to send such signals to the robot and to adapt its behaviour due to the gotten signals.

To sum it our diploma thesis deals with programming the microcontroller to provide a user interface for the robot. This process is explained in detail on the following pages.

3.3Team

3.3.1 Christina Franziska Astleithner

Abb. 1 Astleithner Christina

Kontakt

Ober St. Georgen 22
4372 St. Georgen am Walde
☎ 0680 32 530 32
✉ christina.astleithner@gmail.com

Persönliche Angaben

Geburtsdatum: 13.08.1996 in Amstetten
Religionsbekenntnis: röm.-kath.

Familie

Josef Astleithner, Unternehmer (Holzschlägerung und –rückung)
Monika Astleithner, Landwirtin
3 Geschwister

Schulische Ausbildung

2002 – 2006 Volksschule St. Georgen am Walde
2006 – 2010 Hauptschule St. Georgen am Walde
2010 – 2015 HTL für Informatik in Perg

Berufspraxis

Juli 2013 Praktikum bei Atos IT Solutions und Services GmbH, Linz
August 2013 Praktikum bei MIC Datenverarbeitung GmbH, Linz

Interessen

Musizieren (Querflöte, Gitarre)
Lesen
Geschichten schreiben
Zeichnen
digitale Grafiken

3.3.2 Irene Stöger

Kontakt

Neumühlstraße 27
4284 Tragwein
☎ 0681 107 513 16
✉ stoeger.irene@gmx.at

Abb. 2 Irene Stöger

Persönliche Angaben

Geburtsdatum: 03.02.1996 in Linz
Religionsbekenntnis: röm.-kath.

Familie

Johann Stöger, Layouter
Doris Stöger, Einzelhandelskauffrau
1 Geschwister

Schulische Ausbildung

2002 – 2006 Volksschule Tragwein
2006 – 2010 Hauptschule Tragwein
2010 – 2015 HTL für Informatik in Perg

Berufspraxis

Sommer 2012 Praktikum bei Engel Austria GmbH, Schwertberg
Sommer 2014 Praktikum bei technosert electronic GmbH, Wartberg/Aist

Interessen

Tae Kwon Do
Musik hören
Computer und Internet
Freunde treffen

3.4Projektumfeld

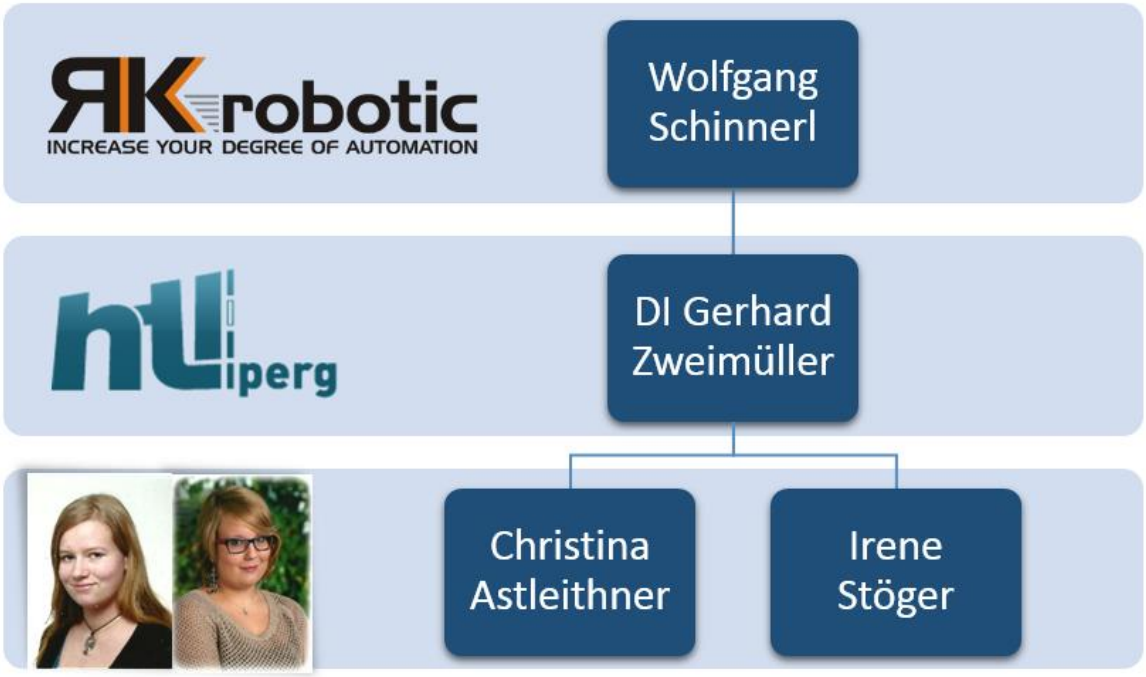


Abb. 3 Projektumfeld, Hierarchie

Der Auftraggeber unserer Diplomarbeit ist das Unternehmen RK Automatisierungstechnik GmbH, wo uns Herr Wolfgang Schinnerl als Ansprechpartner zur Verfügung steht.

Da die Diplomarbeit im Rahmen der Matura an der HTL Perg abgewickelt wird, steht uns seitens der Schule Herr Prof. Dipl.-Ing. Gerhard Zweimüller mit Rat und Tat als Betreuungslehrer zur Seite.

Die beiden Diplomanden, die das Projekt durchführen, sind Christina Astleithner und Irene Stöger. Sie sind auch Verfasser dieser Ausarbeitung.

3.4.1 Schule

Abb. 4 HTL-Perg von außen

Kontakt

HTBLA Perg
Machlandstraße 48
4320 Perg
☎ + 43 (0) 7262 539 26
🖷 + 43 (0) 7262 539 26 – 6
🖱 www.htl-perg.ac.at

Der Ausbildungsschwerpunkt der HTL Perg liegt im Bereich der Informatik. Das heißt, man beschäftigt sich intensiv mit der Softwareentwicklung, wobei in dieser Schule auch wirtschaftliche Aspekte berücksichtigt werden.

Ebenfalls wird eine umfassende Allgemeinbildung geboten.
Durch das 8 wöchige Praktikum sowie den zahlreichen Professoren aus der „echten Welt" wird eine praxisnahe Ausbildung gewährleistet.

3.4.2 Betreuungslehrer

Kontakt

Gerhard Zweimüller
✉ g.zweimueller@htl-perg.ac.at

Herr Prof. Zweimüller begleitet uns schon seit der zweiten Klasse im Gegenstand „Netzwerksysteme und Verteilte Systeme". Da auch unsere Diplomarbeit sehr stark in die Richtung der Kommunikation zwischen verschiedenen Komponenten geht, fällt sie diesem Fach zu.

Er steht uns auch regelmäßig am Freitag nach der Schule zur Verfügung. Zum einen hilft er uns weiter, wenn wir auf scheinbar unlösbare Probleme stoßen, und zum anderen geht er seiner Aufgabe als Betreuungslehrer nach, nämlich sich über unsere laufenden Fortschritte zu informieren.

Wir möchten uns gerne bei Prof. Zweimüller für sein Engagement und seine Hilfe über die Dauer unserer Diplomarbeit bedanken.

3.4.3 Auftraggeber

3.4.3.1 RK Automatisierungstechnik Gmbh

Kontakt

RK Automatisierungstechnik GmbH
Buchenstraße 4
4230 Pregarten
☎ + 43 (0) 7236 314 16
🖷 + 43 (0) 7262 630 48
🖱 www.rkrobotic.at

Abb. 5 RK Automatisierungstechnik GmbH Logo

Das Unternehmen RK Automatisierungstechnik GmbH bietet speziell auf den Kunden abgestimmte, individuelle Automatisierungslösungen, welche in enger Zusammenarbeit mit den Projektteams des Kunden entstehen. Die wichtigsten Schwerpunkte sind die Inbetriebnahme, Roboterintegration und –optimierung.

3.4.3.2 Wolfgang Schinnerl

Kontakt

Wolfgang Schinnerl
Plesching 13
4040 Steyregg
✉ wolfgangschinnerl@gmx.at

Wolfgang Schinnerl stellte die Schnittstelle zum Auftraggeber dar. An ihn konnten wir uns wenden, sobald wir Informationen benötigten. Er war während der gesamten Dauer unserer Arbeit für uns beinahe pausenlos erreichbar und hat sich sehr für unser Projekt engagiert.

Dafür möchten wir ihm an dieser Stelle noch einmal für alles, was er für uns getan hat, danken.

3.5 Projektauftrag

3.5.1 Die Idee

3.5.1.1 Problemstellung

Das Unternehmen RK-Automatisierungstechnik GmbH hat nach einem einfachen Weg gesucht, bei öffentlichen Veranstaltungen, wie etwa Berufs- oder Firmenmessen, einen bleibenden Eindruck auf die Veranstaltungsbesucher zu hinterlassen. Durch Beobachtung von anderen Unternehmen kommt man schnell auf die Idee, den Messebesuchern eine Art „Kostprobe" der Unternehmenstätigkeit zur Verfügung zu stellen. So dekorieren etwa Floristen ihren Messestand mit den schönsten Blumengestecken, während Bäckereien kleine Leckereien an die Besucher verteilen. Auf diese Weise erhält das Publikum einen schnellen, praktischen Einblick ins Firmengeschehen. Leider stellt sich diese Lösung bei einem Unternehmen, das in der Automatisierungstechnik tätig ist, als nur schwer umsetzbar heraus.

Das eine Problem ist, dass die Industrieroboter, mit denen die RK-Automatisierungstechnik GmbH arbeitet, meist sehr unhandlich und schwer zu transportieren sind. Außerdem sind sie in der Praxis Teil eines größeren Systems und können in den meisten Fällen nur durch die Zusammenarbeit mit zusätzlichen Komponenten ihre volle Funktion erfüllen. Der Anschluss und Transport all dieser Komponenten stellt natürlich wieder zusätzlichen Aufwand dar.

Ein anderes Problem ist die Funktion des Roboters selbst. Automatisiert werden meistens nur einzelne Tätigkeiten wie etwa ein einzelner Verarbeitungsschritt eines Produktes am Fließband. Ein Industrieroboter alleine ist im Normalfall nicht für die Verarbeitung eines ganzen Produktes zuständig. Stattdessen werden mehrere Roboter hintereinander geschaltet, die alle einen einzigen Arbeitsschritt erledigen. Folglich ist es schwer, eine Funktion für den Roboter zu finden, die auf einem Messestand das Interesse von Besuchern weckt und nebenbei auch noch einen bleibenden Eindruck hinterlässt.

Wieder ein anderes Problem ist die Bedienung des Roboters, da die Steuerung eines einzelnen Industrieroboters meistens manuell erfolgt und Laien meist nicht gut damit umgehen können.

3.5.1.2 Lösung

Die Lösung des Problems stellt eine transportable Zelle dar, in der ein Yaskawa-Motoman und alle für seine Funktion benötigten Komponenten bereits fertig verbaut und verkabelt sind. So dauert der Aufbau einer derartigen Installation an einem Veranstaltungsort nur mehr wenige Minuten, da die Zelle als Ganzes von einem Ort zum anderen gebracht werden kann. Durch die Mobilität werden außerdem weniger Personen für den Aufbau des Messestands benötigt.

Um nun das Publikum für sich zu gewinnen, soll der Roboter live das Firmenlogo mittels Lego®steinen auf eine Lego®setzplatte aufbauen. Die fertig bestückten Setzplatten können dann von Veranstaltungsbesuchern mit nach Hause genommen werden, damit diese sich auch später noch an die Firma erinnern können.

Die intuitive Bedienung des Roboters soll über eine graphische Benutzeroberfläche erfolgen.

3.5.1.3 Abgrenzung Diplomarbeit – Was wir dazu beitragen

Es soll eine graphische Benutzeroberfläche zur Verfügung gestellt werden, über die der Motoman sowohl von den Vortragenden als auch von den Messebesuchern bedient werden kann. Auch diese wird später in der Zelle verbaut. Kontrolliert wird die Oberfläche über einen Mikrocontroller, der die zentrale Steuereinheit des Systems repräsentiert und später auch die Kommunikation zwischen GUI, Sensorik und Motoman übernehmen soll. Aufgabe der Diplomarbeit ist nun einerseits die Programmierung dieser Benutzeroberfläche auf dem Mikrocontroller und andererseits das Auffinden von Lösungen für die Kommunikation zwischen Industrieroboter und dem Mikrocontroller selbst über das Profibus®-Protokoll.

3.5.2 Zusammenfassung Aufgaben

Im Folgenden sind alle Arbeiten, die im Rahmen der Diplomarbeit zu erledigen waren, noch einmal grob zusammengefasst:

- Recherche und Auswahl einer geeigneten Steuerplatine, die als zentrale Steuereinheit für den Roboter, die Sensorik und die Benutzeroberfläche dient
- Anbindung der Komponenten, welche die Benutzeroberfläche darstellen, an den Mikrocontroller
- Herausarbeiten von Lösungsansätzen für die Kommunikation von Mikrocontroller und Roboter via Profibus®
- Recherche über Profibus® allgemein und Erstellen einer Abhandlung darüber
- Entwurf und Programmierung einer graphischen Benutzeroberfläche
- Entwurf und Programmierung der Bedienerführung im Touchpanel
- Dokumentation zur entwickelten Software

4 Entstehung

4.1 Ausgangslage und Zielsetzung

4.1.1 Ausgangslage

Die RK Automatisierungstechnik GmbH will sich in Zukunft auf Messeständen besser präsentieren können. Um dies zu erreichen, wird ein Industrieroboter gemeinsam mit allen für seine Funktion benötigten Komponenten in eine mobile Zelle verbaut. Der Roboter soll dann dazu in der Lage sein, mit Legosteinen das Firmenlogo auf eine Legosetzplatte zu bauen.

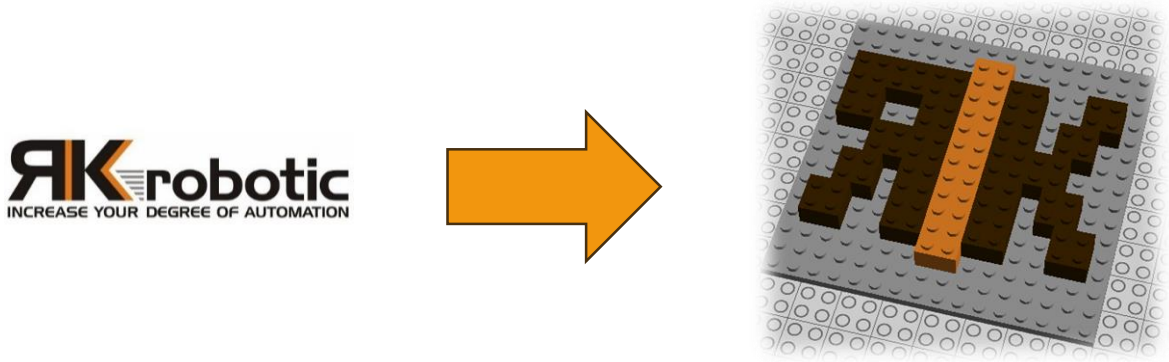


Abb. 6 Aufbau des Logos mit Legosteinen

Die Zufuhr neuer Legosteine und Setzplatten soll über eine Art Schublade erfolgen. Durch Sensoren soll der Roboter außerdem erkennen können, ob noch Bausteine oder Setzplatten vorhanden sind, oder ob neue zugeführt werden müssen. In diesem Fall setzen die Sensoren ein Signal ab.

Die Bedienung des Roboters erfolgt über eine graphische Benutzeroberfläche mit einem Touchpanel, über das Benutzereingaben getätigt werden können. Über die GUI werden Befehle auf den Roboter abgesetzt und Fehlermeldungen seitens des Roboters ausgegeben.

Um alle benötigten Komponenten miteinander zu verbinden, soll ein Mikrocontroller als zentrale Steuereinheit fungieren. Dieser ist für folgende Aufgaben zuständig:

- Verarbeitung und Weitergabe von Signalen der Sensoren an den Roboter
- Darstellung der Benutzeroberfläche auf einem Display
- Verarbeitung von Benutzereingaben

- Weiterleitung von Befehlen an den Roboter
- Weitergabe von Fehler- und Statusmeldungen an den Roboter

Für die Kommunikation zwischen Mikrocontroller und Roboter soll das Profibus®-Protokoll verwendet werden.

Der Industrieroboter steht bereits zur Verfügung. Es ist ein Yaskawa Motoman® und ist bisher vom Auftraggeber zu Schulungszwecken verwendet worden. Für die anderen Komponenten soll noch eine passende Hardware ausgesucht werden.

4.1.2 Systemübersicht

Die folgende Darstellung gibt einen Überblick über das Zusammenspiel der Komponenten.

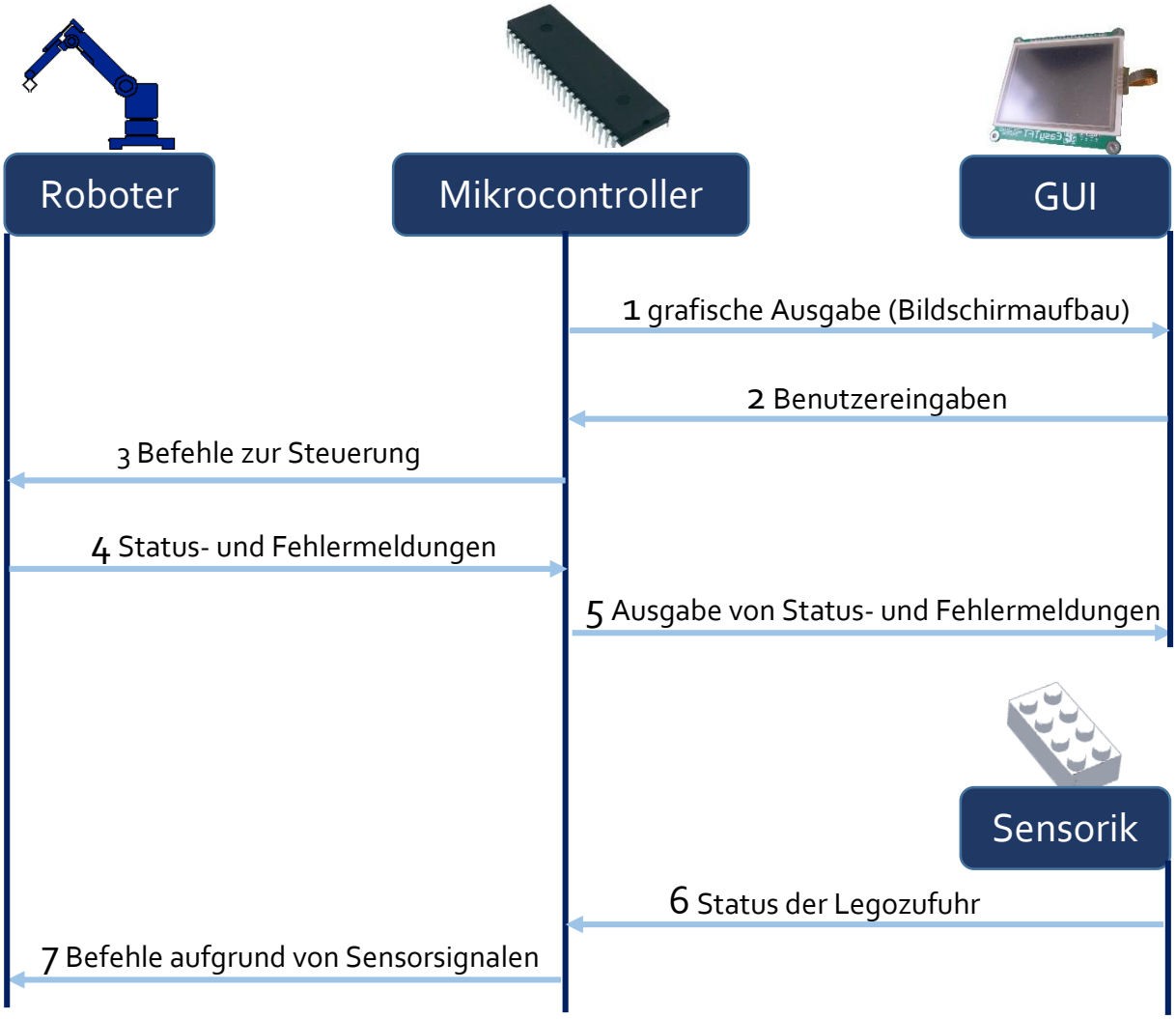


Abb. 7 Übersicht über Kommunikation

1. **grafische Ausgabe (Bildschirmaufbau)**
   Der Mikrocontroller ist für den Aufbau und das Layout aller grafischen Elemente (Buttons, Bilder, Textfelder, …) zuständig, die auf der GUI angezeigt werden. Er bestimmt, welche Elemente zu welcher Zeit dargestellt werden.
2. **Benutzereingaben**
   Über ein Touchpanel kann der Benutzer die GUI bedienen. Bei einer Berührung des Panels werden die Koordinaten der Berührung an den Mikrocontroller gesendet, der diese, je nachdem, welche Ansicht momentan auf der GUI aktiv ist, auswertet.
3. **Befehle zur Steuerung**
   Der Mikrocontroller startet und beendet das Roboterprogramm, das die Roboterachsen mechanisch bewegt und dadurch Lego® baut. Der Ablauf des Programms kann durch Benutzereingaben auf der GUI, die vom Mikrocontroller ausgewertet werden, beeinflusst werden.
4. **Status- und Fehlermeldungen**
   Der Roboter ist in der Lage, Status- und Fehlermeldungen abzusetzen. Diese werden an den Mikrocontroller gesendet und davon verarbeitet.
5. **Ausgabe von Status- und Fehlermeldungen**
   Die vom Roboter gesendeten Status- und Fehlermeldungen werden auf der GUI ausgegeben.
6. **Status der Legozufuhr**
   Für den Fall, dass die Bausteine ausgehen oder die Legozufuhr auf eine andere Weise ins Stocken gerät, setzt der Sensor ein Signal an den Mikrocontroller ab.
7. **Befehle aufgrund von Sensorsignalen**
   Der Mikrocontroller verarbeitet und interpretiert die Sensor-Signale. Aufgrund der Ergebnisse teilt er dem Roboter mit, wie dieser dann auf die Veränderungen in seiner Umwelt reagieren soll.

4.1.3 Komponentenübersicht

Für die Realisierung des Systems werden, wie aus der vorhergehenden Grafik ersichtlich ist, verschiedene elektronische Komponenten benötigt. Diese sind hier kurz aufgelistet. Die konkreten Hardwarekomponenten werden in einem späteren Kapitel **5.1 Hardware** noch einmal genauer erläutert.

4.1.3.1 Roboter

Ein Industrieroboter der Marke Yaskawa Motoman® soll später mithilfe eines Greifers Lego® bauen können. Besagter Roboter befindet sich schon seit einiger Zeit im Besitz des Auftraggebers und ist bisher für Schulungszwecke verwendet worden.

4.1.3.2 Steuerplatine

Als zentrale Steuereinheit wird eine Platine verwendet, die einen Mikrocontroller beherbergt. Sie dient als Herzstück des Systems und verbindet sämtliche Komponenten miteinander. Dafür benötigt sie dementsprechende Schnittstellen, damit alle Komponenten angeschlossen werden können.

4.1.3.3 Display

Auf diesem Display wird die graphische Benutzeroberfläche ausgegeben. Hier können Benutzer Befehle an den Roboter absetzen und sich Status- und Fehlermeldungen seitens des Roboters ansehen.

4.1.3.4 Touchpanel

Über das Display wird ein Touchpanel gelegt, damit Benutzereingaben bequem und einfach durch Touch-Technologie erfolgen können. Dies trägt vor allem dazu bei, dass die Steuerung so intuitiv wie möglich bleibt.

4.1.3.5 physikalische Verbinder

Um alle Komponenten miteinander zu verbinden, also sie einerseits physikalisch an denselben Stromkreis anzuschließen und andererseits eine logische Kommunikationsmöglichkeit für die elektronischen Komponenten herzustellen, werden natürlich viele verschiedene Kabel und Schnittstellen benötigt.

4.1.4 Aufgaben und Zuständigkeiten

In diesem Kapital werden alle im Rahmen des Projektes zu erledigenden Aufgaben aufgeführt. Weiters wird hier erläutert, bei welchen Personen bzw. Organisationen die Zuständigkeiten liegen. Kurz: „Wer ist wofür verantwortlich?".
Als Ressourcen stehen einerseits die zwei Personen des Diplomanden-Teams und andererseits der Auftraggeber, also die Organisation RK Automatisierungstechnik GmbH, zur Verfügung.

RK = Auftraggeber, CA = Christina Astleithner, IS = Irene Stöger

Aufgabe	Zuständiger
Entwurf und Aufbau der mobilen Roboterzelle	RK
Schreiben des Roboterprogramms zum Bauen mit Lego®steinen	RK
Installieren der Sensorik	RK
Physikalische Verbindung zwischen allen Komponenten herstellen	RK
Recherche und Auswahl einer geeigneten Steuerplatine + Bildschirm für die Benutzeroberfläche	CA
Anbindung Display & Zusatzmodule an den Mikrocontroller	CA, IS
Heraussuchen von Lösungsansätzen für die Kommunikation zwischen Mikrocontroller & Roboter über Profibus®	IS
Programmierung der Benutzeroberfläche	CA
Dokumentation der entwickelten Software	CA
Touch-Bedienung programmieren	CA
Schreiben einer Ausarbeitung über das Profibus®-Protokoll	IS

Tabelle 1 Zuständigkeiten

4.1.5 Zielsetzung

„Erstens kommt es anders, und zweitens als man denkt." – WILHELM BUSCH

Zu Projektbeginn wurde eine Reihe von Aufgaben definiert, die das Diplomarbeitsteam zu erfüllen hatte.
Im Laufe der Diplomarbeit hat sich jedoch herausgestellt, dass ein gewisses Ziel, nämlich die Ansteuerung des Roboters über das Profibus®-Protokoll, in gegebener Zeit und mit gegebenen Ressourcen nicht zu erfüllen war. Dieses Problem wurde erst bekannt, als sich die beiden Diplomanden im Zuge von Recherchen mit dem Protokoll beschäftigten.

Die Hauptproblematik im Umgang mit Profibus® besteht darin, dass das Profibus®-Protokoll bis vor kurzem von drei Patenten (von Siemens und Endress+Hauser) geschützt war. Dementsprechend war eine offene Implementierung des Protokolls aus rechtlicher Sicht nicht möglich. Folglich gibt es keine ausgereiften, freien Implementierungen des Busses, die das Diplomarbeitsteam für die Roboteransteuerung verwenden kann. Mehr Informationen zur Profibus®-Problematik können im Kapitel **7.1 Problematik** gefunden werden.

Alternativ dazu könnten zwar bereits fertige Lösungen (z.B. Siemens S7) in Einsatz gebracht werden, allerdings würden dadurch sehr hohe Kosten entstehen. Deshalb haben sich die beiden Diplomanden gemeinsam mit dem Auftraggeber darüber geeinigt, die ursprüngliche Zielsetzung der Diplomarbeit im Sinne des Auftraggebers abzuändern.

4.1.5.1 Ursprüngliche Zielsetzung

Zu Projektbeginn wurden folgende Aufgaben definiert, die im Rahmen der Diplomarbeit zu erledigen wären:

- Recherche und Auswahl einer geeigneten Steuerplatine, die als zentrale Steuereinheit für den Roboter, die Sensorik und die Benutzeroberfläche dient
- Recherche und Auswahl eines geeigneten Displays zur Realisierung der Benutzeroberfläche
- Anbindung der Zusatzmodule an den Mikrocontroller (Touchpanel, Display, Profibusschnittstelle)

- Programmierung der Datenkommunikation zwischen Mikrocontroller und Roboter (RS485, Profibus)
- Entwurf der Mikrocontroller Programmstruktur
- Programmierung des Steuerablaufs
- Entwurf und Programmierung der Bedienerführung im Touchpanel
- Test und Inbetriebnahme der entwickelten Mikrocontroller-Software
- Dokumentation zur entwickelten Software

4.1.5.2 Tatsächliche Zielsetzung

Im Laufe der Diplomarbeit wurden die zu erledigenden Aufgaben dahingehend abgeändert:

- Recherche und Auswahl einer geeigneten Steuerplatine, die als zentrale Steuereinheit für den Roboter, die Sensorik und die Benutzeroberfläche dient
- Recherche und Auswahl eines geeigneten Displays zur Realisierung der Benutzeroberfläche
- Anbindung der Zusatzmodule an den Mikrocontroller (Touchpanel, Display)
- Herausarbeiten von möglichen Lösungsansätzen für die Kommunikation zwischen Mikrocontroller und Roboter via Profibus
- Recherche über Profibus allgemein und Erstellen einer Abhandlung darüber
- Entwurf und Programmierung einer graphischen Benutzeroberfläche
- Entwurf und Programmierung der Bedienerführung im Touchpanel
- Dokumentation zur entwickelten Software

5 Ressourcenplanung

5.1Hardware

5.1.1 Yaskawa Motoman® Industrieroboter

Dieser blaue Industrieroboter befindet sich im Besitz des Auftraggebers und wurde bisher für Schulungszwecke eingesetzt.

Abb. 8 Diplomandin mit Roboter

Abb. 9 Elektronik Roboter

Die Stromversorgung erfolgt über ein Netzteil, das in dem grauen Kasten verbaut ist, der auf Abbildung *Abb. 9 Elektronik Roboter* zu sehen ist. Dieser beherbergt auch sämtliche Schnittstellen zum Roboter sowie alle elektronischen Komponenten, die zu dessen Betrieb notwendig sind. Demnach ist hier auch ein Profibusmodul für die Kommunikation nach außen hin verbaut.

Manuell kann der Roboter über ein Interface-Panel bedient werden (siehe Abbildung *Abb. 10 Interface Panel Roboter*). Damit kann man zum Beispiel die Roboterachsen bewegen. Für Laien ist die Benutzeroberfläche dieses Panels leider nicht einfach zu bedienen.

Abb. 10 Interface Panel Roboter

Der Industrieroboter verfügt über 7 verschiedene Achsen, die sich unabhängig voneinander bewegen lassen, wie es mit der folgenden Abbildung dargestellt wird:

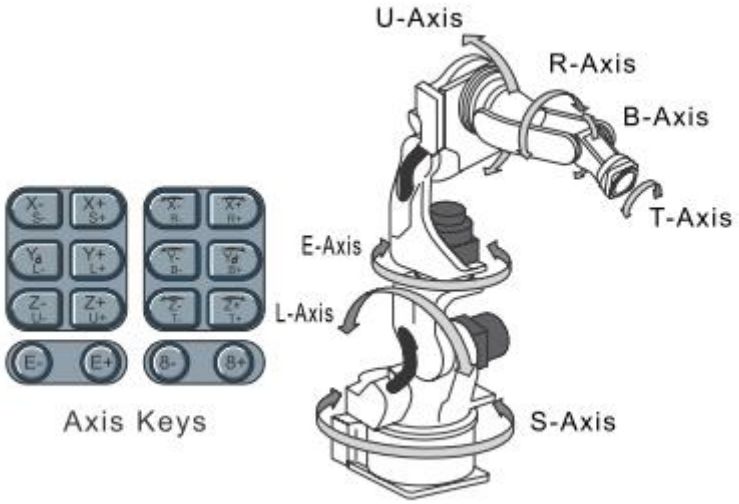


Abb. 11 Roboter Achsen

5.1.2 EasyAVR™ v7

Das EasyAVR™ v7 ist ein Entwicklungsboard des Herstellers MikroElektronika. Es wurde vor allem wegen seiner hohen Konnektivität gewählt, da es als Herzstück der Konstruktion dient und somit Schnittstellen für alle verwendeten Komponenten aufweisen soll, um diese miteinander zu verbinden. Geliefert wird es mit einem ATmega 32-Mikrocontroller. Ein großer Vorteil dieses Boards ist auch, dass Komponenten jederzeit ausgetauscht werden können. So beherbergt es zum Beispiel mehrere Slots für verschiedenste Mikrocontroller.
Im Folgenden werden alle Komponenten des Boards, mit denen wir uns beschäftigt haben, genauer erläutert.

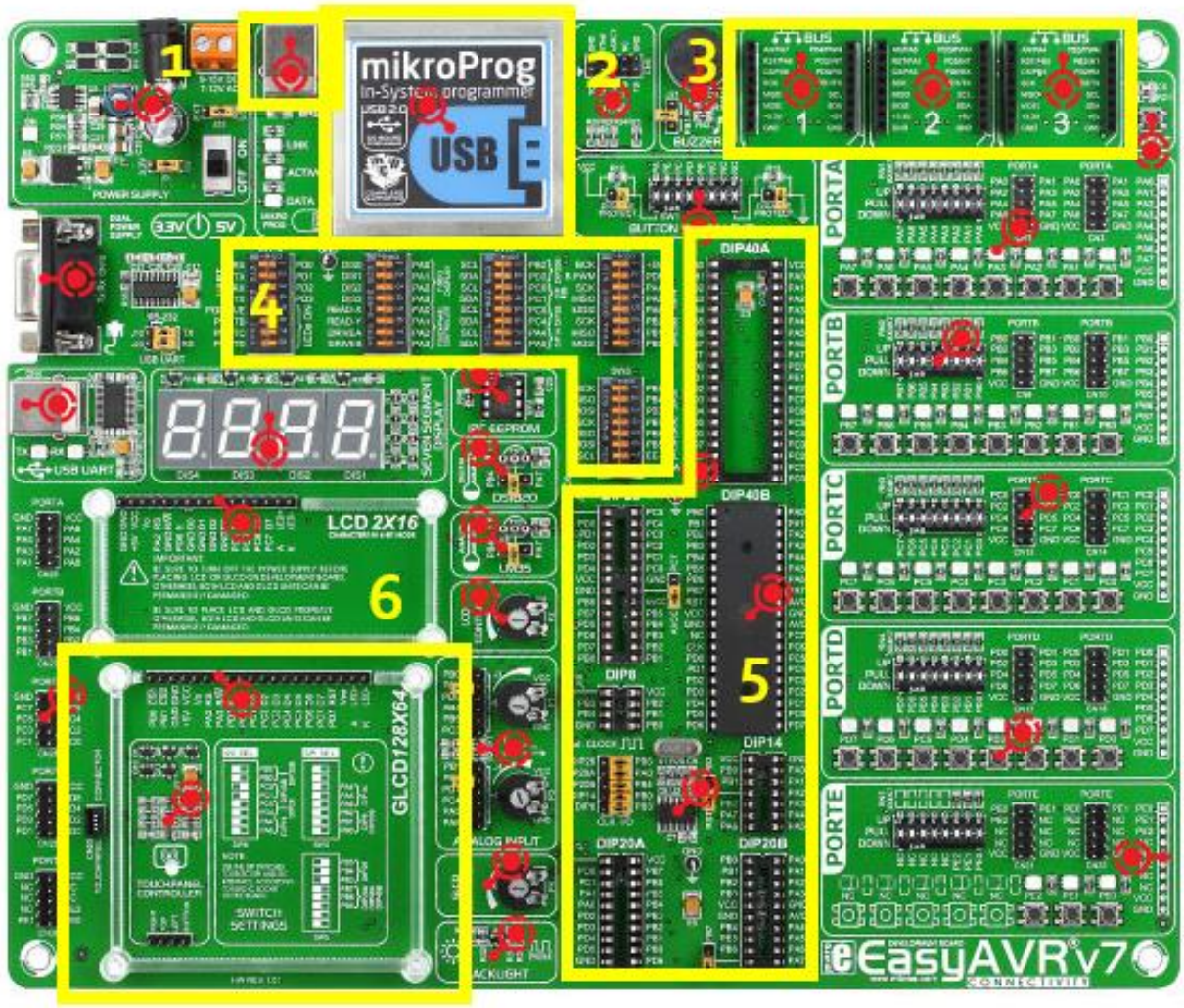


Abb. 12 EasyAvr Komponenten

1 Stromversorgung/USB-Verbindung

Über diese Schnittstelle kann das Board mithilfe eines USB-Kabels an einen Rechner oder an eine andere Stromquelle angeschlossen werden. Sie bietet eine von zwei Möglichkeiten, wie das Entwicklungsboard mit Strom versorgt werden kann. Gleichzeitig dient diese Schnittstelle dazu, sich mit einem Rechner zu verbinden, der daraufhin ein Programm auf den Mikrocontroller schreiben bzw. vom Mikrocontroller lesen kann.

2 mikroProg™ on-board-Programmer

Dieser dient zur Programmierung des Mikrocontrollers auf dem Entwicklerboard. mikroProg™ ist eine Technologie, die unter anderem die Programmierung von ungefähr 65 verschiedenen AVR® Mikrocontrollern von Atmel® über USB 2.0 erlaubt. Somit kann der Mikrocontroller, der sich auf dem Board befindet, jederzeit ausgetauscht werden, ohne dass am Entwicklungsrechner etwas an der Software-Konfiguration geändert werden muss. Außerdem erleichtert dieser on-board-Programmer die Verbindungsherstellung mit dem Rechner ungemein. Dieser benötigt nur mehr entsprechende USB-Treiber und die AVRFlash-Software (siehe Kapitel **5.2.2 AVRFlash**) zur Benutzung von mikroProg™.

3 mikroBUS™ Sockets

Der mikroBUS™ ist ein relativ neuer Standard von MikroElektronika. Diese Technologie ermöglicht es, zusätzliche Boards, sogenannte Click-Boards, via Plug-and-Play anzustecken bzw. auszustecken, ohne etwas an der Hardware-Konfiguration ändern zu müssen. Diese Sockets werden für die beiden Click-Boards OPTO click und RS485 click verwendet (siehe Kapitel **5.1.8 RS485** und **5.1.9 OPTO click**).

4 Dip-Schalter

Diese Schalter ermöglichen es, verschiedene Hardwarekomponenten des Boards hinzu- bzw. wegzuschalten. Damit kann die Hardwarekonfiguration jederzeit durch Betätigen eines Schalters verändert werden. In unserem Fall sind die notwendigen Schalter für Hintergrundlicht für das TFT-Display sowie für das Einlesen von Koordinaten vom Touchpanel aktiv, wie die folgende Abbildung zeigt:

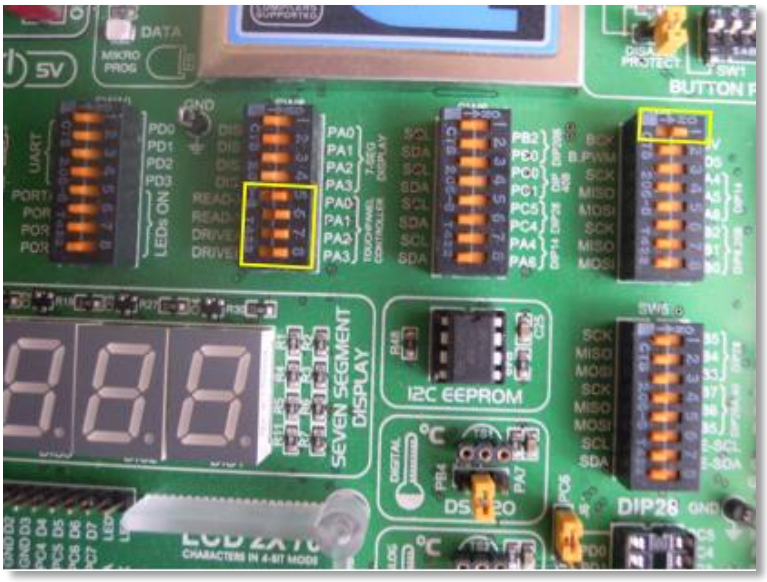


Abb. 13 Dip-Switch-Einstellungen

5 Mikrocontroller-Sockets

Wie der *Abb. 12 EasyAvr Komponenten* zu entnehmen ist, befinden sich hier verschiedenste Schnittstellen für verschiedenste Mikrocontroller. Für den ATmega 1284P steht uns beispielsweise der Socket DIP40B zur Verfügung, der die richtige Anzahl an Pins für diese Art von Mikrocontroller bereitstellt. Dank der mikroProg™-Technologie lassen sich die Mikrocontroller jederzeit austauschen ohne großartig etwas an der aktuellen Konfiguration ändern zu müssen.

6 GLCD-Socket

Dieser Socket bietet alle notwendigen Konnektoren für ein GLCD-Display und kann auch für ein TFT-Display verwendet werden. Um das Display zu betreiben müssen die entsprechenden Dip-Schalter richtig eingestellt werden (siehe oben). Links neben dem GLCD-Socket befindet sich außerdem ein Anschluss für ein Touchpanel.

5.1.2.1 Alternativen zu EasyAVR™ v7

Eine Aufgabe dieser Diplomarbeit war die Recherche geeigneter Hardware-Komponenten. Bei der Wahl der Steuerplatine haben wir uns gemeinsam mit dem Auftraggeber letztendlich für das EasyAVR™ entschieden. Neben diesem standen noch zwei weitere Platinen zur Auswahl:

UNI-DS6

Das UNI-DS6 ist ein Entwicklungsboard von MikroElektronika, das Mikrocontroller von verschiedenen Herstellen unterstützt.

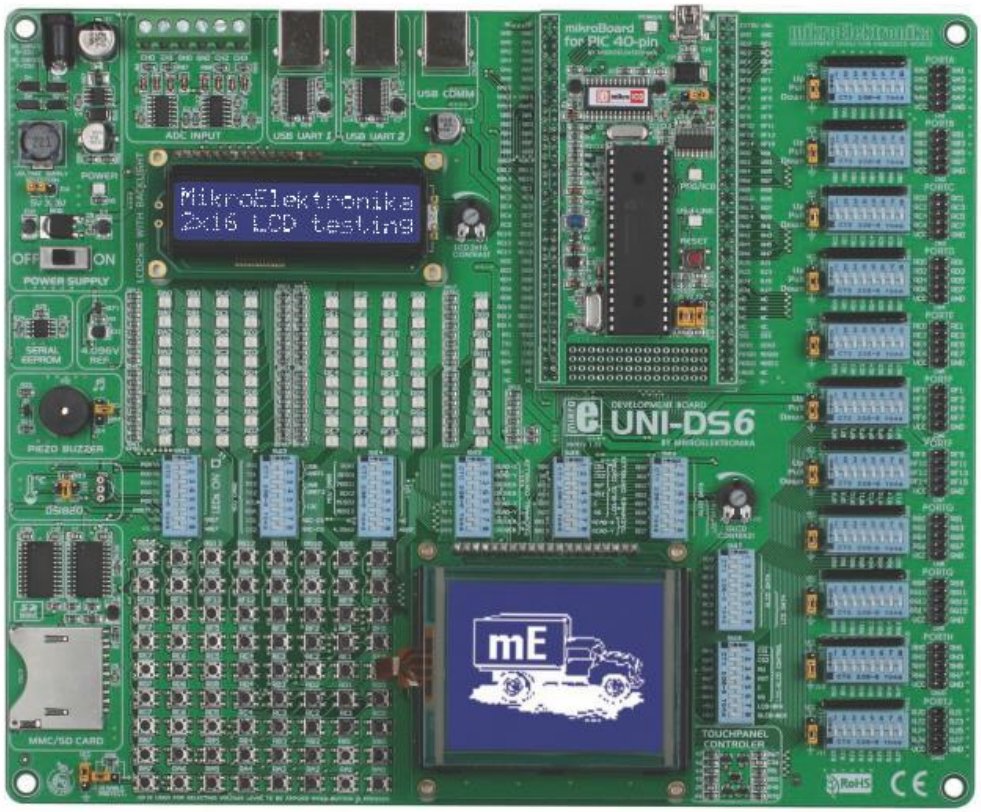


Abb. 14 UNI-DS6

Im Vergleich zum EasyAVR™ befinden sich die Mikrocontroller-Sockets für die Unterstützung verschiedener Mikrocontroller nicht direkt am Board, sondern werden durch den mikroBoard-Socket realisiert. Auf diesem wird ein Mikroboard platziert, welches unter anderem den Mikrocontroller und einen Built-in-Programmer enthält. Dies ist einerseits ein Vorteil, da eine hohe Konnektivität gewährt wird, und andererseits ein Nachteil, da für jeden Mikrocontroller ein mikroBoard gekauft werden muss.
Das Board beherbergt außerdem drei USB-UART-Module, ein A/D-Konverter-Modul, ein EEPROM-Modul, einen GLCD-Socket und vieles mehr.

BIGdsPIC6

Das BIGdsPIC6 ist ebenfalls ein Entwicklungsboard von MikroElektronika.

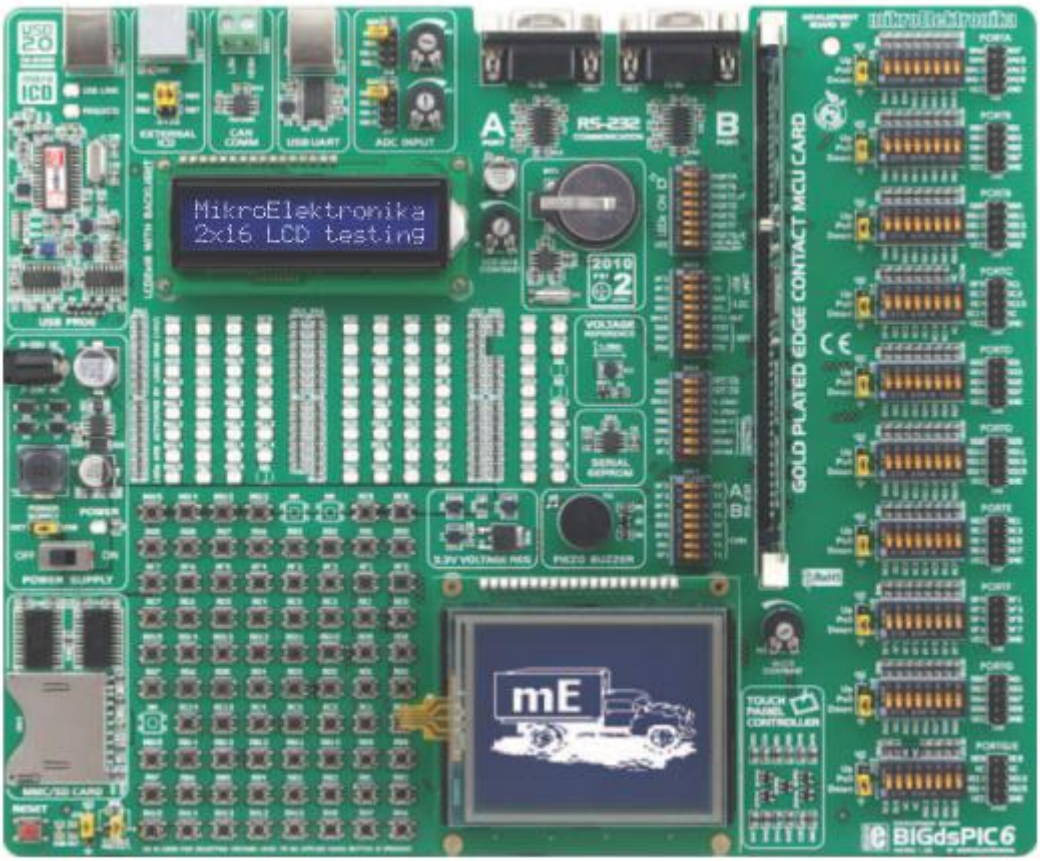


Abb. 15 BIGdsPIC6

Auf dem Board befindet sich ein Konnektor für eine MCU-Card, also eine Steckplatine mit einem Mikrocontroller. Leider werden bei diesem Board nur Mikrocontroller der dsPIC-Familie von Microchip unterstützt. Wie die beiden anderen Entwicklungsboards hat auch diesen ein EEPROM-Modul, ein A/D-Konverter-Modul und einen GLCD-Socket. Programmiert wird der Mikrocontroller hier mithilfe eines dsPICFlash-Programmers. Selbstverständlich steht hierfür eine USB-Schnittstelle zur Verfügung.

5.1.3 Atmel® ATmega32

Der ATmega32 ist ein 8-bit-Mikrocontroller des Herstellers Atmel® und gehört der AVR®-Familie an. Er ist bereits im Lieferumfang des EasyAVR™ enthalten und wurde anfangs als zentrale Steuereinheit eingesetzt, ehe er gegen den Atmel® ATmega 1284P ausgetauscht wurde. Er wies für gegebene Anforderungen zu wenig Arbeits- und Programmspeicher auf.

Abb. 16 Atmega32

5.1.3.1 Funktionalitäten

Der ATmega32 verfügt über folgende Funktionalitäten:

- 32 KB Flash-Programmspeicher
- 1024 Bytes EEPROM
- 2 KB SRAM
- 32 Arbeitsregister für generelle Zwecke
- 32 I/O-Leitungen für generelle Zwecke
- ein JTAG-Interface
- On-Chip Debugging Support und Programmierung
- 3 flexible Timer
- Interne und externe Interrupts
- seriell programmierbares USART
- ein SPI serielles Interface
- ein 10-bit ADC
- ein programmierbarer Timer mit internem Oszillator

- ein SPI serieller Port
- 6 verschiedene Energiesparmodi

Die hier vorkommenden Begriffe und die Bedeutung der genannten Funktionalitäten werden im **Glossar (10.1)** bzw. im nächsten Kapitel **5.1.4** erklärt.

5.1.4 Atmel® ATmega1284P

Der ATmega1284P ist ebenfalls ein 8-bit-Mikrocontroller des Herstellers Atmel® und gehört der AVR®-Familie an. Er ersetzt den Atmel® ATmega32 als zentrale Steuereinheit, da er über weit mehr Speicherkapazitäten verfügt.

5.1.4.1 Komponenten

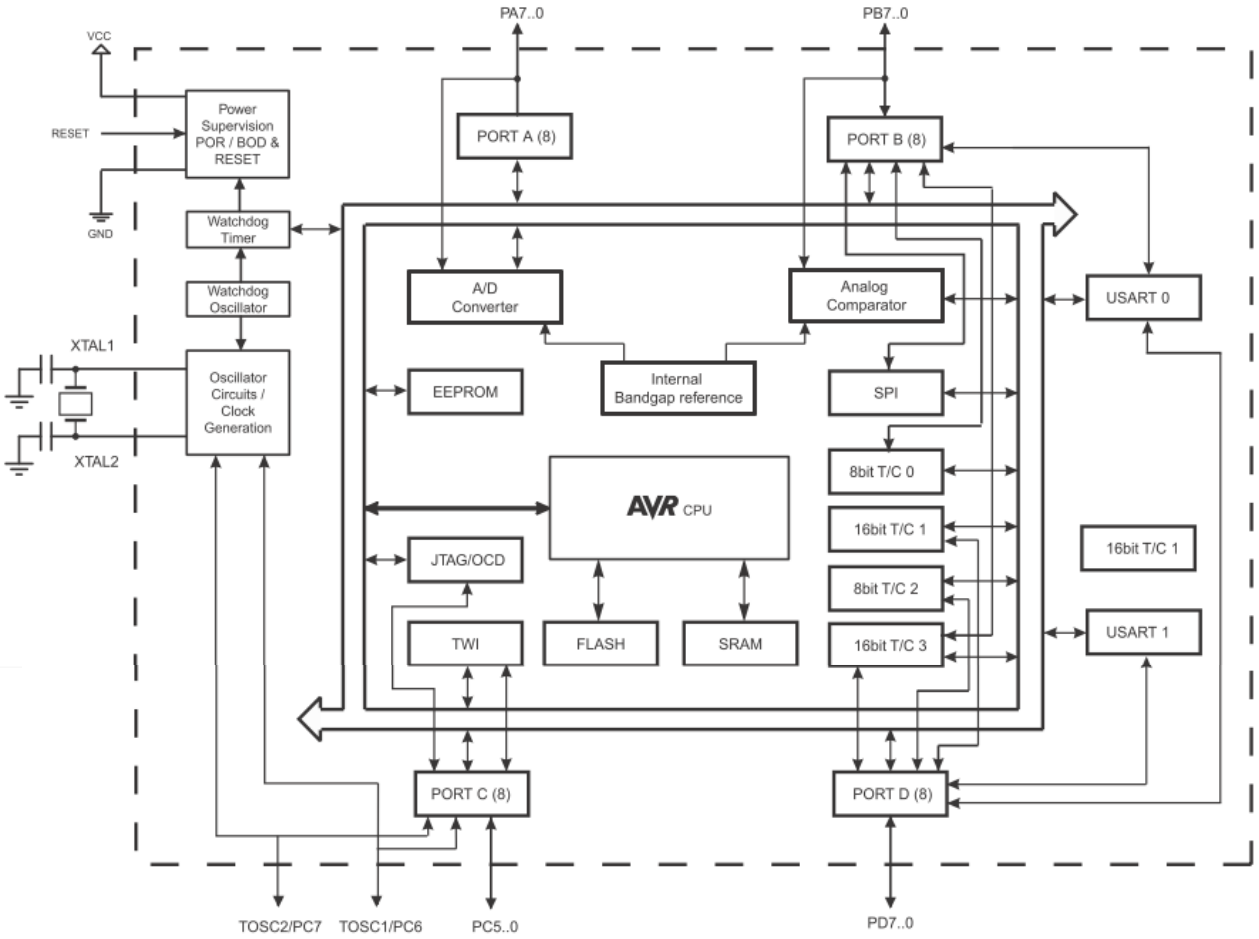


Abb. 18 Blockdiagramm der Komponenten des Atmega1284P

AVR® CPU

Die AVR® CPU dient als zentrale Steuereinheit. Ihre Hauptaufgabe ist die korrekte Ausführung des Mikrocontroller-Programms. Um dem nachzugehen muss die CPU auf Speicherregister zugreifen, Rechenoperationen ausführen, die Peripherie steuern und mit Interrupts umgehen.

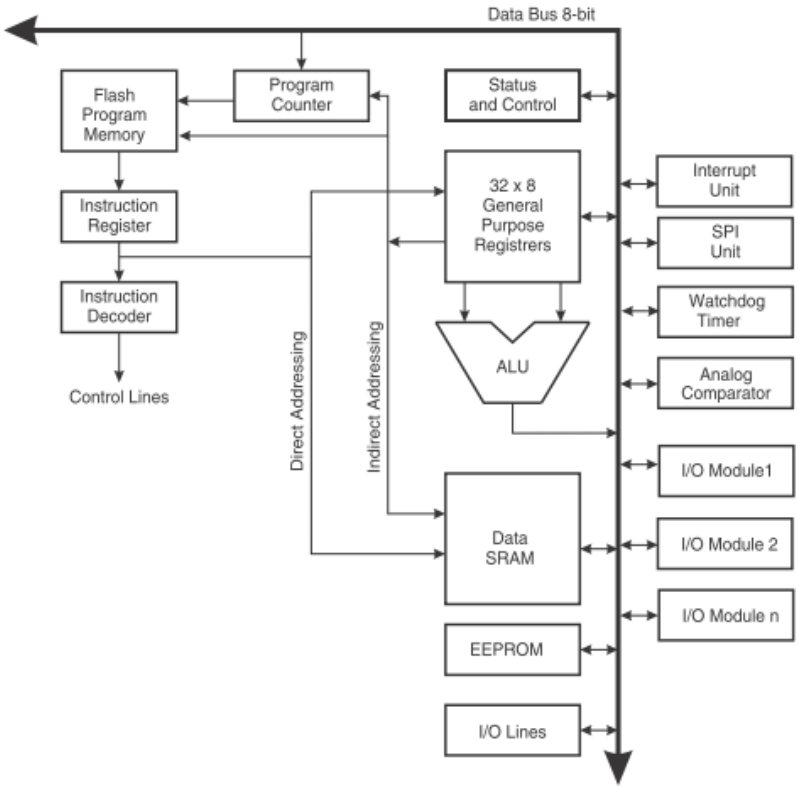


Abb. 19 Blockdiagramm der AVR CPU-Architektur

Die AVR® CPU ist nach der Harvard-Architektur aufgebaut. Somit sind Programmspeicher und Datenspeicher logisch und physisch voneinander getrennt und es kann auf beide Speicher gleichzeitig zugegriffen werden.

Die ALU (Arithemtic Logic Unit) bildet das Herzstück des Prozessors. Sie führt die eigentlichen Rechenoperationen aus und braucht daher direkten Zugriff auf die 32 Arbeitsregister.

Die Operationen im Programmspeicher werden via Single-Level-Pipelining betrieben. Dadurch kann während der Ausführung einer Operation bereits die nächste aus dem Programmspeicher geladen werden.

Selbige CPU befindet sich übrigens auch im ATmega32.

Flash-Programmspeicher

In diesem Speicher wird der fertig kompilierte Programmcode abgelegt. Auf ihn kann sowohl schreibend als auch lesend von außen zugegriffen werden.

Der ATmega1284P bietet einen Programmspeicher von 128 KB, welcher über ein SPI-serielles Interface wiederprogrammiert werden kann. Selbstverständlich ist dieser Speicher nicht flüchtig. Somit bleibt das darin gespeicherte Programm auch nach Abschaltung der Versorgungsspannung enthalten.

Der Speicher ist in zwei Bereiche unterteilt. Einerseits gibt es eine Sektion für das eigentliche Programm (Application Flash Section) und andererseits einen Bootloader-Bereich, welcher dafür zuständig ist, neue Programme richtig in den Speicher zu laden.

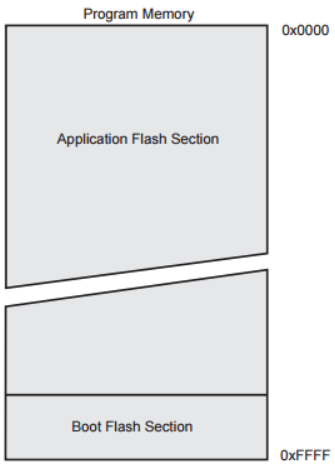


Abb. 20 Programmspeicherunterteilung

SRAM Datenspeicher

Der SRAM ist ein flüchtiger, statischer Arbeitsspeicher. Lese- und Schreibzugriffe können beliebig oft erfolgen, weshalb hier neben dem Programm-Stack auch die in einem Programm verwendeten Variablen abgelegt werden. Der Datenspeicher unterteilt sich folgendermaßen:

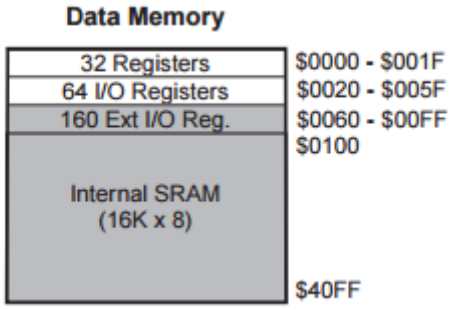


Abb. 21 SRAM Datenspeicher

Die ersten 32 Adressen sind für die Speicherregister reserviert und die nächsten 64 für die I/O-Register. Darauf folgen 160 Adressen für erweiternde I/O-Register und letztendlich die 16 KB interner SRAM, der für die Speicherung von Variablen genutzt werden kann.

EEPROM Datenspeicher

Dieser Datenspeicher ist nicht flüchtig und bleibt somit auch nach der Trennung von der Versorgungsspannung erhalten. Der EEPROM im ATmega1284P beläuft sich auf eine Größe von 4KB und übersteht in etwa um die 100.000 Schreib- bzw. Lesezyklen. Die darin gespeicherten Informationen können elektrisch gelöscht werden.

General Purpose Working Register

Der ATmega1284P weist 32 solcher Register zu jeweils 8 Bit auf. Die ALU hat direkten Zugriff auf diese Register, welche für die Bearbeitung von Operanden bei Operationen, die von der ALU durchgeführt werden, nützlich sind.

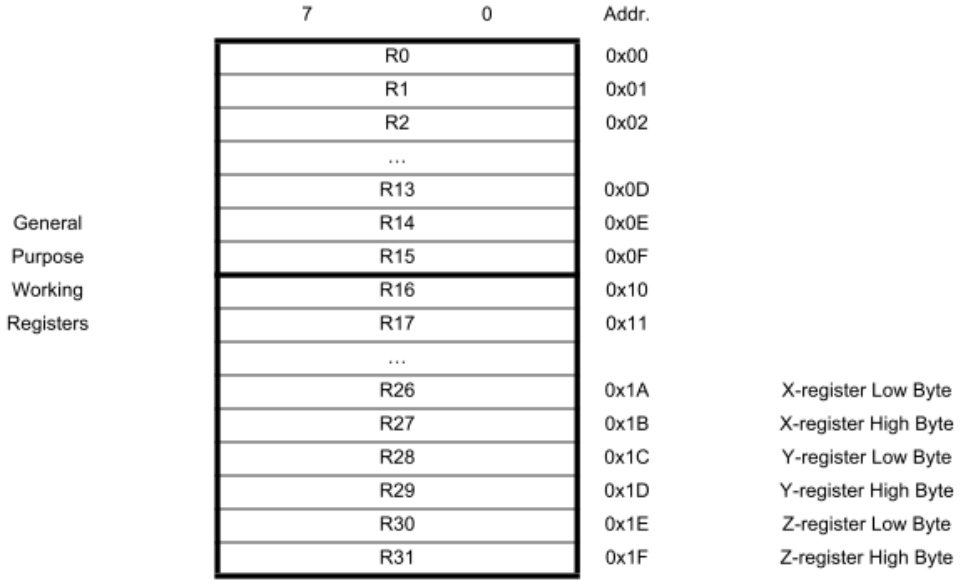


Abb. 22 AVR CPU General Purpose Working Registers

SPI-Interface

Über diese Schnittstelle können Programme in den Programmspeicher des Mikrocontrollers geladen werden.

USART

USART ist eine Hardware-Komponente, die Daten zwischen parallelen und seriellen Formaten übersetzen kann. Sie ist für die serielle Kommunikation verschiedener Peripherie-Komponenten zuständig.

A/D-Konverter

Der Analog-zu-Digital-Konverter kann analoge Signale (analoge Spannungswerte) in digitale konvertieren, damit diese vom Mikrocontroller weiterverarbeitet werden können.

Oszillator

Ein Mikrocontroller benötigt eine Taktversorgung, um die internen Abläufe im Prozessor in einer zeitlich geordneten Reihenfolge ausführen zu können. Diese Taktversorgung wird von einem sogenannten Oszillator übernommen. Die Taktfrequenz ist ausschlaggebend für die Rechengeschwindigkeit des Mikrocontroller-Prozessors. Der ATmega1284P besitzt eine Taktfrequenz von 20 MHz.

5.1.5 GLCD Display

Das GLCD-Display wurde von Wolfgang Schinnerl zur Verfügung gestellt. Es hat eine Auflösung von 128x64 Pixeln. Große Nachteile sind allerdings die geringe Auflösung und die Möglichkeit, nur zwei Farben abzubilden, weshalb die Benutzeroberfläche für das TFT-Display programmiert wurde. Für die benötigten grafischen Elemente, besonders für Schriftzüge, wie etwa die Beschriftung der Buttons oder Fehlermeldungen, wäre hier einfach nicht genug Platz gewesen.

Abb. 23 GLCD

5.1.6 EasyTFT Board

Dieses Board des Herstellers MikroElektronika beherbergt ein TFT-Display mit einer Auflösung von 320x240 Pixeln, das von einem ILI9341 Display-Controller betrieben wird. Außerdem enthält es GLCD-kompatible Konnektoren, weshalb es auch in den GLCD-Socket unseres Entwicklungsboards (siehe Kapitel **5.1.2 EasyAVR™ v7**) passt. Das Display kann 262 verschiedene Farben darstellen. Somit ist es, was die Auflösung und Farbdarstellung betrifft, dem GLCD-Display weit überlegen. Es wurde ebenfalls vom Auftraggeber bezahlt.

Abb. 24 EasyTFT

Das EasyTFT hätte zwar einen Touch Screen integriert, dessen Benutzung würde aber im Zusammenhang mit dem EasyAVR™ v7 zusätzliche Löteingänge erfordern, weshalb ein Touchpanel benötigt wird.

5.1.7 Touch Panel

Das Touch Panel wurde von Wolfgang Schinnerl zur Verfügung gestellt und hat eine Auflösung von 128x64, also ideal für das GLCD-Display. Wird das Touch Panel gemeinsam mit dem EasyTFT benutzt, muss man daher die Koordinaten einer Berührung umrechnen, da die jeweiligen Auflösungen verschieden sind.

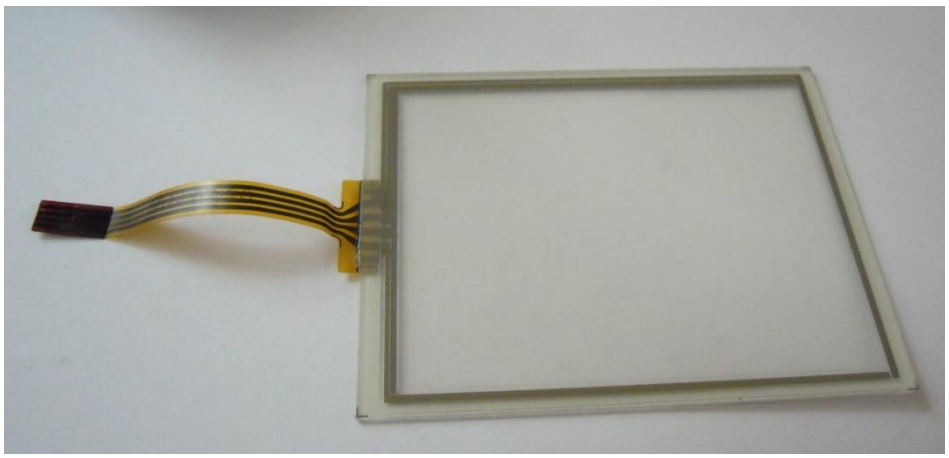


Abb. 25 Touch Panel

5.1.8 RS485 click 5V

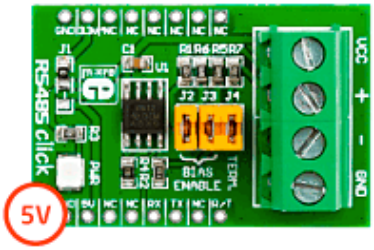


Abb. 26 RS485 Click-Board

Dieses Click-Board wird durch die mikroBUS™-Technologie (siehe **5.1.2 EasyAVR™ v7**) angesteuert. Es beherbergt eine RS485-Schnittstelle, die als physikalische Verbindung für die Profibus-Kommunikation zum Roboter hin dienen soll.

5.1.9 OPTO click

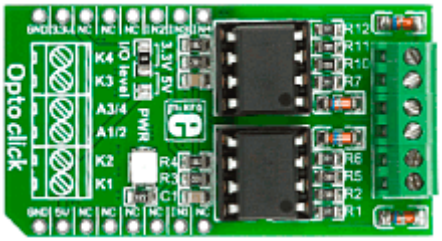


Abb. 27 Opto Click-Board

An diesem Click-Board soll später die Sensorik angeschlossen werden, welche für die Überwachung der Lego®-Zufuhr zuständig ist. Es beinhaltet Optokoppler, die die Sensoren galvanisch vom Hauptboard (EasyAVR™) trennen können.

5.2 Software

Im Folgenden wird die verwendete Software genauer erläutert und begründet, warum wir uns für diese entschieden haben.

5.2.1 mikroC PRO for AVR®

5.2.1.1 Allgemein

Diese Entwicklungsumgebung für die Programmiersprache C wurde für die Programmierung des Mikrocontrollers verwendet. Sie ist perfekt auf die Hardwarekomponenten von MikroElektronika und Mikrocontroller von AVR® abgestimmt. "MikroC PRO for AVR®" benutzt AVRFlash (siehe unten), um Schreib- und Lesezugriffe auf den Mikrocontroller abzusetzen. Der geschriebene Code wird mit einem ANSI C-Compiler kompiliert, der speziell auf Mikrocontroller-Programmierung angepasst ist. Außerdem bietet mikroC eine Vielzahl an Software- bzw. Hardwarebibliotheken und praktische Tools.

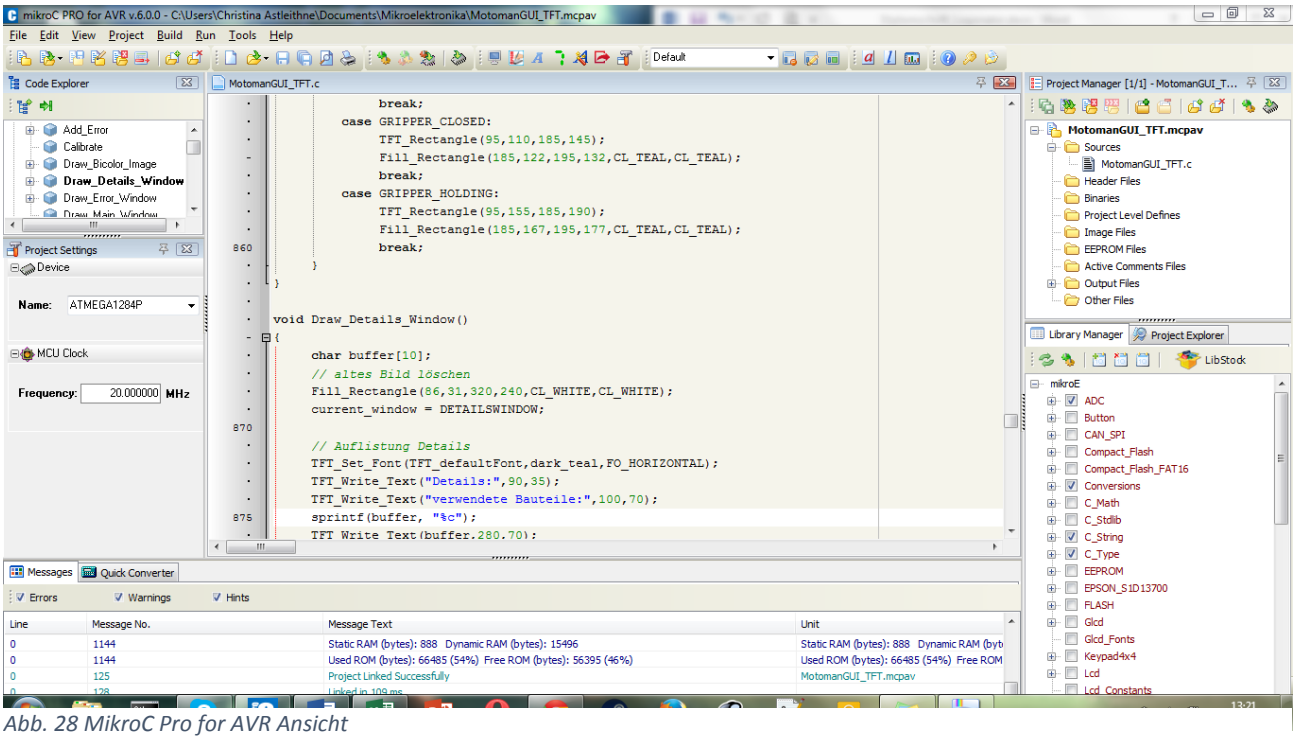


Abb. 28 MikroC Pro for AVR Ansicht

Im Folgenden werden einige nützliche Zusatzfunktionalitäten von mikroC Pro for AVR® genauer erläutert.

5.2.1.2 Library-Manager

Mithilfe des Library-Managers lassen sich die mitgelieferten Bibliotheken von MikroElektronika jederzeit mit einem Klick hinzufügen. Dazu hakt man einfach jene Bibliotheken an, die man ins Projekt einbinden will.

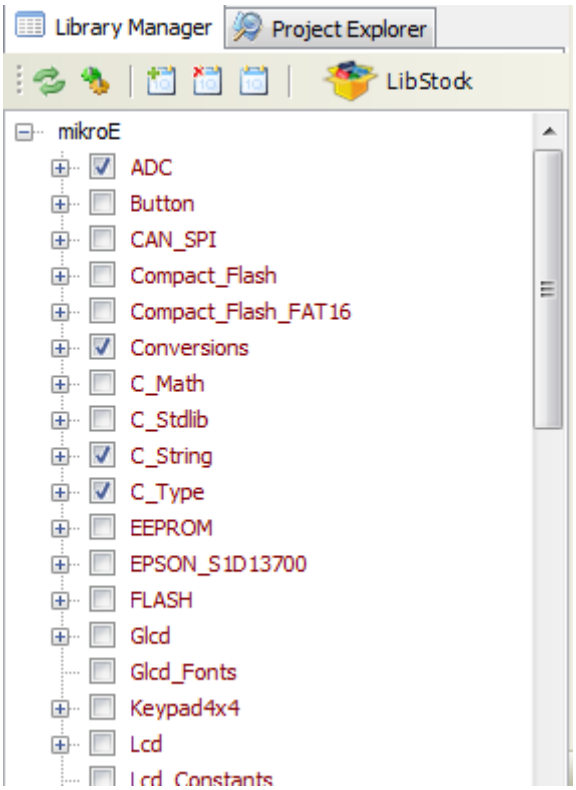


Abb. 29 Library-Manager Bibliothekenübersicht

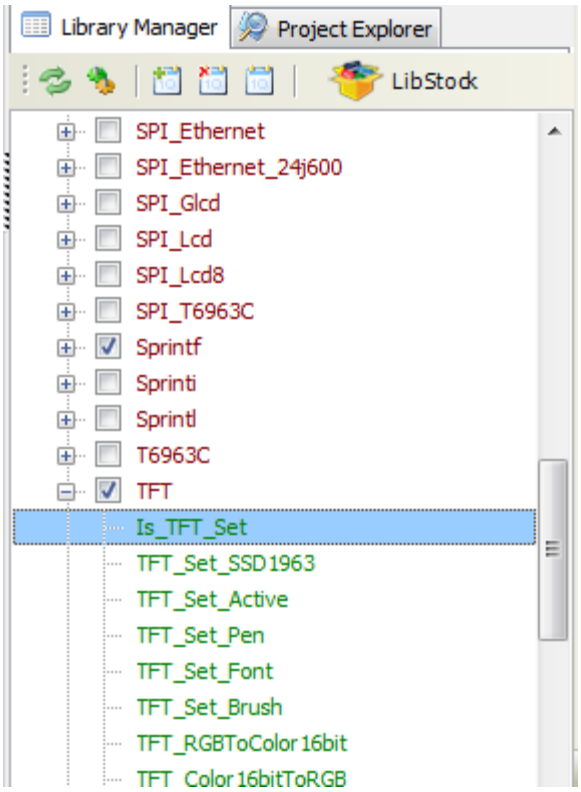
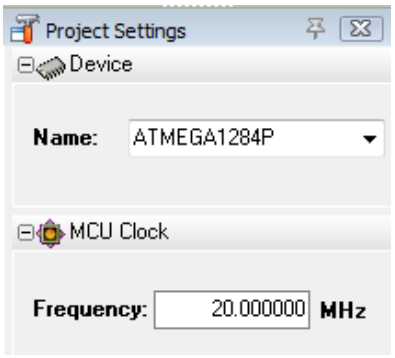


Abb. 30 Library-Manager Methodenübersicht

mikroC PRO enthält außerdem eine übersichtliche Dokumentation zu den mitgelieferten Bibliotheken. Leider wird für deren Funktionen kein Source-Code bereitgestellt. Alle für das Projekt verwendeten Bibliotheken werden im Kapitel **6.2.1 Verwendete mikroC-Bibliotheken** aufgelistet und beschrieben. mikroC bietet sogar eine auf Mikrocontroller abgestimmte C-Standardbibliothek an.

5.2.1.3 Spezialisierung auf Mikrocontroller



mikroC PRO for AVR® unterscheidet zwischen verschiedenen Arten von Mikrocontrollern. So werden beispielsweise beim Anlegen eines neuen Projektes die Art des Mikrocontrollers und dessen Taktfrequenz angegeben und in den Projekteigenschaften definiert.

Abb. 31 Project Settings

Aufgrund der Projekteigenschaften können dann die Header-Dateien der Bibliotheken von MikroElektronika an den entsprechenden Mikrocontroller angepasst werden. Außerdem werden beim Kompilieren Berechnungen zum Speicher des Mikrocontrollers durchgeführt. So erhält man einen guten Überblick, wie viel Speicher derzeit vom Programm belegt wird und wie viel noch zur Verfügung steht. Reicht der Speicher des Mikrocontrollers nicht aus, wird dies ebenfalls von der Entwicklungsumgebung durch einen Kompilierfehler mitgeteilt.

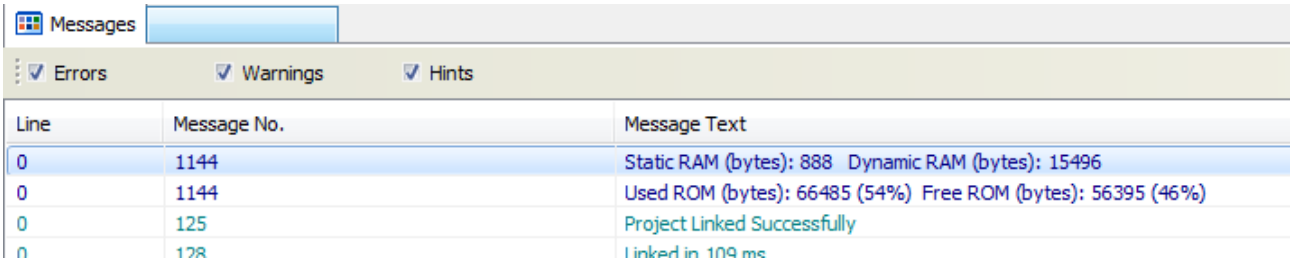


Abb. 32 Speicherberrechnung mikroC

5.2.2 AVRFlash

Der AVRFlash-Programmer nutzt den mikroProg™ on-board-Programmer (siehe Kapitel **5.1.2 EasyAVR™ v7**), um Schreibe- und Lesezugriffe auf den Speicher des Mikrocontrollers durchzuführen. Damit wird das Programm auf den Mikrocontroller geladen.

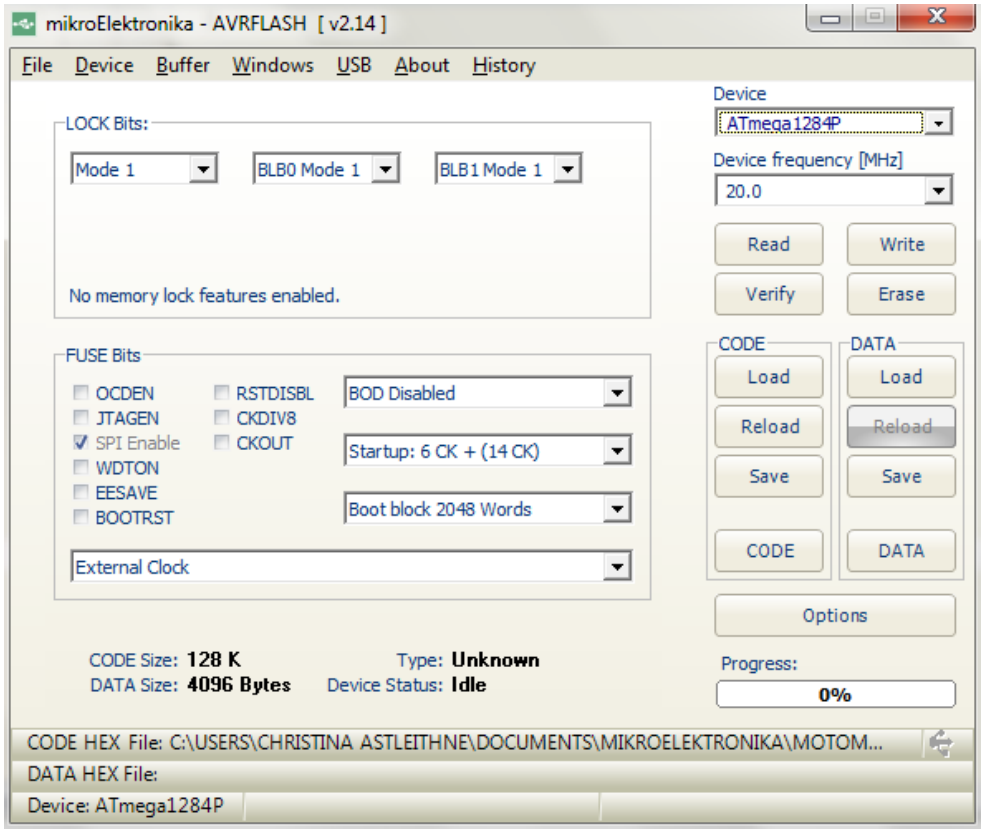


Abb. 33 AVRFlash-Programmer

5.3 Sonstige Software-Tools

5.3.1 Microsoft® Office 2013

Microsoft® Office 2013 ist ein umfassendes Paket aus verschiedenen Programmen, welche man im Büroalltag brauchen kann.
Wir verwenden unterschiedliche Programme aus dem Office-Paket.

Microsoft® Word

- laufende Recherchen
- Erstellung diverser Dokumente
- Erstellung dieser schriftlichen Ausarbeitung

Abb. 34 Microsoft Word Logo

Microsoft® PowerPoint

- Erstellung des Layouts für die verschiedenen Ansichten der Bildschirme, welche für das TFT-Display entwickelt worden sind
- Erstellen von Projektpräsentationen

Abb. 35 Microsoft Powerpoint Logo

Microsoft® Excel

- Kostenplanung aufstellen, Diagramme zeichnen (siehe **Kapitel 5.4 Kostenplanung**)
- Abbildungsverzeichnis führen

Abb. 36 Microsoft Excel Logo

5.3.2 GanttProject

GanttProject ist eine freie Anwendung zur Erstellung von Gantt-Diagrammen und zur Ressourcenverwaltung für Projekte. Verwendet wurde es für die Erstellung eines Gantt-Plans, der alle Meilensteine beinhaltet.

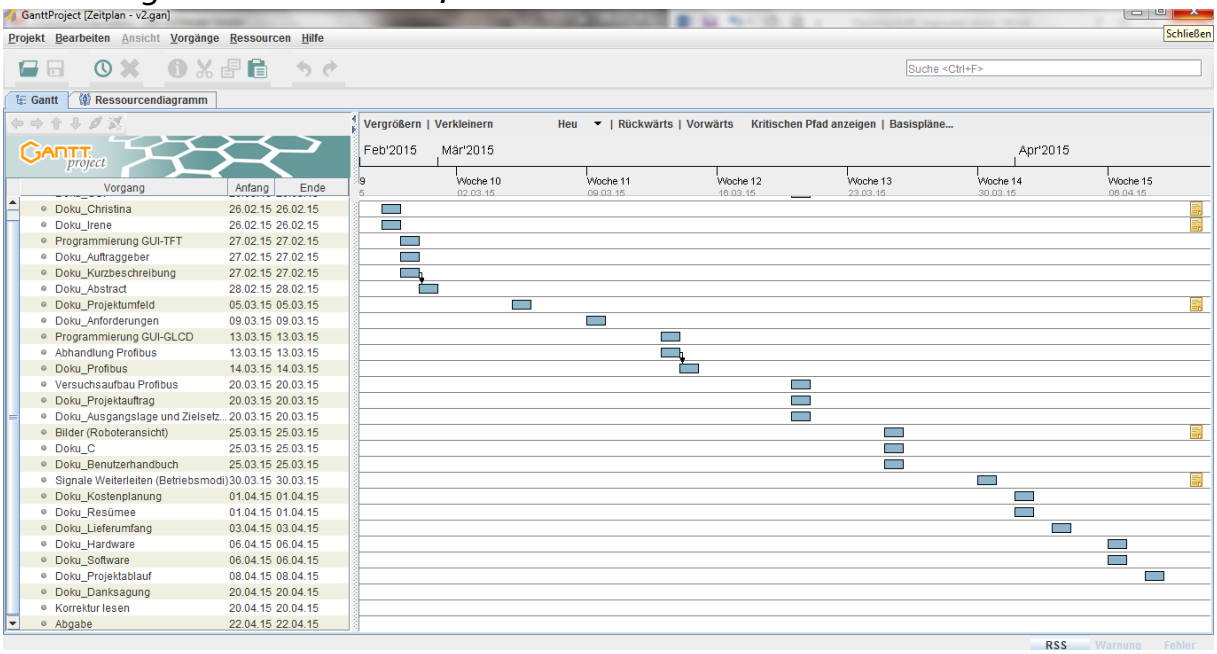


Abb. 37 Ganttplan ab Februar

5.3.3 Google Drive™

Google Drive ist ein Web-Dienst von Google zur Speicherung und Synchronisierung von Dateien. Benutzer können Dateien in der Cloud speichern und für andere freigeben. Verwendet wurde der Dienst für eine gemeinsame Dateiablage, auf die alle am Projekt beteiligten Personen Zugriff haben.

Abb. 38 Google Drive Logo

5.3.4 GIMP

Abb. 39 GIMP Logo

GIMP (GNU Image Manipulation Program) ist ein kostenloses und freies Bildbearbeitungsprogramm unter der GNU-Lizenz. Es enthält viele nützliche Tools und Funktionen zur Bildbearbeitung und wurde für das Modifizieren von Grafiken für diverse Projektpräsentationen, Dokumente und diese Diplomschrift verwendet. Ein häufiger Verwendungszweck war etwa das Transparent-machen von Hintergründen.

5.4Kostenplanung

5.4.1 Budget

Die Diplomarbeit wird im Rahmen der Matura der HTL Perg angefertigt, daher entstehen auch keine Kosten für die beiden Diplomanden.

Diese Kostenaufstellung dient lediglich der Darstellung, wie viel ein Projekt in diesem Umfang in der Wirtschaft kosten würde. Selbstverständlich wurde für die Arbeitsleistung oder andere Kosten nichts in Rechnung gestellt.

Einige Kosten sind allerdings tatsächlich aufgetreten, wie zum Beispiel für die Entwicklungsumgebung, oder auch für unser Entwicklungsboard. Diese wurden aber von dem Unternehmen RK Automatisierungstechnik GmbH übernommen. Sie werden in der folgenden Aufstellung berücksichtigt.

5.4.2 Arbeitsleistung

In der folgenden Tabelle wird berechnet, wie viel unsere Arbeitsleistung in der Wirtschaft wert wäre. Der Stundensatz ist nur ein Richtwert und kann in der Realität variieren.

Kostenträger	Stunden	Stundensatz	Betrag
Christina Astleithner	200	€ 35,00	€ 7.000,00
Irene Stöger	200	€ 35,00	€ 7.000,00
Gesamt			**€ 14.000,00**

Tabelle 2 Kosten Arbeitsleistung

5.4.3 Hardwarekosten

Die Hardwarekosten wurden sowohl von der Firma RK Automatisierungstechnik GmbH als auch von der Schule übernommen.

S = Schule, RK = Auftraggeber

Kostenträger	Anzahl	Stückkosten ($)	Betrag	Bezahlt von
EasyARV™ v7	1	$ 149,00	$ 149,00	RK
EasyTFT™ Board	1	$ 35,00	$ 35,00	RK
RS4485 click 5V	1	$ 18,00	$ 18,00	RK
Opto click	1	$ 21,00	$ 21,00	RK
Wire Jumper	1	$ 3,00	$ 3,00	RK
ATmega1284P-Pu	1	$ 5,30	$ 5,30	RK
Raspberry Pi	1	$ 40,00	$ 40,00	S
ATmega88	1	$ 3,00	$ 3,00	S
MAX-3232 RS-232 Transceiver	1	$2,00	$ 2,00	S
Gesamt			**€ 245,91**	

Tabelle 3 Kosten Hardware

Wechselkurs: 0,89, lt. http://www.waehrungsrechner-euro.com/

5.4.4 Lizenzkosten

Für die Entwicklung sowie Dokumentation wurden verschiedene Software Produkte benötigt. In der nachstehenden Tabelle wird veranschaulicht, wie viel eine Neuanschaffung unserer Programme kosten würde.

Kostenträger	Anzahl	Stückkosten	Betrag
Microsoft® Windows 7 Home Premium	1	€ 17,90	€ 17,90
Microsoft® Windows 7 Ultimate	1	€ 79,90	€ 79,90
Microsoft® Office 2013 Professional	2	€ 539,00	€ 1078,00
mikroC PRO for AVR®	2	€ 249,00	$ 498,00
Gesamt			**€ 1619,02**

Tabelle 4 Kosten Software

5.4.5 Sonstige Kosten

Die sonstigen Kosten sind Annahmen, welche Kosten während dem Ablauf der Diplomarbeit zusätzlich entstanden wären.

Kostenträger	Betrag
Internetgebühren	€ 17,00
Telefongebühren	€ 10,00
Büromaterial	€ 5,00
Gesamt	**€ 32,00**

Tabelle 5 sonstige Kosten

5.4.6 Gesamtkosten

In der nachstehenden Tabelle sind die Gesamtbeträge noch einmal zusammengefasst. Dabei wird deutlich, dass in der realen Arbeitswelt der Großteil der Kosten durch die Arbeitsleistung entstanden wäre.

Kostenträger	Betrag
Arbeitsleistung	€ 14.000,00
Lizenzen	€ 1.619,02
Hardware	€ 245,90
Sonstige	€ 32,00
Gesamt	**€ 15.896,92**

Tabelle 6 Gesamtkosten

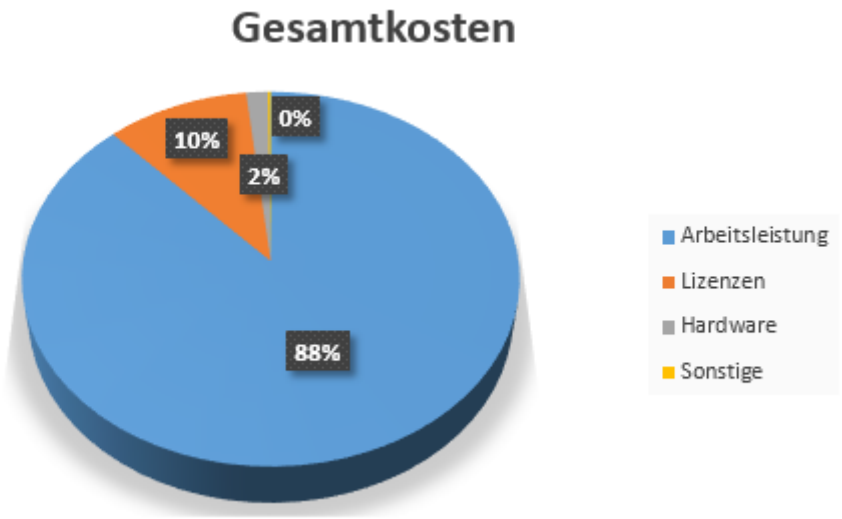


Abb. 40 Gesamtkosten Diagramm

5.5 Lieferumfang

In diesem Kapitel wird beschrieben, was der Auftraggeber vom Diplomandenteam geliefert bekommt.

5.5.1 Dokumente

Im Lieferumfang sind zwei Dokumente enthalten. Eines davon ist diese Diplomschrift. Das andere ist eine Abhandlung über das Profibus-Protokoll, welches allgemeine Fakten über das Protokoll und außerdem konkrete Lösungsansätze für gegebene Ausgangslage liefert.

5.5.2 Software

Der Auftraggeber erhält vom Diplomandenteam ein C-Programm für einen Mikrocontroller, das eine Benutzeroberfläche für den Industrieroboter auf einem TFT-Display realisiert und außerdem die Interaktion mit der Oberfläche über ein Touch-Panel ermöglicht.

6 Programmierung

Eine der zwei Hauptbeschäftigungen bei unserer Diplomarbeit war die Programmierung eines Mikrocontrollers. Dieser ist dann dazu in der Lage, eine graphische Benutzeroberfläche auf einem TFT-Display darzustellen, und Benutzereingaben über ein Touch-Panel zu verwalten. Diese Benutzeroberfläche soll später die Bedienung des Industrieroboters vereinfachen. In diesem Kapitel wird das dafür geschriebene Programm genau erklärt.

6.1 Programmspezifikationen

Das Programm wurde in mikroC Pro for AVR® (siehe **5.2.1**) für den Mikrocontroller ATmega1284P geschrieben, wobei dieser dank der Entwicklungsumgebung jederzeit ausgetauscht werden kann, ohne viel am Programm ändern zu müssen. Er läuft auf einer Frequenz von 20 MHz.

6.2 Bibliotheken

Für die Programmierung wurden die speziell für Mikrocontroller optimierten Bibliotheken von MikroElektronika verwendet, die bereits im Lieferumfang von mikroC Pro for AVR® enthalten sind. Diese sind zwar halbwegs übersichtlich, wenn auch wenig genau dokumentiert, doch leider steht kein Source-Code zur Verfügung, mit dessen Hilfe die Funktionsfähigkeit der Library-Funktionen kontrolliert werden könnte. Wenn die erwarteten Werte herauskommen, kann die Funktion verwendet werden und wenn nicht, muss nach einem alternativen Lösungsansatz gesucht werden. Dies hat dazu geführt, dass während der Programmentwicklung zwangsweise nach einem Try-and-Error-Ansatz vorgegangen wurde, da bei der erstmaligen Verwendung einer Funktion nicht immer bekannt war, ob sie überhaupt mit den gegebenen Input-Parametern funktioniert bzw. ob sie überhaupt die richtigen Ausgabewerte liefert.

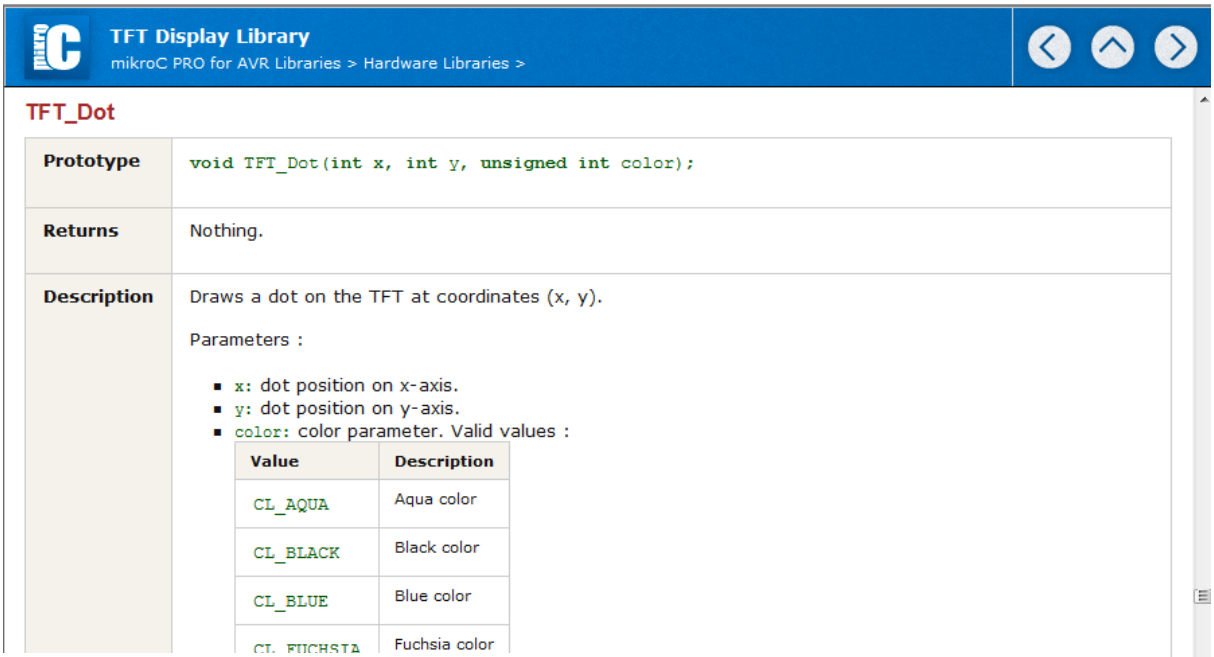


Abb. 41 Beispiel Dokumentation der mikroC Pro vor AVR Libraries

Ein Vorteil der Bibliotheken von MikroElektronika ist allerdings, dass die Funktionsaufrufe für verschiedene Mikrocontroller gleich sind. So kann jederzeit der Mikrocontroller ausgetauscht werden, ohne viel am Programmcode ändern zu müssen. Dafür wird in den Projektspezifikationen eines mikroC-Projektes festgelegt, auf welchen Mikrocontroller es laufen soll. Dementsprechend werden dann die passenden Bibliotheken hinzugefügt.

Um die Bibliotheken ins Projekt einzubinden, wird der Library-Manager von mikroC (siehe **5.2.1.2**) verwendet.

6.2.1 Verwendete mikroC-Bibliotheken

Es folgt eine Liste aller für das Programm verwendeten Bibliotheken, deren Funktionen und deren Verwendungszweck.

ADC Library

Diese Bibliothek bietet Routinen für die Verwendung eines Analog-zu-Digital-Konverter-Moduls. Ein solches Modul ist in vielen Mikrocontrollern, wie im ATmega1284P bereits verbaut.

verwendete Funktionen:

ADC_Init()	
Parameter	Nichts
Return-Wert	Nichts
Beschreibung	Initialisiert ein ADC-Modul
Verwendung	Initialisiert das ADC-Modul, um die analogen Signale, die vom Touch-Panel bei einer Berührung abgesetzt werden, in digitale umzuwandeln

Tabelle 7 Funktionen der ADC-Bibliothek

ANSIC C String Library (C_String)

Diese Bibliothek stellt eine Reihe von Standard ANSI C-Bibliotheksfunktionen zur Verfügung, die für die Manipulation von Zeichenketten und Arbeitsspeicher-Operationen zuständig sind.

verwendete Funktionen:

strcpy()	
Parameter	**char *to** Zeichenkette, in die kopiert werden soll **char *from** Zeichenkette, die kopiert werden soll
Return-Wert	**char*** kopierte Zeichenkette
Beschreibung	Kopiert eine übergebene Zeichenkette in eine andere übergebene Zeichenkette und überschreibt deren Inhalt dabei
Verwendung	Kopiert einen neuen Fehlermeldungstext in eine dafür vorhergesehene Strukturvariable

Tabelle 8 Funktionen der C_String-Bibliothek

Sprint Library (Sprintf)

Diese Bibliothek enthält Äquivalente zu den ANSI C-Standard-Funktionen sprintf(), sprintl() und sprinti().

verwendete Funktionen:

sprintf()	
Parameter	**char *wh** Zeichenkette, in die kopiert werden soll **const code char *f** Format-String **...** Variablen oder Literale, die im Format-String vorkommen
Return-Wert	Nichts
Beschreibung	Schreibt den Format-String in eine übergebene Zeichenkette und wandelt dabei alle Literale, die im Format-String vorkommen, nach Angabe um
Verwendung	Wird zur Ausgabe von Integer-Werten als Text auf dem Easy-TFT-Board verwendet

Tabelle 9 Funktionen der Sprint-Library

TFT Library (TFT, TFT_Defs)

Diese Bibliothek bietet eine Reihe von Funktionen für die Initialisierung und die grafische Ausgabe am TFT-Display. Die Verwendung dieser Bibliothek setzt voraus, dass die Display-Connections für das TFT-Display zum Mikrocontroller bereits auf entsprechenden Ports des Mikrocontrollers definiert sind.

verwendete Funktionen:

TFT_Init_ILI9341_8bit()	
Parameter	**unsigned int display_width** Länge des TFT-Displays **unsigned char display_height** Höhe des TFT-Displays
Return-Wert	Nichts
Beschreibung	Initialisiert einen ILI9341 Display-Controller im 8-bit-Betriebsmodus
Verwendung	Initialisiert einen ILI9341 Display-Controller im 8-bit-Betriebsmodus für die Ansteuerung des TFT-Displays

TFT_Fill_Screen()	
Parameter	**unsigned int color** Farbe mit der Bildschirm gefüllt werden soll
Return-Wert	Nichts
Beschreibung	Füllt den Bildschirm des TFT-Displays mit der übergebenen Farbe
Verwendung	Löscht die bisherige Bildschirmdarstellung mit einer entsprechenden Farbe um Platz für neue grafische Elemente zu machen
TFT_Dot()	
Parameter	**int x** X-Koordinate des zu zeichnenden Punktes **int y** Y-Koordinate des zu zeichnenden Punktes **unsigned int color** Farbe des zu zeichnenden Punktes
Return-Wert	Nichts
Beschreibung	Malt einen Bildpunkt, der durch die übergebenen Koordinaten beschrieben wird, mit einer übergebenen Farbe aus
Verwendung	Zeichnet Eckpunkte für die Kalibrierungs-Funktion des Touch-Panels, wird beim Zeichnen von Bildern benötigt, um diese Bildpunkt für Bildpunkt zu zeichnen
TFT_Set_Font()	
Parameter	**const char far *activeFont** Font, momentan wird hier nur der Wert TFT_defaultFont (also Tahoma 14x16) unterstützt **unsigned int font-color** Textfarbe **char font_orientation** Textrichtung (vertikal oder horizontal)
Return-Wert	Nichts
Beschreibung	Ändert das aktuelle Textformat, in dem alle auf den Display geschriebenen Texte dargestellt werden
Verwendung	Ändern der Schriftfarbe

TFT_Write_Text()	
Parameter	**unsigned char *text** zu schreibende Zeichenkette **unsigned int x** X-Koordinate **unsigned int y** Y-Koordinate
Return-Wert	Nichts
Beschreibung	Schreibt eine übergebene Zeichenkette von einem linken oberen Eckpunkt aus, der durch die übergebenen Koordinaten beschrieben wird, auf das Display
Verwendung	Ausgabe von jeglichem Text für die Benutzeroberfläche auf das TFT-Display
TFT_Set_Pen()	
Parameter	**unsigned int pen_color** Farbe von zu zeichnenden Linien **char pen_width** Breite von zu zeichnenden Linien
Return-Wert	Nichts
Beschreibung	Legt die Farbe und Breite für das Zeichnen von Linien, Kreisen und Rechtecken fest
Verwendung	Ändern der Farbe und Breite von Linien, Kreisen und Rechtecken
TFT_Set_Brush()	
Parameter	**char brush_enabled** aktiviert/deaktiviert die Ausfüllung von Rechtecken oder Kreisen mit einer Farbe bzw. einem Farbverlauf **unsigned int brush_color** Füllfarbe bei einfarbiger Füllung **char gradient_enabled** gibt an, ob ein Farbverlauf verwendet werden soll **char gradient_orientation** Richtung des Farbverlaufs **unsigned int gradient_color_from** Startfarbe des Farbverlaufs **unsigned int gradient_color_to** Endfarbe des Farbverlaufs
Return-Wert	Nichts

Beschreibung	Legt die Farbe bzw. den Farbverlauf fest, mit dem Kreise und Rechtecke gefüllt werden
Verwendung	Definiert Füllfarben bzw. verschiedene Farbverläufe, mit denen zu zeichnende Rechtecke oder Kreise gefüllt werden sollen
TFT_Rectangle()	
Parameter	**int x_upper_left** X-Koordinate der linken oberen Ecke **int y_upper_left** Y-Koordinate der linken oberen Ecke **int x_bottom_right** X-Koordinate der rechten unteren Ecke **int y_bottom_right** Y-Koordinate der rechten unteren Ecke
Return-Wert	Nichts
Beschreibung	Zeichnet ein Rechteck von einem linken oberen Eckpunkt aus, der durch übergebene Koordinaten definiert wird, bis zu einem rechten, unteren Eckpunkt, der durch übergebene Koordinaten definiert wird
Verwendung	Zeichnen von rechteckigen Formen, wie etwa Buttons, Leisten, etc.
TFT_RGBToColor16bit()	
Parameter	**char rgb_red** Rotanteil der zu erzeugenden Farbe **char rgb_green** Grünanteil der zu erzeugenden Farbe **char rgb_blue** Blauanteil der zu erzeugenden Farbe
Return-Wert	**unsigned int** Integer-Wert, der die erzeugte Farbe repräsentiert
Beschreibung	Konvertiert ein 5:6:5-RGB-Format in ein echtes Farbformat, das durch einen Integer-Wert repräsentiert wird und gibt diesen Wert zurück
Verwendung	Definieren von eigenen Farben neben den vordefinierten Farbkonstanten der TFT-Library

Tabelle 10 Funktionen der TFT-Bibliothek

Touch Panel Library (TouchPanel)

Diese Bibliothek bietet Funktionen an, die das Arbeiten mit dem Touch Panel erleichtern. Sie setzt allerdings voraus, dass die Touch Panel-Connections zum Mikrocontroller bereits auf entsprechenden Ports des Mikrocontrollers definiert sind.

verwendete Funktionen:

TP_Init()	
Parameter	**unsigned int display_width** Länge des Touch-Panels **unsigned int display_height** Höhe des Touch-Panels **char readX_ChNo** ADC-Channel, von dem die X-Koordinate einer Berührung gelesen wird **char readY_ChNo** ADC-Channel, von dem die Y-Koordinate einer Berührung gelesen wird
Return-Wert	Nichts
Beschreibung	Initialisiert ein Touch-Panel
Verwendung	Initialisiert das Touch-Panel
TFT_Set_ADC_Threshold()	
Parameter	**unsigned int color** Farbe mit der der Bildschirm gefüllt werden soll
Return-Wert	Nichts
Beschreibung	Legt den Ansprechwert für ein Touch-Panel fest, also den Wert, ab den eine Berührung erkannt wird
Verwendung	Legt den Ansprechwert für das Touch-Panel fest
TP_Calibrate_Bottom_Left()	
Parameter	Nichts
Return-Wert	Nichts
Beschreibung	Kalibriert die linke untere Ecke des Touch Panels
Verwendung	Kalibriert die linke untere Ecke des Touch Panels
TP_Calibrate_Upper_Right()	
Parameter	Nichts
Return-Wert	Nichts
Beschreibung	Kalibriert die rechte obere Ecke des Touch Panels
Verwendung	Kalibriert die rechte obere Ecke des Touch Panels

TP_Press_Detect()	
Parameter	Nichts
Return-Wert	1 falls das Touch Panel berührt wurde 0 falls nicht
Beschreibung	Erkennt, ob das Touch Panel berührt wurde
Verwendung	Abfrage, ob gedrückt wurde (Polling)
TP_Get_Coordinates()	
Parameter	**unsigned int *x_coordinate** Farbe von zu zeichnenden Linien **unsigned int *y_coordinate** Breite von zu zeichnenden Linien
Return-Wert	0 falls die gelesenen Koordinaten innerhalb der Reichweite des Displays liegen 1 falls nicht
Beschreibung	Liest die Koordinaten einer Touch-Panel-Berührung und legt diese in übergebenen Variablen ab
Verwendung	Lesen der Druckkoordinaten und Speichern dieser Koordinaten in dafür vorhergesehene Variablen

Tabelle 11 Funktionen der Touch Panel Bibliothek

6.3 Funktionalitäten

6.3.1 Komponenten vorbereiten

Bevor die eigentliche Anwendung starten kann, müssen noch einige Vorbereitungen getroffen werden.
Um das TFT-Display und das Touch-Panel verwenden zu können, müssen diese zu allererst initialisiert werden, wofür die Funktion Initialize() definiert wurde.

```
530  void Initialize()
     {
       TFT_Init_ILI9341_8bit(320,240);
       TFT_Fill_Screen(CL_WHITE);

       ADC_Init();
       TP_Init(128, 64, 0, 1);
       TP_Set_ADC_Threshold(900);

       Init_Errors();
540  }
```

Abb. 42 Code-Auszug Initialize()

TFT-LCD intialiseren

Die Initialisierung des Displays läuft über den auf dem EasyTFT™ (siehe **5.1.6**) vorhandenen Display-Controller, in diesem Fall ein ILI9341-Bit. Dementsprechend muss auch die richtige Initialisierungs-Funktion aus der TFT Library benutzt werden, also TFT_Init_ILI9341_8bit(). Die beiden Parameter stehen für die Bildschirmauflösung (320x240).
Außerdem funktioniert die TFT-Bibliothek nur, wenn einige Konstanten vordefiniert werden. Diese beschreiben die Portverbindungen für die Signale zwischen TFT-Display und Mikrocontroller.

```
     // TFT display connections
     char TFT_DataPort at PORTC;
     sbit TFT_WR at PORTB1_bit;
380  sbit TFT_RD at PORTB0_bit;
     sbit TFT_CS at PORTD6_bit;
     sbit TFT_RS at PORTA2_bit;
     sbit TFT_RST at PORTD7_bit;

     char TFT_DataPort_Direction at DDRC;
     sbit TFT_WR_Direction at DDB1_bit;
     sbit TFT_RD_Direction at DDB0_bit;
     sbit TFT_CS_Direction at DDD6_bit;
     sbit TFT_RS_Direction at DDA2_bit;
390  sbit TFT_RST_Direction at DDD7_bit;
```

Abb. 43 Code-Auszug TFT-Display-Verbindungen

ADC initialisieren

Um die Eingabe-Signale aus dem Touch Panel verarbeiten zu können, müssen sie erst von einem A/D-Konverter in digitale Signale umgewandelt werden, weshalb auch der A/D-Konverter, der sich im Mikrocontroller befindet, initialisiert werden muss. Dies erfolgt durch die ADC_Init()-Methode aus der ADC Library.

Touch Panel initialisieren

Auch das Touch Panel muss mithilfe der Funktion TP_Init() aus der Touch Panel Library initialisiert werden. Wieder wird hier die Auflösung des Panels übergeben. Außerdem kann noch bestimmt werden, aus welchem ADC-Channel die X- bzw. Y-Koordinate gelesen wird. Da der ADC bei diesem Projekt ohnehin nur für das Touch Panel verwendet wird, werden die Kanäle 0 und 1 dafür hergenommen.
Wie die TFT-Bibliothek benötigt auch die Touch Panel-Bibliothek Konstanten, die die Portverbindungen angeben.

```
// Touch Panel module connections
sbit DriveA at PORTA2_bit;
sbit DriveB at PORTA3_bit;
sbit DriveA_Direction at DDA2_bit;
sbit DriveB_Direction at DDA3_bit;
```

Abb. 44 Code-Auszug Touch Panel Modulverbindungen

Touch Panel kalibrieren

Dies allein reicht noch nicht für die Inbetriebnahme des Panels. Es muss auch noch kalibriert werden, weshalb die Funktion Calibrate() definiert wurde, welche auf zwei Funktionen aus der Touch Panel-Bibliothek zugreift: TP_Calibrate_Bottom_Left() & TP_Calibrate_Upper_Right(). Mithilfe dieser Kalibrierung werden dem Touch Panel die minimalen bzw. maximalen Werte vorgegeben.

```
void Calibrate()
{
  TFT_Dot(0,239,CL_FUCHSIA);
  TFT_Set_Font(TFT_defaultFont,CL_TEAL,FO_HORIZONTAL);
  TFT_Write_Text("TOUCH BOTTOM LEFT",100,115);
  TP_Calibrate_Bottom_Left();
  Delay_ms(1000);

  TFT_Dot(0,239,CL_TEAL);
  TFT_Dot(319,0,CL_TEAL);
  Fill_Rectangle(100, 115, 250, 130, CL_WHITE, CL_WHITE);
  TFT_Set_Font(TFT_defaultFont,CL_TEAL,FO_HORIZONTAL);
  TFT_Write_Text("TOUCH UPPER RIGHT",100,115);
  TP_Calibrate_Upper_Right();

  Delay_ms(1000);
}
```

Abb. 45 Code-Auszug Calibrate()

Der Benutzer muss nun bei Start der Anwendung die linke untere Ecke bzw. die rechte obere Ecke berühren, um die Kalibrierung vorzunehmen. Anschließend wird der Startbildschirm gezeichnet und die eigentliche Anwendung beginnt.

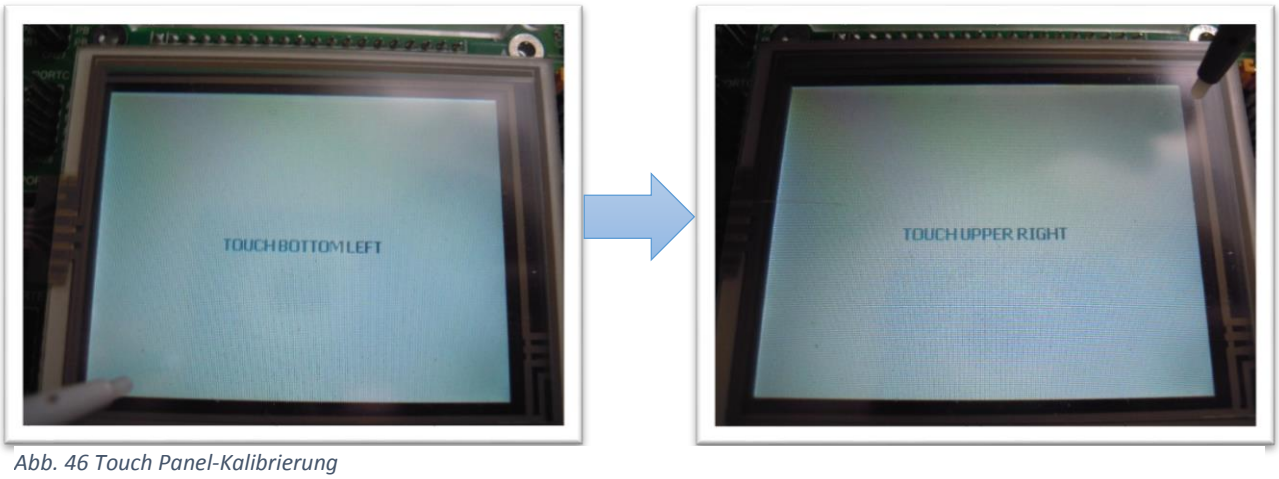


Abb. 46 Touch Panel-Kalibrierung

6.3.2 Allgemeiner Bildschirmaufbau

Das Layout für die Benutzeroberfläche lässt sich in zwei Bereiche untergliedern. Einerseits gibt es einen statischen Teil, der während der gesamten Laufzeit über großteils gleich bleibt, und einen dynamischen Teil, der aufgrund von Benutzereingaben etc. immer wieder mit veränderten Inhalten neu gezeichnet werden muss.
Der statische Bereich besteht aus zwei Bereichen. Erstens gibt es eine Buttonleiste, über die man bequem andere Ansichten aufrufen kann. Zweitens gibt es die Statusleiste, bei der man über Symbole ebenfalls in andere Fenster navigieren kann.

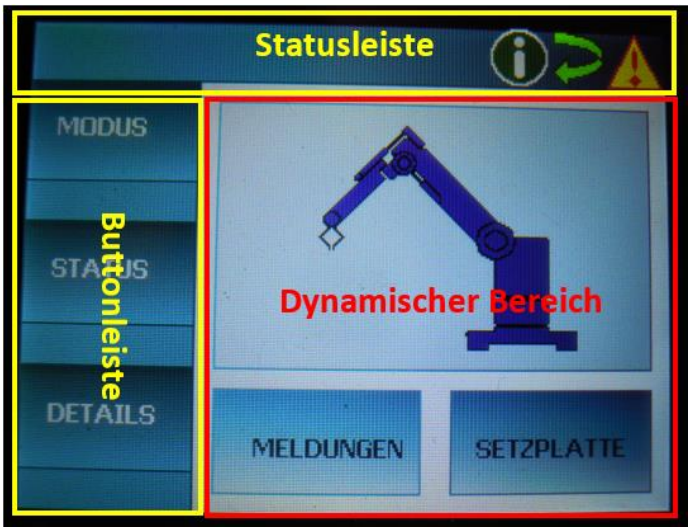


Abb. 47 Bereiche des Bildschirms

Wie die folgende Abbildung zeigt, ändert sich bei einem Wechsel der Ansicht nur der dynamische Bereich. Demnach wird auch nur dieser neu gezeichnet.

Im Falle der Abbildung unten geschieht die Änderung, wenn der Benutzer auf den Status-Button drückt.

Abb. 48 Änderung des dynamischen Bildschirm-Bereichs

Bei jeder Änderung des dynamischen Bereichs wird das alte Fenster gelöscht, indem eine weiße Fläche darüber gezeichnet wird. Auf dieser Basis können dann die neuen Ansichten aufgebaut werden.

Um das Zeichnen der Bildschirmelemente zu erleichtern, wurden zwei eigene Hilfsfunktionen definiert: **Fill_Rectangle()** und **Fill_Gradient_Rectangle()**.

Diese greifen jeweils auf die Methoden der TFT Library (siehe Kapitel **6.2.1**) TFT_Set_Pen(), TFT_Set_Brush() und TFT_Rectangle() zu. Somit benötigt der Benutzer dieser Funktionen nur mehr einen Funktionsaufruf anstelle von drei.

Fill_Rectangle()	
Parameter	**int x_upper_left** X-Koordinate der linken oberen Ecke **int y_upper_left** Y-Koordinate der linken oberen Ecke **int x_bottom_right** X-Koordinate der rechten unteren Ecke **int y_bottom_right** Y-Koordinate der rechten unteren Ecke **unsigned int border_color** Farbe des Rahmens **unsigned int fill_color** Farbe der Füllung
Return-Wert	Nichts
Beschreibung	Zeichnet ein umrahmtes, einfarbig gefülltes Rechteck an die Position, die durch die übergebenen Koordinaten bestimmt wird
Fill_Rectangle()	
Parameter	**int x_upper_left** X-Koordinate der linken oberen Ecke **int y_upper_left** Y-Koordinate der linken oberen Ecke **int x_bottom_right** X-Koordinate der rechten unteren Ecke **int y_bottom_right** Y-Koordinate der rechten unteren Ecke **unsigned int color1** obere Farbe des Farbverlaufs **unsigned int color2** untere Farbe des Farbverlaufs
Return-Wert	Nichts
Beschreibung	Zeichnet ein Rechteck an die durch übergebene Koordinaten bestimmte Position und füllt es mit einem vertikalen Farbverlauf

6.3.3 Bilder zeichnen

Die Benutzeroberfläche enthält einige Bilder, die in Form von Bitmaps statisch im Programmcode gespeichert sind. Bitmaps brauchen in diesem Fall sehr wenig Speicherplatz und sind deswegen für die Mikrocontrollerprogrammierung bestens geeignet, da hier nur geringe Speicherkapazitäten zur Verfügung stehen.

Um den benötigten Speicher noch geringer zu halten, wird die Bitmap durch ein Char-Array repräsentiert. Damit kann zwar nicht mehr jede mögliche Farbe repräsentiert werden, aber speicherintensivere Datentypen, wie z.B. Integer, würden bei diesen Mengen an Array-Elementen zu viel Platz verbrauchen. Derzeit werden nur zweifarbige Bilder unterstützt (inkl. Transparenz).

Um die benötigten Bilder, die mit GIMP erstellt und anschließend als .png oder .jpg abgespeichert worden sind, in C-Code zu konvertieren, wurde folgendes Tool von Digole Solutions, einem Hersteller von verschiedenen Hardwarekomponenten, verwendet:

http://www.digole.com/tools/PicturetoC_Hex_converter.php

Die folgende Abbildung zeigt eines dieser Char-Arrays, das ein Warndreieck darstellt, welches später am TFT-Display gezeichnet wird.

MotomanGUI_TFT.c

```
/* Bitmaps für auszugebende Bilder*/
// Warn-Dreieck
const char triangle_bmp[841] = {
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,1,2,1,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,1,2,1,1,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,1,2,2,2,1,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,1,2,2,2,1,1,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,1,2,2,1,2,2,1,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,1,1,2,2,1,2,2,1,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,1,2,2,1,1,1,2,2,1,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1,1,2,2,1,1,1,2,2,1,1,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,1,2,2,2,1,1,1,2,2,2,1,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,1,1,2,2,2,2,1,1,1,2,2,2,1,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,1,2,2,2,2,2,1,1,1,2,2,2,2,1,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,1,2,2,2,2,2,1,1,1,2,2,2,2,1,1,0,0,0,0,0,0,
0,0,0,0,0,0,0,1,2,2,2,2,2,2,1,1,1,2,2,2,2,2,1,0,0,0,0,0,0,
0,0,0,0,0,0,1,1,2,2,2,2,2,2,1,1,1,2,2,2,2,2,1,1,0,0,0,0,0,
0,0,0,0,0,0,1,2,2,2,2,2,2,2,1,1,1,2,2,2,2,2,2,1,0,0,0,0,0,
0,0,0,0,0,1,1,2,2,2,2,2,2,2,1,1,1,2,2,2,2,2,2,1,1,0,0,0,0,
0,0,0,0,0,1,2,2,2,2,2,2,2,2,1,1,1,2,2,2,2,2,2,2,1,0,0,0,0,
0,0,0,0,1,1,2,2,2,2,2,2,2,2,1,1,1,2,2,2,2,2,2,2,1,1,0,0,0,
0,0,0,0,1,2,2,2,2,2,2,2,2,2,1,1,1,2,2,2,2,2,2,2,1,0,0,0,0,
0,0,0,1,1,2,2,2,2,2,2,2,2,2,1,1,1,2,2,2,2,2,2,2,1,1,0,0,0,
0,0,0,1,2,2,2,2,2,2,2,2,2,2,1,1,1,2,2,2,2,2,2,2,1,0,0,0,0,
0,0,1,1,2,2,2,2,2,2,2,2,2,2,1,1,1,1,2,2,2,2,2,2,2,1,1,0,0,
0,0,1,2,2,2,2,2,2,2,2,2,2,1,1,1,1,1,1,2,2,2,2,2,2,2,1,0,0,
0,1,1,2,2,2,2,2,2,2,2,2,2,1,1,1,1,1,2,2,2,2,2,2,2,2,1,1,0,
0,1,2,2,2,2,2,2,2,2,2,2,2,1,1,1,2,2,2,2,2,2,2,2,2,2,1,0,
1,1,2,2,2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,2,2,2,1,1,
1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
};
```

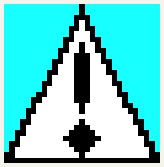


Abb. 49 Bitmap für Warndreieck

Das Bild hat eine Auflösung von 29x29, weshalb das Bitmap-Array 841 (29x29) Stellen benötigt. Die Nullen repräsentieren einen transparenten Hintergrund während die Werte 1 & 2 jeweils verschiedene Farben darstellen.

Damit diese Bitmaps nun auf dem TFT-Display als Bilder dargestellt werden, gibt es die Funktion **Draw_Bicolor_Image()**, die in folgender Tabelle grob und später noch genauer beschrieben wird:

Draw_Bicolor_Image()	
Parameter	**int x_upper_left** X-Koordinate der linken oberen Ecke, von der aus das Bild gezeichnet werden soll **int y_upper_left** Y-Koordinate der linken oberen Ecke, von der aus das Bild gezeichnet werden soll **unsigned int color1** erste Farbe im Bild **unsigned int color2** zweite Farbe im Bild **int img_height** Höhe des Bildes **int img_width** Länge des Bildes **unsigned int to_draw** Bildkonstante, die das zu zeichnende Bild repräsentiert
Return-Wert	Nichts
Beschreibung	Zeichnet ein zweifarbiges Bild anhand einer übergebenen Bildkonstante von einer Position aus, die durch die übergebenen Koordinaten gekennzeichnet wird.

Tabelle 12 Funktionsbeschreibung von Draw_Bicolor_Image()

Damit die Methode weiß, an welcher Bitmap sie sich zu orientieren hat, sprich, welches Bild gezeichnet werden soll, wird jedes Bild über eine Bildkonstante repräsentiert.

```
// Bildkonstanten
#define TRIANGLE 1
#define AUTOMATIC 2
#define INFO 3
#define HAND 4
#define ARROW 5
#define MOTOMAN 6
```

Abb. 50 Bildkonstanten

In der Funktion wird das zu zeichnende Bild Pixel für Pixel anhand einer übergebenen Bildhöhe und –länge in einer Schleife durchlaufen.

In erwähnter Schleife wird mit einem Switch die Bildkonstante angesehen und je nach dieser wird das nächste zu zeichnende Pixel aus der entsprechenden Bitmap gelesen. Anhand des Wertes der jeweiligen Position im Bitmap-Array wird dann mit der Methode TFT_Dot ein Pixel auf dem Display mit einer entsprechenden Farbe ausgemalt.

```
switch(to_draw)
{
     case 1:
          if (triangle_bmp[j+(i*img_width)] != 0)
          {
             if (triangle_bmp[j+(i*img_width)] == 1)
             {
                TFT_Dot(x_upper_left+j, y_upper_left+i, color1);
             }
             if (triangle_bmp[j+(i*img_width)] == 2)
             {
                TFT_Dot(x_upper_left+j, y_upper_left+i, color2);
             }
          }
     break;
     case 2:
          if (automatic_bmp[j+(i*img_width)] != 0)
          {
             if (automatic_bmp[j+(i*img_width)] == 1)
```

Abb. 51 Code-Auszug Draw_Bicolor_Image()

Das Dreieck aus *Abb. 49 Bitmap für Warndreieck* wird nun folgendermaßen auf dem TFT-Display gezeichnet:

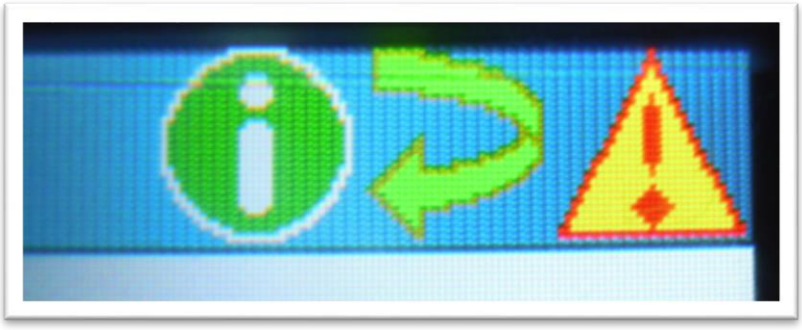


Abb. 52 Bilddarstellung auf TFT-Display

6.3.4 Startbildschirm

Der Startbildschirm wird bei Start der Anwendung gleich nach der Kalibrierung des Touch Panels angezeigt. Von dort aus lässt sich auf alle anderen Ansichten navigieren. Außerdem zeigt dieser Bildschirm ein simples Bild vom Roboter.

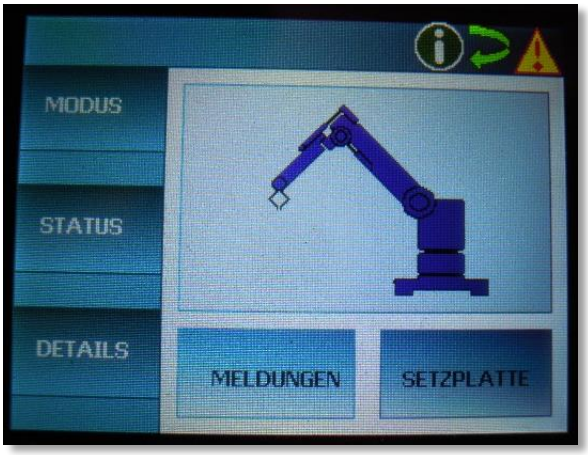


Abb. 53 Startbildschirm

6.3.5 Setzplattenansicht

Auf diese Ansicht kann man nur über den Startbildschirm gelangen. Sie dient zur Darstellung der Setzplatte, an der der Roboter gerade baut. Diese wird durch ein Array repräsentiert, das vorerst statisch im Code hinterlegt ist.

```
/* Setzplatten-Array
* 0 = frei
* 1 = belegt
* -1 = soll frei bleiben
*/
char plate[16] = {
  1,-1,-1,-1,
  1,1,-1,-1,
  -1,0,0,-1
  -1,-1,0,0};
```

Abb. 54 Code-Auszug Setzplattenarray

Die Elemente des Arrays können jeweils die Werte 0, 1 oder -1 belegen. -1 steht dabei für ein freibleibendes Feld, 0 für ein Feld, das noch mit einem Legostein besetzt werden soll und 1 steht für ein bereits belegtes Feld. Ziel ist es, alle 0er später in 1er zu verwandeln, sprich, der Roboter sollte alle unbelegten Felder belegen.
Ein Feld soll jeweils einen 2x2-Noppen-Legostein repräsentieren.

Im linken Bild wird die Setzplatte dargestellt, wie sie momentan durch das Setzplattenarray aus dem Code-Auszug repräsentiert wird und rechts wird gezeigt, wie die fertige Setzplatte aussehen soll, wobei die hellblauen Steine für die Teile stehen, die noch nicht gebaut sind.

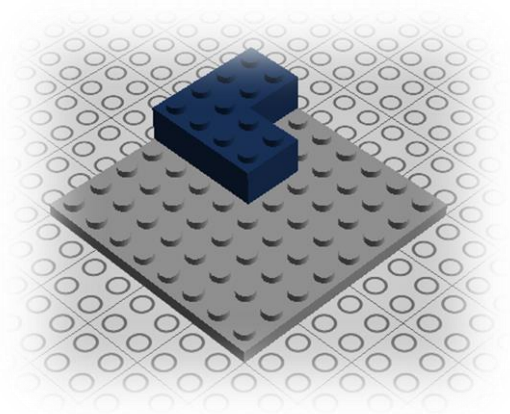


Abb. 55 unfertige Setzplatte laut Array

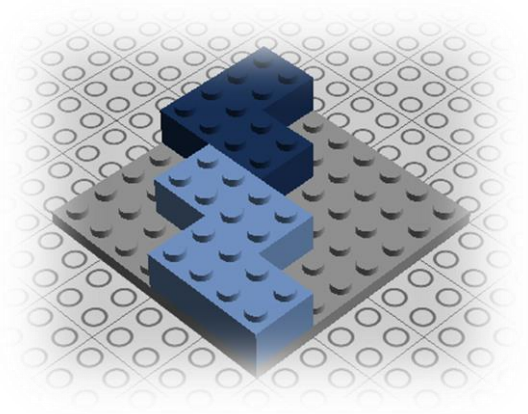


Abb. 56 fertige Setzplatte laut Array

Am TFT-Display wird die Setzplatte auf der Setzplattenansicht nun folgendermaßen dargestellt:

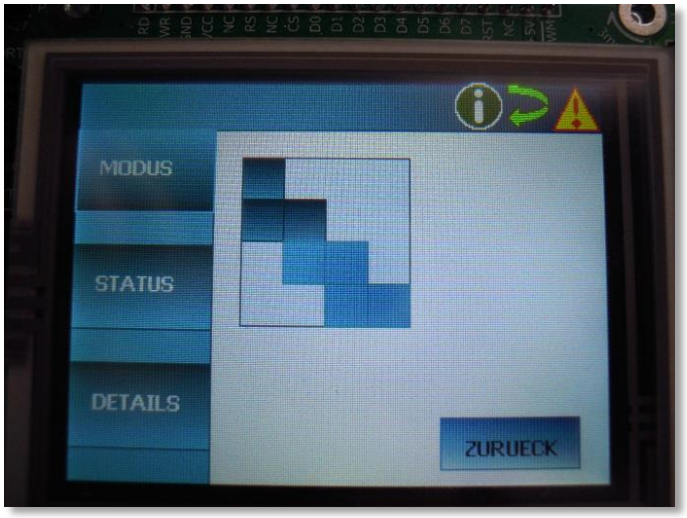


Abb. 57 Setzplattenbildschirm

Die mit dem Farbverlauf gefüllten Kästchen stehen für die bereits gesetzten Steine, die dunkleren, einfarbigen für die noch zu setzenden und der hellere Hintergrund repräsentiert jene Felder, die leer bleiben sollen.

Um die im Bild dargestellte Setzplatte zu zeichnen, wurde eine Methode Draw_Plate() definiert.

Draw_Plate()	
Parameter	**int x** X-Koordinate der linken oberen Ecke, von der aus die Platte gezeichnet werden soll **int y** Y-Koordinate der linken oberen Ecke, von der aus die Platte gezeichnet werden soll **int width** Länge der zu zeichnenden Setzplatte in Pixel **int height** Höhe der zu zeichnenden Setzplatte in Pixel **int img_height** Breite der Setzplatte in 2x2-Noppen-Bausteinen **int plate_width** Länge der Setzplatte in 2x2-Noppen-Bausteinen
Return-Wert	Nichts
Beschreibung	Zeichnet eine Setzplatte, die verbaute, noch nicht verbaute und leerbleibende Felder veranschaulicht, anhand des Setzplattenarrays

Tabelle 13 Funktionsbeschreibung Draw_Plate()

Die Größe des Feldes, in das die Setzplatte gezeichnet werden soll, ist variabel. Länge bzw. Höhe eines einzelnen Feldes (tile_width, tile_width) ergibt sich aus der übergebenen Länge bzw. Höhe, die durch die Anzahl an platzierbaren Bausteinen (Breite in Bausteinen/Länge in Bausteinen) dividiert wird.

```
void Draw_Plate(int x, int y, int width, int height,
                    int plate_width, int plate_height)
{
    int i, j;
    int tile_width = width/plate_width;
    int tile_height = height/plate_height;

    for (i = 0; i < plate_height; i++)
    {
680     for (j = 0; j < plate_width; j++)
        {
```

Abb. 58 Code-Auszug Draw_Plate() Größen

Anschließend wird das Setzplattenarray, dessen Größe durch die Parameter plate_width & plate_height bestimmt wird, in zwei Schleifen durchlaufen.

In den Schleifen wird der Wert an der jeweiligen Position im Array ermittelt. Anhand dieses Wertes werden dann die verschiedenen Rechtecke gezeichnet: Ein Farbverlauf-Rechteck im Falle einer 1, ein sehr helles Rechteck im Falle einer -1 und ein einfarbiges, umrahmtes Rechteck im Falle einer 0.

```
            switch (plate[j+(i*plate_width)])
            {
                  // Feld muss nicht besetzt werden
                  case -1:
                        Fill_Rectangle(x+(j*tile_width),y+(i*tile_height),
                                       x+(j*tile_width)+tile_width,
                                       y+(i*tile_height)+tile_height,
                                       light_teal, lighter_teal);
                        break;
                  case 0:
                        Fill_Rectangle(x+(j*tile_width),y+(i*tile_height),
                                       x+(j*tile_width)+tile_width,
                                       y+(i*tile_height)+tile_height,
                                       CL_TEAL, light_teal);
                        break;
                  case 1:
                      Fill_Gradient_Rectangle(x+(j*tile_width),y+(i*tile_height),
                                       x+(j*tile_width)+tile_width,
                                       y+(i*tile_height)+tile_height,
                                       dark_teal, CL_TEAL);
                        break;
            }
      }
```

Abb. 60 Code-Auszug Draw_Plate() Unterscheidung

Um nun das Ergebnis aus *Abb. 57 Setzplattenbildschirm* zu erhalten, wird daher folgender Funktionsaufruf benötigt:

```
      // Platte zeichnen (100x100)
      Fill_Rectangle(102,47,202,147,dark_teal,lighter_teal);
      Draw_Plate(102,47,100,100,4,4);
```

Abb. 61 Code-Auszug Draw_Plate()-Aufruf

6.3.6 Auswahl des Betriebsmodus

Der Industrieroboter soll auf 3 verschiedene Arten betrieben werden können:

Automatischer Betrieb

In diesem Betriebsmodus muss sich der Benutzer nicht darum kümmern, wie die Bausteine auf die Setzplatte kommen. Der Roboter produziert die Setzplatten automatisch, bis ihm die Steine ausgehen, er ausgeschaltet wird oder der Betriebsmodus gewechselt wird.

Manueller Betrieb

Wird der Roboter manuell betrieben, so wird er nicht durch das Roboterprogramm gesteuert, das ihm sagt, wie er die Achsen bewegen soll, um die Bausteine aufzubauen. Stattdessen kann ihn der Benutzer manuell über das ursprüngliche, wuchtige Interface-Panel (siehe Kapitel 2**15.1.1 Yaskawa Motoman® Industrieroboter**) bedienen und die Achsen nach Lust und Laune bewegen.

Tipp-Betrieb

Befindet sich der Roboter im Tipp-betrieb, so lassen sich die Bewegungen Schritt-für-Schritt ausführen. Es wird wie beim automatischen Betrieb das Roboterprogramm für die Achsenbewegungen verwendet, allerdings kann der Benutzer beim Tipp-Betrieb bestimmen, wann der Roboter weiterbauen soll und wann er aufhören soll. Dafür werden ihm ein Start- und ein Stop-Button zur Verfügung gestellt.

```
/* Betriebsmodus
 * 1 = automatischer Betrieb
 * 2 = manueller Betrieb
 * 3 = Tippbetrieb
 */
#define AUTO_MODE 1
#define MANU_MODE 2
#define TIPP_MODE 3
char selected_mode = 1;
```

Abb. 62 Code-Auszug Modus-Konstanten

Für die Auswahl einer dieser Betriebsmodi steht auf dem TFT-Display eine eigene Ansicht zur Verfügung, welche durch einen Button in der Buttonleiste oder über ein Symbol auf der Symbolleiste aufgerufen werden kann. Der gewünschte Betriebsmodus wird durch Druck auf einen Button aktiviert.
Für jeden Betriebsmodus gibt es eine Konstante. Der aktuelle Betriebsmodus wird über die Variable selected_mode bestimmt.

Die Betriebsmodus-Ansicht enthält drei Buttons für die Auswahl des Betriebsmodus, wobei der Button für den aktuellen Modus mit einer Umrahmung gekennzeichnet wird. In der Symbolleiste gibt es außerdem ein Symbol, das den aktuellen Betriebsmodus repräsentiert, wie in den folgenden Abbildungen beschrieben wird:

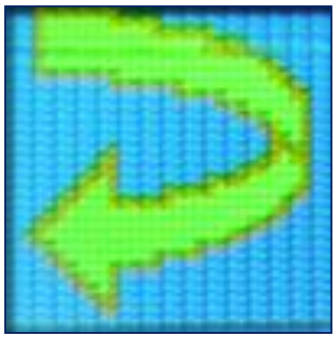


Abb. 65 Symbol Automatik

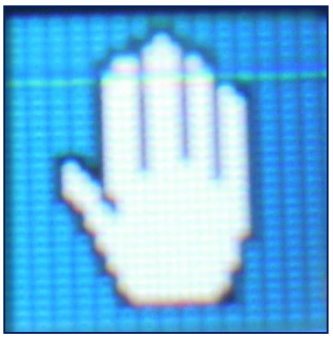


Abb. 63 Symbol manuell

Abb. 64 Symbol Tipp

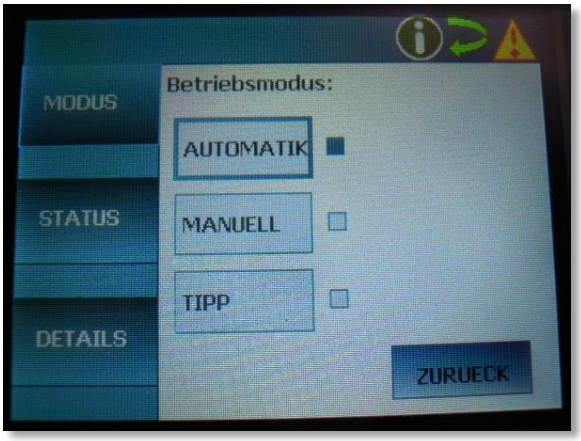


Abb. 67 Modus-Ansicht: Automatik

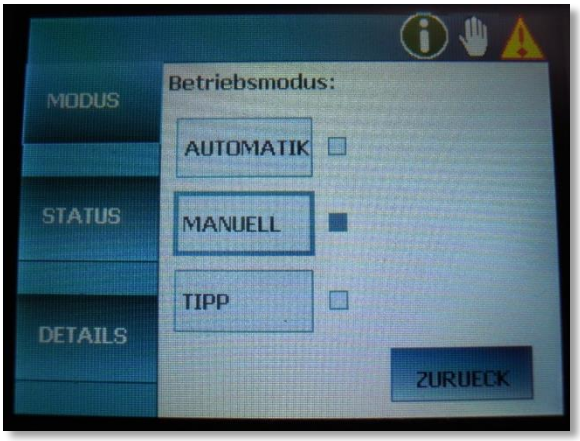


Abb. 66 Modus-Ansicht: Manuell

Bei Aktivierung des Tipp-Betriebsmodus wird die Ansicht um einen Start- und Stop-Button erweitert, mit dem der Benutzer später die Bewegungsabläufe des Roboters starten bzw. stoppen können soll.

Abb. 68 Modus-Fenster: Tipp-Betrieb

6.3.7 Statusanzeige des Greifers

Um die Bausteine in die "Hand" zu nehmen, benötigt der Roboter einen Greifer. Mit diesem kann er die Bausteine hochheben, auf die Setzplatte setzen und dort wieder auslassen. Besagter Greifer kann vier verschiedene Zustände (Status) annehmen:

- Offen
- Geschlossen
- Einen Baustein haltend
- Undefiniert

Der undefinierte Status beschreibt einen unbekannten Greiferzustand. Auftreten kann dies, wenn zum Beispiel der Greifer kaputt oder gar nicht am Roboter befestigt ist.

Der aktuelle Status wird durch die Variable gripper_state repräsentiert. Für die möglichen Zuständen sind Konstanten vordefiniert.

```
/* aktueller Greiferstatus
 * 1 = Greifer ist offen
 * 2 = Greifer ist geschlossen
 * 3 = Greifer hält einen Bauteil
 * 4 = undefinierter Status
 */
#define GRIPPER_OPEN 1
#define GRIPPER_CLOSED 2
#define GRIPPER_HOLDING 3
#define GRIPPER_UNDEFINED 0
char gripper_state = 0;
```

Abb. 70 Code-Auszug: Greifer-Konstanten

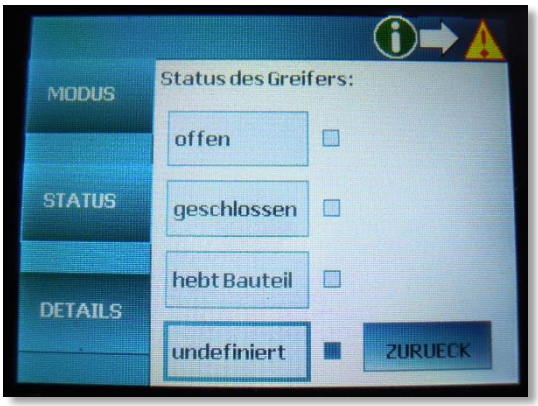


Abb. 69 Status-Bildschirm

Die Ansicht für den Greifer-Status kann über einen Button in der Buttonleiste betreten werden. Darin werden alle möglichen Zustände des Greifers angezeigt und der aktuelle Zustand umrahmt. Bestimmt soll dieser später über den Roboter werden, indem er ein Signal ausgibt, das den Greiferzustand repräsentiert. Derzeit ist der Status undefiniert, da kein Roboter und somit auch kein Greifer angeschlossen sind.

6.3.8 Details-Anzeige

In der Detail-Ansicht werden verschiedene statistische Details angezeigt, die rein der Information dienen. Angezeigt werden:

- die Anzahl der beim aktuellen Durchlauf bereits verbauten Bauteile
- die Anzahl der beim aktuellen Durchlauf fertiggestellten Setzplatten
- die Anzahl der Bausteine, die noch verbaut werden müssen, um die aktuelle Setzplatte fertigzustellen

Für die Speicherung der eben genannten Werte werden folgende Variablen verwendet:

```
/* Statistische Zahlen*/
// Anzahl verwendeter Bauteile bei der aktuellen Platte
unsigned int bricks_count = 3;
// Anzahl fertiger Setzplatten
unsigned int finished_plates = 4;
// Anzahl Bauteile, die für die aktuelle Setzplatte noch verbaut werden müssen
unsigned int bricks_to_next_plate = 5;
```

Abb. 71 Code-Auszug Details Variablen

Diese Daten werden dann später über den Roboter ermittelt.

Um für die Details-Ansicht die drei Variablen auf das TFT-Display zu schreiben, wurde die Methode sprintf() verwendet, welche normalerweise in der C-Standardbibliothek zu finden ist. Da uns diese je doch in mikroC Pro for AVR® (siehe Kapitel **5.2.1.2**) nicht zur Verfügung steht, stellt uns mikroC eine Mikrocontroller-Variante dieser Funktion zur Verfügung.

Die Ansicht kann entweder über das Info-Symbol in der Statusleiste oder über den Detail-Button in der Buttonleiste erreicht werden.

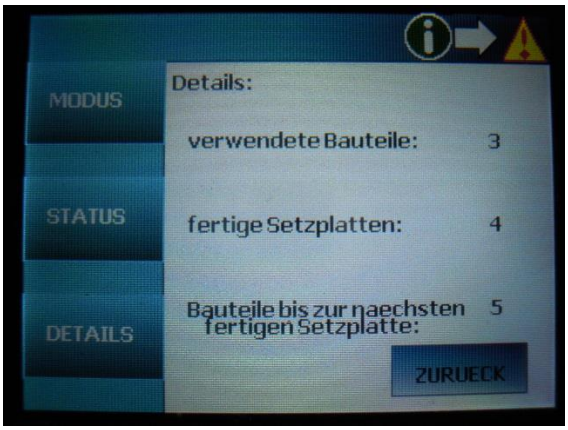


Abb. 73 Detailbildschirm

Abb. 72 Info-Symbol

6.3.9 Fehlermeldungen

Für den Umgang mit auftretenden Fehlern wurde die Struktur error_t definiert. Sie umfasst eine Zeichenkette (err_msg), die eine Fehlermeldung von bis zu 30 Zeichen speichern kann und einen Integer (priority), der die Priorität des Fehlers speichert. Diese gibt an, wie schwerwiegend der Fehler ist. Der Meldungstext ist auf 30 Zeichen beschränkt, weil dies die ungefähre Anzahl der Zeichen ist, die in einer Reihe am dynamischen Teil des TFT-Displays ausgegeben werden können ohne dass sie vom Rand verschluckt werden. Außerdem spart dies Speicherplatz, was in der Mikrocontroller-Programmierung eine nicht irrelevante Rolle spielt.
Alle Fehlermeldungen werden in dem Array "errors" gespeichert, welches insgesamt 10 Status- oder Fehlermeldungen zur selben Zeit beinhalten kann.

```
/* Datentyp für Fehler- oder Statusmeldungen
 *          err_msg = Meldungstext, der ausgegeben wird, max. 30 Zeichen
 *          priority = Priorität/Wichtigkeit des Fehlers
 */
typedef struct error_s {
        char err_msg[30];
        unsigned int priority;
} error_t;

// Liste der aufgetretenen Fehler- oder Statusmeldungen
error_t errors[10];
```

Abb. 74 Code-Auszug Struktur für Fehlermeldungen

Um mit diesem Array arbeiten zu können, muss es zuallererst einmal initialisiert werden. Dafür gibt es die Funktion Init_Errors(), die zu Beginn des Programmes aufgerufen wird.

```
void Init_Errors()
{
    int i;
    error_t std_error;
    std_error.priority = 255;
    for (i = 0; i < sizeof(errors) / sizeof(errors[0]); i++)
    {
        errors[i] = std_error;
    }
}
```

Abb. 75 Code-Auszug Init_Errors()

Dabei werden alle Elemente des Arrays mit einem sogenannten Standard-Error belegt (std_error). Dieser hat eine Priorität von 255 und repräsentiert ein "leeres" Error-Element. In Programmiersprachen, die auf höheren Ebenen arbeiten, wie etwa Java oder C#, könnte man sich diesen Vorgang ersparen, da hier alle Variablen nicht-elementarer Datentypen ab dem Zeitpunkt ihrer Definition mit dem Wert **null** versehen werden. In C hingegen weiß man nicht, was in einer Variable steht, die noch nicht definiert worden ist. Darum braucht man einen Wert, der ein noch nicht initialisiertes Element beschreibt. Ansonsten würde man beim späteren Arbeiten mit diesem Array nicht wissen, ob sich an den jeweiligen Positionen tatsächlich gültige Werte befinden. Der Standard-Error ist mit dem Wert **/0** vergleichbar, der in C bei Zeichenketten das Ende eines Strings bzw. das Ende eines Char-Arrays angibt.

Für das Aufnehmen einer neuen Fehlermeldung in die Liste aller Fehler wurde die Funktion Add_Error() definiert.

Add_Error()	
Parameter	**char *msg** Meldungstext des Fehlers **unsigned char priority** Priorität des Fehlers
Return-Wert	Nichts
Beschreibung	Erzeugt einen neuen Fehler mit den gegebenen Parametern und nimmt ihn in die Liste aller Fehlermeldungen (error-Array) auf, indem der Fehler auf die nächste freie Position gelegt wird.

Tabelle 14 Funktionsbeschreibung Add_Error()

Um die nächste freie Position zu ermitteln wird das Array durchlaufen und nach Standard-Errors durchsucht, mit welchen das Array in der Funktion Init_Errors() befüllt wurde. Standard-Errors werden dadurch erkannt, dass sie eine Priorität von 255 besitzen. Sobald das Programm also auf besagten Standard-Error stößt, kann an dieser Position ein neu erzeugter Fehler, der sich aus den übergebenen Parametern zusammensetzt, eingefügt werden.

Auf dem Display werden alle Fehler, die sich momentan in der Fehlerliste befinden, durch die Fehleransicht ausgegeben. Über das Warndreieck in der Symbolleiste oder durch einen entsprechenden Button am Startbildschirm lässt sich diese aufrufen.

Zu Testzwecken wurden hier bereits drei Fehlermeldungen angelegt. Die Anzeige sieht folgendermaßen aus:

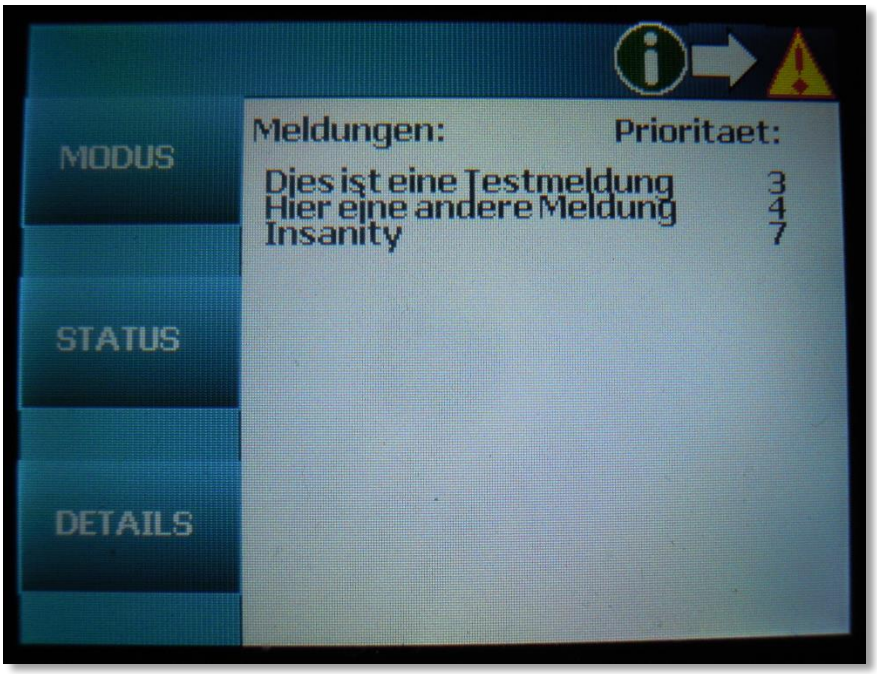


Abb. 77 Fehlermeldungs-Bildschirm

Abb. 76 Fehler-Symbol

6.4Benutzereingabenverwaltung

Damit der Benutzer auch mit der Benutzeroberfläche auf dem Display interagieren kann, werden die Eingaben, welche er über das Touch Panel tätigt, im Programmcode verwaltet.

Dabei fragt das Programm zyklisch nach, ob eine Benutzereingabe erfolgt ist, sprich, ob das Touch Panel berührt worden ist. Die Methode nennt man Polling. Die Abfrage erfolgt in einer Endlosschleife in der main()-Methode des Programmes:

```
    while(1)
    {
      if(TP_Press_Detect())
      {
              if (TP_Get_Coordinates(&x_coord, &y_coord) == 0)
950           {
```

Abb. 78 Schleife für Benutzereingaben

Sobald eine Berührung erkannt wurde, werden die Koordinaten in die beiden Integer-Variablen x_coord und. y_coord gespeichert. Da die Auflösung des Touch Panels(128x64) jedoch nicht mit der Auflösung des TFT-Displays(320x240) übereinstimmt, müssen diese noch umgerechnet werden.

Das Touch Panel besitzt 128 Pixel in der Breite, das Display 320. Für die Umrechnung der X-Koordinate aus dem Touchpanel wird somit ein Faktor von 320/128 = **2,5** benötigt.
In der Länge hat das Touch Panel 64 Bildpunkte während es vom Display mit 240 in den Schatten gestellt wird. Zur Umrechnung der Y-Koordinate besteht also ein Faktor von 240/64 = **3,75**.

Somit können die Koordinaten, die aus der Benutzereingabe gewonnen werden, verarbeitet werden.

Zuallererst wird abgefragt, ob die Berührung innerhalb der Button-Leiste erfolgt ist. Als nächstes muss herausgefunden werden, welcher Button bzw. ob überhaupt ein Button betroffen ist, da sich in der Buttonleiste schließlich auch Leerräume befinden.

```
// ein Button aus der Buttonleiste wurde gedrückt
if((x_coord*2.5 > 0) && (x_coord*2.5 < 85) && (y_coord*3.75 > 30))
{

    TFT_Set_Font(TFT_defaultFont,CL_WHITE,FO_HORIZONTAL);
    // Modus-Button wurde gedrückt
    if((y_coord*3.75 < 80))
    {
        Fill_Gradient_Rectangle(0,30,85,80,light_teal,dark_teal);
        TFT_Write_Text("MODUS",15,45);
        Draw_Mode_Window();
    }
    // Status-Button wurde gedrückt
    if((y_coord*3.75 > 100) && (y_coord*3.75 < 150))
```

Abb. 79 Code-Auszug Benutzereingabe Buttonleiste

Dann können je nach gedrücktem Button entsprechende Maßnahmen getroffen werden. Wird z.B. der Modus-Button gedrückt, wird im dynamischen Teil der Anzeige auf dem TFT-Display die Betriebsmodus-Ansicht (siehe Kapitel **6.3.6**) aufgebaut (Draw_Mode_Window()).

Neben der Buttonleiste gibt es natürlich auch noch die Statusleiste. Hier kann der Benutzer eines der Symbole drücken.

Nach dem gleichen Schema wie bei der Buttonleiste wird zu allererst abgefragt, ob sich die Berührung in der Statusleiste befindet. Dann wird anhand der gedrückten Symbole unterschieden.

```
    // ein Symbol aus der Statusleiste wurde gedrückt
    else if((y_coord*3.75 > 0) && (y_coord*3.75 < 30))
    {
        // Info-Symbol wurde gedrückt
        if((x_coord*2.5 > 230) && (x_coord*2.5 < 260))
        {
                    Draw_Details_Window();
        }

        // Modus-Symbol wurde gedrückt
        if((x_coord*2.5 > 260) && (x_coord*2.5 < 290))
        {
```

Abb. 80 Code-Auszug Benutzereingabe Statusleiste

Dementsprechend werden wieder die verschiedenen Ansichten gezeichnet, z.B. das Details-Window bei Druck des Info-Symbols.

Befindet sich die Berührung seitens des Benutzers weder in der Button- noch in der Statusleiste, so kann sie nur mehr im dynamischen Teil des Displays erfolgt sein. In diesem Fall wird es ein wenig komplizierter, denn da sich der dynamische Teil stets ändert, muss zuerst einmal herausgefunden werden, in welchem Fenster man sich gerade befindet, da jedes Fenster andere interaktive Elemente (z.B. Buttons) darstellt und daher andere Flächen aufweist, die auf Berührungen reagieren sollen.

Zur Bestimmung der aktuellen Ansicht sind Konstanten für alle möglichen Ansichten definiert worden. Die Variable current_window gibt dabei das aktuelle Fenster an.

```
/* aktuelles Fenster
 * 0 = Startfenster
 * 1 = Setzplatte
 * 2 = Betriebsmodus
 * 3 = Status
 * 4 = Details
 * 5 = Fehler
 */
#define MAINWINDOW 0
#define PLATEWINDOW 1
#define MODEWINDOW 2
#define STATUSWINDOW 3
#define DETAILSWINDOW 4
#define ERRORWINDOW 5
char current_window = 0;
```

Abb. 81 Code-Auszug Fensterkonstanten

Sobald die Ansicht gewechselt wird, wird auch der Wert von current_window geändert. Somit kann problemlos abgefragt werden, welches Fenster aktiv ist.

Je nach aktiver Ansicht können so verschiedene Aktionen ausgeführt werden. Beispielsweise reagieren die Buttons für die Auswahl des Betriebsmodus auch wirklich nur im Betriebsmodus-Fenster auf Berührungen.

```
// der dynamische Teil des Bildschirms wurde berührt
else
{
     // welches Fenster ist aktiv?
     switch(current_window)
     {
      case MAINWINDOW:
           if((y_coord*3.75 > 180) && (y_coord*3.75 < 230))
           {
                // Meldungen-Button wurde gedrückt
                if((x_coord*2.5 > 95) && (x_coord*2.5 < 195))
                {
                     Draw_Error_Window();

                }
                // Setzplatte-Button wurde gedrückt
                if((x_coord*2.5 > 210) && (x_coord*2.5 < 310))
                {
                     Draw_Plate_Window();
                }
           }
           break;
      case PLATEWINDOW:
           // Zurück-Button gedrückt
           if((y_coord*3.75 > 200) && (y_coord*3.75 < 230)
              && (x_coord*2.5 > 220) && (x_coord*2.5 < 302))
```

Abb. 82 Code-Auszug Benutzereingaben im dynamischen Teil des Layouts

In diesem Beispiel wird etwa die Reaktivität des Startbildschirms implementiert. Die beiden Buttons – Meldungen und Setzplatte – gibt es nur im Startbildschirm. Und deshalb sollten sie auch nur dann auf Benutzereingaben reagieren, wenn dieser gerade auf dem Display dargestellt wird.

7 Profibus

7.1 Profibus® Allgemein

Profibus® ist die Abkürzung für Process Field Bus und ist ein Standard in der Automatisierungstechnik. Die Entwicklung geht auf ein Projekt des Verbunds in Deutschland 1987 zurück, wo 21 Unternehmen bzw. Institutionen an einem Rahmenplan namens "Feldbus" arbeiteten.
Der Zweck dieses Projekts sollte die Verwirklichung und Ausbreitung des bitseriellen Feldbusses sein.

Es gibt drei Varianten des Profibus®:

- PROFIBUS® DP (Dezentrale Peripherie)
- PROFIBUS® PA (Prozess-Automation)
- PROFIBUS® FMS (Fieldbus Message Specification)

Da für die Diplomarbeit nur der PROFIBUS® DP relevant ist, wird hier auch nur dieser genauer erläutert.

7.2 Profibus® DP

Der Profibus® DP arbeitet auf der Feldebene, das heißt in der Automatisierungspyramide auf Level 1.

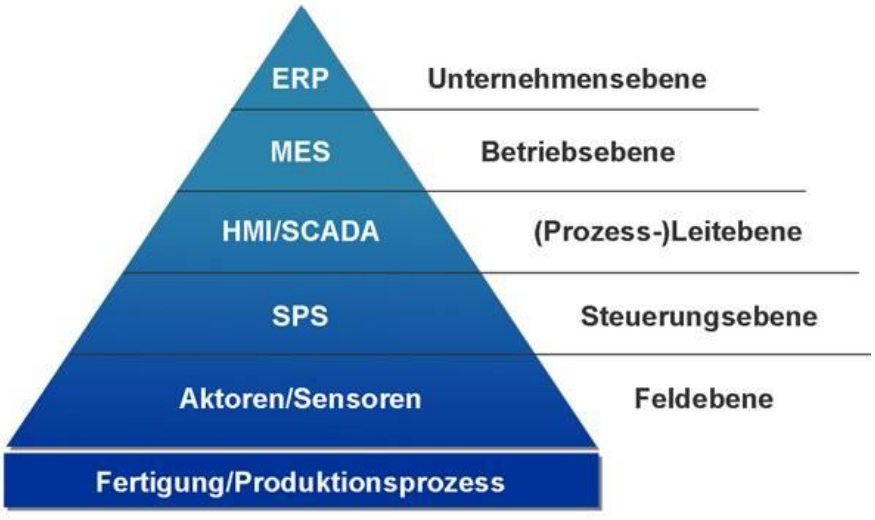


Abb. 83 Automatisierungspyramide

Auf dieser Schicht kommunizieren die zentralen Automatisierungsgeräte (Bsp. ATmega1284P/Mikrocontroller) mithilfe einer seriellen Verbindung mit den dezentralen Feldgeräten (Bsp. Roboterarm). Bei den DP-Grundfunktionen sind die notwendigen Funktionen für die Kommunikation festgelegt. Darüber hinaus sind auch azyklische Kommunikationsdienste möglich, diese werden auch DPV1-Funktionen genannt.

Bei der zyklischen Nutzdatenübertragung liest der Master zyklisch die Eingangsinformationen von den Slaves aus und schreibt die Ausgangsinformation an jene Slaves zurück. Beim PROFIBUS®-DP sind Übertragungsraten von bis zu 12 Mbit/s erreichbar. Somit entsteht die Möglichkeit, 1024 Bit Ein-/Ausgangsdaten auf 32 Teilnehmer in weniger als 2 ms zu verteilen.

Weiters gibt es auch leistungsfähige Funktionen zur Diagnose und der Inbetriebnahme, so wird der Datenverkehr sowohl von Seiten des Masters als auch von Seiten der/des Slaves überwacht.

Auch gibt es Funktionen für eine möglichst rasche Fehlerlokalisierung. Dabei werden die Diagnosemeldungen beim Master zusammengefasst. Unterscheiden kann man dann diese in drei Stufen:

- **Stationsbezogene Diagnose**
  Meldungen bzgl. Betriebsbereitschaft – z.B. Übertemperatur
- **Modulbezogene Diagnose**
  Meldungen bzgl. Fehler innerhalb eines I/O-Segments – z.B. 8-Bit-Ausgangsmodul
- **Kanalbezogene Diagnose**
  Meldungen bzgl. Fehler eines einzelnen Ein-/Ausgangs – z.B. Kurzschluss von Ausgang 2

7.2.1 Konfiguration

Beim PROFIBUS®-DP sind sowohl Mono- als auch Multi-Master-Systeme möglich. Das heißt die Slaves können entweder von einem einzelnen Master, wie bei unserer Diplomarbeit, oder auch von mehreren Mastern Befehle

erhalten. Als Maximum in einem DP-System können 126 Geräte (Slaves und Master gemeinsam) an den Bus angeschlossen werden.
Bei der Festlegung zur Konfiguration des Systems werden Informationen wie die Anzahl der Stationen, Datenkonsistenz der I/O-Daten oder auch die verwendeten Busparameter erfasst.
In allen DP-Systemen sind verschiedene Geräte mit unterschiedlichen Rollen vorzufinden. Dabei gibt es drei Typen:

1. Profibus® DP Slave

Als Slave werden Peripheriegeräte verwendet. Das sind in erster Linie ausführende Geräte, wie zum Beispiel Ventile, Messumformer, oder wie in unserem Fall der Roboterarm.
Die Aufgabe des Slaves ist es Eingangsinformationen einzulesen und Ausgangsinformationen an die Peripherie weiter zu geben. Wie groß diese Informationen sein können hängt vom Gerät ab. Es sind aber maximal 246 Byte Eingangsdaten sowie 246 Byte Ausgangsdaten möglich.

2. DP Master Klasse 1 (DPM1)

Der DP Master Klasse 1 bildet die zentrale Einheit, welche durch einen festgelegten Nachrichtenzyklus Informationen mit den Slaves austauscht. In jedem Profibus®-DP System ist mindestens ein Master der Klasse 1 notwendig. Üblicherweise werden Geräte wie SPS oder PC verwendet um diese Aufgabe zu übernehmen.
Es werden folgende Master-Slave-Funktionen unterstützt:

- Erfassen von Diagnoseinformationen der DP-Slaves
- zyklischer Nutzdatenbetrieb
- Parametrierung und Konfigurierung der DP-Slaves
- Steuerung von DP-Slaves mit Steuerkommandos

Das User-Interface kann diese Anwendungsfunktionen eigenständig erledigen.

3. DB Master Klasse 2 (DPM2)

Beim Master der Klasse 2 handelt es sich um Engineering- bzw. Bediengeräte. Sie dienen dem Festlegen der Konfigurationen und Parametereinstellungen des DP-Systems und müssen nicht permanent am Bus angeschlossen sein. Im Gegensatz zum DP Master Klasse 1 agiert jener auf Klasse 2 nur zur Bedarfsdatenübertragung.

7.2.1.1 Adressierung

Die Adressen der Teilnehmer können relativ frei zwischen 0 und 126 vergeben werden. Dabei ist von Bedeutung, dass jeder Teilnehmer eine eindeutige Adresse besitzt.

Adresse	Zweck
0	reserviert für Diagnosewerkzeuge
1 … n	Die Master sollten die niedrigsten Adressen haben. Ein einzelner Master hat damit die Adresse 1.
n … 125	Slave Adressen
126	reserviert als Auslieferungsadresse (default) für Stationen, deren Adresse über den Bus eingestellt wird
127	Broadcast-Adresse

Tabelle 15 Profibus-Adressbereiche

7.2.1.2 GSD-Dateien

GSD-Dateien (Geräte spezifische Daten) sind jene Dateien, die Eigenschaften eines Profibus®-Teilnehmers beschreiben. Beispiele hierfür wären die Baudrate, Ein-/Ausgänge, Statusmeldungen, etc.
Dabei ist die Struktur dieser Dateien fix vorgeschrieben. Dies bedeutet, dass beim Anbinden eines Gerätes von einem anderen Hersteller die GSD-Datei in das Parametrierwerkzeug des Masters importiert werden muss. Die PNO (PROFIBUS Nutzer Organisation) verwaltet die GSD-Dateien aller Hersteller.

7.2.1.3 Zustände

Damit die Geräte austauschbar bleiben, gibt es bei PROFIBUS®-DP ein standardisiertes Systemverhalten. Dieses wird im Großen und Ganzen durch den DP Master Klasse 1 festgelegt, von dem der Betriebszustand entweder lokal oder mittels dem Bus vom Projektierungsgerät aus gesteuert werden kann. Während den Hauptzuständen ist der Kontakt zwischen Master und Diagnosetools möglich. Die Betriebszustände sind folgendermaßen definiert:

- Stop
  Es wird kein Datenaustausch zwischen Master und Slave vorgenommen.
- Clear
  Die Ausgänge der Slaves werden vom Master in sicherem Zustand gehalten, während dieser die Eingangsinformationen der Slaves liest.

- Operate
  Dieser Zustand ist aktiv, wenn gerade ein Datenaustausch stattfindet. Beim wiederholt auftretenden Zyklus werden die Eingänge der Slaves gelesen und die Ausgangsdaten zu ihnen übermittelt.

7.2.1.4 Überblick der Technologien

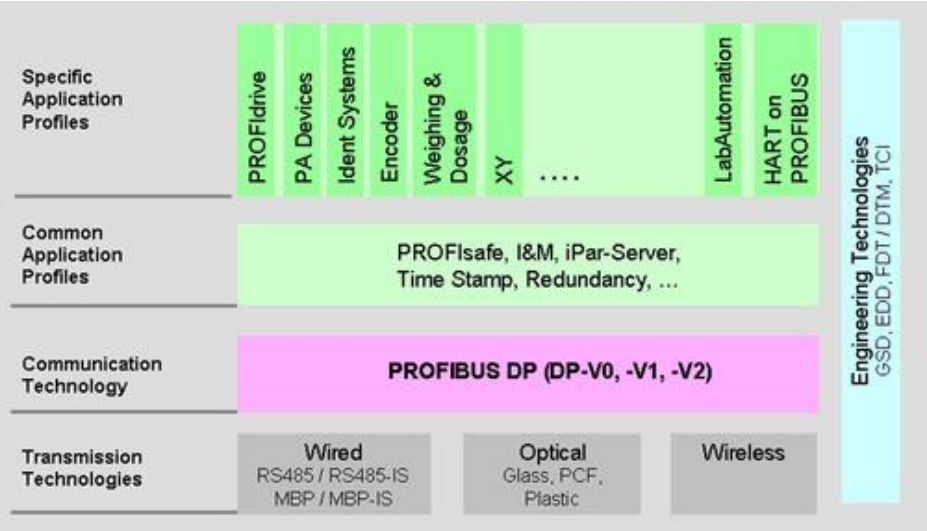


Abb. 84 Profibus Technologien

In der oben stehenden Grafik sind die Zusammenhänge zwischen den einzelnen Technologien, die zur Verwendung von PROFIBUS® notwendig sind, schön ersichtlich.

7.2.2 Übertragungstechniken

Als Übertragungstechnik nutzt Profibus®-DP das RS485. Dies ist heutzutage sehr verbreitet. Es arbeitet mit einer verdrillten Zweidrahtleitung und es sind Übertragungsraten von bis zu 12 MBaud möglich. Typischerweise werden die Feldgeräte in einer Busstruktur angeschlossen.

RS485 wird auch vereinzelt EIA-485 genannt und ist ein Schnittstellen-Standard bei der digitalen, leitungsgebundenen und differentiell seriellen Datenübertragung. Wegen der symmetrischen Signalübertragung ist es kaum anfällig für elektromagnetische Störungen.

Das Leitungspaar ist notwendig, da sowohl der invertierte, als auch der nichtinvertierte Pegel eines 1-Bit Datensignals übertragen wird. Der Empfänger bildet dann die Differenz der Spannungen und erstellt wieder das ursprüngliche Datensignal.

Durch den integrierten Widerstand beim Sender, ist dieser kurzschlussfest und wird durch das Gegensenden eines zweiten Senders nicht defekt.

Der RS485 unterscheidet sich von anderen Bussen dahingehend, dass nur die elektrischen Schnittstellenbedingungen definiert sind. Da das Protokoll anwendungsspezifisch gewählt werden kann, verstehen sich RS485 Geräte unterschiedlicher Hersteller meistens nicht. Sollen dennoch Daten übertragen werden, wird häufig auf UART gesetzt.

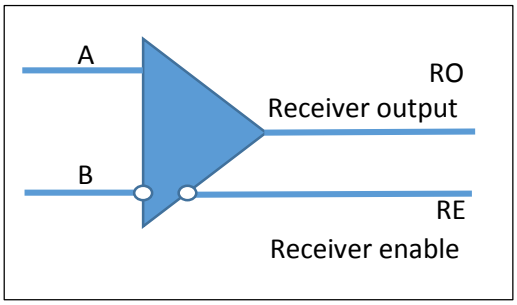


Abb. 86 RS485 Receiver

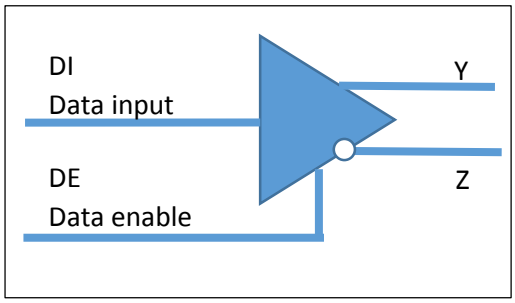


Abb. 85 RS485 Transmitter

7.2.2.1 RS485 Installation

Zur Verbindung der Netzwerk-Elemente wird bei der RS485 die Verwendung des Kabeltyps A dringend empfohlen. Beim Anschluss ist besonders darauf zu achten, dass die Datenleitungen nicht vertauscht werden. Auch sollten die Datenleitungen möglichst separat von eventuell vorhandenen Starkstrom führenden Kabeln sein. Ein Grund für die Wahl dieses Kabeltyps ist, dass Typ A geschirmt ist. Somit ist es gewappnet gegen elektromagnetische Störstrahlungen. Darüber hinaus wird ein Potentialausgleich empfohlen.
Die RS485 geeigneten Steckverbinder die am Markt erhältlich sind, unterscheiden sich in ihrer Schutzart.
Aus der Praxis weiß man, dass Schwierigkeiten mit der Übertragungstechnik bei PROFIBUS® Systemen meist durch falsche Verkabelung und Installation hervorgerufen werden.

7.2.2.2 UART

UART (Universal Asynchronous Receiver Transmitter) ist eine elektronische Schaltung, welche sehr gängig für die Realisierung digitaler serieller Schnittstellen an PCs bzw. Mikrocontrollern ist. Es kann entweder in Form eines

eigenen Bauteils oder auch als Funktionsblock in einem „intelligenteren" Bauteil wie etwa einem Mikrocontroller vorkommen.

Die Übertragung funktioniert über einen seriellen digitalen Datenstrom, der einen fixen Rahmen bietet. Dieser Frame sieht wie folgt aus:

Startbit	D0	D1	D2	D3	D4	D5	D6	D7	(D8)	(Parity Bit)	Stopbit

Abb. 87 UART-Frame

Meist werden aber nur Rahmen im Umfang von acht oder neun Bits gesendet.

Das Besondere an UART ist, dass es asynchron arbeitet. In Folge dessen benötigt die Übertragungsleitung kein eigenes Taktsignal. Allerdings muss nach dem Senden eines Datenframes eine Busruhe von mindestens 33 Bit-Zeiten gegeben sein, damit der Empfänger den Sendeabschluss erkennen kann. Für die Kommunikation zwischen UART-Baugruppen müssen ein Sender und ein Empfänger der anderen Baugruppe am Stecker gegenüberstehen.

7.2.3 Protokolle

Für die Übertragung wird das FDL-Protokoll (Fieldbus Data Link) verwendet, welches als Basisprotokoll für alle anderen Profibus-Protokolle gilt. Oft wird es auch Buszugriffsprotokoll genannt. Es sorgt für eine einheitliche Datenübertragung bei PROFIBUS®. Dabei sind die Funktionen für alle Profile identisch.
Einer der Vorteile dieses Protokolls ist die Geschwindigkeit, da durch die hohe Hardwarenähe ein dementsprechender Telegrammdurchsatz möglich ist. Ein Nachteil ist, dass es nicht für die Übertragung von kleinen Datenmengen, also jenen unter 240 Byte, geeignet ist.

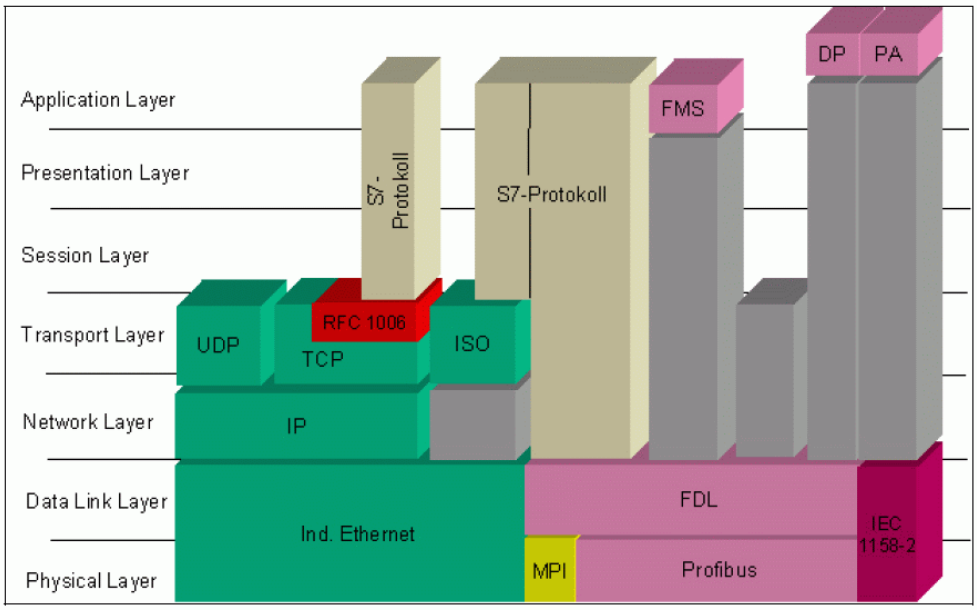


Abb. 88 Profibus im Schichtenmodell

Anhand dieser Grafik von Siemens ist zu erkennen, wie komplex der Zusammenhang der verschiedenen Protokolle ist. Der besonders relevante Teil für die Diplomarbeit ist der violett eingefärbte Teil der Grafik.

Für eine große Übertragungssicherheit bei einer FDL-Kommunikation sorgen die Telegrammformate des PROFIBUS® Layer 2. Tritt ein Fehler bei den Telegrammen auf, wird der Übertragungsvorgang automatisch wiederholt.

7.2.3.1 Osi-Referenzmodell

Schicht	OSI-Referenzmodell	PROFIBUS®
7	Anwendungsschicht	PROFIBUS®DP Protokoll DP-V0, PD-V1, DP-V2
6	Darstellungsschicht	Nicht verwendet
5	Sitzungsschicht	
4	Transportschicht	
3	Vermittlungsschicht	
2	Sicherungsschicht	Fieldbus Data Link (FDL) Master-Slave Verfahren Token-Passing Verfahren
1	Physikalische Schicht	Übertragungstechnik RS485, Optisch, MBP

Schicht 7

Wenn man Profibus® anhand des OSI-Referenzmodells betrachtet befinden sich auf der Anwendungsschicht drei DP-Protokolle. Sie wurden Schrittweise entwickelt und werden als „DP-V0", „DP-V1" und „DP-V2" bezeichnet. Ihre Definition sieht folgendermaßen aus:

- DP-V0
  DP-V0 bietet den zyklischen Austausch von Daten und Diagnosen. Vor allem in der allgemeinen Automatisierungstechnik sowie in der Maschinensteuerung sind Geräte, die diese zyklische Funktion unterstützen, zu finden.
  Schutzfunktionen
  Die Nachrichtenübertragung hat eine Hamming-Distanz von 4. Somit werden alle 1-Bit Fehler erkannt und korrigiert. Die 2-Bit Fehler werden bei der HD=4 nur erkannt und nicht korrigiert.
  Eine weite Schutzfunktion ist die sogenannte Ansprechüberwachung beim DP Slave. Diese erkennt von selbst ob der Master ausgefallen ist.
  Auch werden die Ausgänge der Slaves mittels Zugriffsschutz beschützt.

- DP-V1
  Durch DP-V1 ist der azyklische Austausch von Daten und Alarmbehandlungen gegeben. Besonders Geräte aus der Verfahrenstechnik unterstützen diese Erweiterung. Weiters wurde mit dieser Version der Online-Zugriff auf Busteilnehmer über Engineering-Tools ermöglich.
  Im Folgenden ist ein Beispiel anhand eines Read-Dienstes veranschaulicht.

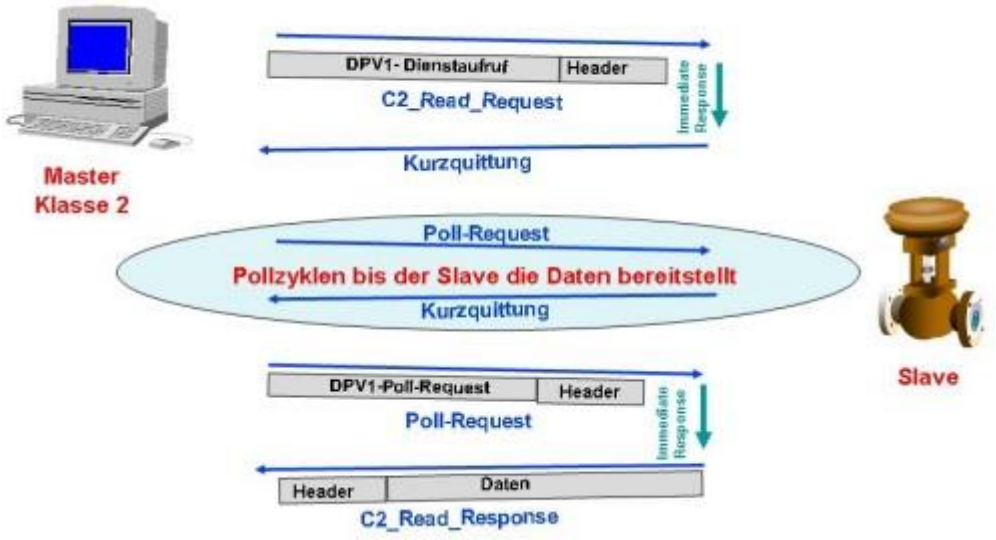


Abb. 89 DP V1 Read-Dienst

Weil der Slave für die Aufbereitung der benötigten Daten oft einige Millisekunden benötigt, wird meist nur der Empfang der Anfrage bestätigt, noch bevor Daten gesendet werden. Dadurch kann der Master seine zyklische Datenverarbeitung fortsetzen und wird nicht lange aufgehalten. Bei den weiteren Zyklen fragt der Master mittels Poll-Telegrammen den Slave immer wieder, ob die Daten nun endlich fertig sind. Ist dieser noch nicht fertig, sendet der Slave eine Kurzquittung. Falls die Daten bereit sind, überträgt er diese als Antwort.
Die Bedarfsdatenübertragung kann sowohl mit einem Klasse 1 Master als auch mit einem Klasse 2 Master zu dem Slave funktionieren. Wird einer der Klasse 2 verwendet, muss vor dem ersten Read/Write Dienstes noch eine C2 Verbindung durch den Initiate-Dienst erfolgen.

- DP-V2
  Bei DP-V2 wird der isochrone Austausch der Daten, der Slave-Querverkehr (Kommunikation zwischen Slaves) sowie die Taktsynchronisation ermöglicht. Dies ist vor allem in der Fertigungstechnik und der Robotersteuerung zu finden.

Schicht 2

Auf der Sicherungsschicht des OSI-Referenzmodells ist das, wie bereits weiter oben erwähnte FDL zu finden. Dies arbeitet mittels einem hybriden Zugriffsverfahren. Zusammengesetzt ist dies aus dem Token-Passing und dem Master-Slave-Verfahren.

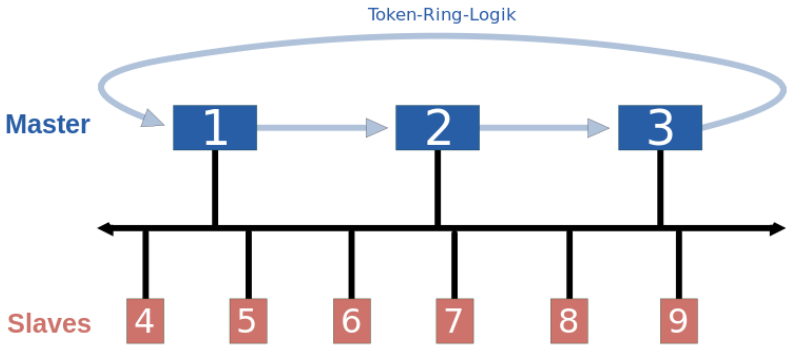


Abb. 90 Hybrid Zugriffsverfahren

Weiters können verschiedene Telegrammarten verwendet werden, welche durch verschiedene Startdelimiter unterschieden werden können.

Telegrammart	Telegrammaufbau	Codierung
Keine Daten	SD1 – DA – SA – FC – FCS – ED	SD1 = 0x10
Daten – variable Länge	SD2 – LE – LEr – SD2 – DA – SA – FC – PDU – FCS – ED	SD2 = 0x68
Daten – feste Länge	SD3 – DA – SA – FC – PDU – FCS – ED	SD3 = 0xA2
Token	SD4 – DA – SA	SD4 = 0xDC
Kurzquittung	SC	SC = 0xE5

Tabelle 16 Telegrammtypen

Legende
SD ... Startdelimiter
DA ... Destination Address
SA ... Source Address
FC ... Function Code
FCS ... Frame Check Sequence
DSAP ... Destination Service Access Point
SSAP ... Source Service Access Point
LE ... Datenlänge mitsamt DA, SA, FC, DSAP, SSAP
LEr ... Wiederholung der Datenlänge
ED ... End Delimiter

Der Telegrammtyp „Daten – feste Länge" kommt in der Praxis selten vor, da meist eine dynamische Länge wie man sie unter „Daten – variable Länge" hat bevorzugt wird.

Die FCS wird durch das Summieren der Bytes innerhalb der Länge berechnet. Dabei sichert man jedes Byte mit einer geraden Parität und sendet es asynchron durch Start- und Stop-Bits.

Alle Bytes des Telegramms müssen ohne einer Unterbrechung bzw. Pause zwischen Stop und dem nächsten Start-Bit übertragen werden. Der neue Beginn eines Telegramms wird durch den Master mittels einer SYN-Pause (Busruhe) von 33 Bit bekanntgegeben.

Weiters können die Ein- und Ausgänge durch Steuerkommandos synchronisiert werden. Eine andere Möglichkeit ist, dass der Sync- bzw. Freeze-Mode aktiv ist. Dadurch werden beim Sync-Mode die Ausgänge und beim Freeze-Mode die Eingänge synchronisiert.

Schicht 1

Auf der Schicht der Bitübertragung im OSI-Referenzmodell sind drei mögliche Verfahren definiert.

- RS485
  RS485 ist eine der drei Möglichkeiten die man auf Schicht 1 bei Profibus hat. Dieses Verfahren wird genauer unter dem Kapitel 7.2.2 Übertragungstechniken erklärt.
- Optisch
  Profibus® bietet auf der Schicht 1 weiters die Möglichkeit zur optischen Übertragung. Dies erfolgt mittels Lichtwellenleiter, wo dann Stern-, Bus- oder Ring-Topologien zum Einsatz kommen. Die Ring-Topologie kann sogar redundant verwendet werden. Das heißt, dass das Netzwerk bis zu einem gewissen Grad Ausfall sicherer gemacht werden kann, da notfalls die Ersatzleitung einspringt. Durch die optische Variante der Übertragung können sehr lange Distanzen realisiert werden, welche bis zu 15km weit reichen können.
- MBP
  MBP steht für Manchester Bus Powered. Dabei werden Daten und die Speisung der Feldgeräte über dasselbe Kabel abgewickelt. Durch Begrenzung der Leistung kann die Verwendung auch in

explosionsgefährdeten Umgebungen ermöglicht werden. Ist dies der Fall, spricht man von einem eigensicheren Bereich. In solchen Bereichen haben die Buskabel mit einer anderen Spezifikation als Kabeltyp A (RS485), einen blauen Mantel. Bei MBP kann die Bustopologie eine Länge von bis zu 1,9 km besitzen, und die Verzweigungen zu den Geräten 120m lang sein. Die Bitrate ist bei MBP auf 31,25 kbit/s festgelegt. Hauptsächlich verwendet man diese Übertragungstechnik bei der Prozessautomation mit PROFIBUS® PA.

7.2.4 Kommunikation bei PROFIBUS®

Die PROFIBUS® Geräte kommunizieren mittels PROFIBUS® DP, das für alle Anwendungen ein einheitliches Kommunikationsprotokoll zur Verfügung stellt. Es erlaubt sowohl zyklische, als auch azyklische Kommunikation und legt die Regeln fest.
Im Innersten des Ablaufs für die Kommunikation steht das Master-Slave-Verfahren. Wo ein aktiver Teilnehmer (Master) den angeschlossenen passiven Teilnehmern (Slaves) zyklisch zum Datenaustausch auffordert. Im Anfrage-Telegramm des Masters stehen die Ausgangsdaten, also zum Beispiel Sollwerte. Der Slave antwortet mit einem Antwort Telegramm, das die Eingangsdaten beinhaltet. Das sind zum Beispiel aktuelle Messwerte.

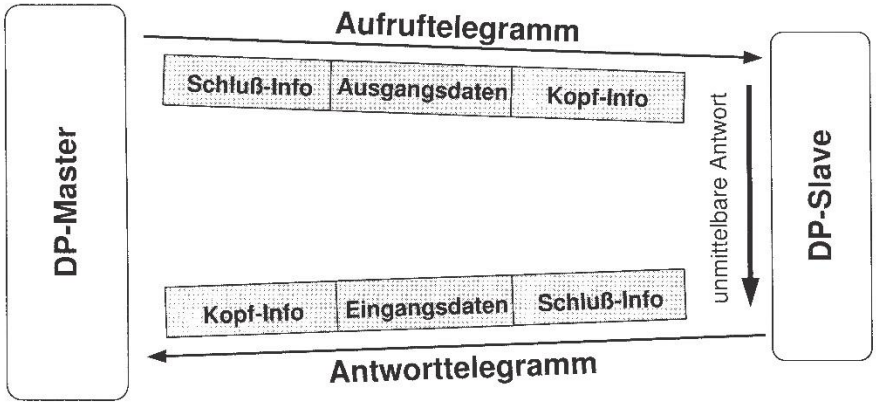


Abb. 91 Master-Slave Telegramm

Weiters gibt es noch die Bedarfsdatenübertragung, wo beispielsweise Geräteeinstellungen über PROFIBUS® gesendet werden. Dabei greift der Master azyklisch auf die Daten eines Slaves zu.

7.2.4.1 Zyklische Kommunikation

Nachdem die Projektierung (mittels Projektierungswerkzeug) in den Master Klasse 1 geladen ist, wird von diesem die zyklische Kommunikation zu den Slaves aufgebaut (MS0-Kanal). Bei dieser Phase überprüft der Slave die erhaltenen Daten vom Master, in zwei Stufen.
Zu Beginn werden die eingestellten Parameter, welche an den Slave gesendet wurden, überprüft. Dazu zählen Parameter wie Masteradresse, Watchdog-Zeit und ID. Die Ident-Nummer muss für jeden Gerätetyp eindeutig sein. Eine weitere zyklische Kommunikation ist nur möglich, wenn die erhaltene Ident-Nummer der vom Slave gespeicherten Nummer entspricht.
Danach werden die Informationen bezüglich der projektierten Module an den Slave übertragen und geprüft. Zur Kommunikation kommt es dann nur, wenn die physikalisch verfügbaren Module auch mit den eingestellten Modulen übereinstimmen.
Die Verifikation eines erfolgreichen Kommunikationsaufbaus erfolgt durch das Senden der angeforderten Diagnosedaten. Fehlerhafte Parameter bzw. Konfigurationsdateien werden durch passende Errors in der Profibus®-Standarddiagnose dem Master vom Slave angezeigt.

7.2.4.2 Azyklische Kommunikation

Im Zentrum der azyklischen Kommunikation steht der Master, welcher den „Wunsch" hat Geräteparameter der Slaves zu lesen oder zu schreiben. Durch diese Geräteparameter kann ein Gerät von einem zentralen Bedienungswerkzeug eingestellt und an spezielle Aufgaben angepasst werden.
In Bezug auf die azyklische Kommunikation gibt es zwei Kommunikationskanäle, welche die Bezeichnung MS1- und MS2-Kanal besitzen.
MS1-Kommunikation ist nur dann möglich, wenn zwischen Master und Slave ein zyklischer Austausch der Daten erfolgt. Da ein Slave zu einem Zeitpunkt nur mit einem Master zyklisch kommunizieren kann, folgt dass er auch nur eine MS1-Verbindung haben kann. Diese Verbindung wird bereits implizit mit der zyklischen Kommunikation, durch entsprechende Parameterdaten, aufgebaut. Überwacht wird die MS1-Verbindung durch die Watchdog-Zeit.
Bei der MS2-Verbindung ist es für den Slave nicht notwendig sich in einer zyklischen Kommunikation zu befinden. Weiters kann sie auch mit mehr als nur einem Master eingegangen werden. Der Aufbau erfolgt explizit durch den

Master. Dieser baucht in der Regel nur die betroffene Geräteadresse wissen und nicht, wie bei der zyklischen Kommunikation, die Gerätestammdaten. Die Verbindung verfügt außerdem über eine separate Zeitüberwachung. Die MS2-Verbindung wird geschlossen, wenn sie über einer festgelegten Zeit nicht verwendet wird.
Dienste für den azyklischen Datenverkehr zwischen DPM1 und den Slaves:

Read	lesen eines Datenblocks des Slaves (durch den Master)
Write	schreiben eines Datenblocks des Slaves (durch den Master)
Alarm	Der Slave sendet einen Alarm zum Master, welcher dann bestätigt wird. Solange die Bestätigung nicht erhalten wurde, kann der Slave keine weiteren Alarmmeldungen senden.
Alarm_Ack	Bestätigung der Alarmmeldung
Status	senden einer Statusmeldung an den Master (keine Bestätigung)

Tabelle 17 Dienste azyklischer Datenverkehr

7.3 Problematik

7.3.1 Keine Bibliotheken

Profibus® ist für die Kommunikation zwischen dem Roboter und dem Mikrocontroller notwendig, da der Roboter nur über Profibus® nach außen hin kommunizieren kann. Der Plan war, den Mikrocontroller als Master und den Roboter als Slave zu konfigurieren.
Zu diesem Zeitpunkt war noch nicht klar, welche Ausmaße dies annehmen würde, denn zu Beginn der Diplomarbeit gab es die Annahme, sich nur auf eine Profibus Bibliothek stützen zu müssen, um eine Basiskommunikation aufzubauen. Dem war leider nicht so, weil Profibus zwar mittlerweile ein offener Standard ist, da die Lizenzen der Firmen Siemens AG und Endress+Hauser AG schon abgelaufen sind, allerdings gibt es noch immer keinerlei offene Implementierungen.

7.4 mögliche Lösungsansätze

Im Folgenden sind verschiedene Lösungsansätze herausgearbeitet, welche unter Umständen zu einer funktionierenden Kommunikation zwischen dem Roboter und dem Mikrocontroller führen können.

7.4.1 PROFIBUS® on Raspberry Pi®

Das Projekt PROFIBUS® on Raspberry Pi® wurde von dem Schweizer Michael Büsch in Angriff genommen.

Dabei wird versucht einen Raspberry Pi® als DP-Master Klasse 1 für einen Siemens ET-200S Slave, zu verwenden. Sein Programm ist auch frei erhältlich und unter der GNU General Public License version 2 lizensiert.
Laut den Angaben auf seiner Website (www.bues.ch/cms/hacking/profibus.html) ist es möglich mit dem Slave zu kommunizieren. Das heißt, das Programm müsste von den Konfigurationen für den Siemens ET-200S Slave auf die Konfiguration des Motoman®-Roboters geändert werden. Weiters solle die Hardware-Schicht (OSI Schicht1) wie erwartet arbeiten. Die Bereiche der OSI-Schicht 2 und 7 sind in Arbeit, dabei solle der Großteil der Schicht 2 bereits implementiert sein und von Schicht 7 nur das allernotwendigste Minimum vorhanden sein. Im Zuge der Diplomarbeit wurde auch mit Herrn Büsch Kontakt aufgenommen und ihm unser Projekt erklärt. Seiner Einschätzung zufolge müsste dies mit kleinen Änderungen für unser Projekt reichen, aber es sei schwer einzuschätzen. Außerdem müsste die GSD-Datei des Roboters manuell interpretiert werden, da kein GSD Interpreter im Projekt vorhanden ist.

7.4.1.1 Voraussetzungen

Bevor das Projekt auf dem Raspberry Pi® laufen kann, wird das py-spidev Package benötigt. py-spidev ist ein Python Modul zur Verbindung von SPI Geräten mittels dem spidev Linux Kernel Treiber. Weiters muss "device_tree=" auf "disable" gesetzt werden. Es könnte auch auf "enable" bleiben, allerdings muss dann der AtMega88 Salve statisch in den Raspberry Pi® Device Tree eingetragen werden.

Zur Realisierung des DP-Masters ist folgende Hardware notwendig:

- Raspberry Pi®
- Atmel ATmega88
- MAX-3232 RS-232 Tranceiver
- RS-232 zu RS-485 Konverter
- Punktlochrasterplatine

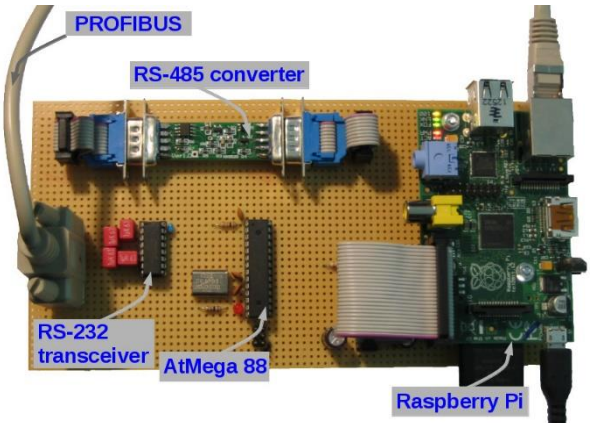


Abb. 92 Profibus on RaspberryPi Aufbau

Für die konkrete Konfiguration ist der Schaltplan notwendig. Dieser ist im Projektordner unter /schematics/cp-phy.pdf zu finden.
Der AtMega88 fungiert als Teil der OSI-Schicht1. Durch ihn werden die echtzeitrelevanten Funktionen für das Senden und Empfangen der Telegramme gehandhabt. Verändert werden die Telegrammdaten allerdings von ihm nicht. Dies geschieht auf dem Raspberry Pi®, welcher die Funktion des DP-Masters innehat. Der Quellcode für den AtMega88 befindet sich im Projektordner unter „firmware".

Die Kosten für die Hardwarekomponenten belaufen sich in etwa auf 60 Euro um einen Master für dieses Projekt zu realisieren.

7.4.2 Single-Chip-Profibusschnittstelle

Abb. 93 Anybus IC

Der Anybus-IC für Profibus®, welcher von HSM Industrial Networks GmbH entwickelt wurde, ist eine Single-Chip-Schnittstelle. Es ist eine vollständige Profibus®-DP Schnittstelle, welche nur 8cm² groß ist. Darauf vorhanden ist ein Profibus®-Protokollchip, sowie ein Flash- und RAM Speicher. Weiters sind auf dem Board analoge Schnittstellen vorhanden, wie zum Beispiel Optokoppler, DC/DC-Wandler und Bustreiber. Der vorhandene Mikroprozessor regelt automatisch den Profibus®-Busverkehr. Außerdem kann der Anybus-IC entweder alleine oder in Kombination mit einem externen Mikroprozessor betrieben werden. Wenn das Gerät einen eigenen Mikroprozessor hat, kann Anybus-IC den Geräteprozessor entlasten, als Profibus-UART arbeiten und die gesamte Profibus®-Protokollverarbeitung übernehmen.

Die maximale Profibus®-Datenbreite von 128 Byte Input und 128 Byte Output, sowie 926 kbit/s – 12 Mbit/s Baudraten werden unterstützt. Der Anybus-IC erkennt die Baudrate automatisch.

7.4.3 Siemens-Produkte

PROFIBUS® DP-Master Produkte von Siemens haben im Generellen den Vorteil, dass Siemens lange Zeit das Patent auf PROFIBUS® hatte und somit über ein gigantisches Knowhow verfügt. Auch werden von Siemens Kurse angeboten und die Produkte verfügen über einen technischen Support.
Im Folgenden ist nur das am meisten verwendete Produkt angeführt, da Siemens eine sehr umfangreiche Palette an Simatic-Geräten bietet und nicht jedes angeführt werden kann.

7.4.3.1 SIMATIC S7

Mittels der STEP 7 Programmiersoftware oder auch mit welcher von Fremdherstellern wird die Simatic programmiert. Abgesehen von der CPU kann das Automatisierungssystem auch noch mit diversen digitalen und analogen Peripheriebaugruppen sowie vorverarbeitenden, intelligenten Baugruppen bestückt werden. In der Industrie werden Simatic Steuerungen für sowohl kleine Kompaktgeräte, als auch für Hochleistungs-SPS verwendet. Speziell die Simatic S7 Steuerungen haben den Vorteil, sehr robust gegenüber elektromagnetischen Störungen und klimatischen Beanspruchungen zu sein. Auch sind sie leicht ausbaufähig.

Von den Simatic S7 gibt es verschiedene Typen, die sich aber im Großen und Ganzen sehr ähneln. Die meist verkaufte Siemens-SPS Variante ist die Simatic S7-300. Sie besitzt integrierte PROFIBUS®- und RS485-Schnittstellen, welche die Kommunikation zu anderen Geräten, wie z.B. dem Industrieroboter, ermöglicht. Auch ist sie mit einer Abmessung von nur 125x130 mm (HxT) relativ klein und kompakt.
Preislich liegt das Einsteigerpaket bei ca. 850€

7.4.4 Sitara™ AM335x Starter-Kit

In den meisten heutzutage verwendeten Lösungen wird eine Architektur wie in folgender Grafik verwendet.

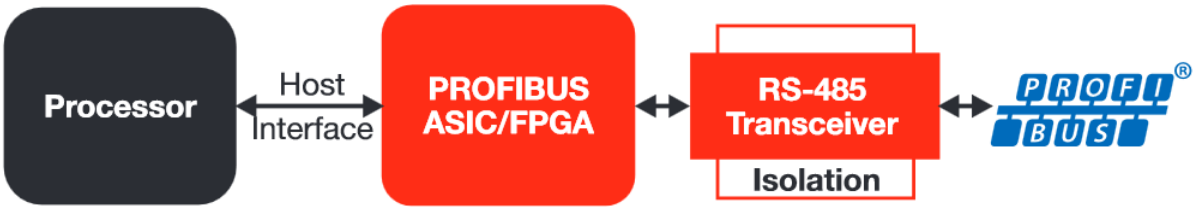


Abb. 94 Typische PROFIBUS Implementation

Bei solchen Lösungen, läuft am Mikroprozessor der PROFIBUS® den die Anwendungsschicht benötigt und die industrielle Applikation. Ein eigener ASIC oder FPGA, welcher sich dann um die Implementierung des PROFIBUS Protokolls kümmert und ein RS485 Transceiver stellt die Verbindung zur Physikalischen Schicht her.
Ein ASIC (application-specific integrated circuit) ist ein integrierter Schaltkreis, der für eine bestimmte Aufgabe hergestellt wurde. Der FPGA (Field Programmable Gate Array) ist ebenfalls ein integrierter Schaltkreis, dieser kann aber über eine logische Schaltung programmiert werden.

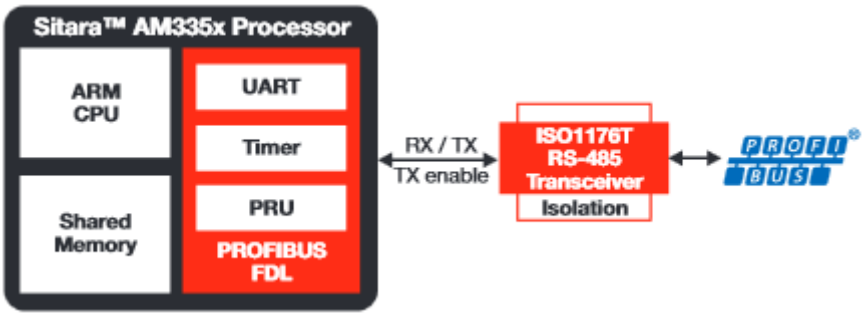


Abb. 95 Profibus®-Lösung von Texas Instruments

Texas Instruments hat nun die Funktionen des PROFIBUS® in den Sitara™ AM335x ARM Cortex™-A8 Prozessor integriert. Dadurch verbindet sich das Gerät direkt mit dem RS485 Tranceiver und benötigt nicht länger einen PROFIBUS® ASIC oder FPGA.
Die PRU (programmable real-time unit) implementiert eine Echtzeit-PROFIBUS-Nachrichtenübertragung sowie eine Rahmenüberprüfung und die Kommunikation mit dem ARM Prozessor. Eine der PRUs steuert den on-chip UART, was eine Übertragungsrate von 12Mbaud ermöglicht.

Das Starter Kit bietet die Möglichkeit, schnell mit dem Sitara™ ARM® Cortex™-A8 AM335x Processor zu starten. Es beschleunigt die Entwicklung von intelligenten Geräten sowie Industrie- und Netzwerkanwendungen.
Die Kosten für dieses Starter Kit liegen in etwa bei 200$ (USD).

8 Resümee

8.1 Zukunftsaussichten

Auf die Frage, wie es nun mit diesem Projekt weitergeht, ob der Roboter eines Tages tatsächlich auf Messeständen mit seinen Baukünsten begeistern wird, lässt sich keine eindeutige Antwort geben, denn es gibt einige Faktoren, die dagegen sprechen. Der wohl bedeutendste ist die Profibus®-Problematik.

Wie dem Kapitel **7.4** zu entnehmen gibt es durchaus Mittel, mit denen sich eine Kommunikation zum Roboter hin aufbauen lässt. Nur stellt sich hier die Frage, ob sich der Aufwand und die Kosten, die man in solche Lösungen investiert, auch tatsächlich auszahlen. Fertige Lösungen, wie die von Siemens oder Texas Instruments, bringen einen nicht irrelevanten Kostenfaktor mit sich. Die Implementierung des Open Source-Projektes von Michael Büsch ist zwar halbwegs kostengünstig, allerdings besteht hier keine Gewissheit, ob es damit auch tatsächlich funktionieren würde, da das Projekt selbst noch in den Kinderschuhen steckt und besonders für die Kommunikation auf höheren Schichten noch keine ausgereiften Lösungsansätze vorhanden sind.

Weiters müsste noch ein gewisser Arbeitsaufwand in das Projekt investiert werden. Zum einen müsste das Roboterprogramm für den Bauprozess geschrieben werden, was sich angesichts dessen, dass noch nicht sicher ist, ob die Kommunikation zum Roboter hin überhaupt funktioniert, als verlorene Liebesmüh entpuppen könnte.

Sollte das Projekt trotz allem weiterhin fortgesetzt werden, so müsste man wahrscheinlich die Kommunikation mit dem Roboter und die Verarbeitung der Sensor-Signale auf eine andere Komponente auslegen und sie nicht auf dem Mikrocontroller, der die grafische Benutzeroberfläche betreibt, laufen lassen. Der Grund: Das Aufbauen des Layouts und die Verwaltung der Benutzereingaben sind speicherintensiver als anfangs angenommen. Mit dem Programm der Diplomanden wird bereits etwa die Hälfte des verfügbaren Speichers belegt. Um trotzdem einen flüssigen Programmablauf zu gewähren, müssten also die anderen Aufgaben ausgelagert werden.

8.2 Gewonnene Erfahrungen

Im Laufe dieser Diplomarbeit haben wir einige sehr wertvolle Erfahrungen gemacht, die ich an dieser Stelle gerne noch einmal zusammenfasse.

Erstens lernten wir, wie wertvoll freie Software und offene Standards sind. In vielen Bereichen der Software-Entwicklung werden diese bereits als selbstverständlich angesehen. Zu nahezu jeder Problemstellung steht heutzutage schon eine freie Software-Lösung bereit. Das Produkt aus dem Ehrgeiz eines einzelnen oder mehrerer Hobby-Entwickler, die die Ergebnisse ihres Arbeitsaufwands gerne mit der Welt teilen. Der Umgang mit Profibus® hat uns gelehrt, diese Dinge mehr zu schätzen.

Zweitens erfuhren wir die Vorzüge vom Programmieren auf höheren Ebenen. In der Zeit, wo wir uns mit Mikrocontrollerprogrammierung in C herumschlagen mussten, lernten wir höhere Programmiersprachen wie Java oder C#, die hauptsächlich bei uns an der HTL unterrichtet werden, wieder zu lieben. In C und vor allem in der Mikrocontrollerprogrammierung tauchten uns bisher unbekannte Probleme auf. Probleme, um die sich beim Programmieren, wie wir es gewohnt sind, bereits der Compiler gekümmert hat.

Drittens haben wir die Erfahrung gemacht, dass ein Projekt nicht immer so abläuft, wie es geplant ist. Man kann von einer anfangs gegebenen Aufstellung nicht immer 100%ig sagen, dass diese auch mit gegebenen Mitteln lösbar ist, sofern man sich noch nicht intensiv damit beschäftigt hat. Obwohl die Probleme, die wir mit proprietären Systemen hatten, eine Zeit lang an unserer Moral nagten, mussten wir uns gen Zukunft wenden und mit abgeänderter Aufgabenstellung das Bestmögliche aus unserem Projekt machen.

Zusammenfassend lässt sich also sagen, dass sich das Projekt auch im Falle der Nicht-Weiterführung für uns gelohnt hat, denn wir konnten daraus wertvolle Erfahrungen gewinnen.

9 Impressum

Diplomarbeit

Schule

HTL Perg für Informatik
Machlandstraße 48
4320 Perg

Abb. 96 HTL-Perg Logo

Schuljahr

2014/2015

Klasse

5AHIF

Projektteam

Astleithner Christina
Stöger Irene

Auftraggeber

RK Automatisierungstechnik GmbH
Buchenstraße 4
4230 Pregarten

Abb. 97 Auftraggeber Logo

Ansprechpartner seitens des Auftraggebers

Wolfgang Schinnerl

Betreuungslehrer

DI Gerhard Zweimüller

10 Anhang

10.1 Glossar

5:6:5-RGB

5:6:5-RGB ist ein Format zur Speicherung von Farbinformationen im Speicher von Computern. Es wird auch mit dem Begriff 16-bit high color in Verbindung gebracht, da es die Farbe durch 16 Bits repräsentiert. Damit sind 65.536 verschiedene Farben möglich. 5:6:5 bedeutet, dass jeweils 5 Bits für die Rot- und Blaukanäle zuständig sind, während 6 Bits den Grünanteil repräsentieren.

ADC

ADC ist eine Abkürzung für Analog-to-Digital-Converter (Analog-Digital-Umsetzer). Dieses elektrische Gerät wandelt analoge Eingangssignale (z.B. vom Touch Panel) in einen digitalen Datenstrom um, der dann wiederum weiterverarbeitet werden kann.

ALU

Die ALU (**A**rithmetic **L**ogic **U**nit) ist für die Ausführung arithmetischer und bitweiser logischer Operationen zuständig. Sie bildet das Herzstück einer CPU. Sie nimmt Daten entgegen, an denen eine Operation durchgeführt werden soll (Operanden) und gibt das Ergebnis der durchgeführten Operationen aus.

ANSI C

ANSI C ist einer von mehreren möglichen Standards für die Programmiersprache C. Diese Standards existieren, um Portabilität und Kompatibilität zwischen C-Compilern zu gewähren.

Baud

Baud ist eine Einheit. Sie gibt die Schrittgeschwindigkeit (Symbolrate) in der Nachrichtentechnik an. So bedeutet eine Geschwindigkeit von 1 Baud, dass 1 Symbol pro Sekunde übertragen wird.

Bitmap

Eine Bitmap stellt ein digitales Bild dar. Sie besteht aus einer Matrix aus Bits, bei denen jedes Bit ein Pixel auf dem Display repräsentiert. Die Werte der einzelnen Bits stehen normalerweise für die Farbe des jeweiligen Bildpunktes.

Broadcast

Ein Broadcast (Rundfunk) ist eine Nachricht, die von einem Punkt im Netzwerk an alle Teilnehmer gesendet wird.

CPU

Eine **CPU** (**C**entral **P**rocessing **U**nit) ist ein elektronischer Schaltkreis in einem Computer, der die Operationen eines Programmes durchführt, indem er arithmetische, logische, Kontroll- sowie I/O-Operationen ausführt.

EEPROM

EEPROM steht für **E**lectrically **E**rasable **P**rogrammable **R**ead-**O**nly **M**emory und ist ein nichtflüchtiger Speicher. Das heißt, darin gespeicherte Daten bleiben auch nach Abschalten der anliegenden Versorgungsspannung erhalten. Die darin gespeicherten Informationen können elektrisch gelöscht werden. EEPROMs werden häufig bei Mikrocontrollern eingesetzt, um Informationen über die Laufzeit des Mikrocontroller-Programms hinaus zu speichern.

GLCD

GLCD steht für **G**raphic **L**iquid-**C**rystal **D**isplay und ist eine Art von LCD-Display, die zwei verschiedene Farben darstellen kann. Dabei wird eine Farbe durch die LCD-Hintergrundbeleuchtung bestimmt, während die andere eine Kontrastfarbe dazu darstellt.

ILI9341

ILI9341 ist ein SoC-Treiber für ein TFT-LCD-Display mit einer Auflösung von 240x320 Pixeln und 172.800 Bytes grafischen RAM für die Darstellungsdaten der 240x320 Pixeln.

JTAG

JTAG steht für **J**oint **T**est **A**ction **G**roup und bezeichnet einen Standard, der eine Methodik für das Testen und Debuggen integrierter Schaltungen (z.B. Mikrocontroller) beschreibt.

LCD

LCD steht für **L**iquid **C**ristal **D**isplay und bezeichnet einen Bildschirm, der mithilfe von Flüssigkeitskristallen Bilder anzeigen kann. Diese Flüssigkeitskristalle beeinflussen je nach der anliegenden Spannung die Polarisationsrichtung von Licht und können somit verschiedene Farben darstellen.

Mikrocontroller (µ-Controller)

Mikrocontroller sind programmierbare Systeme, die aus nur einem Chip bestehen (Ein-Chip-Systeme). Dabei befindet sich im Gegensatz zu herkömmlichen Prozessoren die gesamte Peripherie (Prozessor [MCU], Timer, Speicherregister, Input/Output-Schnittstellen,...) auf diesem einen Chip. Mikrocontroller alleine sind nutzlos. Man benötigt eine Schaltung, in die der Controller eingesetzt wird.
Mikrocontroller werden dort verwendet, wo sie einen begrenzten, speziellen Aufgabenbereich abdecken können.

Optokoppler

Ein Optokoppler ist ein elektronisches Bauteil, das für die Übertragung eines Signals zwischen zwei galvanisch getrennten Stromkreisen zuständig ist.

Oszillator

Ein Mikrocontroller benötigt eine Taktversorgung, um die internen Abläufe im Prozessor in einer zeitlich geordneten Reihenfolge ausführen zu können. Diese Taktversorgung wird von einem sogenannten Oszillator übernommen. Die Taktfrequenz ist ausschlaggebend für die Rechengeschwindigkeit des Mikrocontrollers. Der ATmega1284P besitzt eine Taktfrequenz von 20 MHz.

Peripheriegerät

Ein Peripheriegerät ist eine Komponente, die sich nicht innerhalb einer Zentraleinheit befindet. Sozusagen ist ein Peripheriegerät vergleichbar mit einem Zubehör. Es erbringt Leistungen die ihm befohlen werden, zum Beispiel das Setzen von Bausteinen auf eine Platte.

Polling

Polling ist eine Methode, die den Status eines Geräts oder eine Wertänderung durch zyklisches Abfragen ermittelt.

PROFIBUS®

Profibus® ist die Abkürzung für **Pro**cess **Fi**eld **Bus** und ist ein Standard in der Automatisierungstechnik. Er wird zur Kommunikation zwischen sowohl einem Master (erteilt Befehle) und einem Slave (erledigt Befehle) verwendet als auch zwischen zwei oder mehr Mastern. Profibus® dient damit also dem schnellen Austausch zwischen der Steuerung und den ausführenden Geräten.

RAM

RAM (**R**andom-**A**ccess **M**emory, Speicher mit wahlfreiem Zugriff) ist ein Datenspeicher, der oft bei Computern als Arbeitsspeicher verwendet wird. Dabei kann eine Speicherzelle direkt über ihre Speicheradresse angesprochen werden.

SoC

SoC bedeutet System-on-a-Chip (auch Ein-Chip-System) und bezeichnet die Integration von Funktionen eines Systems auf einem einzigen Chip, also einem integrierten Schaltkreis.

SPI

SPI (**S**erial **P**eripheral **I**nterface) ist ein Bussystem für einen synchronen, seriellen Datenbus. Es wird üblicherweise auf kurze Distanzen verwendet. Die Kommunikation erfolgt im Full-Duplex-Modus und basiert auf einer Master-Slave-Architektur., wobei es nur einen Master gibt.

SPS

SPS (**S**peicher**p**rogrammierbare **S**teuerung) ist ein Gerät, welches eine steuernde bzw. regulierende Aufgabe erfüllt. Es erteilt also in den meisten Fällen Befehle.

SRAM

SRAM bezeichnet eine statische Art von RAM. Die darin gespeicherten Informationen sind flüchtig, weshalb sie bei Abschaltung der Betriebsspannung verloren gehen. Solange diese jedoch anliegt, können die Dateninhalte im statischen RAM beliebig lange gespeichert werden.

TFT-LCD

TFT-LCD steht für **T**hin-**F**ilm-**T**ransistor **L**iquid-**C**rystal **D**isplay und ist eine Art von LCD-Display, das die TFT-Technologie nutzt, um die Bildqualität gegenüber gewöhnlichen LCD-Displays zu verbessern. TFT-Displays werden zum Beispiel bei Computerbildschirmen, Handys, Spielekonsolen und Navigationssystemen verwendet.

Try-and-Error

Try-and-Error (Versuch und Irrtum) ist eine Methode der Problemlösung, bei der so lange verschiedene Lösungsmöglichkeiten probiert werden, bis die gewünschte Lösung gefunden wird.

USART (UART)

USART (oder UART) steht für **U**niversal **S**erial **A**synchronous **R**eceiver/**T**ransmitter ist eine Hardware-Komponente, die Daten zwischen parallelen und seriellen Formaten übersetzen kann. UART wird üblicherweise in integrierten Schaltkreisen eingesetzt und wird für die serielle Kommunikation über den seriellen Port eines Computers oder Peripheriegeräts verwendet. Heutzutage sind UARTs üblicherweise auch in Mikrocontrollern inkludiert.

Hamming-Distanz

Als Hamming-Distanz wird die Anzahl der unterschiedlichen Stellen zwischen zwei Blöcken (Codewörter) mit fester Länge genannt. Sie wird zur Fehlererkennung und –korrektur verwendet. Das wird realisiert, indem Dateneinheiten, die über eine Übertragungsstecke empfangen wurden, mit gültigen Zeichen verglichen werden. Eine mögliche Korrektur erfolgt mit dem Wahrscheinlichkeitsprinzip. Ob tatsächlich eine Erkennung bzw. Korrektur stattfinden kann, wird durch die Hamming-Distanz bestimmt.

10.2 Literaturverzeichnis

Benutzerhandbücher

ATmega1284P Datasheet, http://www.atmel.com/images/doc2503.pdf

ATmega32 Datasheet, http://www.atmel.com/images/doc8059.pdf

EasyAVR™ v7 Benutzerhandbuch,
http://www.mikroe.com/downloads/get/1969/easyavr_v7_manual_v101.pdf

EasyTFT Benutzerhandbuch,
http://www.mikroe.com/downloads/get/1928/easytft_manual_v101.pdf

UNI-DS6 Benutzerhandbuch,
http://www.mikroe.com/downloads/get/1631/unids6_manual_v100.pdf

BIGdsPIC6 Benutzerhandbuch,
http://www.mikroe.com/downloads/get/1091/bigdspic6_manual_v100.pdf

Yaskawa Motoman® Benutzerhandbuch, zur Verfügung gestellt von RK Automatisierungstechnik GmbH

Profibushandbuch, http://profibus.felser.ch

Profibus

Wikipedia, http://en.wikipedia.org/wiki/Profibus,
http://upload.wikimedia.org/wikipedia/commons/0/0e/Automatisierungspyramide2.svg
http://de.wikipedia.org/wiki/EIA-485
http://de.wikipedia.org/wiki/Universal_Asynchronous_Receiver_Transmitter
http://de.wikipedia.org/wiki/Simatic

Academic.ru, http://de.academic.ru/dic.nsf/dewiki/1150547

x-technik IT & Medien GmbH,
http://cdn.x-technik.com/upload/images/81405.jpg

PROFIBUS & PROFINET International (PI),
http://www.profibus.com/technology/profibus/overview/

Siemens
https://cache.industry.siemens.com/dl/files/555/26098555/img_53881/v1/net_fdl_protokoll_01.gif
http://www.conrad.at/ce/de/product/198180/Siemens-SIMATIC-S7-300-Einsteigerbox-6ES7312-5BE03-4YB0-24-VDC
https://support.industry.siemens.com/cs/document/26098555?lc=de-WW

Michael Büsch, www.bues.ch/cms/hacking/profibus.html

HSM Industrial Network GmbH,
http://www.anybus.de/products/abic_profibus.shtml
http://www.feldbusse.de/Profibus/Profibus-DP.shtml

Profibushandbuch,
http://profibus.felser.ch
http://www.profibus.felser.ch/einfuehrung/profidp.pdf

Texas Instruments, http://www.ti.com.cn/cn/lit/wp/spry155b/spry155b.pdf,
http://www.ti.com/tool/tmdssk3358#2

HTW Dresden,
http://www2.htw-dresden.de/~huhle/micros/MC-Docs/profibus.pdf

Fritz Kübler GmbH,
https://www.kuebler.com/PDFs/Feldbus_Multiturn/specification_DP.pdf

https://manuel.mausz.at/

Google Books,
https://books.google.at/books?id=N8huAnJjJbwC&pg=PA559&dq=funktionen+dpm2&hl=de&sa=X&ei=pyUMVaTzIMGqywP1vICYDg&ved=0CB4Q6AEwAA#v=onepage&q=funktionen%20dpm2&f=false

PBMaster, Tran Duy Khanh,
http://www.pbmaster.org/de/index.pl?action=patents

DKE Deutsche Kommission Elektrotechnik
Elektronik Informationstechnik in DIN und VDE
https://www.dke.de/de/Service/Nachrichten/documents/typ3profibus.pdf

Sonstiges

Wikipedia, http://de.wikipedia.org, http://en.wikipedia.org, genutzt für **5.3**, **7** und **10.1**

Dokumentation der mikroC Pro vor AVR® Libraries

TechTerms.com, http://techterms.com/definition/bitmap

Newhaven Display International, Inc.
http://www.newhavendisplay.com/9F0CE163-CE06-4D94-A97A-056AB146B9F7/FinalDownload/DownloadId-1E5506BB8BC883C5B95BEC39B1A7F193/9F0CE163-CE06-4D94-A97A-056AB146B9F7/app_notes/ILI9341.pdf

10.3 Abbildungsverzeichnis

Die in dieser Arbeit enthaltenen Abbildungen stammen aus verschiedenen Quellen, die in folgender Tabelle angeführt sind.

Abbildung	Quelle
Abb. 0	halbtransparenter Legostein auf dem Deckblatt, erstellt von Christina Astleithner mit GIMP
Abb. 1	Astleithner Christina, Foto: Hartlauer
Abb. 2	Irene Stöger, Foto: Albin
Abb. 3	SmartArt, zusammengestellt von Irene Stöger, beinhaltet: • Abb. 1 • Abb. 2 • Abb. 96 • Abb. 97
Abb. 4	HTL-Perg
Abb. 5	RK Automatisierungstechnik GmbH
Abb. 6	zusammengestellt aus Abb. 97 und einem anderen Bild, das von Christina Astleithner mit dem LEGO® Digital Designer erstellt wurde
Abb. 7	Grafik zusammengestellt aus verschiedenen Word-Formen, Textfeldern und Bildern, beinhaltet: • Abb. 0 (Legostein auf Deckblatt) • eine Pixelgrafik des Industrieroboters (erstellt von Irene Stöger) • eine Fotografie eines Mikrocontrollers (entnommen vom Conrad-Online-Shop) • eine Fotografie des TFT-Displays mit Touchpanel (von Christina Astleithner)
Abb. 8 – 10	Foto von Irene Stöger
Abb. 11	entnommen aus dem Benutzerhandbuch des Yaskawa Motoman®
Abb. 12	Grafik zusammengestellt aus Word-Formen, Textfeldern und einer Fotografie des Entwicklungsboards, entnommen aus dessen Produktbeschreibung im MikroElektronika Online-Shop
Abb. 13	Grafik zusammengestellt aus Word-Formen und einer Fotografie von Christina Astleithner
Abb. 14	entnommen aus dem Benutzerhandbuch des UNI-DS6
Abb. 15	entnommen aus dem Benutzerhandbuch des BIGdsPIC6

Abb. 16	Foto von Christina Astleithner
Abb. 17 – 22	entnommen aus dem Datenblatt des ATmega32
Abb. 23	Foto von Christina Astleithner
Abb. 24	entnommen aus der Produktbeschreibung des EasyTFT™ im MikroElektronika Online-Ship
Abb. 25	Foto von Christina Astleithner
Abb. 26, 27	entnommen aus den jeweiligen Produktbeschreibungen im MikroElektronika Online-Ship
Abb. 28 - 33	Screenshot von Christina Astleithner
Abb. 34 – 36	Microsoft Office
Abb. 37	Screenshot von Christina Astleithner
Abb. 38	Google Drive
Abb. 39	GIMP
Abb. 40	Diagramm erstellt von Irene Stöger mit Microsoft Excel
Abb. 41	Screenshot von Christina Astleithner
Abb. 42 - 45	Auszüge aus dem Programmcode von Christina Astleithner
Abb. 46	Foto von Christina Astleithner
Abb. 47 - 49	Grafik, besteht aus Word-Formen und –Textfeldern sowie Fotografien bzw. Auszüge aus dem Programmcode von Christina Astleithner
Abb. 50, 51	Auszüge aus dem Programmcode von Christina Astleithner
Abb. 52, 53	Foto von Christina Astleithner
Abb. 54	Auszüge aus dem Programmcode von Christina Astleithner
Abb. 55, 56	Grafik erstellt mit dem LEGO® Digital Designer
Abb. 57	Foto von Christina Astleithner
Abb. 58 – 69	Auszüge aus dem Programmcode von Christina Astleithner
Abb. 70 - 73	Foto von Christina Astleithner
Abb. 74, 75	Auszüge aus dem Programmcode von Christina Astleithner
Abb. 76, 77	Foto von Christina Astleithner
Abb. 78 - 82	Auszüge aus dem Programmcode von Christina Astleithner
Abb. 83	cdn.x.-technik.com
Abb. 84	Grafik von PROFIBUS & PROFINET International
Abb. 85, 86	erstellt mit Word-Formen & -Textfeldern von Irene Stöger
Abb. 87	Word-Tabelle erstellt von Irene Stöger
Abb. 88	Grafik von Siemens
Abb. 89	Grafik von HSM Industrial Networks GmbH
Abb. 90	Grafik von Wikimedia
Abb. 91	Grafik von Hengstler GmbH
Abb. 92	Foto von Michael Büsch
Abb. 93	Foto von HSM Industrial Networks GmbH

Abb. 94	Grafik von Texas-Instruments
Abb. 95	Grafik von Texas-Instruments
Abb. 96	HTL-Perg
Abb. 97	RK Automatisierungstechnik GmbH

10.4 Tabellenverzeichnis

Die in dieser Arbeit enthaltenen Tabellen stammen ausschließlich von den beiden Diplomanden selbst.