



FACHHOCHSCHUL-BACHELORSTUDIENGANG

Automatisierungstechnik - SMS

**Modulares und skalierbares elektronisches System
für autonome Roboter**

ALS BACHELORARBEIT EINGEREICHT

zur Erlangung des akademischen Grades

Bachelor of Science in Engineering

von

Michael Zauner

02/2009

Betreuung der Bachelorarbeit durch

DI Walter Rokitansky

Ich erkläre ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen nicht benutzt, die den benutzten Quellen entnommenen Stellen als solche kenntlich gemacht habe und dass diese Arbeit mit der vom Begutachter beurteilten Arbeit übereinstimmt.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

.....
Michael Zauner

Aschach/Donau, 1. Februar 2009

KURZFASSUNG

Studenten und Mitarbeiter der Fachhochschule Wels nehmen seit dem Jahr 2005 mit großem Erfolg an nationalen und internationalen Roboterwettbewerben teil. Im Jahr 2006 wurde das Robo-Racing-Team gegründet, und es wurde sehr bald klar, dass eine einheitliche Plattform für die Steuerung der Roboter entwickelt werden muss. Dabei wurden zwei Kategorien unterschieden, eine Steuerung für kleine und einfache Roboter und eine Steuerung für große und komplexe Roboter.

Die erste Art der Steuerung sollte besonders kleine Abmessungen haben, jedoch äußerst flexibel sein. Diese Steuerung wurde mit dem Mini-Board realisiert.

Die zweite Art der Steuerung sollte ein hohes Maß an Flexibilität und an Skalierbarkeit aufweisen und sollte darüber hinaus die zentrale Steuerung von Aufgaben, wie dem Einlesen von Sensoren oder die Ansteuerung von Servos bzw. Motoren, entlasten. Um diese Forderungen zu realisieren, wurde das System als Multi-Prozessor-Einheit ausgeführt, dabei werden alle Teilaufgaben auf verschiedene Module verteilt. Diese Module wurden so klein wie möglich umgesetzt, um einen kompakten Aufbau von Robotern zu ermöglichen und sind durch eine SPI-Schnittstelle miteinander verbunden. Zu den einzelnen Modulen des modularen elektronischen Systems zählen:

- das Main-Board
- das Sensor/Servo-Board
- das Motor-Board
- der Seriell → Funk-Umsetzer und der Funk → USB-Umsetzer
- das LCD-Modul
- Anschluss-Boards für Versorgung und Programmierung

Die nachfolgende Arbeit beschäftigt sich mit der genauen Beschreibung des modularen elektronischen Systems und dessen Software.

INHALTSVERZEICHNIS

1.	MOTIVATION	1
2.	MODULARES EL. SYSTEM V2.....	5
2.1.	POWER SUPPLY DER MODULE	6
2.1.1.	Step Down Converter	6
2.1.2.	500 mA Power Supply	7
2.1.3.	4 A Power Supply	9
2.2.	PROGRAMMIER-INTERFACE.....	10
2.3.	SPI INTERFACE	11
3.	MAIN-BOARD	13
3.1.	HARDWARE	13
3.2.	SOFTWARE	17
3.2.1.	Treiber Main-Board.....	17
3.2.2.	Multitasking System.....	29
3.2.3.	SPI Schnittstelle	33
4.	MOTOR-BOARD	41
4.1.	HARDWARE	41
4.2.	SOFTWARE	45
4.2.1.	Treiber	45
4.2.2.	SPI Meldungen.....	52
5.	SENSOR/SERVO-BOARD	58
5.1.	HARDWARE.....	58
5.2.	SOFTWARE	62
5.2.1.	Treiber Sensor-Board	62
5.2.2.	SPI Meldungen Sensor-Board.....	75
5.2.3.	Treiber Servo-Board.....	78
5.2.4.	SPI-Meldungen Servo-Board	80
6.	LCD-INTERFACE.....	82
6.1.	HARDWARE	82
6.1.1.	LCD Modul	82
6.1.2.	LCD.....	85

6.1.3.	Tastatur.....	86
6.2.	SOFTWARE	87
6.2.1.	Treiber	87
6.2.2.	SPI Meldungen.....	92
7.	FUNK ZU USB UMSETZER.....	93
7.1.	FUNK-MODUL	93
7.2.	USB ZU UART KONVERTER.....	95
7.3.	INTERFACE ZUM MODULAREN SYSTEM.....	96
7.4.	PC INTERFACE	99
8.	PROGRAMMIER-BOARD.....	101
9.	VCC-BOARD	104
10.	WEITERFÜHRENDE ARBEITEN.....	106
10.1.	MOTOR-BOARD	106
10.2.	SPI HUB.....	107
10.3.	POSITIONSERKENNUNG.....	107
10.4.	EINFACHE BILDVERARBEITUNG.....	107
10.5.	LCD-MODUL.....	108
11.	ZUSAMMENFASSUNG.....	109
12.	LITERATUR.....	110
12.1.	BÜCHERLISTE.....	110
12.2.	INTERNET	110
13.	ANHANG.....	112
13.1.	PROGRAMMTEILE MOTOR-BOARD.....	112
13.2.	PROGRAMMTEILE SENSOR-BOARD.....	117
13.3.	PROGRAMMTEILE LCD-MODUL.....	123

ABBILDUNGSVERZEICHNIS

Abbildung 1.1	Roboternetz RNFRA-Board für den Roboter „Hotspoiler“	1
Abbildung 1.2	Roboter „Hotspoiler“ für die RobotChallenge 2006.....	2
Abbildung 1.3	Modulares System der Version 1	3
Abbildung 2.1	Übersicht des modularen Systems der 2. Generation.....	5
Abbildung 2.2	Abwärtswandler oder Step-Down-Converter.....	6
Abbildung 2.3	Online Berechnungstool WEBENCH®	7
Abbildung 2.4	Typische Anwendung des LM2594	7
Abbildung 2.5	Typische Anwendung des LM2677	9
Abbildung 2.6	Bestückte Leiterplatte des Programmier Boards.....	10
Abbildung 2.7	Funktionsweise der SPI-Schnittstelle.....	12
Abbildung 2.8	Blockschaltbild des SPI-Buses.....	12
Abbildung 3.1	Bestückte Leiterplatte des Mainboards	13
Abbildung 3.2	Flussdiagramm Hauptprogramm Main-Board	18
Abbildung 3.3	Flussdiagramm „UART abarbeiten“	19
Abbildung 3.4	Meldung „vorwärts fahren“	20
Abbildung 3.5	Meldung „rückwärts fahren“	21
Abbildung 3.6	Meldung „Drehung links“	22
Abbildung 3.7	Meldung „Drehung rechts“	23
Abbildung 3.8	Meldung „einzelnen Motor ansteuern“	24
Abbildung 3.9	Meldung „Servo ansteuern“	25
Abbildung 3.10	Schematische Darstellung der Wegberechnung.....	26
Abbildung 3.11	State-Diagramm des Motoransteuerung.....	27
Abbildung 3.12	Zeitverhalten eines kooperativen Multitasking Systems	29
Abbildung 3.13	Zeitverhalten eines präemptiven Multitasking Systems	30
Abbildung 4.1	Bestückte Leiterplatte des DC-Motorboards.....	41
Abbildung 4.2	Blockdiagramm des BTS7750GP	44
Abbildung 4.3	Flussdiagramm Hauptprogramm Motor-Board.....	46
Abbildung 4.4	Blockschaltbild Drehzahlregler.....	47
Abbildung 4.5	Blockschaltbild Positionsregler.....	48
Abbildung 4.6	v-t bzw. s-t Diagramm.....	49

Abbildung 4.7 Schematische Darstellung der Positionsberechnung.....	50
Abbildung 4.8 Meldung MB_INIT_MSG	52
Abbildung 4.9 Meldung MB_STOP_MSG.....	52
Abbildung 4.10 Meldung MB_EMERGENCY_STOP_MSG.....	53
Abbildung 4.11 Meldung MB_FORWARD_MSG.....	53
Abbildung 4.12 Meldung MB_BACKWARD_MSG	53
Abbildung 4.13 Meldung MB_TURN_RIGHT_MSG	54
Abbildung 4.14 Meldung MB_TURN_LEFT_MSG	54
Abbildung 4.15 Meldung MB_SET_MOTORS_INDIVIDUAL_MSG.....	54
Abbildung 4.16 Meldung MB_DRIVE_MOTORS_ONE_MSG	55
Abbildung 4.17 Meldung MB_SET_MOTORS_INDIVIDUAL_ONE_MSG.....	55
Abbildung 4.18 Meldung MB_GET_POSITON_MSG.....	56
Abbildung 4.19 Meldung MB_SET_POSITION_MSG	56
Abbildung 4.20 Meldung MB_GET_STATUS_MSG.....	57
Abbildung 4.21 Meldung MB_SET_STEPPER_FORWARD_MSG.....	57
Abbildung 4.22 Meldung MB_SET_STEPPER_BACKWARD_MSG	57
Abbildung 5.1 Bestückte Leiterplatte des Sensor/Servoboards	58
Abbildung 5.2 Flussdiagramm Hauptprogramm.....	64
Abbildung 5.3 Flussdiagramm „Sensor einlesen“	65
Abbildung 5.4 Analoge Ausgangsspannung vs. Distanz des GP2D12 Sensors.....	66
Abbildung 5.5 Analoge Ausgangsspannung vs. Distanz des GP2D120 Sensors.....	67
Abbildung 5.6 Analoge Ausgangsspannung vs. Distanz des GP2Y0A02F Sensors	67
Abbildung 5.7 I ² C Bus-Konventionen	68
Abbildung 5.8 Schreibezyklus auf den SRF08/10 Sensor	69
Abbildung 5.9 Vom SRF08/10 Sensor lesen	70
Abbildung 5.10 Abstrahlcharakteristik des Ultraschallsensors SRF08/10	70
Abbildung 5.11 Normierte Kennlinie des digitalen Filters (Filterkoeffizient 3)	71
Abbildung 5.12 Störverhalten des Filters 1. Ordnung	72
Abbildung 5.13 Gegenüberstellung adaptives Filter und Filter 1. Ordnung.....	73
Abbildung 5.14 Meldung SB_INIT_MSG.....	75
Abbildung 5.15 Meldung SB_READ_SENSOR_MSG.....	76
Abbildung 5.16 Meldung SB_SET_PIN_MSG	77
Abbildung 5.17 Meldung SB_SET_IO_MSG	77
Abbildung 5.18 Timing bei der Ansteuerung der einzelnen Servos	79

Abbildung 5.19	Meldung SV_INIT_MSG.....	80
Abbildung 5.20	Meldung SV_SET_SERVO_MSG	81
Abbildung 6.1	Bestückte Leiterplatte des LCD-Moduls.....	82
Abbildung 6.2	LCD-Display 204A der Fa. Displaytech Ltd.	85
Abbildung 6.3	Blockdiagramm des LCD-Displays	85
Abbildung 6.4	Flussdiagramm Hauptprogramm.....	88
Abbildung 6.5	Meldung LCD_SET_COLOUR_MSG	92
Abbildung 7.1	ER400TRS Funk Modul	93
Abbildung 7.2	Datenübertragung des Funkmoduls.....	94
Abbildung 7.3	Vereinfachtes Block Diagramm des FT232BL Chips	95
Abbildung 7.4	Bestückte Leiterplatte des Universal Interfaces.....	96
Abbildung 7.5	Bestückte Leiterplatte des USB/Funk Konverters	99
Abbildung 8.1	Bestückte Leiterplatte des Programmier Boards (PC-Interface).....	101
Abbildung 9.1	Bestückte Leiterplatte des VCC-Verteiler Boards	104
Abbildung 9.2	Prinzipschaltbild des VCC-Verteiler Boards	105

TABELLENVERZEICHNIS

Tabelle 2.1 Steckerbelegung von K2 (Prog.-IF)	11
Tabelle 3.1 Steckerbelegung von K1 (Funk- u. RS232-Anbindung)	14
Tabelle 3.2 Auswahlmöglichkeiten der verschiedenen Schnittstellen	14
Tabelle 3.3 Steckerbelegung von K2 (Akku Kontrolle)	14
Tabelle 3.4 Steckerbelegung von K3 (Prog.-IF)	15
Tabelle 3.5 Steckerbelegung von K4 bis K7 (I/O 1...4 – PORTA).....	15
Tabelle 3.6 Steckerbelegung von K8 (I/O 5 – PORTB)	15
Tabelle 3.7 Steckerbelegung von K9 (Power-Supply).....	15
Tabelle 3.8 Steckerbelegung für K10 bis K23 (SPI-IF).....	16
Tabelle 3.9 Steckerbelegung von K24 (Massestift)	16
Tabelle 3.10 Softwaremodule des Main-Boards	17
Tabelle 3.11 Beschreibung der Meldung „vorwärts fahren“	20
Tabelle 3.12 Beschreibung der Meldung „rückwärts fahren“	21
Tabelle 3.13 Beschreibung der Meldung „Drehung links“	22
Tabelle 3.14 Beschreibung der Meldung „Drehung rechts“	23
Tabelle 3.15 Beschreibung der Meldung „Drehung links“	24
Tabelle 3.16 Beschreibung der Meldung „Servo ansteuern“	25
Tabelle 3.17 Variablen der Datenstruktur „Multitasking“	32
Tabelle 3.18 Befehl zum Servo-Board initialisieren	34
Tabelle 3.19 Befehl zum Sensor-Board initialisieren	34
Tabelle 3.20 Befehl zum Motor-Board initialisieren	35
Tabelle 3.21 Befehl zum Sensor auslesen	35
Tabelle 3.22 Befehl zum Setzen/Rücksetzen eines Ausgangs	35
Tabelle 3.23 Befehl zum Umschalten zwischen Ein- bzw. Ausgang.....	36
Tabelle 3.24 Befehl zum Setzen eines Winkels eines ausgewählten Servos	36
Tabelle 3.25 Befehl zum Ausgeben einer Drehbewegung	36
Tabelle 3.26 Befehl zum Vor- bzw. Rückwärtsfahren.....	37
Tabelle 3.27 Befehl zum Setzen von individuellen Geschwindigkeiten und Richtungen	37
Tabelle 3.28 Befehl zum Setzen einer individuellen Geschwindigkeiten und Richtungen	37
Tabelle 3.29 Befehl zum Setzen einer individuellen Geschwindigkeiten und Richtungen	38

Tabelle 3.30	Befehl zum Auslesen der relativen Bewegung und des Bewegungsstatus'	38
Tabelle 3.31	Befehl zum Auslesen der Orientierung und der Position.....	38
Tabelle 3.32	Befehl zum Setzen der Orientierung und der Position.....	39
Tabelle 3.33	Befehl zum Stoppen der Motoren	39
Tabelle 3.34	Befehl zum kontrollierten gegen Null Fahren der Motoren.....	39
Tabelle 3.35	Befehl zum Vor- bzw. Rückwärtsfahren eines Schrittmotors.....	39
Tabelle 3.36	Befehl zum Anfordern der Strategie und ev. Spielfarbe	40
Tabelle 4.1	Steckerbelegung von K1 (Prog.-IF)	42
Tabelle 4.2	Steckerbelegung von K2 (Massestift)	42
Tabelle 4.3	Steckerbelegung für K3 (SPI-IF)	42
Tabelle 4.4	Steckerbelegung von K4 (Power-Supply).....	42
Tabelle 4.5	Steckerbelegung für K5 und K6 (Motor-IF)	43
Tabelle 4.6	Steckerbelegung für K7 (I/O-IF).....	43
Tabelle 4.7	Steckerbelegung von KL1 und KL2 (Motoranschluss)	43
Tabelle 4.8	Steckerbelegung von KL3 (geschalteter Ausgang).....	43
Tabelle 4.9	Softwaremodule des Motor-Boards	45
Tabelle 4.10	Unterstützte Motortypen	52
Tabelle 5.1	Steckerbelegung von K1 (Power-Supply).....	59
Tabelle 5.2	Steckerbelegung von K2 (Prog.-IF)	59
Tabelle 5.3	Steckerbelegung von K3 (Massestift)	59
Tabelle 5.4	Steckerbelegung für K22 (SPI-IF)	59
Tabelle 5.5	Steckerbelegung von K4 bis K11 (S/S 1...8 – PORTA)	60
Tabelle 5.6	Steckerbelegung von K12 bis K19 (S/S 9...16 – PORTC).....	60
Tabelle 5.7	Steckerbelegung für K20 und K21 (LinenSensor1/2).....	61
Tabelle 5.8	Softwaremodule des Sensor-Boards.....	62
Tabelle 5.9	k_i -Werte der jeweiligen Sensoren.....	66
Tabelle 5.10	Messbereiche der Sensoren	67
Tabelle 5.11	Schreibe-Register des SRF08/10.....	69
Tabelle 5.12	Unterstützte Sensortypen	75
Tabelle 5.13	Softwaremodule des Servo-Boards	78
Tabelle 5.14	Unterstützte Servotypen	80
Tabelle 6.1	Steckerbelegung von K1 (Power-Supply).....	83
Tabelle 6.2	Steckerbelegung von K2 (LCD-IF).....	83
Tabelle 6.3	Steckerbelegung von K3 (Prog.-IF)	84

Tabelle 6.4 Steckerbelegung von K4 (Tastatur-IF).....	84
Tabelle 6.5 Steckerbelegung für K5 (SPI-IF)	84
Tabelle 6.6 Steckerbelegung von K6 (Massestift)	85
Tabelle 6.7 Softwaremodule des LCD-Interfaces	87
Tabelle 6.8 Befehl zum Initialisieren des LCD-Displays	89
Tabelle 6.9 Befehl zum Löschen des LCD-Displays	89
Tabelle 6.10 Befehl zum Setzen des Cursors	90
Tabelle 6.11 Befehl zum Schreiben eines Zeichens.....	90
Tabelle 6.12 Befehl zum Schreiben einer Zeichenkette aus dem RAM	90
Tabelle 6.13 Befehl zum Schreiben einer Zeichenkette aus dem Flash.....	90
Tabelle 6.14 Variablen der Datenstruktur „sKey“	91
Tabelle 7.1 Ab Werk Einstellungen des ER400TRS Moduls	93
Tabelle 7.2 Auswahlmöglichkeiten der verschiedenen Schnittstellen.....	96
Tabelle 7.3 Steckerbelegung von K1 (RS232-IF).....	97
Tabelle 7.4 Steckerbelegung von K2 (Roboter IF)	97
Tabelle 7.5 Steckerbelegung von K3 (TTL IF).....	98
Tabelle 7.6 Steckerbelegung von KL1 (Antennen Anschluss)	98
Tabelle 7.7 Steckerbelegung von K1 (USB-Port).....	99
Tabelle 7.8 Steckerbelegung von KL1 (Antennen Anschluss)	100
Tabelle 8.1 Steckerbelegung von K1 (Prog.-IF des Masters zum PC)	101
Tabelle 8.2 Steckerbelegung von K2 (Prog.-IF zum Master)	102
Tabelle 8.3 Steckerbelegung von K3, K5, K7, K9, K11, K13 und K15 (Prog.-IF zum PC)	102
Tabelle 8.4 Steckerbelegung von K4, K6, K8, K10, K12, K14 und K16 (Slaves Prog.-IF)	103
Tabelle 9.1 Steckerbelegung von K1 bis K16 (Power-Supply)	104
Tabelle 9.2 Steckerbelegung von KL1 (Hauptschalter)	105
Tabelle 9.3 Steckerbelegung von KL2 (Akku Anschluss).....	105

LISTINGS

Listing 3.1 Programm eines Muster Tasks.....	33
Listing 3.2 Initialisierungsroutine des Muster Task.....	33
Listing 13.1 Drehzahlregler des 1. Motors.....	113
Listing 13.2 Positionsregler des 1. Motors.....	114
Listing 13.3 Unterprogramm zur Ansteuerung eines Schrittmotors	116
Listing 13.4 Main-Loop des Sensorboards	117
Listing 13.5 Unterprogramm zum Abarbeiten der Sensoren	119
Listing 13.6 Unterprogramm „SRF08/10 initialisieren“	119
Listing 13.7 Unterprogramm „Messung starten“	120
Listing 13.8 Unterprogramm „SRF08/10 auslesen“	120
Listing 13.9 Unterprogramm „digitales Filter 1. Ordnung“	121
Listing 13.10 Unterprogramm „adaptives Filter“	122
Listing 13.11 Unterprogramm zur Behandlung von Tastern	124

1. MOTIVATION

Im Wintersemester 05/06 baute eine Gruppe von Studenten in einem fachübergreifenden Projekt (FUP) ein Roboter für den Bewerb RobotChallenge. Auf der Homepage des Veranstalters wird die RobotChallenge wie folgt beschrieben: „Die RobotChallenge ist der größten Robotik-Wettbewerb in Österreich und einer der größten der Welt.“¹ Die Studenten setzten sich zum Ziel bei den Bewerb

- Parallelslalom
- Slalom Enhanced
- Hindernislauf

teilzunehmen. Da bis zum Bewerb nur 5 Monate Zeit waren, wurde nach fertigen Lösungen für die Steuerelektronik gesucht. Diese Elektronik sollte sowohl die Steuereinheit als auch die Leistungselektronik für die Motoransteuerung beinhalten. Nach eingehender Suche fiel die Wahl auf das RNBFRA-Board der Fa. RoboterNetz. Wie aus Abbildung 1.1 ersichtlich ist, bietet dieses Board eine Vielzahl an Möglichkeiten für den Einsatz als zentrale Steuereinheit eines Roboters.

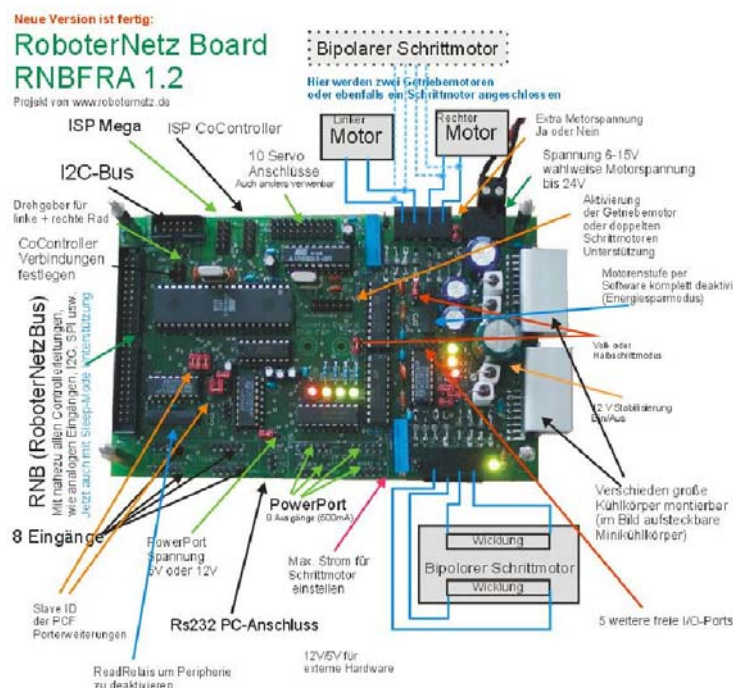


Abbildung 1.1 Roboternetz RNFRA-Board² für den Roboter „Hotspoiler“

¹ [RobotChallenge, 2008]

² [robotikhardware, 2008]

In Abbildung 1.2 ist der fertig aufgebaute Roboter „Hotspoiler“³ der Studenten dargestellt. Die Studentengruppe erreichte bei der RobotChallenge mit diesem Roboter im Parallelslalom den 3. Platz und bei den Bewerben Slalom Enhanced und Hindernislauf den 1. Platz.



Abbildung 1.2 Roboter „Hotspoiler“ für die RobotChallenge 2006

Nach diesen Erfolgen wurde im Herbst 2006 das Robo-Racing-Team der Fachhochschule Wels gegründet. Das klare Ziel dieses Projekts ist die erfolgreiche Teilnahme an nationalen sowie internationalen Roboterbewerben.

Im Wintersemester 06/07 entwickelten Studenten im Rahmen eines FUPs wieder einen Roboter für die RobotChallenge. Die Studenten setzten sich zum Ziel bei den Bewerben

- Minisumo
- Parallelslalom
- Slalom Enhanced

teilzunehmen. Um diese Ziel zu erreichen musste eine eigene zentrale Steuereinheit entwickelt werden, da das RNBFR-Board, mit seine Abmessungen von 16 cm x 10 cm, für den Minisumo Roboter zu groß war, denn die maximalen Abmessungen eines Minisumo Roboters darf maximal 10 cm x 10 cm betragen. Daher wurde das Miniboard entwickelt. Dieses Board besitzt einen ähnlichen Leistungsumfang wie das RNBFR-Board, misst allerdings nur 4,5 cm x 4,5 cm. Das konnte dadurch erreicht werden, dass der Großteil der Bauteile in SMD ausgeführt wurden und die Leiterplatte beseitig bestückt ist.

³ [FUP RobotChallenge, 2006]

Ebenso begann ein Berufspraktikum⁴, das sich zum Ziel setzte bei den Wettbewerben EUROBOT und RoboGames teilzunehmen.

Auf der Homepage der Austrobot wird die EUROBOT wie folgt beschrieben:

„Der Event EUROBOT ist aus dem französischen Nationalen Wettbewerb entstanden. An EUROBOT können Teams aus allen Ländern der Welt teilnehmen, obwohl der Schwerpunkt und der Austragungsort des Wettbewerbs in Europa liegt. Es nehmen pro Land bis zu drei Teams teil, in den vergangenen Jahren ist die Anzahl der teilnehmenden Mannschaften auf circa 50 gewachsen. EUROBOT findet Ende Mai statt.“⁵

Der amerikanische Veranstalter beschreibt seinen Bewerb auf seiner Homepage als:

„RoboGames is the world's largest open robot competition (even the Guinness Book of World Records says so!) with over 70 different events“⁶

Um die verschiedenen Roboter für diese Wettbewerbe zu bauen, ist es von Vorteil, wenn die Elektronik modular aufgebaut ist. D.h. die grundlegenden Einheiten wie das Einlesen von Sensoren oder das Ansteuern von Motoren soll auf separaten Boards abgearbeitet werden. Dadurch wird die zentrale Steuereinheit entlastet. Ziel des Berufspraktikums war es diese erste Version des modularen Systems zu entwickeln. In Abbildung 1.3 ist das modulare System der ersten Generation dargestellt.

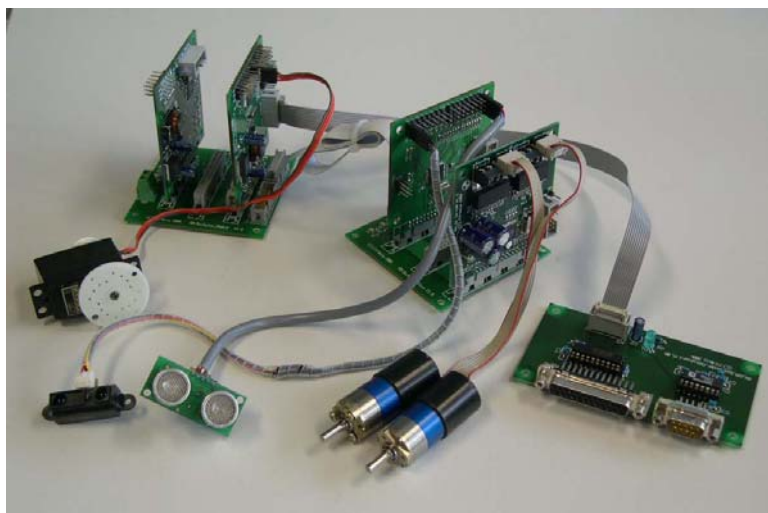


Abbildung 1.3 Modulares System der Version 1

⁴ [Berufspraktikum, 2006]

⁵ [austrobot, 2008]

⁶ [robogames, 2008]

In der Praxis wies das modulare System der ersten Generation einige Schwachstellen auf. Zu diesen Schwachstellen zählten

- Bei den Stiftleisten auf den Boards konnte es zu Wackelkontakten mit dem Stecker kommen
- Es gab nur 8 Steckplätze für Slave-Boards
- Die Backplanes mit den eingesteckten Boards benötigen sehr viel Platz (ca. 8 cm x 8 cm x 10 cm)

um nur einige zu nennen.

2. MODULARES EL. SYSTEM V2

In Abbildung 2.1 ist eine Übersicht des modularen Systems der zweiten Generation dargestellt. Das System besteht aus

- Main-Board
- Sensor-Board
- Servo-Board
- Motor-Board
- LCD-Board

Diese Boards werden in den nächsten Kapiteln eingehend erklärt.

Auf jedem dieser Boards befindet sich der Microcontroller ATmega644 der Fa. Atmel. Das Main-Board verteilt die Aufgaben an die Slaves, dadurch wird eine echte Parallelverarbeitung der Aufgaben erreicht und das Main-Board entlastet. Die Boards sind mittels SPI-Bus an den Master angebunden. Diese Schnittstelle wurde wegen ihrer hohen Übertragungsgeschwindigkeit gewählt.

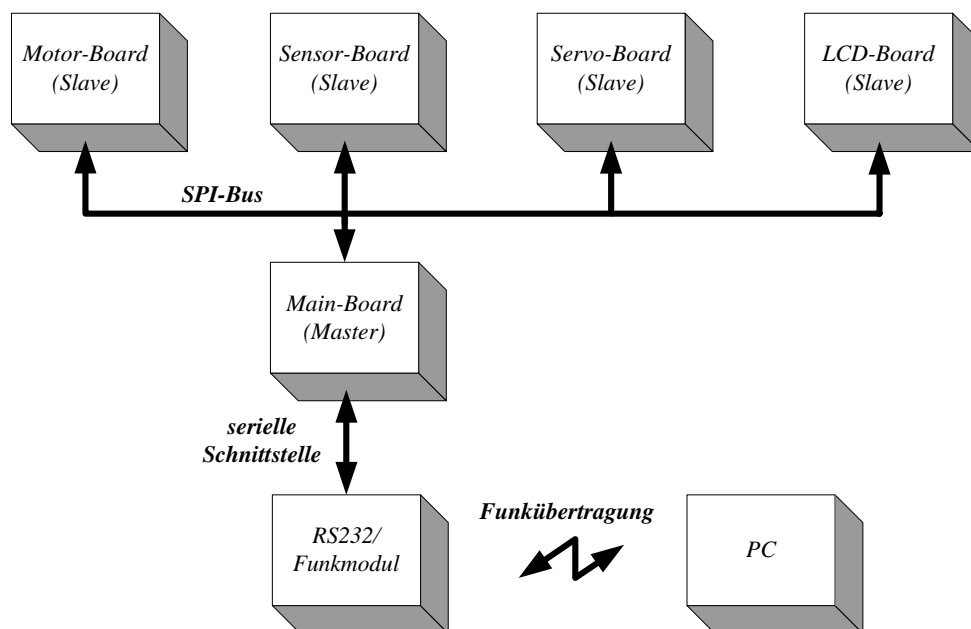


Abbildung 2.1 Übersicht des modularen Systems der 2. Generation

Da es bei dem modularen System der ersten Generation Probleme mit den Steckern gab, werden beim neuen System Stecker der Fa. Micromatch verwendet. Diese Stecker zeichnen sich durch einen sehr hohen Halt und einer sehr geringen Bauhöhe aus.

In der Folge werden die allgemeinen Elemente des modularen Systems der zweiten Generation erörtert.

2.1. POWER SUPPLY DER MODULE

2.1.1. Step Down Converter

Der Abwärtswandler, oder auch Step-Down-Converter, erzeugt aus einer Gleichspannung eine kleinere Gleichspannung. In Abbildung 2.2 ist das prinzipielle Schaltbild dargestellt.

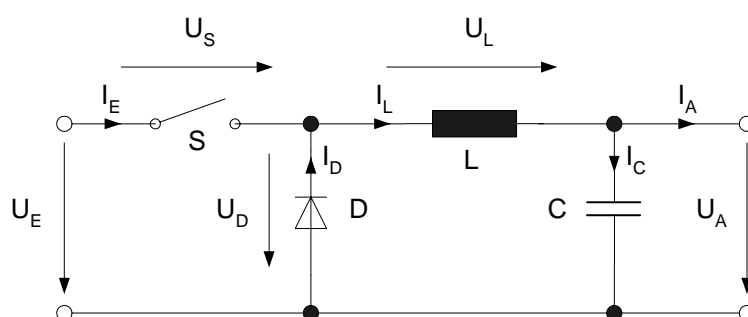


Abbildung 2.2 Abwärtswandler oder Step-Down-Converter

Mit dem Schalter S wird aus der Eingangsspannung eine pulsförmige Spannung erzeugt. Diese Spannung wird mit dem LC-Filter geglättet.

Der große Vorteil des Abwärtswandlers besteht darin, dass der Längstransistor nicht im Linearbetrieb sondern im Schaltbetrieb angesteuert wird. Dadurch entstehen kleiner Verluste am Transistor und es können größere Eingang- zu Ausgangsspannungswandlungen, bei gleichbleibenden Strom, bewerkstelligt werden als bei einem Längsregler.

Die im modularen System der zweiten Generation verwendeten Spannungsregler sind von der Fa. National Semiconductor⁷. Diese Firma bietet auf ihrer Homepage ein Online Berechnungstool für ihre Spannungswandler an.



Abbildung 2.3 Online Berechnungstool WEBENCH®

In Abbildung 2.3 ist das Berechnungsprogramm dargestellt. Es müssen die Eckdaten, wie Eingangsspannungsbereich, Ausgangsspannung und der Ausgangsstrom eingegeben werden. Nach der Berechnung wird dem Anwender eine mögliche Auswahl an Bausteinen und die dazugehörigen Designs vorgeschlagen.

2.1.2. 500 mA Power Supply

In Abbildung 2.4 wird eine typische Anwendung des Spannungsreglers LM2594 von National Semiconductor gezeigt. Mit diesem IC kann ein maximaler Ausgangsstrom von 500 mA erreicht werden.

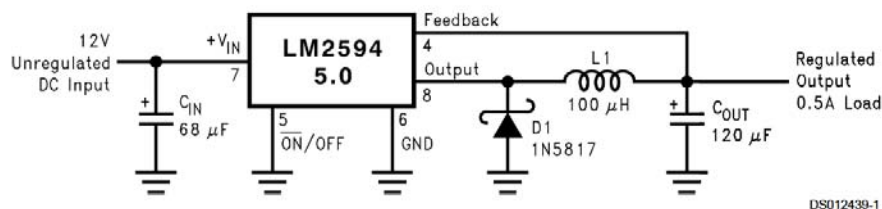


Abbildung 2.4 Typische Anwendung des LM2594⁸

⁷ [National Semiconductor, 2008]

⁸ [National Semiconductor LM2594, 2002]

Dieser Baustein hat folgende Kenndaten

- 3.3 V, 5 V, 12V und einstellbare Ausgangsspannung (beim modularen System wird die Version mit der einstellbaren Ausgangsspannung verwendet)
- Max. Ausgangsstrom von 500 mA
- Eingangsspannungsbereich bis zu 60 V (bei HV Typ)
- Benötigt nur vier externe Bauteile (Eingangs- u. Ausgangskondensator, Spule und Freilaufdiode)
- Arbeitsfrequenz von 150 kHz
- Eingebaute thermische Abschaltung und Strombegrenzung

In Gleichung 2.1 ist der Zusammenhang zwischen der Ausgangsspannung V_{OUT} , der Referenzspannung V_{REF} und den Widerständen R_1 und R_2 dargestellt. Dabei ist R_2 zwischen dem Ausgangskondensator und dem Feedback-Eingang und R_1 zwischen dem Feedback-Eingang und Masse geschaltet.

$$V_{OUT} = V_{REF} \cdot \left(1 + \frac{R_2}{R_1}\right) \quad \text{bei } V_{REF} = 1,23V \quad (2.1)$$

Aus Gleichung 2.1 ergibt sich für V_{OUT} , bei den Bauteilwerten $R_1 = 2.2 \text{ k}\Omega$ und $R_2 = 6.8 \text{ k}\Omega$, ein Wert von ca. 5V.

$$V_{OUT} = 1.23V \cdot \left(1 + \frac{6.8k\Omega}{2.2k\Omega}\right) = 5.03V$$

Der Schaltregler LM2594 wird auf dem Main-Board, dem Motor-Board und dem LCD-Modul verwendet.

2.1.3. 4 A Power Supply

In Abbildung 2.5 wird eine typische Anwendung des Spannungsreglers LM2677 von National Semiconductor gezeigt. Mit diesem IC kann ein maximaler Ausgangsstrom von 5 A erreicht werden.

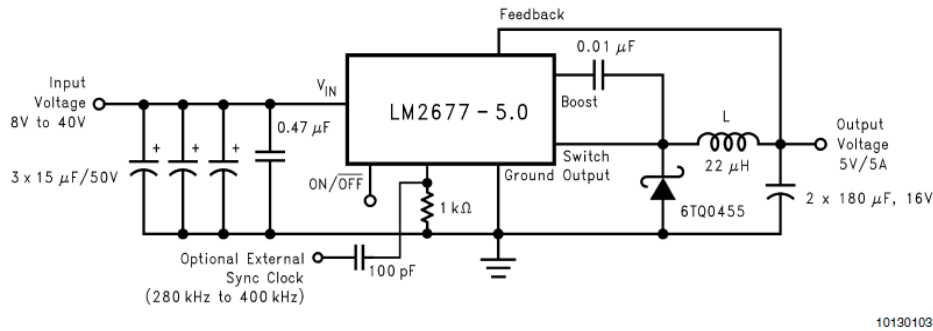


Abbildung 2.5 Typische Anwendung des LM2677⁹

Dieser Baustein hat folgende Kenndaten

- 3.3 V, 5 V, 12V und einstellbare Ausgangsspannung (beim modularen System wird die Version mit der einstellbaren Ausgangsspannung verwendet)
- Wirkungsgrad von bis zu 92%
- Max. Ausgangsstrom von 5 A
- Eingangsspannungsbereich bis zu 40 V
- Arbeitsfrequenz von 260 kHz
- Temperaturbereich von -40 °C bis 125 °C

In Gleichung 2.2 ist der Zusammenhang zwischen der Ausgangsspannung V_{OUT} , der Referenzspannung V_{REF} und den Widerständen R_1 und R_2 dargestellt. Dabei ist R_2 zwischen dem Ausgangskondensator und dem Feedback-Eingang und R_1 zwischen dem Feedback-Eingang und Masse geschaltet.

$$V_{OUT} = V_{REF} \cdot \left(1 + \frac{R_2}{R_1} \right) \quad \text{bei } V_{REF} = 1,21V \quad (2.2)$$

⁹ [National Semiconductor LM2677, 2008]

Aus Gleichung 2.2 ergibt sich für V_{OUT} , bei den Bauteilwerten $R_1 = 2.2 \text{ k}\Omega$ und $R_2 = 6.8 \text{ k}\Omega$, ein Wert von ca. 5V.

$$V_{OUT} = 1.21V \cdot \left(1 + \frac{6.8k\Omega}{2.2k\Omega}\right) = 4.95V$$

Der Schaltregler LM2677 wird auf dem Sensor- bzw. Servo-Board verwendet.

2.2. PROGRAMMIER-INTERFACE

Die Entwicklungsumgebung CodeVisionAVR der Fa. HP InfoTech bietet eine Vielzahl von integrierten ISP-Schnittstellen. Für das modulare System wurde die Programmier-Schnittstelle STK 200/300 gewählt. Bei dieser Schnittstelle wird der Microcontroller über die parallele Schnittstelle des PCs programmiert.

Wie in Abbildung 2.6 dargestellt ist, befindet sich neben der Programmier-Schnittstelle auch noch ein RS232 Interface auf dem Programmier-Board. Mit dieser Schnittstelle können Debug-Meldungen vom Prozessor zum PC übertragen werden. Somit wird Erstellung der Software erleichtert, da durch die Ausgabe von Statusmeldungen leichter ein Fehler in der Software gefunden werden kann.

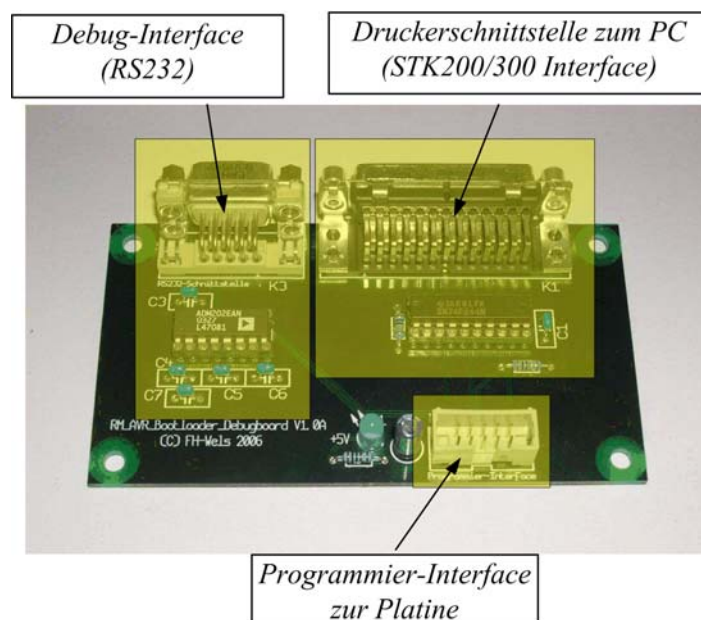


Abbildung 2.6 Bestückte Leiterplatte des Programmier Boards

Zur Verbindung zwischen dem Programmier-Board und dem Microcontroller-Board dient der Stecker K2. Die genaue Pinbelegung dieser Steckers ist in Tabelle 2.1 dargestellt.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	Pluspol der Versorgung für das Programmier-Board
2	RxD	Empfangsleitung der Debug-Schnittstelle
4	SCK	Clock-Leitung der Programmierschnittstelle
5	TxD	Sendeleitung der Debug-Schnittstelle
6	MISO	Master In Slave Out der Programmierschnittstelle
8	MOSI	Master Out Slave In der Programmierschnittstelle
9	GND	Minuspole der Versorgung für das Programmier-Board
10	/Reset	Reset-Leitung der Programmierschnittstelle
3, 7	NC	nicht verwendet

Tabelle 2.1 Steckerbelegung von K2 (Prog.-IF)

2.3. SPI INTERFACE

Das „Serial Peripheral Interface“ (SPI) ermöglicht eine High Speed Verbindung zwischen dem Master und den Slaves. Beim modularen System werden die Daten mit einer Geschwindigkeit von 625 kBaud übertragen.

In Abbildung 2.7 ist die Verbindung zwischen dem Master und einem Slave dargestellt. Das System besteht aus zwei Schieberegistern und einem Takterzeuger. Der Master initiiert die Datenübertragung indem er die Slave Select Leitung (/SS) des entsprechenden Slaves auf Low Potential legt. Dadurch werden die zu sendenden Daten im Slave und im Master in das Schieberegister geschrieben. Anschließend erzeugt der Master das benötigten Clock Signal. Die Daten werden über den Master Out – Slave In Pin (MOSI) vom Master zum Slave übertragen, über den Master In – Slave Out Pin (MISO) werden die Daten vom Slave zum Master transferiert.

Im Microcontroller kann ausgewählt werden ob zuerst das MSB oder LSB übertragen wird. Im modularen System wird zuerst das MSB gesandt.

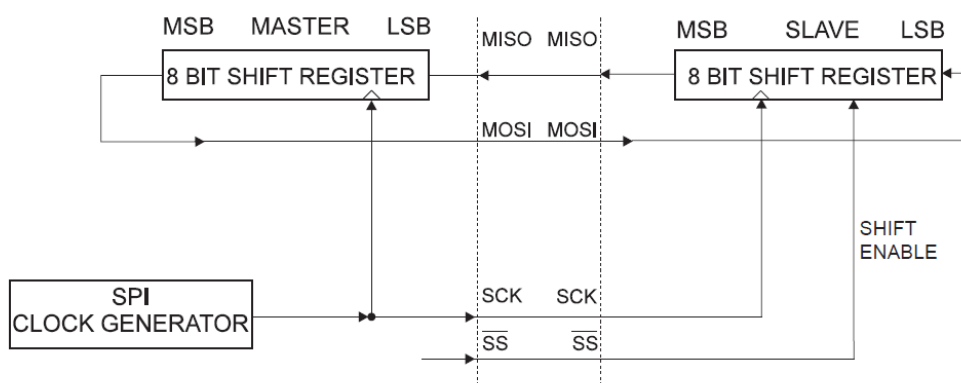


Abbildung 2.7 Funktionsweise der SPI-Schnittstelle¹⁰

In Abbildung 2.8 ist das Prinzipschaltbild der Kommunikation des modularen Systems dargestellt. Der Master ist mit allen Slaves über die Leitungen MISO, MOSI und SCK verbunden. Auf jedem der 14 SPI-Steckern ist ein individuelle Slave Select Anschluss ausgeführt. Mittels diesem Anschluss können die einzelnen Module aktiviert und eine Datenübertragung aufgebaut werden.

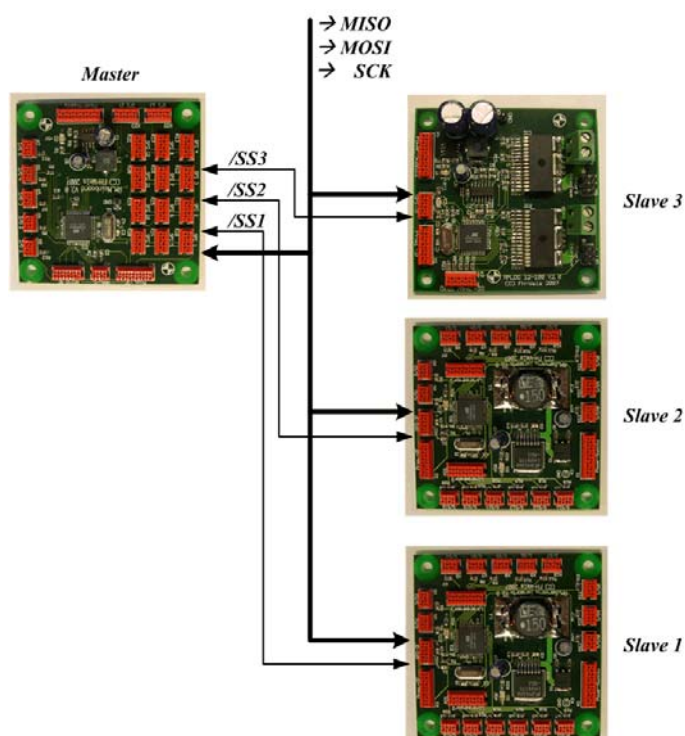


Abbildung 2.8 Blockschaltbild des SPI-Buses

¹⁰ [Atmel, 2006]

3. MAIN-BOARD

3.1. HARDWARE

Das Main-Board bildet die zentrale Steuereinheit des modularen Systems. Im modularen System der ersten Generation war es nur möglich 8 Slaves anzusteuern. Bei komplexeren Systemen kann das allerdings zu wenig sein. Daher wurde auf dem modularen System der zweiten Generation 14 Schnittstellen zum Ansprechen von Slaves implementiert.

Neben den 14 SPI-Schnittstellen (Pinbelegung von K10 bis K23 siehe Tabelle 3.8) befinden sich noch

- 5 analoge/digitale I/Os (K4 bis K7 siehe Tabelle 3.5 und K8 siehe Tabelle 3.6)
- Ein Anschluss zur Ausgabe der Akkuspannung (über drei LEDs – Grün/Gelb/Rot) (K2 siehe Tabelle 3.3)
- Ein Anschluss zum universalen Kommunikations-Board (K1 siehe Tabelle 3.1) – dabei kann zwischen den Schnittstellen RS232, Funkschnittstelle und seriellen Schnittstelle mit TTL Pegel ausgewählt werden (siehe Tabelle 3.2)
- Versorgungs-Interface (K9 siehe Tabelle 3.7)
- Programmier-Interface (K3 siehe Tabelle 3.4)

Für die Versorgung der Elektronik und der Peripherie mit 5V sorgt der Schaltregler LM2594 der Fa. National Semiconductor. Mit diesem Schaltregler ist ein maximaler Ausgangsstrom von 500 mA möglich.

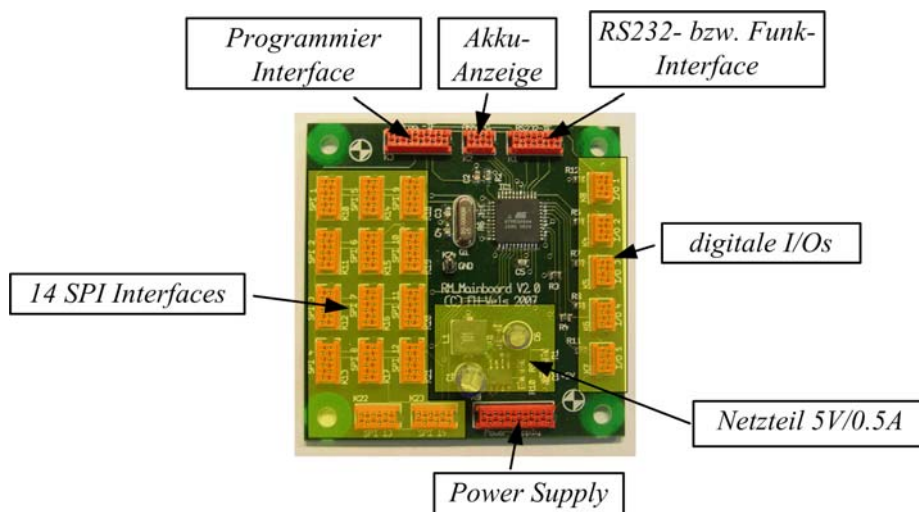


Abbildung 3.1 Bestückte Leiterplatte des Mainboards

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	+5V-Versorgung für das Funk- und RS232 Modul
2	CS1	Chip-Select Leitung 1
3	+12V	+12V-Versorgung
4	CS2	Chip-Select Leitung 2
5	GND	Minuspol der Versorgung
6	CS3	Chip-Select Leitung 3
7	TxD	Sendeleitung der seriellen Schnittstelle
8	RxD	Empfangsleitung der seriellen Schnittstelle

Tabelle 3.1 Steckerbelegung von K1 (Funk- u. RS232-Anbindung)

<i>CS3</i>	<i>CS2</i>	<i>CS1</i>	<i>Funktion</i>
X	0	0	Verbindung: Microcontroller → Funkmodul
X	0	1	Verbindung: Microcontroller → RS232-Schnittstelle
X	1	0	Verbindung: Microcontroller → serielle Schnittstelle mit TTL-Pegel
X	1	1	Keine Funktion

Tabelle 3.2 Auswahlmöglichkeiten der verschiedenen Schnittstellen

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	Akku Grün	LED für Akkustatus „Voll“
2	Akku Gelb	LED für Akkustatus „Mittel“
3	GND	Rückleitung für die LED's
4	Akku Rot	LED für Akkustatus „Leer“

Tabelle 3.3 Steckerbelegung von K2 (Akku Kontrolle)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	Pluspol der Versorgung für das Programmier-Board
2	RxD	Empfangsleitung der Debug-Schnittstelle
4	SCK	Clock-Leitung der Programmierschnittstelle
5	TxD	Sendeleitung der Debug-Schnittstelle
6	MISO	Master In Slave Out der Programmier-Schnittstelle
8	MOSI	Master Out Slave In der Programmier-Schnittstelle
9	GND	Minuspole der Versorgung für das Programmier-Board
10	/Reset	Reset-Leitung der Programmier-Schnittstelle
3, 7	NC	nicht verwendet

Tabelle 3.4 Steckerbelegung von K3 (Prog.-IF)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	+5V-Versorgung
2	+12V	+12V-Versorgung
3	GND	GND
4	I/O	analoger Eingang, digitaler Eingang/Ausgang

Tabelle 3.5 Steckerbelegung von K4 bis K7 (I/O 1...4 – PORTA)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	+5V-Versorgung
2	+12V	+12V-Versorgung
3	GND	GND
4	I/O	digitaler Eingang/Ausgang

Tabelle 3.6 Steckerbelegung von K8 (I/O 5 – PORTB)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1,3,5,7,9,11	+12V	Pluspol der Versorgung
2,4,6,8,10,12	GND	Minuspole der Versorgung

Tabelle 3.7 Steckerbelegung von K9 (Power-Supply)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	/CS	Chip-Select für Slaves
2	SCK	Synchroner Clock
3	MISO	Master In - Slave Out zur Datenübertragung vom Slave
4	MOSI	Master Out - Slave In zur Datenübertragung zum Slave
5,6	GND	GND der Schnittstelle

Tabelle 3.8 Steckerbelegung für K10 bis K23 (SPI-IF)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	GND	Massestift für Messungen

Tabelle 3.9 Steckerbelegung von K24 (Massestift)

3.2. SOFTWARE

3.2.1. Treiber Main-Board

Wie in Tabelle 3.10 dargestellt, ist die Software des Main-Board in 14 Module aufgeteilt. Es wurde darauf geachtet, dass die Module leicht auf andere Projekte bzw. Plattformen portiert werden können.

<i>Modul</i>	<i>Funktion</i>
RM_Mainboard_main.c	Hauptprogramm
multitasking.c	Multitaskingsystem des Main-Boards (siehe Kapitel 3.2.2)
adc.c	stellt die Treiber für den AD-Wandler zur Verfügung
ports.c	stellt die Treiber/Makros für den Zugriff auf die I/Os zur Verfügung
uart.c	stellt Treiber für die serielle Schnittstelle zur Verfügung; über diese Schnittstelle kann auch auf die Steuerung und Sensorik des Roboters zugegriffen werden
timer0.c	initialisiert den Timer 0 → 100 µs - Zeitbasis für das Multitaskingsystem
spi.c	stellt die Treiber für die SPI-Schnittstelle zur Verfügung; hier läuft die Kommunikation mit den Slaves
rtrlan.c	hier befinden sich alle Meldungen des Busses
colour.c	dient zur Feststellung der Spielfarbe
obstacle.c	stellt die komplette Gegnererkennung zur Verfügung
debug.c	stellt einen Debug-Task zur Verfügung; hier können Programmteile getestet werden, ohne dass die ursprünglichen Programmteile verändert werden müssen
drive.c	stellt die komplette Fahrkontrolle zur Verfügung; es können beliebige Koordinaten angefahren werden
main_routine.c	hier befinden sich die Tasks für die Ablaufsteuerung
playtime.c	zur Überwachung der Spielzeit

Tabelle 3.10 Softwaremodule des Main-Boards

Hauptprogramm

Wie in Abbildung 3.2 dargestellt, wird im Hauptprogramm zuerst die Hardware des Microcontrollers und das Multitaskingsystem initialisiert (Funktion *InitDevice()*). Im Anschluss daran werden Setup-Informationen an die Slave-Boards übergeben. In der Main-Loop arbeitet der Systemkernel des Multitaskingsystems die anstehenden Tasks ab (Funktion *MultitaskingSystem ()*).

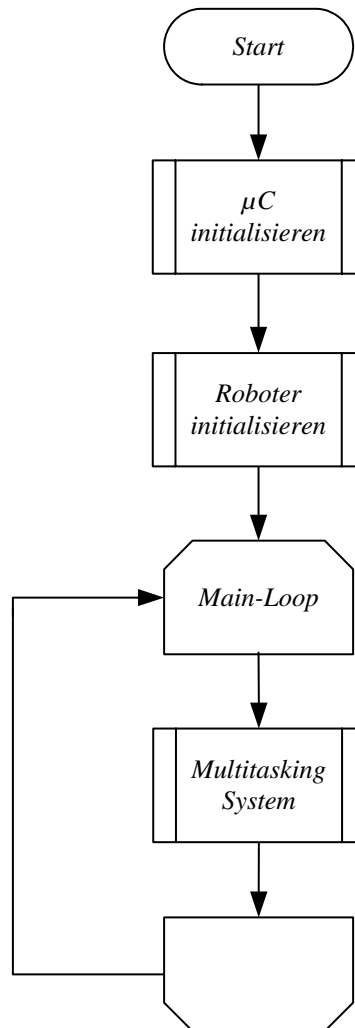


Abbildung 3.2 Flussdiagramm Hauptprogramm Main-Board

UART

Der UART wird beim modularen System als Debug-Schnittstelle verwendet. Es werden einerseits Meldungen vom Roboter zum PC übertragen, wie auch Meldungen vom PC zum Roboter.

Die empfangenen Meldungen werden in der seriellen Interrupt-Routine des Microcontroller, wie in Abbildung 3.3 dargestellt, abgearbeitet.

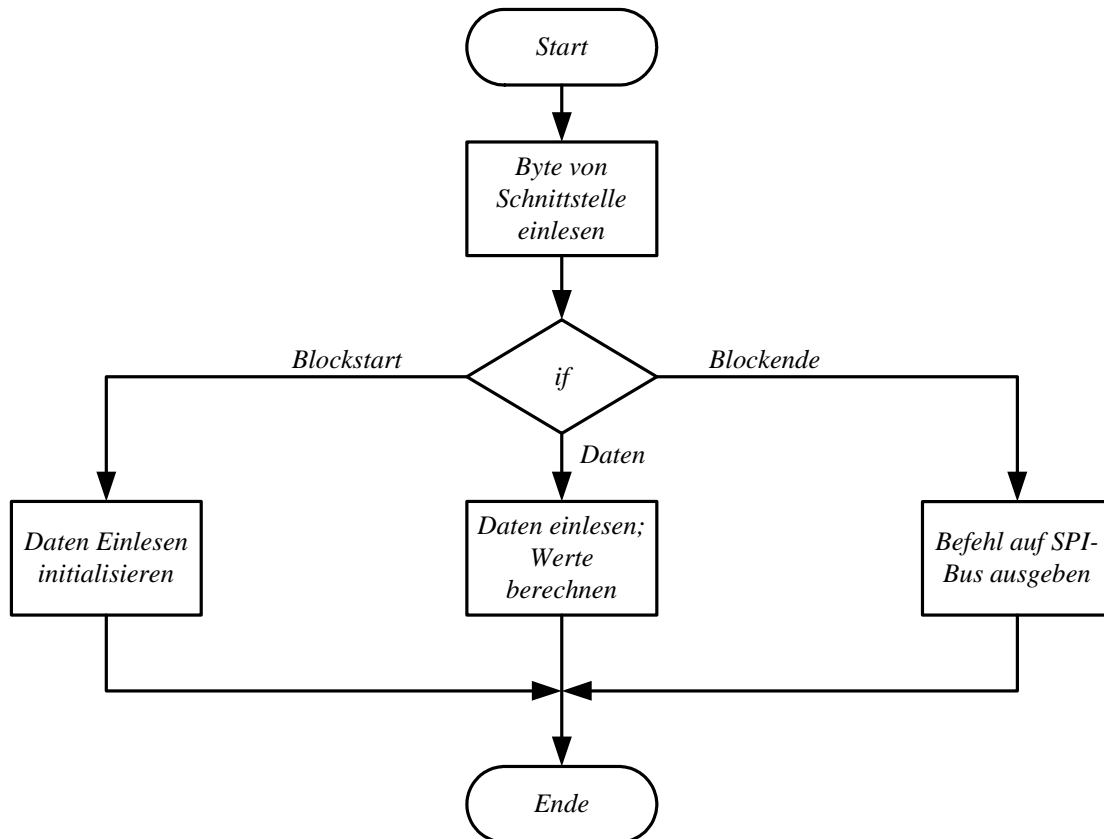


Abbildung 3.3 Flussdiagramm „UART abarbeiten“

Es wurde ein PC-Programm für die „Fernsteuerung“ des Roboters in LabView entwickelt. Mit diesem Programm können dem Roboter Meldungen fürs Fahren oder zur Ansteuerung der Servos gesendet werden.

Somit können Einstellungen am Roboter vorgenommen werden, ohne dass das Programm ständig verändert und der Roboter neu programmiert werden muss.

Im Folgenden werden die wichtigsten Meldungen näher beschrieben. Bei allen Meldungen ist das Format gleich, denn alle Meldungen beginnen mit einem Start-Byte (# → 0x23) und enden mit einem Stopp-Byte (* → 0x2A). Das ist notwendig, dass sich der Empfänger auf die Meldung synchronisieren kann. Damit das Start- bzw. Stopp-Byte in der Meldung nicht auftaucht, werden die Daten nicht als Binärwert übertragen sondern als ASCII-Zeichen, d.h.

der Wert 100 wird nicht in einem Byte als 0x64 übertragen sondern in drei Bytes 1 (0x31) 0 (0x30) 0 (0x30). Dadurch kann bei der Übertragung auf ESC-Sequenzen, was die Übertragung verkomplizieren würde, verzichtet werden.

Nach dem Start-Byte wird das Modus-Byte übertragen, daraus kann der Befehl abgelesen werden, z.B. vorwärts fahren oder rückwärts fahren. Im Anschluss werden die Daten gesendet.

Meldung „vorwärts fahren“

Wie in Abbildung 3.4 und Tabelle 3.11 dargestellt, wird dem Roboter nach dem Modus-Byte zuerst die Geschwindigkeit übertragen und dann die zu fahrende Distanz. Der Bereich, in dem die Geschwindigkeit ausgewählt werden kann, beträgt 0 ... 99 Inkremente pro ms, das entspricht bei einem Raddurchmesser von 3 cm und einem Inkrementgeber mit 1024 Impulsen pro Umdrehung eine Geschwindigkeit von ca. 1,2 m/s. Die Distanz kann von 0 ... 99.999 Einheiten eingestellt werden, wobei auf dem Roboter eingestellt werden muss, ob es sich um mm oder Inkremente handelt.

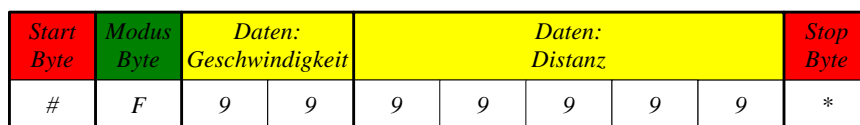


Abbildung 3.4 Meldung „vorwärts fahren“

<i>Byte Nr.</i>	<i>Wert [ASCII]</i>	<i>Beschreibung</i>
1	#	Start-Byte
2	F	Meldungsnummer
3	0 ... 9	Zehnerstelle Geschwindigkeit
4	0 ... 9	Einerstelle Geschwindigkeit
5	0 ... 9	Zehntausenderstelle Weg
6	0 ... 9	Tausenderstelle Weg
7	0 ... 9	Hunderterstelle Weg
8	0 ... 9	Zehnerstelle Weg
9	0 ... 9	Einerstelle Weg
10	*	Stopp-Byte

Tabelle 3.11 Beschreibung der Meldung „vorwärts fahren“

Meldung „rückwärts fahren“

Wie in Abbildung 3.5 und Tabelle 3.12 dargestellt, wird dem Roboter nach dem Modus-Byte zuerst die Geschwindigkeit übertragen und dann die zu fahrende Distanz. Der Bereich, in dem die Geschwindigkeit ausgewählt werden kann, beträgt 0 ... 99 Inkremente pro ms, das entspricht bei einem Raddurchmesser von 3 cm und einem Inkrementgeber mit 1024 Impulsen pro Umdrehung eine Geschwindigkeit von ca. 1,2 m/s. Die Distanz kann von 0 ... 99.999 Einheiten eingestellt werden, wobei auf dem Roboter eingestellt werden muss, ob es sich um mm oder Inkremente handelt.

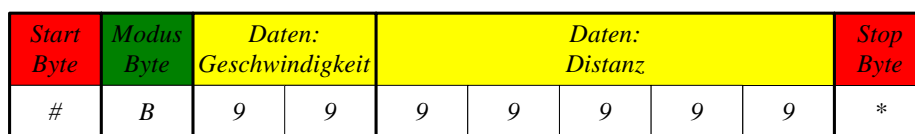


Abbildung 3.5 Meldung „rückwärts fahren“

<i>Byte Nr.</i>	<i>Wert [ASCII]</i>	<i>Beschreibung</i>
1	#	Start-Byte
2	B	Meldungsnummer
3	0 ... 9	Zehnerstelle Geschwindigkeit
4	0 ... 9	Einerstelle Geschwindigkeit
5	0 ... 9	Zehntausenderstelle Weg
6	0 ... 9	Tausenderstelle Weg
7	0 ... 9	Hunderterstelle Weg
8	0 ... 9	Zehnerstelle Weg
9	0 ... 9	Einerstelle Weg
10	*	Stop-Byte

Tabelle 3.12 Beschreibung der Meldung „rückwärts fahren“

Meldung „Drehung links“

Wie in Abbildung 3.6 und Tabelle 3.13 dargestellt, wird dem Roboter nach dem Modus-Byte zuerst die Drehrichtung, dann die Geschwindigkeit und zum Abschluss der zu fahrende Winkel übertragen. Für Drehung nach links wird ein *l* übergeben. Der Bereich, in dem die Geschwindigkeit ausgewählt werden kann, beträgt 0 ... 99 Inkremente pro ms, das entspricht bei einem Raddurchmesser von 3 cm und einem Inkrementgeber mit 1024 Impulsen pro Umdrehung eine Geschwindigkeit von ca. 1,2 m/s pro Rad. Der Winkel kann von 0 ... 99.999 Einheiten eingestellt werden, wobei auf dem Roboter eingestellt werden muss, ob es sich um Zehntel Grad oder Inkremente handelt.

<i>Start Byte</i>	<i>Modus Byte</i>		<i>Daten: Geschwindigkeit</i>		<i>Daten: Winkel</i>				<i>Stop Byte</i>
#	<i>T</i>	<i>l</i>	9	9	9	9	9	9	*

Abbildung 3.6 Meldung „Drehung links“

<i>Byte Nr.</i>	<i>Wert [ASCII]</i>	<i>Beschreibung</i>
1	#	Start-Byte
2	<i>T</i>	Meldungsnummer
3	<i>l</i>	Drehrichtung (<i>l</i> ... links)
4	<i>0 ... 9</i>	Zehnerstelle Geschwindigkeit
5	<i>0 ... 9</i>	Einerstelle Geschwindigkeit
6	<i>0 ... 9</i>	Tausenderstelle Weg
7	<i>0 ... 9</i>	Hunderterstelle Weg
8	<i>0 ... 9</i>	Zehnerstelle Weg
9	<i>0 ... 9</i>	Einerstelle Weg
10	*	Stopp-Byte

Tabelle 3.13 Beschreibung der Meldung „Drehung links“

Meldung „Drehung rechts“

Wie in Abbildung 3.7 und Tabelle 3.14 dargestellt, wird dem Roboter nach dem Modus-Byte zuerst die Drehrichtung, dann die Geschwindigkeit und zum Abschluss der zu fahrende Winkel übertragen. Für Drehung nach rechts wird ein *r* übergeben. Der Bereich, in dem die Geschwindigkeit ausgewählt werden kann, beträgt 0 ... 99 Inkremente pro ms, das entspricht bei einem Raddurchmesser von 3 cm und einem Inkrementgeber mit 1024 Impulsen pro Umdrehung eine Geschwindigkeit von ca. 1,2 m/s pro Rad. Der Winkel kann von 0 ... 99.999 Einheiten eingestellt werden, wobei auf dem Roboter eingestellt werden muss, ob es sich um Zehntel Grad oder Inkremente handelt.

<i>Start Byte</i>	<i>Modus Byte</i>		<i>Daten: Geschwindigkeit</i>		<i>Daten: Winkel</i>				<i>Stop Byte</i>
#	<i>T</i>	<i>r</i>	9	9	9	9	9	9	*

Abbildung 3.7 Meldung „Drehung rechts“

<i>Byte Nr.</i>	<i>Wert [ASCII]</i>	<i>Beschreibung</i>
1	#	Start-Byte
2	<i>T</i>	Meldungsnummer
3	<i>r</i>	Drehrichtung (<i>r</i> ... rechts)
4	<i>0 ... 9</i>	Zehnerstelle Geschwindigkeit
5	<i>0 ... 9</i>	Einerstelle Geschwindigkeit
6	<i>0 ... 9</i>	Tausenderstelle Weg
7	<i>0 ... 9</i>	Hunderterstelle Weg
8	<i>0 ... 9</i>	Zehnerstelle Weg
9	<i>0 ... 9</i>	Einerstelle Weg
10	*	Stopp-Byte

Tabelle 3.14 Beschreibung der Meldung „Drehung rechts“

Meldung „einzelnen Motor ansteuern“

Mit dieser Meldung kann jeder Motor individuell angesteuert werden. Wie in Abbildung 3.8 und Tabelle 3.15 dargestellt, wird dem Roboter nach dem Modus-Byte zuerst die Motornummer, dann die Drehrichtung und zum Abschluss die Geschwindigkeit übertragen. Die Motornummer kann von 1 ... 16 betragen. Für Rechtslauf des Motors wird ein *r* übergeben, für Linkslauf ein *l*. Der Bereich, in dem die Geschwindigkeit ausgewählt werden kann, beträgt 0 ... 99 Inkremente pro ms.

<i>Start Byte</i>	<i>Modus Byte</i>	<i>Daten: Motor Nr.</i>		<i>Daten: Richtung</i>	<i>Daten: Geschwindigkeit</i>		<i>Stop Byte</i>
#	R	1	6	l/r	9	9	*

Abbildung 3.8 Meldung „einzelnen Motor ansteuern“

<i>Byte Nr.</i>	<i>Wert [ASCII]</i>	<i>Beschreibung</i>
1	#	Start-Byte
2	R	Meldungsnummer
3	0 ... 1	Zehnerstelle Motornummer
4	0 ... 9	Einerstelle Motornummer
5	l/r	Drehrichtung (l/r ... links/rechts)
6	0 ... 9	Zehnerstelle Geschwindigkeit
7	0 ... 9	Einerstelle Geschwindigkeit
8	*	Stop-Byte

Tabelle 3.15 Beschreibung der Meldung „Drehung links“

Meldung „Servo ansteuern“

Mit dieser Meldung kann jeder Servo individuell angesteuert werden. Wie in Abbildung 3.9 und Tabelle 3.16 dargestellt, wird dem Roboter nach dem Modus-Byte zuerst die Servonummer und dann der Drehwinkel übertragen. Die Servonummer kann von 1 ... 16 betragen. Für den Drehwinkel wird ein Wert von 0 ... 255 übergeben, wobei der Wert 0 dem Linksanschlag des Servos entspricht und der Wert 255 dem Rechtsanschlag. Wie viele Grad pro Einheit überstrichen werden, hängt vom verwendeten Servo ab, für einen Servo mit einem Stellwinkel von $\pm 70^\circ$ werden ca. $0,55^\circ$ pro Einheit überstrichen.

Start Byte	Modus Byte	Daten: Servo Nr.		Daten: Winkel			Stop Byte
#	S	1	6	2	5	5	*

Abbildung 3.9 Meldung „Servo ansteuern“

Byte Nr.	Wert [ASCII]	Beschreibung
1	#	Start-Byte
2	S	Meldungsnummer
3	0 ... 1	Zehnerstelle Servonummer
4	0 ... 9	Einerstelle Servonummer
5	0 ... 2	Hunderterstelle Servostellung
6	0 ... 9	Zehnerstelle Servostellung
7	0 ... 9	Einerstelle Servostellung
8	*	Stopp-Byte

Tabelle 3.16 Beschreibung der Meldung „Servo ansteuern“

Wegberechnung

Die Wegberechnung findet im Modul *drive.c* statt. Mit dem Programm *DriveToPosition* wird die Bewegung gestartet. In das Programm wird die gewünschte X- bzw. Y-Position, wer die Bewegung initiiert hat und ob der Roboter vorwärts oder rückwärts fahren soll, übergeben.

Wie in Abbildung 3.10 dargestellt, wird von der aktuellen Position und Orientierung der zu drehende Winkel φ und die zu fahrende Strecke l berechnet.

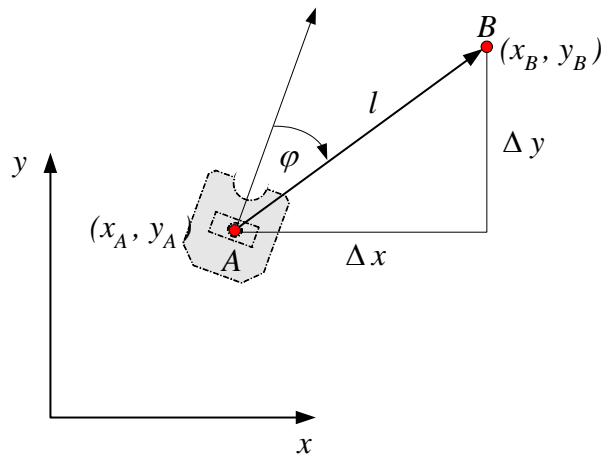


Abbildung 3.10 Schematische Darstellung der Wegberechnung

Für die Strecke l ergibt sich nach Gleichung 3.1 folgender Wert:

$$l = \sqrt{\Delta x^2 + \Delta y^2} \quad (3.1)$$

$$l = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

Der zu drehende Winkel kann aus der aktuellen Orientierung φ' und dem Endwinkel berechnet werden (siehe Gleichung 3.2).

$$\varphi = \varphi' - \arcsin\left(\frac{\Delta y}{l}\right) \quad (3.2)$$

Bei positiven Drehwinkel φ muss nach rechts gedreht werden, bei einem negativen Drehwinkel nach links. Wenn der Roboter die Strecke rückwärts fahren soll, muss die aktuelle Orientierung umgerechnet werden. Für diese Umrechnung muss zur aktuellen Orientierung 180° addiert werden.

Der Ablauf des Programms ist in Abbildung 3.11 dargestellt. Es wird über die Programmaufrufe *DriveToPosition* (es werden die Koordinaten absolut angefahren), *DriveToPositionRelative* (es wird eine Strecke gefahren) oder *TurnToPosition* (der Roboter dreht sich relativ um einen bestimmten Winkel) die Bewegung initiiert.

Dabei wird ein Task angelegt, der die gesamte Berechnung, Befehlsweitergabe an das Motorboard und die Bewegungsüberwachung übernimmt. Wenn die Bewegung abgeschlossen ist, wird der aufrufende Task wieder „aufgeweckt“.

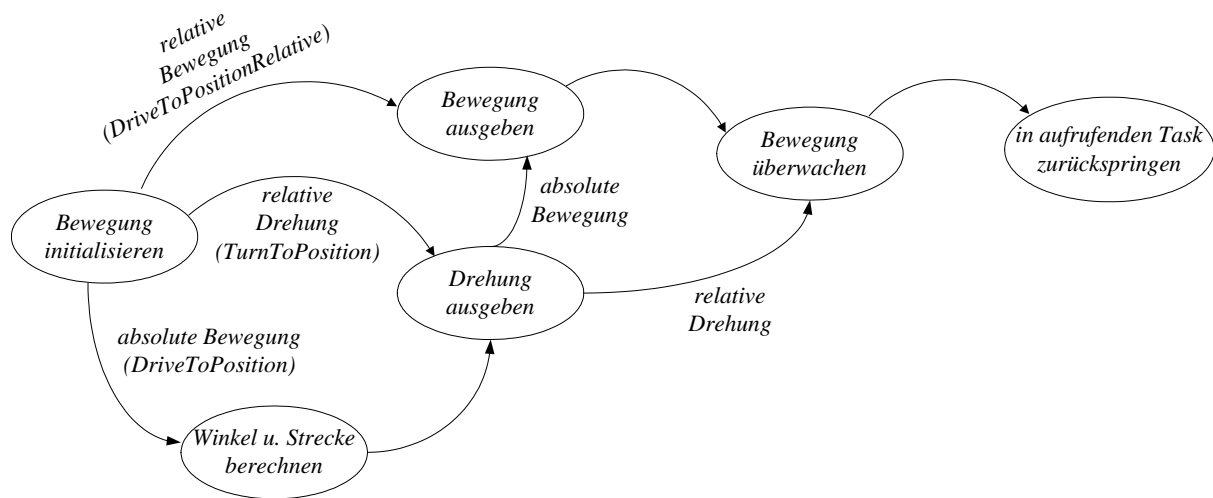


Abbildung 3.11 State-Diagramm des Motoransteuerung

Übergabeparameter in das Programm *DriveToPosition*:

- *siXPos* – absolute X-Koordinate in mm
- *siYPos* – absolute Y-Koordinate in mm
- *ucSpeed* – Geschwindigkeit
- *ucDir* – gibt an ob der Roboter die Position vorwärts bzw. rückwärts anfahren soll
- *ucCallFrom* – aufrufender Task → dieser Task wird nach Beendigung der Bewegung wieder „aufgeweckt“

Übergabeparameter in das Programm *DriveToPositionRelative*:

- *uiDistance* – absolute X-Koordinate in mm
- *ucSpeed* – Geschwindigkeit
- *ucDir* – gibt an ob der Roboter die Position vorwärts bzw. rückwärts anfahren soll
- *ucCallFrom* – aufrufender Task → dieser Task wird nach Beendigung der Bewegung wieder „aufgeweckt“

Übergabeparameter in das Programm *TurnToPosition*:

- *siAngle* – absolute X-Koordinate in $0,1^\circ$
- *ucSpeed* – Geschwindigkeit
- *ucDir* – gibt an ob der Roboter die nach rechts oder links drehen soll
- *ucCallFrom* – aufrufender Task → dieser Task wird nach Beendigung der Bewegung wieder „aufgeweckt“

3.2.2. Multitasking System¹¹

Im modularen System der ersten Generation wurden die Aufgaben des Main-Boards in einer State-Machine abgearbeitet. Dadurch konnte immer nur eine Aufgabe pro Zeit erledigt werden und es konnte entweder gefahren oder die Spielelemente verwaltet werden. Dieser Umstand brachte einige Nachteile für die Performance des Roboters.

Daher war es ein großes Ziel, für das modulare System der zweiten Generation ein Multitasking System zu implementieren. Natürlich kann bei einem Multitasking System nur eine Aufgabe pro Zeit abgearbeitet werden, jedoch wird durch eine bessere Verwaltung der Aufgaben eine Quasigleichzeitigkeit erreicht.

Im Folgenden sollen zwei verschiedene Ansätze für ein Multitasking System näher betrachtet werden, bevor auf das realisierte System eingegangen wird.

Kooperatives Multitasking

Beim kooperativen Multitasking werden die Tasks nacheinander abgearbeitet, wobei jeder Task soviel Rechenzeit bekommt wie benötigt wird (siehe Abbildung 3.12). Dieses Multitasking System hat einen geringen Verwaltungsaufwand und benötigt nur wenig System-Ressourcen. Die Anforderungen an die programmierten Tasks sind dafür um so höher, denn um einen reibungslosen Ablauf zu gewährleisten, dürfen diese nicht zu komplex sein und zuviel Systemzeit verbrauchen. Ebenso würde das System zum Stillstand kommen, wenn ein Task in einer Endlosschleife verharren würde.

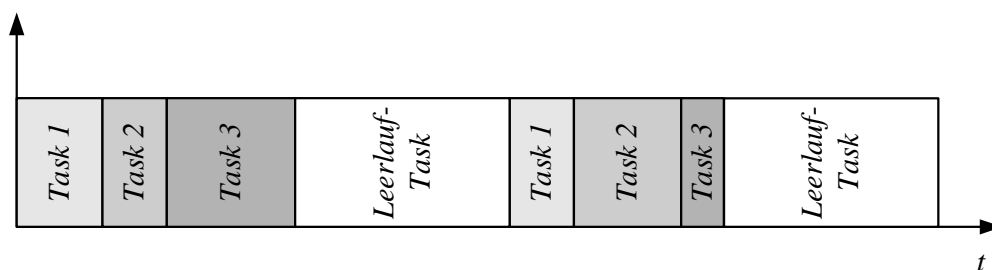


Abbildung 3.12 Zeitverhalten eines kooperativen Multitasking Systems

¹¹ [Bräunl, 2003]

Präemptives Multitasking

Beim präemptiven Multitasking werden den Tasks vom Betriebssystemkern bestimmte Zeitschlitze zur Abarbeitung der Aufgaben zugewiesen. Nach Ablauf der zur Verfügung stehenden Zeit wird der Task „schlafen“ gelegt und der nächste Task bekommt die Rechenzeit zur Verfügung gestellt (siehe Abbildung 3.13). Bei diesem Verfahren stehen die auszuführenden Tasks in einer Warteschlange und werden nach dem sog. Round-Robin-Scheduling abgearbeitet.

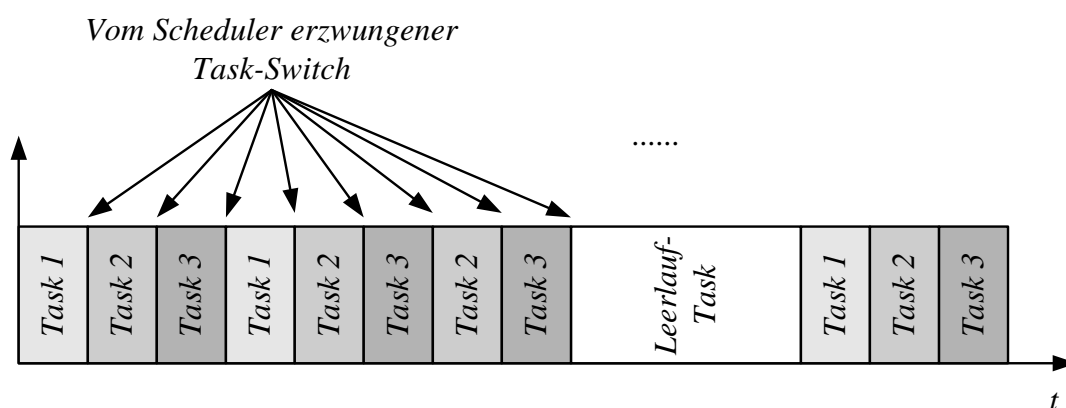


Abbildung 3.13 Zeitverhalten eines präemptiven Multitasking Systems

Die Ansprüche an die programmierten Tasks sinken bei diesem System erheblich, da kein Task das System vollständig zum Erliegen bringen kann. Im Gegenzug steigt der Aufwand beim Kernel, denn es muss darauf geachtet werden, dass alle verwendeten Variablen eines Tasks und die Rücksprungadresse in den Task zwischengespeichert werden müssen.

Multitasking System des modularen Systems

Beim modularen System der zweiten Generation wurde darauf geachtet, dass das Multitasking-System nur sehr wenige Ressourcen in Anspruch nimmt. Daher wurde ein kooperatives System umgesetzt.

Das Multitasking System benötigt ca. 320 Byte im Codespeicher und 16 Byte pro Task im Datenspeicher. Um ein deterministisches Verhalten zu erreichen wird ein Timer benötigt. Für diese Aufgabe wurde der Timer 0 gewählt, wobei alle 100 μ s ein Interrupt ausgelöst wird.

In das Multitasking System wurde keine Prioritätsstufen implementiert, d.h. es gibt keine Reihung der Tasks nach Wichtigkeit. Ein solches System würde den Verwaltungsaufwand nur erhöhen und es ist einfacher, wichtige Aufgaben in einem Interrupt abzuarbeiten.

Das Multitasking System bietet einfache Methoden zur Verwaltung der Tasks und zur Kommunikation zwischen den Tasks

- *SET_TASK(Task_Nbr, Aktion)* – mit dieser Methode kann ein Task freigegeben werden (*Aktion = ENABLE*) bzw. sein Status auf zyklisch gesetzt werden (*Aktion = CYCLE*)
- *TASK_IS_ENABLED(Task_Nbr)* – es kann abgefragt werden ob der Task freigegeben ist
- *DISABLE_TASK(Task_Nbr)* – Task sperren
- *SET_CYCLE(Task_Nbr, Zykluszeit)* – die Zykluszeit kann auf ein vielfaches von 100 µs gesetzt werden, d.h. Zykluszeit = 100 entspricht einer effektiven Zeit von 10 ms
- *IS_CYCLE_TASK(Task_Nbr)* – zum Abfragen ob der Task ein zyklischer Task ist
- *SET_CALLER(Task_Nbr, Caller_Task_Nbr)* – um festzustellen von welcher Quelle der Task aktiviert wurde (*SYSTEM_CALL – 0xFF, CYCLE_CALL – 0xFE* bzw. Tasknummer eines anderen Tasks)
- *CALLER_IS(Task_Nbr)* – zum Abfragen wer den Task aufgerufen hat
- *SET_TASK_HANDLE(Task_Nbr, Handle)* – es wird die Adresse des aufzurufenden Unterprogramms (Task) auf einen Functionpointer geschrieben
- *WRITE_TO_MAILBOX(Task_Nbr, Mailbox_Nbr, Data)* – Es wird das Byte (*Data*) mit der Nummer (*Mailbox_Nbr*) in die Mailbox geschrieben
- *READ_FROM_MAILBOX(Task_Nbr, Mailbox_Nbr)* – Es wird das Byte mit der Nummer (*Mailbox_Nbr*) aus der Mailbox gelesen

Zur Verwaltung des Tasks wurde eine Struktur angelegt, die den Status und das Verhalten des Tasks widerspiegelt. Die enthaltenen Daten sind Tabelle 3.17 zu entnehmen.

<i>Variable</i>	<i>Datentyp</i>	<i>Beschreibung</i>
<i>ucStatus</i>	<i>unsigned char – 8 Bit</i>	Gibt über den Zustand Auskunft 0x00 ... Task ist auf Status <i>DISABLE</i> 0x01 ... Task ist auf Status <i>ENABLE</i> 0x02 ... Task ist auf Status <i>CYCLE</i>
<i>uiIntervall</i>	<i>unsigned int – 16 Bit</i>	Intervalldauer in 100 μ s - Zyklen
<i>ucCaller</i>	<i>unsigned char – 8 Bit</i>	Aufrufendes Programm 0xFF ... Systemaufruf 0xFE ... zyklischer Aufruf ansonsten Tasknummer
<i>ucMailBox[10]</i>	<i>unsigned char – 8 Bit</i> (Array von 10 Byte)	Mittels der Mail-Box kann zwischen den Tasks kommuniziert werden
<i>unsigned char</i> (*fpTaskHandle)(void)	<i>Functionpointer – 16 Bit</i>	Pointer auf das auszuführende Programm

Tabelle 3.17 Variablen der Datenstruktur „Multitasking“

Mit dem Parameter *MAX_TASK* im Headerfile *multitask.h* kann die Anzahl der maximal zur Verfügung stehenden Tasks begrenzt werden. Es wird empfohlen nur so viele Tasks anzulegen, wie verwendet werden, denn ansonsten kann sich der Task-Switch verlängern und es wird mehr Speicher im Datenspeicher belegt als nötig ist.

Im folgenden wird erklärt wie ein Task angelegt und initialisiert wird.

Es muss eine Routine für den Task angelegt werden (Muster siehe Listing 3.1). Als Rückgabeparameter muss dem Multitasking System der neue Status (*ENABLE*, *DISABLE* oder *CYCLE*) übermittelt werden. Wenn der Task als zyklischer Task angelegt ist, muss noch zusätzlich die Zykluszeit gesetzt werden (*SET_CYCLE*).

```

/*****
***  FUNKTIONNAME:      MusterTask          ***
***  FUNKTION:         zeigt das Format eines Tasks          ***
***  TRANSMIT PARAMETER:  Taskstatus          ***
***  RECEIVE PARAMETER.:  NO                  ***
*****/
unsigned char ucMusterTask(void)
{
// ****          hier steht das Programm des Tasks          ****
// ****

// ****

// Status des Tasks zurückgeben (ENABLE, DISABLE oder CYCLE)
return(DISABLE);
}

```

Listing 3.1 Programm eines Muster Tasks

In der Initialisierungsroutine (Muster siehe Listing 3.2) eines Tasks muss der Status des Tasks beschrieben werden (*SET_TASK*) und der Handle auf das Unterprogramm übergeben werden (*SET_TASK_HANDLE*). Wenn der Task als zyklischer Task angelegt werden soll, muss noch eine Zykluszeit gesetzt werden (*SET_CYCLE*).

```

/*****
***  FUNKTIONNAME:      InitMusterTask      ***
***  FUNKTION:         initialisiert den MusterTask          ***
***  TRANSMIT PARAMETER:  NO                  ***
***  RECEIVE PARAMETER.:  NO                  ***
*****/
void InitMusterTask(void)
{
SET_CYCLE(MUSTERTASK_TASKNBR, 10000);
SET_TASK(MUSTERTASK_TASKNBR, CYCLE);
SET_TASK_HANDLE(MUSTERTASK_TASKNBR, ucMusterTask);
}

```

Listing 3.2 Initialisierungsroutine des Muster Task

3.2.3. SPI Schnittstelle

Im folgenden (Tabelle 3.18 bis Tabelle 3.36) werden die Befehle, über die vom Master auf die Slaves via SPI-Schnittstelle zugegriffen werden kann, vorgestellt.

Das Präfix *uc* bei Variablen oder bei Funktionen zeigt an, dass es sich um einen Datentypen *unsigned char* (8-Bit) handelt. Das Präfix *ui* bei Variablen oder bei Funktionen zeigt an, dass es sich um einen Datentypen *unsigned int* (16-Bit) handelt, *pui* bedeutet Pointer auf *unsigned int*.

<i>Funktionsname</i>	<i>InitServoBoard</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucServoNbr	Servonummer (1 ... 16)
	ucServoTyp	Servotype <i>SV_NOT_INIT</i> (0x00) ... kein Servo <i>SV_CARSON</i> (0x01) ... Servo der Fa. Carson <i>SV_HI TECH</i> (0x02) ... Servo der Fa. HiTech
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.18 Befehl zum Servo-Board initialisieren

<i>Funktionsname</i>	<i>InitSensorBoard</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucSensorNbr	Sensornummer (1 ... 16)
	ucSensorTyp	Sensortype <i>SB_NOT_INIT</i> (0x00) ... kein Sensor <i>SB_GP2D120</i> (0x01) ... GP2D120 Sensor der Fa. Sharp <i>SB_GP2D12</i> (0x02) ... GP2D120 Sensor der Fa. Sharp <i>SB_AXE045</i> (0x03) ... AXE045 Sensor der Fa. Roboterteile <i>SB_DIGITAL</i> (0x04) ... digitaler Eingang <i>SB_GP2Y0A02F</i> (0x05) ... GP2Y0A02F Sensor der Fa. Sharp <i>SB_SRF08</i> (0x06) ... SRF08 Sensor der Fa. Roboterteile
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.19 Befehl zum Sensor-Board initialisieren

<i>Funktionsname</i>	<i>InitMotorboard</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucMotortype	Sensortype <i>MB_NOT_INIT</i> (0x00) ... kein Motor <i>MB_DC_MOTOR</i> (0x01) ... zwei DC-Motoren <i>MB_STEPPER</i> (0x02) ... ein Schrittmotoren
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.20 Befehl zum Motor-Board initialisieren

<i>Funktionsname</i>	<i>ucReadSensor</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucSensorNbr	Sensornummer (1 ... 16)
<i>Rückgabewerte</i>	Sensorwert (unsigned int – 16 Bit Wert) GP2D12, GP2D120, GP2Y0A02F, SRF08 → Abstand in mm AXE045 → weiß ... 0x00 blau ... 0x01 rot ... 0x02 grün ... 0x03 digitaler Eingang → Eingangszustand (Low ... 0, High ... 1)	

Tabelle 3.21 Befehl zum Sensor auslesen

<i>Funktionsname</i>	<i>SetIO</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucIOPin	Pinnummer (1 ... 16)
	ucValue	0 ... Low wird ausgegeben 1 ... High wird ausgegeben
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.22 Befehl zum Setzen/Rücksetzen eines Ausgangs

<i>Funktionsname</i>	<i>SetPin</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucIOPin	Pinnummer (1 ... 16)
	ucValue	0 ... Pin als Input initialisieren 1 ... Pin als Output initialisieren
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.23 Befehl zum Umschalten zwischen Ein- bzw. Ausgang

<i>Funktionsname</i>	<i>SetServoI</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucServoNbr	Servonummer (0 ... 15)
	ucAngle	Winkel → Wertebereich 0 ... 255 entspricht dem maximal möglichen Winkel des Servos (z.B. 140°)
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.24 Befehl zum Setzen eines Winkels eines ausgewählten Servos

<i>Funktionsname</i>	<i>MotorTurn</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucDirection	<i>LEFT</i> (0x00) ... Linksdrehung <i>RIGHT</i> (0x01) ... Rechtsdrehung
	ucSpeed	Geschwindigkeit (Inkmente pro 2 ms)
	ucDistance	Winkel in 0,1°
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.25 Befehl zum Ausgeben einer Drehbewegung

<i>Funktionsname</i>	<i>MotorDrive</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucDirection	<i>FORWARD</i> (0x00) ... vorwärts fahren <i>BACKWARD</i> (0x01) ... rückwärts fahren
	ucSpeed	Geschwindigkeit (Inkremente pro 2 ms)
	ucDistance	Weg in mm
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.26 Befehl zum Vor- bzw. Rückwärtsfahren

<i>Funktionsname</i>	<i>MotorSetIndividual</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucDirectionRight	<i>FORWARD</i> (0x00) ... vorwärts Bewegung <i>BACKWARD</i> (0x01) ... rückwärts Bewegung
	ucSpeedRight	Geschwindigkeit (Inkremente pro 2 ms)
	ucDirectionLeft	<i>FORWARD</i> (0x00) ... vorwärts Bewegung <i>BACKWARD</i> (0x01) ... rückwärts Bewegung
	ucSpeedLeft	Geschwindigkeit (Inkremente pro 2 ms)
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.27 Befehl zum Setzen von individuellen Geschwindigkeiten und Richtungen

<i>Funktionsname</i>	<i>MotorSetIndividualOne</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucMotorNbr	Motornummer (1 bzw. 2)
	ucSpeedRight	Geschwindigkeit (Inkremente pro 2 ms)
	ucDirectionLeft	<i>FORWARD</i> (0x00) ... vorwärts Bewegung <i>BACKWARD</i> (0x01) ... rückwärts Bewegung
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.28 Befehl zum Setzen einer individuellen Geschwindigkeiten und Richtungen

<i>Funktionsname</i>	<i>MotorDriveOne</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucMotorNbr	Motornummer (1 bzw. 2)
	ucDirectionLeft	<i>FORWARD</i> (0x00) ... vorwärts Bewegung <i>BACKWARD</i> (0x01) ... rückwärts Bewegung
	ucSpeedRight	Geschwindigkeit (Inkrement pro 2 ms)
	ucDistance	Weg in Inkremente
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.29 Befehl zum Setzen einer individuellen Geschwindigkeiten und Richtungen

<i>Funktionsname</i>	<i>ucMotorStatus</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
<i>Rückgabewerte</i>	Bewegungsstatus	
	<i>MOTION_UNDER_PROCESS</i> (0x00) ... Bewegung wird noch ausgeführt	
	<i>MOTION_READY</i> (0x01) ... Bewegung ist fertig	
	<i>MOTION_ERROR</i> (0x02) ... Fehler ist aufgetreten	
	puiDistanceLeft	Inkrement des linken Motors
	puiDistanceRight	Inkrement des rechten Motors

Tabelle 3.30 Befehl zum Auslesen der relativen Bewegung und des Bewegungsstatus'

<i>Funktionsname</i>	<i>uiGetPosition</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
<i>Rückgabewerte</i>	Orientierung	
	Absoluter Winkel des Roboters in 0,1°	
	puiXPosition	Absolute X-Koordinate des Roboters
	puiYPosition	Absolute Y-Koordinate des Roboters

Tabelle 3.31 Befehl zum Auslesen der Orientierung und der Position

<i>Funktionsname</i>	<i>SetPosition</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	uiAngle	Absoluter Winkel des Roboters in 0,1°
	uiXPosition	Absolute X-Position des Roboters in mm
	uiYPosition	Absolute Y-Position des Roboters in mm
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.32 Befehl zum Setzen der Orientierung und der Position

<i>Funktionsname</i>	<i>MotorStop</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.33 Befehl zum Stoppen der Motoren

<i>Funktionsname</i>	<i>MotorEmergencyStop</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.34 Befehl zum kontrollierten gegen Null Fahren der Motoren

<i>Funktionsname</i>	<i>StepperDrive</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	ucDirection	<i>FORWARD</i> (0x00) ... vorwärts fahren <i>BACKWARD</i> (0x01) ... rückwärts fahren
	ucSpeed	Geschwindigkeit (Inkrement pro 1 ms)
	ucDistance	Schritte des Motors
<i>Rückgabewerte</i>	kein Rückgabewert	

Tabelle 3.35 Befehl zum Vor- bzw. Rückwärtsfahren eines Schrittmotors

<i>Funktionsname</i>	<i>ucLCDGetStrategy</i>	
<i>Übergabewerte</i>	ucChannel	Adresse am SPI-Bus (1 ... 14)
	pucPlayColour	Spielfarbe
<i>Rückgabewerte</i>	Strategie	
	Eventuell eine Spielfarbe, falls diese am Display manuell verändert wurde	

Tabelle 3.36 Befehl zum Anfordern der Strategie und ev. Spielfarbe

4. MOTOR-BOARD

4.1. HARDWARE

Das Motor-Board stellt die Leistungsansteuerung der Motoren zur Verfügung. Mit dem Motor-Board können entweder bis zu zwei DC-Motoren oder ein Schrittmotor angesteuert werden. Die beiden DC-Motoren können dabei gemeinsam, z.B. für eine Fahreinheit, oder individuell angesteuert werden.

Die komplette Motorregelung und Positionsberechnung wird auf dem Motor-Board abgewickelt. Es können Motoren mit Spannungen bis zu 35 V und einer Leistung von 100 W angeschlossen werden.

Auf dem Motor-Board befinden sich die in Abbildung 4.1 dargestellten Schnittstellen, dazu gehören

- Versorgung (K4 siehe Tabelle 4.4)
- Programmierung (K1 siehe Tabelle 4.1)
- Kommunikation mit dem Master via SPI (K3 siehe Tabelle 4.3)
- digitale/analoge I/Os (K7 siehe Tabelle 4.6)
- zwei Motoranschlüsse mit Impulsgeberauswertung (KL1 und K5 bzw. KL2 und K6 siehe Tabelle 4.5 und Tabelle 4.7)

Für die Versorgung der Elektronik und der Peripherie mit 5V sorgt der Schaltregler LM2594 der Fa. National Semiconductor. Mit diesem Schaltregler ist ein maximaler Ausgangsstrom von 500 mA möglich.

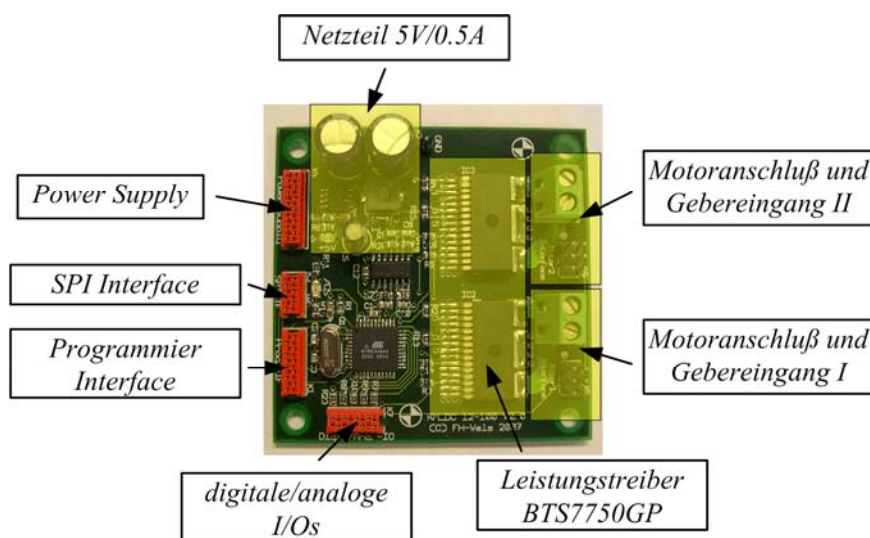


Abbildung 4.1 Bestückte Leiterplatte des DC-Motorboards

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	Pluspol der Versorgung für das Programmier-Board
2	RxD	Empfangsleitung der Debug-Schnittstelle
4	SCK	Clock-Leitung der Programmierschnittstelle
5	TxD	Sendeleitung der Debug-Schnittstelle
6	MISO	Master In Slave Out der Programmier-Schnittstelle
8	MOSI	Master Out Slave In der Programmier-Schnittstelle
9	GND	Minuspole der Versorgung für das Programmier-Board
10	/Reset	Reset-Leitung der Programmier-Schnittstelle
3, 7	NC	nicht verwendet

Tabelle 4.1 Steckerbelegung von K1 (Prog.-IF)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	GND	Massestift für Messungen

Tabelle 4.2 Steckerbelegung von K2 (Massestift)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	/CS	Chip-Select des Slaves
2	SCK	Synchroner Clock
3	MISO	Master In - Slave Out zur Datenübertragung zum Master
4	MOSI	Master Out - Slave In zur Datenübertragung zum Slave
5,6	GND	GND der Schnittstelle

Tabelle 4.3 Steckerbelegung für K3 (SPI-IF)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1,3,5,7,9,11	+12V	Pluspol der Versorgung
2,4,6,8,10,12	GND	Minuspole der Versorgung

Tabelle 4.4 Steckerbelegung von K4 (Power-Supply)

Die Schnittstelle für den Motoranschluss und den Impulsgeber wurde von der Fa. Faulhaber¹² übernommen. Diese Steckerbelegung ist für alle DC-Kleinstmotoren der Fa. Faulhaber gleich.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	Motor -	Minusanschluss des Motors
2	Motor +	Plusanschluss des Motors
3	GND	Minuspol der Geber-Versorgung
4	+5V	Pluspol der Geber-Versorgung
5	INCB	Gebereingang B-Kanal
6	INCA	Gebereingang A-Kanal

Tabelle 4.5 Steckerbelegung für K5 und K6 (Motor-IF)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	+5V-Versorgung für das Funk- und RS232 Modul
2	I/O1	digitaler Eingang/Ausgang
3	+12V	+12V-Versorgung
4	I/O2	digitaler Eingang/Ausgang
5	GND	Minuspol der Versorgung
6	I/O3	digitaler Eingang/Ausgang
7	TxD	Sendeleitung der seriellen Schnittstelle
8	RxD	Empfangsleitung der seriellen Schnittstelle

Tabelle 4.6 Steckerbelegung für K7 (I/O-IF)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	Motor +	Minusanschluss des Motors
2	Motor -	Plusanschluss des Motors

Tabelle 4.7 Steckerbelegung von KL1 und KL2 (Motoranschluss)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	+5V-Versorgung des Ausgangs
2	SWITCH	Schaltausgang gegen GND

Tabelle 4.8 Steckerbelegung von KL3 (geschalteter Ausgang)

¹² [Faulhaber, 2006]

BTS7750GP¹³

Der BTS7750GP Chip ist Teil der TrilithIC-Familie der Fa. Infineon. Dieser Chip kann als H-Brücke zur Motoransteuerung verwendet werden und weist unter anderem folgende Features auf:

- Frei konfigurierbar als H-Brücke, Halbbrücke oder vier individuelle Schalter
- Für DC-Motor Ansteuerungen optimiert
- Low $R_{DS,ON}$: 70m Ω high-side switch, 45m Ω low-side switch
- Max. Spitzenstrom von 12 A
- Kleiner Sperrstrom von 5 μ A
- Kurzschlusssicher
- Übertemperaturabschaltung
- Interne Klemmdioden
- Bis zu 1 kHz PWM-Frequenz

In Abbildung 4.2 ist das Blockdiagramm des BTS7750GP dargestellt. Daraus kann die Anschlussbelegung des ICs entnommen werden.

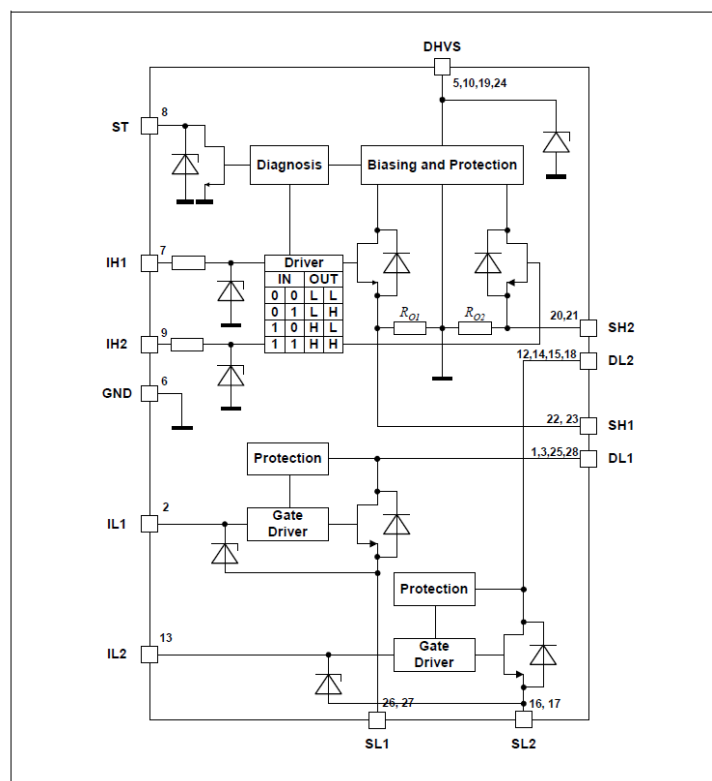


Abbildung 4.2 Blockdiagramm des BTS7750GP

¹³ [Infineon, 2008]

4.2. SOFTWARE

4.2.1. Treiber

Wie in Tabelle 4.9 dargestellt, ist die Software des Motor-Boards in 14 Module aufgeteilt. Es wurde darauf geachtet, dass die Module leicht auf andere Projekte bzw. Plattformen portiert werden können.

<i>Modul</i>	<i>Funktion</i>
main.c	Hauptprogramm
extint.c	zur Auswertung der Impulsgeber
timer0.c	im Timer 0 Interrupt wird der Drehzahlregler abgearbeitet
ports.c	stellt die Treiber/Makros für den Zugriff auf die I/Os zur Verfügung
uart.c	stellt Treiber für die serielle Schnittstelle zur Verfügung; über diese Schnittstelle kann auch auf die Steuerung und Sensorik des Roboters zugegriffen werden
adc.c	stellt die Treiber für die A/D-Wandlung zur Verfügung
spi.c	stellt die Treiber für die SPI-Schnittstelle zur Verfügung
acomp.c	stellt die Treiber für den Analog-Komparator zur Verfügung
timer1.c	stellt die PWM-Einheit für den 1. Motor zur Verfügung
timer2.c	stellt die PWM-Einheit für den 2. Motor zur Verfügung
motor.c	stellt alle Treiber zur Ansteuerung der DC-Motoren zur Verfügung
schrittmotor.c	stellt alle Treiber zur Ansteuerung den Schrittmotoren zur Verfügung
convert.c	hier werden die Weg- und Winkelwerte in Inkremente umgerechnet und umgekehrt
position.c	zur Berechnung der aktuellen Position des Roboters

Tabelle 4.9 Softwaremodule des Motor-Boards

Hauptprogramm

Nach der Initialisierung des Microcontrollers werden in der Main-Loop die vom Master gesendeten Befehle abgearbeitet. Wie in Abbildung 4.3 ersichtlich ist, wird solange der Master das Motor-Board nicht initialisiert hat, auf die Initialisierung gewartet.

Nach der Initialisierung, zur Ansteuerung von DC-Motoren oder eines Schrittmotors, werden die Motorbefehle abgearbeitet.

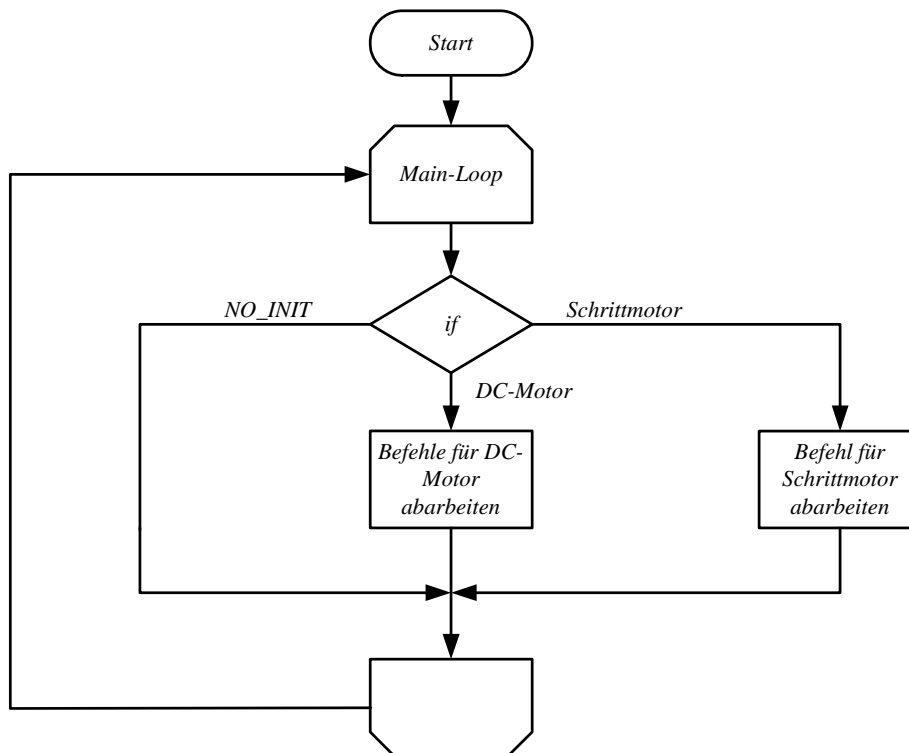


Abbildung 4.3 Flussdiagramm Hauptprogramm Motor-Board

Drehzahlregler

Wie in Abbildung 4.4 dargestellt, handelt es sich bei dem Drehzahlregler um einen einfachen P-Regler. Diese Art von Regler hat den Nachteil eine bleibende Regelabweichung zu haben, dieser Umstand ist für den Einsatz im Roboter jedoch zweitrangig, da es nicht unbedingt notwendig ist mit einer exakten Geschwindigkeit zu fahren. Viel wichtiger ist es das die Relativgeschwindigkeiten der Antriebsräder zueinander stimmen und eine vorgegebene Position exakt angefahren werden kann. Diese beiden Aufgaben werden jedoch nicht vom Geschwindigkeitsregler sondern vom Positionsregler übernommen. Dieser Regler wird weiter unten näher beschrieben.

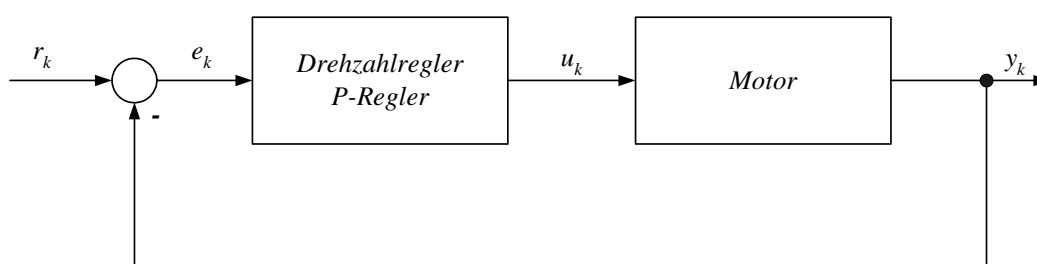


Abbildung 4.4 Blockschaltbild Drehzahlregler

Der Geschwindigkeitsregler wird im Timer 0 des Microcontrollers in einem 5 ms Intervall abgearbeitet. Mittels des Unterprogramms *MotorRegler(ucKpTimer0)* werden die Geschwindigkeiten beider Motoren geregelt, wobei mit der Variable *ucKpTimer0* der Verstärkungsfaktor K_P des Reglers übergeben wird.

Positionsregler

Beim Positionsregler kann unter zwei unterschiedlichen Aufgaben unterschieden werden, nämlich zwischen einer Linearbewegung um eine vorgegebene Strecke und einer Rotationsbewegung um die eigene Achse um einen bestimmten Winkel. Diese beiden Aufgaben werden mittels der beiden Unterprogramme $ucDrive(ucDirection, uiSteps, ucSpeed)$ und $ucTurn(ucDirection, uiSteps, ucSpeed)$ realisiert. Als Übergabeparameter werden die Richtung, *FORWARD* oder *BACKWARD*, für die Linearbewegung und *LEFT* oder *RIGHT* für die Drehbewegung, die Anzahl der Inkremente und die Geschwindigkeit übergeben. Als Rückgabewert liefern die beiden Programme den Status der Bewegung, *MOTION_UNDER_PROCESS* bzw. *MOTION_READY*, zurück.

Die Struktur des Positionsreglers ist in Abbildung 4.5 dargestellt. Dabei ist zu erkennen, dass dem Positionsregler eine Soll-Position vorgegeben wird und der Regler diese mit den Ist-Positionen vergleicht. Aus diesen Werten werden die Geschwindigkeitssollwerte für die beiden Motoren berechnet und diese an den Geschwindigkeitsregler weitergegeben. Eine weitere Aufgabe des Positionsreglers ist es, dass die Räder im Gleichlauf angesteuert werden, d.h. dass nicht ein Rad eine größere Strecke absolviert als das andere Rad. Dadurch würde aus der Linearbewegung ein Kreisbogen werden.

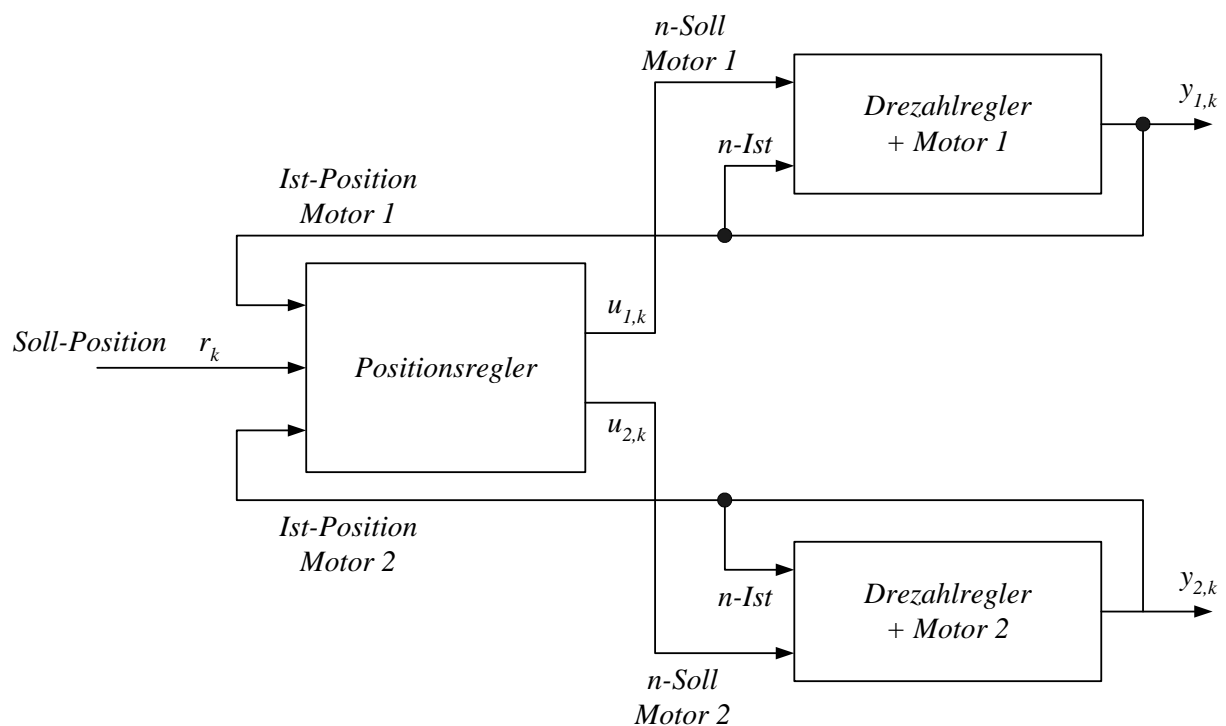


Abbildung 4.5 Blockschaltbild Positionsregler

Um die Bewegungen des Roboters fließend zu gestalten, ist es notwendig die Motoren des Roboters, wie in Abbildung 4.6 dargestellt, anzusteuern. Dabei werden die Motoren in einer Up-Slope-Zeit T_{US} von einer Minimalgeschwindigkeit v_{min} auf die Endgeschwindigkeit v_{max} in einer Rampe hochgefahren. Anschließend werden die Motoren auf dieser Geschwindigkeit gehalten (T_{max}) und zum Abschluss wird die Geschwindigkeit in der Down-Slope-Phase T_{DS} wieder heruntergefahren.

Die Up- bzw. Down-Slope-Zeit erstreckt sich für Distanzen unter 3000 Inkremente auf ein Viertel der Distanz, über 3000 Inkremente werden diese Zeiten auf 750 Inkremente beschränkt.

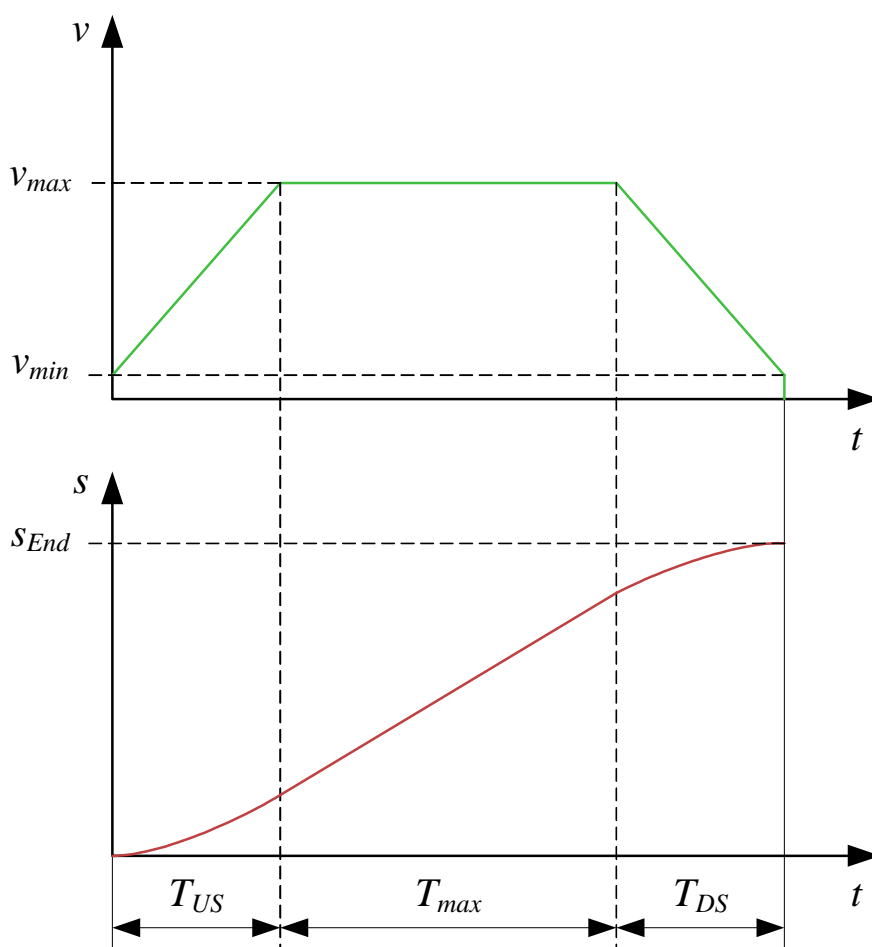


Abbildung 4.6 v-t bzw. s-t Diagramm

Positionsberechnung

Die Position wird mittels der Inkremente zweier Laufräder berechnet. Wie in Abbildung 4.7 dargestellt, misst der Roboter die Inkremente des linken und des rechten Laufrades und berechnet daraus in einem 5 ms-Zyklus den differentiellen Weg ds und den Winkel des Roboters φ_n . Aus diesen beiden Werten kann die absolute Position (x_R , y_R und φ_R) des Roboters berechnet werden.

Für diese Art der Berechnung müssen die Anfangsposition (x_0 , y_0) und der Anfangswinkel (φ_0) bekannt sein, da von diesen Werten ausgehend alle anderen berechnet werden. Die Berechnung der Position und der Orientierung sind in den Gleichungen 4.1 bis 4.5 dargestellt.

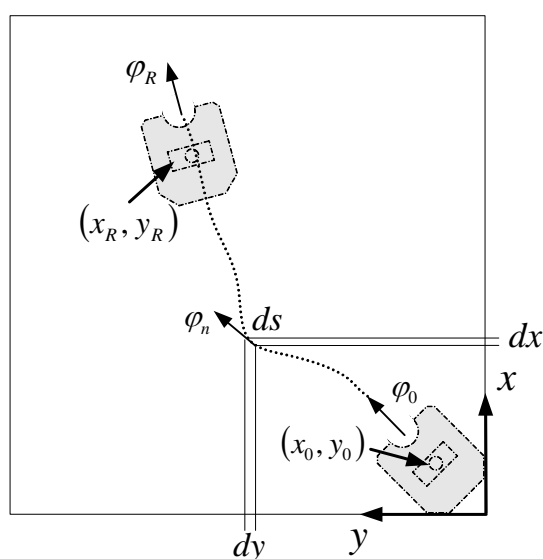


Abbildung 4.7 Schematische Darstellung der Positionsberechnung

$$s_n = \frac{1}{2} \cdot (\text{Inkremente_rechts} + \text{Inkremente_links}) \cdot k_s \quad (4.1)$$

$$\varphi_R = \varphi_0 + (\text{Inkremente_rechts} - \text{Inkremente_links}) \cdot k_D \quad (4.2)$$

$$ds = s_n - s_{n-1} \quad (4.3)$$

$$x_R = x_{R-1} + ds \cdot \cos \varphi_R \quad (4.4)$$

$$y_R = y_{R-1} + ds \cdot \sin \varphi_R \quad (4.5)$$

Schrittmotoransteuerung

Mit dem Motor-Board kann ein Schrittmotor angesteuert werden. Mit dem Unterprogramm *SetStepper(ucDirection, uiSteps, ucSpeed)* wird der Motor angesteuert, wobei der Übergabeparameter *ucDirection* angibt in welche Richtung der Motor laufen soll (0 ... Rechtslauf, 1 ... Linkslauf), *uiSteps* gibt die Anzahl der Schritte an und *ucSpeed* gibt an wie schnell der Motor laufen soll. Als Wert für die Geschwindigkeit wird ein Faktor angegeben, der angibt wie schnell von einer Phase auf die nächste umgeschaltet werden soll und ist ein Vielfaches von einer ms.

Bei Rechtslauf gilt die Phasenfolge $C \rightarrow B \rightarrow D \rightarrow A$, wobei A der Plusausgang von Motor 1, B der Minusausgang von Motor 1, C der Plusausgang von Motor 2 und D der Minusausgang von Motor D ist. Für Linkslauf gilt die Phasenfolge $A \rightarrow D \rightarrow B \rightarrow C$.

4.2.2. SPI Meldungen

Init-Meldung

Bei der Meldung zum Initialisieren des Motor-Boards (siehe Abbildung 4.8) wird als erstes Byte die Meldungskennung (*MB_INIT_MSG*) übertragen und danach wird der Motortype zum Motor-Board geschickt (siehe Tabelle 4.10).

Meldungsformat Master → Motorboard:

1. Byte

2. Byte

<i>Meldungsnummer (MB_INIT_MSG)</i>	<i>Motortype</i>
---	------------------

Abbildung 4.8 Meldung MB_INIT_MSG

<i>Meldung</i>	<i>Wert</i>	<i>Motorart</i>	<i>Anzahl der Motoren</i>
<i>MB_NOT_INIT</i>	0x00	-	-
<i>MB_DC_MOTOR</i>	0x01	DC-Motor	2
<i>MB_STEPPER</i>	0x02	Schrittmotor	1

Tabelle 4.10 Unterstützte Motortypen

Stopp-Meldungen

Bei den Meldungen zum stoppen der Motoren wird lediglich die Meldungsnummer übertragen (siehe Abbildung 4.9 und Abbildung 4.10). Der Unterschied zwischen den beiden Meldungen besteht darin, dass bei der *MB_STOP_MSG* die Motoren abrupt gestoppt werden und bei der *MB_EMERGENCY_STOP_MSG* die Motoren kontrolliert gegen Null gefahren werden.

Die *MB_EMERGENCY_STOP_MSG* sollte für Motoren verwendet werden, die ein großes Trägheitsmoment beaufschlagt haben.

Meldungsformat Master → Motorboard:

1. Byte

<i>Meldungsnummer (MB_STOP_MSG)</i>

Abbildung 4.9 Meldung MB_STOP_MSG

Meldungsformat Master → Motorboard:

1. Byte

Meldungsnummer (MB_EMERGENCY_STOP_MSG)

Abbildung 4.10 Meldung MB_EMERGENCY_STOP_MSG

Linearbewegungs-Meldungen

Bei den Meldungen zur Ausführung einer Linearbewegung werden zuerst die Meldungsnummer, dann die Geschwindigkeit und zum Schluss zwei Bytes für die zu fahrende Strecke übertragen (siehe Abbildung 4.11 und Abbildung 4.12).

Als Distanz wird der Weg in mm übergeben, dieser wird auf dem Motor-Board in Inkremente umgerechnet.

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte
Meldungsnummer (MB_FORWARD_MSG)	Speed	Distance (High-Byte)	Distance (Low-Byte)

Abbildung 4.11 Meldung MB_FORWARD_MSG

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte
Meldungsnummer (MB_BACKWARD_MSG)	Speed	Distance (High-Byte)	Distance (Low-Byte)

Abbildung 4.12 Meldung MB_BACKWARD_MSG

Rotationsbewegungs-Meldungen

Bei den Meldungen zur Ausführung einer Rotationsbewegung um die eigene Achse wird zuerst die Meldungsnummer, dann die Geschwindigkeit und zum Schluss zwei Bytes für den zu drehenden Winkel übertragen (siehe Abbildung 4.13 und Abbildung 4.14).

Der Winkel wird dabei in 0,1°-Schritten übergeben, dieser wird auf dem Motor-Board in Inkremente umgerechnet.

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte
Meldungsnummer (MB_TURN_RIGHT_MSG)	Speed	Distance (High-Byte)	Distance (Low-Byte)

Abbildung 4.13 Meldung MB_TURN_RIGHT_MSG

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte
Meldungsnummer (MB_TURN_LEFT_MSG)	Speed	Distance (High-Byte)	Distance (Low-Byte)

Abbildung 4.14 Meldung MB_TURN_LEFT_MSG

Meldung zum individuellen Ansteuern der Motoren

Um die Motoren unabhängig voneinander anzusteuern, kann mit der Meldung MB_SET_MOTORS_INDIVIDUAL_MSG auf die Motoren zugegriffen werden. Dabei werden, wie in Abbildung 4.15 dargestellt, zuerst die Meldungsnummer, dann die Richtung und die Geschwindigkeit des ersten Motors und abschließend die Richtung und die Geschwindigkeit des zweiten Motors übertragen.

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte	5. Byte
Meldungsnummer (MB_SET_MOTORS_INDIVIDUAL_MSG)	Direction Motor 1	Speed Motor 1	Direction Motor 2	Speed Motor 2

Abbildung 4.15 Meldung MB_SET_MOTORS_INDIVIDUAL_MSG

Meldungen zum Ansteuern einzelner Motoren

Um nur einen Motor anzusteuern, dienen die Meldungen *MB_DRIVE_MOTORS_ONE_MSG* und *MB_SET_MOTORS_INDIVIDUAL_ONE_MSG*.

Wobei mit der ersten Meldung die Position des Motors um einen bestimmten Wert verändert wird, also die Positionsregelung im Vordergrund steht. Dabei werden, wie in Abbildung 4.16 dargestellt, zuerst die Meldungsnummer, dann die Motornummer, anschließend die Richtung und die Geschwindigkeit und zum Abschluss die zu absolvierende Distanz übertragen. Die Distanz wird in Inkrementen übergeben.

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte
Meldungsnummer (<i>MB_DRIVE_MOTORS_ONE_MSG</i>)	Motor-Nbr.	Direction	Speed	Distance (High-Byte)	Distance (Low-Byte)

Abbildung 4.16 Meldung *MB_DRIVE_MOTORS_ONE_MSG*

Mit der zweiten Meldung wird der Motor mit einer bestimmten Geschwindigkeit angesteuert, also steht die Geschwindigkeitsregelung im Vordergrund. Es werden nach der Meldungsnummer die Geschwindigkeit und zum Abschluss die Richtung übertragen (siehe Abbildung 4.17).

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte
Meldungsnummer (<i>MB_SET_MOTORS_INDIVIDUAL_ONE_MSG</i>)	Motor-Nbr.	Speed	Direction

Abbildung 4.17 Meldung *MB_SET_MOTORS_INDIVIDUAL_ONE_MSG*

Meldungen zum Auslesen und Setzen der Positionen

Da das Motor-Board die Positionsberechnung des Roboters zur Aufgabe hat, kann mit den Meldungen *MB_GET_POSITION_MSG* und *MB_SET_POSITION_MSG* die aktuelle Position ausgelesen bzw. gesetzt werden.

Beim Auslesen der Position wird zuerst die Meldungsnummer und anschließend sechs Leerbytes übertragen (siehe Abbildung 4.18). Während dieser sechs Bytes liest der Master die Positionsdaten ein. Zuerst werden zwei Byte für die Orientierung (in $0,1^\circ$), dann die X-Position (in mm) und zum Abschluss die Y-Position (in mm) übertragen.

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte	7. Byte
Meldungsnummer (<i>MB_GET_POSITION_MSG</i>)	Leerbyte	Leerbyte	Leerbyte	Leerbyte	Leerbyte	Leerbyte

Meldungsformat Motorboard → Master:

1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte	7. Byte
Leerbyte	Winkel [$0,1^\circ$] (High-Byte)	Winkel [$0,1^\circ$] (Low-Byte)	X-Position [mm] (High-Byte)	X-Position [mm] (Low-Byte)	Y-Position [mm] (High-Byte)	Y-Position [mm] (Low-Byte)

Abbildung 4.18 Meldung *MB_GET_POSITON_MSG*

Beim Setzen der Position wird zuerst die Meldungsnummer, dann zwei Byte für die Orientierung (in $0,1^\circ$), anschließend zwei Byte für die X-Position (in mm) und zum Abschluss zwei Byte für die Y-Position (in mm) übertragen (siehe Abbildung 4.19).

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte	7. Byte
Meldungsnummer (<i>MB_SET_POSITION_MSG</i>)	Winkel [$0,1^\circ$] (High-Byte)	Winkel [$0,1^\circ$] (Low-Byte)	X-Position [mm] (High-Byte)	X-Position [mm] (Low-Byte)	Y-Position [mm] (High-Byte)	Y-Position [mm] (Low-Byte)

Abbildung 4.19 Meldung *MB_SET_POSITION_MSG*

Meldung zum Auslesen des Motor-Status

Mit der Meldung *MB_GET_STATUS_MSG* kann der aktuelle Bewegungsstatus, die zurückgelegten Inkremente des ersten und des zweiten Motors ausgelesen werden. Dieser Inkrementwert wird bei jedem neuen Fahrbefehl auf Null gesetzt.

Beim Auslesen des Status wird zuerst die Meldungsnummer und anschließend fünf Leerbytes übertragen (siehe Abbildung 4.20). Während dieser fünf Bytes liebt der Master den Bewegungsstatus (*MOTION_UNDER_PROCESS* bzw. *MOTION_READY*) und die Inkrementwerte aus.

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte
Meldungsnummer (<i>MB_GET_STATUS_MSG</i>)	Leerbyte	Leerbyte	Leerbyte	Leerbyte	Leerbyte

Meldungsformat Motorboard → Master:

1. Byte	2. Byte	3. Byte	4. Byte	5. Byte	6. Byte
Leerbyte	Motion-Status	Inkr. Motor 1 (High-Byte)	Inkr. Motor 1 (Low-Byte)	Inkr. Motor 2 (High-Byte)	Inkr. Motor 2 (Low-Byte)

Abbildung 4.20 Meldung *MB_GET_STATUS_MSG*

Schrittmotor-Meldungen

Bei den Meldungen zur Ausführung einer Schrittmotorbewegung werden zuerst die Meldungsnummer, dann die Geschwindigkeit und zum Schluss zwei Bytes für die zu fahrende Strecke übertragen (siehe Abbildung 4.21 und Abbildung 4.22).

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte
Meldungsnummer (<i>MB_SET_STEPPER_FORWARD_MSG</i>)	Wait-States [ms]	Distance (High-Byte)	Distance (Low-Byte)

Abbildung 4.21 Meldung *MB_SET_STEPPER_FORWARD_MSG*

Meldungsformat Master → Motorboard:

1. Byte	2. Byte	3. Byte	4. Byte
Meldungsnummer (<i>MB_SET_STEPPER_BACKWARD_MSG</i>)	Wait-States [ms]	Distance (High-Byte)	Distance (Low-Byte)

Abbildung 4.22 Meldung *MB_SET_STEPPER_BACKWARD_MSG*

5. SENSOR/SERVO-BOARD

5.1. HARDWARE

Da die Anbindung von Sensoren und Servos eine ähnliche Schnittstelle haben, wurde das Sensor- und Servo-Board zu einem Board vereinheitlicht. In der ersten Version des modularen Systems waren diese Boards noch getrennt ausgeführt.

An dem Sensor/Servo Board können bis zu 16 Sensoren bzw. Servos angeschlossen werden. Diese maximale Anschlusszahl hängt vom Typen des Sensors ab, denn der Sensor SRF08/10 benötigen eine I²C-Schnittstelle, und für diese Schnittstelle werden zwei Ports benötigt. Die maximale Anschlusszahl verringert sich in diesem Fall auf 8 Sensoren.

Das Servo/Sensor-Board ist, wie aus Abbildung 5.1, mit den Schnittstellen für

- Versorgung (K1 siehe Tabelle 5.1)
- Programmierung (K2 siehe Tabelle 5.2)
- Kommunikation mit dem Master via SPI (K22 siehe Tabelle 5.4)

ausgestattet.

Für die Versorgung der Elektronik und der Peripherie mit 5 V sorgt der Schaltregler LM2677 der Fa. National Semiconductor. Mit diesem Schaltregler ist ein maximaler Ausgangsstrom von 5 A möglich.

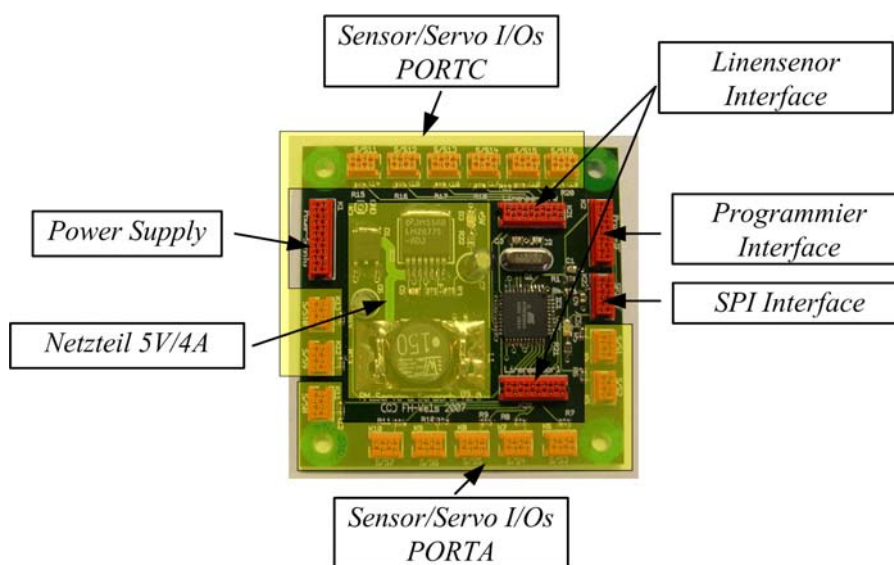


Abbildung 5.1 Bestückte Leiterplatte des Sensor/Servoboards

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1,3,5,7,9,11	+12V	Pluspol der Versorgung
2,4,6,8,10,12	GND	Minuspole der Versorgung

Tabelle 5.1 Steckerbelegung von K1 (Power-Supply)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	Pluspol der Versorgung für das Programmier-Board
2	RxD	Empfangsleitung der Debug-Schnittstelle
4	SCK	Clock-Leitung der Programmierschnittstelle
5	TxD	Sendeleitung der Debug-Schnittstelle
6	MISO	Master In Slave Out der Programmier-Schnittstelle
8	MOSI	Master Out Slave In der Programmier-Schnittstelle
9	GND	Minuspole der Versorgung für das Programmier-Board
10	/Reset	Reset-Leitung der Programmier-Schnittstelle
3, 7	NC	nicht verwendet

Tabelle 5.2 Steckerbelegung von K2 (Prog.-IF)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	GND	Massestift für Messungen

Tabelle 5.3 Steckerbelegung von K3 (Massestift)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	/CS	Chip-Select des Slaves
2	SCK	Synchroner Clock
3	MISO	Master In - Slave Out zur Datenübertragung zum Master
4	MOSI	Master Out - Slave In zur Datenübertragung zum Slave
5,6	GND	GND der Schnittstelle

Tabelle 5.4 Steckerbelegung für K22 (SPI-IF)

Wie in Tabelle 5.5 und Tabelle 5.6 näher beschrieben, stehen für die Anbindung der Sensoren bzw. Servos 16 Anschlüsse, K4 bis K19, zur Verfügung. Diese Anschlüsse sind mit jeweils

- 5 V
- 12 V
- GND
- Signalleitung

ausgestattet.

Dabei werden die Anschlüsse K4 bis K11 auf PORTA des Microcontrollers angeschlossen und können somit wahlweise als digitale oder analoge Eingänge verwendet werden. Die Anschlüsse K12 bis K19 liegen auf PORTC und können nur als digitale Eingänge verwendet werden.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	+5V-Versorgung für den Sensor/Servo
2	+12V	+12V-Versorgung für den Sensor/Servo
3	GND	GND für den Sensor/Servo
4	Sensor Eingang	analoger Eingang, digitaler Eingang/Ausgang

Tabelle 5.5 Steckerbelegung von K4 bis K11 (S/S 1...8 – PORTA)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	+5V-Versorgung für den Sensor/Servo
2	+12V	+12V-Versorgung für den Sensor/Servo
3	GND	GND für den Sensor/Servo
4	Sensor Eingang	digitaler Eingang/Ausgang

Tabelle 5.6 Steckerbelegung von K12 bis K19 (S/S 9...16 – PORTC)

Auf den Steckern K20 und K21 (siehe Tabelle 5.7) wurden der PORTA und PORTC auf je einen Stecker ausgeführt. An diesen Anschlüssen kann zum Beispiel ein Linien-Board mit 8 Reflexlichtschranken angeschlossen werden.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	+5V-Versorgung für das Linien-Board
2, 3, 4, 5, 6, 7, 8, 10	+12V	Eingänge (PORTA bzw. PORTC)
9	GND	GND für das Linien-Board

Tabelle 5.7 Steckerbelegung für K20 und K21 (LinenSensor1/2)

5.2. SOFTWARE

5.2.1. Treiber Sensor-Board

Bei der Erstellung der Software wurde darauf geachtet, dass die einzelnen Module so flexibel wie möglich auch in anderen Projekten eingesetzt werden können.

Wie in Tabelle 5.8 dargestellt ist die Software in 13 Module aufgeteilt. Dabei kann, in Module die direkt auf die Hardware des Microcontrollers zugreift (Hardware Abstraction Layer), und in Module, die unabhängig von der Hardware funktionieren, unterschieden werden.

<i>Modul</i>	<i>Funktion</i>
RM_Sensorboard_main.c	Hauptprogramm
uart.c	stellt Treiber für die serielle Schnittstelle, für z.B. eine Debug-Schnittstelle, zur Verfügung
adc.c	stellt Treiber für den AD-Wandler zur Verfügung
ports.c	stellt die Treiber/Makros für den Zugriff auf die I/Os zur Verfügung
spi.c	stellt die Treiber für die SPI-Schnittstelle zur Verfügung; hier läuft die Kommunikation mit dem Master-Modul
acomp.c	stellt die Treiber für den Analog-Komparator zur Verfügung
gp2d12.c	stellt die Treiber für den Sensor GP2D12 zur Verfügung; Ausgabewert des Sensors in mm
gp2d120.c	stellt die Treiber für den Sensor GP2D120 zur Verfügung; Ausgabewert des Sensors in mm
axe045.c	stellt die Treiber für den Sensor AXE045 zur Verfügung
filter.c	stellt die Treiber für das digitale Filter 1. Ordnung zur Verfügung
gp2y0a02f.c	stellt die Treiber für den Sensor GP2Y0A02F zur Verfügung; Ausgabewert des Sensors in mm
i2c_mike.c	stellt die Treiber für die I ² C-Schnittstelle zur Verfügung
srf08.c	stellt die Treiber für den Sensor SRF08/10 zur Verfügung; Ausgabewert des Sensors in mm

Tabelle 5.8 Softwaremodule des Sensor-Boards

Module mit direktem Zugriff auf die Hardware

- Treiber für die UART (uart.c)
- Treiber für den ADC (adc.c)
- Treiber für den Port-Zugriff (port.c) üblicherweise sollen die Deklarationen für die Port- bzw. Pinnamen im Headerfile ports.h getätigt werden
- Treiber für das SPI (spi.c)
- Treiber für den Analogkomparator (acomp.c)

Module ohne direktem Hardwarezugriff

- Treiber für die verschiedenen Sensoren (gp2d12.c, gp2d120.c, axe045.c, srf08.c und gp2y0a02f.c)
- Digitaler Filter (filter.c)
- Implementierung des I²C-Protokolls (i2c_mike.c)
- Hauptprogramm (RM_Sensorboard_main.c)

Hauptprogramm

Wie in Abbildung 5.2 dargestellt ist, wird im Hauptprogramm (siehe Listing 13.4) zuerst die Hardware des Microcontrollers initialisiert (Funktion *InitDevice()*). In der Main-Loop werden alle 16 Sensoren abgearbeitet (Funktion *uiReadSensor(ucSensorIndex)*).

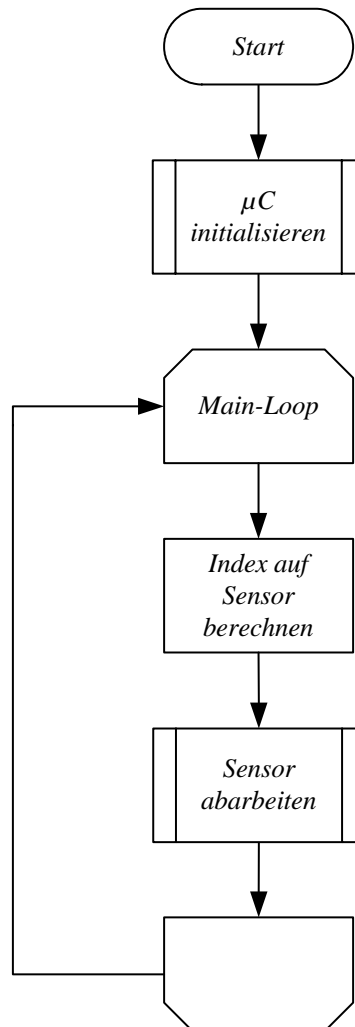


Abbildung 5.2 Flussdiagramm Hauptprogramm

Sensor Abarbeitung

In das Unterprogramm zum Abarbeiten der Sensoren wird ein Index auf den entsprechenden Sensor übergeben. Wie in Abbildung 5.3 dargestellt, wird nach dem Sensortyp gesucht und das jeweilige Programm aufgerufen. Für die Abstandssensoren (GP2D120, GP2D12, GP2Y0A02F und SRF08/10) wird die Entfernung in mm zurückgegeben. Der Farbsensor gibt die jeweilige Farbe als Zahl codiert zurück (z.B. blau → 0). Der Rückgabewerte des digitalen Eingangs ist der Zustand am entsprechenden Pins, also 0 oder 1.

Wenn der ausgewählte Sensor nicht initialisiert ist wird ein Leerlauf-Task aufgerufen.

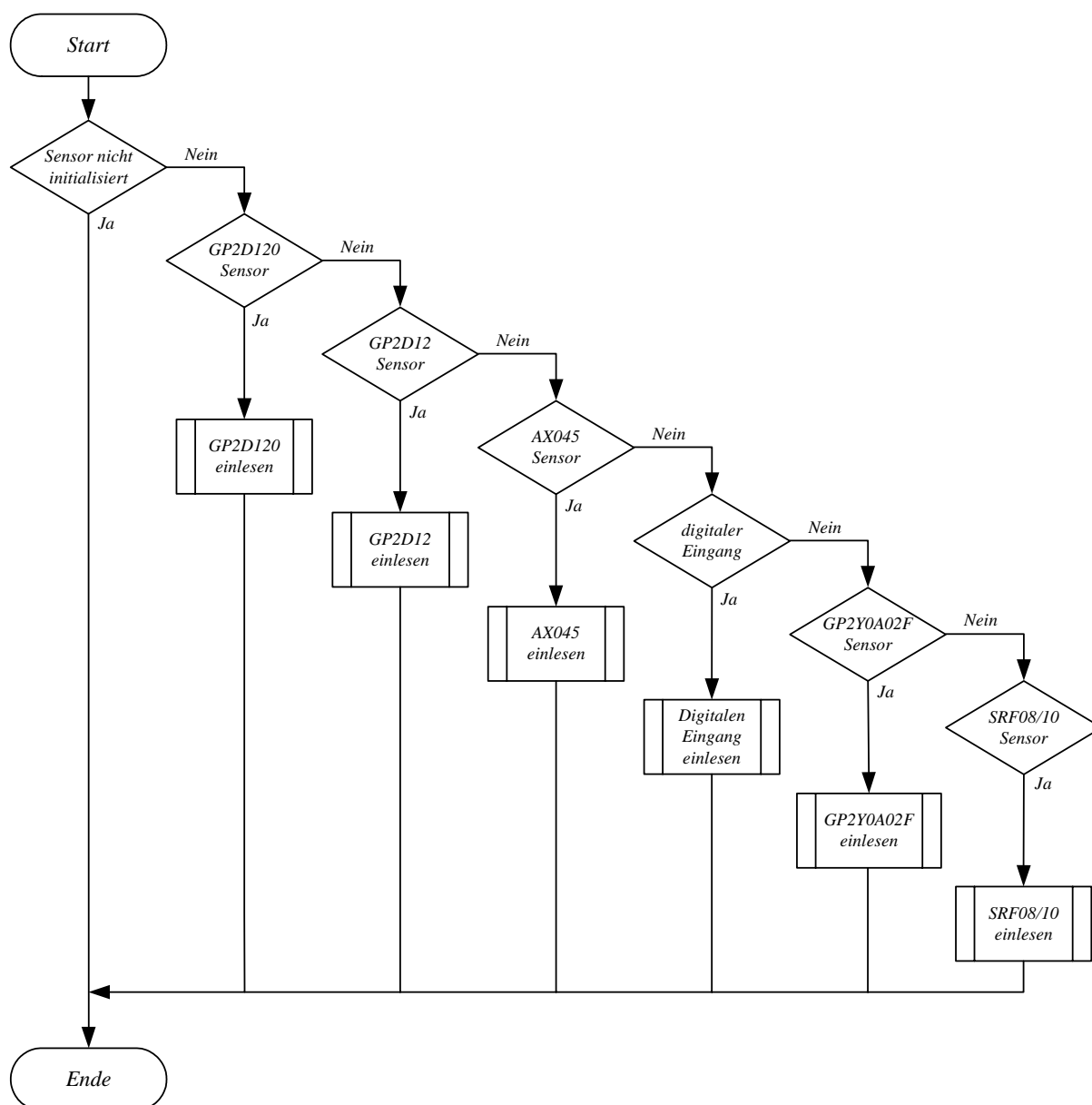


Abbildung 5.3 Flussdiagramm „Sensor einlesen“

GP2D12, GPD120 und GP2Y0A02F

Die Kennlinien der Sensoren GP2D12, GPD120 und GP2Y0A02F verlaufen in einem weiten Bereich nach einer Funktion wie in Gleichung 5.1 beschrieben (siehe Abbildung 5.4. bis Abbildung 5.6)

$$L = \frac{k_i}{V_o} \tag{5.1}$$

Wobei L die Entfernung zum Objekt, V_o die gemessene Spannung am Sensor und k_i ein Sensorspezifischer Faktor sind. In Tabelle 5.9 sind die entsprechenden k_i -Werte der Sensoren aufgelistet.

<i>Sensor</i>	<i>k_i-Wert</i>
<i>GP2D12</i>	63.000
<i>GPD120</i>	26.840
<i>GP2Y0A02F</i>	103.831

Tabelle 5.9 k_i -Werte der jeweiligen Sensoren

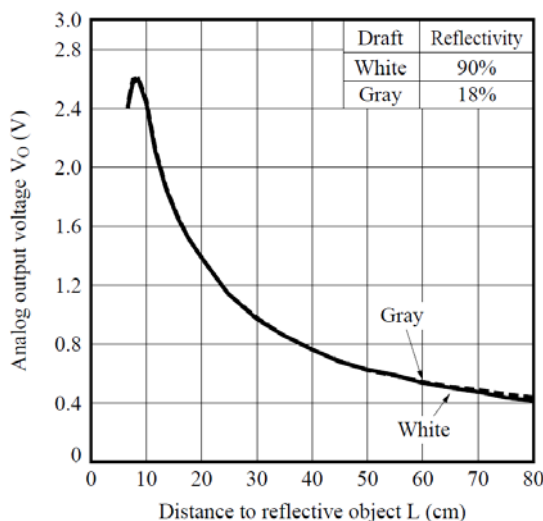


Abbildung 5.4 Analoge Ausgangsspannung vs. Distanz des GP2D12 Sensors¹⁴

¹⁴ [Sharp GP2D12, 2007]

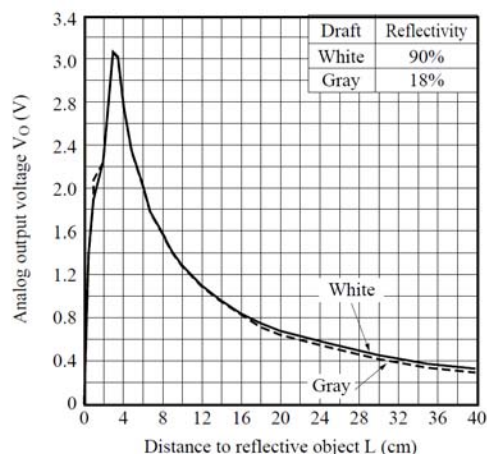


Abbildung 5.5 Analoge Ausgangsspannung vs. Distanz des GP2D120 Sensors¹⁵

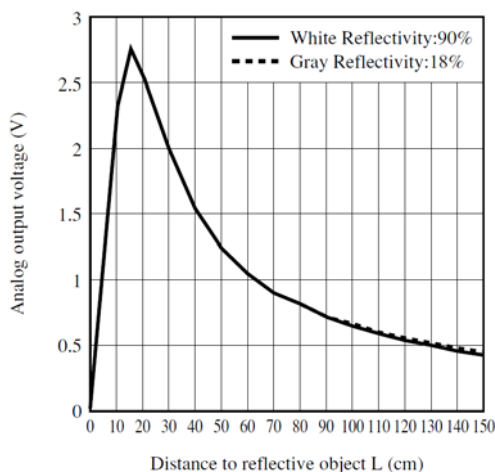


Abbildung 5.6 Analoge Ausgangsspannung vs. Distanz des GP2Y0A02F Sensors¹⁶

Da es in den Randbereichen der Kennlinie zu Fehlmessungen kommen würde, muss der Messbereich eingeschränkt werden (siehe Tabelle 5.10.). Innerhalb der gültigen Bereiche wird der Distanzwert in mm ausgegeben, außerhalb wird der Wert -1 vom Programm zurückgegeben.

<i>Sensor</i>	<i>L [mm]</i>	<i>V_o [V]</i>
<i>GP2D12</i>	100 - 800	0,37 - 2,69
<i>GPD120</i>	40 - 340	0,39 - 2,69
<i>GP2Y0A02F</i>	350 - 1150	0,56 - 1,69

Tabelle 5.10 Messbereiche der Sensoren

¹⁵ [Sharp GP2D120, 2008]

¹⁶ [Sharp GP2Y0A02F, 2007]

I²C-Bus¹⁷, SRF08/10

Der I²C-Bus (Inter Integrated Circuit Bus) wurde von der Fa. Philips entwickelt. Beim I²C-Bus handelt es sich um einen seriellen 3-Draht Bus mit den Anschlüssen SCL (Clock-Leitung, SDA (Daten-Leitung) und einer GND-Leitung.

Über die SCL-Leitung wird vom Master ein Takt vorgegeben und die Daten über die SDA-Leitung zum bzw. vom Slave geschrieben. Übliche Taktfrequenzen können bis zu 400 kHz betragen. Die hier näher beschriebene Sensoren SRF08/10 können mit einer Frequenz von bis zu 40 kHz betrieben werden.

Wenn keine Datenübertragung (siehe Abbildung 5.7) stattfindet, befinden sich die SDA und SCL auf High. Um eine Datenübertragung zu starten, wechselt SDA von High auf Low, SCL verbleibt dabei auf High (Start-Bedingung). Zum Abschluss der Datenübertragung wechselt SDA wieder von Low auf High, wobei SCL High-Pegel aufweist (Stopp-Bedingung). Bei der Datenübertragung darf SDA nur geändert werden, wenn SCL auf Low-Pegel liegt, da ansonsten eine Start- bzw. Stopp-Bedingung erkannt werden wird.

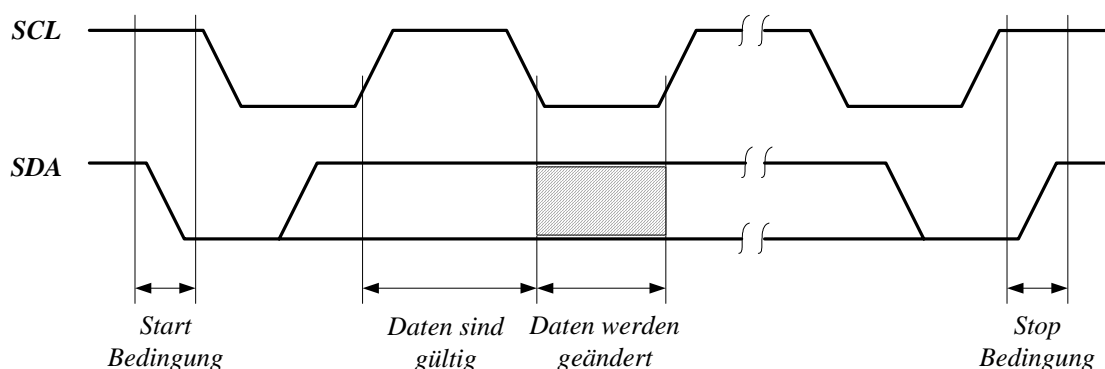


Abbildung 5.7 I²C Bus-Konventionen

In Abbildung 5.8 ist ein Schreibzyklus auf den SRF08/10 Sensor dargestellt. Nach der Start-Bedingung wird ein Byte mit der Sensoradresse übertragen. Dabei wird mit dem LSB (R/W-Bit) die Datenübertragungsrichtung festgelegt. Für einen Schreibzyklus wird das R/W-Bit auf Low-Pegel geschaltet, für eine Lesezyklus auf High-Pegel. Jedes übertragene Byte wird vom Slave mit einem Acknowledge quittiert. Dabei setzt der Master die Datenleitung auf High und der Slave zieht das SDA-Signal auf Low.

¹⁷ [Dembowski, 1997, S.473ff]

Im zweiten Byte wird die Registeradresse übertragen, in die geschrieben werden soll. Zum Abschluss werden die zu schreibenden Daten übertragen.

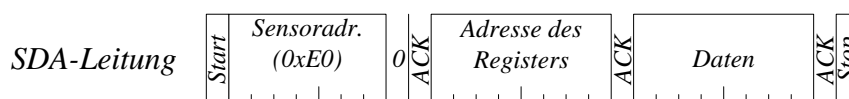


Abbildung 5.8 Schreibezyklus auf den SRF08/10 Sensor

Register	Adresse	Funktion
Befehls-Register	0x00	Abzuarbeitender Befehl des SRF08/10 Moduls 0x50 ... Messung auslösen – Ergebnis in Zoll 0x51 ... Messung auslösen – Ergebnis in cm 0x52 ... Messung auslösen – Ergebnis in μ s
Verstärkungs-Register	0x01	Verstärkung des Messsignals
Reichweiten-Register (R_R)	0x02	Dauer des Messzyklus

Tabelle 5.11 Schreibe-Register des SRF08/10¹⁸

Der SRF08/10-Sensor sendet ein Ultraschallsignal aus und wartet, die im Reichweiten-Register (R_R siehe Tabelle 5.11) eingestellte Zeit, auf ein Echo. Somit kann mit Gleichung 5.2 die maximale Reichweite des Sensors eingestellt werden, wobei s_{max} (in mm) die maximal zu messende Entfernung ist.

$$R_R = \frac{s_{max} - 43mm}{43mm} \quad (5.2)$$

Wenn die maximale Reichweite reduziert wird, sollte auch die Verstärkung nachjustiert werden. Diese kann durch das Schreiben auf das Verstärkungsregister erreicht werden. Die Adresse des Verstärkungsregister ist Tabelle 5.11 zu entnehmen. Der Hersteller gibt keinen direkten Zusammenhang zwischen der Reichweite und der Verstärkung an, da diese auch von der Umgebung des Sensors abhängig ist. Es wird daher empfohlen die Verstärkung solange zu reduzieren, bis ein vernünftiges Messergebnis erreicht wird.

¹⁸ [roboterteile SRF10, 2008]

In Abbildung 5.9 ist ein Lesezyklus auf den SRF08/10 Sensor dargestellt. Nach der Start-Bedingung wird ein Byte mit der Sensoradresse übertragen. Um die Registeradresse, aus dem gelesen werden soll, zu übergeben, wird das R/W-Bit auf Low-Pegel geschaltet. Im zweiten Byte wird diese Registeradresse übertragen.

Anschließend wird wieder mit einer Start-Bedingung fortgefahren und das Byte mit der Sensoradresse übertragen. Für einen Lesezyklus wird nun das R/W-Bit auf High-Pegel geschaltet und anschließend die Daten vom SRF08/10-Modul eingelesen.

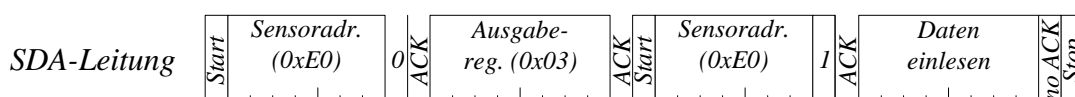


Abbildung 5.9 Vom SRF08/10 Sensor lesen

Da der Messwert in cm zurückgegeben wird, muss bis zu einer Reichweite von 2,55 m nur das Low-Byte (Registeradresse 0x03) ausgelesen werden. Über 2,55 m muss auch das High-Byte (Registeradresse 0x02) ausgelesen werden.

In Abbildung 5.10 ist die Empfindlichkeit in dB über Abstrahlcharakteristik in Grad des SRF08/10 dargestellt. Mit dem Verstärkungs-Register kann die Breite der Messkeule eingeschränkt werden.

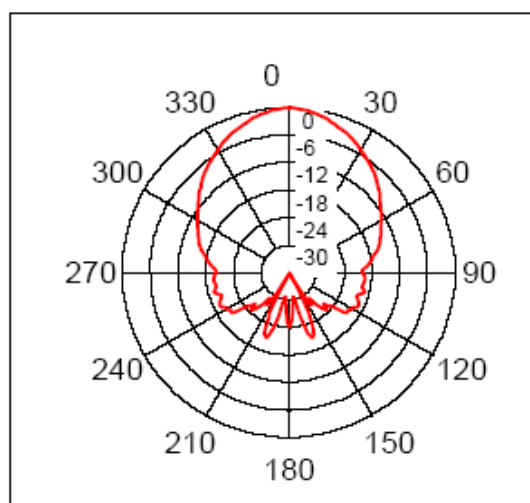


Abbildung 5.10 Abstrahlcharakteristik des Ultraschallsensors SRF08/10¹⁹

¹⁹ [roboterteile SRF10, 2008]

AXE045²⁰

Der AXE045 basiert auf dem TCS230-Chip der Firma TAOS. Das Herzstück dieses Chips ist ein 8x8 Photodioden-Array. Von diesen 64 Photodioden sind 16 Dioden mit einem Blaufilter, 16 Dioden mit einem Rotfilter, 16 Dioden mit einem Grünfilter und 16 Dioden ohne Filter ausgeführt. Somit kann der RGB-Anteil und die Intensität des einfallenden Lichtes gemessen werden.

Das Modul ist außerdem mit zwei weißen LEDs ausgestattet. Dadurch wird eine definierte Beleuchtung des Objektes gewährleistet.

Der TCS230-Chip wandelt die gemessene Spektralverteilung in eine Frequenz um, die danach in eine proportionale analoge Spannung umgesetzt wird.

Das Sensor-Board übergibt die einzelnen Farben als Zahlen an das Main Board. Diese Zahlen sind im Headerfile rrtlan.h definiert.

Digitaler Filter

Da die Messsignale innerhalb des Roboters mit einer sehr hohen Wahrscheinlichkeit störbehaftet sind, ist es notwendig die Signale zu filtern. Zu diesem Zweck wurde ein digitales Filter auf dem Sensor Board implementiert.

Eine einfache Implementierung ist in Abbildung 5.11 dargestellt. Dabei handelt es sich um ein digitales Filter 1. Ordnung. Das entspricht einem einfachen RC-Filter das dem Sensorsignal nachgeschaltet wäre.

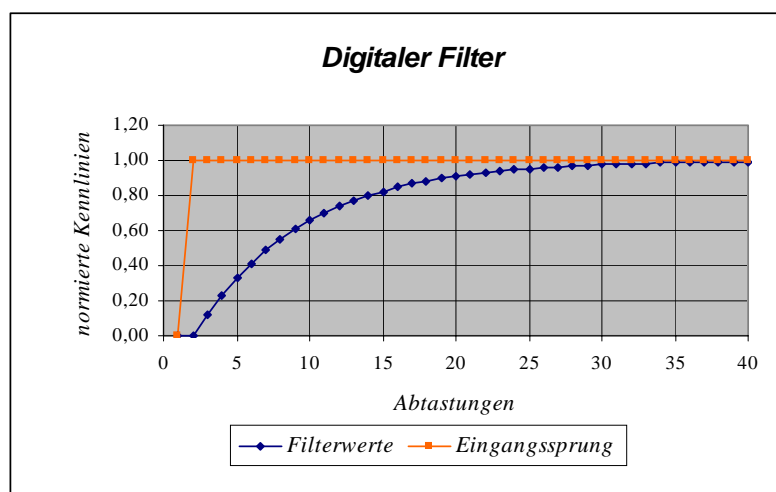


Abbildung 5.11 Normierte Kennlinie des digitalen Filters (Filterkoeffizient 3)

²⁰ [axe045, 2008]

Diese Filter kann sehr einfach mit der rekursiven Funktion (Gleichung 5.3) nachgebildet werden. Dabei ist u_k der Ausgabewert und $u_{e,k}$ der Eingangswert zum Abtastzeitpunkt k . u_{k-1} ist der Ausgabewert zum Zeitpunkt $k-1$ und 2^m ist der Filterkoeffizient. Der Filterkoeffizient wurde als Potenzwert von 2 implementiert, da eine solche Division sehr schnell vom Microcontroller durch Schiebeoperationen ausgeführt werden kann. Die Zeitkonstante des Filters ist somit vom Wert m abhängig.

$$u_k = \frac{\sum_k u_{e,k} - u_{k-1}}{2^m} \quad (5.3)$$

Um das zeitliche Verhalten des Filters auf eine Eingangsspannungsänderung zu berechnen, müssen zwei aufeinanderfolgende Ausgabewerte subtrahiert werden (siehe Gleichung 5.4). In Abbildung 5.12 ist die Antwort auf eine Störung dargestellt.

$$\Delta u_k = \frac{\sum_{k=1}^n u_{e,k} - u_{k-1}}{2^m} - \frac{\sum_{k=1}^{n-1} u_{e,k} - u_{k-1}}{2^m}$$

$$\Delta u_n = \frac{u_{e,n} - u_{n-1}}{2^m} \quad (5.4)$$

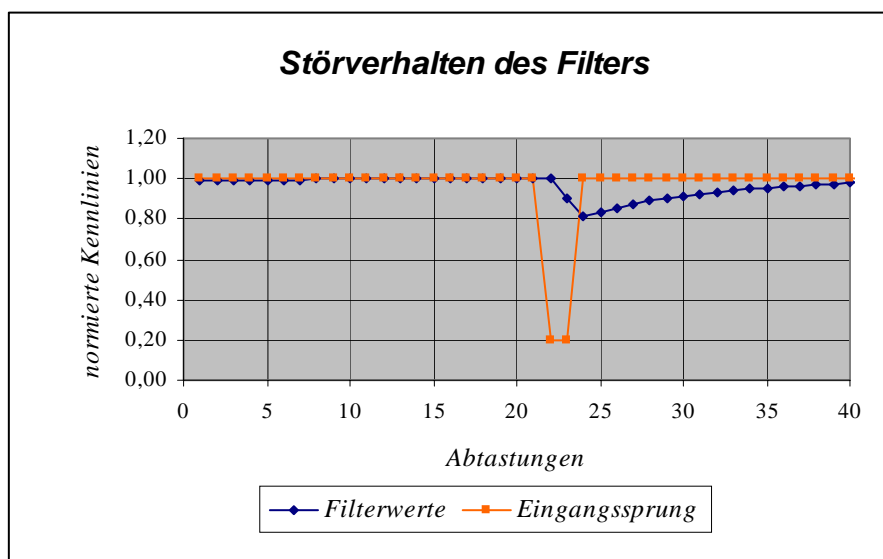


Abbildung 5.12 Störverhalten des Filters 1. Ordnung

Daraus folgt: um Störungen stark zu unterdrücken, muss ein hoher Filterkoeffizient gewählt werden. Dies steht jedoch im klaren Widerspruch zu der Forderung, dass das Filter rasch auf Eingangsänderungen reagieren soll. Denn um hohe Fahrgeschwindigkeiten des Roboters zu ermöglichen, darf der Sensorwert nicht um einige 100 ms verzögert werden.

Um diese beiden Forderungen zu erfüllen, wurde ein adaptiver Filteralgorithmus entwickelt. Es wird der Filterkoeffizient als Funktion der Eingangssignaländerung angepasst (siehe Gleichung 5.5).

$$u_k = \sum_k \frac{u_{e,k} - u_{k-1}}{2^{m(\Delta u_{e,k}, k)}} \quad (5.5)$$

Wie in Gleichung 5.6 dargestellt, wird bei einer relativen Eingangsspannungsänderung von 5% der Filterkoeffizient auf den Maximalwert gesetzt und bei jeder weiteren Abtastung um eins verringert.

$$m(\Delta u_{e,k}, k) := \begin{cases} 6 & \text{für } \Delta u_{e,k} \geq 5\% \\ 5 & \text{für } k + 1 \\ 4 & \text{für } k + 2 \\ 3 & \text{für } k + 3 \\ 2 & \text{für } k + 4 \\ 1 & \text{für } k + 5 \end{cases} \quad (5.6)$$

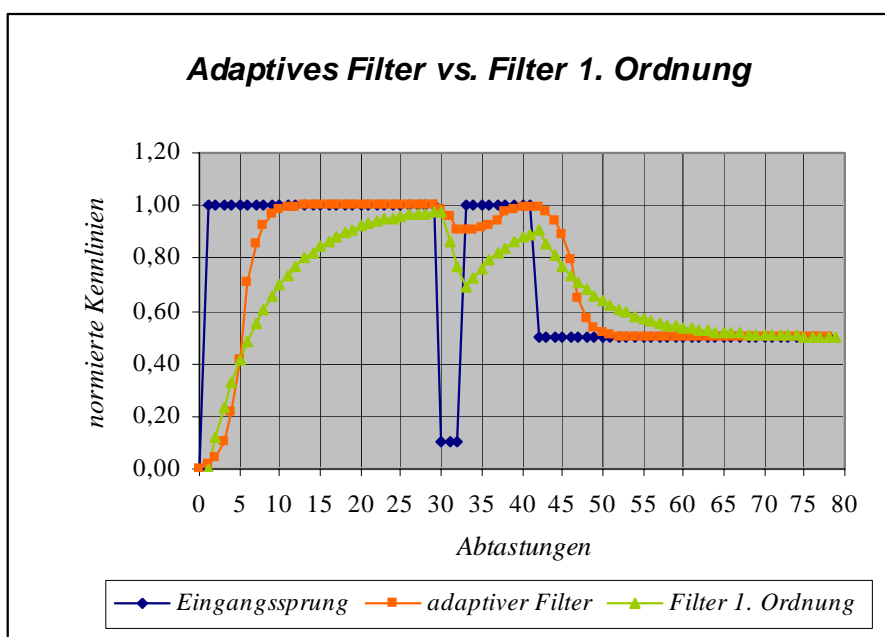


Abbildung 5.13 Gegenüberstellung adaptives Filter und Filter 1. Ordnung

Wie aus Abbildung 5.13 ersichtlich ist hat das adaptive Filter eine bessere Störspannungsunterdrückung (Werte um die Abtastung 30) und eine schnelleres Ansprechen bei Werteänderungen.

5.2.2. SPI Meldungen Sensor-Board

Init-Meldung

Bei der Meldung zum Initialisieren der Sensoreingänge (siehe Abbildung 5.14) wird als erstes Byte die Meldungskennung (*SB_INIT_MSG*) übertragen. Danach wird die Sensornummer des zu initialisierenden Sensors an das Sensor Board geschickt. Als drittes Byte wird der Sensortyp übertragen. Die möglichen Sensortypen sind in Tabelle 5.12 aufgelistet.

Meldungsformat Master → Sensorboard:

1. Byte	2. Byte	3. Byte	4. Byte	5. Byte
Meldungsnummer (<i>SB_INIT_MSG</i>)	Sensornummer (0 ... 15)	Sensortype	Messdelay (Highbyte) nur bei SRF08/10	Messdelay (Lowbyte) nur bei SRF08/10

Abbildung 5.14 Meldung *SB_INIT_MSG*

Meldung	Wert	Sensor	Sensorart	Messbereich	Messart
<i>SB_NOT_INIT</i>	0x00	-	-	-	-
<i>SB_GP2D120</i>	0x01	GP2D120	IR-Sensor	4cm – 30cm	analog
<i>SB_GP2D12</i>	0x02	GP2D12	IR-Sensor	10cm – 80cm	analog
<i>SB_AXE045</i>	0x03	AXE045	Farbsensor	RGB-Farben	analog
<i>SB_DIGITAL</i>	0x04	dig. Eingang	-	-	Digital
<i>SB_GP2Y0A02F</i>	0x05	GP2Y0A02F	IR-Sensor	20cm – 150cm	Analog
<i>SB_SRF08</i>	0x06	SRF08/10	Ultraschall-Sensor	3cm – 600cm	I2C

Tabelle 5.12 Unterstützte Sensortypen

Für den Sensor SRF08/10 muss noch zusätzlich ein Messdelay (M_D) übergeben werden. Dieses gibt an wie viel Zeit (T) zwischen Initiation der Messung und dem Auslesen des Sensors verstreichen soll. Die Zeit ist in 100 μ s Takten aufgeteilt. Zum Berechnen des Wertes kann Gleichung 5.7 verwendet werden.

$$M_D = \frac{T}{100\mu s} \quad (5.7)$$

Wird der Sensortyp SRF08/10 ausgewählt, werden zwei Sensorports auf dem Sensor-Board benötigt. Denn pro Sensorport ist nur eine Signalleitung ausgeführt, und der Sensor SRF08/10 benötigt zwei Signalleitungen (SCL, SDA). Die Stecker sind der Reihe nach zusammen-

geschlossen, K4 und K5, K6 und K7, usw. SCL liegt auf den Steckern mit dem geraden Index, also auf K4, K6,... . SDA liegt auf den Steckern mit ungeradem Index, also auf K5, K7,... .

Read-Sensor Meldung

Bei der Meldung zum Auslesen der Sensoren (siehe Abbildung 5.15) wird als erstes Byte die Meldungskennung (*SB_READ_SENSOR_MSG*) übertragen. Danach wird die Sensornummer des Sensors an das Sensor-Board geschickt. Als drittes und viertes Byte wird jeweils ein Leerbyte übermittelt, gleichzeitig schickt das Sensor Board die jeweiligen Messwerte zum Master. Distanzen werden in zwei Bytes übertragen, dabei wird zuerst das High-Byte und dann das Low-Byte übertragen. Der Messwert wird in mm ausgegeben. Digitale Eingänge geben den jeweiligen Zustand im Low-Byte zurück.

Meldungsformat Master → Sensorboard:

1. Byte	2. Byte	3. Byte	4. Byte
<i>Meldungsnummer (SB_READ_SENSOR_MSG)</i>	<i>Sensornummer (0 ... 15)</i>	<i>Leerbyte</i>	<i>Leerbyte</i>

Meldungsformat Sensorboard → Master:

1. Byte	2. Byte	3. Byte	4. Byte
<i>Leerbyte</i>	<i>Leerbyte</i>	<i>Distanz [mm] (Highbyte)</i>	<i>Distanz [mm] (Lowbyte)</i>

Abbildung 5.15 Meldung SB_READ_SENSOR_MSG

Set-Pin Meldung

Das Sensor-Board kann auch als erweitertes I/O-Board verwendet werden. Da alle Pins nach dem Start des Sensor-Boards auf Eingang initialisiert werden, kann mit dieser Meldung der entsprechende Pin von Eingang auf Ausgang, und umgekehrt, umgeschaltet werden. Zuerst wird die Meldungskennung (*SB_SET_PIN_MSG*) übertragen (siehe Abbildung 5.16). Danach wird die Pin-Nummer an das Sensor-Board geschickt. Als drittes Byte wird der jeweilige Zustand übermittelt (0 → Eingang, 1 → Ausgang).

Meldungsformat Master → Sensorboard:

<i>1. Byte</i>	<i>2. Byte</i>	<i>3. Byte</i>
<i>Meldungsnummer (SB_SET_PIN_MSG)</i>	<i>Pin Nummer (0 ... 15)</i>	<i>Wert (0/1)</i>

Abbildung 5.16 Meldung SB_SET_PIN_MSG

Set-I/O Meldung

Das Sensor-Board kann auch als erweitertes I/O-Board verwendet werden. Bei der Meldung zum Setzen von Ausgängen (siehe Abbildung 5.17) wird als erstes Byte die Meldungskennung (*SB_SET_IO_MSG*) übertragen. Danach wird die I/O-Nummer an das Sensor-Board geschickt. Als drittes Byte wird der jeweilige Zustand übermittelt.

Meldungsformat Master → Sensorboard:

<i>1. Byte</i>	<i>2. Byte</i>	<i>3. Byte</i>
<i>Meldungsnummer (SB_SET_IO_MSG)</i>	<i>I/O Nummer (0 ... 15)</i>	<i>Wert (0/1)</i>

Abbildung 5.17 Meldung SB_SET_IO_MSG

5.2.3. Treiber Servo-Board

Wie in Tabelle 5.13 dargestellt ist die Software in 7 Module aufgeteilt. Es wurde darauf geachtet, dass die Module leicht auf andere Projekte bzw. Plattformen portiert werden können.

<i>Modul</i>	<i>Funktion</i>
RM_Servoboard_main.c	Hauptprogramm
uart.c	stellt Treiber für die serielle Schnittstelle, für z.B. eine Debug-Schnittstelle, zur Verfügung
ports.c	stellt die Treiber/Makros für den Zugriff auf die I/Os zur Verfügung
spi.c	stellt die Treiber für die SPI-Schnittstelle zur Verfügung; hier läuft die Kommunikation mit dem Master-Modul
timer1.c	Im Timer 1 Interrupt werden die Servos 1 bis 8 angesteuert
timer2.c	Im Timer 2 Interrupt werden die Servos 9 bis 16 angesteuert
servo.c	stellt die Umrechnungsfunktionen für Winkel-in-Pulsdauer zur Verfügung

Tabelle 5.13 Softwaremodule des Servo-Boards

Ansteuerung der Servos

Um den Servo auf einen bestimmten Winkel zu stellen, muss mit einer Wiederholfrequenz von 50 Hz, ein Puls ausgegeben werden. Dieser Puls variiert von Hersteller zu Hersteller, die Pulsdauer für die unterstützten Hersteller wird weiter unter beschrieben.

Das Servo-Board kann bis zu 16 Servos ansteuern, wobei Servo 1 bis Servo 8 vom Timer 1 und Servo 9 bis Servo 16 vom Timer 2 angesteuert werden. Die Ansteuerung der Servos erfolgt wie in Abbildung 5.18 dargestellt, d.h. es wird zuerst der Puls für Servo 1 ausgegeben, 2,3 ms später wird mit Servo 2 fortgefahren usw. bis Servo 8. Nach dem Puls von Servo 8 wird gewartet bis die vollen 20 ms vorüber sind, um anschließend wieder bei Servo 1 zu beginnen. Somit kann gewährleistet werden, dass jeder Servo alle 20 ms angesteuert wird.

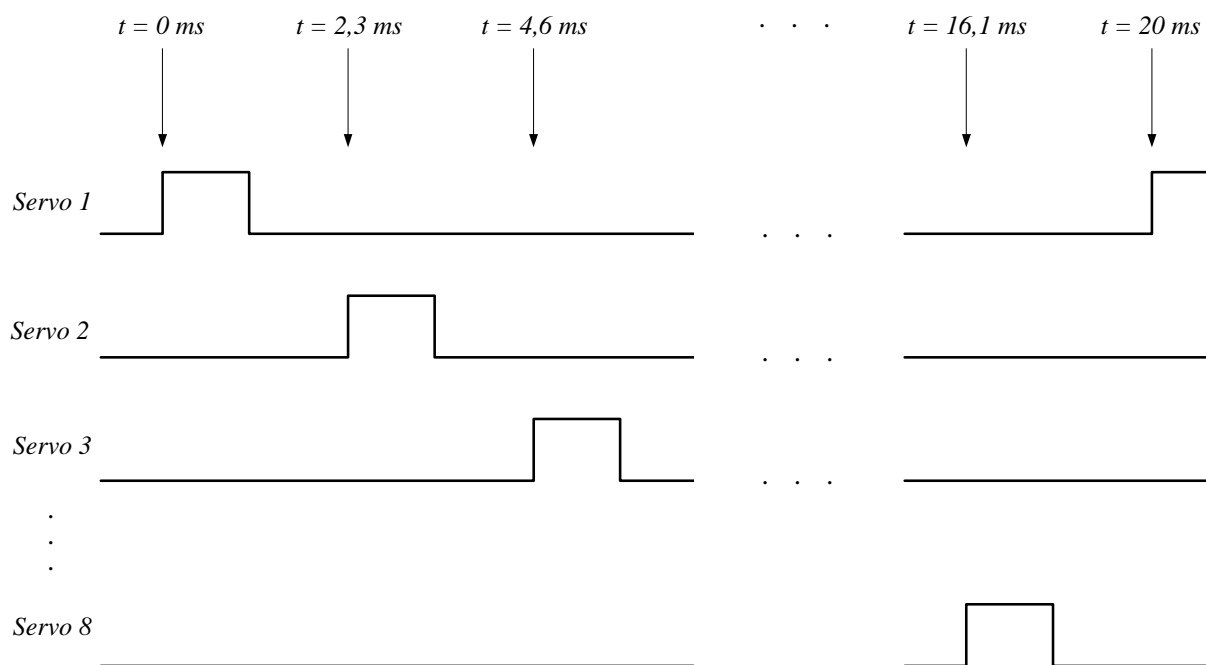


Abbildung 5.18 Timing bei der Ansteuerung der einzelnen Servos

Wenn ein neuer Winkel vom Master übergeben wird, wird die aktuelle Impulsdauer im Hauptprogramm ausgerechnet und in der Pause vom letzten Puls (Servo 8) bis zum ersten Puls (Servo 1) in die Interrupt-Routine übergeben. Somit werden undefinierte Zustände beim Überschreiben der Werte verhindert.

5.2.4. SPI-Meldungen Servo-Board

Init-Meldung

Bei der Meldung zum Initialisieren der Servos (siehe Abbildung 5.19) wird als erstes Byte die Meldungskennung (*SV_INIT_MSG*) übertragen. Danach wird die Servonummer des zu initialisierenden Servos an das Servo Board geschickt. Als drittes Byte wird der Servotyp übertragen. Die möglichen Servotypen sind in Tabelle 5.14 aufgelistet.

Meldungsformat Master → Servoboard:

<i>1. Byte</i>	<i>2. Byte</i>	<i>3. Byte</i>
<i>Meldungsnummer (SV_INIT_MSG)</i>	<i>Servonummer (0 ... 15)</i>	<i>Servotype</i>

Abbildung 5.19 Meldung *SV_INIT_MSG*

<i>Meldung</i>	<i>Wert</i>	<i>Servo</i>	<i>Impulsdauer</i>
<i>SV_NOT_INIT</i>	0x00	-	-
<i>SV_BLUEBIRD</i>	0x01	BMS-136, BMS706, BMS-630	1,2 ms ... 1,6 ms ²¹
<i>SV_HI TECH</i>	0x02	HS-815BB	0,9 ms ... 2,1 ms ²²

Tabelle 5.14 Unterstützte Servotypen

²¹ [Bluebird, 2008]

²² [HiTech, 2008]

Set-Servo-Meldung

Bei der Meldung zum Setzen der Winkel (siehe Abbildung 5.17) wird als erstes Byte die Meldungskennung (*SV_SET_SERVO_MSG*) übertragen. Danach wird die Servonummer an das Servo-Board geschickt. Als drittes Byte wird der jeweilige normierte Winkel übermittelt, es wird ein Wert von 0 ... 255 übertragen und dieser Wert vom Servo-Board auf die entsprechende Impulsdauer umgerechnet. Beim normierten Winkel entspricht der Wert 0, 0°, 127 entspricht der Mittelstellung des Servos und 255 entspricht dem maximalen Winkel.

Meldungsformat Master → Servo-Board:

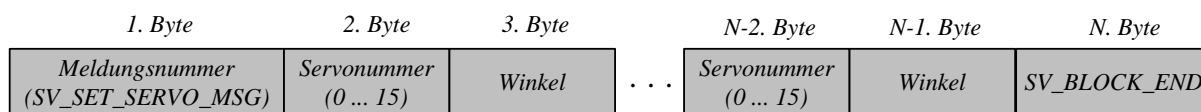


Abbildung 5.20 Meldung SV_SET_SERVO_MSG

Die Meldung hat keine fixe Länge, sonder es können beliebig viele Servos und deren Winkel übertragen werden. Abgeschlossen wird die Meldung mit *SV_BLOCK_END* (0xFF).

6. LCD-INTERFACE

6.1. HARDWARE

6.1.1. LCD Modul

Da es sehr oft nützlich ist, Statusmeldungen auszugeben oder einfache Einstellungen am Roboter vorzunehmen, ohne dass das Main-Board neu programmiert wird, wurde das LCD-Modul entwickelt.

Mit dem LCD-Modul kann das LCD-Display 204A der Fa. Displaytech Ltd. angesteuert werden. Es können jedoch andere Displays mit dem gleichen Interface angeschlossen werden, dies erfordert jedoch eine Adaption der Software.

Das LCD-Modul ist, wie aus Abbildung 6.1 hervorgeht, mit den Schnittstellen für

- Versorgung (K1 siehe Tabelle 6.1)
- Interface zum LCD-Display (K2 siehe Tabelle 6.2)
- Programmierung (K3 siehe Tabelle 6.3)
- Tastatur-Interface (K4 siehe Tabelle 6.4)
- Kommunikation mit dem Master via SPI (K5 siehe Tabelle 6.5)

ausgestattet.

Für die Versorgung der Elektronik und der Peripherie mit 5V sorgt der Schaltregler LM2594 der Fa. National Semiconductor. Mit diesem Schaltregler ist ein maximaler Ausgangsstrom von 500 mA möglich.

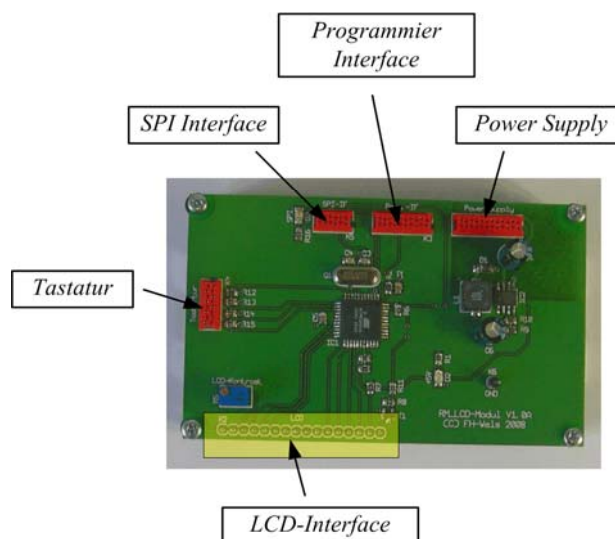


Abbildung 6.1 Bestückte Leiterplatte des LCD-Moduls

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1,3,5,7,9,11	+12V	Pluspol der Versorgung
2,4,6,8,10,12	GND	Minuspole der Versorgung

Tabelle 6.1 Steckerbelegung von K1 (Power-Supply)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	GND	Minuspole der Versorgung für das LCD-Display
2	+5V	Pluspol der Versorgung für das LCD-Display
3	V ₀	Kontrasteinstellung des LCD-Displays
4	RS	Register Auswahl RS = 0: Befehls Register RS = 1: Daten Register
5	R/W	Read/Write R/W = 0: Schreiben R/W = 1: Lesen
6	EN	Enable
7,8,9,10	DB0 ... 3	Datenleitungen 0 ... 3 (werden nicht verwendet)
11	DB4	Datenleitung 4
12	DB5	Datenleitung 5
13	DB6	Datenleitung 6
14	DB7	Datenleitung 7
15	A	Anode der Hintergrundbeleuchtung
16	K	Kathode der Hintergrundbeleuchtung

Tabelle 6.2 Steckerbelegung von K2 (LCD-IF)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	Pluspol der Versorgung für das Programmier-Board
2	RxD	Empfangsleitung der Debug-Schnittstelle
4	SCK	Clock-Leitung der Programmierschnittstelle
5	TxD	Sendeleitung der Debug-Schnittstelle
6	MISO	Master In Slave Out der Programmier-Schnittstelle
8	MOSI	Master Out Slave In der Programmier-Schnittstelle
9	GND	Minuspole der Versorgung für das Programmier-Board
10	/Reset	Reset-Leitung der Programmier-Schnittstelle
3, 7	NC	nicht verwendet

Tabelle 6.3 Steckerbelegung von K3 (Prog.-IF)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1, 3, 5, 7	+5V	Pluspol der Versorgung für den Tastenblock
2	T1	Taste 1 → Menü-Taste
4	T2	Taste 2 → Up-Taste
6	T3	Taste 2 → Down-Taste
8	T4	Taste 2 → Enter-Taste

Tabelle 6.4 Steckerbelegung von K4 (Tastatur-IF)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	/CS	Chip-Select des Slaves
2	SCK	Synchroner Clock
3	MISO	Master In - Slave Out zur Datenübertragung zum Master
4	MOSI	Master Out - Slave In zur Datenübertragung zum Slave
5,6	GND	GND der Schnittstelle

Tabelle 6.5 Steckerbelegung für K5 (SPI-IF)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	GND	Massestift für Messungen

Tabelle 6.6 Steckerbelegung von K6 (Massestift)

6.1.2. LCD²³

Bei dem LCD-Display 204A der Fa. Displaytech Ltd. (siehe Abbildung 6.2) handelt sich um ein Standard Display mit einem zum Hitachi HD44780 kompatiblen Chip. Das Display hat 4 Zeilen mit jeweils 20 Zeichen. Die Steckerbelegung des LCD-Interfaces ist Tabelle 6.2 zu entnehmen.

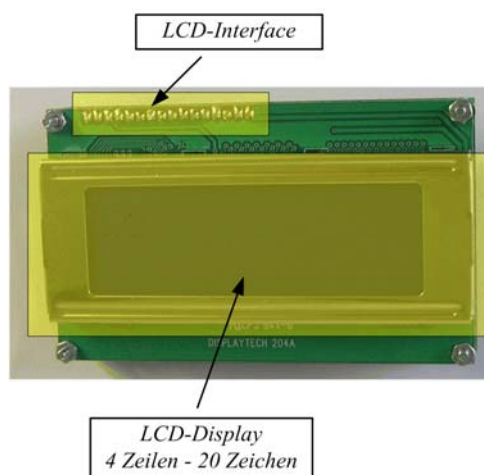


Abbildung 6.2 LCD-Display 204A der Fa. Displaytech Ltd.

In Abbildung 6.3 ist das Blockschaltbild des LCD-Displays zu entnehmen.

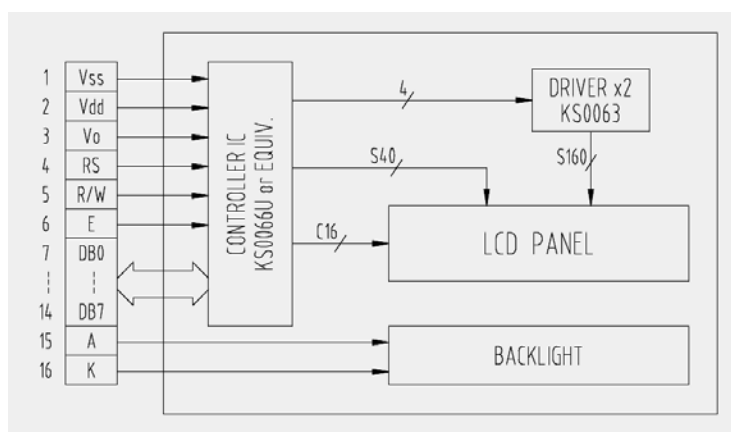


Abbildung 6.3 Blockdiagramm des LCD-Displays

²³ [Displaytech, 2007]

6.1.3. Tastatur

Bei der Tastatur handelt es sich um eine Tastenfeld mit vier Tastern, die individuell beschriftet werden können. Der Anschluss des Tastenfeldes erfolgt über einen 5 pol. Steckers, wobei Pin 1 der gemeinsame Anschluss der Taster ist und Pin 2 bis 5 zu den einzelnen Tastern führt.

Auf dem LCD-Modul ist der gemeinsame Anschluss auf +5V geschaltet und die einzelnen Taster sind mittels Pull-Down Widerstände gegen Masse geschlossen, die Tastereingänge sind also highaktiv. In der Beschaltung ist keine Tasterentprellung vorgesehen, diese wird in der Software behandelt.

6.2. SOFTWARE

6.2.1. Treiber

Bei der Erstellung der Software wurde darauf geachtet, dass die einzelnen Module so flexibel wie möglich auch in anderen Projekten eingesetzt werden können.

Wie in Tabelle 6.7 dargestellt ist die Software in 9 Module aufgeteilt. Dabei kann, in Module die direkt auf die Hardware des Microcontrollers zugreift (Hardware Abstraction Layer), und in Module, die unabhängig von der Hardware funktionieren, unterschieden werden.

<i>Modul</i>	<i>Funktion</i>
main.c	Hauptprogramm
uart.c	stellt Treiber für die serielle Schnittstelle, für z.B. eine Debug-Schnittstelle, zur Verfügung
adc.c	stellt Treiber für den AD-Wandler zur Verfügung
ports.c	stellt die Treiber/Makros für den Zugriff auf die I/Os zur Verfügung
spi.c	stellt die Treiber für die SPI-Schnittstelle zur Verfügung; hier läuft die Kommunikation mit dem Master-Modul
acomp.c	stellt die Treiber für den Analog-Komparator zur Verfügung
timer0.c	erzeugt einen 1 ms – und 100 ms – Takt für das Hauptprogramm
filter.c	stellt die Treiber für das digitale Filter 1. Ordnung zur Verfügung
key.c	stellt die Treiber zum Einlesen der Tastatur zur Verfügung

Tabelle 6.7 Softwaremodule des LCD-Interfaces

Hauptprogramm

Im Hauptprogramm wird zuerst der Microcontroller initialisiert und anschließend das Startbild am LCD-Display ausgegeben. In der Main-Loop gibt es zwei unterschiedliche Zeitintervalle, nämlich eine 1 ms Intervall und ein 100 ms Intervall (siehe Abbildung 6.4).

Im 1 ms Intervall wird die Tastatur eingelesen und die Batteriespannung gemessen und in ein 0,1 V – Format umgerechnet.

Im 100 ms Intervall werden die Befehle, die sich aus den gedrückten Tasten ergeben, abgearbeitet und das LCD-Display angesteuert.

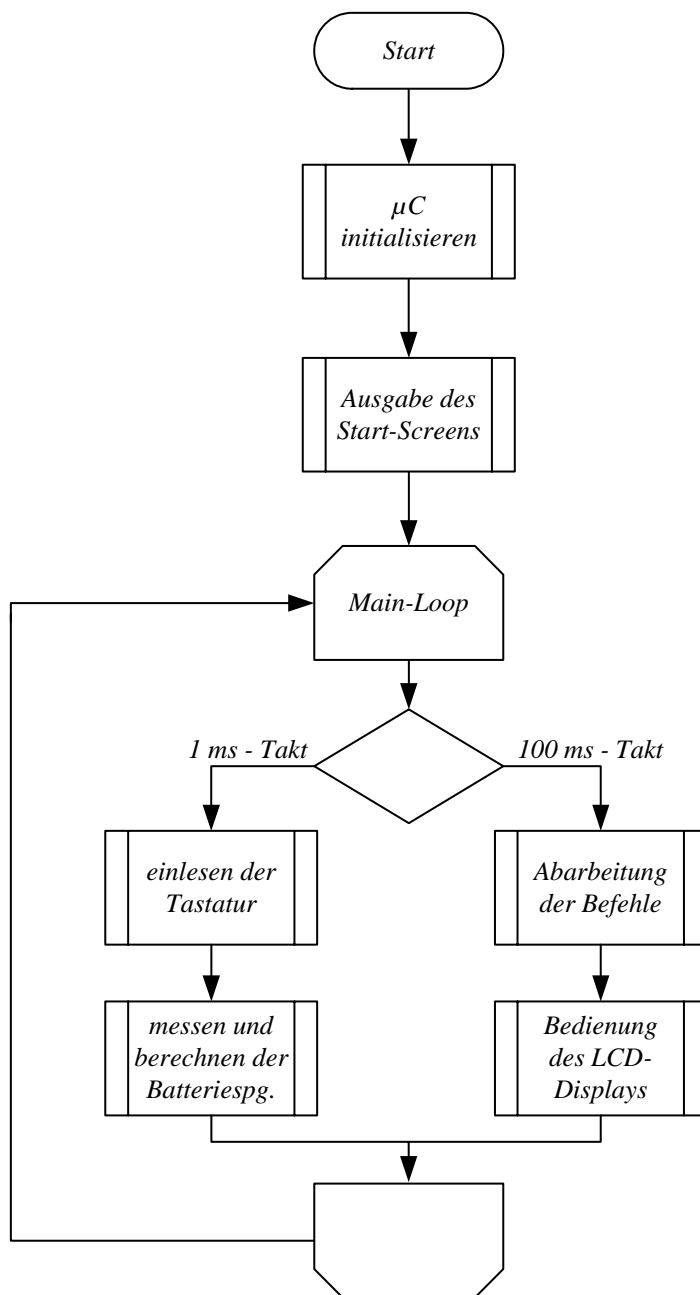


Abbildung 6.4 Flussdiagramm Hauptprogramm

LCD-Ansteuerung²⁴

Zur Ansteuerung des LCD-Displays wurden die Funktionen aus der Programm-Bibliothek des C-Compilers verwendet. Diese stellt umfangreiche Befehle zum Ansteuern des Displays zur Verfügung.

Die Programm-Bibliothek unterstützt alphanumerische Displays die mit dem Hitachi HD44780 Chip, oder einem vergleichbaren Chip, ausgestattet sind. Um auf die LCD-Funktionen zuzugreifen zu können, muss das Headerfile lcd.h in da Projekt eingebunden werden.

Es werden die LCD-Displays mit den Formaten 1x8, 2x12, 3x12, 1x16, 2x16, 2x20, 4x20, 2x24 und 2x40 unterstützt.

In den Tabelle 6.8 bis Tabelle 6.13 werden die verwendeten Befehle der Bibliothek näher erklärt:

<i>Funktionsname</i>	<i>lcd_init</i>	
<i>Funktion</i>	Initialisiert das LCD-Display und setzt den Cursor auf den Zeilen- und Spaltenindex 0 (Cursor links oben im Display) Diese Funktion muss vor allen anderen Befehlen aufgerufen werden	
<i>Übergabewerte</i>	<i>lcd_columns</i>	Übergibt die Anzahl der Spalten des Displays; z.B. 20 für ein Display mit 20 Zeichen
<i>Rückgabewerte</i>	1 ... wenn das Display erkannt wird 0 ... wenn kein Display erkannt wird	

Tabelle 6.8 Befehl zum Initialisieren des LCD-Displays

<i>Funktionsname</i>	<i>lcd_clear</i>
<i>Funktion</i>	Löscht das LCD-Display und setzt den Cursor auf den Zeilen- und Spaltenindex 0 (Cursor links oben im Display)
<i>Übergabewerte</i>	Kein Übergabeparameter
<i>Rückgabewerte</i>	Kein Rückgabeparameter

Tabelle 6.9 Befehl zum Löschen des LCD-Displays

²⁴ [hpinfotech, 2005]

Funktionsname	<i>lcd_gotoxy</i>	
Funktion	Setzt den Cursor auf die angegebene Stelle Bei den Zeilen und Spalten wird bei Null zu zählen begonnen	
Übergabewerte	<i>x</i>	Spaltennummer
	<i>y</i>	Zeilennummer
Rückgabewerte	Kein Rückgabewert	

Tabelle 6.10 Befehl zum Setzen des Cursors

Funktionsname	<i>lcd_putchar</i>	
Funktion	Schreibt ein Zeichen auf die aktuelle Cursorposition	
Übergabewerte	<i>c</i>	Das zu schreibende Zeichen
Rückgabewerte	Kein Rückgabewert	

Tabelle 6.11 Befehl zum Schreiben eines Zeichens

Funktionsname	<i>lcd_puts</i>	
Funktion	Schreibt eine Zeichenkette auf die aktuelle Cursorposition; Mit dem Befehl <i>sprintf</i> können zuerst in die Zeichenkette beliebig Variablen eingefügt werden bevor er am Display ausgegeben wird.	
Übergabewerte	<i>*str</i>	Pointer auf die zu schreibenden Zeichenkette Die Zeichenkette wird aus dem RAM gelesen
Rückgabewerte	Kein Rückgabewert	

Tabelle 6.12 Befehl zum Schreiben einer Zeichenkette aus dem RAM

Funktionsname	<i>lcd_putsf</i>	
Funktion	Schreibt eine Zeichenkette auf die aktuelle Cursorposition	
Übergabewerte	<i>*str</i>	Pointer auf die zu schreibenden Zeichenkette Die Zeichenkette wird aus dem Flash gelesen
Rückgabewerte	Kein Rückgabewert	

Tabelle 6.13 Befehl zum Schreiben einer Zeichenkette aus dem Flash

Tastatur

Mit dem Modul `key.c` wurde ein einheitliches Programm für die Behandlung von Tastern realisiert. Das Modul besteht aus dem Unterprogramm `HandleKey(tsKey* psKey, unsigned char ucKey)` und der Datenstruktur `tsKey` (siehe Tabelle 6.14).

In das Unterprogramm werden eine Pointer auf eine Taster-Struktur und der jeweilige Zustand des Tasters übergeben. Im Unterprogramm wird der Status des Tasters ermittelt (`PRESSED`, `RISING_EDGE`, `FALLING_EDGE` und wie lange der Taster eventuell schon gedrückt ist) und in die Struktur eingetragen. Weiters werden die Taster digital „entprellt“ bevor der Status ermittelt wird.

Das Modul für die Tastenbehandlung bietet einfache Methoden für den Zugriff auf die einzelnen Eigenschaften der Taster.

- `IS_RISING_EDGE(Key_Handle)` – gibt 1 zurück wenn eine steigende Flanke am entsprechenden Taster erkannt wird
- `IS_FALLING_EDGE(Key_Handle)` – gibt 1 zurück wenn eine fallende Flanke am entsprechenden Taster erkannt wird
- `IS_PRESSED(Key_Handle)` – gibt 1 zurück wenn der entsprechenden Taster gedrückt ist
- `CLEAR_RISING_EDGE(Key_Handle)` – löscht Flag für die steigende Flanke
- `CLEAR_FALLING_EDGE(Key_Handle)` – löscht Flag für die fallende Flanke

Variable	Datentyp	Beschreibung
<i>ucStatus</i>	<i>unsigned char – 8 Bit</i>	Gibt über den Zustand Auskunft 0x00 ... Taste nicht gedrückt 0x01 ... Taste nicht gedrückt <i>PRESSED</i> 0x02 ... steigende Flanke <i>RISING_EDGE</i> 0x03 ... fallende Flanke <i>FALLING_EDGE</i>
<i>ucFailureCounter</i>	<i>unsigned char – 8 Bit</i>	Zähler zum digitalen „Entprellen“ des Tasters
<i>uiPressTime</i>	<i>unsigned int – 16 Bit</i>	gibt zurück wie lange der entsprechende Taster gedrückt ist (wird bei jedem Aufruf um eins erhöht)

Tabelle 6.14 Variablen der Datenstruktur „sKey“

6.2.2. SPI Meldungen

Mit der Meldung *LCD_SET_COLOUR_MSG* wird dem LCD-Modul die vom Roboter ermittelte Spielfarbe übergeben. Vom LCD-Modul werden die ausgewählte Strategie wie auch der Spielfarben-Modus zurückgegeben (siehe Abbildung 6.5). Als Spielfarben-Modus wird entweder der Wert *AUTOMATIK* oder die Farben *ROT*, *BLAU* oder *GRUEN* zurückgegeben, dies dient dazu, um bei einer Fehlfunktion der automatischen Spielfarbenerkennung die Farbauswahl vom Bediener „overruled“ werden kann. Beim Wert *AUTOMATIK* bleibt die Farbauswahl, ansonsten wird der Wert vom LCD-Modul übernommen.

Meldungsformat Master → LCD-Board:

1. Byte	2. Byte	3. Byte
<i>Meldungsnummer (LCD_SET_COLOUR_MSG)</i>	<i>Spielfarbe</i>	<i>Leerbyte</i>

Meldungsformat LCD-Board → Master:

1. Byte	2. Byte	3. Byte
<i>Leerbyte</i>	<i>Strategie</i>	<i>Spielfarbe</i>

Abbildung 6.5 Meldung *LCD_SET_COLOUR_MSG*

7. FUNK ZU USB UMSETZER

7.1. FUNK-MODUL²⁵

Das ER400TRS (Abbildung 7.1) Funk-Modul der Fa. Easy Radio beinhaltet eine Sende- und Empfangseinheit, einen sog. Transceiver. Die Daten werden vom Host über die serielle Schnittstelle zum Funk Modul übertragen und von dort aus via Funk zum Empfänger.



Abbildung 7.1 ER400TRS Funk Modul

Die Sendefrequenz des Moduls beträgt 433 – 434 MHz, die maximale Reichweite 250 m. Die Empfangsempfindlichkeit liegt bei einer Baudrate von 19.200 Baud typischerweise bei -103 dbm. Die Sendeleistung kann von 1 mW bis zu 10 mW, in 1 mW Schritten, eingestellt werden. Das Modul kann von 3,3 V bis 5,5 V versorgt werden und weist einen sehr kleinen Versorgungsstrom auf (19 mA beim Empfangen und 23 mA beim Senden).

In Tabelle 7.1 sind die wichtigsten ab Werk Einstellungen vermerkt. Diese Einstellungen können jedoch abgeändert werden.

<i>Parameter</i>	<i>Ab Werk Einstellungen</i>
Baudrate	19.200 Baud
Datenformat	1 Start, 8 Data, No Parity, 1 Stop
End of Data Delay	2 * Bytelänge
Sendedauer	13,2 ms + (n Bytes * 0,8 ms)
Puffergröße	1 – 180 Bytes

Tabelle 7.1 Ab Werk Einstellungen des ER400TRS Moduls

²⁵ [Easy-Radio, 2007]

In Abbildung 7.2 ist eine Datenübertragung des Funk-Moduls dargestellt. Dem Funkmodul können maximal 180 Bytes über die serielle Schnittstelle übergeben werden. Nach einer Sendepause von der Dauer von zwei Bytes beginnt das Funk Modul die Daten über die Funkschnittstelle zu übertragen. Die Übertragungsdauer T berechnet sich nach Gleichung 7.1.

$$T = 13,2ms + (0,8ms \cdot n \text{ Bytes}) \quad (7.1)$$

Die Übertragungszeit setzt sich aus einer fixen Zeit zur Übertragung des Meldungs-Overheads und der variablen Datenlänge zusammen.

Wenn das volle Datenpaket übertragen wurde, beginnt das Funk-Modul mit der Übermittlung zum Empfänger-Host.

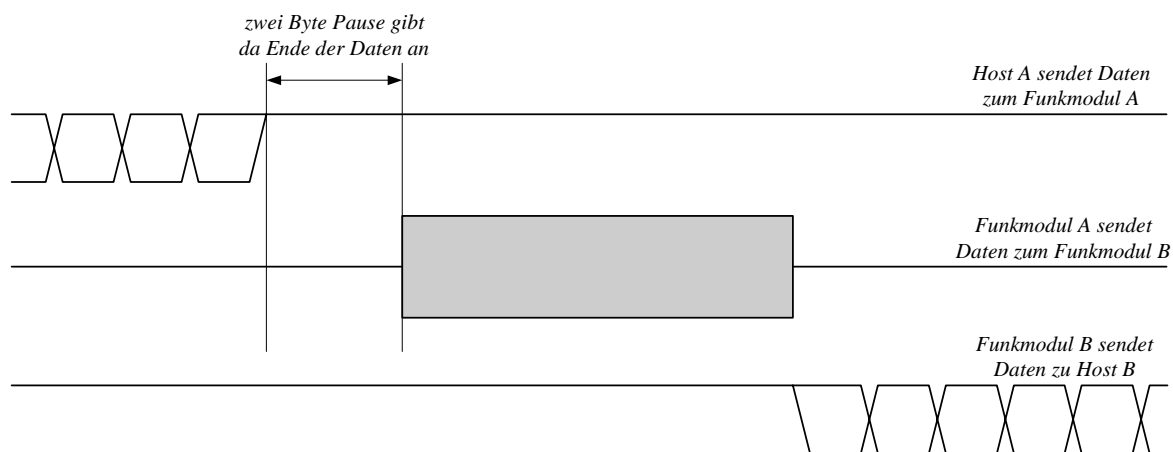


Abbildung 7.2 Datenübertragung des Funkmoduls

7.2. USB ZU UART KONVERTER²⁶

Der FT232BM Chip ist die zweite Generation eines USB zu UART Umsetzers der Fa. FTDI.

Dieser Chip weist unter anderem folgende Features auf:

- Single Chip USB ↔ asynchroner, serieller Datentransfer
- Volles Modem Interface
- Datenraten bis zu 300 M Baud
- 384 Byte Empfangsbuffer, 128 Byte Sendebuffer
- Unterstützt USB High-Power Geräte
- Integrierte 6 MHz ↔ 48 MHz PLL
- 4,4 V bis 5,25 Single Supply
- USB1.1 und USB2.0 kompatibel
- USB VID, PID, Seriennummer und Produktbeschreibung in externem EEPROM

Für den Chip sind Treiber auf der Homepage des Herstellers verfügbar, die eine virtuelle COM-Schnittstelle am Computer einrichten. Dadurch kann auf den Chip wie auf eine serielle Schnittstelle zugegriffen werden.

In Abbildung 7.3 ist das vereinfachte Blockschaltbild des FT232BL Chips dargestellt.

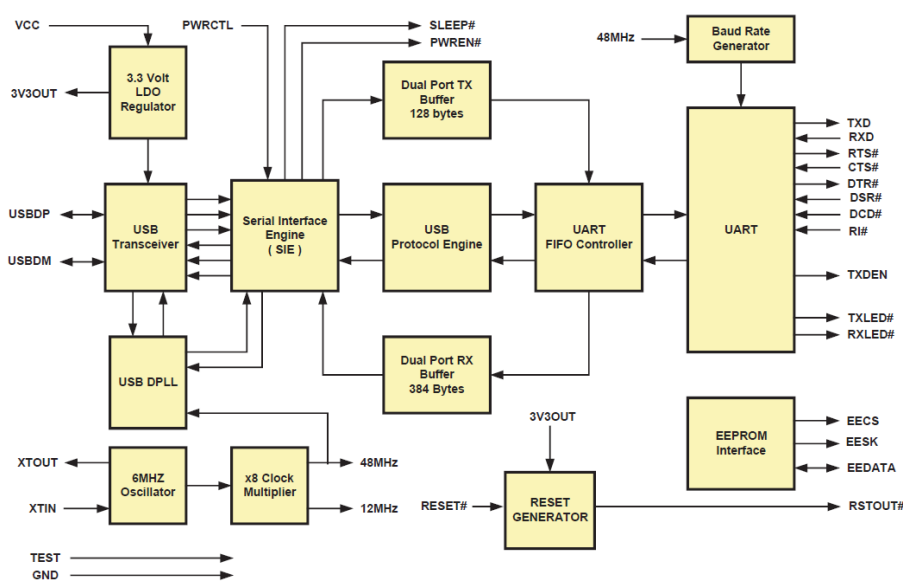


Abbildung 7.3 Vereinfachtes Block Diagramm des FT232BL Chips

²⁶ [FTDI, 2004]

7.3. INTERFACE ZUM MODULAREN SYSTEM

In Abbildung 7.4 ist das roboterseitige Funk-Modul dargestellt. Es besteht aus einem Anschluss für die Datenübertragung via

- Funk-Modul ER400TRS
- RS232 Schnittstelle
- Serielle Schnittstelle mit TTL-Pegel

Über das Roboter-Interface kann zwischen diesen drei Schnittstellen umgeschaltet werden. In Tabelle 7.2 sind die Auswahlmöglichkeiten dargestellt. Mit den LEDs D3 bis D6 wird der Signalpegel der Funkschnittstelle angezeigt.

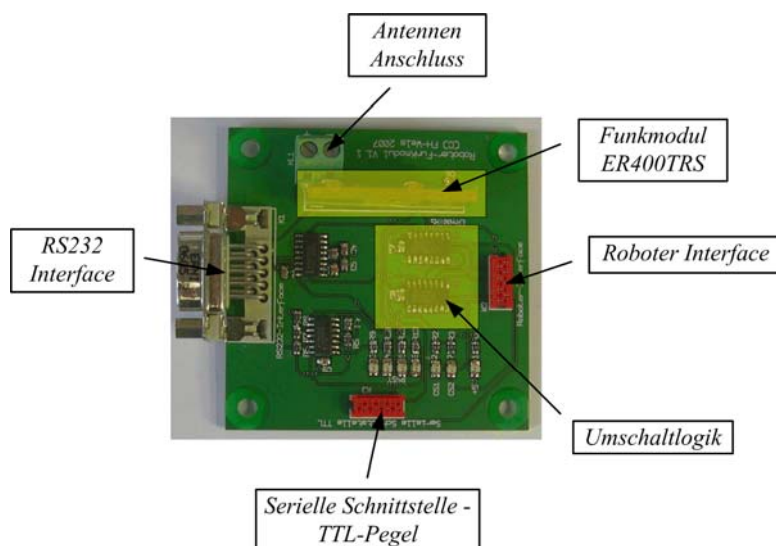


Abbildung 7.4 Bestückte Leiterplatte des Universal Interfaces

<i>CS3</i>	<i>CS2</i>	<i>CS1</i>	<i>Funktion</i>
x	0	0	Verbindung: Microcontroller → Funkmodul
x	0	1	Verbindung: Microcontroller → RS232-Schnittstelle
x	1	0	Verbindung: Microcontroller → serielle Schnittstelle mit TTL-Pegel
x	1	1	Keine Funktion

Tabelle 7.2 Auswahlmöglichkeiten der verschiedenen Schnittstellen

In Tabelle 7.3 ist die Anschlussbelegung des RS232 Interfaces beschrieben. Die Handshake-Leitungen wurden dabei nicht ausgeführt. Um einen Computer an das Modul anzuschließen wird ein ausgekreuztes Kabel benötigt.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
2	RxD	Empfangsleitung
3	TxD	Sendeleitung
5	GND	Minuspol der Versorgung des Konverters
1,4,6,7,8,9	NC	Nicht angeschlossen

Tabelle 7.3 Steckerbelegung von K1 (RS232-IF)

In Tabelle 7.4 wird das Interface in Richtung Roboter näher beschrieben. Es kann über die Leitungen CS1 und CS2, wie weiter oben beschrieben, die gewünschte Schnittstelle ausgewählt werden. Weiters kann über den Reserve Pin noch ein digitales Signal direkt übertragen werden. Über diesen Stecker wird das Modul versorgt.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	+5V-Versorgung für das Funk- und RS232 Modul
2	CS1	Chip-Select Leitung 1
3	+12V	+12V-Versorgung
4	CS2	Chip-Select Leitung 2
5	GND	Minuspol der Versorgung
6	Res	I/O in Richtung K3
7	TxD	Sendeleitung der seriellen Schnittstelle
8	RxD	Empfangsleitung der seriellen Schnittstelle

Tabelle 7.4 Steckerbelegung von K2 (Roboter IF)

In Tabelle 7.5 ist die Schnittstelle zu einem andern Microcontroller-Board dargestellt.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1, 2, 3, 4	NC	Nicht angeschlossen
5	GND	Masseleitung
6	Res	I/O in Richtung K2
7	RxD	Empfangsleitung der seriellen Schnittstelle
8	TxD	Sendeleitung der seriellen Schnittstelle

Tabelle 7.5 Steckerbelegung von K3 (TTL IF)

In Tabelle 7.6 ist der Antennen Anschluss näher beschrieben. Das Modul funktioniert auch ohne Antenne jedoch nur wenige Meter. Es wird empfohlen eine $\lambda/4$ -Antenne zu verwenden.

$$L = \frac{c}{4 \cdot f} \quad (7.2)$$

Um die Länge L der $\lambda/4$ -Antenne kann mit Hilfe von Gleichung 7.2 berechnet werden. Dabei ist c die Lichtgeschwindigkeit und f die Sendefrequenz des Moduls. Für das ER400TRS Modul ergibt sich bei einer Sendefrequenz von 433 MHz eine Antennenlänge von ca. 17 cm.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	ANT	Antennen Anschluss
2	GND	Minusanschluss der Antenne

Tabelle 7.6 Steckerbelegung von KL1 (Antennen Anschluss)

7.4. PC INTERFACE

In Abbildung 7.5 ist das computerseitige Funk-Modul dargestellt. Es besteht aus einem Antennenanschluss und einen USB-Anschluss für die Datenübertragung zum und vom Computer.

Mit den LEDs D2 bis D5 wird der Signalpegel der Funkschnittstelle angezeigt.

Das Modul besteht aus einem ER400TRS Modul und einem FT232BL Chip. Diese beiden Bausteine wurden bereits in Kapitel 7.1 und 7.2 näher beschrieben. Das Modul wird über die USB-Schnittstelle mit 5 V versorgt. Auf dem Modul befindet sich auch ein EEPROM mit einer voreingestellten VID und PID. Das bringt den Vorteil, dass das Modul nur an einem USB-Port des Computers installiert werden muss und dann an allen anderen USB-Ports des PCs verwendet werden kann. Wenn diese IDs fehlen, müsste das Modul für jeden USB-Port separat installiert werden.

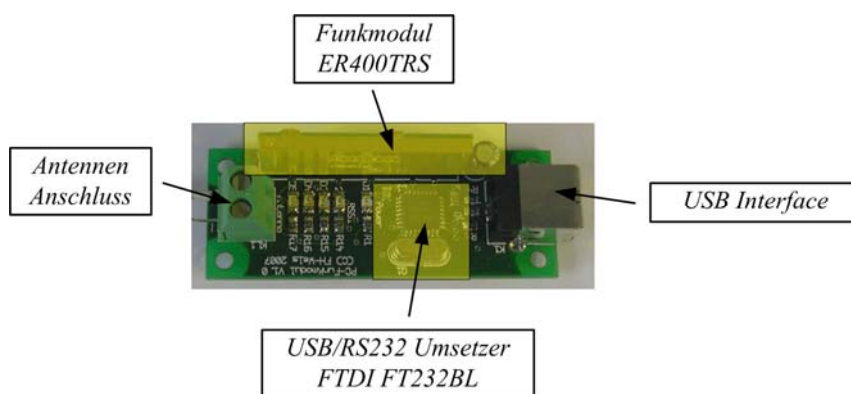


Abbildung 7.5 Bestückte Leiterplatte des USB/Funk Konverters

In Tabelle 7.7 und Tabelle 7.8 sind die Anschlussbelegung des USB-Ports und des Antennenanschlusses näher beschrieben. Wie die Länge der Antenne berechnet wird, kann aus Kapitel 7.3 entnommen werden.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	Pluspol der Versorgung des Konverters
2	USBDM	Invertierte Daten
3	USBDP	Nicht invertierte Daten
4	GND	Minuspole der Versorgung des Konverters

Tabelle 7.7 Steckerbelegung von K1 (USB-Port)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	ANT	Antennen Anschluss
2	GND	Minusanschluss der Antenne

Tabelle 7.8 Steckerbelegung von KL1 (Antennen Anschluss)

8. PROGRAMMIER-BOARD

Um alle im Roboter befindlichen Module von außen programmieren zu können, wurde das Programmier-Board entwickelt (siehe Abbildung 8.1). Dieses Board besteht aus 16 Steckern. Wobei diese 16 Stecker paarweise verbunden sind. 8 Stecker führen dabei ins Roboterinnere zu den einzelnen Boards und 8 Stecker sind von außen für das Programmier-Interface zugänglich.



Abbildung 8.1 Bestückte Leiterplatte des Programmier Boards (PC-Interface)

Während des Programmierens der Slave-Boards muss das Master Board inaktiv geschaltet werden. Daher muss das Master-Board auf die Anschlüsse K1/K2 (siehe Tabelle 8.1 und Tabelle 8.2) angeschlossen werden. Denn an diesen Steckern kann die Resetleitung des Masters gegen Low-Pegel gezogen werden.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	Pluspol der Versorgung für das Programmier-Board
2	RxD	Empfangsleitung der Debug-Schnittstelle
4	SCK	Clock-Leitung der Programmierschnittstelle
5	TxD	Sendeleitung der Debug-Schnittstelle
6	MISO	Master In Slave Out der Programmier-Schnittstelle
8	MOSI	Master Out Slave In der Programmier-Schnittstelle
9	GND	Minuspole der Versorgung für das Programmier-Board
10	/Reset	Reset-Leitung der Programmier-Schnittstelle
3, 7	NC	nicht verwendet

Tabelle 8.1 Steckerbelegung von K1 (Prog.-IF des Masters zum PC)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	Pluspol der Versorgung für das Programmier-Board
2	RxD	Empfangsleitung der Debug-Schnittstelle
4	SCK	Clock-Leitung der Programmierschnittstelle
5	TxD	Sendeleitung der Debug-Schnittstelle
6	MISO	Master In Slave Out der Programmier-Schnittstelle
8	MOSI	Master Out Slave In der Programmier-Schnittstelle
9	GND	Minuspole der Versorgung für das Programmier-Board
10	/Reset	Reset-Leitung der Programmier-Schnittstelle
3, 7	NC	nicht verwendet

Tabelle 8.2 Steckerbelegung von K2 (Prog.-IF zum Master)

Die Slave Boards können an die restlichen Stecker angeschlossen werden. Die Steckerbelegung ist in Tabelle 8.3 und Tabelle 8.4 dargestellt.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	Pluspol der Versorgung für das Programmier-Board
2	RxD	Empfangsleitung der Debug-Schnittstelle
4	SCK	Clock-Leitung der Programmierschnittstelle
5	TxD	Sendeleitung der Debug-Schnittstelle
6	MISO	Master In Slave Out der Programmier-Schnittstelle
8	MOSI	Master Out Slave In der Programmier-Schnittstelle
9	GND	Minuspole der Versorgung für das Programmier-Board
10	/Reset	Reset-Leitung der Programmier-Schnittstelle
3, 7	NC	nicht verwendet

Tabelle 8.3 Steckerbelegung von K3, K5, K7, K9, K11, K13 und K15 (Prog.-IF zum PC)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+5V	Pluspol der Versorgung für das Programmier-Board
2	RxD	Empfangsleitung der Debug-Schnittstelle
4	SCK	Clock-Leitung der Programmierschnittstelle
5	TxD	Sendeleitung der Debug-Schnittstelle
6	MISO	Master In Slave Out der Programmier-Schnittstelle
8	MOSI	Master Out Slave In der Programmier-Schnittstelle
9	GND	Minuspole der Versorgung für das Programmier-Board
10	/Reset	Reset-Leitung der Programmier-Schnittstelle
3, 7	NC	nicht verwendet

Tabelle 8.4 Steckerbelegung von K4, K6, K8, K10, K12, K14 und K16 (Slaves Prog.-IF)

9. VCC-BOARD

Die Aufgabe des in Abbildung 9.1 dargestellten VCC-Verteiler Boards ist es eine geschaltete Akkuspannung für die im Roboter befindlichen Boards bereitzustellen.

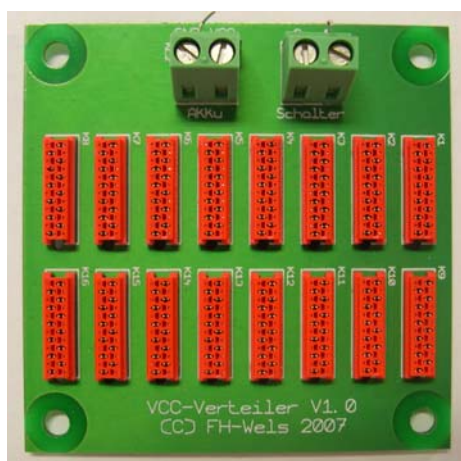


Abbildung 9.1 Bestückte Leiterplatte des VCC-Verteiler Boards

Das VCC-Verteiler Board bietet Anschlüsse für 16 Module. Die Anschlussbelegung ist Tabelle 9.1 zu entnehmen.

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1,3,5,7,9,11	+12V	Pluspol der Versorgung
2,4,6,8,10,12	GND	Minuspole der Versorgung

Tabelle 9.1 Steckerbelegung von K1 bis K16 (Power-Supply)

In Abbildung 9.2 ist das Prinzipschaltbild des VCC-Verteiler Boards dargestellt. Wie ersichtlich ist, wird über KL2 (siehe Tabelle 9.3) der Akku angeschlossen, der über KL1 (siehe Tabelle 9.2) und dem Hauptschalter an die Ausgänge weitergeschaltet wird.

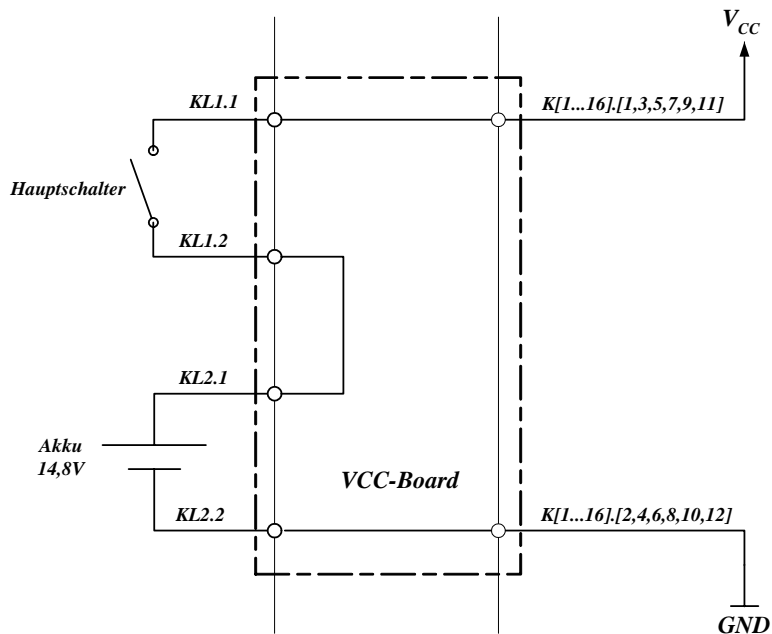


Abbildung 9.2 Prinzipschaltbild des VCC-Verteiler Boards

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	SWITCH 1	Schalteingang 1
2	SWITCH 2	Schalteingang 2

Tabelle 9.2 Steckerbelegung von KL1 (Hauptschalter)

<i>Pin Nummer</i>	<i>Pin Bezeichnung</i>	<i>Funktion</i>
1	+12V	Pluspol des Akkus
2	GND	Minuspole des Akkus

Tabelle 9.3 Steckerbelegung von KL2 (Akku Anschluss)

10. WEITERFÜHRENDE ARBEITEN

Ein umfangreiches Projekt, wie das modulare elektronische System, bietet immer wieder Möglichkeiten neu Teilgebiete zu bearbeiten bzw. schon fertige Module weiter zu verbessern oder eventuelle auftretende Schwachstellen zu beseitigen.

Im Folgenden sind einige Ideen für neue weiterführende Projekte, wie auch Verbesserungen und Erweiterungen des aktuellen Systems kurz angerissen. Diese Liste kann natürlicherweise nicht vollständig sein, da neue Projekte in der Robotik wahrscheinlich weiter Leistungsmerkmale von einem elektronischen System fordern werden, die aus heutiger Sicht noch nicht zu erkennen sind.

10.1. MOTOR-BOARD

Ein Nachteil des Motor-Boards ist, dass der Leistungstreiber BTS7750GP nur mit einer maximalen Frequenz von 1 kHz angesteuert werden kann. Dadurch befindet sich nicht nur die Grundfrequenz sondern auch die ersten Oberwellen im hörbaren Bereich, was ein pfeifendes Geräusch bei der Ansteuerung der Motoren zur Folge hat. Eine Aufgabe ist es daher, einen Leistungstreiber mit einer höheren Schaltfrequenz zu finden.

Ein weiterer Nachteil des Motor-Boards ist es, dass die Encoder-Auswertung am Microcontroller in Software stattfindet und bei hohen Drehzahlen eine hohe Rechenleistung in Anspruch nimmt. Es sollte daher nach einem anderen Prozessor gesucht werden, der die Encoder-Auswertung in Hardware realisiert hat wie z.B. die dsPIC-Familie der Fa. Microchip. Diese Auswertung könnte jedoch extern von einem CPLD bzw. FPGA übernommen werden.

Weiters sollten der Drehzahl- wie auch der Positionsregler neu überarbeitet werden und eventuell als PI- bzw. Zustandsregler implementiert werden. Darauf aufsetzend könnten auch komplexe Pfadplanungsalgorithmen in MATLAB entwickelt werden und diese auf den Microcontroller implementiert werden.

10.2. SPI HUB

Da es für komplexere Anwendungen sein könnte, dass 14 Slave-Schnittstellen am Main-Board nicht ausreichen, sollte ein SPI-Hub entwickelt werden. Dieser Hub kann sowohl passiv als auch aktiv sein, d.h. ein passiver Hub leitet die Meldungen an die Teilnehmer weiter ohne aktiv in den Prozess einzugreifen, ein aktiver Hub greift dagegen in den Meldungsprozess ein. Ein aktiver Hub erhält z.B. Teilaufgaben vom Main-Board übertragen, die wiederum auf an ihm angeschlossene Slaves verteilt werden.

10.3. POSITIONSERKENNUNG

Die aktuelle Positionserkennung arbeitet mit zwei unabhängigen Inkrementgebern, die jeweils an einem Laufrad am Roboter montiert sind. Dadurch wird die Berechnung der Position von den Antriebsmotoren unabhängig. Diese Berechnung erzeugt keinen Fehler wenn auf den Räder ein Schlupf auftritt.

Ein Nachteil bei der bestehenden Positionserkennung ist es jedoch, dass es bei Kollisionen mit anderen Robotern zu Fehlmessungen kommen kann, wenn der Roboter z.B. quer zu den Laufrädern verschoben wird. Daher sollte ein den Boden berührungsloses Messsystem aufgebaut werden, das keine Vorzugsrichtung, wie ein Messrad, besitzt. Ein solches Messsystem könnte z.B. mittels eines Maus-Sensors realisiert werden.

10.4. EINFACHE BILDVERARBEITUNG

Bei vielen Roboter-Bewerben ist es von Vorteil wenn der Roboter über eine einfache Objekterkennung verfügen würde. Diese Objekterkennung könnte mit preisgünstigen Bildverarbeitungs-Modulen, wie z.B. dem Modul Pob-EYE²⁷ oder CMUcam3²⁸, realisiert werden. Bei diesen Modulen werden bereits viele Bildverarbeitungs-Funktionen mitgeliefert und es ist möglich den Sourcecode an die eigene Anwendung anzupassen.

²⁷ [Pob-Technology, 2007]

²⁸ [CMUcam, 2008]

10.5. LCD-MODUL

In der aktuellen Version des LCD-Moduls hat das Main-Board keine Möglichkeit auf dem LCD-Display Meldungen auszugeben, d.h. unter anderem, dass die gesamte Menüverwaltung auf dem LCD-Modul realisiert werden muss. Daraus folgt allerdings, dass die Software für das LCD-Modul für jeden Bewerb angepasst werden muss.

Eine weitere Möglichkeit wäre es wenn die LCD-Ausgabe Funktionen, wie in Kapitel 6.2.1 näher beschrieben, über die SPI-Schnittstelle dem Main-Board zugänglich gemacht werden würden. Dadurch könnte die gesamte Menüführung des LCD-Moduls am Main-Board abgearbeitet werden, und die Software des LCD-Moduls bräuchte nicht mehr geändert werden.

11. ZUSAMMENFASSUNG

Das modulare elektronische System zum Bau autonomer Roboter besteht aus drei verschiedenen Grundmodulen und zwei Zusatzmodulen. Zu den Grundmodulen zählen das Main-Board, das Sensor/Servo-Board und das Motor-Board, zu den Zusatzmodulen gehören das Kommunikations-Board (Umsetzer Seriell → Funk und Funk → USB) und das LCD-Modul.

Mit den Grundmodulen kann ein vollständiger Roboter mit dessen kompletter Sensorik und Aktorik aufgebaut werden. Es wurden bereits zahlreiche Sensoren, Servos und viele Motorbefehle auf den verschiedenen Boards implementiert. Um komplexe Aufgaben einfach und elegant zu lösen, wurde für das Main-Board ein schlankes Multitasking System entworfen. Dadurch können die verschiedensten Teilprogramme quasiparallel abgearbeitet werden, ohne dass das Programm an Übersichtlichkeit verliert.

Die Zusatzmodule ermöglichen es, eine einfache Mensch-Maschinen-Schnittstelle zu realisieren. Es können mittels des Funkmoduls einfache Statusmeldungen zu einem Host-PC gesandt werden, oder der Roboter kann mit einfachen Befehlen ferngesteuert werden, was gerade in der Inbetriebnahmephase von größter Bedeutung sein kann.

Das LCD-Modul ermöglicht dem Anwender vor Ort am Roboter Änderungen durchzuführen, ohne dass das Programm geändert werden muss, oder eine Verbindung zu einem Host-PC nötig ist. Außerdem können wichtige Statusmeldungen dem Anwender zugänglich gemacht werden.

Das vorgestellte System bietet eine Menge an Möglichkeiten Roboter in einer relativ kurzen Zeit zu bauen, bietet allerdings noch ein weites Betätigungsfeld für Adaptionen und neue Projekte.

12. LITERATUR

12.1. BÜCHERLISTE

- [Dembowski, 1997] Dembowski, Klaus: Computerschnittstellen und Bussysteme, Hüthig Verlag Heidelberg, 1997
- [FUP RobotChallenge, 2006] Brandsteidl Bernhard, Buchinger Andreas, Buchinger Manuel, Edlinger Raimund, Niederkofler Christian: Fachübergreifende Projektarbeit Roboter, 2006
- [Berufspraktikum, 2006] Brandsteidl Bernhard, Edlinger Raimund: Berufspraktikumsbericht, 2006
- [Bräunl, 2003] Bräunl, Thomas: Embedded Robotics, Springer Verlag Berlin Heidelberg, 2003
- [hpinfotech, 2005] Haiduc, Pavel: Hilfe für CodeVisionAVR C-Compiler, HP InoTech s.r.l., 2005

12.2. INTERNET

- [robotikhardware, 2008] <http://www.shop.robotikhardware.de/>, Stand vom 14.10.2008
- [Sharp GP2D12, 2007] <http://www.sharpsme.com/>, Stand vom 16.08.2007
- [Sharp GP2D120, 2008] <http://www.sharpsme.com/>, Stand vom 20.10.2008
- [Sharp GP2Y0A02F, 2008] <http://www.sharpsme.com/>, Stand vom 20.10.2008
- [National Semiconductor, 2008] <http://www.national.com/analog/power>,
Stand vom 07.11.2008
- [National Semiconductor LM2594, 2002] <http://www.national.com/analog/power>,
Stand vom 05.03.2002
- [National Semiconductor LM2677, 2008] <http://www.national.com/analog/power>,
Stand vom 02.10.2008
- [roboterteile SRF10, 2008] <http://www.roboter-teile.de/Shop/index.php>,
Stand vom 15.09.2008
- [Atmel, 2006] <http://www.atmel.com/products/avr/default.asp>, Stand vom 07.09.2006
- [axe045, 2008] <http://www.rev-ed.co.uk/docs/axe045.pdf>, Stand vom 27.10.2008
- [Easy-Radio, 2007] <http://www.roboter-teile.de/Shop/index.php>, Stand vom 24.08.2007

- [FTDI, 2004] <http://www.ftdichip.com/FTProducts.htm>, Stand vom 10.05.2004
- [RobotChallenge, 2008] <http://www.robotchallenge.at/>, Stand vom 06.11.2008
- [austrobot, 2008] <http://austrobot.info/>, Stand vom 07.11.2008
- [robogames, 2008] <http://www.robogames.net/>, Stand vom 07.11.2008
- [Bluebird, 2008] <http://www.blue-bird-model.com/>, Stand vom 11.11.2008
- [HiTech, 2008] <http://www.hitecrd.com/>, Stand vom 25.08.2008
- [Infineon, 2008] <http://www.infineon.com/>, Stand vom 26.10.2006
- [Faulhaber, 2006] <http://www.faulhaber-group.de/>, Stand vom 06.09.2006
- [Pob-Technology, 2007] <http://www.pob-technology.com/>, Stand vom 16.01.2007
- [CMUcam, 2008] <http://www.cmucam.org/>, Stand vom 18.11.2008
- [Displaytech, 2007] <http://www.displaytech.com.hk/>, Stand vom 13.09.2007

13. ANHANG

13.1. PROGRAMMTEILE MOTOR-BOARD

```

// *****
// ****          MOTOR 1          ****
// *****
// wenn der Motor sich zurückdreht beim Drehzahlsollwert eine Vorzeichen-
// korrektur durchführen
if(bDirectionMotor1 == BACKWARD)
{
    uiSpeed = ((signed int)(ucSpeedMotor1)) * (-1);
}
// ansonsten Sollwert übernehmen
else
{
    uiSpeed = (signed int)(ucSpeedMotor1);
}

// Reglerabweichung bestimmen
// Reglerabweichung = (Sollwert - Istwert) * k
siReglerAbweichung = (uiSpeed - siGetRegler1Value()) * (signed
int)(ucPAnteil);

// positiver Reglerabweichung
if(siReglerAbweichung > 0)
{
    // die Richtung auf "Vorwärts" stellen
    // bei pos. Sollwert: z.B. 50 - 30 = 20 d.h. Drehzahl zu klein ->
    // nach vorwärts fahren (beschleunigen)
    // bei neg. Sollwert: z.B. -50 - (-60) = 10 d.h. Drehzahl zu groß ->
    // Motor bremsen also Richtungswechsel durchführen
    ucDirection = FORWARD;

    // Reglerbegrenzung: wenn der Wert über 255 ist
    if(siReglerAbweichung >= 255)
    {
        siReglerAbweichung = 255;
    }
}
// negative Reglerabweichung
else
{
    // die Richtung auf "Rückwärts" stellen
    // bei pos. Sollwert: z.B. 50 - 60 = -10 d.h. Drehzahl zu groß ->
    // Motor bremsen also Richtungswechsel durchführen
    // bei neg. Sollwert: z.B. -50 - (-30) = -20 d.h. Drehzahl zu klein ->
    // nach rückwärts fahren (beschleunigen)
    ucDirection = BACKWARD;

    // Reglerbegrenzung: wenn der Wert über 255 ist
    if(abs(siReglerAbweichung) >= 255)
    {
        siReglerAbweichung = 255;
    }
}

```

```

// ansonsten Absolutwert der Reglerabweichung erzeugen
else
{
    siReglerAbweichung = abs(siReglerAbweichung);
}
}
// Werte für Richtung und PWM am Motor setzen
SetMotor1Regler(ucDirection, (unsigned char)(siReglerAbweichung));
    
```

Listing 13.1 Drehzahlregler des 1. Motors

```

//*****
//*****
//****          MOTOR 1          ****
//*****
//*****
if(ucMotorNbr == 1)
{
    ucPWMValue1 = ucSpeed;
    //*****
    //*****
    //****          SOLLWERT SETZEN          ****
    //*****
    //*****
    // bei Vorwärtsrichtung Sollwert positiv übergeben
    if(ucDirection == FORWARD)
    {
        siDistancel = (signed int) (uiSteps);
    }
    // bei Rückwärtsrichtung Sollwert negativ übergeben
    else
    {
        siDistancel = (signed int) (uiSteps) * (-1);
    }

    //*****
    //*****
    //****          POSITIONSREGLER DER BEIDEN MOTOREN          ****
    //*****
    //*****
    // MOTOR 1
    // Reglerabweichung bestimmen (Sollwert - Istwert)
    siReglerAbweichung = siDistancel - siGetMotor1Value();
    // Stellgröße ermitteln (P-Regler)
    siStellGroessel = (siReglerAbweichung * KP);

    // bei einer negativen Stellgröße muss die Motorrichtung umgekehrt werden
    // (da Istwert > Sollwert)
    if(siStellGroessel < 0)
    {
        // Betrag der Stellgröße bilden
        siStellGroessel = siStellGroessel * (-1);
        // Richtung auf Rückwärts stellen
        ucDir1 = BACKWARD;
    }
    else
    {
        // Richtung auf Vorwärts stellen
    }
}
    
```

```

        ucDir1 = FORWARD;
    }

    //*****
//*****
    //****          AUSGABEWERT BEGRENZEN UND MOTOREN ANSTEUERN
    ****

//*****

//*****
    // MOTOR 1
    // Stellgröße begrenzen
    if(siStellGroessel > (signed long)(ucPWMValue1)) siStellGroessel = (signed
long)(ucPWMValue1);

    // MOTOR 1
    // Motor ansteuern
    SetMotor1(ucDir1, (unsigned char)(siStellGroessel));

    // wenn Bewegung abgeschlossen ist
    // -> MOTION_READY zurückgeben
    if(abs(siGetMotor1Value()) == uiSteps)
    {
        if(!ucDisableResetMotorCheck) ucMotorCheck1 = MOTOR_CHECK_OFF;
        return(MOTION_READY);
    }
}

```

Listing 13.2 Positionsregler des 1. Motors

```

/*****
*** Funktionsname:      SetStepper          ***
*** Funktion:          steuert Schrittmotor an          ***
*** Übergabe-Para.:   ucDirection ... Richtungsangabe (0 ... rechts) ***
***                   uiSteps ... Anzahl der Schritte (max. 65532) ***
***                   ucSpees ... Wartezeit zw. Schr. = ucSpeed*1ms ***
*** Rückgabe-Para.:   Keine                ***
*** Erstellt:         Zauner Michael (20-12-2006)          ***
*** Änderungen:      ***
*****/
void SetStepper(unsigned char ucDirection, unsigned int uiSteps, unsigned char ucSpeed)
{
    unsigned char ucPause = 0;

    // bei Richtungswechsel muss noch ein Phasenkorrektor bezüglich der Phase
    // gemacht werden
    if(ucLastDirection != ucDirection)
    {
        if(uiStepStateOld == 0)
        {
            uiStepStateOld = 4;
        }
        else if(uiStepStateOld == 1)
        {
            uiStepStateOld = 2;
        }
    }
}

```

```

else if(uiStepStateOld == 2)
{
    uiStepStateOld = 1;
}
else if(uiStepStateOld == 3)
{
    uiStepStateOld = 0;
}
}

// Um beim letzten Schritt weiterzumachen, wird die zuletzt angesteuerte
// Phase reaktiviert
uiStepState = uiStepStateOld;

// Schritte ausgeben (solange bis alle Schritte ausgegeben werden oder bis das
// Kommando auf Stop gesetzt wird)
while((uiStepState < (uiSteps + uiStepStateOld)) && (ucCommandToGo != MB_STOP_MSG))
{
    // bei Linkslauf gilt die Phasenfolge A -> D -> B -> C
    if(ucDirection == FORWARD_STEPPER)
    {
        if((uiStepState % 4) == 0)
        {
            SET_PHASE_A;
            ucLastPhase = PHASE_A;
        }
        else if((uiStepState % 4) == 1)
        {
            SET_PHASE_D;
            ucLastPhase = PHASE_D;
        }
        else if((uiStepState % 4) == 2)
        {
            SET_PHASE_B;
            ucLastPhase = PHASE_B;
        }
        else if((uiStepState % 4) == 3)
        {
            SET_PHASE_C;
            ucLastPhase = PHASE_C;
        }
    }
    // bei Rechtslauf gilt die Phasenfolge C -> B -> D -> A
    else
    {
        if((uiStepState % 4) == 0)
        {
            SET_PHASE_C;
            ucLastPhase = PHASE_C;
        }
        else if((uiStepState % 4) == 1)
        {
            SET_PHASE_B;
            ucLastPhase = PHASE_B;
        }
        else if((uiStepState % 4) == 2)
        {
            SET_PHASE_D;
            ucLastPhase = PHASE_D;
        }
        else if((uiStepState % 4) == 3)
        {

```

```

        SET_PHASE_A;
        ucLastPhase = PHASE_A;
    }
}
uiStepState++;
// Pause erzeugen (ucSpeed * 1ms)
while(ucPause++ < ucSpeed)
{
    delay_ms(1);
}

CLR_PHASE_A;
CLR_PHASE_B;
CLR_PHASE_C;
CLR_PHASE_D;

ucPause = 0;
}

// letzte Richtung speichern (wird ev. für Phasenkorrektur bei nächster
// Ansteuerung benötigt)
ucLastDirection = ucDirection;
// zuletzt angesteuerte Phase zwischenspeichern (um beim Nächstenmal keinen
// Schritt zu verlieren)
uiStepStateOld = (uiStepState % 4);
// nach Ende der Ansteuerung alle Phasen wieder auf null legen um Strom zu sparen
CLR_PHASE_A;
CLR_PHASE_B;
CLR_PHASE_C;
CLR_PHASE_D;
}

```

Listing 13.3 Unterprogramm zur Ansteuerung eines Schrittmotors

13.2. PROGRAMMTEILE SENSOR-BOARD

```

/*****
***  FUNCTIONNAME:      main                ***
***  FUNCTION:         Hauptprogramm        ***
***  TRANSMIT-PARAMETER: NO                ***
***  RECEIVE-PARAMETER: NO                ***
*****/
void main(void)
{
    // Index zum Auslesen der Sensoren
    unsigned char ucSensorIndex = 0;

    // µC initialisieren
    InitDevice();

    // Main-Loop
    while(1)
    {
        // Index auf Sensor erhöhen
        ucSensorIndex++;
        if(ucSensorIndex > 15)
        {
            ucSensorIndex = 0;
        }
        // Sensor abarbeiten
        ReadSensor(ucSensorIndex);
    }
}

```

Listing 13.4 Main-Loop des Sensorboards

```

/*****
***  FUNCTIONNAME:      ReadSensor      ***
***  FUNCTION:         zur Erfassung aller Sensordaten      ***
***  TRANSMIT-PARAMETER: NO          ***
***  RECEIVE-PARAMETER: i ... Index auf Sensor          ***
*****/
void ReadSensor(unsigned char i)
{
    // ****
    // ***              Kommando: "Init"              ***
    // ****
    if(ucSensorType[i] == SB_NOT_INIT)
    {
        // ****
        // ***              Kommando: "SB_GP2D120"              ***
        // ****
        else if(ucSensorType[i] == SB_GP2D120)
        {
            uiDistanceMm[i] = IRsensor_gp2d120(i);
        }

        // ****
        // ***              Kommando: "SB_GP2D12"              ***
        // ****
        else if(ucSensorType[i] == SB_GP2D12)
        {
            uiDistanceMm[i] = IRsensor_gp2d12(i);
        }

        // ****
        // ***              Kommando: "AXE045"              ***
        // ****
        else if(ucSensorType[i] == SB_AXE045)
        {
            uiDistanceMm[i] = uiColourSensorAXE045(i);
        }

        // ****
        // ***              Kommando: "BUMPER"              ***
        // ****
        else if(ucSensorType[i] == SB_DIGITAL)
        {
            uiDistanceMm[i] = (((!(i & 0x08) && (PINA & (0x01 << (i & 0x07)))) ||
                ((i & 0x08) && (PINC & (0x01 << (i & 0x07)))))) ? 1 : 0);
        }

        // ****
        // ***              Kommando: "GP2Y0A02F"              ***
        // ****
        else if(ucSensorType[i] == SB_GP2Y0A02F)
        {
            uiDistanceMm[i] = IRsensor_gp2y0a02f(i);
        }

        // ****
        // ***              Kommando: "SRF08"              ***
        // ****
        else if(ucSensorType[i] == SB_SRF08)
        {
            StartMeasurmentSRF08(i);
            DelaySRF08(100);
        }
    }
}

```

```

        uiDistanceMm[i] = uiReadSRF08(i);
    }
}

```

Listing 13.5 Unterprogramm zum Abarbeiten der Sensoren

```

/*****
***   FUNKTIONNAME:           InitSRF08           ***
***   FUNKTION:              initialisiert den SRF08/10   ***
***                           - Messbereichseinschränkung auf 1m   ***
***                           - Verstärkungsfaktor auf 1         ***
***   TRANSMIT PARAMETER:    NO                   ***
***   RECEIVE PARAMETER.:    ucSensorNumber        ***
*****/
void InitSRF08(unsigned char ucSensorNumber, unsigned int uiDelay)
{
    // ****           I2C-Schnittstelle initialisieren           ****
    // ****           Reichweite auf 1m beschränken               ****
    InitI2C(ucSensorNumber);

    // ****           Delay zwischen "start Messung" und "Messung auslesen"   ****
    // ****           initialisieren                               ****
    // ****           Reichweite auf 1m beschränken               ****
    uiMeasurmentDelay[ucSensorNumber] = uiDelay;

    // ****           Verstärkungsfaktor einstellen               ****
    StartI2C(ucSensorNumber);
    // Adresse des Moduls ausgeben (0xE0)
    ucWriteByte(0xE0, ucSensorNumber);
    // Weitenregister anwählen (0x02)
    ucWriteByte(0x02, ucSensorNumber);
    // Wert für Weitenbeschränkung ausgeben x = (Weite[mm] - 43mm) / 43mm
    // 0x18 <-> 1m
    ucWriteByte(0x18, ucSensorNumber);
    StopI2C(ucSensorNumber);

    // ****           Verstärkungsfaktor einstellen               ****
    StartI2C(ucSensorNumber);
    // Adresse des Moduls ausgeben (0xE0)
    ucWriteByte(0xE0, ucSensorNumber);
    // Verstärkungsregister anwählen (0x01)
    ucWriteByte(0x01, ucSensorNumber);
    // Verstärkung ausgeben (je kleiner die Verstärkung, desto eingeschränkter
    // ist die Sensorkeule!)
    ucWriteByte(0x03, ucSensorNumber);
    StopI2C(ucSensorNumber);
}

```

Listing 13.6 Unterprogramm „SRF08/10 initialisieren“

```

/*****
***  FUNKTIONNAME:          StartMeasurmentSRF08          ***
***  FUNKTION:             startet den Messvorgang SRF08/10 ***
***  TRANSMIT PARAMETER:   NO                          ***
***  RECEIVE PARAMETER.:   ucSensorNumber              ***
*****/
void StartMeasurmentSRF08(unsigned char ucSensorNumber)
{
    StartI2C(ucSensorNumber);
    // Adresse des Moduls ausgeben (0xE0)
    ucWriteByte(0xE0, ucSensorNumber);
    // Befehlsregister anwählen (0x00)
    ucWriteByte(0x00, ucSensorNumber);
    // Befehl "Messung auslösen - Ergebnis in cm" (0x51) ausgeben
    ucWriteByte(0x51, ucSensorNumber);
    StopI2C(ucSensorNumber);
}

```

Listing 13.7 Unterprogramm „Messung starten“

```

/*****
***  FUNKTIONNAME:          uiReadSRF08                  ***
***  FUNKTION:             liest den Sensor aus (Ergebnis in mm) ***
***  TRANSMIT PARAMETER:   Entfernung in mm (jedoch nur auf cm genau) ***
***  RECEIVE PARAMETER.:   ucSensorNumber              ***
*****/
unsigned int uiReadSRF08(unsigned char ucSensorNumber)
{
    unsigned int uiErgebnis = 0;

    // **** 1. Startsequenz: um das Ausgaberegister auszuwählen ****
    // ****
    StartI2C(ucSensorNumber);
    // Adresse des Moduls ausgeben (0xE0)
    ucWriteByte(0xE0, ucSensorNumber);
    // Ausgaberegister auszuwählen (da nur bis 100cm gemessen wird nur das
    // LowByte auslesen --> 0x03, HighByte ist Register 0x02)
    ucWriteByte(0x03, ucSensorNumber);

    // **** 2. Startsequenz: um das Register zu lesen ****
    // ****
    StartI2C(ucSensorNumber);
    // Adresse des Moduls ausgeben (0xE0) mit Lesebefehl kombiniert (LSB = 1)
    ucWriteByte(0xE1, ucSensorNumber);
    // Ergebnis lesen und kein ACK ausgeben
    uiErgebnis = (unsigned int)(ucReadByte(NO_ACK, ucSensorNumber));
    StopI2C(ucSensorNumber);

    // Messwert filtern (für den SRF08/10 Sensor werden die Filteradressen
    // 8 ... 15 verwendet --> um Adresskonflikte mit den ADC-Kanälen zu verhindern)
    uiErgebnis = uiFilter((ucSensorNumber + 8), uiErgebnis);

    // Ergebnis in mm umrechnen und zurückgeben
    return(uiErgebnis * 10);
}

```

Listing 13.8 Unterprogramm „SRF08/10 auslesen“

```

/*****
***  FUNKTIONNAME:          uiFilter          ***
***  FUNKTION:             filtert den übergebenen ADC-Wert          ***
***  TRANSMIT PARAMETER:   gefilterter Wert          ***
***  RECEIVE PARAMETER.:   ucFilterNumber ... Kanalnummer          ***
***                      uiAdcValue ... übergebener ADC-Wert          ***
*****/
unsigned int uiFilter(unsigned char ucFilterNumber, unsigned int
uiAdcValue)
{
    // Variablen werden zur Berechnung benötigt
    unsigned int y;

    // alten Filter-Wert auslesen
    y = uiFilterValue[ucFilterNumber];

    // neuer FilterWert = alter FilterWert - alter AusgabeWert + ADCWert
    y = (y - uiOut[ucFilterNumber]) + uiAdcValue;

    // Filter-Wert speichern
    uiFilterValue[ucFilterNumber] = y;

    // neuer AusgabeWert = neuer FilterWert / FILTER_COEFFIZIENT
    y >>= ucFilterCoeffizient[ucFilterNumber];

    // Ausgabe-Wert speichern
    uiOut[ucFilterNumber] = y;

    return(y);
}

```

Listing 13.9 Unterprogramm „digitales Filter 1. Ordnung“

```

/*****
***  FUNKTIONNAME:          uiFilter          ***
***  FUNKTION:             filtert den übergebenen ADC-Wert          ***
***  TRANSMIT PARAMETER:   gefilterter Wert          ***
***  RECEIVE PARAMETER.:   ucFilterNumber ... Kanalnummer          ***
***                      uiAdcValue ... übergebener ADC-Wert          ***
*****/
unsigned int uiFilter(unsigned char ucFilterNumber, unsigned int uiAdcValue)
{
    // Variablen werden zur Berechnung benötigt
    unsigned int y;
    static unsigned int uiAdcValueOld;

    // alten Filter-Wert auslesen
    y = uiFilterValue[ucFilterNumber];

    // Filterzwischenwert = alter FilterWert - alter AusgabeWert
    y = y - uiOut[ucFilterNumber];

    // wenn es zu einer Änderung von min. 5% gekommen ist
    if(abs(uiAdcValueOld - uiAdcValue) > (uiAdcValue / 20))
    {
        // Filterkoeffizient auf 6 setzen und Filterwert anpassen
        while(ucFilterCoeffizient[ucFilterNumber] != 6)
        {
            ucFilterCoeffizient[ucFilterNumber]++;
            y <<= 1;
        }
    }
    // solange der Filterkoeffizient größer 1 ist Filterwert anpassen
    else if(ucFilterCoeffizient[ucFilterNumber] >= 2)
    {
        ucFilterCoeffizient[ucFilterNumber]--;
        y >>= 1;
    }

    // neuer FilterWert = Filterzwischenwert + ADCWert
    y += uiAdcValue;

    // Filter-Wert speichern
    uiFilterValue[ucFilterNumber] = y;

    // neuer AusgabeWert = neuer FilterWert / FILTER_COEFFIZIENT
    y >>= ucFilterCoeffizient[ucFilterNumber];

    // Ausgabe-Wert speichern
    uiOut[ucFilterNumber] = y;

    uiAdcValueOld = uiAdcValue;

    return(y);
}

```

Listing 13.10 Unterprogramm „adaptives Filter“

13.3. PROGRAMMTEILE LCD-MODUL

```

*****
*** Funktionsname:   HandleKey
***
*** Funktion:       bearbeitet Taster
*** Übergabe-Para.: psKey ... Pointer auf Taster
***                ucKey ... Tastenwert
***                (0 -> nicht gedrückt, !0 ... gedrückt)
*** Rückgabe-Para.: Keine
*** Erstellt:      Zauner Michael (15-11-2006)
***
*** Änderungen:
***
*****/
void HandleKey(tsKey* psKey, unsigned char ucKey)
{
    // überprüfen ob Taste gedrückt ist
    if(ucKey)
    {
        // Entprellzeit realisieren
        if(psKey->ucFailureCounter <= 5)
        {
            psKey->ucFailureCounter++;
        }
        // nach Entprellzeit den Status des Tasters entsprechen setzen
        else
        {
            // beim erstmal Drücken erkennen
            // -> steigende Flanke ausgeben
            // -> fallende Flanke rücksetzen
            if(!(psKey->ucStatus & PRESSED))
            {
                psKey->ucStatus |= RISING_AGE;
                psKey->ucStatus &= ~FALLING_AGE;
            }
            // Status auf gedrückt setzen
            psKey->ucStatus |= PRESSED;

            // die Zeit wie lange die Taste gedrückt ist ermitteln
            // (max. Wert 60001)
            if(psKey->uiPressTime <= 60000)
            {
                psKey->uiPressTime++;
            }
        }
    }
    // bei nicht gedrückter Taste
    else
    {
        // Entprellzeit realisieren
        if(psKey->ucFailureCounter > 0)
        {
            psKey->ucFailureCounter--;
        }
        // nach Entprellzeit den Status des Tasters entsprechen setzen
        else
        {
            // beim erstmal Loslassen erkennen
            // -> steigende Flanke rücksetzen

```

```
// -> fallende Flanke setzen
if(psKey->ucStatus & PRESSED)
{
    psKey->ucStatus &= ~RISING_AGE;
    psKey->ucStatus |= FALLING_AGE;
}
// Status auf nicht gedrückt setzen
psKey->ucStatus &= ~PRESSED;
// Drückzeit resettieren
psKey->uiPressTime = 0;
}
}
```

Listing 13.11 Unterprogramm zur Behandlung von Tastern