HOCHSCHULE
MITTWEIDA
UNIVERSITY OF
APPLIED SCIENCES

Bachelorarbeit

Herr

Christian Mahner

Konzeption und Aufbau einer Smart Home Anwendung, bestehend aus Kommunikationssoftware und embedded Software unter Verwendung des NXP LPC1768

Mittweida, 2015

Fakultät Elektro- und Informationstechnik

Bachelorarbeit

Konzeption und Aufbau einer Smart Home Anwendung, bestehend aus Kommunikationssoftware und embedded Software unter Verwendung des NXP LPC1768

Autor:
Herr

Christian Mahner

Studiengang:
Elektro- und Informationstechnik

Seminargruppe:
EI11wI-B

Erstprüfer:
Prof. Dr. Dr.-Ing. Hartmut Luge

Zweitprüfer:
M.Sc. Rico Thomanek

Einreichung:
Mittweida, 27.02.2015

Bibliografische Beschreibung:

Mahner,Christian:

Konzeption und Aufbau einer Smart Home Anwendung bestehend aus Kommunikationssoftware und embedded Software unter Verwendung des NXP LPC1768. - 2015. – 13 Seiten Verzeichnisse, 69 Seiten Inhalt, 2 Seiten Anlagen.

Mittweida, Hochschule Mittweida, Fakultät Elektro- und Informationstechnik, Bachelorarbeit, 2015

Referat:

Ziel dieser Bachelorarbeit ist die Konzeption, Entwicklung und prototypische Implementierung einer Smart Home Anwendung. Am Beispiel des Controllers NXP LPC1768 wird gezeigt, wie eine solche Anwendung aussehen kann. Dabei wird sowohl auf die Entwicklung der Kommunikationssoftware, als auch der Firm- und Hardware eingegangen. Darüber hinaus werden die verschiedenen Ebenen der Kommunikationstechnik und die verwendeten Protokolle erläutert.

Inhalt

Abbildungsverzeichnis

Tabellenverzeichnis

Codeverzeichnis

Abkürzungsverzeichnis

AAL	Ambient Assisted Living
ADC	Analog Digital Converter
CE	Consumer Electronics
DAC	Digital Analog Converter
DIN	Deutsches Institut für Normung
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ISM	Industrial, Scientific and Medical Band
ISO	International Organization for Standardization
ITK	Informations- und Telekommunikationstechnologie
ITU	International Telecommunication Union
LAN	Local Area Network
LCD	Liquid Crystal Display
LED	Light Emitting Diode
OSI	Open Systems Interconnection
PPP	Point-to-Point Protocol
PPPoE	Point-to-Point Protocol over Ethernet
PWM	Pulse-width Modulation
SRD	Short Range Devices
SSP	Synchronous Serial Port
TCP	Transmission Control Protocol

UDP	User Datagram Protocol
VDE	VDE Verband der Elektrotechnik Elektronik Informationstechnik e.V.
WLAN	Wireless Local Area Network
WSP	Wireless Short-Packet

1 Einleitung

Diese Graduierungsarbeit beschäftigt sich mit der Konzeption und prototypischen Implementierung einer Smart Home Applikation unter Verwendung des Mikrocontrollers NXP LPC1768. Die Arbeiten erfolgten an der „Hochschule Mittweida – University of Applied Sciences“ und zum großen Teil im Rahmen des Forschungsclusters „Smart City Mittweida“ welches durch das Sächsische Staatsministerium für Wissenschaft und Kunst gefördert wurde.

1.1 Kapitelübersicht

In diesem Kapitel wird auf die Zielstellung dieser Arbeit eingegangen und eine kurze Einführung in den Begriff „Smart Home“ gegeben. Kapitel 2 beschäftigt mit allgemeinen Informationen zu Smart Home. Details über den verwendeten Controller LPC1768 und das darauf basierende Controllermodul Chip1768 sind in Kapitel 3 und Kapitel 4 zu finden. In Kapitel 5 werden einige Voruntersuchungen und die dafür verwendeten Beispielprogramme beschrieben, Kapitel 6 umfasst schließlich die während der Bearbeitung erstellte Applikation. Es werden sowohl die Software als auch die Hardware und deren Funktion beschrieben. Informationen über den damit durchgeführten Funktionstest werden kurz in Kapitel 7 ausgeführt. Abschließend sind in Kapitel 8 eine Zusammenfassung und ein Ausblick auf weitere Verbesserungen und Entwicklungen zu finden.

1.2 Smart Home

Ein Smart Home ist ein Wohn- oder Büroraum, im weiteren Sinne auch -gebäude, bei dem die Erhöhung und Verbesserung von Sicherheit, Komfort und Energieeffizienz, durch die Verwendung von vernetzter Elektronik, im Vordergrund stehen. Im Gegensatz dazu bezeichnet man gewerblich genutzte Räumlichkeiten oder Gebäude als Smart Building. Im Weiteren wird auf eine Unterscheidung zwischen gewerblich und privat genutzten Immobilien verzichtet und der Begriff Smart Home übergreifend verwendet. Weitere Synonyme für ein Smart Home sind Begriffe wie Connected Home, Intelligentes Haus und Smart House, um nur einige in den letzten Jahren entstandene Kreationen zu erwähnen. Die „Deutsche Kommission Elektrotechnik Elektronik Informationstechnik im DIN und VDE“, welche sich seit Jahren intensiv mit der Standardisierung im Bereich des Smart Homes beschäftigt, definiert ein Smart Home wie folgt:

1. *Die Bedürfnisse der Bewohner/-innen werden durch eine Vielzahl von Sensoren und smarten Geräten erfasst, die eine intuitive Ansteuerung ermöglichen.*

2. *Die aufgenommenen Informationen werden unter Berücksichtigung des aktuellen Zustandes und der Antizipation potentieller Zustände verarbeitet.*

3. *Es folgt eine Aktion auf die aufgenommenen Informationen und die darauf basierende Interpretation. Hierzu dient ein ausgereiftes Connected Home Netzwerk, welches ein simples und sicheres Zusammenspiel der Geräte aus den Bereichen der Unterhaltungselektronik (CE), der Informations- und Kommunikationstechnik (ITK), Elektrohaushalt (Herd, Kühlschrank, etc.) und Haustechnik (Alarmanlagen, Heizungs- und Lichtsteuerung, etc.) über Schnittstellen, Software, etc. mit Hilfe von drahtgebundenen bzw. drahtlosen Technologien ermöglicht.* [1, p. 12]

Der gesamte Markt für Smart Homes, kann in mehrere Teilbereiche untergliedert werden, für die jeweils eigene Normen und Standards existieren (Abbildung 1.1).

- Sicherheit
- Entertainment/Kommunikation
- Gesundheit/AAL/Wellness
- Smart-Home-Infrastruktur/Automation
- Energiemanagement

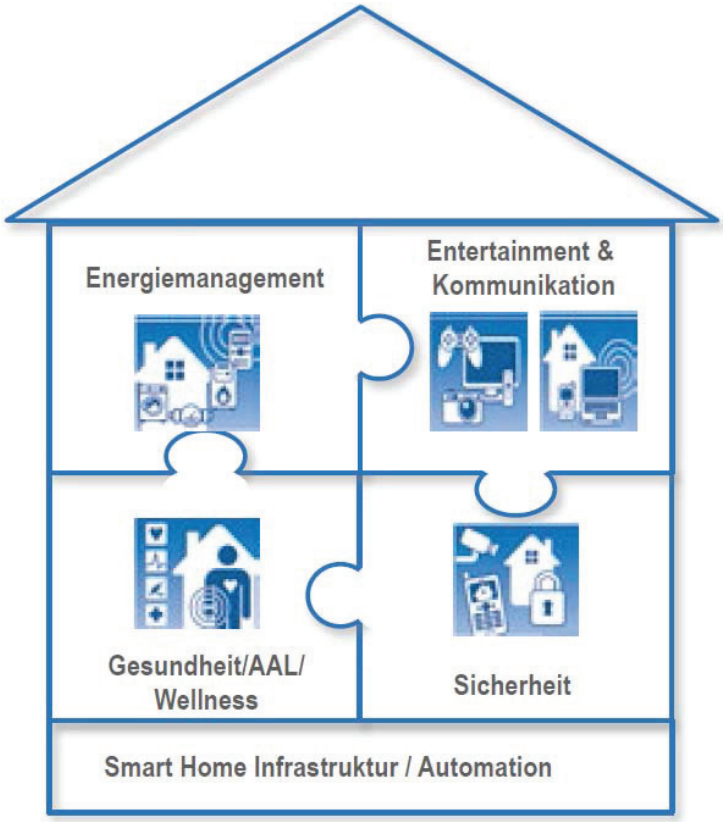


Abbildung 1.1: Teilbereiche des Smart Homes[1]

Zum Teil existieren für diese Bereiche bereits einheitliche Standards, welche aber aktuell und auch in Zukunft erweitert und ausgebaut werden, wobei zu berücksichtigen ist, dass im Moment noch kein Normungsmandat[2] für den Bereich „Smart Home + Building“ in Europa existiert, auch wenn Teilbereiche durch die Europäische Union bereits thematisiert wurden.

1.3 Zielstellung

Im Rahmen der Bachelorarbeit soll eine beispielhafte Implementierung einer Smart Home Anwendung unter Verwendung des NXP LPC1768 konzipiert und getestet werden. Schwerpunkt ist dabei die Einsatzfähigkeit des Mikrocontrollers für Entwicklungen im Be-

[1] [1, p. 48]

[2] Beauftragung internationaler und nationaler Normungsgremien mit der Definition und Erarbeitung neuer Standards für bestimmte Gebiete durch die EU-Kommission

reich Smart Home, sowie im Bereich der Lehre an der Hochschule Mittweida und der dort, im Rahmen der Ausbildung, durchgeführten Versuche zu untersuchen.

Für den Bereich der Entwicklung unter dem Gesichtspunkt „Smart Home“ sind die technischen Eigenschaften wie Peripherie, Leistungsaufnahme und Langzeitverfügbarkeit des Mikrokontrollers vorranging, im Bereich Lehre hingegen kommt es vor Allem auf die Verfügbarkeit einer möglichst preiswerten und umfangreichen Entwicklungsumgebung, sowie die Möglichkeit an, einfache Beispiele mit den auszubildenden Studenten umsetzen zu können.

Zusätzlich wird eine Software zur Steuerung des Smart Home Applikation benötigt, welche auf einem Windows-System betrieben werden kann, sowie kompatibel zu bisherigen Soft- und Hardwarelösungen der Professur Kommunikationstechnik der HS Mittweida ist.

2 Smart Home

In diesem Kapitel sollen Grundlegende Informationen zu Smart Home erläutert werden.

2.1 Aufbau

Allen Smart Home Systemen ist gemein, dass sie aus Teilnehmern bestehen, welche Sensoren, Aktoren, Recheneinheiten, Programmierschnittstellen oder unter Umständen sogar alles gleichzeitig sein können, dabei sind allerdings alle Teilnehmer über einen oder mehrere Kommunikationskanäle miteinander verbunden. Die Teilnehmer können in unterschiedlichsten Topologien, in denen ihnen verschiedene Rollen zukommen können, organisiert werden.

2.2 Ausgewählte Topologien

Auch bei Smart Home werden die Teilnehmer in, in der Kommunikationstechnik bereits bekannten, Topologien organisiert. Die verbreitetesten sind dabei die Linientopologie, die Sterntopologie, die Mashtopologie, die Baumtopologie, sowie der klassische Bus.

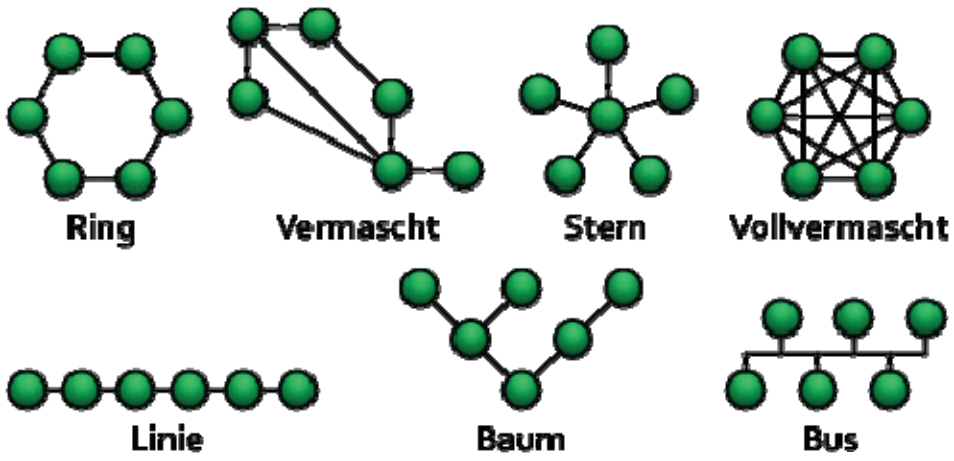


Abbildung 2.1: Ausgewählte Netzwerktopologien[3]

[3] http://commons.wikimedia.org/wiki/File:NetworkTopologies.png

2.3 Stand der Technik

Die am Markt vertretenen Hersteller und Konsortien favorisieren dabei verschiedene Topologien und Übertragungswege und vermischen diese teilweise. So ermöglicht der KNX Standard der KNX Association, welcher 2006 als ISO/IEC 14543-3-1 standardisiert wurde, z.B. das Erstellen einer Linie, welche wiederum um weitere abzweigende Linien erweitert werden kann (Abbildung 2.2).

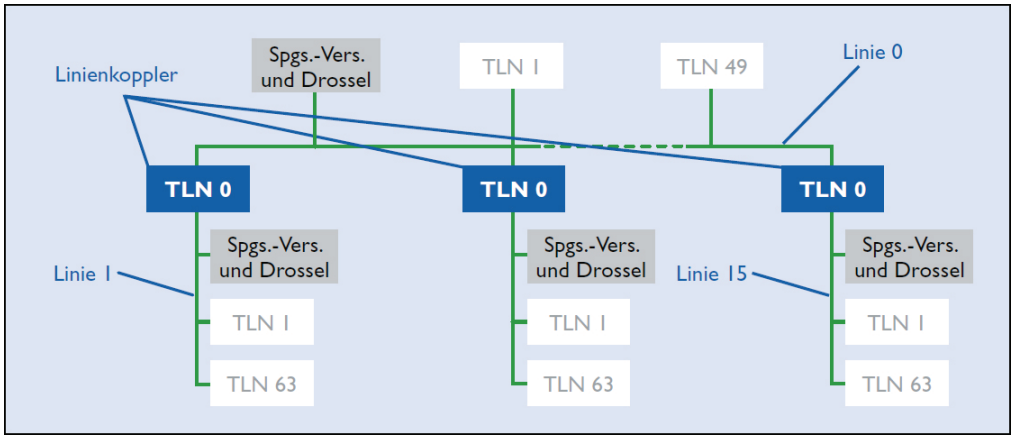


Abbildung 2.2: Zu einem Bereich zusammengeschlossene Linien[4]

Diese Netzwerke bilden sogenannte Bereiche, welche durch weitere Linien mit anderen Bereichen verbunden werden können (Abbildung 2.3). Es scheint also, als ob die Linien- und Baumtopologie miteinander verwachsen. Dabei kommen bei KNX sowohl Funk, in Form von KNX-RF, als auch kabelgebundene Kommunikationswege zum Einsatz.

[4] [6] Seite 8

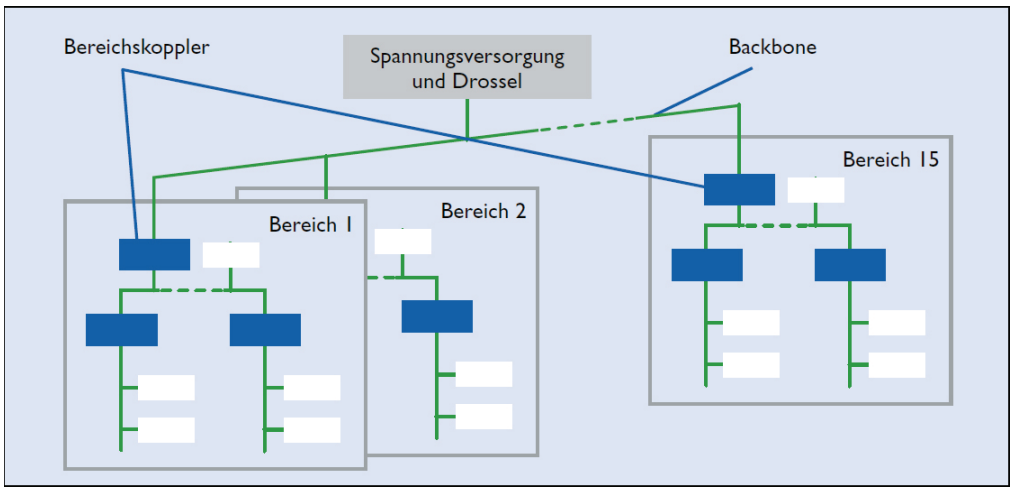


Abbildung 2.3: Durch Koppler verbundene Bereiche[5]

Andere Anbieter wie Belkin, und AVM verwenden IEEE 802.11 (W-LAN) und die Sterntopologie, wobei es sich beim zentralen Knoten um den W-LAN Router handelt und die einzelnen Geräte über Anwendungen auf Smartphones oder Webinterfaces angesteuert werden können. AVM bietet, wie KNX auch, mit dem Powerline 546E die Möglichkeit Geräte über Powerline zu verbinden und zu steuern. Andere große Unternehmen wie die deutsche Telekom, Miele und Vatenfall haben sich zu einem Konsortium namens Quivicon zusammengeschlossen[6] um eine nationale Plattform zur Verfügung zu stellen, dabei werden Standards wie HomeMatic von eQ-3 und ZigBee[7] verwendet. Wieder andere Hersteller setzen auf komplette Eigenentwicklungen, wie z.B. EnOcean mit WSP, ein Funksystem im ISM-Band, welches mittlerweile als ISO/IEC 14543-3-10 zum Standard erklärt wurde. Ein weiterer Vertreter ist der in Dänemark entwickelte Standard Z-Wave, welcher durch die mittlerweile über 250 Mitglieder der Z-Wave Alliance[8] entwickelt und genutzt wird. Auch dieser, in ITU-G.9959 spezifizierte, Standard verwendet, wie einige andere, sowohl das ISM-, als auch das SRD-Frequenzband, wobei die Teilnehmer in einem von einem Primärcontroller gesteuertem, vermaschten Netz organisiert werden.

Allgemein kann festgestellt werde, dass es viele verschiedene Ansätze und Lösungen für den Bereich der smarten Homeautomation gibt, wobei sich in den letzten Jahren immer

---

[5] [6] Seite 8

[6] [8]

[7] IEEE 802.15.4

[8] [7]

mehr Unternehmen zu großen Konsortien zusammenschließen um einheitliche Standards erarbeiten und etablieren zu können. Dieser Trend wird mit Sicherheit auch zukünftig anhalten und die Landschaft der Smart Home-Technologien weiter vereinheitlichen, sodass eine immer bessere Interoperabilität gewährleistet werden kann. Im Moment ist diese Leider nicht immer gegeben. Abgesehen davon, dass keine Kommunikation zwischen den verschiedenen Standards möglich ist, besteht bei zu allgemein gefassten Standards wie ZigBee das Problem, dass nicht einmal die Interoperabilität von Geräten welche den gleichen Standard verwenden gegeben ist. Wie bereits in **1.2** erwähnt, treiben die nationalen und internationalen Standardisierungsgremien, sowie die Europäische Kommission den Prozess der Standardisierung weiter voran, sodass die Grenzen zwischen den verschiedenen Lösungen immer mehr verschwinden werden.

3 Der Mikrocontroller NXP LPC1768

Bei dem Mikrocontroller LPC1768 von NXP handelt es sich um ein Derivat des Cortex-M3 der Firma ARM, welcher auf der ARM-Architektur ARMv7 basiert. Die ARMv7 wurde als etwas leistungsstärkere alternative zu bereits erhältlichen 8-Bit Mikrokontrollern entworfen und wird von vielen etablierten Herstellern unter Lizenz hergestellt.

3.1 Beschreibung des LPC1768

3.1.1 Übersicht über die OnChip-Peripherie

Der LPC1768 ist aufgrund seines verhältnismäßig großen Speichers und umfangreichen Ausstattung an Peripherie großflächig einsetzbar (Tabelle 3.1).

Peripherie	Anzahl
DAC	1
ADC	1
Watchdog Timer	1
Timer	4
GPIO	160
UART	4
CAN	2
USB Host/Device/OTG	1/1/1
I2C/I2S	3/1
SPI/SSP	1/2
Ethernet	1

Tabelle 3.1: Ausgewählte Peripherie des LPC1768

Die für dieses Projekt wichtigen Komponenten sind dabei der DAC (Digital-Analog-Wandler), ADC (Analog-Digital-Wandler), Timer, GPIO (Portpins), SSP und die Ethernet-Schnittstelle. Weitere Ausführungen zu diesen Elementen sind unter 3.2 zu finden.

3.1.2 Wichtige Kenngrößen

Besonders von Vorteil sind die geringe Leistungsaufnahme und die geringe Betriebsspannung von 3.3V, welche somit auch so genannte „low power“-Anwendungen, also Anwendungen mit besonders geringem Energieverbrauch, zulassen (Tabelle 3.2). Wird zwischen den operativen Phasen des Controllers der „deep power-down mode“, also ein Tiefschlafmodus verwendet, bei dem große Teile der Hardware deaktiviert werden, sinkt der Stromverbrauch auf 630nA und ermöglicht besonders lange Batterielaufzeiten.

Kenngröße	Bedingung	Typisch	Max	Einheit
V_{DD}	-	3.3	4.6	V
I_{DD}	f=100Mhz	42	100	mA
	deep power-down mode, RTC aktiv	630	-	nA
T_j	-	-	150	°C
P_{tot}	-	-	1.5	W
I_{OH}	-	4	-	mA
	kurzzeitig	-	40	mA

Tabelle 3.2: Ausgewählte Kenngrößen des LPC1768

Wird zum Beispiel eine Anwendung realisiert, bei der der Controller pro Stunde insgesamt nur eine Minute mit 42mA versorgt werden muss und den Rest der Zeit im Tiefschlafmodus verbringt, kann mit einem handelsüblichen Lithium-Ionen-Akku (3,6V, 1000mAh) eine Laufzeit von ca. 31 Tagen erreicht werden.

$$Laufzeit\ t = \frac{1000mA \cdot h}{42\ mA \cdot {}^{1}/_{60} + 0{,}630\ mA \cdot {}^{59}/_{60}} = 757{,}86h \approx 31d$$

3.2 Beschreibung ausgewählter Peripherie

3.2.1 Der Analog-Digital-Konverter (ADC)

Der Analog-Digital-Konverter dient der Umsetzung analoger Spannungen in digitale Daten. Dabei wird die Analogspannung je nach Auflösung quantisiert und einem digitalen

Wert zugeordnet. Beim ADC des LPC1768 werden positive Spannungen bis 3.3V mit bis zu 12Bit, also auf 806 µV genau, aufgelöst. Der Konverter besitzt 8 Kanäle, sodass 8 verschiedene Spannungen nacheinander digitalisiert werden können. Zur Steuerung des ADC besitzt der Controller folgende Register:

Register	Beschreibung	Zugriff	Reset-Wert
ADCR	A/D Control Register, Das Register ADCR muss beschrieben werden um den Arbeitsmodus zu wählen, bevor eine Konvertierung beginnen kann	R/W	1
ADGDR	A/D Global Data Register, enthält ADC-Done-Bit und das Ergebnis des letzten Wandlung	R/W	-
ADINTEN	A/D Interrupt Enable Register, enthält Bits welche einen Interrupt bei gesetztem Done-Bit erlauben oder verbieten	R/W	0x100
ADDRx (x = 0 … 7)	A/D Channel x Data Register,enthält das Ergebnis der letzten Konvertierung auf Kanal x	RO	-
ADSTAT	A/D Status Register, enthält sowohl DONE- und OVERRUNG-Flags für alle Kanäle, als auch das A/D Interrupt-/DMA-Flag	RO	0
ADTRM	ADC trim Register, zur Korrektur von Abweichungen	R/W	0x00000F00

Tabelle 3.3: Registerübersicht ADC

Verwendet wurden die Register ADCR, ADSTAT, ADGDR und ADINTEN.[9]

3.2.2 Der Digital-Analog-Konverter (DAC)

Der Digital-Analog-Konverter erzeugt aus einem digitalen Eingangswert eine proportionale Ausgangsspannung. Dies geschieht beim LPC1768 für Ausgangsspannungen bis zu 3.3V

[9] Vgl. Kapitel 6.3

bei einem maximalen Strom von 4mA, mit einer Spannungsauflösung von 10Bit, oder 3,2 mV. Zur Steuerung des DAC verwendet der Controller folgende Register:

Register	Beschreibung	Zugriff	Reset-Wert
DACR	D/A Converter Register, enthält den digitalen Wert, der in einen analogen konvertiert werden soll, sowie ein Bit zur Zustandskontrolle (Ein/Aus)	R/W	0
DACCTRL	DAC Control Register, steuert DMA und Timerfunktion	R/W	0
DACCNTVAL	DAC Counter Value Register, enthält den Reload-Wert für DAC DMA/Interrupttimer	R/W	0

Tabelle 3.4: Registerübersicht DAC

Verwendet wurde das Register DACR.[10]

3.2.3 Der Synchronous Serial Port (SSP)

Beim Synchronous Serial Port handelt es sich um einen Controller, welcher die serielle Übertragung von Daten über einen Bus unter Verwendung verschiedener Standards ermöglicht. Dabei stehen SPI, 4-wire SSI und Microwire zur Verfügung. Gesteuert wird der SSP-Controller über folgende Register:

[10] Vgl. Kapitel 6.3

Register	Beschreibung	Zugriff	Reset-Wert
CR0	Control Register 0, bestimmt die serielle Taktrate, den Bustyp und die Datengröße	R/W	0
CR1	Control Register 1, bestimmt master/slave und Anderes	R/W	0
DR	Data Register, Schreiben füllt Übertragungsregister (FIFO), lesen leert es	R/W	0
SR	Status Register	RO	-
CPSR	Clock Prescale Register	R/W	0
IMSC	Interrupt Set and Cleat Register	R/W	0
RIS	Raw Interrupt Status Register	R/W	-
MIS	Masked Interrupt Status Register	R/W	0
ICR	SSPICR Interrupt Clear Register	R/W	-
DMACR	DMA Control Register	R/W	0

Tabelle 3.5: Registerübersicht SSP

3.2.4 Der Timer

Beim Timer handelt es sich um einen Controller, der einen Zähler mit einem festgelegten Takt bedient und somit eine Zeitmessung ermöglicht. Diese Grundfunktion wird bei modernen Controllern wie dem LPC1768 durch viele andere Funktionen erweitert, so lässt sich die Zählrichtung, der Eingangstakt und die Anzeige eines Ereignisses an einem Ausgang durch einen Pegelwechsel einstellen. Der LPC1768 besitzt insgesamt vier Timer, wobei jeder ein Zählregister mit einer Größe von 32Bit besitzt.

Register	Beschreibung	Zugriff	Reset-Wert
IR	Interrupt Register. Schreiben löscht Interrupts, Lesen identifiziert anstehenden Interrupt	R/W	0
TCR	Timer Control Register. Steuert die Zählfunktionen des Timers	R/W	0
TC	Timer Counter, 32-Bit Zählregister des Timers	R/W	0
PR	Prescale Register, TC wird erhöht, wenn PC mit PR übereinstimmt	R/W	0
PC	Prescale Counter. Wird erhöht, bis PR erreicht wurde und erhöht damit TC – entspricht Vorteiler	R/W	0
MCR	Match Control Register.Steuert ob ein Match-Ergeinis einen Interrupt auslöst und TC zurücksetzt	R/W	0
MRx x=0…3	Match Register x, Kann TC zurücksetzen wenn Match-Wert erreicht ist, außerdem TC und PC stoppen und Interrupts auslösen	R/W	0
CTCR	Counter Control Register, Entscheidet zwischen Zähl- und Timer-Modus und legt die Signalquellen für den Zähl-Modus fest	R/W	0

3.2.5 Der Ethernet-Controller

Beim Ethernet-Controller handelt es sich um ein Bauteil, welches die Kommunikation über Ethernet ermöglicht. Der dafür benötigte Transceiver ist nicht integriert und muss dem Controller vorgeschaltet werden. Des Weiteren wird ein Ethernet Media Access Controller benötigt, welcher in diesem Fall durch den Chip1768[11] zur Verfügung gestellt wird.

[11] vgl. Kapitel 4

4 Das Controllermodul Chip1768

Der Chip1768 ist ein Modul der Firma Elektronikladen, welcher auf Basis des LPC1768 von NXP welches zum einen für den Betrieb nötige Peripherie zum anderen eine einfach zu handhabende Bauform bereitstellt. Einige Eigenschaften des Chip178, sowie wichtige Kenngrößen werden in diesem Kapitel erläutert.

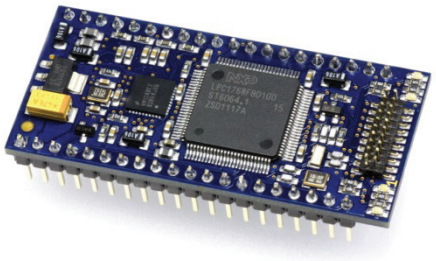


Abbildung 4.1: Chip178 von Elektronikladen[12]

4.1 Beschreibung des Chip1768

4.1.1 Peripherie, Funktionen und Pinbelegung

Das Modul Chip1768 liefert die für den Betrieb des LPC1768 nötige oder optionale Peripherie, wie z.B. die Spannungsversorgung, den Ethernet-Transceiver, ein Debug- und Programmierschnitstelle sowie eine Auswahl der vorhandenen GPIO-Pins. Das Modul bietet einen guten Kompromiss zwischen Abmessung und zur Verfügung stehenden Funktionen des LPC1768.

[12] [9] Seite 1

Pin	1. Funktion	2. Funktion	3. Funktion	4. Funktion	Pin am LPC1768
P5	P0[9]	I^2S (TX_SDA)	SSP1 (MOSI)	MAT2[3]	76
P6	P0[8]	I^2S (TX_WS)	SSP1 (MISO)	MAT2[2]	77
P7	P0[7]	I^2S (TX_CLK)	SSP1 (SCK)	MAT2[1]	78
P8	P0[6]	I^2S (RX_SDA)	SSP1 (SSEL)	MAT2[0]	79
P9	P0[0]	CAN1 (RD1)	UART3 (TX)	I^2C1 (SDA)	46
P10	P0[1]	CAN1 (TD1)	UART3(RX)	I^2C1 (SCL)	47
P11	P0[18]	UART1 (DCD)	SSP0 (MOSI)	SPI (MOSI)	60
P12	P0[17]	UART1 (CTS)	SSP0 (MISO)	SPI (MISO)	61
P13	P0[15]	UART1 (TXD)	SSP0 (SCK)	SPI (SCK)	62
P14	P0[16]	UART1 (RXD)	SSP0 (SSEL)	SPI (SSEL)	63
P15	P0[23]	A/D0[0]	I^2S (RX_CLK)	CAP3[0]	9
P16	P0[24]	A/D0[1]	I^2S (RX_WS)	CAP3[1]	8
P17	P0[25]	A/D0[2]	I^2S (RX_SDA)	UART3 (TXD)	7
P18	P0[26]	A/D0[3]	DAC out	UART3 (RXD)	6
P19	P1[30]	V_{BUS}	A/D0[4]		21
P20	P1[31]	SSP1 (SCK)	A/D0[5]		20
P21	P2[5]	PWM1[6]	UART1 (DTR)	TRACEDATA[0]	68
P22	P2[4]	PWM1[5]	UART1 (DSR)	TRACEDATA[1]	69
P23	P2[3]	PWM1[4]	UART1 (DCD)	TRACEDATA[2]	70
P24	P2[2]	PWM1[3]	UART1 (CTS)	TRACEDATA[3]	73
P25	P2[1]	PWM1[2]	UART1 (RXD)		74
P26	P2[0]	PWM1[1]	UART1 (TXD)		75
P27	P0[11]	UART2 (RXD)	I^2C2 (SCL)	MAT3[1]	49
P28	P0[10]	UART2 (TXD)	I^2C2 (SDA)	MAT3[0]	48
P29	P0[5]	I^2S (RX_WS)	CAN2 (TD)	CAP2[1]	80
P30	P0[4]	I^2S (RX_CLK)	CAN2 (RD)	CAP2[0]	81
D+	P0[29]	USB (D+)			29
D-	P0[30]	USB (D-)			30
RX+ *RX-* *TX+* *TX-*	(Ethernet)				-
IF+	P0[2]	UART0 (TX)	A/D0[7]		98
IF-	P0[3]	UART0 (RX)	A/D0[6]		99
ISP	P2[10]	!EINT0	NMI		41

Abbildung 4.2: Pinbelegung des Chip1768[13]

Wie in der Übersicht zu sehen, stehen Pins für die Busse, Timer, Wandler, Ethernet, für die PWM sowie für die Programmierung und das Debuggen zur Verfügung.

[13] [9] Seite 8

4.1.2 Wichtige Kenngrößen

Der Chip1768 besitzt 40 Anschlusspins und einen Linearregler, welcher die Peripherie und den LPC1768 versorgt. Der Regler ermöglicht Eingangsspannungen bis 12V, sollte aber aufgrund der Abwärme im Bereich von 5V betrieben werden. Die maximale Stromaufnahme ist mit 200mA angegeben.

Kenngröße	Bedingung	Typisch	Max	Einheit
V_{DD}	-	5	12	V
I_{DD}	-		200	mA
Pins	-	40	-	-

Tabelle 4.1: Kenngrößen Chip1768

5 Untersuchungen am NXP LPC1768

Im Vorfeld der eigentlichen Entwurfs- und Programmierarbeit wurden einige Untersuchungen am LPC1768 beziehungsweise dem Chip1768 hinsichtlich ihrer Tauglichkeit für dieses Projekt durchgeführt. Dazu wurden einige kleine Beispielprogramme erstellt, welche bereits in der Lehre der Professur Kommunikationstechnik der Hochschule Mittweida eingesetzt werden.

5.1 Ausgewählte Untersuchungen an Beispielprogrammen

5.1.1 Entwicklungsumgebung und Grundlegende Controllerfunktionen

Um die Funktionen und Bedienbarkeit der durch ARM bzw. KEIL gelieferten Entwicklungsumgebung sowie des Controllers zu bewerten, wurde ein simples Programm erstellt, welches Portpins in einer zeitlichen Abfolge ansteuert. Dabei werden auf dem Chip1768 vorhandene und an diesen Pins angeschlossene LEDs bedient. Zum einen dient es der Einarbeitung in die Entwicklungsumgebung und der Registerstruktur des Controllers, zum anderen bietet es einen einfachen Einstieg für Studenten.

```
01 init_lcd();
02 lcd_cursor(4,2);
03 lcd_putstring((unsigned char*)"Hello world!");
```

Quellcode 5.1: Beispielprogramm 1

Zu Beginn wird das LC-Display und die zur Steuerung benötigten Portpings initlalisiert (Quellcode 5.1,Zeile 1). Im Anschluss wird der Cursor auf dem Display auf die gewünschte Position gesetzt und der Text „Hello world!“ ausgegeben (Quellcode 5.1, Zeile 2-3).

```
01 LPC_GPIO1->FIODIR |=  (1<<led[0]) |
02                       (1<<led[1]) |
03                       (1<<led[2]) |
04                       (1<<led[3]);
05
06 LPC_GPIO1->FIOCLR |=  (1<<led[0]) |
07                       (1<<led[1]) |
08                       (1<<led[2]) |
09                       (1<<led[3]);
```

Quellcode 5.2: Beispielprogramm 1

Um die Portpins zum Schalten der LEDs nutzen zu können, werden diese über das Register „FIODIR“ als Ausgänge konfiguriert (Quellcode 5.2, Zeile 1-4). Im Array „led“ sind dafür die korrespondierenden Bitpositionen der einzelnen Pins gespeichert, dies ermöglicht eine übersichtliche Verwaltung der Pins, sowie das schrittweise Ansteuern der LEDs in einer Schleife. Danach werden die Pins in den Ausgangszustand „0“ versetzt (Quellcode 5.2, Zeile 6-9).

```
01 while (true)
02 {
03     int i=0;
04
05     LPC_GPIO1->FIOCLR |=  (1<<led[prev_pos]);
06     LPC_GPIO1->FIOSET |= (1<<led[pos]);
07
08     prev_pos=pos;
09     pos++;
10
11     if (pos==4) pos=0;
12     for (i=0; i<8300000; i++);
13 }
```

Quellcode 5.3: Beispielprogramm 1

In der darauf folgenden While-Schleife werden die LEDs der Reihe nach an- bzw. ausgeschaltet. Dabei dienen die Variablen „pos“ und „prev_pos“ als Speichervariable für die aktuelle Position des Lauflichtes, sowie die vorhergehende. Die letzte aktivierte LED wird deaktiviert, die aktuelle Position aktiviert. Anschließend wird die aktuelle Position in „prev_pos“ gespeichert und die Position um eins erhöht. Ist der Zähler bei vier angekommen, beginnt die Schleife wieder bei der Anfangsposition. Um das aufleuchten der LEDs für das menschliche Auge wahrnehmbar zu machen, wird mit Hilfe einer for-Schleife eine Verzögerung erzeugt. (Quellcode 5.3).

Für die Ansteuerung des LC-Displays im 4-Bit-Modus, wurde eine Bibliothek für den Controller HD44780 erstellt, welche im weiteren Verlauf des Projektes genutzt wurde.

5.1.2 AD-Wandler

Die Funktion des AD-Wandlers wurde getestet, in dem die Analogspannung am Eingangspin gemessen und auf einem LC-Display ausgegeben wird, auch hier wurde die bereits vorhandene Bibliothek für das Display verwendet. Die Initialisierung und Steuerung des AD-Wandlers ist in wenigen Zeilen realisiert worden. Während der Programmentwicklung wurde allerdings festgestellt, dass der AD-Wandler unwillkürlich sogenannte Glitches, also Falschmessungen, generiert. Da bei diesem Versuch nur die prinzipielle Funktion und Bedienung des Wandlers im Mittelpunkt stand, wurde der Fehler toleriert. Weitere Informationen zu diesem Fehler und dessen Relevanz für die Anwendung des LPC1768 befinden sich in Kapitel 5.2.

```
01 LPC_ADC->ADCR |= (1<<21);
02 LPC_ADC->ADCR |= (1<<0) | (0xFF<<8);
03 LPC_PINCON->PINSEL1 |= (1<<14);
```

Quellcode 5.4: Beispielprogramm 2

Der AD-Wandler wird durch das Setzen des Bits „PDN“, an Position 21, im Register ADCR aktiviert. Außerdem muss die Funktion des Portpins für den ADC konfiguriert werden, dies geschieht im Register „PINSEL1“ (Quellcode 5.4, Zeile 1-3).

```
01 LPC_ADC->ADCR |= (1<<24);
```

Quellcode 5.5: Beispielprogramm 2

Anschließend wird die erste Konvertierung gestartet, dazu muss das Bit 24 im Register „ADCR“ gesetzt werden (Quellcode 5.5).

```
01 while (true)
02 {
03     if (((LPC_ADC->ADDR0)>>31)&0x01)
04     {
05         char s_result[6] = {"0.000\0"};
06         int result = ((LPC_ADC->ADDR0)>>4)&0xFFF;
07         double d_result = (double)result;
08         d_result = 3.3/4096*result;
09         sprintf(s_result,"%5.3f",d_result);
10         lcd_cursor(1,1);
11         lcd_putstring((unsigned char*)s_result);
12         lcd_putchar('V');
13         LPC_ADC->ADCR |= (1<<24);
14     }
15 }
```

Quellcode 5.6: Beispielprogramm 2

In der darauf folgenden While-Schleife wird geprüft ob die Konvertierung abgeschlossen ist (Quellcode 5.6, Zeile 3), dies wird signalisiert, indem das Bit 31 des Registers „ADDR0“ gesetzt ist. In diesem Fall wird der Digitalwert aus dem Register „ADDR0“ gelesen, in einen String konvertiert und auf dem Display ausgegeben. Dazu wird er mithilfe der Funktion „sprintf()“ entsprechend formatiert. Abschließend wird eine neue Konvertierung des ADC gestartet (Quellcode 5.6, Zeile 5 ff).

5.1.3 DA-Wandler und Zustandsabfrage von Portpins

In diesem Versuch wurde die Funktion und Bedienung des DA-Wandlers, sowie die Zustandsabfrage von Portpins und somit die Realisierbarkeit von Schaltern überprüft. Dazu wurde am Ausgang des Wandlers eine Spannung ausgegeben, welche sich abhängig davon welcher Schalter gedrückt wird, mit einer festgelegten Schrittweite ändert. Diese Vorgehensweise wurde unter anderem gewählt, da die Funktion den Studenten mithilfe des Oszilloskops in geeigneter Weise visuell näher gebracht werden kann. Des Weiteren besteht die Möglichkeit einen festen, im Programmcode festgelegten Wert auszugeben.

```
01 int dac_value=0;
02 LPC_SC->PCLKSEL0 |= (3<<22);
03 LPC_PINCON->PINSEL1 |= (1UL<<21);
04 LPC_DAC->DACR = (dac_value<<6);
```

Quellcode 5.7: Beispielprogramm 3

Zu Beginn wird der D/A-Wandler konfiguriert, hierzu wird der Takt des Wandlers mithilfe des Registers PCLKSEL0 auf 25Mhz festgelegt, indem die Bits 22 und 23 gesetzt werden (Quellcode 5.7, Zeile 2). Anschließend muss die Funktion für den Portpin P0.26 auf „DAC“

geändert werden, dies geschieht durch das Setzen des Bits 21 im Register „PINSEL1“ (Quellcode 5.7, Zeile 3). Bevor das Programm in eine Schleife übergeht, wird der Wandler noch mit dem Ausgangswert „0“ initialisiert (Quellcode 5.7, Zeilen 1 und 4).

```
01 while (continous)
02 {
03     if (!(((LPC_GPIO0->FIOPIN)>>16)&0x1))
04     {
05         if ((dac_value+62)>=1024) dac_value=1023;
06         else dac_value+=62;
07         LPC_DAC->DACR = (dac_value<<6);
08         wait_ms(150);
09     }
10     else if (!(((LPC_GPIO0->FIOPIN)>>6)&0x1))
11     {
12         if ((dac_value-62)<0) dac_value=0;
13         else dac_value-=62;
14         LPC_DAC->DACR = (dac_value<<6);
15         wait_ms(150);
16     }
17 }
```

Quellcode 5.8: Beispielprogramm 3

Wurde das Symbol „continous“ mit Hilfe der Präprozessoren auf „true“ gesetzt, so arbeitet das Programm kontinuierlich die Schleife ab (Quellcode 5.8, Zeile 1). Innerhalb dieser Schleife wird geprüft welcher der Beiden Taster, an den Pins P0.16 und P0.6 betätigt wurde (Quellcode 5.8, Zeilen 3 und 10). Pin P0.16 erhöht die ausgegebene Spannung in etwa um 0.2V pro Schritt, was einem digitalen Wert von 62 entspricht, P0.6 verringert die Spannung (Quellcode 5.8, Zeilen 5 und 12). Durch die if-Anweisungen wird die untere bzw. obere Grenze des Wertes auf 0 und 1023 festgelegt, was genau der Größe des entsprechenden Registers des DAC entspricht (Quellcode 5.8, Zeilen 6 und 13). Mit Hilfe der Funktion „wait_ms()“ wird eine Softwareverzögerung ausgeführt um die Taster zu entprellen (Quellcode 5.8, Zeilen 8 und 15).

```
01 LPC_DAC->DACR = (DAC_Value<<6);
02
03 while (true)
04 {
05 }
```

Quellcode 5.9: Beispielprogramm 3

Ist das Symbol „continous“ nicht „true“, so wird die erste Schleife übersprungen und der Wert, welcher durch das Symbol „DAC_Value“ festgelegt wurde, ausgegeben. Die zweite While-Schleife verhindert das Beenden des Programms, sodass die Spannung am Ausgang des DAC konstant bei dem eingestellten Wert bleibt (Quellcode 5.9).

Abbildung 5.1 zeigt eine Darstellung der Ausgangsspannung auf einem Oszilloskop. Zu sehen ist wie die Spannung schrittweise mit Hilfe der Taster bis zum Maximum erhöht und anschließend wieder verringert wird.

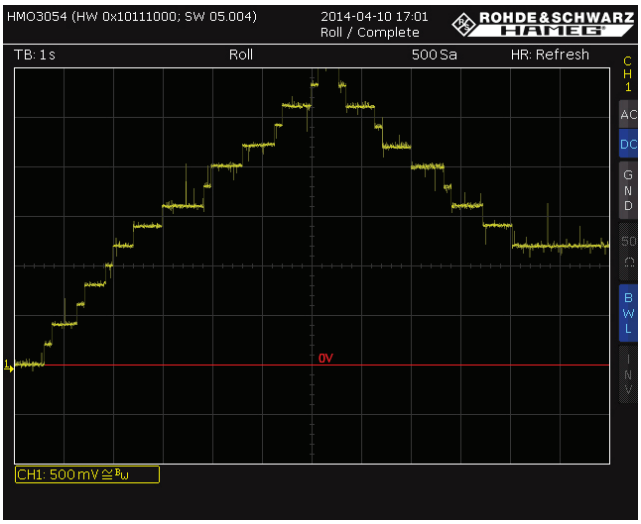


Abbildung 5.1: Ausgangsspannung des D/A-Wandlers

5.1.4 Ethernet und TCP/IP

Der letzte und etwas umfangreichere Versuch widmete Sich der Ethernet-Schnittstelle, sowie dem TCP/IP-Stack von KEIL, welcher unter Lizenz genutzt werden darf. Dazu wurde der Stack entsprechend der Dokumentation von KEIL[14] implementiert und eine Software in C# für Windows geschrieben. Diese Software überträgt ein Byte an den Controller, welcher entsprechend des Inhaltes, welcher „0“ oder „1“ sein kann, eine LED an- oder ausschaltet.

[14] [11]

```
01 LPC_GPIO1->FIODIR |= (1<<18);
02 LPC_GPIO1->FIOCLR |= (1<<18);
03 SysTick->LOAD = (SystemCoreClock / 100) - 1;
04 SysTick->CTRL = 0x05;
```

Quellcode 5.10: Beispielprogramm 4

Als erstes wird der Portpin, an dem die LED angeschlossen ist als Ausgang definiert und auf „0“ gesetzt, sowie der SysTickTimer initalisiert und gestartet. Dieser gibt den Takt für das zyklische Ausführen des TCP-Stacks vor, da die hier verwendete Bibliothek von KEIL im Poll-Verfahren arbeitet. Der SysTickTimer wird mit dem CPU-Takt, also 100MHz, getaktet und generiert einen Interrupt, sobald das Zählregister, welches mit jedem Takt erniedrigt wird, den Wert 0 erreicht hat. In diesem Fall wird das Zählregister mit dem Wert „`(SystemCoreClock / 100) - 1`“, also 999999 aus dem Register „LOAD“ geladen. Der Interrupt wird dementsprechend nach 10 ms erreicht, da 10^6 Zyklen nötig sind (Quellcode 5.10). Die Anzahl der nötigen Zyklen berechnet sich wie folgt:

$$n = 100 \cdot 10^6 MHz \cdot 10ms = 100 \cdot 10^3$$

Nach dem der TCP-Stack mit Hilfe der proprietären Funktion „init_TcpNet()“ initialisiert wurde, wird ein TCP-Socket auf Port 80 erstellt, auf dem der Server lauscht. Die Funtkion „tcp_callback“ wird aufgerufen, sobald der Stack ein Ereignis an der Ethernetschnittstelle registriert (Quellcode 5.11).

```
01 init_TcpNet();
02
03 socket_tcp = tcp_get_socket(TCP_TYPE_SERVER,0,1,tcp_callback);
04 if ( socket_tcp != 0 )
05 {
06     tcp_listen(socket_tcp,80);
07 }
```

Quellcode 5.11: Beispielprogramm 4

In der darauf folgenden While-Schleife werden bei jedem Durchlauf die Funktionen „timer_poll()“, „main_TcpNet()“ und „send_data()“ aufgerufen (Quellcode 5.12), wobei es sich bei „main_TcpNet“ um eine durch den Netzwerkstack vorgegebene Funktion handelt. Diese muss zur korrekten Funktion der Netzwerkanbindung regelmäßig aufgerufen werden.

```
01 while (true)
02 {
03     timer_poll();
04     main_TcpNet();
05     send_data();
06 }
```

Quellcode 5.12: Beispielprogramm 4

Funktion „timer_poll()“

Die Funktion überprüft das Register „CTRL“ des SysTickTimers auf das Bit 16. Dieses wird gesetzt, wenn das Zählregister den Wert 0 erreicht hat, was aller 10 ms geschieht. Ist dies der Fall, so wird die Funktion „timer_tick()“ ausgeführt. Diese wird wie die Funktion „main_TcpNet()“ durch die Bibliothek von KEIL vorgegeben und muss zyklisch ausgeführt werden.

```
01 void timer_poll()
02 {
03     if ((SysTick->CTRL) & 0x10000)
04     {
05         timer_tick ();
06     }
07 }
```

Quellcode 5.13: Beispielprogramm 4, Funktion timer_poll()

Funktion send_data()

Die Funktion dient dem Versenden von Daten an einen, auf dem vorhanden Socket verbundenen, Client. Ist die, durch die „tcp_callback“-Funktion veränderte, Variable „send_ack“ wahr, so wird der Socket darauf überprüft, ob ein Client verbunden und das Senden von Daten möglich ist (Quellcode 5.14, Zeile 3 ff). Trifft beides zu, so wird entsprechend des Zustandes von Portpin P1.18 eine „1“ oder eine „0“ versendet (Quellcode 5.14, Zeile 11 ff).

```
01 void send_data()
02 {
03     if ( send_ack )
04     {
05         if (tcp_get_state(socket_tcp) == TCP_STATE_CONNECT)
06         {
07             if (tcp_check_send(socket_tcp))
08             {
09                 U8 *sendbuf;
10                 sendbuf = tcp_get_buf(1);
11                 *sendbuf = (((LPC_GPIO1->FIOPIN>>18)&0x01)==1)
12                 ? '1' : '0';
13                 tcp_send(socket_tcp,sendbuf,1);
14             }
15         }
16         send_ack = false;
17     }
18 }
```

Quellcode 5.14: Beispielprogramm 4, Funktion send_data()

Funktion tcp_callback()

Diese Funktion wird aufgerufen, sobald eines der in der Bibliothek vordefinierten Ereignisse eintritt. Möglich sind folgende Ereignisse:

- TCP_EVT_CONREQ - Verbindungsanfrage
- TCP_EVT_ABORT - Verbindungsabbruch
- TCP_EVT_CONNECT - Verbindung hergestellt
- TCP_EVT_CLOSE - Verbindung geschlossen
- TCP_EVT_ACK - Client bestätigt Datenempfang
- TCP_EVT_DATA - eingehende Daten

Bei diesem Beispielprogramm wird auf den Empfang von Daten geprüft (Quellcode 5.15, Zeile 9). Entsprechend des Inhalts, welcher „0“ oder „1“ sein kann, wird das Portpin P1.18 auf „1“ oder „0“ gesetzt und die daran angeschlossene LED an- bzw. ausgeschaltet. Die Ereignisse „TCP_EVT_CONREQ“ und „TCP_EVT_CONNECT“, welche bei einer Verbindungsanfrage und dem erfolgreichen Herstellen einer Verbindung ausgelöst werden, beenden die Funktion mit dem Rückgabewert „1“, welcher dem TCP-Stack signalisiert weiterzuarbeiten (Quellcode 5.15, Zeile 10 ff).

```
01 U16 tcp_callback(U8 socket, U8 event, U8 *rcv_data,U16 par)
02 {
03     if ( socket != socket_tcp )
04     {
05         return (0);
06     }
07     switch (event)
08     {
09     case TCP_EVT_DATA:
10         if (*rcv_data == '1')
11         {
12             LPC_GPIO1->FIOSET |= (1<<18);
13         }
14         else if (*rcv_data == '0')
15         {
16             LPC_GPIO1->FIOCLR |= (1<<18);
17         }
18         send_ack = true;
19         break;
20     case TCP_EVT_CONREQ:
21         return (1);
22     case TCP_EVT_CONNECT:
23         return (1);
24     }
25     return (0);
26 }
```

Quellcode 5.15: Beispielprogramm 4, Funktion tcp_callback()

5.2 Der A/D-Wandler, Fehler und Relevanz

Wie bereits erwähnt wurde während der Voruntersuchungen festgestellt, dass der ADC des LPC1768 Glitches[15] erzeugt. Um einen Fabrikationsfehler auszuschließen, wurde das Programm auf weiteren anderen Derivaten getestet und mehrmals überprüft. Nach Recherche zu diesem Fehler, scheint es sich dabei um einen Designfehler in der Architektur des Chips zu handeln, eine offizielle Mitteilung von NXP oder ARM dazu gibt es allerdings nicht. Diese Schlussfolgerung basiert auf ähnlichen Fehlerbeschreibungen in einschlägigen Internetforen[16], sowie dem Supportforum von KEIL[17], in denen ein ähnliches Verhalten mehrfach geschildert wird.

[15] Glitch – kurzzeitiger Fehler in digitalen System, welcher in einem falschen Impuls resultiert

[16] http://developer.mbed.org/forum/mbed/

[17] http://www.keil.com/forum/

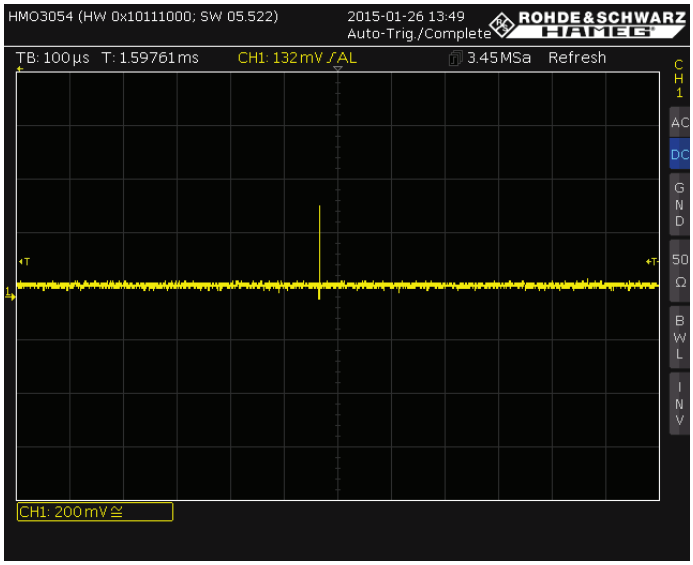


Abbildung 5.2: Glitch des A/D-Wandlers

Bei dem Fehler handelt es sich um eine Falschmessung, welche gelegentlich Werte im oberen Bereich des Spannungsbereiches umsetzt. Dies ist für die in diesem Fall geplante Anwendung weniger relevant, da keine kurzzeitigen kontinuierlichen Veränderungen gemessen werden. Für eine quasistatische, langsame Änderung bzw. eine Messung mit geringer Entropie ist dieser Fehler also vernachlässigbar, da er Interpoliert oder aussortiert werden kann. Sollen kontinuierliche Signale abgetastet werden, so sollten Hard- oder Softwarefilter, wie z.B. ein Tiefpass oder Algorithmus zur Mittelung von Werten, eingesetzt werden.

6 Die Smart Home Applikation

Bei der erstellten Applikation handelt es sich um einen Aufbau welcher die prinzipielle Funktion eines Smart Home anhand eines LPC1768 demonstrieren soll. Dabei werden der Controller und seine Peripherie über eine PC-Software, welche über Ethernet mit dem Controller kommuniziert, gesteuert und überwacht. Sowohl die Firmware als auch die Steuersoftware und die Hardwareschaltung wurden im Rahmen dieses Projektes erstellt.

6.1 Allgemeines

Das System ist nach dem klassischen Server-Client-Prinzip aufgebaut, wobei der Controller dem Server entspricht, welcher durch den PC mithilfe der Steuersoftware als Client, bedient wird. Dabei wird die Verbindung über klassisches Ethernet mit 100BASE-TX, also 100Mbit/s Übertragungsrate (IEEE 802.3 Clause 25[18]), hergestellt. Als Transportprotokoll wird TCP/IP eingesetzt, auf der Anwendungsebene kommt ein eigenes von der Professur Kommunikationstechnik entwickeltes Protokoll zum Einsatz[19].

6.2 Übertragungsprotokoll

Bis auf die Übertragungsebene der Transportschicht nach dem OSI-Modell, setzt das System auf bewährte Standards. So wird IEEE 802.3 für die unteren beiden Schichten, also die physische und die Bitübertragungsschicht, eingesetzt, darüber folgt das Internetprotokoll in Schicht 3, der Vermittlungsschicht und das TCP als Transportschicht, Schicht 4. Dies ermöglicht eine große Kompatibilität, vor Allem zu den als Client einsetzbaren Geräten. Des Weiteren kann die physische Schicht durch Gateways erweitert werden, sodass zum Beispiel auch IEEE 802.11[20], also Wireless LAN eingesetzt werden kann.

Zusätzlich wurde ein, durch die Professur Kommunikationstechnik entwickeltes, Protokoll auf der Anwendungsschicht implementiert. Dies dient der Authentifizierung von Teilnehmern und der anschließenden Übertragung der Nutzdaten, nach einem festen Schema.

---

[18] [13]

[19] Vgl. Kapitel 6.2.4

[20] [12]

6.2.1 Das ISO OSI-Referenzmodell

Das OSI-Referenzmodell ist ein Modell welches von ITU[21] und ISO[22] dazu verwendet wird Netzwerkprotokolle in Schichten zu beschreiben. Es wurde im Jahr 1983 durch die ITU, beziehungsweise 1984 durch die ISO, standardisiert und soll dabei helfen die Kommunikation über verschiedene technische Systeme hinweg zu vereinheitlichen und dadurch wiederum die Entwicklung zu vereinfachen und voranzutreiben. Dabei wird die Kommunikation zwischen Teilnehmern in sieben Schichten eingeteilt, wobei jede einen beschränkten Aufgabenbereich hat (Abbildung 6.1). Dabei werden die untersten vier Schichten als transportorientierte, die darüber liegenden drei Schichten als anwendungsorientierte Schichten zusammengefasst.

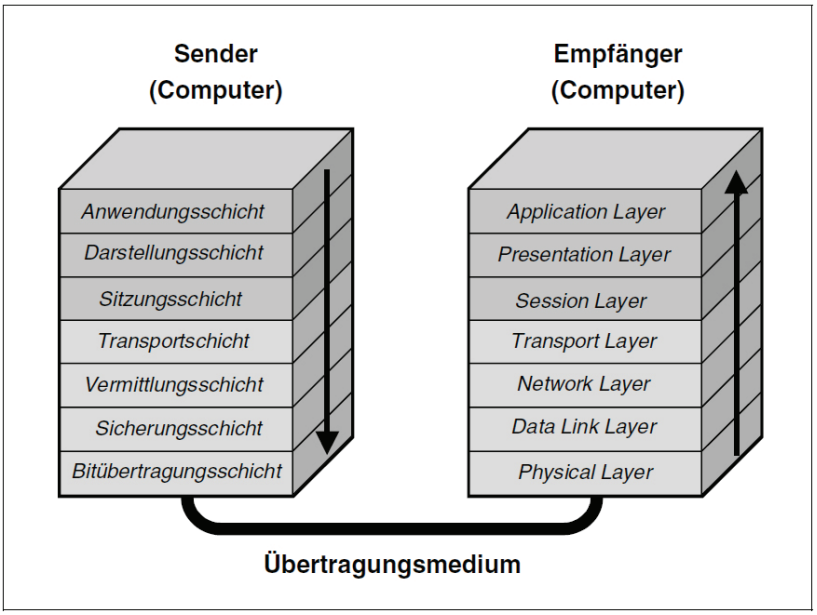


Abbildung 6.1: Das OSI-Referenzmodell[23]

[21] [14]

[22] [15]

[23] [4] Seite 67

6.2.1.1 Bitübertragungsschicht

Die Bitübertragungsschicht ist die erste und unterste Schicht des Modells, sie enthält alles Nötige um eine physische Verbindung zwischen den Teilnehmern herzustellen. Darüber hinaus wird definiert in welcher Art und Weise die Informationen auf Bitebene übertragen werden.

6.2.1.2 Sicherungsschicht

Die Sicherungsschicht unterteilt den Informationsstrom in Blöcke und fügt Redundanzen zur Sicherung der übertragenen Informationen hinzu. Dazu werden in der Regel Prüfsummen oder Paritäten verwendet.

6.2.1.3 Vermittlungsschicht

In der Vermittlungsschicht sorgen Protokolle für das Vermitteln der Informationen zum gewünschten Empfänger. Dazu zählt das schalten von Verbindungen, oder das Weitervermitteln einzelner Datenpakete, sowie das auswählen geeigneter Routen zum Ziel.

6.2.1.4 Transportschicht

Die Transportschicht dient der Segmentierung des Datenstroms und sorgt dafür, dass die Daten beim Empfänger in der richtigen Reihenfolge wieder zusammengesetzt, und somit der Datenstrom rekonstruiert wird. Sie ermöglicht den darüber liegenden Schichten außerdem einen einheitlichen Zugriff auf die Transportinfrastruktur in Netzwerken unabhängig von deren Eigenschaften. Die Transportschicht unterscheidet sowohl verbindungslose Transportprotokolle, wie UDP, als auch verbindungsorientierte Transportprotokolle, wie TCP.

6.2.1.5 Sitzungsschicht

Die Sitzungsschicht beschreibt den Ablauf von virtuellen Verbindungen zwischen Anwendungsprozessen. Dabei ermöglicht den Auf- und Abbau, sowie die Synchronisation der Verbindung. Sollte die Verbindung in einer darunter liegenden Schicht unterbrochen werden, kann diese an, während der Sitzung angelegten, Punkten wiederhergestellt werden.

6.2.1.6 Darstellungsschicht

Die Darstellungsschicht ermöglicht die einheitliche Darstellung von Informationen auf unterschiedlichen Systemen. Sie enthält Methoden zur Kodierung und Konvertierung um die Unterschiede zwischen verschiedenen, angewendeten Standards auszugleichen. Auch sind die Funktionen zur Datenkompression und -verschlüsselung in der Darstellungsschicht angesiedelt.

6.2.1.7 Anwendungsschicht

Bei der Anwendungsschicht handelt es sich um anwendungsspezifische Funktionen, welche den Zugriff auf die darunterliegenden Schichten, sowie die Ein- und Ausgabe von Informationen ermöglichen.

6.2.2 Implementierung der physischen und der Bitübertragungsschicht

Wie bereits erwähnt ermöglicht der Einsatz der weit verbreiteten Standards einen einfachen und höchst kompatiblen Aufbau. Ethernet, als Grundstein des modernen Internets, steht großflächig zur Verfügung, es gibt nahezu überall bereits vorhandene Infrastruktur, sowie Endgeräte welche IEEE 802.3 unterstützen. Des Weiteren gibt es eine Vielzahl von Gateways um den Übergang in andere Netzwerke zu ermöglichen, so z.B. in ein Wireless LAN nach IEEE 802.11. Der Server kann einfach an einen Router angeschlossen werden, welcher über beide Standards verfügt und ermöglicht so den kabellosen Zugriff auf die Smart Home Applikation.

6.2.3 Implementierung der Vermittlungs- und Transportschicht

6.2.3.1 Das Internetprotokoll (IP)

Die Vermittlungsschicht dient der Adressierung der Teilnehmer. In diesem Fall kommt das Internetprotokoll in der Version vier (IPv4) zum Einsatz. Es ist ebenso wie das Ethernet essentieller Bestandteil des Internets und dementsprechend genauso großflächig einsetzbar. Auf die Implementierung des Nachfolgestandards IPv6 wurde bewusst verzichtet, da alle gängigen Systeme IPv4 unterstützen, jedoch noch nicht von allen die Version 6 unterstützt wird. Der Server verwendet dabei die fest eingestellte, nicht-öffentliche Adresse 192.168.0.101/24, der Client muss sich dementsprechend im gleichen Netzwerk mit der Adresse 192.168.0.0/24 befinden sofern eine direkte Verbindung hergestellt werden soll, beispielhaft wurde hier 192.168.0.100/24 gewählt (Abbildung 6.2).

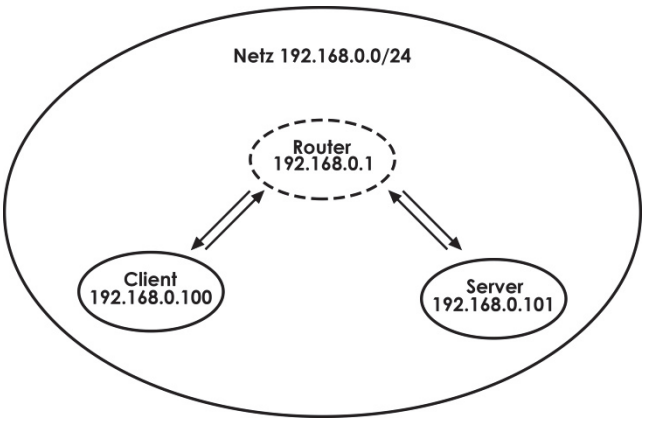


Abbildung 6.2: Netzwerkkonfiguration auf Vermittlungsebene

6.2.3.2 Das Transmission Control Protocol (TCP)

Bei TCP handelt es sich um ein verbindungsorientiertes Protokoll, welches Verbindungen zwischen Computern mit Hilfe von Sockets herstellt. Dabei findet die Übertragung in drei Phasen statt, dem Verbindungsaufbau, der Datenübertragung und dem Verbindungsabbau. Die Übertragung erfolgt dabei segmentiert, wobei die Reihenfolge der einzelnen Datenpakete erhalten bleibt (Abbildung 6.6). Verloren gegangene oder fehlerhafte Pakete werden erkannt und erneut übertragen, sodass die Übertragung als sicher angesehen werden kann, wobei sicher nicht mit geheim gleichzusetzen ist.

Der Verbindungsaufbau findet bei TCP in Form des sogenannten „Three-Way-Handshakes“ statt. Dabei Sendet der Client eine Verbindungsanfrage an den Server, in dem das „SYN“-Flag übertragen wird. Dieser antwortet mit „SYN,ACK“, hergestellt ist sie allerdings erst, wenn der Client dies dem Server noch einmal bestätigt, in dem „ACK“ übertragen wird (Abbildung 6.3).

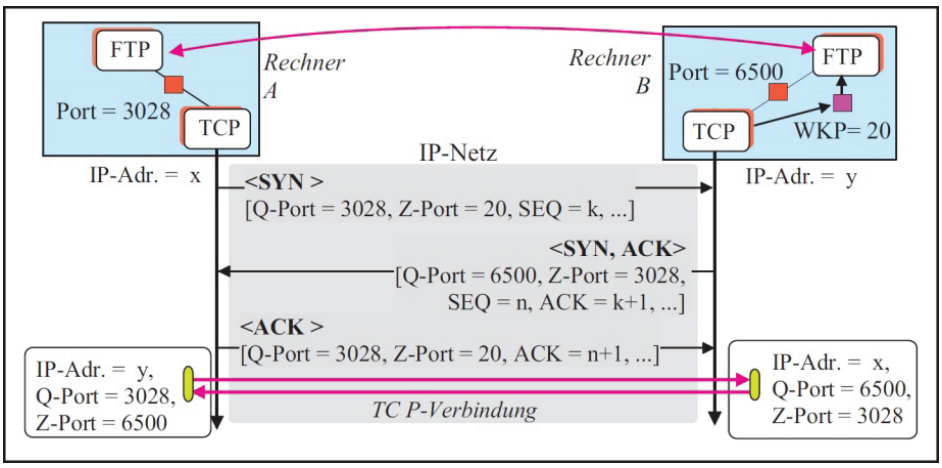


Abbildung 6.3: Three-Way-Handshake des TCP[24]

Während des Handshakes werden alle für die Verbindung nötigen Informationen, wie zum Beispiel die zu verwendenden Ports, ausgetauscht. Des Weiteren legen beide Teilnehmer eine Sequenznummer fest. Mit Hilfe dieser Sequenznummer können die einzelnen Übertragungen identifiziert werden. Dabei wird die Sequenznummer der Pakete während des Handshakes um eins erhöht, um die Übertragung zu bestätigen. Wählt Rechner A also die Sequenznummer „k“, so bestätigt Rechner B dies durch Senden der Quittungsnummer „k+1“ und übermittelt seine eigene Sequenznummer „n“. Rechner A quittiert diese nun mit „n+1“. (Abbildung 6.3)

Anschließend können Daten zwischen den Teilnehmern ausgetauscht werden. Dies geschieht prinzipiell ähnlich wie die Übertragung des Handshakes. Der Unterschied ist, dass Pakete jetzt quittiert werden, indem die empfangene Sequenznummer um die Größe des Paketes in Bytes erhöht wird. Wird die „Maximum-Segment-Lifetime“, also die maximale Lebensdauer eines Segments, überschritten bevor der Sender eine Quittierung erhalten hat, so wird die Übertragung wiederholt bis die Quittierung erfolgt ist, oder die maximale, durch den Benutzer festgelegte, Anzahl an Wiederholungen erreicht ist (Abbildung 6.4).

[24] [2] Seite 123

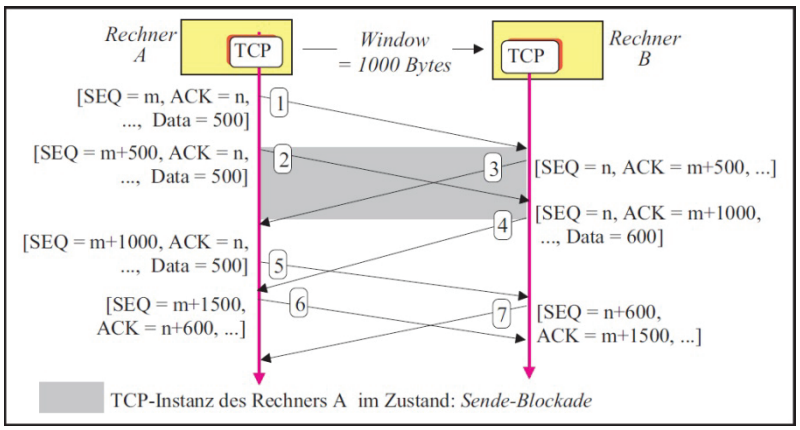


Abbildung 6.4: TCP-Datenübertragung[25]

Der Abbau der Verbindung, auch „Teardown“ genannt, wird bei TCP eingeleitet, indem einer der Teilnehmer, z.B. Rechner A, ein Paket mit gesetztem „FIN“-Flag und der Sequenznummer „n“ übermittelt. Rechner B beantwortet dies mit „FIN,ACK“ und der Quittungsnummer „n+1“. Anschließend überträgt Rechner B die Flags „FIN,ACK“ und seine Sequenznummer „n“. Antwortet Rechner A mit „ACK“ und der Quittungsnummer „n+1“, ist die Verbindung abgebaut und der Socket kann neu vergeben werden (Abbildung 6.5). Wird die Verbindung während des Abbaus unterbrochen, so verbleibt diese in einem halboffenen Zustand. Dies führt dazu, dass keine Daten mehr übertragen werden können, der Socket allerdings auch nicht neu vergeben werden kann, da er weiterhin durch die Verbindung blockiert wird, bis er durch eine Zeitüberschreitung vom Betriebssystem oder dem Anwender freigegeben wird.

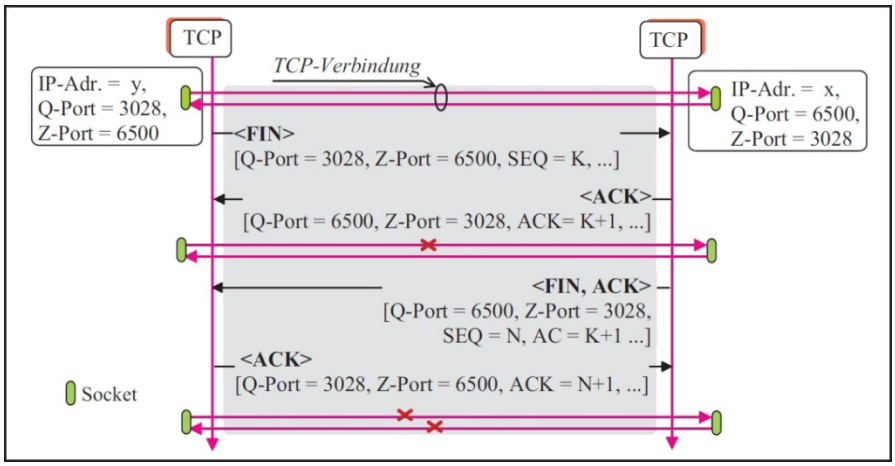


Abbildung 6.5: Verbindungsabbau des TCP[26]

[25] [2] Seite 130

Abbildung 6.6 zeigt den kompletten Lebenszyklus einer TCP-Verbindung.

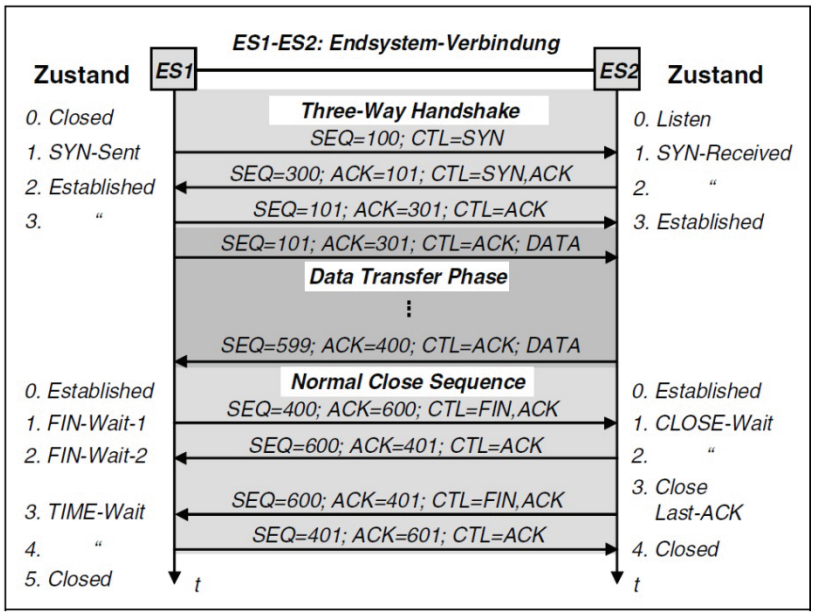


Abbildung 6.6: Ablauf einer TCP-Übertragung[27]

Pro Paket stehen 1460 Byte für Nutzdaten zur Verfügung, welche nahezu voll ausgeschöpft werden. Um eine Segmentierung der Pakete zu vermeiden, welches zu Problemen mit dem verwendeten TCP-Stack des LPC1768 führen könnte, wird diese Größe durch das Protokoll der Anwendungsschicht eingehalten. Theoretisch bietet IP die Möglichkeit 64 KiB zu übertragen, Ethernet gestattet allerdings nur 1500 Byte pro Datenpaket - abzüglich der Header für IP[28] und TCP[29] von jeweils typischerweise 20 Byte, ergeben sich also 1460 Byte pro Übertragung, was für diese Anwendung auch völlig ausreichend ist. Findet die Kommunikation über DSL statt, fallen weitere 8 Byte für die Header

[26] [2] Seite 125

[27] [4] Seite 324

[28] [16]

[29] [17]

des Point-to-Point Protocols[30] an, sodass sich die Größe der Nutzdaten auf 1452 verringern würde.

6.2.4 Implementierung der Anwendungsschicht

Auf der Anwendungsschicht werden eigene zur Funktion der Software nötige Protokolle implementiert. In diesem Fall wurde eines von der Professur Kommunikationstechnik entwickeltes Protokoll verwendet. Es besteht aus einem einfachen Handshake um unzulässige Verbindungen auszuschließen, sowie der anschließenden Datenübertragung (Abbildung 6.7). Des Weiteren stehen eine PC-Software in C#, sowie eine Firmware in C für die Eingabe, Ausgabe und Verarbeitung der übertragenen Daten zur Verfügung.

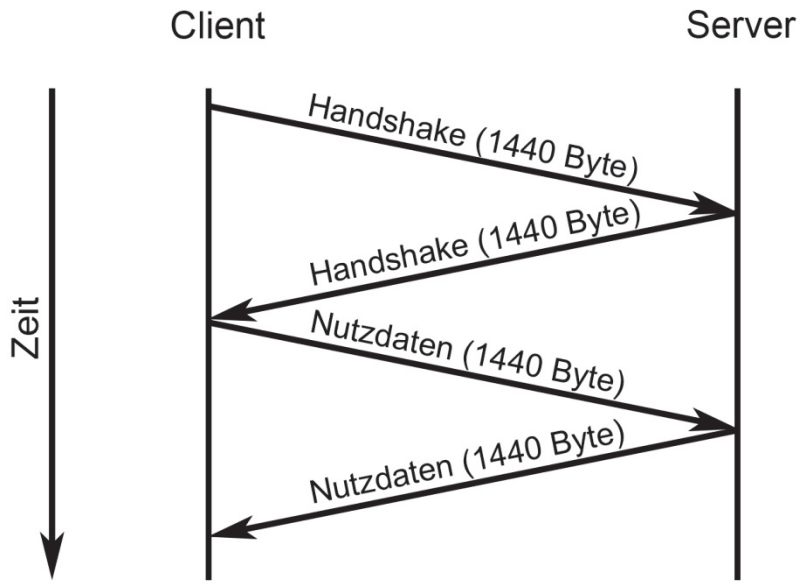


Abbildung 6.7: Anwendungsprotokoll

Da es sich hier nur um eine erste Entwicklungsstufe handelt, wurde auf den Einsatz von Verschlüsselungs- und Authentifizierungsmechanismen verzichtet, prinzipiell ist die Möglichkeit allerdings vorhanden. Um eine Segmentierung zu vermeiden, wurde eine maximale Größe der Daten von 1440 Bytes festgelegt.

[30] Nach [18] und [19] bestehend aus 2 Bytes für das Feld „Protocol“ der PPP-Einkapselung und 6 Bytes für den PPPoE-Header

Das Protokoll setzt eine Bereits bestehende Verbindung bis zur Transportschicht voraus. Ist diese Vorhanden sendet der Client den ersten Teil des Handshakes (Tabelle 6.1).

Byte	0	1	2	…	1439
Inhalt	„B“	„S“	„H“		„H“

Tabelle 6.1: Handshake des Anwendungsprotokolls

Der Server vergleicht die eintreffenden Daten mit dem Handshake und sendet diesen bei Übereinstimmung an den Client zurück. Das ist das Signal für die Clientsoftware die eigentlichen Nutzdaten zu übermitteln. Der Server verarbeitet die Daten, aktualisiert diese und sendet sie zurück an den Client (Tabelle 6.2), welcher wiederum die aktualisierten Daten des Servers verarbeitet. Im Anschluss wird die TCP-Verbindung abgebaut.

Byte	0	1	2 bis 7	8 bis 17	18-21
Inhalt	„B“	„S“	6x dig. Ausgang „0“	2x DAC „0.000“	4x dig. Eingang „0“
Byte			**22 bis 41**	**42-44**	
Inhalt			4x ADC „0.000“	Uhrzeit (als Character) „HMS“	

Tabelle 6.2: Nutzdaten des Anwendungsprotokolls

Abbildung 6.8 zeigt einen kompletten Zyklus einer Übertragung. Sowohl der Verlauf des TCP-Protokolls, als auch des Anwendungsprotokolls sind ersichtlich, wobei in den Paketen 8 und 10 der Handshake und in den Paketen 11 und 13 die Nutzdaten übertragen werden.

No.	Time	Source	Destination	Protocol	Length	Info
5	8.696940000	192.168.0.100	192.168.0.101	TCP	66	1284→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1452 WS=4 SACK_PERM=1
6	8.720966000	192.168.0.101	192.168.0.100	TCP	60	80→1284 [SYN, ACK] Seq=0 Ack=1 Win=1460 Len=0 MSS=1460
7	8.721053000	192.168.0.100	192.168.0.101	TCP	54	1284→80 [ACK] Seq=1 Ack=1 Win=69696 Len=0
8	8.723194000	192.168.0.100	192.168.0.101	TCP	1494	[TCP segment of a reassembled PDU]
9	8.783939000	192.168.0.101	192.168.0.100	TCP	60	80→1284 [ACK] Seq=1 Ack=1441 Win=1460 Len=0
10	8.783941000	192.168.0.101	192.168.0.100	TCP	1494	[TCP segment of a reassembled PDU]
11	8.784097000	192.168.0.100	192.168.0.101	TCP	1494	[TCP segment of a reassembled PDU]
12	8.795602000	192.168.0.101	192.168.0.100	TCP	60	80→1284 [ACK] Seq=1441 Ack=2881 Win=1460 Len=0
13	8.796902000	192.168.0.101	192.168.0.100	TCP	1494	[TCP segment of a reassembled PDU]
14	8.797030000	192.168.0.100	192.168.0.101	TCP	54	1284→80 [FIN, ACK] Seq=2881 Ack=2881 Win=69696 Len=0
15	8.820498000	192.168.0.101	192.168.0.100	TCP	60	80→1284 [FIN, ACK] Seq=2881 Ack=2882 Win=1460 Len=0
16	8.820580000	192.168.0.100	192.168.0.101	TCP	54	1284→80 [ACK] Seq=2882 Ack=2882 Win=69696 Len=0

Abbildung 6.8: Protokoll einer Datenübertragung

6.3 Firmware

Bei der Firmware handelt es sich um ein C-Programm, welches mit Hilfe der KEIL-IDE µVision 4 erstellt wurde. Dabei kommen einige proprietäre Bibliotheken von KEIL, als auch einige selbst erstelle Bibliotheken zum Einsatz. Das Programm selbst wurde in Form eines Zustandsautomaten realisiert, welcher aus vier verschiedenen Zuständen besteht.

6.3.1 Zustandsautomat

Einfache Firmware für Mikrocontroller wird häufig als Zustandsautomat realisiert, da es eine einfache Möglichkeit zur Strukturierung von Programmen ist. Auf die Verwendung eines Betriebssystems wurde verzichtet, da die Controller nur wenige Aufgaben abarbeiten und das Planen von Aufgaben nicht nötig ist. Generell ist eine Implementierung häufig auch zu teuer oder aufgrund der begrenzten Ressourcen nicht möglich ist.

Ein weiterer Vorteil der individuellen Softwarearchitektur ist die Unabhängigkeit von anderen Entwicklern und Firmen und die dadurch gegebene Langzeitstabilität der Software. Das heißt, die Software ist unabhängig von eventuellen Updates des Herstellers und kann somit nach der einmaligen Implementierung unbegrenzt ohne Änderungen verwendet werden.

Der hier verwendete Moore-Automat besitzt vier Zustände, wobei Zustand S_0 nur einmalig nach dem Starten der Anwendung durchlaufen wird (Abbildung 6.9).

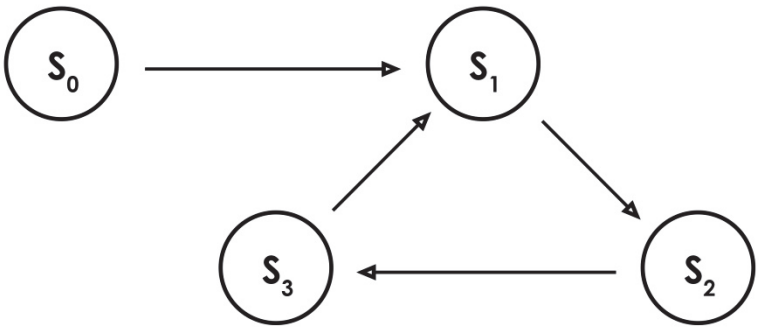


Abbildung 6.9: Automatendiagramm der Firmware

Die Zustände sind in diesem Fall wie folgt definiert:

- S_0: - Initialisierung aller Variablen, Funktionen und der Peripherie, Starten des TCP-Servers
- S_1: - Warten auf Verbindungsanfrage, Herstellen der Verbindung
- S_2: - Überprüfung und Beantwortung des Handshakes
- S_3: - Empfang von Daten vom Client, Senden von Daten an Client, Verbindungsabbau

Auf Details zu den einzelnen Zuständen wird in den folgenden Abschnitten näher eingegangen.

6.3.2 Schematischer Aufbau

Die Firmware lässt sich in einzelne, funktionale Blöcke unterteilen (Abbildung 6.10). Im Block Initialisierung werden die nötigen Einstellungen für die verwendete Peripherie wie A/D-Wandler , D/A-Wandler , Netzwerkschnittstelle , Display und die Timer vorgenommen. Im Anschluss wird der Block „Netzwerk“ zyklisch ausgeführt. Dieser steht dabei für die durch die Netzwerkbibliothek von ARM/KEIL vorgegebenen Funktionen. Wird durch eine der Funktionen ein Ereignis ausgelöst, wird der Block „TCP-Stack“, welcher die TCP-Callback-Funktion[31] enthält aufgerufen und entsprechend des Ereignisses weiter verfahren.

Wurde ein Handshake empfangen, wird der entsprechende Block ausgeführt und die Schleife wird weiter ausgeführt. Wurde der Empfang des gesendeten Handshakes und der Empfang von Nutzdaten bestätigt, ruft der Block „TCP-Stack“ den Block „Nutzdaten“ auf, welcher die empfangenen Daten verarbeitet und die Nutzdaten aktualisiert, anschließend sorgt „Nutzdaten“ für das versenden der Aktualisierten Daten. Während der gesamten Verarbeitung, können Interrupts ausgelöst werden, welche im entsprechenden Block zusammengefasst werden. In diesem Fall beinhaltet dieser den Abschluss einer AD-Wandlung.

[31] Callback-Funktion, auch Rückruffunktion – wird Funktionen als Parameter übergeben und kann unter bestimmten Bedingungen aufgerufen werden

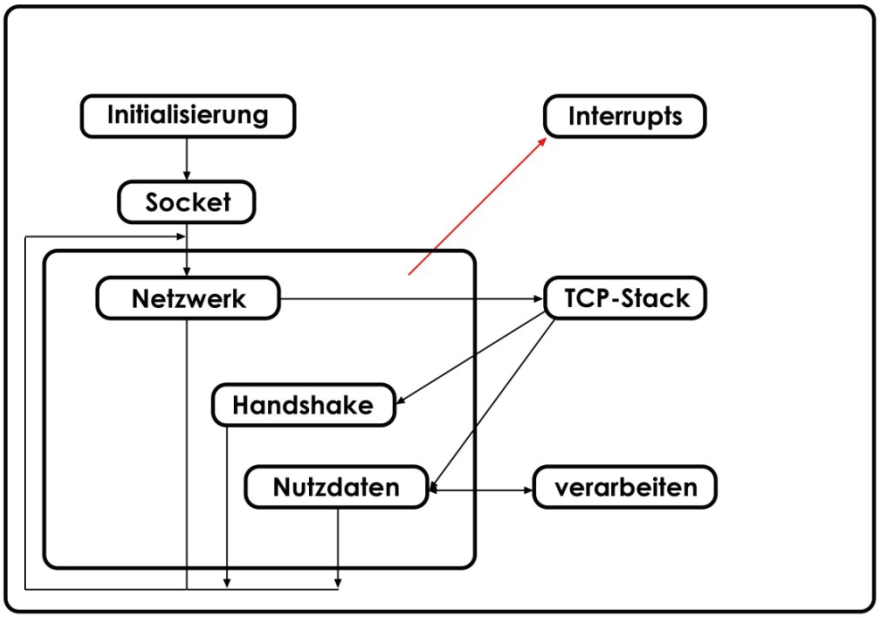


Abbildung 6.10: Schematischer Aufbau der Firmware

6.3.3 LC-Display

Angeschlossen an das Modul Chip1768 ist ein 16x4 LC-Display, um die vom Server bereitgestellten Informationen grafisch darstellen zu können. Dabei wird das Modul „GDM2004D“ des Herstellers „Xiamen Ocular“ verwendet. Auf diesem Modul befindet sich die nötige Beschaltung für den Betrieb des LC-Displays, sowie ein, zum weit verbreiteten Controller HD44780 kompatibler Controller, der ST7066U-0A, zur Ansteuerung des Displays. Für den Betrieb werden je nach Konfiguration 12 oder 16 Pins benötigt (Tabelle 6.3).

Pin	Symbol	Beschreibung	Portpin am µC
1	V_{SS}	Spannungsversorgung (GND)	GND
2	V_{DD}	Spannungsversorgung (+5V)	+5V
3	V_0	Kontrasteinstellung	-
4	RS	„register select“ – Signal	P0.18
5	R/W	„read/write select“- Signal	P0.1
6	E	„operation enable“ – Signal	P0.0
7-10	DB0-DB3	Datenleitung 0-3, ungenutzt bei 4-Bit-Modus	GND
11-14	DB4-DB7	Datenleitung 4-7	P0.6 .. P0.9
15	LED+	Displaybeleuchtung (+5V)	+5V
16	LED-	Displaybeleuchtung (GND)	GND

Tabelle 6.3: Pinbelegung des LCD-Moduls GDM2004D

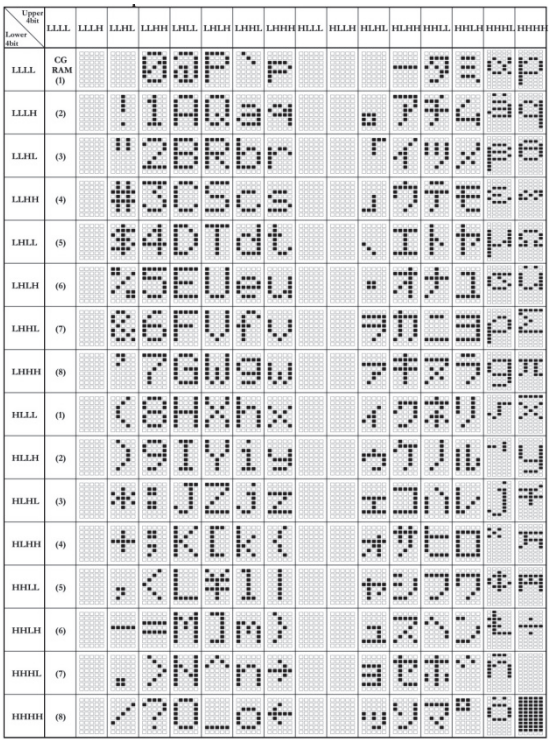


Abbildung 6.11: Zeichntabelle des ST7066U-0A[32]

[32] Vgl. [5] Seite 11

Das Display ist in einer Matrix organisiert, welche über den Controller angesteuert werden kann. Dazu besitzt der ST7066U-0A elf digitale Eingänge, davon acht für Anzeigedaten, welche entsprechend der Übersicht in **Abbildung 6.11** geschaltet werden müssen um die zugehörigen Zeichen darzustellen. Des Weiteren gibt es eine Reihe von Befehlen, welche damit umgesetzt werden können. Diese dienen zum einen zur Initialisierung sowie der Konfiguration, zum anderen der Steuerung des Controllers (Abbildung 6.12) im Betrieb.

Instruction	Instruction code										Description	Execution Time (fosc= 270 KHZ
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Write "20H" to DDRA and set DDRAM address to "00H" from AC	1.53ms
Return Home	0	0	0	0	0	0	0	0	1	-	Set DDRAM address to "00H" From AC and return cursor to Its original position if shifted. The contents of DDRAM are not changed.	1.53ms
Entry mode Set	0	0	0	0	0	0	0	1	I/D	SH	Assign cursor moving direction And blinking of entire display	39us
Display ON/ OFF control	0	0	0	0	0	0	1	D	C	B	Set display (D), cursor (C), and Blinking of cursor (B) on/off Control bit.	
Cursor or Display shift	0	0	0	0	0	1	S/C	R/L	-	-	Set cursor moving and display Shift control bit, and the Direction, without changing of DDRAM data.	39us
Function set	0	0	0	0	1	DL	N	F	-	-	Set interface data length (DL: 8-Bit/4-bit), numbers of display Line (N: =2-line/1-line) and, Display font type (F: 5x11/5x8)	39us
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Set CGRAM address in address Counter.	39us
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Set DDRAM address in address Counter.	39us
Read busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Whether during internal Operation or not can be known By reading BF. The contents of Address counter can also be read.	0us
Write data to Address	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Write data into internal RAM (DDRAM/CGRAM).	43us
Read data From RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Read data from internal RAM (DDRAM/CGRAM).	43us

Abbildung 6.12: Befehlsübersicht des ST7066U-0A[33]

Der auf der Matrix des Displays darzustellende Inhalt wird im DDRAM des Controllers gespeichert, wobei jedes Byte im Speicher einer Position in der Matrix und somit auf dem Display entspricht (Abbildung 6.13). Die Adresse „00“ steht dabei für die erste Stelle in der ersten Zeile, Adresse „40“ für die erste Stelle in der zweiten Zeile, usw. Trotz der hier dargestellten vier Zeilen, wird das Display vom Controller als zweizeilig betrachtet.

[33] Vgl. [5] Seite 7

DDRAM address:

Display position

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53
14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67

DDRAM address

Abbildung 6.13: Speicher des ST7066U-0A[34]

6.3.3.1 Initialisierung des LC-Displays

Die Initialisierung des Displays erfolgt mit Hilfe der in **Abbildung 6.12** dargestellten Befehle. Um die Anzahl der benötigten Pins zu reduzieren, kann das Display im 4-Bit-Modus angesteuert werden. Dazu werden die Befehle in 4-Bit lange Abschnitte geteilt und einzeln übertragen, wobei zuerst die oberen und anschließend die unteren 4-Bit übertragen werden. Um das Senden von Befehlen zu vereinfachen, wurde die Funktion „lcd_command(int cmd)“ erstellt (Quellcode 6.1).

```
01 void lcd_command(int cmd)
02 {
03     LPC_GPIO0->FIOSET |= (1<<0);
04     LPC_GPIO0->FIOCLR |= (1<<18);
05     LPC_GPIO0->FIOCLR |= (1<<1);
06     LPC_GPIO0->FIOCLR |= (15<<6);
07     LPC_GPIO0->FIOSET |= (((cmd>>4)&0x0F)<<6);
08     LPC_GPIO0->FIOCLR |= (1<<0);
09     delay_ms(50);
10     if ( cmd != 0x20 && cmd != 0x30)
11     {
12         LPC_GPIO0->FIOSET |= (1<<0);
13         LPC_GPIO0->FIOCLR |= (15<<6);
14         LPC_GPIO0->FIOSET |= (((cmd)&0x0F)<<6);
15         LPC_GPIO0->FIOCLR |= (1<<0);
16     }
17     delay_ms(50);
18 }
```

Quellcode 6.1: Funktion lcd_command() für Bibliothek des Displays

[34] Vgl. [5] Seite 5

Das Senden von Befehlen folgt einem festgelegten Schema:

1. Setzen des Enable-Bits auf HIGH (Quellcode 6.1, Zeile 3)
2. Setzen des RS-Bits auf LOW (Quellcode 6.1, Zeile 4)
3. Setzen des R/W-Bits auf LOW (Quellcode 6.1, Zeile 5)
4. Setzen der Datenbits (Quellcode 6.1, Zeile 6 f)
5. Setzten des Enable-Bits auf LOW (Quellcode 6.1, Zeile 8)
6. Punkte 1-5 analog für zweites Halbbyte (Quellcode 6.1, Zeile 10 ff)

Dabei bestimmt das RS-Bit, ob die Daten als Befehl oder als Zeichen interpretiert werden sollen, das R/W-Bit legt fest ob geschrieben oder gelesen wird und die Datenbits sind entsprechend der Vorgaben zu setzen (Abbildung 6.11 und Abbildung 6.12) , Die fallende Flanke am Eingang des Enable-Bits löst die Verarbeitung durch den Controller aus.

Im Anschluss wird einige Millisekunden gewartet, um sicherzugehen, dass die Operationen des Controllers ausgeführt wurden.

Um Zeichen auf dem Display ausgeben zu können, müssen einige Einstellungen vorgenommen werden. Dazu wurde die Funktion „lcd_init()“ erstellt (Abbildung 6.2). Diese muss einmalig beim Starten der Anwendung ausgeführt werden und konfiguriert den Controller für die anschließenden Aufgaben.

```
01 void init_lcd(void)
02 {
03     LPC_GPIO0->FIODIR |= (1<<0) | (1<<1) | (15<<6) | (1<<18);
04     delay_ms(200);
05     lcd_command(0x30);
06     lcd_command(0x30);
07     lcd_command(0x30);
08     lcd_command(0x20);
09     lcd_command(0x2C);
10     lcd_command(0x14);
11     lcd_command(0x06);
12     lcd_command(0x0C);
13     lcd_command(0x01);
14 }
```

Quellcode 6.2: Funktion lcd_init() für Bibliothek des Displays

Als erstes werden die verwendeten Ports als Ausgänge definiert (Quellcode 6.2, Zeile 3). Anschließend wird der Controller durch dreimaliges Senden des Befehls „0x30“ zurückgesetzt (Quellcode 6.2, Zeile 5-7). Der Befehl „0x20“ versetzt das Display in den 4-Bit-Modus, „0x2C“ entspricht den Optionen für ein 2-Zeiliges Display, sowie die Zeichengröße 5x11 Bildpunkte (Quellcode 6.2, Zeile 8 f.).

Entsprechend **Abbildung 6.12: Befehlsübersicht des ST7066U-0A** setzt sich der Befehl wie folgt zusammen:

Befehl	RS	R/W	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Function set	0	0	0	0	1	DL	N	F	-	-

Tabelle 6.4: Beispielhafte Erklärung der Befehlsstruktur des ST7066U-0A

Die Platzhalter DL, N und F sind die folgt definiert:

- DL – 1 = 8-Bit Modus / 0 = 4-Bit Modus
- N – 1 = zweizeiliges Display / 0 = einzeiliges Display
- F – 1 = 5x11 Zeichenformat / 0 = 5x8 Zeichenformat

Der Controller soll im 4-Bit Modus angesteuert werden um so wenig wie nötig Portpins zu verwenden, DL hat also den Wert „0“. Da das Display vier Zeilen besitzt, der Speicher jedoch wie ein zweizeiliges Display aufgebaut ist, wird für N „1“ gewählt. Die Zeichengröße des Displays ist 5x11, F bekommt somit ebenfalls den Wert „1“. Die untersten zwei Bits werden ignoriert, ihnen muss also kein Wert zugewiesen werden. Da die Ports vor jedem Befehl auf „0“ gesetzt werden, behalten diese ihren Zustand. Setzt man den Befehl entsprechend des Schemas zusammen, erhält man für den Zustand der Datenbits das Byte „00101100“, Hexadezimal „0x2C“. Alle anderen Befehle sind analog organisiert.

Das Verhalten des Cursors wird mit der Funktion „0x14“ bestimmt, in diesem Fall orientiert er sich am linken Bildrand und wird beim Schreiben automatisch eine Stelle nach rechts verschoben. Die Befehle „0x06“ sowie „0x0C“ sorgen dafür, dass das Display eingeschaltet und der Cursor nicht sichtbar ist. Abschließend wird das Display mit „0x01“ geleert (Quellcode 6.2, Zeile 10-13).

6.3.3.2 Verwendung der Bibliothek

Neben den in 6.3.3.1 bereits erwähnten Funktionen, stellt die Bibliothek einige weitere zur Verfügung. Diese Funktionen sollen hier aber nur hinsichtlich ihres Namens, ihrer Aufgaben sowie Syntax erwähnt werden. Der komplette Quellcode ist in der **Anlage, Teil 2** zu finden.

Funktion	Beschreibung
lcd_cursor(int x,int y)	Positioniert den Cursor bei Koordinate x,y
lcd_putchar(unsigned char c)	schreibt einen einzelnen Buchstaben „c“
lcd_putstring(unsigned char *s)	schreibt einen String „*s”
lcd_write_number(int data)	schreibt die Ziffer „data“
lcd_check_site(char *site,char *steuerstring)	schreibt Inhalt von Seite „*site”

Tabelle 6.5: Weitere Funktionen der Bibliothek für den ST7066U-0A

Die Parameter „x“ und „y“ der Funktion „lcd_cursor()“ können, entsprechend der Displaygröße, Werte zwischen 0 und 20 bzw. 0 und 3 annehmen.

Die Funktion lcd_check_site() ermöglicht es, Informationen auf verschiedenen Seiten darzustellen und zwischen ihnen zu wechseln. Dazu werden zwei Taster abgefragt, welche die Seitenzahl, welche der Funktion über den Zeiger „*site“ mittgeteilt wird, erhöhen oder verringern. Welche Inhalte dargestellt werden, wird in der Funktion selbst in der Bibliothek festgelegt. In der momentanen Version werden die Uhrzeit, sowie die analogen und digitalen Ein- und Ausgänge dargestellt.

6.3.4 Hauptprogramm

Zu Beginn des Programms, werden einige Definitionen und Initialisierungen vorgenommen, im Anschluss geht das Programm in eine unendliche While-Schleife über, welche zyklisch Werte misst und ausgibt und auf Daten an der Ethernetschnittstelle wartet.

Im Folgenden sollen einige prägnante Ausschnitte der Software erläutert werden.

Zu Beginn wird im Hauptprogramm die nötige Peripherie wie LC-Display, SysTickTimer, LEDs, der Netzwerkstack, ADC, DAC, sowie einige Systemvariablen initialisiert. Aus Gründen der Übersichtlichkeit wurden die nötigen Befehle in eigene Funktionen ausgegliedert, welche im kompletten Quelltext im **Anhang, Teil 2** zu finden sind. Da zukünftig der Status der Initialisierung auf dem LC-Display dargestellt werden soll und dieser zu schnell für den Betrachter verschwinden würde, wurden mit Hilfe von Softwareschleifen kurze Pausen von 200ms eingefügt um dem Betrachter das Starten der Anwendung visuell anzeigen zu können.

```
01    init_lcd();
02    delay_ms(200);
03    init_systicktimer();
04    init_network();
05    init_rtc();
06    delay_ms(200);
07    delay_ms(200);
08    init_led();
08    delay_ms(200);
10    init_TcpNet();
11    delay_ms(200);
12    init_string(string);
13    delay_ms(200);
14    init_handshake(handshake);
15    delay_ms(200);
16    init_adc();
17    init_dac();
18    delay_ms(200);
```

Quellcode 6.3: Initialisierung der Firmware

Als nächstes wird ein TCP-Socket initialisiert. Der Socket kann sowohl als Client als auch als Server fungieren, welches durch den Parameter „TCP_TYPE_CLIENT_SERVER“ definiert wird. Des Weiteren wird eine Callback-Funktion angegeben, welche aufgerufen wird sobald eines der in der Library definierten Ereignisse ausgelöst wird[35]. Anschließend wird auf die Existenz des Sockets geprüft und der Server auf dem angegeben Port mit Hilfe der Listen-Funktion gestartet.

```
01    socket_tcp = tcp_get_socket (TCP_TYPE_CLIENT_SERVER, 0,
      1, tcp_callback);

02    if (socket_tcp != 0) {
03          tcp_listen (socket_tcp, PORT);
04    }
```

Quellcode 6.4: Socket-Erstellung und Starten des Servers

Im nächsten Programmabschnitt wird eine allzeit wahre While-Schleife abgearbeitet, in der der TCP-Stack entsprechend der Dokumentation der Library gepollt, also regelmäßig abgefragt und bei entsprechend gesetzten Zustandsvariablen in andere Zustände, des hier verwendeten Automaten, gewechselt wird. Der Automat besteht in diesem Fall aus

[35] Vgl. Kapitel 5.1.4

vier Zuständen. Der erste Zustand ist die Initialisierung, als nächstes folgt die While-Schleife. Zustand drei und vier dienen jeweils der Kommunikation mittelst TCP/IP und entsprechen gleichzeitig die zwei Stufen des Protokolls auf Anwendungsebene, nämlich dem Handshake in Zustand drei und der eigentlichen Datenübertragung in Zustand vier.

```
01    while (1) {
02          timer_poll ();
03          main_TcpNet ();
04          if ( connected ) {
05                send_data(handshake,socket_tcp);
06                ready_to_receive = __TRUE;
07                connected = __FALSE;
08          }
09          if ( ready_to_send ) {
10                steuerstring_setzen(string);
11                send_data(string,socket_tcp);
12                ready_to_send = __FALSE;
13          }
14          lcd_check_site(&lcd_site,(char*)string);
15    }
```

Quellcode 6.5: Hauptschleife der Firmware

Die Variable „conncted“ steht dafür für eine vollständig hergestellte Verbindung. Ist dies der Fall, wird der Handshake gesendet und der Automat wechselt in den nächsten Zustand, in dem die Daten an den Client übertragen werden, nachdem sie aktualisiert wurden. Die Verbindung wird durch den Client beendet und der Server wieder in den Zustand zwei überführt.

Im Zustand S_1 werden die Funktionen „timer_poll“ und „main_TcpNet“ ausgeführt. Die Funktion „timer_poll“ führt in regelmäßigen Abständen, in diesem Fall 10 ms, die proprietäre Funktion „timer_tick“ aus, welche durch die Ethernet-Bibliothek von KEIL vorgegeben wird. Des Weiteren wird die Konvertierung des ADC zyklisch, alle 50 ms gestartet.

```
01 void timer_poll(void)
02 {
03     /* System tick timer running in poll mode */
04     if (SysTick->CTRL & 0x10000)
05     {
06
07         timer_tick ();
08         AD_done++;
09         if ( AD_done == 5 )
10         {
11             AD_done = 0;
12             LPC_ADC->ADCR |= (1<<16);
13             LPC_ADC->ADINTEN     |=  ( 1 <<  0);
14             LPC_ADC->ADINTEN     &=  ~(1 <<  8);
15         }
16     }
17 }
```

Quellcode 6.6: Funktion timer_poll

Die ebenfalls durch die Bibliothek vorgegebene Funktion „main_TcpNet“ muss ebenso zyklisch aufgerufen werden, um die Ethernetschnittstelle auf die vordefinierten Ereignisse zu prüfen. Trifft eines dieser Ereignisse ein, so wird die bei der Initialisierung festgelegte Callback-Funktion „tcp_callback“ aufgerufen.

```
01 U16 tcp_callback (U8 soc, U8 evt, U8 *ptr, U16 par)
02 {
03     par = par;
04
05     if (soc != socket_tcp)
06     {
07         return (0);
08     }
09     switch (evt)
10     {
11     case TCP_EVT_DATA:
12         data_received(ptr);
13         break;
14
15     case TCP_EVT_CONREQ:
16         return (1);
17
18     case TCP_EVT_CONNECT:
19         return (1);
20     }
21     return (0);
22 }
```

Quellcode 6.7: TCP-Callback

Der KEIL-Stack übergibt dieser Funktion den verwendeten Socket, das ausgelöste Event, einen Pointer auf die übertragenen Daten und den Port des Clients, wenn es sich um eine Verbindungsanfrage handelt, ansonsten die Größe der empfangenen Daten. Zuerst wird der Socket überprüft um sicherzustellen, dass die Verbindung auch auf dem festgelegten Socket aufgerufen wird (Quellcode 6.7, Zeile 5 ff). Im Anschluss wird überprüft welches Ereignis aufgerufen wurde. Bei „TCP_ECVT_CONREQ“, also einer Verbindungsanfrage wird die Funktion beendet und „1“ zurückgegeben. Dies Signalisiert, dass die Verbindung angenommen werden soll (Quellcode 6.7, Zeile 15 ff). Bei „TCP_EVT_CONNECT“ wurde die Verbindung erfolgreich hergestellt, es wird erneut „1“ zurückgegeben und auf eintreffende Daten gewartet (Quellcode 6.7, Zeile 18 ff). Werden Daten empfangen, was dem Ereignis „TCP_EVT_DATA“ entspricht, wird die Funktion „data_received“ aufgerufen (Quellcode 6.7, Zeile 11 ff).

Wurde die Verbindung erfolgreich Hergestellt und anschließend Daten empfangen, werden diese mit dem Handshake verglichen. Bei Übereinstimmung setzt die Funktion „send_handshake“ die Variable „connected“ des Hauptprogramms auf „__TRUE“ und der Automat geht in den Zustand S_2 über (Quellcode 6.8, Zeile 3 ff).

```
01 void data_received (U8 *buffer)
02 {
03     if ( !memcmp(buffer,handshake,1440) )
04     {
05         send_handshake();
06         return;
07     }
08     if ( ready_to_receive )
09     {
10         memcpy(string,buffer,1440);
11         steuerstring_auswerten(string);
12         send_steuerstring();
13         ready_to_receive =__FALSE;
14     }
15     return;
16 }
```

Quellcode 6.8: Auswertung der Empfangenen Daten

Der Zustand S_2 ist durch die Übertragung des Handshakes an den Client gekennzeichnet. Wurde dieser gesendet, wechselt das Programm in den Zustand S_3. Im Zustand S_3 wartet der Controller erneut auf Daten am Ethernetport und erwidert den Empfang mit dem Senden der eigenen Daten. Dabei wird die Empfangene Zeichenkette ausgewertet (Quellcode 6.8, Zeile 8 ff), die Zustände des Systems aktualisiert und zurückgesendet (Quellcode 6.9Quellcode 6.8). Anschließend erwartet der Server, dass der Client die Verbindung abbaut.

```
01 if ( ready_to_send )
02 {
03     steuerstring_setzen(string);
04     send_data(string,socket_tcp);
05     ready_to_send = __FALSE;
06 }
```

Quellcode 6.9: Senden der Nutzdaten

Dabei ist die Funktion „steuerstring_setzen“ diejenige, die die Zeichenkette aktualisiert. Mit „send_data“ wird diese an den Client übertragen. Der Controller wechselt wieder in den Zustand S_1 und wartet auf eingehende Verbindungsanfragen.

6.4 Steuersoftware

Bei der Steuersoftware handelt es sich um eine in C# geschriebene Windows-Anwendung mit grafischer Oberfläche. Für die Entwicklung wurde die Umgebung „Visual Studio 2012“ der Firma Microsoft verwendet.

Die Software dient der Visualisierung des Systemzustandes des Servers auf einem PC und dessen Steuerung.

6.4.1 Anforderungen

Vor Beginn der Entwicklung wurden einige Anforderungen festgelegt, welche es möglichst einzuhalten und umzusetzen galt:

- Erstellung in C#
- grafische Benutzeroberfläche
- Datenbankanbindung zur Protokollierung der Übertragungen
- Visualisierung der übertragenen Daten, sowohl grafisch als auch in Textform
- Verwendung des vorgegebenen Anwendungsprotokolls[36]
- Verwendung von Ethernet, TCP und IPv4

[36] Vgl. Kapitel 6.2.4

6.4.2 Grafische Benutzeroberfläche

Die grafische Benutzeroberfläche, oder auch Graphical User Interface, kurz GUI, besteht im Wesentlichen aus einem Hauptfenster, einem Debug-Fenster, einem Datenlogger und einem Einstellungsfenster. Des Weiteren gibt es einen Menüpunkt „Hilfe“ unter dem eine Hilfedatei, sowie ein Testmodus zu finden sind.

6.4.2.1 Hauptfenster

Das Hauptfenster beinhaltet das Hauptprogramm und wird nach dem Starten der Anwendung automatisch geöffnet. Hier kann eine Datenübertragung gestartet und die entsprechenden Änderungen am Server vorgenommen werden. Außerdem werden alle übertragenen Informationen visualisiert.

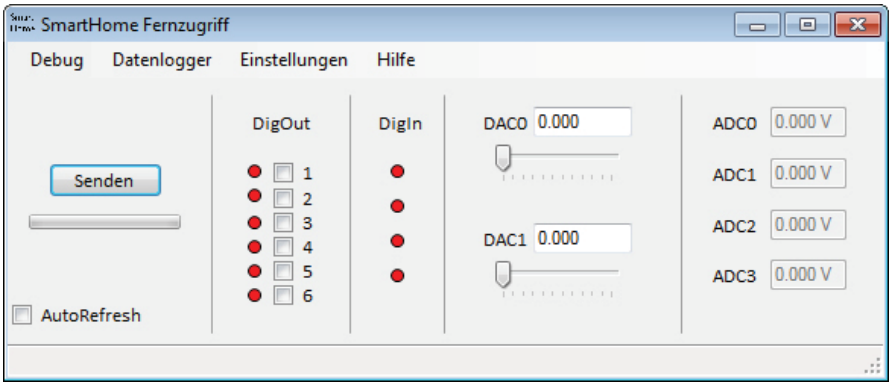


Abbildung 6.14: Hauptfenster der Steuersoftware

In der Menüleiste finden sich Verknüpfungen zum Öffnen der einzelnen Fenster, welche zum Teil den Zugriff auf das Hauptfenster blockieren. Auf der linken Seite befindet sich die Schaltfläche „Senden“, welche eine Übertragung startet, sowie eine dazugehörige Fortschrittsanzeige. Des Weiteren gibt es eine CheckBox[37], welche zyklisch nach 3 Sekunden eine automatische Übertragung auslöst und die Schaltfläche „Senden“ deaktiviert. Der übrige Bereich des Hauptfensters wird durch Objekte vereinnahmt, welche der Visualisierung und Einstellung des Servers dienen. So können die digitalen sowie analogen Ein- und Ausgänge überwacht und gesteuert werden. In der Statusleiste am unteren Rand, werden Informationen zum Status der Übertragung sowie Fehlermeldungen angezeigt.

[37] Bedienelement in grafischen Oberflächen mit mindestens zwei Zuständen, welche z.B. durch einen Haken im Kontrollfeld dargestellt werden können

6.4.2.2 Debug-Fenster

Das Debug-Fenster ermöglicht es, die übertragenen Daten in Textform zu verfolgen, es wird über den Menüpunkt „Debug“ im Hauptfenster geöffnet. Wird eine Übertragung gestartet, so wird hier der Status der Verbindung, sowie die Daten in Sende- und Empfangsrichtung getrennt dargestellt. Dies dient bei der Entwicklung von zu steuernder Hardware oder der Implementierung in ein bestehendes Netz der Fehlersuche und -behebung.

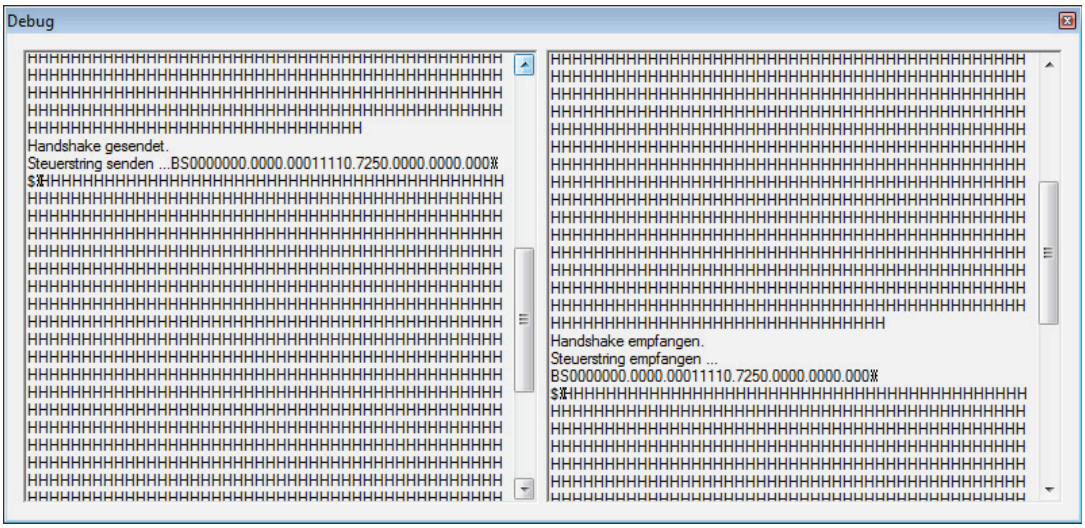
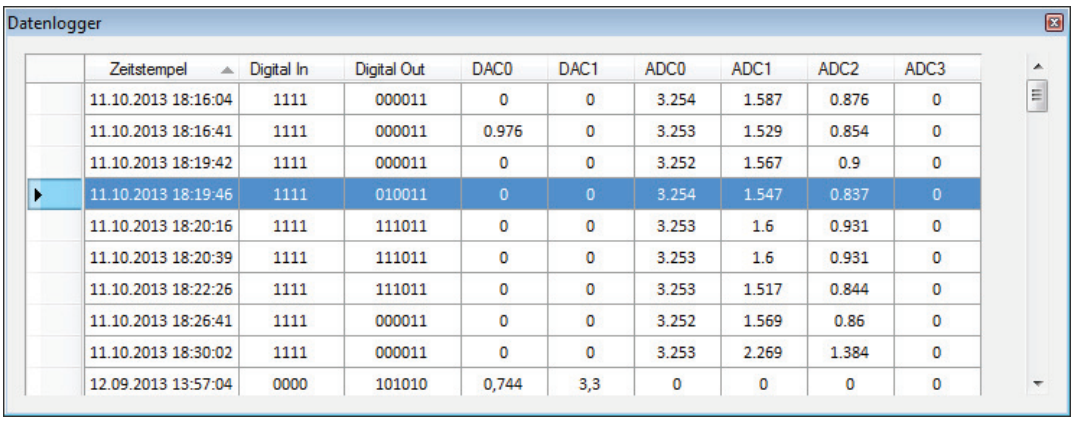


Abbildung 6.15: Debug-Fenster mit übertragenen Daten

6.4.2.3 Datenlogger-Fenster

Das Datenlogger-Fenster wird über den Menüpunkt „Datenlogger“ im Hauptfenster geöffnet und dient der Darstellung der Datensätze mit den protokollierten Übertragungen, welche für jede Übertragung angelegt werden. Dabei werden ein Index, ein Zeitstempel sowie die Übertragenen Daten gespeichert. Die einzelnen Datensätze können innerhalb der Tabelle sortiert, geändert und gelöscht werden. Die Datenbank wird wahlweise im Microsoft Access Format (*.mdb) oder als Textdatei, im CSV-Format[38] gespeichert. Das Löschen und Editieren der Datensätze, sowie das Anlegen des Index wird nur von der Microsoft Access-Datenbank unterstützt.

[38] „Comma-seperated values“ - dient der Speicherung von Datensätzen in Datenbanken, wobei die einzelnen Werte durch Komma getrennt archiviert werden.



Zeitstempel	Digital In	Digital Out	DAC0	DAC1	ADC0	ADC1	ADC2	ADC3
11.10.2013 18:16:04	1111	000011	0	0	3.254	1.587	0.876	0
11.10.2013 18:16:41	1111	000011	0.976	0	3.253	1.529	0.854	0
11.10.2013 18:19:42	1111	000011	0	0	3.252	1.567	0.9	0
11.10.2013 18:19:46	1111	010011	0	0	3.254	1.547	0.837	0
11.10.2013 18:20:16	1111	111011	0	0	3.253	1.6	0.931	0
11.10.2013 18:20:39	1111	111011	0	0	3.253	1.6	0.931	0
11.10.2013 18:22:26	1111	111011	0	0	3.253	1.517	0.844	0
11.10.2013 18:26:41	1111	000011	0	0	3.252	1.569	0.86	0
11.10.2013 18:30:02	1111	000011	0	0	3.253	2.269	1.384	0
12.09.2013 13:57:04	0000	101010	0,744	3,3	0	0	0	0

Abbildung 6.16: Datenlogger der Steuersoftware

6.4.2.4 Einstellungen

Das Einstellungs-Fenster ist in zwei Reiter unterteilt, „Datenbank“ und „Verbindung“. Das Fenster wird beim erstmaligen Starten der Anwendung automatisch aufgerufen. Dieses Verhalten kann durch Setzen der CheckBox „Beim nächsten Start anzeigen“ dauerhaft aktiviert werden. Das Aufrufen des Fensters blockiert den Zugriff auf das Hauptfenster, bis es wieder geschlossen wird.

Der Reiter Verbindung ermöglicht die Konfiguration der Ziel-Sockets, also von IP und Port auf denen der Server lauscht.

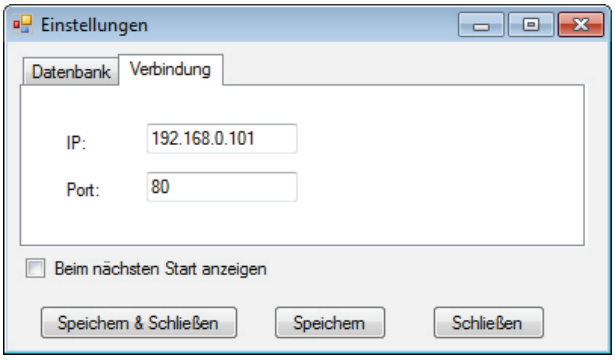


Abbildung 6.17: Einstellungen der Steuersoftware, Reiter Verbindung

Im Reiter „Datenbank“ können Einstellungen für die, vom Datenlogger zu verwendenden, Datenbank vorgenommen werden. Unter anderem können Format und Speicherort angegeben werden.

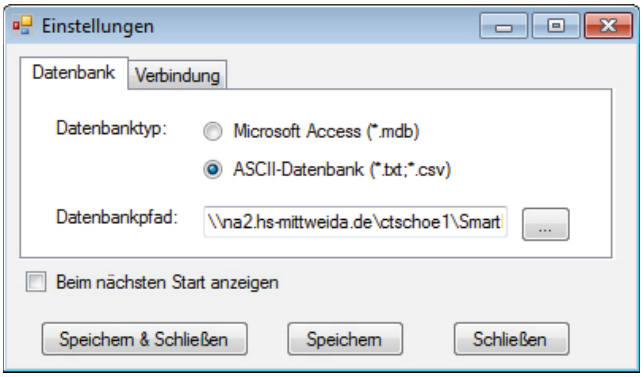


Abbildung 6.18: Einstellungen der Steuersoftware, Reiter Datenbank

6.4.2.5 Menüpunkt „Hilfe“ und Testmodus

Der Menüpunkt „Hilfe“ stellt zum einen eine in HTML verfasste Hilfedatei zur Verfügung um die Grundlegende Funktion und Bedienung der Software zu erläutern. Zum anderen lässt sich über den Unterpunkt „Testmodus“ ein lokaler Server starten.

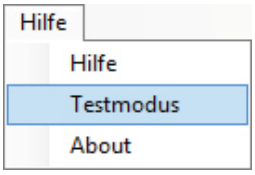


Abbildung 6.19: Menüpunkt "Hilfe"

Der Testmodus ermöglicht mit Hilfe des lokalen Servers das ausführen der Anwendung ohne physisch vorhandene Hardware als Server. Dazu muss zusätzlich die IP „127.0.0.1“, also die Loopback-Adresse als Ziel eingestellt werden und der Port auf dem der Server lauschen soll übernommen werden. Um eine bessere Auswertung des Verkehrs mit Sniffing-Tools[39] wie z.B. „Wireshark“ zu ermöglichen, wurde Standardmäßig der Port 1001, auf dem in der Regel kein oder nur wenig Datenverkehr zu erwarten ist, ausgewählt. Anschließend kann der Server gestartet und eine normale Übertragung simuliert werden. Zusätzlich zeigt die Server-Applikation die Übertragenen Daten in Textform an.

[39] Software welche der Darstellung und Auswertung von Datenvekehr dient

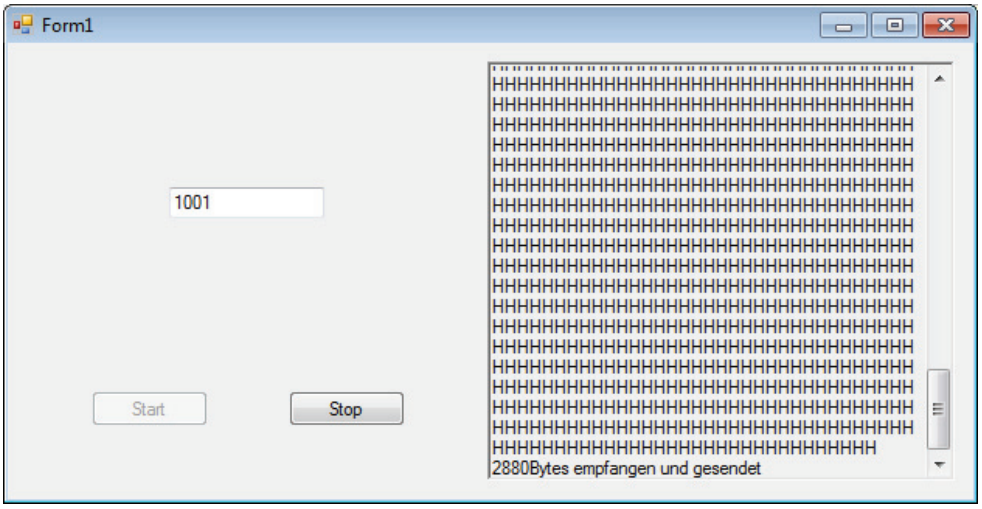


Abbildung 6.20: Testmodus der Steuersoftware mit übertragenen Daten

6.4.3 Ausgewählte Codeauszüge

6.4.3.1 Netzwerkverbindung

Für die Verwaltung von TCP-Verbindungen stellt Microsoft in seinem .NET Framework die Klasse „System.Net.Sockets.TcpClient“ zur Verfügung. Nachdem ein Objekt der Klasse für das IPv4 erstellt wurde, wird versucht eine Verbindung zum Server herzustellen. Ist dies erfolgreich, beginnt die Datenübertragung (Quellcode 6.10, Zeile 1 und Zeile 6). Des Weiteren wird die Dauer festgelegt, nach der der Client eine Zeitüberschreitung meldet. Der Server muss innerhalb dieser Zeitspanne antworten, ansonsten wird die Verbindung vom Client beendet und ein Fehler gemeldet. Da sich die Latenzen bei der Kommunikation im Internet in der Regel im Bereich von einigen Millisekunden abspielt, wurden die Timeouts, jeweils für Sende- und Empfangsrichtung, auf zwei Sekunden festgelegt (Quellcode 6.10, Zeile 2 f). Nach dieser Zeit kann davon ausgegangen werden, dass die Gegenstelle schlicht nicht erreichbar ist.

```
01 TcpClient TCP_Client = new TcpClient(AddressFamily.InterNetwork);
02 TCP_Client.ReceiveTimeout = 2000;
03 TCP_Client.SendTimeout = 2000;
04 try
05 {
06     TCP_Client.Connect(OptionsWindow.host, OptionsWindow.port);
07     progressBar1.Value = 100/6;
08 }
09 catch (Exception exc)
10 {
11     set_toolstripstatuslabel(toolStripStatusLabel, exc.Message);
12     toggle_sent_button();
13 }
```

Quellcode 6.10: TCP-Cient der Steuersoftware – Instanzierung des Clients

Um das unnötige blockieren der grafischen Oberfläche zu verhindern und die Performance der Anwendung zu erhöhen, nutzt das Programm die parallele Abarbeitung einzelner Programmabschnitte in sogenannten Threads. Dabei wird die Abarbeitung auf mehrere Prozessorkerne verteilt, sofern verfügbar bzw. in verschiedenen Zeitschlitzen abgearbeitet falls der Prozessor nur einen Kern besitzt. Nach der Instanzierung eines neuen Thread-Objektes und Starten des Threads (Quellcode 6.11), wartet das Hauptfenster auf bestimmte Signale des Threads, um die grafische Oberfläche zu aktualisieren (Quellcode 6.12, Zeile 3). Der Thread selbst beginnt mit der Übertragung des Handshakes (Quellcode 6.12, Zeile 4).

```
01 if (TCP_Client.Connected)
02 {
03     clientThread = new Thread(new ParameterizedThreadStart(
04                    send_data));
05     clientThread.Start(TCP_Client);
06 }
```

Quellcode 6.11: TCP-Cient der Steuersoftware - Starten des Threads

Im Anschluss wartet der Client auf eine Antwort vom Server. Stimmt diese mit dem gesendeten Handshake überein, werden die zu senden Daten in einen String formatiert und an den Server gesendet (Quellcode 6.12, Zeile 10 ff). Anschließend erwartet der Thread, dass diese aktualisiert vom Server zurückgesendet werden. Ist dies passiert, werden die neuen Daten in die Oberfläche übernommen und der Thread beendet die TCP-Verbindung (Quellcode 6.12, Zeile 31 ff).

```
01 try
02 {
03     handshake_senden.WaitOne(TimeSpan.FromMilliseconds(100));
04     NetStream.Write(byteBuffer, 0, byteBuffer.Length);
05     //Thread.Sleep(0);
06 
07     int totalBytesReceived = 0;
08     int bytesReceived = 0;
09     handshake_empfangen.WaitOne(TimeSpan.FromMilliseconds(100));
10     while (totalBytesReceived < byteBuffer.Length &&
11            bytesReceived == NetStream.Read(byteBuffer,
12            totalBytesReceived, byteBuffer.Length -
13            totalBytesReceived))
14     {
15         totalBytesReceived += bytesReceived;
16     }
17     //Thread.Sleep(0);
18 
19     if (byteBuffer == handshake_sent)
20     {
21         steuerstring_senden.WaitOne(TimeSpan.FromMilliseconds(
22                                     100));
23         byteBuffer = steuerstring_byte;
24         NetStream.Write(byteBuffer, 0, byteBuffer.Length);
25         //Thread.Sleep(0);
26 
27         totalBytesReceived = 0;
28         bytesReceived = 0;
29         steuerstring_empfangen.WaitOne(TimeSpan.FromMilliseconds(
30                                        100));
31         while (totalBytesReceived < byteBuffer.Length &&
32                bytesReceived == NetStream.Read(byteBuffer,
33                totalBytesReceived, byteBuffer.Length -
34                totalBytesReceived))
35         {
36             totalBytesReceived += bytesReceived;
37         }
38         steuerstring_byte = byteBuffer;
39         steuerstring_auswerten_event.Set();
40     }
41 
42 }
```

Quellcode 6.12: TCP-Cient der Steuersoftware - Client-Thread

6.4.3.2 Datenbankanbindung

Für die Anbindung der Datenbanken an das Programm, wurde eine zusätzliche Klasse implementiert um den Quellcode zum einen übersichtlich und zum anderen instanzierbar zu machen. Für den Zugriff auf die Datenbanken werden Treiber von Microsoft verwendet. Diese ermöglichen zum einen den einfachen Zugriff auf die Datenbanken zum anderen sind sie bereits in neueren Windowsbetriebsystemen enthalten. Bei den Treibern handelt es sich um „Microsoft Jet OleDB 4.0“ für Microsoft Access Datenbanken und dem „Microsoft Text Driver“, auch „Microsoft ODBC“ genannt, für die textbasierte Datenbank im CSV-

Format. Der Zugriff auf die Treiber wird mit den Klassen OdbcConnection, OdbcDataAdapter, OleDbConnection und OleDbDataAdapter realisiert (Quellcode 6.13). Für größere Datenbestände wird die Verwendung des Microsoft ODBC Treibers empfohlen, da der Zugriff auf größere Mengen von Datensätzen wesentlich schneller als bei der Jet-Engine erfolgt.

```
01 Connection_mdb = new OleDbConnection("Provider=Microsoft.Jet.
02                  OLEDB.4.0; Data Source =" + path + ";Jet OLEDB:
03                  Engine Type=5");
04 DataAdapter_mdb = new OleDbDataAdapter("SELECT * FROM
05                   Datenlogger order by ID", Connection_mdb);
06 Connection_ascii = new OdbcConnection(@"Driver={Microsoft Text
07                    Driver (*.txt; *.csv)};Dbq=" + path.Replace(
08                    filename,"").Replace("\\\\",@"\\") + ";
09                    Extensions=asc,csv,tab,txt;Persist Security
10                    Info=False");
11 DataAdapter_ascii = new OdbcDataAdapter("SELECT * FROM "+
12                     filename+" ORDER BY zeitstempel",
13                     Connection_ascii);
```

Quellcode 6.13: Datenbankanbindung der Steuersoftware

6.4.3.3 Konfigurationsdatei

Beim erstmaligen Starten der Anwendung wird eine Konfigurationsdatei angelegt, in der die im Fenster „Einstellungen“ festgelegten Werte für den nächsten Aufruf gespeichert werden. Die Konfiguration der Anwendung wird in einer XML-Datei im dazugehörigen Format gespeichert (Quellcode 6.15), der Zugriff erfolgt mithilfe der Klasse XmlTextWriter, welche das Schreiben im XML-Format ermöglicht, sowie der Funktion ReadXml der Klasse DataSet, welche das Einlesen von Daten aus einer XML-Datei in ein C#-Dataset ermöglicht (Quellcode 6.14).

```
01 XmlTextWriter createXML = new XmlTextWriter("options.xml", null);
02
03 DataSet dataset_options = new DataSet();
04 dataset_options.ReadXml("options.xml");
```

Quellcode 6.14: Konfigurationsdatei der Steuersoftware

```
01 <?xml version="1.0" standalone="yes"?>
02 <Options>
03   <Verbindung>
04     <IP>127.0.0.1</IP>
05     <Port>1001</Port>
06   </Verbindung>
07   <Datenbank>
08     <Path>\\na2.hs-mittweida.de
09     \ctschoe1\SmartHome\GUI\SmartHomeGUI\SmartHomeGUI_Async_ASC
10     IIDB\bin\Debug\asciidb.txt</Path>
11     <Typ>asciidb</Typ>
12   </Datenbank>
13   <Sonstiges>
14     <ShowToolTip>false</ShowToolTip>
15   </Sonstiges>
16 </Options>
```

Quellcode 6.15: Konfigurationsdatei "options.xml" im XML-Format

6.4.3.4 Testmodus

Der Testmodus wurde aus rein praktischen Gründen in eine zusätzliche Anwendung ausgelagert, so kann der darin enthaltene Server auch unabhängig der Client-Anwendung auf anderen Windows-PC gestartet werden.

Die Applikation ermöglicht das Starten eines Servers zur Simulation der Hardware, durch das Betätigen der Schaltfläche „Start“. Diese Aktion startet einen Thread, welcher einen TCP-Listener auf dem Angegebenen Port ausführt. Verbindet sich ein Client mit dem entsprechenden Handshake, werden die empfangenen Daten einfach an den Client zurückgesendet. Wird die Schaltfläche „Stop“ ausgeführt, wird der Thread beendet und kann auf einem anderen oder dem gleichen Port erneut gestartet werden.

```
01 this.listenThread = new Thread(new ThreadStart(ListenForClients));
02 this.listenThread.Start();
03
04 private void ListenForClients()
05 {
06     this.tcpListener.Start();
07     while (true && run_server)
08     {
09         while (!(this.tcpListener.Pending()))
10         {
11             if (!tcpListener_running)
12             {
13                 return;
14             }
15             Thread.Sleep(100);
16         }
17
18         if (run_server)
19         {
20             TcpClient client = this.tcpListener.AcceptTcpClient()
21                                ;
22             Thread clientThread = new Thread(new
23                                   ParameterizedThreadStart(
24                                   handle_tcp_client));
25             clientThread.Start(client);
26         }
27     }
28 }
```

Quellcode 6.16: Testmodus der Steuersoftware

Um den Thread des Listeners nicht unnötig zu blockieren, wird die Abarbeitung der Verbindung in einem weiteren Thread, “clientThread”, behandelt. Der Thread liest den Datenstrom aus, schreibt den Inhalt in das Fenster der Benutzeroberfläche und sendet ihn ohne Änderungen an den Client zurück (Quellcode 6.17).

```
01 private void handle_tcp_client(object client)
02 {
03     if (run_server)
04     {
05         TcpClient tcpClient = (TcpClient)client;
06         NetworkStream clientStream = tcpClient.GetStream();
07
08         byte[] bytebuffer = new byte[4096];
09         int bytesRcvd;
10
11         int totalbytesEchoed = 0;
12         while ((bytesRcvd = clientStream.Read(bytebuffer, 0,
13                bytebuffer.Length)) > 0)
14         {
15             set_text(richTextBox1, "\n");
16             set_text(richTextBox1, ASCIIEncoding.ASCII.GetString(
17                       bytebuffer));
18             clientStream.Write(bytebuffer, 0, bytesRcvd);
19             totalbytesEchoed += bytesRcvd;
20         }
21
22         set_text(richTextBox1, "\n" + totalbytesEchoed.ToString()
23                  + "Bytes empfangen und gesendet");
24
25         clientStream.Close();
26         tcpClient.Close();
27     }
28 }
```

Quellcode 6.17: TCP-Server des Testmodus

6.5 Hardware

Um einen Teilnehmer in dieser Smart Home Applikation zu realisieren, wurde ein Aufbau auf einer Lochrasterplatine mit diskreten Bauteilen verwendet. Zum einen bietet die Auswahl eine einfache und sehr flexible Möglichkeit die Schaltung aufzubauen, zum anderen kann sie während der Entwicklung schnell geändert und erweitert werden. Des Weiteren standen alle dafür benötigten Mittel in der Fachgruppe zur Verfügung. Wie **Anlage,Teil 1** entnommen werden kann, wurde neben einem Steckplatz für das Modul Chip1768 folgende Baugruppen entworfen und verbaut:

- Spannungsversorgung
- 230V Relais zum Schalten von Verbrauchern
- Potentiometer für analoge Eingangsspannung und Regelung des Kontrastes des LCD
- LC-Display

- Ethernetschnitstelle.

6.5.1 Spannungsversorgung

Für die Spannungsversorgung der Schaltung wird eine Quelle mit einer Spannung zwischen 8V und 12V bei einem maximalen Strom von 1A benötigt. Die angelegte Spannung wird zum einen durch einen Festspannungsregler auf 5V geregelt um das Modul Chip1768 zu versorgen, zum anderen direkt für die Energieversorgung der verbauten Relais verwendet. Bei dem Festspannungsregler handelt es sich um einen LM317T, welcher den Chip1768 mit 5V und maximal 1.5A versorgt. Die Relais werden direkt über Transistorschalter, welche durch den LPC1768 gesteuert werden, direkt an die Spannungsversorgung angeschlossen.

Die Dimensionierung der Schaltung erfolgte entsprechend des Datenblattes. Demnach ergibt sich die Ausgangsspannung wie folgt:

$$V_O = 1.25V\left(1 + \frac{R_2}{R_1}\right) + 50\mu A \cdot R_2$$

Vernachlässigt man den zweiten Teil des Terms und löst den ersten nach dem Verhältnis R_2/R_1 auf, erhält man 3. Laut Datenblatt wird für R_1 ein Wert von 240Ω empfohlen, da in den gängigen E-Reihen allerdings kein Wert mit 720Ω vorkommt, wurde R_1 auf 180Ω festgelegt, sodass sich für R_2 der Wert 540Ω ergibt. Setzt man die ausgewählten Werte ein ergibt sich für die Ausgangsspannung V_O nun:

$$V_O = 1.25V\left(1 + \frac{540\Omega}{180\Omega}\right) + 50\mu A \cdot 540\Omega = \underline{\underline{5.027V}}$$

Eine Abweichung von 27mV ist in Anbetracht des notwendigen Mehraufwandes für eine geringere Abweichung akzeptabel und beeinträchtigt die Funktion der Schaltung nicht. Die bestimmten Werte für R_1 und R_2 entsprechen dem Widerstand R_3 und der Summe aus R_1 und R_2 in **Abbildung 6.21.**

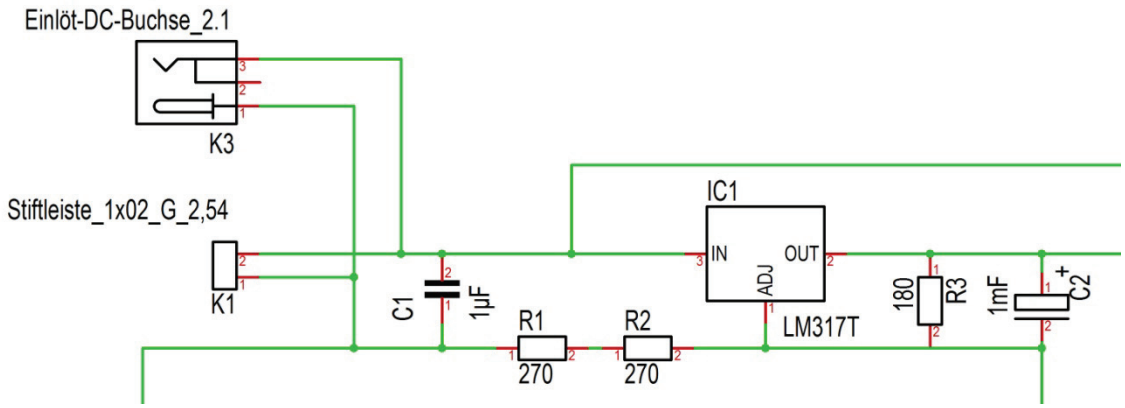


Abbildung 6.21: Spannungsversorgung der Schaltung

6.5.2 Relais

Mit Hilfe der verbauten Relais, wird das Schalten von 230V-Verbrauchern über den LPC1768 ermöglicht. Die digitalen Ausgänge des Controllers schalten jeweils einen Transistor, welcher wiederum die Relais mit einer Schaltspannung von 230V schalten. Verwendet wurden die Relais „OMRON G2RL-2“ und „Song Chuan 899-1C-F-C-E“ und der Transistor BC547B. Wie **Anlage, Teil 1** zu entnehmen ist, werden die Transistoren über Vorwiderstände mit jeweils 1,2kOhm durch die Portpins P0.15 und P0.17 angesteuert. Somit fließt ein Basisstrom von 2.75mA, welcher durch den Transistor auf ca. 55mA verstärket wird. Das Relais benötigt bei 12V einen Schaltstrom von 33.3mA, der Strom der durch die Schaltspule fließt ist also ausreichend groß.

Der Basiswiderstand ergibt sich bei einer Stromverstärkung von 20 und einer Basis-Emitter-Spannung am Portpin des LPC1768 von 3.3V wie folgt:

$$R = \frac{3.3V - 0.7V}{\frac{33.3mA}{20}} = \underline{\underline{1562\Omega}}$$

Um die Relais auch bei etwas geringeren Spannungen sicher durchschalten zu können, wurde ein etwas geringerer Vorwiderstand mit R=1200Ω gewählt, sodass ein Schaltstrom von 55mA zur Verfügung steht.

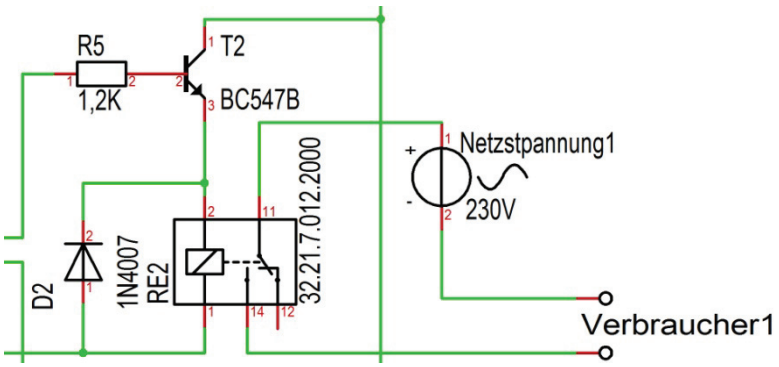


Abbildung 6.22: Relaisschaltung

6.5.3 Potentiometerschaltung

Um an den analogen Eingängen des LPC1768 Spannungen messen zu können, wurde eine Potentiometerschaltung verwendet. Diese steuert zum einen den Kontrast des LC-Displays, zum anderen wird sie gleichzeitig mit Hilfe des ADC gemessen und auf dem Display angezeigt, sowie dem Client zur Verfügung gestellt. Da die Kontrastregelung 5V benötigt, der ADC am Eingang jedoch nur Spannungen bis 3.3V wandelt, wurde ein Spannungsteiler vorgeschaltet.

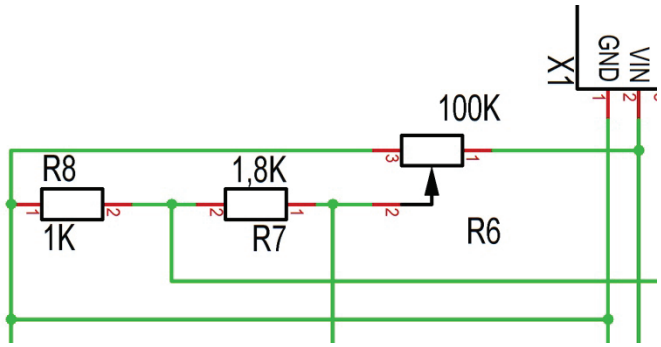


Abbildung 6.23: Potentiometerschaltung für ADC-Eingang und LCD-Kontrast

6.5.4 LC-Display

Wie in 6.3.3 bereits erwähnt, wurde ein 4x20 LC-Display verbaut, um einige Inhalte des Systems darstellen zu können. Diesem Abschnitt können sowohl die Pinbelegung als auch einige Kenndaten entnommen werden. Weitere Informationen zur Schaltung können **Anlagen, Teil 1** entnommen werden. Dem Schaltplan in **Anlagen, Teil 1** zeigt wie das Display-Modul in die Schaltung integriert wurde. Neben den Daten- und Steuerleitungen,

sowie der Spannungsversorgung für den Controller und die Beleuchtung, wurde die Kontrastregelung angeschlossen. Diese wurde über die Potentiometerschaltung (vgl. 6.5.3) realisiert.

6.5.5 Ethernetschnittstelle

Um die Kommunikation mit dem Controller über Ethernet zu ermöglichen, musste eine entsprechende Schnittstelle verbaut werden. Die dafür benötigten Controller sind bereits im LPC1768, bzw. auf dem Chip1768 vorhanden, lediglich ein Transceiver musste hinzugefügt werden. Verwendet wurde der WE7499011001A, welcher entsprechend des Schaltplans in **Anlagen, Teil1** angeschlossen wurde. Der Transceiver ermöglicht die Kommunikation nach Fast Ethernet/100BASE-TX, also LAN mit Übertragungsraten bis zu 100Mbps.

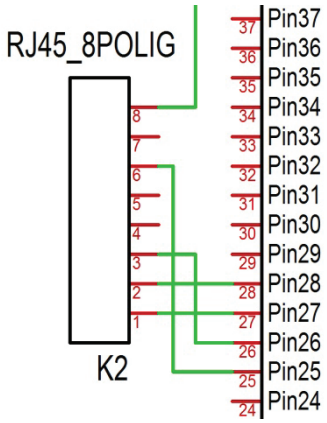


Abbildung 6.24: Verbindung der Ethernetbuchse mit dem LPC1768

7 Funktionstest

Um die Stabilität der Hardwareschaltung und entwickelten Software zu testen, wurde ein Funktionstest über sieben Tage hinweg durchgeführt, bei dem das System ohne Unterbrechung lief und im Abstand von 3 Sekunden Übertragungen durchgeführt wurden.

Während dieses Zeitraums wurden insgesamt ca. 200000 Übertragungen durchgeführt und genau so viele Datensätze übertragen und gespeichert. Während dieses Tests traten beim bestehenden System keinerlei Fehler auf, sowohl Firm- als auch Steuersoftware, haben sich also als äußerst robust und stabil erwiesen.

Neben dem Test im lokalen Netz, wurde die Hardware auch mit Hilfe eines Routers für den Zugriff von außerhalb erreichbar gemacht, sodass die Bedienung der Applikation über Internet ermöglicht wurde. Auch dies verlief ohne Probleme, sodass das Grundprinzip der Firmware als durchaus praktikabel angesehen werden kann.

Die für Windows erstellte Steuersoftware, wurde ebenfalls ausgiebig getestet. Es konnten keine Fehlfunktionen festgestellt werden.

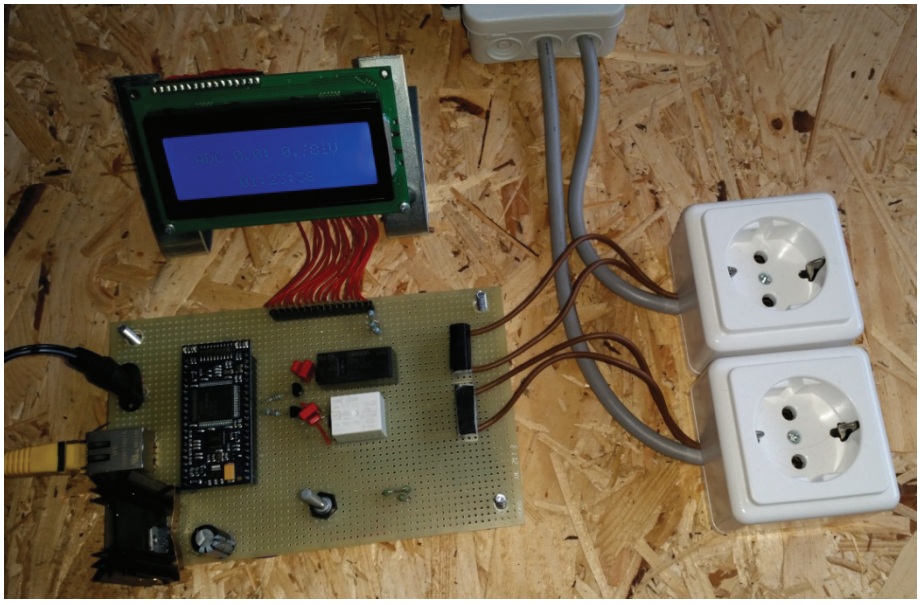


Abbildung 7.1: Versuchsaufbau der Smart Home Applikation

8 Zusammenfassung und Ausblick

Das Ziel dieser Arbeit war es, eine funktionsfähige Applikation für eine Smart Home Anwendung auf Basis des Controllers LPC1768 zu konzipieren und zu realisieren. Dabei sollte die Kommunikation mit Hilfe von TCP/IP und einem von der Professur Kommunikationstechnik der Fakultät Elektro- und Informationstechnik der HSMW entwickelten Anwendungsprotokoll erfolgen.

Während des Projektes, konnte eine funktionierende Smart Home Applikation entwickelt und in Betrieb genommen, sowie getestet werden Die Applikation besteht aus einer Hardwareanwendung auf Basis des LPC1768 mit einer in C geschriebenen Firmware und einer Steuersoftware für Windows welche in C# erstellt wurde. Zum einen wurden alle Vorgaben für das Projekt erfüllt, zum anderen konnte gezeigt werden, dass es möglich ist eine funktionierende Smart Home Anwendung mit auf dem Markt verfügbaren Controllern in bestehende TCP/IP-Infrastrukturen zu integrieren.

In einem nächsten Entwicklungsschritt können allerdings noch einige Verbesserungen und Erweiterungen vorgenommen werden:

- Erstellung eines Konzeptes für die zukünftige Strukturierung des Systems und der zu implementierenden Funktionen
- Erweiterung der Steuersoftware für die Verwaltung mehrerer Teilnehmer
- Spezifizierung spezieller Typen von Teilnehmern, entsprechend der Anwendung z.B. zum Schalten von Steckdosen, Messen von Temperaturen, Stellen von Thermostaten
- Entwicklung einer miniaturisierten, modularen Hardwarebasis sowie eines Energieversorgungskonzeptes
- Erweiterung der Hardware um weitere Kommunikationsmöglichkeiten z.B. W-Lan
- Implementierung verschiedener Sicherheitsfunktionen wie authentifizierten Zugriffsverfahren und Verschlüsselter Datenübertragung
- ggf. Entwicklung eines interaktiven Webinterfaces

Literatur

[1] VDE VERBAND DER ELEKTROTECHNIK e.V., „Die deutsche Normungs-Roadmap Smart Home + Building," 2013.

[2] A. Badach und E. Hoffmann, Technik der IP-Netze, Carl Hanser Verlag GmbH & Co. KG, 2007.

[3] K. Obermann und M. Horneffer, Datennetztechnologien für Next Generation Networks, Vieweg+Teubner, 2009.

[4] J. Scherff, Grundkurs Computernetze, Vieweg Verlag, 2006.

[5] L. Xiamen Ocular Optics Co ., „Xiamen Ocular Optics Co .,Ltd.," 12 07 2011. [Online]. Available: http://www.xmocular.com/Upload/Character/GDM2004DNS_W_BBS0A-19064878889.pdf.

[6] ZVEI - Zentralverband Elektrotechnik- und Elektronikindustrie e.V., „Grundlagenwissen zum KNX Standard".

[7] The Z-Wave Alliance, "The Z-Wave Alliance," Z-Wave Alliance, 22 02 2015. [Online]. Available: http://z-wavealliance.org/.

[8] Deutsche Telekom AG, „QIVICON," Deutsche Telekom AG, 22 02 2015. [Online]. Available: https://www.qivicon.com/de/.

[9] ELMICRO Computer GmbH & Co. KG, „Elektronikladen Microcomputer, Benutzerhandbuch Chip1768," 23 02 2015. [Online]. Available: http://elmicro.com/files/elmicro/chip1768v121_manual_de.pdf.

[10] NXP B.V., „NXP B.V., UM10360 LPX1768 User manual," 24 02 2015. [Online]. Available: http://www.nxp.com/documents/user_manual/UM10360.pdf.

[11] ARM Ltd and ARM Germany GmbH, „KEIL: RL-ARM User's Guide," KEIL by ARM Ltd and ARM Germany GmbH, 24 02 2015. [Online]. Available: http://www.keil.com/support/man/docs/rlarm/.

[12] Institute of Electrical and Electronics Engineers, IEEE, „IEEE 802.11-2012," 24 02 2015. [Online]. Available: http://standards.ieee.org/getieee802/download/802.11-2012.pdf.

[13] Institute of Electrical and Electronics Engineers, IEEE, „IEEE 802.3," 24 02 2015. [Online]. Available: http://standards.ieee.org/about/get/802/802.3.html.

[14] International Telecommunication Union, ITU, „ITU Recommendation X.200 - Information technology - Open Systems Interconnection - Basic Reference Model: The basic model," 24 02 2015. [Online]. Available: http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.200-199407-I!!PDF-E&type=items.

[15] International Organization for Standardization, International Electrotechnical Commission (ISO/IEC), „ISO/IEC 7498-1:1994 - Information technology -- Open Systems Interconnection -- Basic Reference Model: The Basic Model," 24 02 2015. [Online]. Available: http://www.iso.org/iso/catalogue_detail.htm?csnumber=20269.

[16] Internet Engineering Task Force, IETF, „RFC791 - Internet Engineering Task Force," [Online]. Available: http://tools.ietf.org/html/rfc791.

[17] Internet Engineering Task Force, IETF, „RFC793 - Internet Engineering Task Force," 24 02 2015. [Online]. Available: http://tools.ietf.org/html/rfc793.

[18] Internet Engineering Task Force, IETF, „RFC1661 - Internet Engineering Task Force," 24 02 2015. [Online]. Available: http://tools.ietf.org/html/rfc1661.

[19] Internet Engineering Task Force, „RFC2516 - Internet Engineering Task Force," 24 02 2015. [Online]. Available: https://tools.ietf.org/html/rfc2516.

[20] WÜRTH Elektronik, „Datenblatt WE 7499011001A," 25 02 2015. [Online]. Available:

http://katalog.we-online.de/pbs/datasheet/7499011001A.pdf.

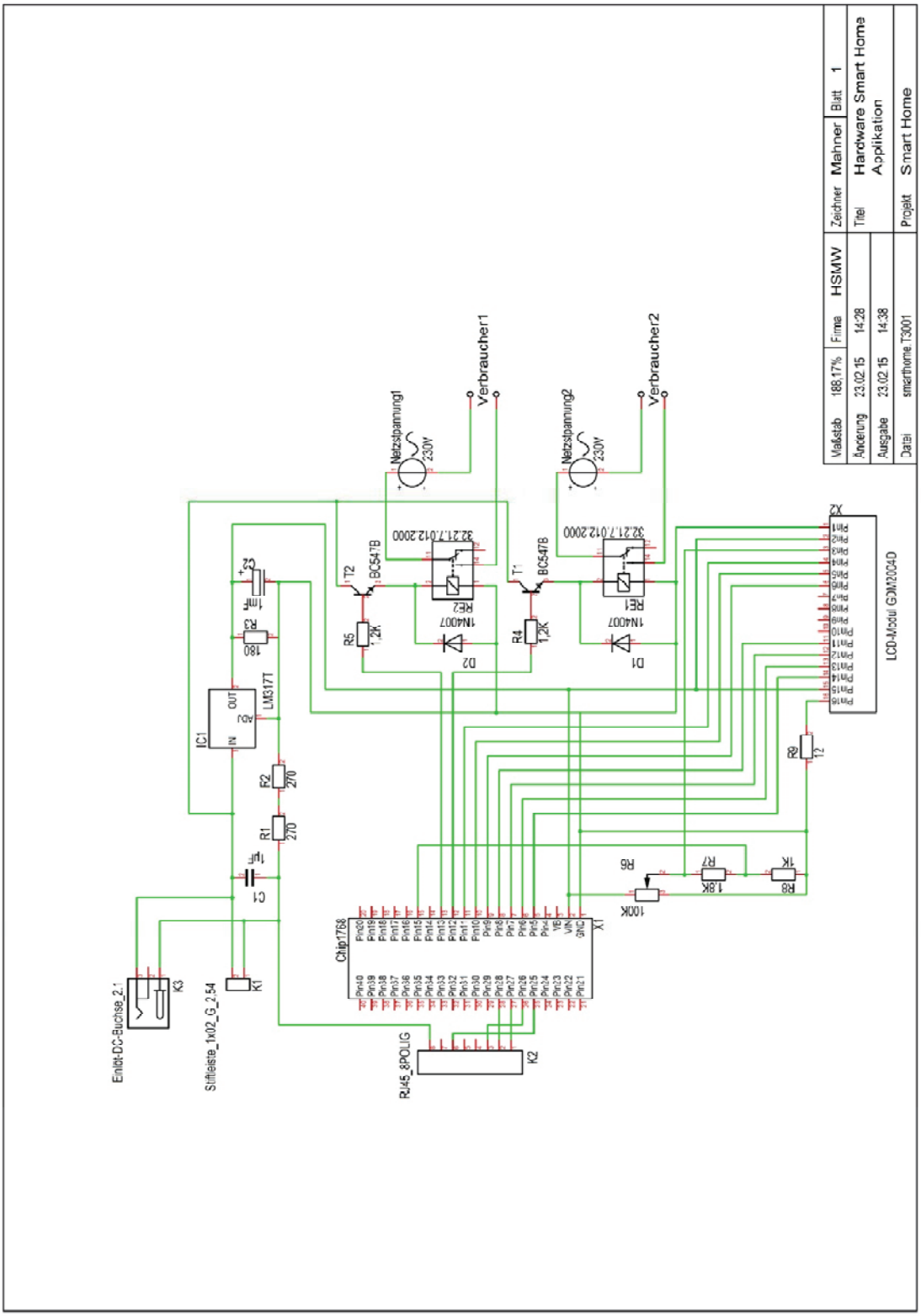
[21] Omron Corporation, „Datenblatt Relais G2RL,“ 25 02 2015. [Online]. Available: http://www.omron.com/ecb/products/pdf/en-g2rl.pdf.

[22] Fairchild Semiconductor Corporation, „Datenblatt Transistor BC547B,“ 25 02 2015. [Online]. Available: https://www.fairchildsemi.com/datasheets/bc/bc547.pdf.

Anlagen

Anlagen, Teil 1

Schaltplan des Hardwareaufbaus



Anlagen, Teil 2

CD mit Programmcode

- Prpjektmappe für Steuersoftware in C#, erstellt mit Visual Studio 2012 und .NET Framework 4

- µVision4-Projekt für Firmware in C, erstellt mit KEIL µVision4

- Beispielprogramme 1 – 5, µVision4-Projekte in C und Projektmappen für Visual Studio 2012 mit .NET Framework 4 in C#

- TARGET3001-Projekt für Schaltplan, erstellt mit TARGET3001 v16

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt.

Mittweida, den 27.02.2015

Christian Mahner