



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

David Bauer

Smart Home Automation mit
Einplatinencomputern

David Bauer
Smart Home Automation mit Einplatinencomputern

Masterthesis eingereicht im Rahmen der Masterprüfung
im Studiengang Automatisierung
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. -Ing. Ulfert Meiners
Zweitgutachter : Prof. Dr. -Ing. Florian Wenck

Abgegeben am 27. November 2015

David Bauer

Thema der Masterthesis

Smart Home Automation mit Einplatinencomputern

Stichworte

Smart Home, Hausautomatisierung, Hardwareentwicklung, Softwareentwicklung, Raspberry Pi, Arduino, I²C, drahtlose Kommunikation, Z-Wave

Kurzzusammenfassung

Diese Arbeit befasst sich mit der Realisierung eines kostengünstigen Smart Home Systems auf Basis von Raspberry Pi und Arduino zur webbasierten Steuerung eines Privathaushaltes. Durch drahtlose Funktechnologie soll das System sowohl für Neuinstallationen als auch für bestehende Elektroinstallationen einsetzbar sein.

David Bauer

Title of the paper

Smart Home Automation with single-board computers

Keywords

Smart Home, Home automation, Hardware development, Software development, Raspberry Pi, Arduino, I²C, Wireless communication, Z-Wave

Abstract

This work deals with the realisation of a reasonable Smart Home system based on Raspberry Pi and Arduino for the web-based control of a private household. By wireless radio technology, the system should be applicable to new installations as well as to existing electric installations.

Vorwort

Danksagung

An dieser Stelle möchte ich mich zunächst bei Prof. Dr.-Ing Ulfert Meiners für das interessante Thema dieser Masterthesis bedanken. Für seine fachliche und kompetente Betreuung und vor allem seine Begeisterung für dieses Thema bedanke ich mich ebenfalls.

Ein Weiterer ganz besonderer Dank gilt meiner lieben Familie, die mich während meines Studiums finanziell und moralisch unterstützt hat. Sie hat es mir ermöglicht mein Studium erfolgreich zu absolvieren und den Weg in eine aussichtsvolle berufliche Zukunft geebnet.

Des Weiteren bedanke ich mich bei meiner Schwester Johanna für ihr großes Arrangement im Korrekturlesen.

Zu guter Letzt möchte ich mich bei meiner Besten, Babette Demmert, die mir stets mit Rat und Tat zur Seite stand, für ihre tatkräftige Unterstützung während meines Studiums an der Hochschule für Angewandte Wissenschaften Hamburg herzlichst bedanken.

Inhaltsverzeichnis

Inhaltsverzeichnis	v
Tabellenverzeichnis	viii
Abbildungsverzeichnis	ix
1 Einführung	1
1.1 Motivation	1
1.2 Zielsetzung dieser Arbeit	2
2 Technische Grundlagen	3
2.1 Raspberry Pi	3
2.2 Arduino	6
2.3 I ² C-Bus	9
2.4 SPI-Bus	12
2.5 Z-Wave	14
3 Marktanalyse und Konzeptentwurf	16
3.1 Marktanalyse	16
3.1.1 Funkübertragung	16
3.1.2 Smart Home - Software	19
3.2 Smart Home Konzept	26
4 Entwicklung und Umsetzung	33
4.1 Smart Home Zentrale	33
4.1.1 Hardware	33
4.1.1.1 Spannungsversorgung	33
4.1.1.2 Tolerante Eingänge	34
4.1.1.3 Ausgänge	35
4.1.1.4 Optionaler I ² C-Anschluss	35
4.1.1.5 Arduino Funk-Transceiver	36
4.1.2 Software	37
4.1.2.1 CoDeSys einrichten	37
4.1.2.2 GPIO - Anpassung	40

4.1.2.3	I ² C-Treiber - Arduino Funk	42
4.1.2.4	Sketch - Arduino Funk	46
4.1.2.5	CoDeSys - Softwarestruktur	47
4.1.2.6	CoDeSys - Webserver	59
4.1.2.7	CoDeSys - Visualisierung	59
4.1.3	Ergebnis	65
4.2	Funk IO-Modul	66
4.2.1	Hardware	66
4.2.1.1	Spannungsversorgung	66
4.2.1.2	IO-Erweiterung	66
4.2.1.3	Ein-/Ausgänge	67
4.2.2	Software	68
4.2.3	Ergebnis	72
4.3	Funk Sensor-Modul	73
4.3.1	Hardware	73
4.3.1.1	Spannungsversorgung	73
4.3.1.2	Adressierung	74
4.3.1.3	Sensoren	74
4.3.2	Software	74
4.3.2.1	TTL-Treiber einrichten	74
4.3.2.2	Arduino Sketch	75
4.3.2.3	Schlafmodi	77
4.4	Z-Wave Controller	80
4.4.1	Hardware	80
4.4.2	Software	80
4.4.2.1	Z-Wave Controller einrichten	80
4.4.2.2	Z-Wave Netzwerk konfigurieren	81
4.4.2.3	JSON - CommandClass	82
4.4.2.4	CoDeSys - Ansteuerung	85
5	Résumé	88
5.1	Bewertung	88
5.2	Aussichten	89
A	Anhang	90
A.1	Smart Home Zentrale	90
A.1.1	Geräteliste	90
A.1.2	Schaltplan	91
A.1.3	Platinenlayout	92
A.2	Funk IO-Modul	93
A.2.1	Geräteliste	93

A.2.2	Schaltplan	94
A.2.3	Platinenlayout	95
A.3	Funk Sensor-Modul	96
A.3.1	Geräteliste	96
A.3.2	Schaltplan	97
A.3.3	Platinenlayout	98
A.4	Bibliotheken	99
A.4.1	Arduino Bibliotheken	99
A.4.2	Oscat Network Bibliothek	100
Abkürzungsverzeichnis		101
Literaturverzeichnis		103

Tabellenverzeichnis

2.1	Raspberry Pi Modelle	5
2.2	I ² C-Bus Modi	10
2.3	SPI - Modi	14
3.1	Funksysteme	20
3.2	Softwarevergleich	25
3.3	Hardwarespezifikation Zentrale	29
3.4	Hardwarespezifikation Funk IO-Modul	30
3.5	Hardwarespezifikation Funk Sensor-Modul	31
4.1	I ² C Ein-/Ausgangspuffer	44
4.2	Funk – Datenstruktur	70
4.3	Arduino – Sleepmodi	78
A.1	Geräteliste – Smart Home Zentrale	90
A.2	Geräteliste – Funk IO-Modul	93
A.3	Geräteliste – Sensor-Modul	96

Abbildungsverzeichnis

2.1	Raspberry Pi Model B+	4
2.2	Raspberry Pi GPIO-Schnittstelle	6
2.3	Arduino UNO	7
2.4	Arduino IDE	9
2.5	I ² C-Bus Master-Slave-Konzept	11
2.6	I ² C Übertragungsverlauf	11
2.7	SPI-Bus mit sternförmiger Infrastruktur	13
2.8	SPI – Abtastung	14
2.9	Z-Wave Modul RaZBerry	15
3.1	openHAB – Visuelle iOS Darstellung	21
3.2	FHEM – Visualisierung	22
3.3	CoDeSys – Entwicklungssystem	23
3.4	CoDeSys – Visualisierungsbeispiel	24
3.5	Smart Home Konzept	27
3.6	Konzept – Smart Home Zentrale	29
3.7	Konzept – Funk IO-Modul	30
3.8	Konzept – Funk Sensor-Modul	31
3.9	Z-Wave Geräte	32
4.1	Spannungsregler-Modul	33
4.2	PSpice Model mit Zener-Diode ZPD3.3V	34
4.3	PSpice Simulation – Pegelwandlung 3.3V	34
4.4	Darlington-Array-Schaltung	35
4.5	Schaltung – Arduino mit RFM12B Funkmodul	37
4.6	Telnet Client – PuTTY	38
4.7	Raspberry – Configuration Tool	39
4.8	CoDeSys – Raspberry Pi Update	39
4.9	CoDeSys – Lizenzierung	40
4.10	Code-Ausschnitt GPIO Treiber	41
4.11	GPIO Konfiguration	42
4.12	Code-Ausschnitt Arduino Funk Treiber	43
4.13	Projektinformation - Arduino Funk Bibliothek	43

4.14 I ² C-Slave FUNK (RFM12B)	44
4.15 Arduino Funk – Bibliotheksmethoden	45
4.16 Code-Ausschnitt – Initialisierung	46
4.17 Struktogramm – Arduino Funk Controller	47
4.18 Klassen - Interfaces	48
4.19 Klasse KL_LICHT	48
4.20 Klasse KL_ROLLO	49
4.21 Aktivitätsdiagramm – Klasse KL_ROLLO	50
4.22 Klasse KL_RAUMKONFIG	51
4.23 Klasse KL_RAUM_TYP1	52
4.24 Klasse KL_RAUM_TYP2	53
4.25 Klasse KL_RAUM_TYP3	53
4.26 Klasse KL_MODEKONFIG	54
4.27 Klasse KL_SIMULATION	55
4.28 Klasse KL_WOHNUNG	56
4.29 Ablaufdiagramm – Klasse KL_WOHNUNG	57
4.30 UML-Diagramm der Smart Home Umgebung	58
4.31 CoDeSys - Webserver	59
4.32 Visu – Hauptseite Smart Home	60
4.33 Visu – Seite Konfiguration	61
4.34 Dialog – Zeiteingabe	62
4.35 Visu – Modi Konfiguration	62
4.36 Dialog – Eingabe	63
4.37 Visu – Temperaturverläufe	63
4.38 Visu – Temperatur-Sollwerte	64
4.39 Smart Home Zentrale	65
4.40 Anschlüsse – Smart Home Zentrale	65
4.41 Portexpander MCP23017	67
4.42 PSpice Model mit Zener-Diode ZPD4.7V	68
4.43 PSpice Model – Pegelwandlung 4.7V	68
4.44 MCP23017 – Initialisierung	69
4.45 MCP23017 – Adressierung	69
4.46 MCP23017 – Register-Zugriff	70
4.47 Struktogramm – Funk IO-Modul	71
4.48 Funk IO-Modul	72
4.49 Funk IO-Modul – Anschlüsse	72
4.50 Arduino Mini Pro	73
4.51 FDTI Treiber Konfiguration	75
4.52 Struktogramm – Funk Sensor	76
4.53 Code-Ausschnitt – Watchdog Timer	77

4.54 MCU Status Register	77
4.55 Watchdog Timer Control Register	77
4.56 Code-Ausschnitt – Sleep-Mode	78
4.57 Funkübertragung – Sensor - Zentrale	79
4.58 GUI – Z-Way-Software	81
4.59 GUI – Z-Way Control	82
4.60 Z-Way – Configuration	83
4.61 Z-Way – Multisensor-Return	84
4.62 Z-Way – Taster3-Return	85
4.63 Code-Ausschnitt – Requestfilter	86
A.1 Schaltplan – Smart Home Zentrale	91
A.2 Platinenlayout – Smart Home Zentrale	92
A.3 Platine – Smart Home Zentrale	92
A.4 Schaltplan – Funk IO-Modul	94
A.5 Platinenlayout – Funk IO-Modul	95
A.6 Platine – Funk IO-Modul	95
A.7 Schaltplan – Sensor-Modul	97
A.8 Platinenlayout – Sensor-Modul	98

1 Einführung

1.1 Motivation

In der heutigen Zeit ist der Begriff Smart, nicht zuletzt durch den großen Erfolg des Smartphones, welches das mobile Telefon zur ausschließlichen Nutzung des Telefonierens ablöste, bereits ein weit verbreiteter Begriff. Wurden die ersten Smartphones noch kritisch von der breiten Masse angenommen, sind die heutigen Geräte mit all ihren Features und optionaler Erweiterbarkeit durch verschiedenste Applikationen kaum wegzudenken. Eine ähnliche Entwicklung, welche sich allerdings noch im Anfangsstadium befindet, ist aktuell auch im Bereich Smart Home deutlich zu erkennen. Von einer Annahme der Systeme in der breiten Masse kann zwar nicht die Rede sein, dennoch steigen immer mehr Anbieter mit eigenen Entwicklungen im Smart Home Bereich zur automatisierten Haussteuerung ein.

Das Prinzip der Vernetzung von einzelnen Gegenständen, die selbständig miteinander kommunizieren, ist grundsätzlich nichts Neues. Bereits 1991 beschrieb der US-amerikanische Informatikwissenschaftler Mark Weisser seine Vision der Vernetzung in seinem Aufsatz „The Computer for the 21st Century [27]“ und schuf darin erstmals den Begriff *Internet of Things (IoT)*¹. Nach seiner damaligen Auffassung schon wird der Computer als alleinstehendes Werkzeug zur aktiven Nutzung sein Dasein verlieren. Viele kleine, eingebettete Systeme werden sich auf sämtliche Gegenstände verteilen und passiv unbemerkt Aufgaben steuern. Im 21. Jahrhundert angekommen, nimmt seine Vision langsam Gestalt an und ist besonders im Bereich Smart Home erkennbar. Kostengünstige Kleinst-Computer bzw. Einplatinencomputer der heutigen Zeit sind in ihrer Leistungsfähigkeit nicht mit Desktop-Rechnern der damaligen Zeit zu vergleichen und ebnen den Weg für das Internet der Dinge. Die Entwicklung der Smart Home Systeme steht zwar noch am Anfang, allerdings steht die zukünftige Vernetzung sämtlicher Gegenstände außer Frage.

Große Firmen wie beispielsweise KNX bieten bereits seit vielen Jahren Smart Home Systeme an, jedoch beruhen die ersten Systeme auf Bus-Kommunikation und sind somit nur bei Neuinstallationen realisierbar. Die Kosten der Gerätschaften solcher Systeme laufen schnell in den fünf-stelligen Bereich und sind somit kaum tragbar für die breite Masse.

¹Internet der Dinge (Internet of Things)

Grund für den aktuellen Boom in diesem Bereich sind die Entwicklungen der letzten Jahre im Bereich der Funk-Übertragung. Neue, kostengünstige Funk-Entwicklungen eröffnen nun die Möglichkeit auch bestehende Hauselektronik zu Smart Home Systemen umzubauen und werden insbesondere durch den geringen Anschaffungspreis auch für die breite Masse interessant.

Autonome Systeme wie bspw. Alarmanlage, Rauchmeldeanlage, Heizungsregelung, Ansteuerung von Beleuchtung oder Messeinrichtungen für Strom- und Wasserverbrauch sollen in Zukunft weichen und zentral über das Smart Home System organisiert und vor allem visualisiert werden. Aktuell hat sich für die Funkübertragung noch kein Standard durchgesetzt. Viele Anbieter setzen eigene Standards und sind zumeist proprietäre Systeme. Die folgende Kundenbindung für zukünftige Komplementärprodukte ist dem Anbieter dadurch gesichert. Die Wahl des geeigneten Systems muss demnach wohl bedacht getroffen werden. Mit Blick auf die Zukunft erweist sich dies aktuell allerdings als nicht leicht. Sind für das System in 5 Jahren noch Geräte mit der speziellen Funktechnik erhältlich? Wird die Funk-Technik von mehreren Anbietern unterstützt? Wie groß ist das Angebot unterschiedlicher Hauselektronik?

1.2 Zielsetzung dieser Arbeit

Ziel dieser Arbeit ist die Realisierung eines kostengünstigen Smart Home Systems für die webbasierte Automatisierung eines Privathaushalt auf Basis von Einplatinencomputern wie Raspberry Pi und Arduino. Das System soll sowohl für Neuinstallationen als auch als Integration in bestehender Installation Anwendung finden. Das System besticht durch eine anwendungsfreundliche und einfache Bedienung. Als weiteres Feature soll das System einen industriellen Funk-Standard der Hausautomatisierung unterstützen und dadurch die Ansteuerung von industrieller Hauselektronik ermöglichen.

2 Technische Grundlagen

2.1 Raspberry Pi

Der Raspberry Pi ist ein Einplatinencomputer, der von der britischen Stiftung *Raspberry Pi Foundation* entwickelt wurde. Der erste Prototyp wurde bereits im Jahre 2006 produziert. Ziel der Entwicklung war es, jungen Menschen den Erwerb von Programmierkenntnissen nahe zu bringen. Der Verkaufspreis sollte entsprechend niedrig sein [6]. Die Leiterplatte des Raspberry Pi bzw. RasPi, wie er unter Kennern genannt wird, ist dabei nicht größer als eine Kreditkarte. Der kleine Rechner besitzt keinen festen Speicher, wodurch das Betriebssystem von einer SD-Karte gebootet werden muss. Er verfügt über einen SD-Kartenleser, einen Ethernetanschluss, USB-Anschlüsse, sowie Audio- und Videoanschlüsse. Je nach Modell verfügt er zudem über 16 bis 40 digitale bzw. analoge Anschluss-Pins. Auf der Hersteller-Seite sind bereits unterschiedliche Linux-Distributionen als Open Source Lizenz frei erhältlich. Das bekannteste, speziell für den Einplatinencomputer angepasste Linux-System ist *Raspbian* und wird an dieser Stelle lediglich erwähnt.

Das Logo der Firma ist eine Himbeere und knüpft an die Tradition, Rechner nach Früchten zu benennen wie bspw. Apple oder Acorn. Der Name wird ähnlich dem englischen Wort für Himbierkuchen „*raspberry pie*“ ausgesprochen. Das „Pi“ steht dabei für „Python Interpreter“, da der erste Prototyp mit fest eingebautem Interpreter für die Programmiersprache *Python* entwickelt wurde[7]. Die Leistung des besagten Prototyps stimmte die Entwickler allerdings nicht sehr zufrieden. Durch den damals anfänglichen Boom der Smartphones kamen dann die sogenannten ARM-Prozessoren auf den Markt. Diese verfügten über eine verhältnismäßig hohe Leistung und waren zudem sehr günstig. Mit Anfang des Jahres 2012 kommt schlussendlich der erste Raspberry Pi für einen Verkaufswert von USD 25-35 auf den Markt. Bis Februar 2015 wurden bereits 5 Mio. Stück verkauft. Es existiert ein großes Zubehör- und Softwareangebot für die verschiedensten Anwendungsbereiche. Auch im Bereich Smart Home gelangte er immer größere Beliebtheit, da bereits Anbieter von Funk-Techniken wie beispielsweise ZigBee oder Z-Wave spezielle Empfangsmodule für den Einplatinencomputer auf den Markt brachten. [5]

Eigenschaften

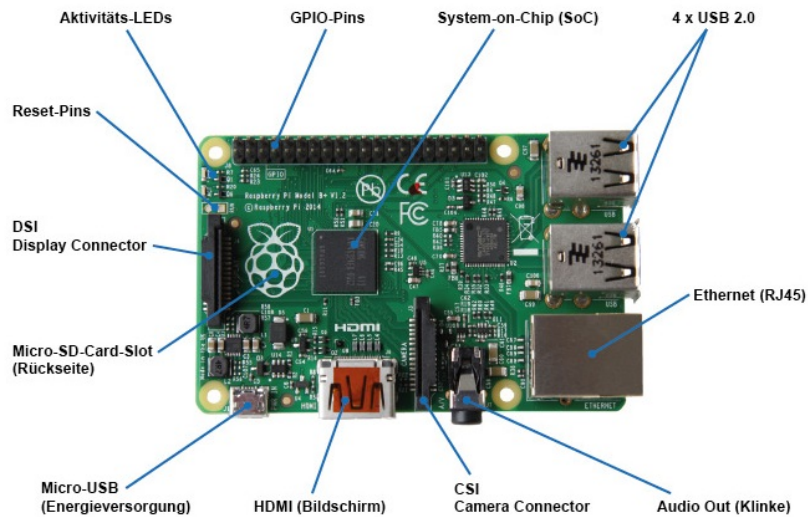


Abbildung 2.1: Raspberry Pi Model B+ [8]

Seit dem 14. Juli 2014 ist das Model B+ (Abb. 2.1) erhältlich und reiht sich als 4. Weiterentwicklung in die Raspberry Pi Model-Serie ein. Herzstück des RasPi ist die BCM2835-Architektur mit einer ARM11-CPU, welche mit 700MHz getaktet und mit einem Arbeitsspeicher von 512MB – RAM ausgestattet ist. Der benannte Prozessor findet auch in Netbooks seine Verwendung. Im Gegensatz zu seinen Vorgängern besitzt er vier USB 2.0 -Anschlüsse sowie 40 programmierbare [GPIO¹](#)-Schnittstellen. Durch den Einsatz von Schaltreglern wurde der Energieverbrauch bei einer Spannungsaufnahme von 5V , von 750mA auf 600mA reduziert. Für Speichermedien wurde ein MicroSD-Slot eingesetzt. Des Weiteren steht ein Audio-Anschluss als 4-polige 3.5-mm-Klinkenbuchse zur Verfügung sowie ein HDMI-Anschluss für Video und eine weitere Kameraschnittstelle für spezielle Erweiterungs-Boards. [8] [9]

In folgender Tabelle 2.1 werden sämtliche Eigenschaften im Vergleich zu weiteren Modellen aufgelistet. Zeitgleich zu dieser Arbeit ist nun auch das neueste Modell Raspberry Pi 2, mit neuen 4-Kern Prozessor und 900MHz Taktung erschienen.

¹Allzweckeingabe/-ausgabe (General Purpose Input/Output)

Tabelle 2.1: Raspberry Pi Modelle [8]

	Modell A	Modell A+	Modell B	Modell B+	PI 2 Modell B
Preis	ca. 25€			ca. 30€	ca. 35€
CPU	1				4
	700MHz				900MHz
RAM	265MB		512MB		1024MB
USB-2.0	1		2	4	
Video	HDMI				
Ton	HDMI (digital)	3.5mm-Klinkenstecker (analog), HDMI (digital)			
Speicher	SD	microSD	SD	microSD	
Netzwerk	-		10/100-MBit Ethernet		
Pins	26	40	26	40	
GPIO-Pins	17	26	17	26	
Leistungsaufnahme	500mA (2.5W)	230mA (1.2W)	700mA (3.5W)	500mA (2.5W)	800mA (4W)
Stromversorgung	5V (Micro-USB-Anschluss)				

GPIO-Schnittstelle

Die neueren Raspberry Pi Modelle A+, B+ sowie das kürzlich erschienene Modell Raspberry 2 sind mit 40 PIN-Anschlüssen ausgestattet. 26 davon sind dabei als **GPIO**-Schnittstellen verwendbar. In folgender Abbildung 2.2 sind die Anschlüsse und deren mögliche Verwendung abgebildet. Die Vorläufermodelle wurden damit um 14 weitere Pins erweitert. Bezogen auf die Belegung hat sich bis Pin 26 nichts geändert, wonach hinsichtlich älterer Programmierungen somit keine Änderungen des Programm-Codes erforderlich sind. Mit Pin 27(ID_SD) und 28(ID_SC) besteht nun auch die Anschluss-Möglichkeit eines ID **EEPROM**² als persistenten Speicherbaustein. Die Schnittstellen bieten Anschluss für externe Bauteile mit 3.3V sowie 5V Spannungsversorgung. Besonders 5V findet bei der Überzahl an 5V-betriebenen Sensoren und Aktoren seine Verwendung. Hierzu sei zu beachten, dass die GPIO-Schnittstellen mit 3.3V-Logik arbeiten und somit nicht 5V-tolerant sind. Die maximale Strombelastung je GPIO-Pin beträgt 16mA! Für 5V-Bauteile muss eine Spannungsanpassung erfolgen und die Strombelastung berücksichtigt werden. Der 5V-Anschluss kann zugleich auch als Spannungsversorgung des Raspberry Pi verwendet werden. Die gekennzeichneten GPIO-Schnittstellen können je nach Programmierung als digitaler Eingang bzw. Ausgang genutzt werden. Einige Pins besitzen zudem weitere, alternative Funktionalitäten. GPIO02/03 bietet Anschluss zur Kommunikation mit dem **I²C**³-Bus (siehe S.12). Die Anschlüsse GPIO14/15 können als serielle **UART**⁴-Schnittstelle genutzt werden. Auch die

²elektrisch, löschbarer, programmierbarer Nur-Lese-Speicher (electrically erasable programmable read-only Memory)

³Serieller Datenbus, I-Quadrat-C (Inter-Integrated-Bus)

⁴Serielle Schnittstelle (Universal Asynchron Receiver Transmitter)

Nutzbarkeit des SPI⁵-Bus (siehe S.14) ist mit den Anschlüssen GPIO7-11 gegeben. Je nach Anwendungsbereich lässt sich der Raspberry um weitere Bausteine beispielsweise AD-Wandler, Funkeinheiten, GPIO-Erweiterungen usw. erweitern. [2][10]

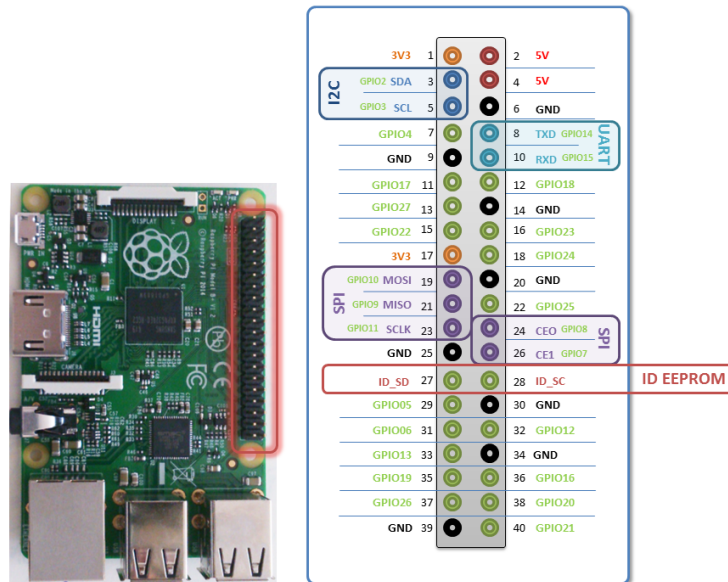


Abbildung 2.2: Raspberry GPIO-Schnittstelle [10]

2.2 Arduino

Der erste Mikrocontroller aus dem Hause Arduino kam bereits im Jahre 2005 auf den Markt. Mittlerweile genießt er eine sehr beachtliche Community, was nicht zuletzt auf den günstigen Anschaffungspreis zurückzuführen ist. Der Name *Arduino* basiert auf einer Bar in der italienischen Stadt Ivrea, wo sich die Gründer Massimo Benzi und Cuartielles David gewöhnlich trafen [3]. Bei Arduino handelt es sich nicht ausschließlich um eine Hardware, sondern auch um eine Software. Dieser Symbiose kommt der sogenannte Name *Physical Computing* zuteil. Er beschreibt die Beziehung zwischen Mensch und Computer, wobei die reale Welt als analoges System gesehen wird. Im Gegensatz dazu agieren und reagieren die Computer im digitalen Umfeld mit den logischen Zuständen 0 und 1. Arduino fällt zudem in die Kategorie *Open Source* und ist somit frei und ohne Lizenz verfügbar. Jedermann bzw. -frau kann sich an unterschiedlichen Entwicklungen beteiligen, was nicht zuletzt für die große Community spricht. [1]

⁵Synchron, serieller Datenbus (Serial Peripheral Interface)

Arduino bietet mittlerweile ein breites Angebot unterschiedlicher Mikrocontroller. Die Beschreibung der einzelnen Modelle würde den Rahmen dieser Arbeit sprengen. Die Modelle unterscheiden sich grundsätzlich in ihrer Bauweise und insbesondere der Anschlussmöglichkeit, sowie dem verwendeten Mikrocontroller-Typ aus dem Hause *Atmel AVR*. Das bekannteste Modell ist der sogenannte *Arduino UNO*, welcher folgend als Referenzmodell im Detail vorgestellt wird. Insbesondere für das Prototyping ist das Modell *UNO* erste Wahl, da die Anschlüsse einfach über Steckmodule zu erreichen sind und somit schnelle Test-Aufbauten ermöglichen.

Eigenschaften



Abbildung 2.3: Arduino UNO

- Mikrocontroller Atmel AVR ATmega328P
- Betriebsspannung 5V
- Eingangsspannung 7-12V
- Digitale Ein-/Ausgänge 14 (davon 6 als Pulsweitenmodulation ([PWM](#)⁶) nutzbar)
- [DC](#)⁷ Strom/[PIN](#)⁸ 20mA
- Digitale [PWM](#) Ein-/Ausgänge 6
- Flash Speicher (ATmega328P) 32KB
- Taktgeschwindigkeit 16MHz [4]

⁶Pulsweitenmodulation (Verzeichnisdienst von Microsoft Windows Server)

⁷Gleichstrom (Direct Current)

⁸Metallischer Anschlussstift auf einer Leiterplatte

Software Arduino IDE

Die integrierte Entwicklungsumgebung *Arduino IDE*⁹ wird, wie bereits erwähnt, mit offener Lizenz zur Verfügung gestellt und ist für Windows, Mac OS X sowie Linux auf der Herstellerseite erhältlich. Die Umgebung ist mit sämtlichen Hardware-Treibern der verschiedenen Arduino-Modelle ausgestattet, welche über das Menü selektiert werden können. Mit dem vorgestellten Modell Arduino UNO, der bereits über einen **USB**¹⁰-Anschluss verfügt, kann eine serielle Verbindung, durch Auswahl des verwendeten Port am Entwicklungs-Rechner, mit der Entwicklungsumgebung hergestellt werden.

Die **IDE** ist sehr einfach aufgebaut. In folgender Abbildung 2.4 ist die Entwicklungsumgebung dargestellt. Die Programmiersprache des Arduinos ist C bzw. C++. Das Programm bzw. *Sketch*, wie es bei Arduino genannt wird, kann grundsätzlich in drei Bereiche eingeteilt werden. In den ersten Zeilen erfolgt klassisch das Einbinden der verwendeten Bibliotheken und Definition der verwendeten Variablen. Der zweite Bereich wird mit der Funktion `void setup()` beschrieben. Sie wird einmalig beim Start des Mikrocontrollers ausgeführt und ist für eventuelle Initialisierungen zuständig. Der letzte Bereich ist die Funktion `void loop()`, dessen Inhalt kontinuierlich durchgeführt wird. Auslagern von eigenen Funktionen ist ebenfalls erlaubt. Mit dem Button „Verifizieren“ in der Entwicklungsumgebung erfolgt die Fehlerüberprüfung, wobei durch Auswahl eines geeigneten Compiler der Sketch-Code für die Verwendung auf dem Zielgerät übersetzt wird. Mit dem Upload-Button erfolgt abschließend das Hochladen des Sketches auf den Mikrocontroller über USB-Verbindung.

Da der Entwicklungsumgebung Arduino IDE ein Debug-Modus gänzlich fehlt, ist der integrierte „Serielle Monitor“ ein sehr hilfreiches Werkzeug um Programm-Fehler zu lokalisieren. Der Aufruf der Funktion `Serial.begin(9600)` initialisiert dabei bspw. eine serielle Datenübertragung mit einer Baudrate von 9600 Bits/sec . Mit der Ausgabe-Anweisung `Serial.println()` können dann Variablen zur Ausführungszeit auf dem seriellen Monitor der Entwicklungsumgebung ausgegeben werden.

⁹Integrierte Entwicklungsumgebung (Integrated Development Enviroment)

¹⁰Universelle Bus Schnittstelle (Universal Serial Bus)

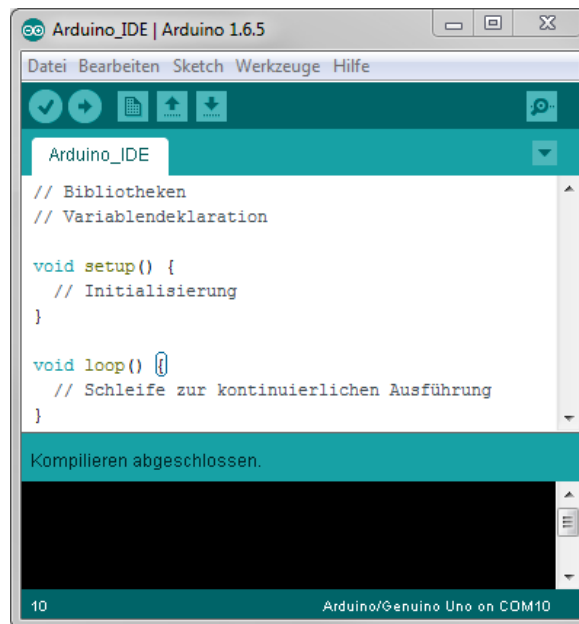


Abbildung 2.4: Arduino IDE

2.3 I²C-Bus

I²C ist ein synchroner, serieller Datenbus, der lediglich mit zwei Signalleitungen, einer Daten- und einer Taktleitung betrieben wird. I²C steht für *Inter-Integrated-Circuit* und wird zur Kommunikation zwischen integrierten Schaltungen (ICs¹¹) über kleine Distanzen verwendet. Entwickelt wurde er Anfang der 80er Jahre von Philips Semiconductors, was heute NXP Semiconductors entspricht. Bei anderen Herstellern wird die Bus-Technik aus Lizenzgründen auch TWI¹² (Two Wire Interface) genannt.

Ein I²C-Bus-System besteht aus mindestens einem Master und weiteren Slave-Teilnehmern bzw. weiteren Mastern, wobei bei Verwendung mehrerer Master die Arbitrierung berücksichtigt werden muss. Die erste Entwicklung von 1982 ließ eine Datenübertragung von 100 kbit/s mit einem 7-Bit-Adressraum für Bus-Teilnehmer zu. Prinzipiell lassen sich damit 128 Adressen ansprechen. Da einige Adressen allerdings reserviert sind beschränkt sich die Adressmöglichkeit auf 112 Adressen. Die erste standardisierte Spezifikation 1.0 (Standard Mode) wurde 1992 mit erweitertem *Fast-Mode* veröffentlicht, der zudem Datenübertragungen von 400 kbit/s ermöglichte und um einen weiteren 10-Bit-Adressraum erweitert wurde. Dies ermöglichte eine Gesamtteilnehmeranzahl von 1136. Beide Adressierungsvarianten

¹¹ Integrierte Schaltkreise (Integrated Circuit)

¹² Konform zu I²C-Bus (Two Wire Interface)

sind im selben I²C-Bus-System anwendbar. Bis Dato wurden weitere Spezifikationen um Hs-Mode, über Fast-Mode-plus bis Ultra Fast-Mode ergänzt und veröffentlicht, die sich lediglich in der Übertragungsgeschwindigkeit unterscheiden. Übertragungsraten höher 400kbit/s werden jedoch von der Großzahl an ICs kaum unterstützt. Bis auf den Ultra-Fast-Modus, der keine Rückmeldungen zulässt, entsprechen alle Modi einer bidirektionalen Kommunikation. Die Eigenschaften der einzelnen Modi werden in folgender Tabelle zusammengefasst. [11][12]

Tabelle 2.2: I²C-Bus Modi [11][12]

Jahr	Modus	Kommunikation	Übertragung
1982	Standard Mode (Sm)	bidirektional	0.1 Mbit/s
1992	Fast Mode (Fm)	bidirektional	0.4 Mbit/s
1998	Highspeed Mode (Hs-Mode)	bidirektional	3.4 Mbit/s
2007	Fast Mode Plus (Fm+)	bidirektional	1 Mbit/s
2012	Ultra Fast Mode (UFm)	unidirektional	5 Mbit/s

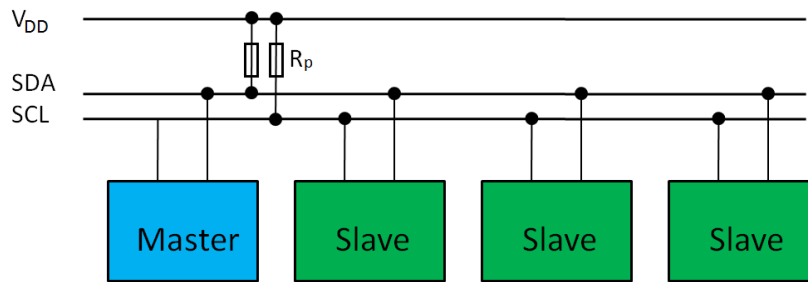
Funktionsweise

Der I²C-Bus benötigt wie bereits erwähnt zwei Signalleitungen, Taktsignal (SCL¹³) und eine Datenleitung (SDA¹⁴). Der Bus arbeitet dabei nach dem Master-Slave-Konzept. Das Taktsignal wird vom Master vorgegeben, der auch die entsprechenden Slaves initiiert und damit zur Kommunikation auffordert. Ein Slave-Teilnehmer kann von sich aus keine Verbindungen zum Master bzw. anderen Slaves initiieren und ist im Grunde nur passiver Bus-Teilnehmer. Je nach verwendetem Protokoll wird die Datenleitung für bidirektionale bzw. unidirektionale Übertragung verwendet. Der Bus arbeitet dabei mit positiver Logik (*Most Significant Bit first*). Bedingt durch die beschränkte Buskapazität von 400pF sind nur kurze Leitungen von 2 bis 3 Metern möglich. Der I²C-Bus ist somit grundsätzlich für systeminterne Kommunikationen, für die er auch entwickelt wurde, anwendbar. Trotzdem können durch sogenannte I²C-Extender auch längere Strecken bis zu 100m erreicht werden. Bei Verwendung mit mehreren Mastern im *Multimaster-Mode*, erfolgt eine Arbitrierung (Zugriffsregelung), die per Spezifikation geregelt ist.

Beide Signalleitungen liegen über Pull-Up-Widerständen an der Versorgungsspannung. Sämtliche Bus-Teilnehmer haben *Open-Collector-Ausgänge*, die zusammen mit den Pull-Up-Widerständen eine *Wired-AND-Schaltung* ergeben. Der High-Pegel als logische Eins muss dabei mindestens $0.7 \times V_{DD}$, und der Low-Pegel als logische Null höchstens $0.3 \times V_{DD}$ betragen.

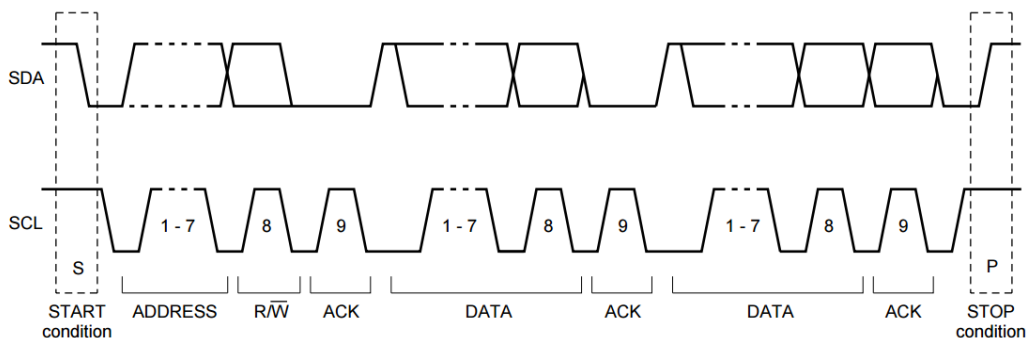
¹³Seriellles Taktsignal (Serial Clock)

¹⁴Seriellles Datensignal (Serial Data)

Abbildung 2.5: I²C-Bus Master-Slave-Konzept

Übertragungsverlauf mit 7-Bit-Adressierung

In Abbildung 2.6 ist ein kompletter Übertragungsverlauf eines Sende- bzw. Empfangsprotokolls abgebildet. Der Master, der das Taktsignal vorgibt, signalisiert durch ein konstantes HIGH am Taktsignal und einer fallenden Flanke am Datensignal den Beginn einer Übertragung. Es folgt die 7-Bit-Adressierung des entsprechenden Slaves, sowie das R/W-Bit. Durch setzen des R/W-Bits legt der Master die Kommunikationsrichtung fest. LOW steht dabei für *write* und somit dafür, um Daten zum Slave zu senden und HIGH für *read*, um Daten von Slave anzufordern. Der adressierte Slave sendet daraufhin ein ACK als Bestätigung. Je nach Kommunikationsrichtung sendet nun der Master oder der entsprechende Slave die Daten. Nach jedem übertragenen Byte gibt der Kommunikationspartner mit einem **ACK¹⁵** Auskunft über dessen Erfolg. Das letzte Byte wird abschließend vom Master mit einem **NACK¹⁶** quittiert und signalisiert das Ende der Übertragung. Abschließend erfolgt das Stop-Signal durch konstantes HIGH am Taktsignal und steigender Flanke am Datensignal. [11]

Abbildung 2.6: I²C Übertragungsverlauf [11]

¹⁵Zustimmung (Acknowledgment)

¹⁶Ablehnung (Negativ-Acknowledgment)

10-Bit-Adressierung

10-Bit-Adressierungen sind im selben Bus-System mit 7-Bit-Adressierungen kombinierbar. Dabei sendet der Master die reservierte Adresse „11110XX“ und verhindert dadurch Komplikationen mit 7-Bit-adressierten Teilnehmern. „XX“ stellt dabei bereits die ersten 2 Positionen der 10-Bit-Adresse dar. Nach dem klassischen Übertragungsverlauf (Abb.2.6) folgt darauf das R/W-Bit und die ACK-Bestätigung, welche nun auch eventuell von mehreren 10-Bit-Slaves erfolgen kann. Nun folgen die restlichen 8 Bit der 10-Bit-Adresse, welche die Kommunikation mit einem ACK bestätigt. Folgend werden je nach Kommunikationsrichtung die einzelnen Bytes übertragen und jeweils mit einem ACK bestätigt. [12]

Arbitrierung

Wie bereits erwähnt kann I²C auch im Multimaster-Mode betrieben werden. Hierfür ist allerdings die Arbitrierung (Zugriffregelung) zu berücksichtigen um Kollisionen zu vermeiden. Da alle Master das Taktsignal vorgeben können, ist eine Synchronisation der Taktgeneratoren voraussetzend. Alle sende-willigen Master hören zunächst den Bus auf „Freiheit“ ab. Sobald ein oder mehrere Master den freien Bus erkennen, beginnen sie unter Umständen gleichzeitig die Start-Bedingung zu senden. Nach dem bereits vorgestellten Übertragungsverlauf folgen Adressierung und Daten. Jeder Master beobachtet dabei, ob seine Ausgabe auch der Bus-Aktivität entspricht. Dominante 0-Bits setzen sich nämlich gegen rezessive 1-Bits durch. Ist der Bus-Zustand nicht mit der Ausgabe des entsprechenden Masters kongruent, so überlässt er dem Master mit dominanter Bit-Ausgabe den Bus und beendet seinen Zugriffsversuch sowie auch die Ausgabe des Taktsignales. Seine Bus-Schnittstelle schaltet er wieder auf Empfang und wartet auf die nächste, mögliche Bus-Freiheit. Dieses Arbitrierungsverfahren wird über alle Bits fortgeführt, bis sich ein Master durchsetzt. [13]

2.4 SPI-Bus

Im Gegensatz zum I²C-Bus besteht der SPI-Bus aus vier Signalleitungen und wird in erster Linie für eine synchrone, serielle Kommunikation von Hostprozessor und Peripheriebausteinen verwendet. Der SPI-Bus wurde von Motorola mit einem „lockeren“ Standard zur seriellen Kommunikation nach dem Master-Slave-Prinzip entwickelt. Anders als der I²C-Bus ist SPI vollduplexfähig, da für jede Kommunikationsrichtung eine separate Datenleitung verwendet wird. Der SPI-Master gibt das Taktsignal (SCKL: Serial-Clock) vor und aktiviert über die Chip-Select-Leitung (CS), oder auch Slave-Select (SS) genannt, den Slave-Baustein mit dem er kommunizieren möchte. Taktsignal sowie die Datensignale MOSI (Master Output, Slave Input) und MISO (Master Input, Slave Output) werden von allen Teilnehmern parallel verwendet (Oftmals auch als SDO-Serial Data Out bzw. SDI-Serial Data In bezeichnet). Die Vernetzung der Bus-Teilnehmer erfolgt dabei standardmäßig als sternförmige Infrastruktur

(Abb. 2.7). Die Grenze der Slave-Teilnehmer wird durch die Anzahl der CS-Signale des Masters bestimmt. Eine weitere Möglichkeit besteht darin, sämtliche Slaves in kaskadierter Form zu vernetzen. Die Datenleitungen verlaufen dabei in Ringform durch sämtliche Bus-Teilnehmer; Datenausgang eines Slaves liefert Dateneingang des folgenden Slaves. Sämtliche Slaves werden dabei als ein größeres Element betrachtet und werden allesamt über dasselbe CS-Signal angesteuert. [14]

Durch den „lockeren“ Standard sind viele Eigenschaften des Bus-Systems nicht festgelegt. Es sind Taktfrequenzen bis in den MHz-Bereich zulässig. Auch die Bitlänge einer Datenübertragung (gewöhnlich 8 Bit), sowie welche Taktflanke zur Signalisierung genutzt wird sind nicht festgelegt. Des Weiteren steht die Reihenfolge der Übertragung offen; **MSB**¹⁷ oder **LSB**¹⁸ zuerst. Jedes SPI-System hat somit seine eigene Konfiguration, wobei es zu Problemen mit inkompatiblen Geräten kommen kann. [14]

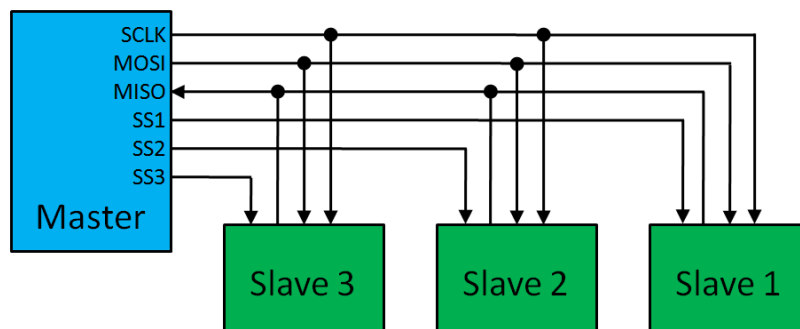


Abbildung 2.7: SPI-Bus mit sternförmiger Infrastruktur [14]

Funktionsweise

Der Master legt, wie bereits erwähnt, das Taktsignal auf der **SCLK**¹⁹-Leitung fest. Über das CS-Signal spricht er den entsprechenden Slave an und zieht dafür das Signal auf Masse. Der Slave wird dadurch aktiviert und „lauscht“ am **MOSI**²⁰-Signal. Seine Daten legt er im Takt zeitgleich auf das **MISO**²¹-Signal. Dabei wird zum einen ein Wort vom Master zum Slave, zum anderen eine Antwort vom Slave zum Master übertragen. [14]

Ein Übertragungsprotokoll wurde von Motorola zwar nicht festgelegt, trotzdem haben sich in der Praxis vier verschiedene Modi durchgesetzt. Beschrieben werden sie mit dem Modi-Parameter Clock Polarität (CPOL) und Clock Phase (CPHA). Die unterschiedlichen

¹⁷Höchstwertige Bit (most significant bit)

¹⁸Niedrigstwertige Bit (least significant bit)

¹⁹Serielltes Taktsignal (Serial Clock)

²⁰Master Out Slave In

²¹Master In Slave Out

Modi mit jeweiliger Parameterbeschreibung sind der Tabelle 2.3 zu entnehmen.

CPOL=0 beschreibt den Clock-Signal-Ruhezustand bei LOW. Kombiniert mit CPHA=0 wird das Datenbit bei der folgenden, steigenden Flanke abgetastet. Mit CPHA=1 erfolgt die Abtastung bei der fallenden Flanke und bewirkt somit eine Verzögerung der Abtastung.

CPOL=1 beschreibt hingegen den Clock-Signal-Ruhezustand bei HIGH. Mit CPHA=0 erfolgt die Abtastung an folgender, fallenden Flanke, bei CPHA=1 wiederum verzögert an darauffolgender, steigender Flanke.

Tabelle 2.3: SPI - Modi

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

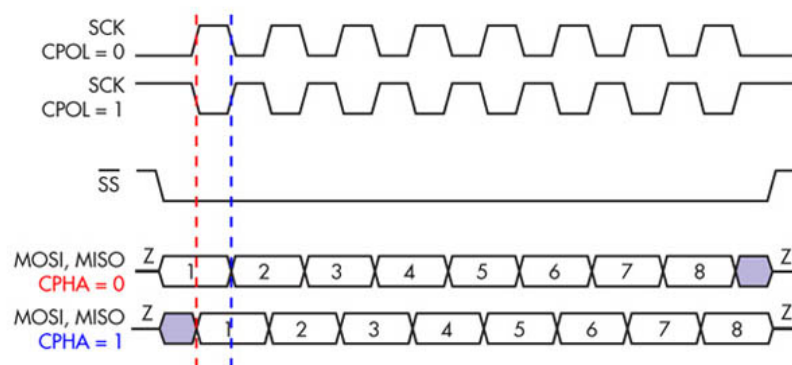


Abbildung 2.8: SPI – Abtastung [14]

2.5 Z-Wave

Z-Wave ist ein drahtloser Kommunikationsstandard, der von zwei dänischen Ingenieuren speziell für die Hausautomatisierung Ende der 90er Jahre entwickelt wurde. Die Funktechnik wurde anfangs von der Firma Zen-Sys mit Sitz in Kopenhagen vertrieben, die 2009 von Sigma Design übernommen wurde. Im Jahre 2005 wurde die sogenannte *Z-Wave-Allianz* als Zusammenschluss von mehr als 200 Herstellern weltweit, welche die Funktechnik einsetzen,

gegründet. Die Z-Wave Zertifizierung, welche nach wie vor von Sigma Design durchgeführt wird, garantiert dabei, dass sämtliche Geräte unterschiedlicher Hersteller kompatibel sind.

Mit mehr als 1300 verschiedenen Produkten bildet die Z-Wave-Allianz das weltweit größte Öko-System von drahtloser Heimvernetzung. 2012 wurde Z-Wave ein öffentlicher Standard und als Standard G.9959 der internationalen Telekommunikationsunion ITU-T standardisiert [ITU2012]. [15]

Für die Funkkommunikation mit Z-Wave ist ein entsprechender Z-Wave-Funk-Transceiver erforderlich. Hauptanbieter der Funk-Hardware ist das amerikanische Unternehmen *Sigma Design*. Neben den notwendigen Bauteilen zur Funkübertragung sind die Module je nach Version mit erweiterten Ein- und Ausgängen für Aktorik und Sensorik für wenige Dollar erhältlich. Zur direkten Programmierung der Modul-Firmware ist allerdings ein spezielles System Development Kit (SDK) erforderlich, das je nach Versionsumfang zwischen 1.500 und 3.500 USD kostet und somit nur Entwicklern von Z-Wave Produkten vorbehalten ist. [21]

Mit dem großen Erfolg des Einplatinencomputer Raspberry Pi bietet die Firma Z-Wave.me nun auch eine spezielle Erweiterungsplatine, das Funk-Modul *RaZBerry*. Es beinhaltet die Z-Wave Hardware von Delta Sigma und wurde speziell für den Raspberry Pi entwickelt. Das Modul besitzt einen GPIO-Stecker und wird direkt auf den GPIO-Anschluss des Raspberry aufgesetzt. Zur Ansteuerung des Moduls ist eine spezielle Serversoftware (Z-Way) notwendig. Die Software bietet eine webbasierte [JSON²²](#)-Schnittstelle und ermöglicht die indirekte Steuerung der Chip-Firmware von Delta Sigma und somit die Steuerung des Z-Wave-Netzwerkes (Abbildung 2.9). Mit dem Einsatz des Z-Wave-Servers sind eigene Smart Home Entwicklungen möglich, allerdings nur in Form der zentralen Steuerung von industriellen Endgeräten mit Z-Wave Technologie. [20]

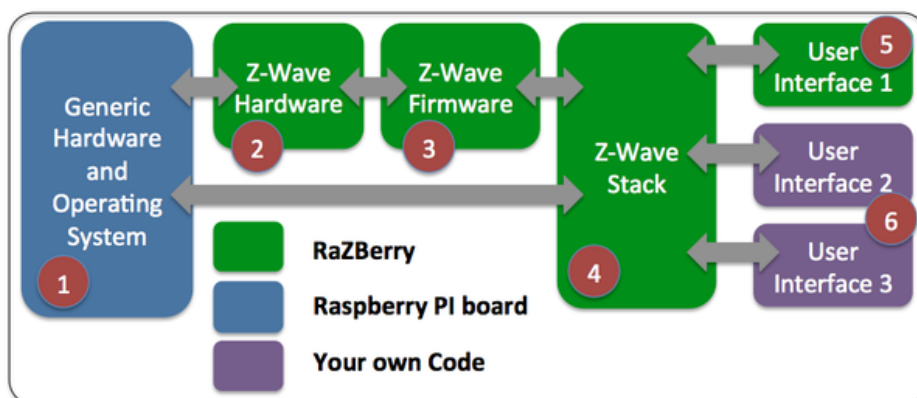


Abbildung 2.9: Z-Wave Modul RaZBerry [20]

²²Kompaktes Datenformat zum Zweck des Datenaustausches zwischen Anwendungen (JavaScript Object Notation)

3 Marktanalyse und Konzeptentwurf

3.1 Marktanalyse

Im Bereich der Smart Home Technologie kommen je nach Hersteller unterschiedliche Funkstandards zum Einsatz, die zugleich das größte Unterscheidungsmerkmal der Systeme darstellen. Viele Unternehmen wollen den aktuellen Hype „Smart Home“ nicht versäumen und daher verfolgen sie unterschiedliche Ansätze, zumal sich speziell für die unverzichtbare Funkübertragung noch kein Standard durchgesetzt hat. Für den Anwender stellt sich somit als wichtigste Frage, welcher Funkstandard auch in Zukunft noch unterstützt wird und von genügend Herstellern supported wird.

3.1.1 Funkübertragung

Für die Funkübertragung stehen allgemein die lizenzfreien ISM-Bänder (Industrial, Scientific and Medical) um die Frequenzen 433MHz , 866MHz sowie 2.4GHz (in Europa) zur Verfügung. Ein Großteil der Privathaushalte nutzt bereits mit W-LAN und Bluetooth das 2.4GHz Netz. Somit liegt der Gedanken nahe das bestehende Netz auch für die Hausautomation zu verwenden. Diesen Gedanken verfolgen auch einzelne Hersteller, allerdings konzentriert sich der Großteil der Smart-Home-Technologien im 866MHz Bereich. Zum einen sind Empfänger und Sender für W-LAN¹-Anbindung relativ teuer, zum anderen wurde das Netz vor allem für Schnelligkeit, Übertragung von großen Datenmengen und Sicherheit entwickelt, was sich im Energieverbrauch niederschlägt. Bei der Hausautomatisierung steht der Fokus vielmehr auf Zuverlässigkeit, Sicherheit und geringem Energieverbrauch bei Übertragung von kleinen Datenmengen.

Es wird eine engere Auswahl von unterschiedlichen Funksystemen getroffen. Folgend werden die Funksysteme ZigBee, EnOcean, Z-Wave, RWE, HomeMatic und KNX einzeln und die wesentliche Unterschiede zusammengefasst betrachtet.

¹Drahtloses, lokales Netzwerk (Wireless, Local Area Network)

ZigBee ist eine Funknetz-Technik, die weltweit mit der Bezeichnung IEEE 802.15.4 standardisiert wurde. Die Standardisierung wurde durch die ZigBee-Allianz entwickelt, welche aus mehr als 250 Unternehmen besteht. Leider lässt die Standardisierung viele Sonderimplementierungen zu, sodass eine Kompatibilität untereinander nicht immer gegeben ist. Die Leistungsaufnahme von ZigBee-Komponenten ist sehr gering, was sehr für den Batteriebetrieb spricht. Ein ZigBee Netzwerk wird in Stern-Topologie um einen ZigBee-Router aufgebaut. Mehrere Router können so ein Netzwerk in Baum-Topologie bilden, wobei ein Koordinator verwendet wird. Fällt ein Router aus, so sind die restlichen Maschen aktiv. Die Schwachstelle ist der Koordinator. Fällt er aus, fällt das ganze Netzwerk aus. Auch vor Hackerangriffe ist ZigBee nicht besonders gut gesichert. Eine Entschlüsselung des ZigBee-Keys gelingt mit entsprechenden Tools sehr schnell. Für Eigenentwicklung bietet ZigBee ein spezielles Funk-Modul RaspBee für den Einplatinencomputer Raspberry Pi. [28] [26]

EnOcean Technologien beruhen darauf, dass für das Senden kurzer Funksignale nur geringe Mengen an Energie erforderlich sind und diese vor Ort aus der Umwelt generiert werden. Dies geschieht entweder durch Sonnenenergie oder bei Bewegungsenergie z.B. bei Betätigen des Lichttasters durch elektrodynamische Energieumwandlung. Die daraus gewonnene Energie reicht aus und ermöglicht batteriefreie und somit wartungsfreie Sender, was durchaus für diese Technologie spricht. Die Erfinder haben ihren Sitz in München und ihren Funk-Standard bereits 2012 patentieren lassen. Sie verwenden in Europa die Frequenz 868MHz und übertragen somit bidirektional mit Rückmeldung. Erweiterte Sicherheitskonzepte wie Verschlüsselung der Daten und Rolling Codes mit wechselnden Schlüsseln bieten einen hohen Sicherheitsstandard. Allerdings wird dies nicht von allen EnOcean-Produkten angewandt. Aktuell stellen bereits mehr als 100 Unternehmen Produkte mit EnOcean-Technologie her wie z.B. Siemens, Zumtobel, Omnio, Wieland, Wago, Electric. Auch Gateways zu den wichtigsten, kabelgebundenen Standards wie KNX, LCN, LON und BACNET werden angeboten, sodass eine Erweiterung von Neubauinstallationen gegeben ist. Die Vorteile dieser Technologie liegen auf der Hand und bilden auch den Abschluss in der höchsten Preisklasse. [29] [26]

RWE und **HomeMatic** wurden von der Firma eQ-3 entwickelt. Als Partner von RWE sind sie auch für deren Funkprotokoll für die RWE Smart Home Geräte verantwortlich. Beide Systeme weisen Ähnlichkeiten auf, sind allerdings nicht miteinander kompatibel. RWE setzt besonderen Wert auf Datensicherheit und Hackerschutz. In allen Geräten kommt das neue erweiterte IPv6 Internet-Protokoll zur Anwendung. RWE wendet sich an den Anwender und legt großen Wert auf einfache Installation, Konfiguration und Bedienung. RWE bietet verhältnismäßig günstige Aktoren und Sensoren. Die Produkte sind nur mit RWE-Produkten kompatibel. Auch Produkte von HomeMatic sind relativ günstig. Die bidirektionale Kommunikation erhöht die Funktionssicherheit. In der Haus- und Gebäudeautomatisierung gibt es eine sehr breite Produktpalette, allerdings wird das Protokoll nur vom eigenen Hersteller

unterstützt. [30] [26]

Z-Wave ist ein drahtloser Kommunikationsstandard der speziell für die Heimautomatisierung entwickelt wurde. Die Z-Wave Alliance, als Zusammenschluss von 160 Herstellern, unterstützt mit ihren Produkten die Z-Wave Technologie. Mit mehr als 600 zertifizierten Produkten ist Z-Wave eines der größten Systeme funkbasierter Produkte. Die Alliance testet Produkte auf Kompatibilität und vergibt demnach Prüfzeichen. So soll die Kompatibilität aller Produkte gewährleistet sein. Die Funkweite im freien Gelände entspricht bis zu 200m, je nach Mauerwerk sollen mindestens 30m erreicht werden. Ähnlich wie ZigBee nutzt Z-Wave ein vermaschtes Netzwerk mit mehreren Netzwerkknoten. Ein Netzwerk kann aus bis zu 232 Geräten bestehen. Knoten mit Netzstromversorgung reichen zudem Informationsdaten von anderen Sensoren weiter. So gelangen auch Funkinformationen von Sensoren die nicht in direkter Erreichbarkeit stehen zum entsprechenden Empfänger. Die Erreichbarkeit der Funkweite wächst somit mit dem System mit. Es wird das Funknetzwerk mit einer Frequenz von 868MHz genutzt, welches Rückmeldungen bzw. Empfangsbestätigungen erlaubt. Die Preise der Aktoren und Sensoren liegen im unteren, mittleren Bereich. [15] [26]

Der Hersteller **KNX** ist bereits seit den 1990er Jahren im Bereich der Gebäudeautomation tätig. Der KNX Feldbus-Standard der Gebäudeautomation, als Weiterentwicklung des Vorgängers **EIB²**, hat einen festen Namen in diesem Bereich und wurde als internationaler Standard anerkannt. Herstellerunabhängig wird der KNX-Standard weltweit von 370 Unternehmen unterstützt. Als große Anbieter zählt in Deutschland auch Siemens, WAGO und ABB zur KNX Germany. Mit der Einführung des Funkstandards KNX-RF können sowohl bestehende KNX Feldbus-Systeme mit Funk-Technologie erweitert, als auch allein betrieben werden. Durch den einheitlichen Standard sind sämtliche KNX-RF Module kompatibel. KNX-RF verwendet die Funkfrequenz von 868MHz mit bidirektionaler Funkübertragung. Durch die maximale Datengröße von 16Byte für eine Datenübertragung kommt es selten zu Datenkollisionen. Für die Signalverarbeitung wird eine Frequenzumtastung (FSK) eingesetzt. Die KNX-RF Technologie weist die größte Anbieter-Unterstützung auf und gibt somit größte Sicherheit für die zukünftige Versorgung von **KNX-RF**-Geräten. Bedingt durch die langjährigen Erfahrungen im Bereich der Hausautomatisierung hat sich die Organisation zum Ziel gesetzt, ihr System als DEN Standard in der Hausautomatisierung zu etablieren. Wie bereits erwähnt sind KNX-Systeme sehr teuer und werden an dieser Stelle auf Grund ihrer Marktstellung lediglich erwähnt. [26]

Das **Auswahlkriterium** für die Einbindung geeigneter, industrieller Funk- Gebäudetechnik besteht in erster Linie durch die Verwendung der Funkfrequenz von 868Mhz. Sie bietet zum einen Frequenzabstand vom überfüllten 433Mhz Netz und unterbindet somit nicht seltene Funkstörungen, zum anderen bietet es die Möglichkeit von Rückmeldungen der

²Europäischer Installationsbus

einzelnen Aktoren und Sensoren, was einen sicheren und vor allem kontrollierbaren Steuerungsbetrieb gewährleisten soll. Nach der Faustregel je tiefer die Übertragungsfrequenz, desto geringer die Dämpfung der Übertragungsleistung, befindet man sich mit 868MHz noch im Radiowellenbereich, der gegenüber dem hochfrequenten W-LAN Netz mit $2,4\text{GHz}$ weniger störanfällig für Gebäudemauern und deren Materialien ist. Auch im medizinischen Bereich, wo Sicherheit und Zuverlässigkeit an oberster Stelle stehen, wird das 868MHz Netz angewandt. Ein weiteres Kriterium ist die industrielle Unterstützung von mehreren Herstellern, was in Hinblick auf die Zukunft ein großes Angebot sichern soll. Zur Umsetzung eines kostengünstigen Systems spielen zu guter Letzt auch die Anschaffungskosten eine wesentliche Rolle.

Die Eigenschaften der einzelnen Funk-Möglichkeiten sind in folgender Tabelle 3.1 zusammengefasst. Unter Berücksichtigung der genannten Kriterien fällt die Auswahl auf ZigBee und Z-Wave. Auch die EnOcean Technologie ist im Bezug auf den Energieverbrauch sehr interessant, fällt allerdings durch die hohen Kosten der einzelnen Produkte raus. ZigBee und Z-Wave werden von einem breiten Herstellermarkt unterstützt. Die Kosten für das Funkmodul für den Raspberry Pi liegen für Z-Wave um einiges höher, trotzdem sprechen die nicht seltenen Kompatibilitätsprobleme von ZigBee Produkten für sich, wodurch die Auswahl auf Z-Wave fällt. Die Nachteile der Funktechnik sind allgemein die Nichtverschlüsselung der Funkdaten, was unerlaubten Zugriff in Reichweite der Funkleistung theoretisch ermöglicht. Da deren Verwendung lediglich für Lampenschaltungen bzw. Übertragung von Messdaten verwendet wird, kann dies vernachlässigt werden. Schaltungsmöglichkeiten von Tür- und Fensterkontakten sollte mit gewählter Funktechnik vermieden werden.

3.1.2 Smart Home - Software

Herzstück des Smart Home Systems bildet ein Raspberry Pi. Durch die große Fangemeinde des Einplatinencomputer gibt es spezielle Software-Architekturen für die Anwendung im Smart Home Bereich. Im Folgenden werden die Programmiermöglichkeiten mit *Python*, *openHAB* und *FHEM* auf deren Eigenschaften untersucht. Des Weiteren wird auch die spezielle Laufzeitumgebung von CoDeSys, die für den Raspberry Pi entwickelt wurde untersucht.

Python ist die Programmiersprache für die der Raspberry Pi ursprünglich entwickelt wurde. Demnach liegt der Gedanken nahe die Smart Home Umgebung mit Python zu realisieren. Python ist eine universelle, höhere Programmiersprache die 1991 durch die Python Software Foundation entwickelt wurde. Die Philosophie dahinter besteht durch gute Programmierbarkeit und der Code fällt im Vergleich zu anderen deutlich kürzer aus. Programmstrukturen werden durch Einrücken gebildet, wodurch die Anwendung von Klammern entfällt. Es unterstützt mehrere Programmierparadigmen wie z.B. objektorientierte, aspektorientierte und

Tabelle 3.1: Funksysteme

	HomeMatic/ Rwe	ZigBee	Z-Wave	EnOcean	KNX-RF
Frequenzen	868MHz (Europa) -bidirektional	433/868MHz (Europa) -bidirektional	868MHz (Europa) -bidirektional	868MHz (Europa) -bidirektional	868MHz (Europa) -bidirektional
Modulation	FSK	BPSK	GFSK	ASK	FSK
Datenrate(kbps)	20	20	9,6/40	125	16,4
Datenkollision	mittel	mittel	mittel	Sehr Gering	Gering
Protokoll Standard	Ja	Ja (unterschiedl. Implementierung möglich)	Ja	Ja	Ja
Kompatibilität	Ja (jeweils)	teilweise	Ja	Ja	Ja
Stromverbrauch	Gering	Gering (Schlafmodus)	gering	Gering bis kein	gering
Topologie	Stern	Stern/Misch mit Koordinator und Router	Stern/Misch	Stern/Misch	Stern
Maximale Knoten	-	2 ¹⁶	232	2 ³²	256
Reichweite in Gebäuden	-	10-100m	20-200m	30-300m	10-100
Sicherheit	AES-128- Verschlüsselung	Keine Verschlüsselung	Keine Verschlüsselung	Spez. Verschlüsselung sowie Rolling Codes	Keine Verschlüsselung
Herstelleranzahl	1, proprietär	>250	>160	>100	>370
Kosten	günstig	mittel	mittel	teuer	Sehr teuer
Raspberry PI Funkmodul	-	RasBee premium mit 200 Knoten (Preis 48€)	Raspberry PI (Preis: 60€)	EnOcean PI (Preis: 36€)	-

funktionale Programmierung.

In Bezug auf den Raspberry PI existieren sehr umfangreiche Bibliotheken, die sämtliche Erweiterungs-Boards des Einplatinencomputer abdecken. Das Einbinden unterschiedlicher Boards bereitet somit keine Probleme.

Für die Visualisierung wird eine GUI³-Erweiterung zur Erstellung einer grafischen Benutzeroberfläche verwendet. Das erste GUI-Toolkit für Python war Tkinter. Mittlerweile gibt es die unterschiedlichsten GUI-Möglichkeiten. Für eine einheitliche Visualisierung auf unterschiedlichen Endgeräten wie Webbrowser, Android- oder iOS- Geräten, muss dabei explizit eine Übersetzung auf HTML5 realisiert werden.

³Grafische Benutzerschnittstelle (Graphical User Interface)

openHAB (open Home Automation Bus) ist eine in Java entwickelte Softwarelösung, die speziell für die Gebäudeautomatisierung entwickelt wurde und unter der Eclipse Public License als Open Source erhältlich ist. Dank Open Source besitzt openHAB eine große und sehr aktive Community. OpenHAB läuft betriebsunabhängig und ermöglicht die Verwendung unterschiedlicher Komponenten wie zum Beispiel KNX, Z-Wave, ZigBee und EnOcean. Die [OSGi⁴](#)-Plattform erlaubt durch ihre modulare Architektur eine flexible Softwareerweiterung für zusätzliche Technologien bzw. Protokolle, mittels entsprechender systemübergreifender Anbindungen, sogenannte Bindings. Der große Nutzen der openHAB-Software besteht darin, unterschiedlichste Technologien der Hausautomatisierung zu einer einheitlichen Bedienoberfläche zu vereinen. [31]

Der Zugriff auf die Visualisierung erfolgt über das Intranet. Mit dem kostenlosen Dienst [my.openHAB](#) sind auch dauerhaft [SSL⁵](#)-gesicherte, web-basierte Zugriffe mit nativen Apps für iOS und Android aus dem Internet möglich. Aufwändigere Lösungen lassen sich auch durch einen eigenen, unabhängigen [DynDNS⁶](#) Service realisieren. [31]

Für die grafische Gestaltung der Oberfläche bietet openHAB den sogenannten openHAB-Designer. Die Devise ist hierbei allerdings "Deklarieren statt Designen". Der Nutzer verfügt nicht über die Freiheit einer pixelgenauen Gestaltung der Bedienoberfläche, sondern kann sich durch Art und Reihenfolge der verwendeten Funktionen aus der spezifischen Bibliothek, baukastenartig eine Bedienung zusammenstellen. Das Ergebnis visualisiert weniger die individuelle Wohnung in Form und Aussehen, sondern unterteilt vielmehr deren Funktionen in unterschiedliche Reiter (Bad, Zimmer, ..., Licht, Temperatur...). Darstellungen für Tablet-PCs und Smartphones können unterschiedlich gestaltet werden. Durch die Verwendung einer Datenbank können Sensordaten archiviert und entsprechend visualisiert werden. In folgender Abbildung ist ein Beispiel in iOS-Optik dargestellt.

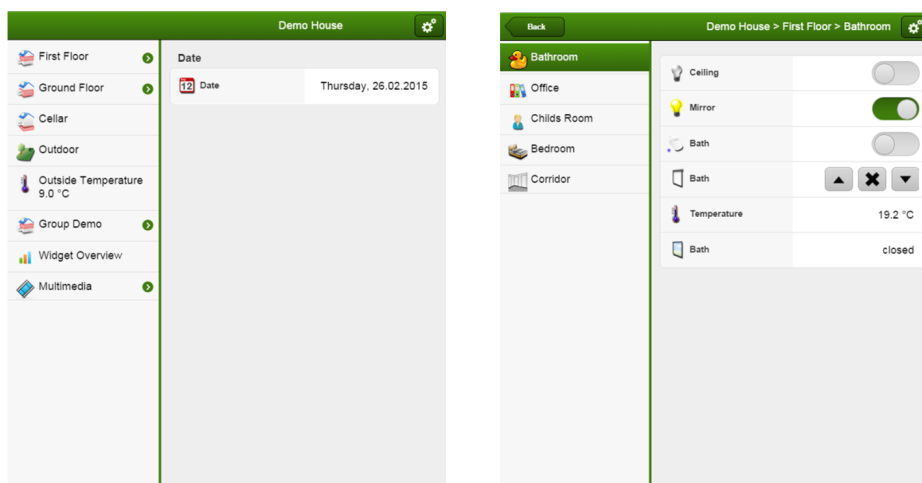


Abbildung 3.1: openHAB – Visuelle iOS Darstellung [31]

⁴Open Services Gateway initiative

⁵Hybrides Verschlüsselungsprotokoll (Secure Socket Layer)

⁶dynamische Aktualisierung der Domäne (Dynamic Domain Name System)

FHEM (Freundliche Hausautomatisierung und Energie-Messung) ist ein Perl-basiertes Serverprogramm für die Hausautomatisierung. Das Serverprogramm wurde unter der GPL als Open Source veröffentlicht und kann über Web, dedizierte Smartphone Apps oder Telnet bedient werden. Das Serverprogramm läuft betriebsunabhängig auf Windows, Linux oder OS X Rechnern aber auch Einplatinencomputern wie Raspberry PI. FHEM unterstützt viele, in der Hausautomation üblichen Protokolle, Fernseh- oder Audiogeräte, Wetterdienste und Online Kalender. Über die FHEM Site findet man Zugang zu einer großen Community die sich bei unterschiedlichsten Projekten gegenseitig unterstützt. Dementsprechend kann auf eine große Bibliothek an Automatisierungs-Lösungen zurückgegriffen werden, die baukastenartig zu einem eigenen Projekt zusammengestellt werden können. Mit den entsprechenden Funkmodulen besteht die Möglichkeit der Anbindung von bspw. Z-Wave, HomeMatic, ZigBee, EnOcean und KNX. [32]

In folgender Abbildung ist die Standard-Darstellung der Visualisierung für Tablet-PC und Smartphone dargestellt. Zum Vergleich zu OpenHAB ist deren Verwendung allerdings nicht zwingend, sondern wird von FHEM lediglich angeboten. Der Gestaltung des Web Frontend sind somit keine Grenzen gesetzt.



Abbildung 3.2: FHEM – Visualisierung [32]

CoDeSys (Controller Development System) wurde 1994 vom Softwarehersteller 3S-Smart Software Solutions aus Kempten entwickelt und ist im Bereich der industriellen Automatisierungstechnik zur Ansteuerung von speicherprogrammierbaren Steuerungen eine feste Größe. Die Entwicklungsumgebung ist kostenfrei auf deren Seite erhältlich. Lizenzkosten werden für die spezielle Anwendung bzw. Runtime-Umgebung der Hardware erhoben. Im

Embedded-Bereich liegen die Preise im zweistelligen Bereich, für den Raspberry Einsatz bspw. bei ca. Euro 40. Die Raspberry Pi Runtime wurde dabei speziell für Test- und Lernzwecke entwickelt.

Die Software deckt unterschiedliche Aspekte der industriellen Automatisierungstechnik in einer Oberfläche ab, wo sie hauptsächlich Ihre Verwendung hat. Alle fünf von der IEC61131-3 (International Electrotechnical Commission) spezifizierten Sprachen (AWL, ST, KOP, FBS, AS) stehen in CoDeSys zur Verfügung. Mit der Kompilierung wird der Applikationscode in einen Maschinencode übersetzt und kann auf die entsprechende Hardware geladen werden. In CoDeSys können unterschiedliche Feldbusse verwendet werden. Das Tool integriert dazu Konfigurationen für die wichtigsten Systeme wie z.B. Profibus, CANopen, EtherCAT, Profinet, Ethernet IP. Für die Verwendung mit dem Raspberry Pi bietet CoDeSys (seit 19.12.2013) eine spezielle Runtime "CoDeSys Control for Raspberry Pi SL". In der üblichen Baumstruktur von CoDeSys kann dabei auf die Architektur bzw. Ein-/Ausgänge des Raspberry Boards zugegriffen werden. In folgender Abbildung 3.3 ist die Entwicklungsumgebung mit Raspberry Pi als Geräteauswahl abgebildet. [19]

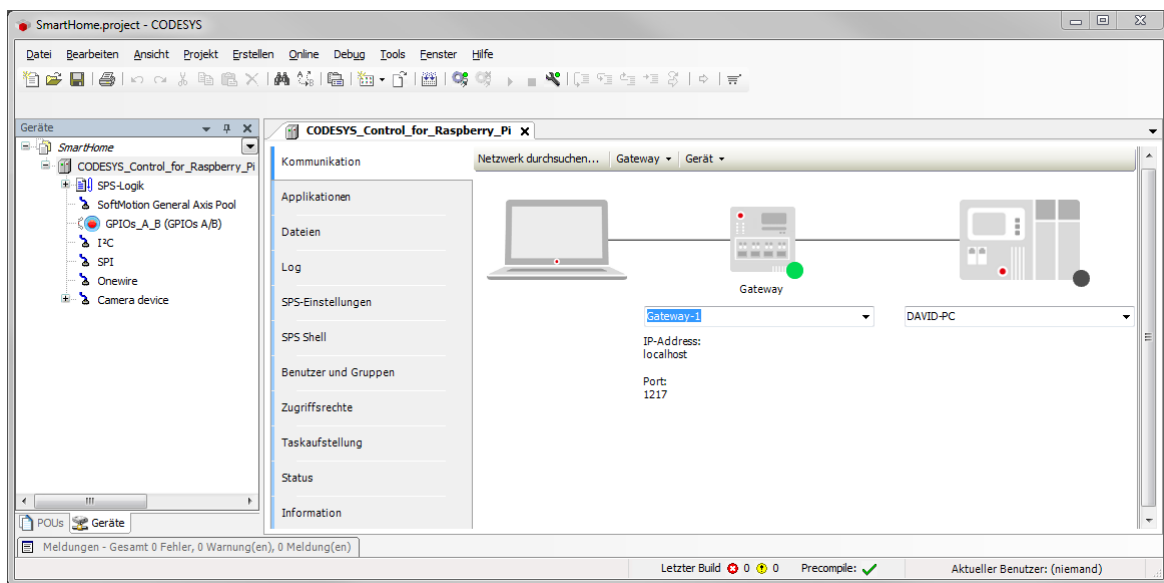


Abbildung 3.3: CoDeSys – Entwicklungssystem

In der Baumstruktur der CoDeSys Darstellung sind die einzelnen Raspberry Schnittstellen (GPIO, Camera Device, I²C, SPI, Onewire) dargestellt. Durch die integrierte WebVisu Funktion fungiert der Raspberry parallel als Webserver und ermöglicht den Zugriff sowohl im Intranet als auch aus dem Internet, dessen Zugriff durch eine sichere SSL-Verbindung abgesichert ist. Für eine einheitliche Visualisierungsdarstellung für Android- sowie iOS-Endgeräte

sorgt der Übersetzungsstandard auf [HTML⁷](#)5.

Eine weitere Stärke von CoDeSys V3 ist die programmiertechnische Organisation mit [OOP⁸](#). Für die Gebäudeautomatisierung ermöglicht dies die einzelnen Raumfunktionen in einmalige Klassen zu organisieren und deren Objekte individuell auf die entsprechende Wohnung anzuwenden. Ein Nachteil liegt in der Visualisierungsbibliothek im Bereich der Hausautomatisierung. Die Ausrichtung der Visualisierungs-Elemente der Standardbibliothek liegt ganz klar im technischen Anlagenbereich. Die Anzahl von hochwertigen Mess- und Steuer- Armaturen überwiegt ganz klar den eher einfach gehaltenen Basic-Elementen, wie beispielsweise Lampen und Schalter.

Da die CoDeSys Raspberry Ergänzung erst knapp über einem Jahr auf dem Markt ist, ist die Bibliothek der Gerätetreiber speziell für Peripheriegeräte des I²C- und SPI-Bus sehr überschaubar. In der folgenden Abbildung sind 2 Visualisierungsbeispiele von CoDeSys für die Verwendung des Raspberry PI dargestellt. Die Visualisierungsstärke im technischen Bereich ist eindeutig erkennbar.

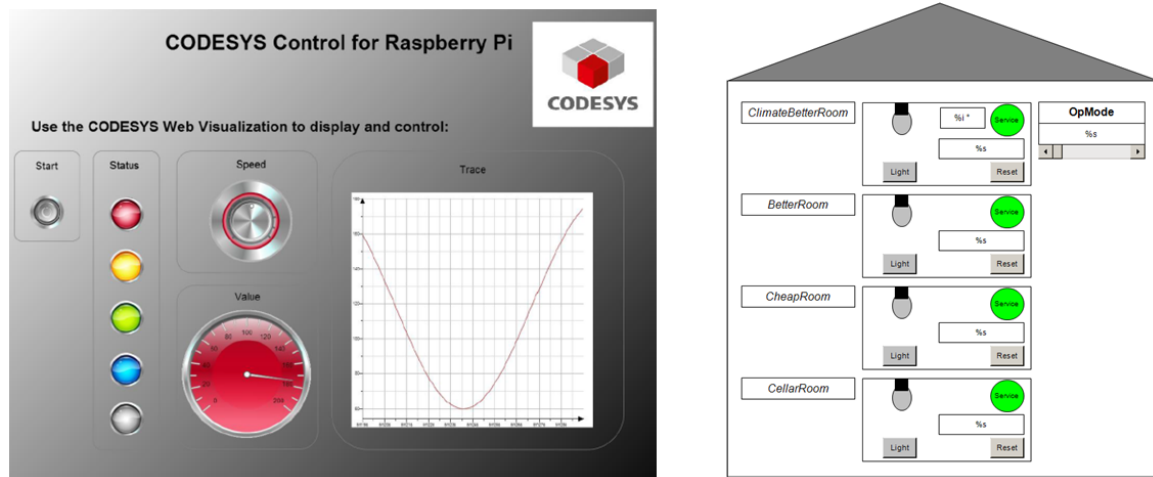


Abbildung 3.4: CoDeSys – Visualisierungsbeispiel [19]

Auswahl der geeigneten Programmier-Oberfläche

Für die Entscheidung über die geeignete Oberfläche wird zunächst der Anspruch an die Visualisierung erläutert. Neben einer kostengünstigen Smart Home Entwicklung auf Basis von Raspberry und Arduino steht eine selbsterklärende, anwendungsfreundliche Bedienung der Hausautomation. Ziel der Arbeit ist ein robustes Endprodukt mit einfacher Konfiguration und intuitiver Visualisierungssteuerung als Verbindung von Anlage und Bewohner, welches durchaus mit marktüblichen Lösungen konkurrieren kann.

⁷Hypertext Auszeichnungssprache (Hypertext Markup Language)

⁸Objektorientierte Programmierung

Eine Steuerung von einzelnen Aktoren über ein verschachteltes Register auf dem Smartphone ist aus technischer Sicht durchaus interessant. Im praktischen Leben ist dabei der spielerische Aspekt wohl um einiges wichtiger als der effektive Nutzen im täglichen Leben. Beim Betreten eines Raumes wird die übliche Schalterbetätigung wohl bevorzugt werden. Der große Nutzen einer Smart Home Umgebung liegt mehr in der Visualisierung von Energieverbrauch-Verläufen oder Temperaturbewegungen und deren Einfluss auf die Heizungsregelung. Bewohner können die Benutzung der Wohnung laufend über die Visualisierung synchronisieren oder Heizvarianten über Wochenpläne konfigurieren. Der Fokus liegt ganz klar auf dem Einsparen von Energiekosten. Aktive Räume oder Belichtungen sollen dabei auf einem Blick in der Visualisierung erkennbar sein. (Optional auch Zustände von Fenstern, Türen und Rollos.)

Ziel ist die Realisierung einer kompletten Wohnungsvisualisierung in der Größe eines Tablet-PCs, welche ausschließlich im Intranet betrieben wird. Zur Visualisierung wird eine 3-Zimmer-Wohnung mit Grundriss und Funktionen vordefiniert. Schaltbare Gegenstände wie Deckenlampen sollen dabei intuitiv durch Berührung schaltbar sein. Archivierte Sensordaten können über ein Menü aufgerufen werden. Temperatur-Sollwerte werden über einen Wochenkalender organisiert. Aktuelle Temperatur- und Luftfeuchtigkeits-Werte der Räume werden entsprechend visualisiert. Bewegungen in Räumen sollen erkannt und für ein definiertes Intervall signalisiert werden. Unterschiedliche Modi (Hand, Sleep, Secure) können konfiguriert und angewählt werden.

In folgender Tabelle sind die Eigenschaften der einzelnen Programmiersprachen bzw. Oberflächen zusammengefasst:

Tabelle 3.2: Softwarevergleich

	Phyton	OpenHAB	FHEM	CoDeSys
Lizenz	OpenSource	OpenSource	OpenSource	Lizenziert (40€)
Biliothek Hardware	Sehr groß	Ja	Ja	Teilweise
Z-Wave Unterstützung	-	Ja	Ja	Eigenentwicklung nötig
Visu-Bibliothek (Smart Home)	-	Ja	Ja	Beschränkt
Freiheit in der Visualisierung	Ja (Toolkit-Erweiterung)	Nein (Registercharakter)	Ja	Ja (mit einfache Designwerkzeug)
Freiheit in Strukturierung	Ja	Nein	Teilweise	Ja
OOP	Ja	Ja (Baukasten)	Ja (Baukasten)	Ja
WebVisu für Android, iOS	-	Ja	Ja (extern)	Ja

Zusammengefasst lässt sich Python schwer mit den restlichen Programmier-Oberflächen vergleichen, da es sich um eine reine Programmiersprache handelt. Die Stärke liegt ganz klar in der umfangreichen Bibliothek. Für die Realisierung der Hausautomation wird allerdings eine funktionale Programmieroberfläche mit integrierter Visualisierungsmöglichkeit, sowie Webvisualisierung mit HTML5-Protokoll bevorzugt. Nach diesem Kriterium und der Freiheit der Visualisierungsgestaltung stehen lediglich FHEM und CoDeSys zur Auswahl. Der erste, direkte Vergleich von FHEM und CoDeSys spricht eindeutig für FHEM. Als Ingenieur der Automatisierungstechnik, insbesondere im Rahmen dieser Masterarbeit, fällt die Entscheidung jedoch auf CoDeSys. CoDeSys ist seit mehr als 10 Jahren fester Bestandteil von automatisierungstechnischen Lösungen im professionellen Anlagenbereich. Im Bezug auf die Verwendung mit Raspberry stehen durchaus offene Problemstellungen im Raum. Auch im Bereich der Visualisierung sind Standardbibliotheken kaum anwendbar, was das Erstellen einer eigenen Bibliothek erfordert. Das Verwenden bzw. Einbinden von industriellen Schalteinheiten über Ethernet kann als weitere Option einfach realisiert werden. Die große Stärke liegt eindeutig in der freien Programmorganisation. Durch eine objektorientierte Struktur sollen zukünftige Erweiterungen oder Änderungen vereinfacht werden. Es bietet die Möglichkeit einen eigenen Baukasten zur Hausautomation zu entwickeln, welcher sich auf individuelle Wohnungsgegebenheiten anpassen lässt.

3.2 Smart Home Konzept

Herzstück der Anlage bildet die *Smart Home Zentrale* auf Basis eines Raspberry Pi. Sie ist zum einen für die Auswertung und Ansteuerung sämtlicher Smart Home Teilnehmer, zum anderen zur Bereitstellung der Smart Home Visualisierung zuständig. Je nach Wohnungsgröße lässt sich die Smart Home Zentrale um maximal 8 Steuerungsmodule, sogenannte *Funk IO-Module*, erweitern. Da die Z-Wave Technologie nur als zentrales Steuermodul zur Ansteuerung von Z-Wave Geräten nutzbar ist, wird ein weiterer Standard-Funk mit einer Frequenz von 868MHz für eigene Entwicklungen eingesetzt. Die erweiterbaren Funk IO-Module sollen die Möglichkeit schaffen, räumlich getrennte Abzweigungen der Elektroinstallation ohne Leitungsverlegung in das Smart Home System zu integrieren. Als weiteres Standard-Funk-Modul wird ein Funk-Sensor entwickelt, der im Intervall Messdaten zu Temperatur, Luftfeuchtigkeit und Helligkeit bereitstellt. Mindestens 8 Funk Sensoren sollen modular erweiterbar sein. Als Ergebnis der Marktanalyse soll auch ein Zugang für industrielle Geräte der Hausautomatisierung mit Z-Wave Funk geschaffen werden. Zur Bereitstellung des Z-Wave Funkes wird ein weiteres Modul auf Basis eines Raspberry Pi als Z-Wave Server eingesetzt.

Sämtliche modulare System-Erweiterungen werden über die Smart Home Zentrale organisiert. Für die softwaretechnische Umsetzung der Zentrale wird die Entwicklungsum-

gebung von CoDeSys eingesetzt. Als Wohnungsbeispiel wird eine 3-Zimmerwohnung mit den Räumen Wohnzimmer, Schlafzimmer, Büro, Bad, Küche und Flur verwendet. Eine objektorientierte Software-Strukturierung soll zukünftige Erweiterungen bzw. individuelle Wohnungsanpassungen erleichtern. Die verwendeten Aus-/Eingänge der Module bspw. zur Ansteuerung von Lampen oder Rollos sollen vom Anwender über die Visualisierung konfigurierbar sein.

Des Weiteren soll das Smart Home System durch vier Betriebsmodi charakterisiert sein. Im Grund-Modus *HAND* können sämtliche Ansteuerungen über die Visualisierung bzw. konfigurierten Eingänge erfolgen. Durch die Anwahl des *SLEEP*-Modus werden Raumfunktionen in einem vom Anwender definierten Zustand angesteuert. Der Modus *SECURE* beschreibt einen definierten Wohnungszustand bei Abwesenheit des Bewohners und löst bei Verletzung der Konfiguration einen Alarm aus. Der Modus *SMART* soll eine Simulation einer belebten Wohnung ermöglichen. Durch eine individuelle Konfiguration von bestimmten Zeitbereichen werden Licht- und Rollosteuern zufällig angesteuert, und sorgen für ein wechselndes Tagesprogramm.

Die Sensordaten werden in einem bestimmten Zeitintervall aktualisiert und stehen durch eine Archivierung der Daten auf der Zentrale zur Visualisierung bereit.

In folgender Abbildung 3.6 sind die einzelnen Module und deren Kommunikationen abgebildet. Die einzelnen Module werden folgend im Einzelnen spezifiziert.

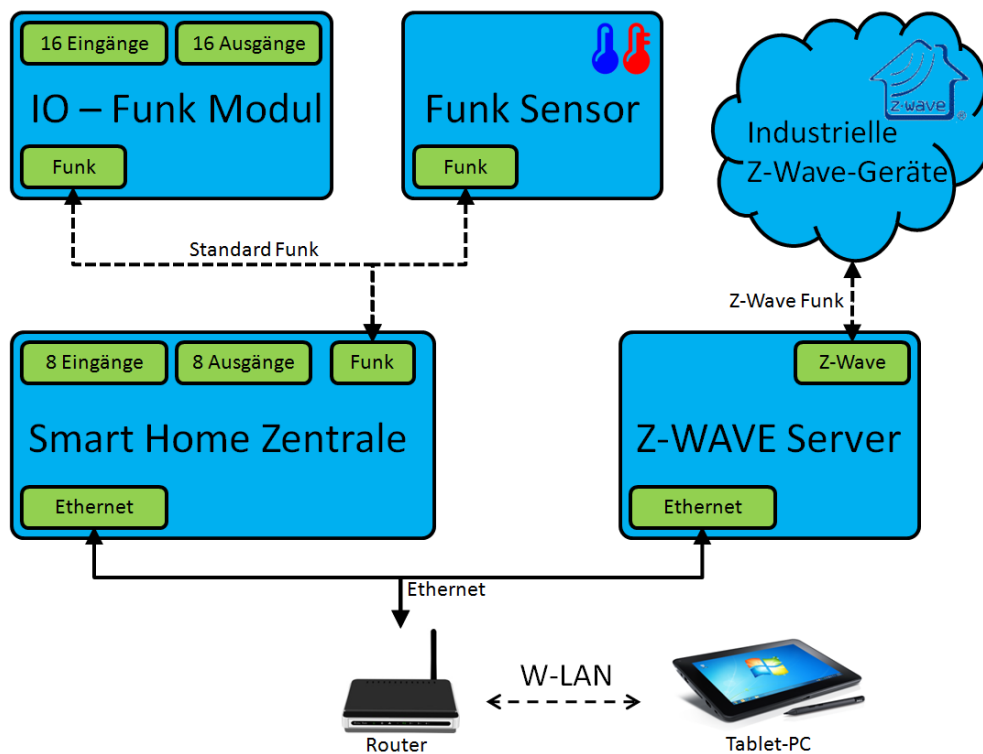


Abbildung 3.5: Smart Home Konzept

Smart Home Zentrale

Zur Verwendung des Raspberry Pi als Smart Home Zentrale wird eine Platine als GPIO-Aufsteck-Modul entwickelt. Die lizenzfreie Version der Platinenlayout-Software EAGLE bietet einen maximalen Layout-Entwurf von $80 \times 100 \text{ mm}$. In Betrachtung der Layout-Größe soll die Platine über Schraubanschlussbuchsen für 8 digitale Eingänge sowie 8 digitale Ausgänge verfügen. Für die Erweiterbarkeit der Anschlüsse sorgt das Funk IO-Modul.

Da die Eingänge des Raspberry Pi eine maximale Spannung von 3.3 V zulassen, wird eine Spannungsanpassung entworfen, die Eingangsspannungen von $5 - 24 \text{ V}$ zulassen. Die Eingänge bieten Anschluss für Taster der Hauselektronik, welche unter Berücksichtigung des Spannungsfalls auch mit externen Spannungsquellen versorgt werden können. Die mit maximal 30 mA belastbaren Ausgänge des Raspberry werden verstärkt und sollen die Ansteuerung von 12 V -Relais zur Schaltung von Schaltkreisen mit 230 V ermöglichen. Die Raspberry-Ausgänge müssen dabei vor induktiven Relay-Rückströmen geschützt werden.

Zur autonomen Spannungsversorgung des Raspberry sowie der Aufsteck-Platine mit sämtlichen Bauteilen wird ein Spannungs-Netzteil mit 12 V eingesetzt. Ein Spannungs-Wandler sorgt dabei für eine geregelte 5 V -Spannungsversorgung des Raspberry Pi. Eventuelle Bauteile mit 3.3 V können direkt über dem geregelten Spannungsausgang des Raspberry betrieben werden.

Die Smart Home Zentrale soll über einen Standard Funk zur Kommunikation mit weiteren Smart Home Entwicklungen verfügen. Als Funkmodul wird das kostengünstige RFM12B-Modul mit einer Übertragungsfrequenz von 868 MHz gewählt. Es ist ein Transceiver-Modul mit SPI-Schnittstelle und kann zum Empfangen und Senden von Funkpaketen eingesetzt werden. Zur Ansteuerungen des Funkmoduls soll ein Mikrocontroller Arduino eingesetzt werden, der somit als Funk-Controller fungiert und ebenfalls auf der Platine integriert wird. Die ausgewerteten Empfangsdaten werden über I²C-Bus an den Raspberry übertragen. Die Kommunikationsschnittstellen der Smart Home Zentrale sind in Abbildung 3.6 dargestellt.

Optional verfügt die Platine über einen externen I²C-Zugang mittels einer RJ45-Buchse. Der Anschluss soll Erweiterungen mit I²C Kommunikation ermöglichen. Eine Zusammenfassung der Hardwarespezifikation folgt in Tabelle 3.3.

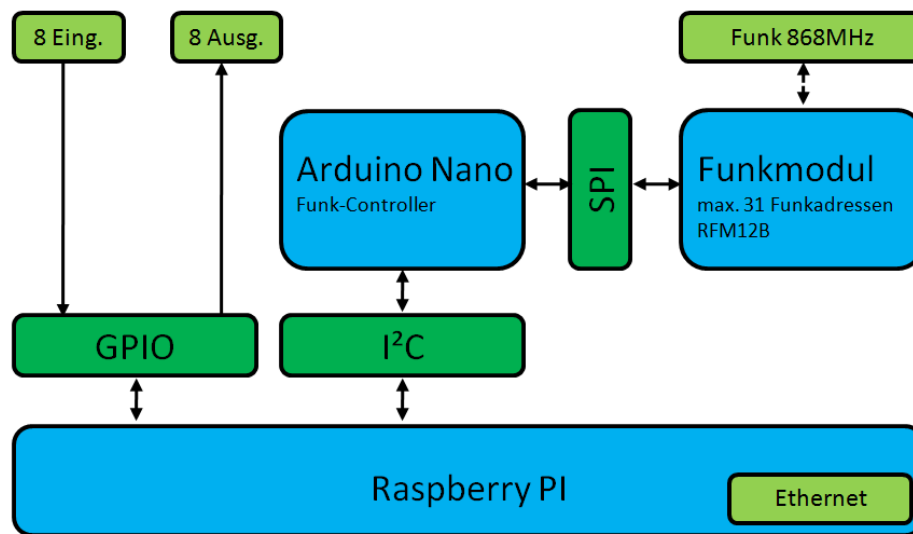


Abbildung 3.6: Konzept – Smart Home Zentrale

Tabelle 3.3: Hardwarespezifikation Zentrale

Hardwarespezifikation	Smart Home Zentrale
Hardware	Raspberry Pi
Platinengröße	80x100mm
Spannungsversorgung V_{IN}	12V DC
Platinen Schnittstelle für Raspberry (GPIO)	2x20 Pin-Stecker
Digitale Eingänge (DI)	8 Schraubanschlussbuchsen
Eingangsspannung (DI)	5 - 24V DC
Digitale Ausgänge (DO)	8 Schraubanschlussbuchsen
Ausgangsspannung (DO)	12V DC
Spannungsausgänge	5V/12V DC
Ethernet Anschluss	In Raspberry integriert
Funkmodul	RFM12B Transceiver (868MHz)
Optionaler I²C-Anschluss (Extern)	RJ45-Buchse

Funk IO-Modul

Das IO-Funk Modul dient als modulare Erweiterung der Smart Home Zentrale. Als Herzstück des Moduls soll der Mikrocontroller Arduino UNO eingesetzt werden. Zur Verwendung als IO-Modul soll eine passende Aufsteck-Platine entwickelt werden. Die Platine verfügt über jeweils 16 Schraubanschlussklemmen für digitale Eingangssignale, sowie digitale Ausgangssignale. Da der Arduino UNO lediglich mit 14 Ein-/Ausgängen ausgestattet ist, soll eine Porterweiterung mittels des I²C-Bausteins MCP23017 erreicht werden. Entsprechend der Spezifikation der Zentrale sollen tolerante Spannungseingänge mit Spannungen von 5 – 24V realisiert werden. Eine Verstärkung der Ausgänge ermöglicht die Ansteuerung von 12V-Relais zur Schaltung von Schaltkreisen mit 230V. Zur Datenübertragung wird

das bereits erwähnte Funkmodul RFM12B eingesetzt. Eingangssignale sollen an die Smart Home Zentrale gesendet werden. Je nach Konfiguration des Smart Home Systems werden die Ausgänge des IO-Funk Moduls entsprechend der Empfangsinformation gesetzt.

Das Funk IO-Modul wird über die Platine mit einer Spannungsversorgung von 12V gespeist. Der Arduino lässt eine alternative Spannungsversorgung von 5 – 12V über den Port-Pin V_{IN} zu. Zur Spannungsversorgung der Platinen-Bauteile stehen die geregelten Spannungsausgänge (3.3V und 5V) des Arduinos zur Verfügung. In folgender Abbildung 3.7 sind die Kommunikationsschnittstellen der Steuerungsbauteile des Funk IO-Moduls dargestellt. Tabelle 3.4 liefert eine Zusammenfassung der Hardwarespezifikation des Funk IO-Moduls.

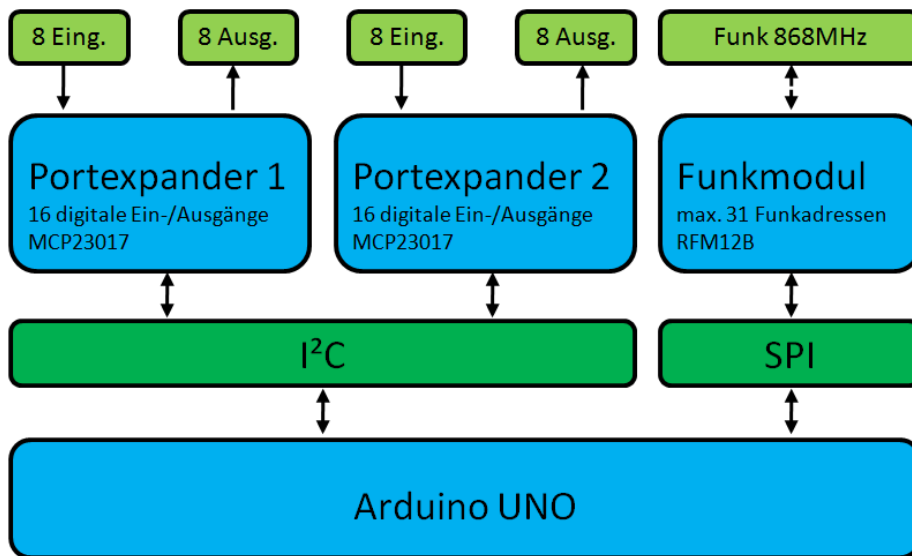


Abbildung 3.7: Konzept – Funk IO-Modul

Tabelle 3.4: Hardwarespezifikation Funk IO-Modul

Hardwarespezifikation	Funk IO-Modul
Hardware	Arduino UNO
Platinengröße	80x100mm
Spannungsversorgung V_{IN}	12V DC
Digitale Eingänge (DI)	16 Schraubanschlussbuchsen
Eingangsspannung (DI)	5 - 24V DC
Digitale Ausgänge (DO)	16 Schraubanschlussbuchsen
Ausgangsspannung (DO)	12V DC
Spannungsausgang	12V DC
Funkmodul	RFM12B Transceiver (868MHz)

Funk Sensor-Modul

Es soll ein Sensor-Modul zur Erfassung von Temperatur und Luftfeuchtigkeit sowie optional Helligkeit entwickelt werden. Das Modul soll mit handelsüblichen Batterien betrieben werden und eine minimale Lebensdauer von einem Jahr aufweisen. Als Herzstück des Sensor-Moduls soll der stromsparende Mikrocontroller Arduino Mini Pro eingesetzt werden. Für eine kostengünstige Umsetzung wird der digitale Sensor DHT22 eingesetzt, der die Messung von Temperatur und Luftfeuchtigkeit ermöglicht. Das DHT22-Modul soll über den Eingang mit Pulsweitenmodulation (PWM) des Arduino ausgelesen werden. Zur Messung der Helligkeit soll eine einfache Photodiode dienen. Für die optionale Photodiode kann der analoge Eingang des Arduinos genutzt werden. Die minimale Lebensdauer soll durch entsprechende Intervall-Messung, sowie Schlafmodus-Betrieb des Mikrocontrollers erreicht werden. In folgender Abbildung 3.8 sind die Kommunikations-Schnittstellen des Sensor-Moduls abgebildet.

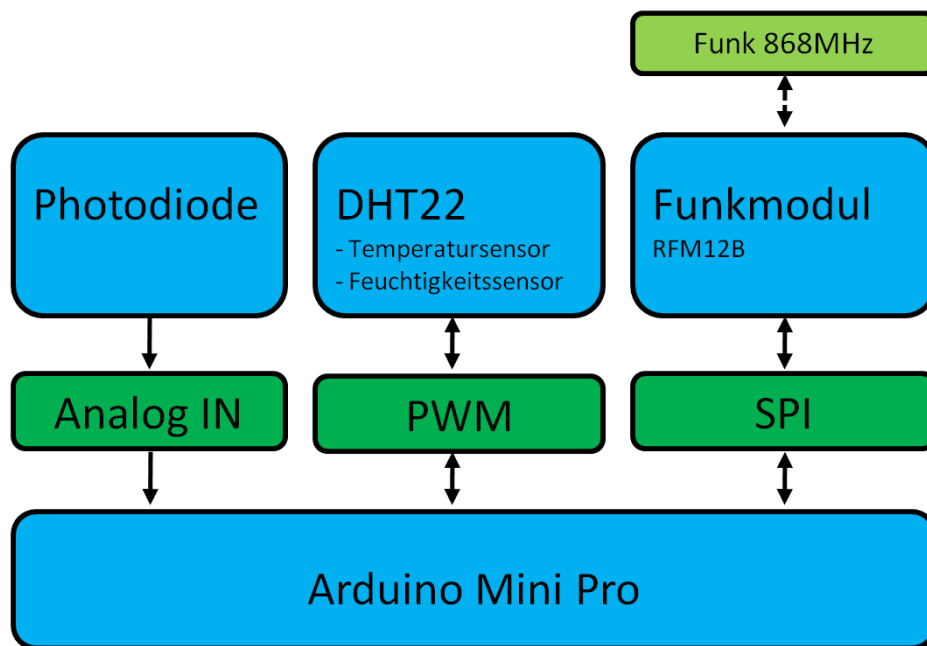


Abbildung 3.8: Konzept – Funk Sensor-Modul

Tabelle 3.5: Hardwarespezifikation Funk Sensor-Modul

Hardwarespezifikation	Funk Sensor-Modul
Hardware	Arduino Micro Pro
Platinengröße	40x50mm
Spannungversorgung V_{IN}	Batterie (4x1.5V)
Temperatursensor	DHT22 (integriert)
Luftfeuchtigkeitssensor	DHT22 (integriert)
Helligkeitssensor (optional)	Photodiode
Funkmodul	RFM12B Transceiver (868MHz)

Z-Wave Umgebung

Wie bereits unter den technischen Grundlagen vorgestellt (2.5), verwandelt das Z-Wave-Modul *Razberry* den Raspberry Pi in ein Z-Wave Gateway. Mit dem Einsatz der Z-Wave Software Z-Way bietet der Z-Wave Server eine sogenannte **JSON**-Schnittstelle zur externen Steuerung des Z-Wave Netzwerkes über Ethernet. Die Ansteuerung der Z-Wave Geräte soll dadurch über die Smart Home Zentrale ermöglicht werden. Die Steuerzugriffe sind durch eine Vielzahl von unterschiedlichen Geräteklassen, welche zudem unterschiedliche Kommandoklassen aufweisen, definiert. Da die Umsetzung sämtlicher Geräteklassen den Rahmen der Arbeit sprengen würde, werden lediglich drei Z-Wave Geräte zur Ansteuerung umgesetzt. Es sollen ein Taster, eine schaltbare Steckdose sowie ein Multisensor mit Z-Wave-Technologie eingesetzt werden.



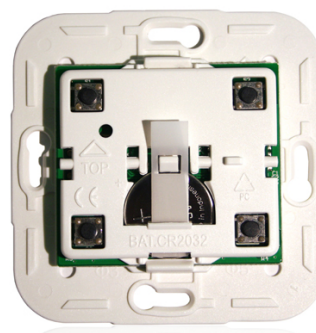
(a) RaZBerry Modul



(b) Multisensor



(c) Steckdose



(d) Taster

Abbildung 3.9: Z-Wave Geräte

4 Entwicklung und Umsetzung

4.1 Smart Home Zentrale

4.1.1 Hardware

Das Herzstück der Smart Home Zentrale bildet der bereits unter Punkt 2.1 vorgestellte Einplatinenrechner Raspberry Pi in der Modellversion B+ mit 700MHz und 500MB RAM. Der On-Board Ethernet-Anschluss bietet bereits den Zugang zum Intranet bzw. Internet. Als Speichermedium wird eine Mikro-SD-Karte mit 8GB verwendet. Für die GPIO-Schnittstelle wird nun im folgenden Verlauf eine Aufsteck-Platine entworfen, die die Anforderungen der Spezifikation erfüllt. Im Rahmen der lizenzfreien EAGLE-Version zur Erstellung von Platinenlayouts, soll die Platine eine Größe von $80 \times 100\text{mm}$ nicht übersteigen. Sämtliche vorgestellte Hardware-Bauteile werden in einem ersten Testaufbau auf einfachen Steckbrettern mit losen Verbindungsbrücken auf deren Funktion getestet.

4.1.1.1 Spannungsversorgung

Das originale Netzteil des Raspberry Pi liefert nur 1A bei 5V -Spannungsversorgung. Die Stromaufnahme des RasPi entspricht dabei im Normalbetrieb 0.7A (bei nicht Beschaltung der GPIOs). Da 300mA nicht für sämtliche Anbauteile der Platine reichen, wird die Spannungsversorgung bzw. deren Anschluss über die Platine realisiert. Als Netzteil soll künftig ein $12\text{V}/\text{DC}$ Netzteil mit 2.5A dienen, welches die Smart Home Zentrale über die passende $5,5 \times 2,1$ Steckerbuchse versorgt.

Zur Spannungswandlung auf 5V wird ein kostengünstiges DC-DC-Abwärtsmodul (Abb.4.1) mit integriertem LM2596 Step-Down-Wandler und einstellbarer Ausgangsspannung verwendet. Mit maximaler Eingangsspannung von 40V , lässt sich die Ausgangsspannung über einen Präzisionspotentiometer von 1.2V bis 37V einstellen. Die maximale Belastung liegt dabei bei 3A . [17]



Abbildung 4.1: Spannungsregler-Modul

4.1.1.2 Tolerante Eingänge

Die Smart Home Zentrale soll über 8 tolerante Eingänge mit einem Spannungsbereich von 5V bis 24V verfügen. Die digitalen Eingänge des RasPi erlauben maximal 3.3V, wobei eine Eingangsspannung größer 2V als HIGH-Pegel bzw. kleiner 0.8V als LOW-Pegel erkannt wird. Als Eingangspegelwandlung wird eine einfache Zener-Diode mit Widerstand verwendet. Die Diode verfügt über eine Durchbruchspannung von 3.3V. Das PSpice-Model zur Ermittlung des passenden Widerstandswertes ist in folgender Abbildung abgebildet. Der Widerstand R1 ist für das dynamische Verhalten, des Erreichens der Durchbruchspannung verantwortlich. Der Widerstand R2 fungiert als Pull-Down-Widerstand und sorgt für einen definiertem LOW-Pegel; R2 wird eher hochohmig mit $47k\Omega$ belegt.

Die Simulation aus Abbildung 4.3 zeigt das beste Ergebnis mit einem guten Kompromiss aus Dynamik und Verlustleistung mit einem Widerstandswert R1 von $2.2k\Omega$. Mit einer Eingangsspannung von 5V wird bereits eine Ausgangsspannung von 2.98V und somit definierten HIGH-Pegel erreicht. Die maximale Verlustleistung bei einer Eingangsspannung von 24V beträgt $196mW$.

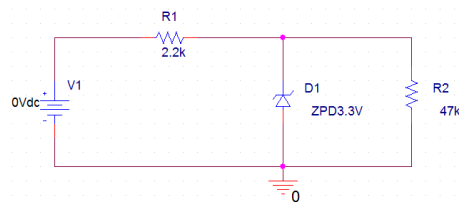


Abbildung 4.2: PSpice Model mit Zener-Diode ZPD3.3V

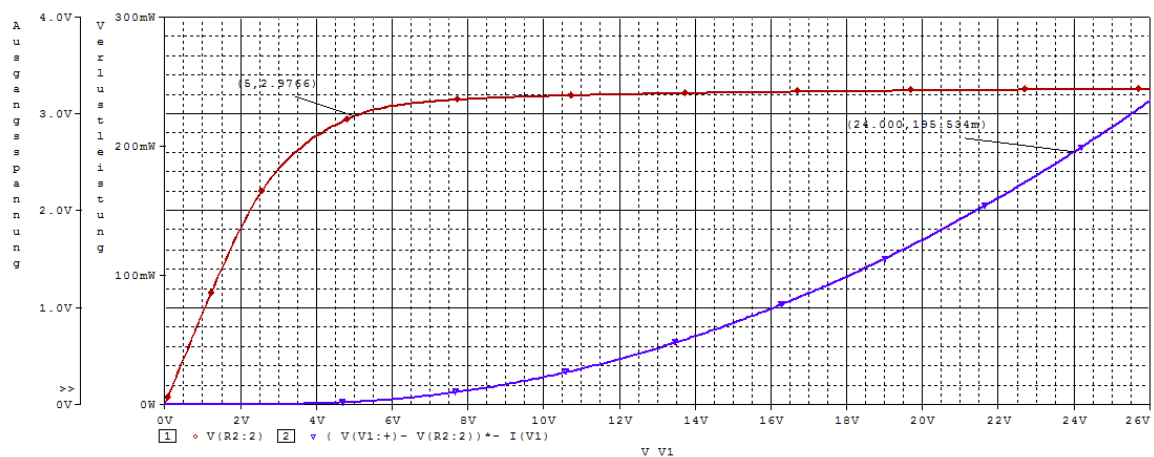


Abbildung 4.3: PSpice Simulation – Pegelwandlung 3.3V

4.1.1.3 Ausgänge

In der Spezifikation wurden 8 digitale Ausgänge zur Ansteuerung von Relais definiert. Die GPIO-Ausgänge des RasPi lassen pro Ausgang einen maximalen Strom von 16mA bei einer Ausgangsspannung von 3.3V zu. Um größere Lasten schalten zu können wird der ULN2803A, ein NPN Darlington-Array, bestehend aus 8 Darlington Schaltungen verwendet. Ein großer Vorteil des IC-Bausteins ist neben seiner kleinen Bauweise die integrierte Freilaufdiode für induktive Lasten. Bei direkter Beschaltung des RasPi's mit Relais könnten die Transistoren des RasPi aufgrund der Gegeninduktionsspannung der Relais-Spule zerstört werden. Jeder Ausgang des Darlington-Array kann bei einer maximalen Ausgangsspannung von 50V maximal mit 500mA belastet werden. In folgender Abbildung 4.4 ist der Schaltungsaufbau des ULN2803A abgebildet. Die Eingänge 1B bis 8B können maximal mit 30V beschaltet werden. Liegt am Eingang eine logische Eins vor, so schaltet, bedingt durch die Darlington-Schaltung, der Ausgang auf Masse. Werden die Relais mit externer Spannung versorgt sind die Massen zu verbinden. [16]

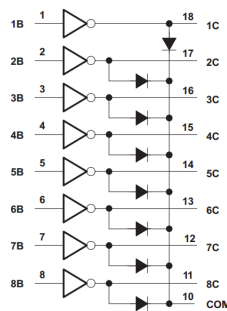


Abbildung 4.4: Darlington-Array-Schaltung [16]

4.1.1.4 Optionaler I²C-Anschluss

Der I²C-Bus ist grundsätzlich nur für Verbindungen kleiner 2m einsetzbar und somit für interne Peripherie-Bausteine gedacht. Durch entsprechende I²C-Expander wie bspw. der P82B96 von NXP Semiconductors sind durch Signal-Verstärkung auch Entfernungen von bis zu 100m erreichbar. Der I²C-Bus könnte somit als Hausbus eingesetzt werden. Jeder Slave-Teilnehmer muss dabei besagten Expander zur Rücktransformation integrieren. Der Hausbus kann bspw. für Verbindungen von Stockwerken eingesetzt werden. Die Entwicklung des verstärkten I²C-Anschlusses wurde in dieser Arbeit nicht vollständig beendet. Die Umsetzung wird aus diesem Grund nicht weiter ausgeführt. Der I²C-Zugang wird im Platinenlayout lediglich für zukünftige Weiterentwicklungen berücksichtigt.

4.1.1.5 Arduino Funk-Transceiver

Zur Funkübertragung wurde sowohl ein Z-Wave-Funk für industrielle Funkgeräte als auch ein Standardfunk für eigene Entwicklungen spezifiziert. Als Standard-Funkmodul wird das kostengünstige Sende- und Empfangsmodul RFM12B in der Version 868MHz von HopeRF eingesetzt. Es arbeitet mit FSK¹-Modulation im 868MHz-Band mit einer Betriebsspannung von 2.2 bis 3.8V. Im Standby-Modus betrifft die Stromaufnahme weniger als 0.3A, was insbesondere batteriebetriebenen Entwicklungen zugute kommt. Zugriff auf das Funkmodul erfolgt über die integrierte SPI-Schnittstelle. Das Modul verfügt über einen 16Bit-FIFO-Empfangspuffer. Der Empfang von Daten wird über das Interrupt-Request-Signal (nIRQ) signalisiert. Die Ansteuerung über einen Mikrocontroller ist somit geradezu prädestiniert. [18]

Zur Ansteuerung des Funkmoduls wird ein Arduino Nano eingesetzt. Er ist sehr kompakt in seiner Bauweise und soll als Funk-Controller des RFM12B-Moduls auf der Platine Platz finden. Der Arduino fungiert dabei zum einen als SPI-Master für die Kommunikation mit dem Funkmodul, zum anderen als I2C-Slave für die Übertragung der Daten zum Raspberry Pi. In folgender Abbildung (4.5) ist die Verdrahtung des Funkmodules abgebildet. Die Arduino Darstellung wird zur besseren Übersicht auf dessen relevante Ein-/Ausgänge für die Ansteuerung des Funkmodules bzw. RasPi's begrenzt. Die Schaltung wurde mit EAGLE erstellt. Die fehlenden Bibliothek-Bauteile wie Arduino Nano und RFM12B-Modul werden dabei neu angelegt.

Über GND und VIN wird der Arduino mit 5V gespeist. Dessen 3.3V Ausgang wird zur Speisung des Funkmodules genutzt. Da Beide mit unterschiedlicher SPI-Spannungs-Logik arbeiten wird ein einfacher Spannungsteiler zur Pegelanpassung verwendet ($R1=R2=R3=4.7k\Omega$, $R4=R5=R6=10k\Omega$). Da das nIRQ-Signal mit negativer Logik arbeitet, sorgt der Pull-Up-Widerstand R10 für einen definierten High-Zustand. Das FFS-Signal (FIFO-Select) wird nicht genutzt und zur Störunterdrückung mit den Widerstand R9= $10k\Omega$ auf Spannung gelegt. Eventueller Spannungsschwankungen am Eingang VDD wird mit einem Kondensator entgegengewirkt. Die Anschlüsse A4 und A5 stehen für den I²C-Bus frei. Als Funkantenne (Anschluss ANT) für den Prototypen wird auf die Anschaffung einer Lambda/4-Antenne verzichtet und ein einfacher Kupferdraht mit entsprechender Länge eingesetzt.

$$\text{Antennenlänge} = \frac{\lambda}{4} = \frac{c}{4 \cdot f} = \frac{300000 \left[\frac{km}{s} \right]}{4 \cdot 868 [MHz]} \approx 8.6cm.$$

¹Frequenzumtastung (Frequency Shift Keying)

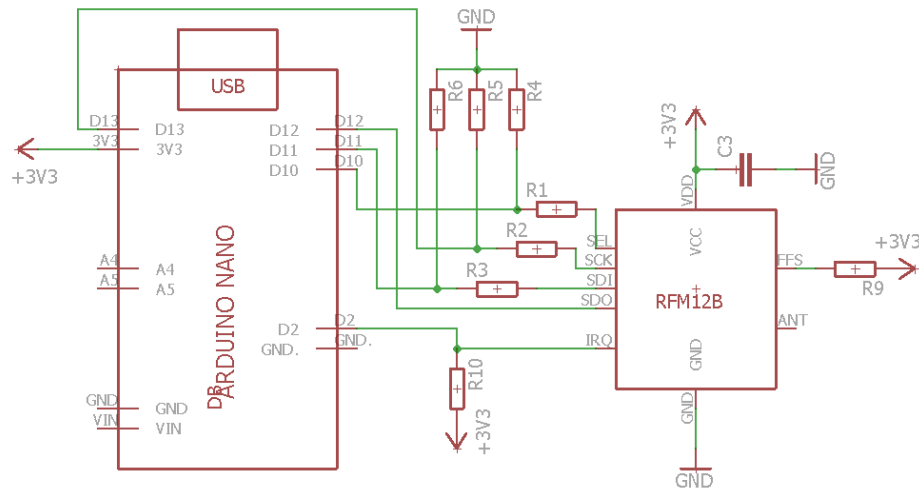


Abbildung 4.5: Schaltung – Arduino mit RFM12B Funkmodul

4.1.2 Software

Im ersten Schritt wird folgend die Laufzeitumgebung von CoDeSys auf dem Raspberry eingerichtet. Der Entwicklungsprozess der Software, speziell in CoDeSys und Arduino IDE, wird nach dem V-Modell organisiert. Sämtliche funktionale Einheiten werden dabei im Einzelnen und abschließend im Gesamten getestet.

4.1.2.1 CoDeSys einrichten

Als Programmiersystem wurde CoDeSys V3 in der Spezifikation festgelegt. Die Entwicklungsumgebung ist kostenlos über den CoDeSys-Store erhältlich. Lizenzkosten werden dabei je Laufzeitumgebung von CoDeSys erhoben. Die Laufzeitumgebung für den Raspberry Pi wurde speziell für Test- und Lehrzwecke entwickelt. In der vollwertigen Demolizenz muss das System nach zwei Stunden neu gestartet werden. Die Laufzeitumgebung für den Raspberry wird als fertiges Installations-Paket mit der vorinstallierten Linux-Distribution *Raspian* als Betriebssystem ausgeliefert. Das Laufzeitsystem verfügt durch die parallel ausgeführten Linux-Applikationen, über keine harten Echtzeiteigenschaften. Nach Herstellerangaben belaufen sich die Verzögerungen im Idealfall auf $50\mu\text{s}$ bis maximal $400\mu\text{s}$, welche für die Anwendung als Smart Home System durchaus vernachlässigbar sind. [19]

Das bootfähige Datenträgerabbild der Laufzeitumgebung wird mit dem Freeware-Tool *Win32 Disk Imager* auf die künftige Speicherkarte des Raspberry Pi geladen. Als Router wird ein [W-LAN-Router](#) mit dynamischer Adressen-Verteilung verwendet. Der Raspberry

wird mit Anschluss zum Intranet und somit Entwicklungsrechner das erste Mal hochgefahren. Über die Kommandozeile des Entwicklungsrechners wird mit dem Befehl `arp/a` die dynamisch zugewiesene IP-Adresse des RasPi identifiziert (192.168.1.3). Sollte dies aus Gründen der Größe des Intranet-Netzes nicht möglich sein, muss der Raspberry lokal mit Tastatur und Bildschirm gestartet werden. Mit Bekanntheit der IP-Adresse erfolgt die Einwahl zur Konfiguration des Raspberry, mithilfe des Freeware Telnet-Client PuTTY, das das [SSH²](#)-Protokoll über den Port 22 für einen sicheren Datenaustausch verwendet (Abb. 4.6). Nach erfolgreicher Einwahl erfolgt die Benutzer- bzw. Password -abfrage des Raspberry. Folgend die Standard-Authentifizierung des Raspberry:

```
login as: pi
password: raspberry
```

Über den Befehl `sudo raspi-config` gelangt man in das *Configuration Tool*. Unter Punkt 1 wird die Speichernutzung der SD-Karte auf die gesamten 8GB expandiert. Über Punkt 8 werden die *Advanced Options* aufgerufen. Manche GPIO's des Raspberry können je nach Konfiguration für unterschiedliche Funktionen genutzt werden (Siehe Abbildung S.6). Ein- bzw. Ausgänge können gegebenenfalls für [UART](#), [SPI](#) oder [I²C](#) aber auch als normale digitale Ein-/Ausgänge genutzt werden. Die Funktionen UART sowie SPI werden an dieser Stelle deaktiviert, da sie als digitale Ein-/Ausgänge genutzt werden sollen. Der I²C-Bus wird aktiviert. Sämtliche Konfigurationen erhalten durch einen Neustart ihre Aktivität.

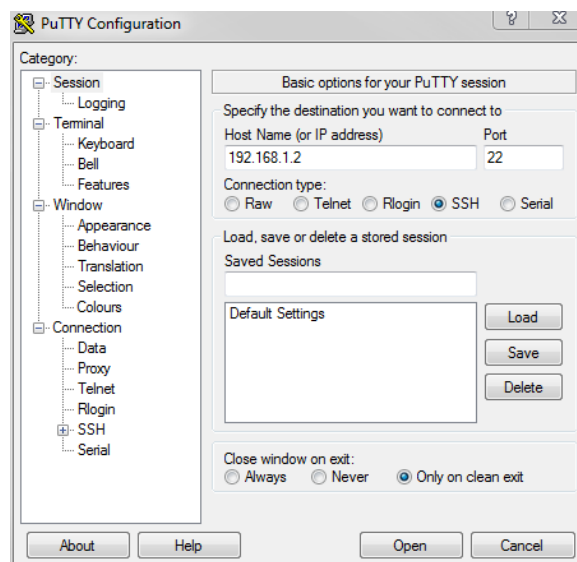


Abbildung 4.6: Telnet Client – PuTTY

²Verschlüsseltes Netzwerkprotokoll (Secure Shell)

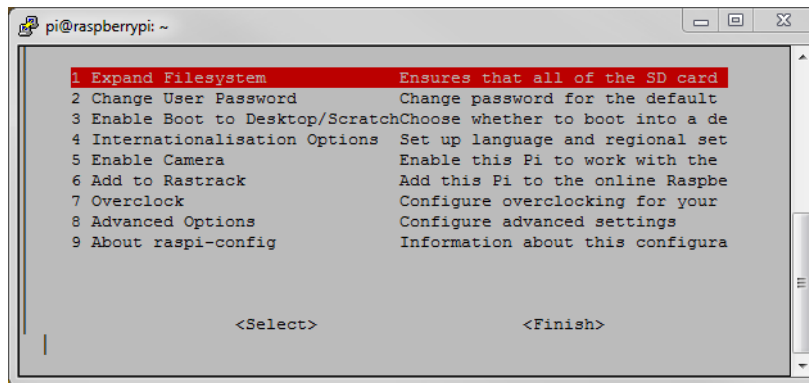


Abbildung 4.7: Raspberry – Configuration Tool

Nach erfolgter Konfiguration des Raspberry Pi kann die Entwicklungsumgebung CoDeSys eingerichtet werden. Das im Installations-Paket enthaltene CoDeSys-Package *CoDeSys_Control_for_Raspberry_Pi* wird im leeren Projekt unter *Tools* → *Package Manager* installiert. Anschließend kann der Raspberry Pi in der Geräteübersicht über das Menü *Geräte anhängen* selektiert werden. Sämtliche Raspberry-Eigenschaften sind nun im Gerätebaum abgebildet. Über die Geräte-Anwahl wird das Netzwerk nach dem Raspberry durchsucht und erhält Zugriff auf die Laufzeitumgebung des RasPi's. Für den ersten Zugriff auf die Laufzeitumgebung wird dessen Software-Version auf die Neueste aktualisiert. Die spezielle Raspberry Aktualisierung erfolgt über *Tools* → *Update RaspberryPi*. In folgender Abbildung (4.8) ist die Entwicklungsumgebung des Raspberry's samt geöffnetem Update-Tool abgebildet.

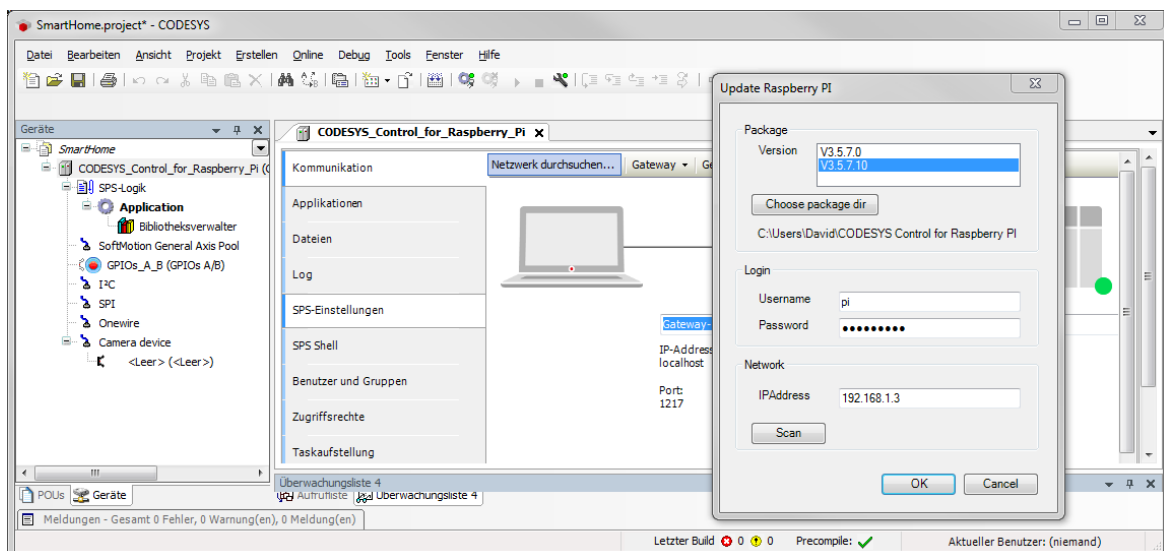


Abbildung 4.8: CoDeSys – Raspberry Pi Update

Für die kontinuierliche Nutzung der Laufzeitumgebung wird die Raspberry Pi Lizenz über den CoDeSys-Store erworben. Das Lizenz-Ticket lässt sich über *Tools* → *Lizenz Manager* mit der Ticketnummer einmalig aktivieren. Durch die Aktivierung ist die Lizenz an den Raspberry Pi gebunden und kann nur auf demselben Gerät wieder eingespielt werden. Für eine Neuinstallation muss die Lizenzdatei allerdings gesichert werden. Über den Reiter *Dateien* in der Geräteübersicht können Daten zwischen Laufzeitumgebung und Entwicklungsrechner übertragen werden. Im Ordner *Backup* der Laufzeitumgebung befindet sich die Lizenzdatei „3SLicenseInfo.tar“, welche auf dem Entwicklungsrechner gesichert wird. Bei einer Neuinstallation erfolgt die erneute Lizenznutzung durch Ablage der Lizenzdatei im Ordner *Restore*. Durch anschließenden Reboot des Systems, kann die Lizenz über den Lizenz Manager aktiviert werden. [19]

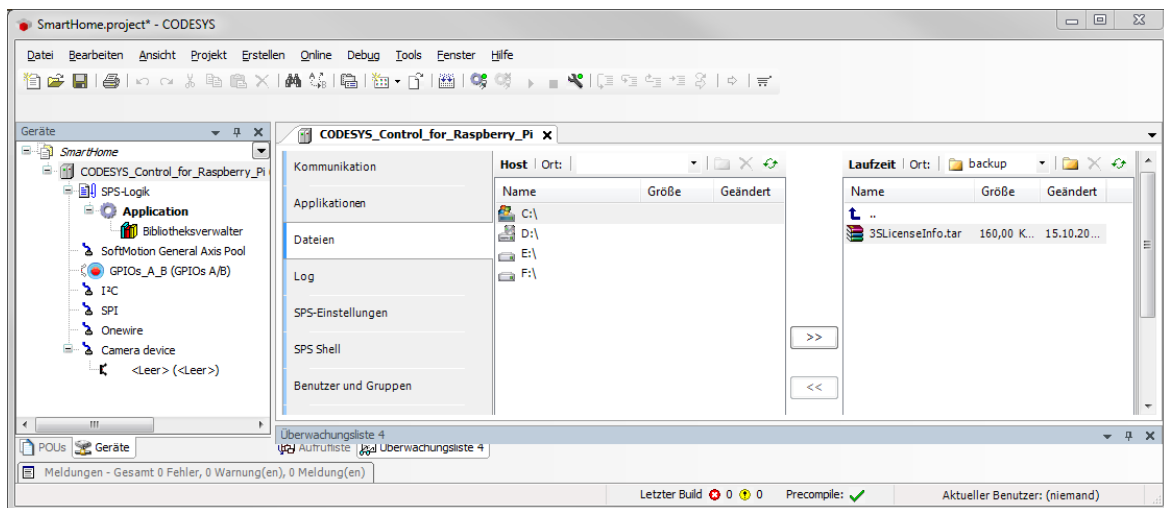


Abbildung 4.9: CoDeSys – Lizenzierung

4.1.2.2 GPIO - Anpassung

Wie bereits erwähnt können einzelne GPIO-Anschlüsse des RasPi's je nach Konfiguration für unterschiedliche Funktionen genutzt werden. In der CoDeSys Standard Treiber-Konfiguration steht diese Auswahl allerdings nicht zur Verfügung. Die GPIO's für SPI und UART sind nur als solches zu verwenden und stehen für die Verwendung als digitale Ein-/Ausgänge in der GPIO-Konfiguration im Gerätebaum nicht zur Verfügung. Eine komfortablere Leiterbahnführung in der Platinenlayout-Planung erfordert allerdings die Nutzung der Anschlüsse als digitale Ein-/Ausgänge. Zur Aktivierung der Anschlüsse GPIO7, GPIO8, GPIO14 und GPIO15 wird die Gerätebeschreibung (GPIOsBplus.devdesc) der GPIO angepasst. In folgender Abbildung 4.10 ist beispielhaft ein Codeausschnitt abgebildet, der der Gerätebeschreibung zur Aktivierung des Host-Parameters für den Anschluss GPIO7 hinzugefügt wird. Die modifizierte Gerä-

tebeschreibung *GPIOs B+ Spez* wird anschließend über *Tools* → *Geräte-Repository* installiert. Mit rechter Maustaste auf GPIO im Gerätebaum kann die Aktualisierung der modifizierten Gerätebeschreibung abgeschlossen werden. Die GPIOs stehen in der GPIO-Übersicht zur Konfiguration zur Verfügung (Abb. 4.11). An dieser Stelle erhalten die gewählten GPIOs nach der Abbildung ihre Deklaration als Input bzw. Output. Über den Reiter *GPIOs E/A-Abbild* werden die deklarierten Ein-/Ausgänge mit ihren Variablen verheiratet. Hierfür wird die Variable *ZENTRALE* mit der definierten Struktur *DUT_Zentrale* in der Variablenliste *GVL_Main* angelegt. Der Struktur-Zugriff mit Zugriff zu den Ein-/Ausgängen ergibt sich aus folgenden Beispielen:

- Eingang 1: `ZENTRALE.Input [0]`
- Ausgang 8: `ZENTRALE.Output [7]`

```
<DeviceInfo>
  <Name name="local:ModelName">GPIOs B+ Spez</Name>
  ...
</DeviceInfo>
...
<HostParameterSet>
  ...
  <Parameter ParameterId="7" type="local:GPIOType">
    <Attributes channel="none" download="true" functional="false" onlineaccess="read"/>
    <Default>255</Default>
    <Name name="local:">GPIO7</Name>
    <Description name="local:">configuration of GPIO7</Description>
  </Parameter>
  ...
</HostParameterSet>
...
```

Abbildung 4.10: Code-Ausschnitt GPIO Treiber

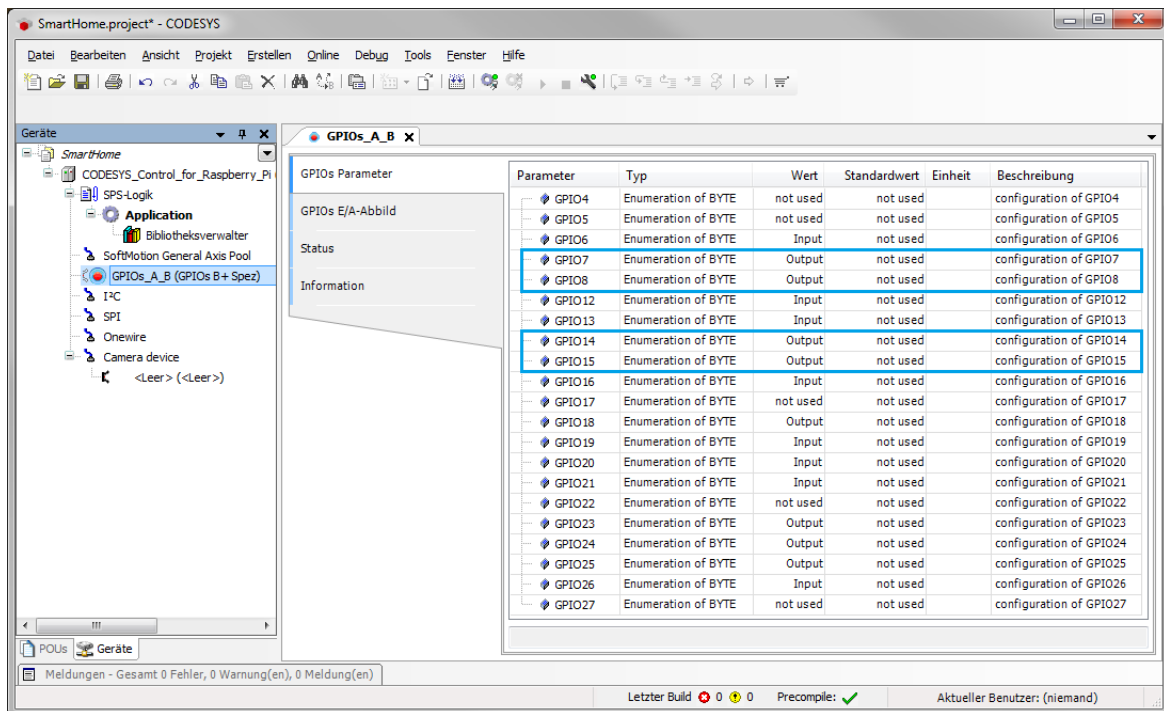


Abbildung 4.11: GPIO Konfiguration

4.1.2.3 I²C-Treiber - Arduino Funk

Als Vorlage für den I²C Treiber für den Arduino Funk dient das SPI-Treiberbeispiel des MCP23008 Bauteils (*MCP3008.devdesc*) samt Bibliothek (*SPI_MCP3008.library*) aus dem CoDeSys Installationspaket. Die Gerätebeschreibung erhält dabei eine eindeutige Bezeichnung sowie die Deklaration als I²C-Slavegerät (siehe Abb. 4.12). In der Gerätebeschreibung werden die FB-Instanz sowie die Methoden *afterReadInputs* und *beforWriteOutputs* der dazugehörigen Bibliothek deklariert. Die Methoden sind bereits von CoDeSys vordefiniert und werden zyklisch vom I²C-Master ausgeführt. Die Methode *afterReadInputs* definiert in der Bibliothek die Eingangspakete und beschreibt die Verarbeitung der Eingangsdaten. Die Methode *beforWriteOutputs* definiert die Ausgangspakete und beschreibt die Verarbeitung der Ausgangsdaten. Der Treiber erhält dabei den Namen RFM12B, der zugleich den Namen der Bibliothek-FB-Instanz darstellt. Die FB-Instanz *RFM12B* ist für die Initialisierung der Bibliothek zuständig und ist ein Erbe der CoDeSys-Klasse *i2C*, wodurch sie die Methoden der Mutter-Klasse erbt. Die enthaltene Zustandsmaschine mit der Zustandsvariable *_iState* beschreibt den Initialisierungsvorgang (*_iState=0*: Initialisierungszustand, *_iState=10*: normaler Betrieb, *_iState=1000*: Fehler). Die Standard Slave-Adresse des Arduino Funk wird mit 2 belegt. Die Projektinformationen der Bibliothek sind entsprechend dem Gerätetreiber zu aktualisieren (Abb. 4.13).

```

...
<DeviceIdentification>
  <Type>500</Type>
  <Id>FFFF_0100</Id>
  <Version>1.1.0.0</Version>
</DeviceIdentification>
<DeviceInfo>
  <Name name="local:ModelName">RFM12B</Name>
  <Description name="local:DeviceDescription">Arduino RFM12B</Description>
  <Vendor name="local:VendorName">David Bauer</Vendor>
  ...
</DeviceInfo>
<Connector moduleType="500" interface="Raspberry.I2C" role="child"explicit="false"
  connectorId="1" hostpath="-1">
  <InterfaceName name="local:PCI">I2C-Bus</InterfaceName>
  <Slot count="1" allowEmpty="false">
  </Slot>
  <DriverInfo needsBusCycle="false">
    <RequiredLib libname="I2C - Arduino RFM12B" vendor="David Bauer"
      version="1.1.0.0" identifier="deviceLib">
      <FBInstance basename="(DeviceName)" ffname="RFM12B">
        <Initialize methodName="Initialize" />
        <CyclicCall methodName="AfterReadInputs" task="#buscycletask"
          whentocall="afterReadInputs" />
        <CyclicCall methodName="BeforeWriteOutputs" task="#buscycletask"
          whentocall="beforeWriteOutputs"
      >
    </RequiredLib>
  </DriverInfo>
  ...

```

Abbildung 4.12: Code-Ausschnitt Arduino Funk Treiber

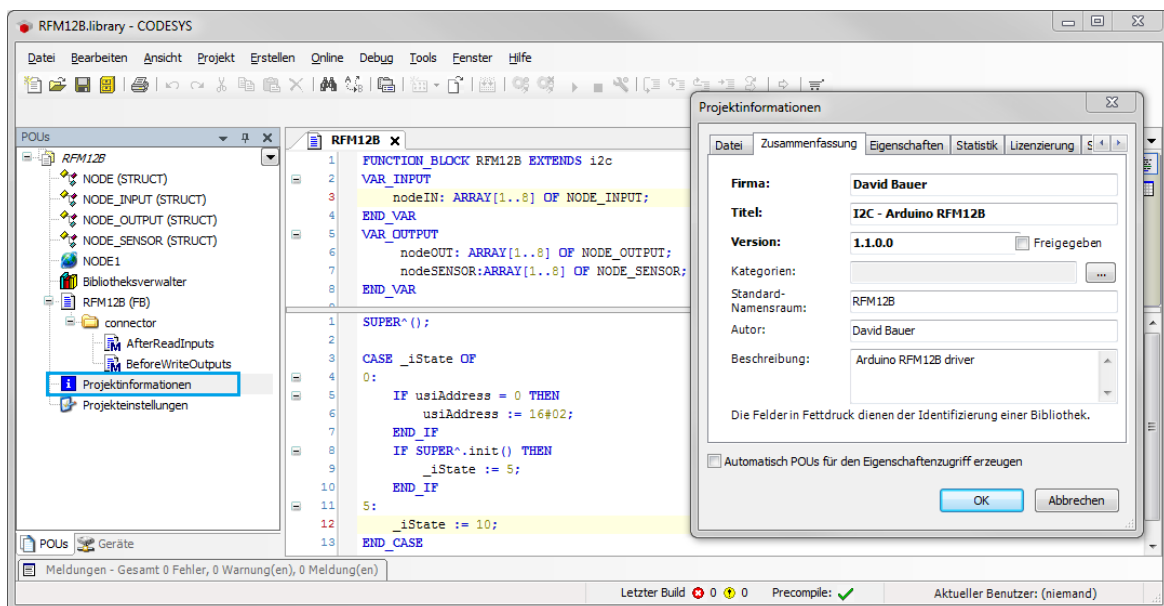
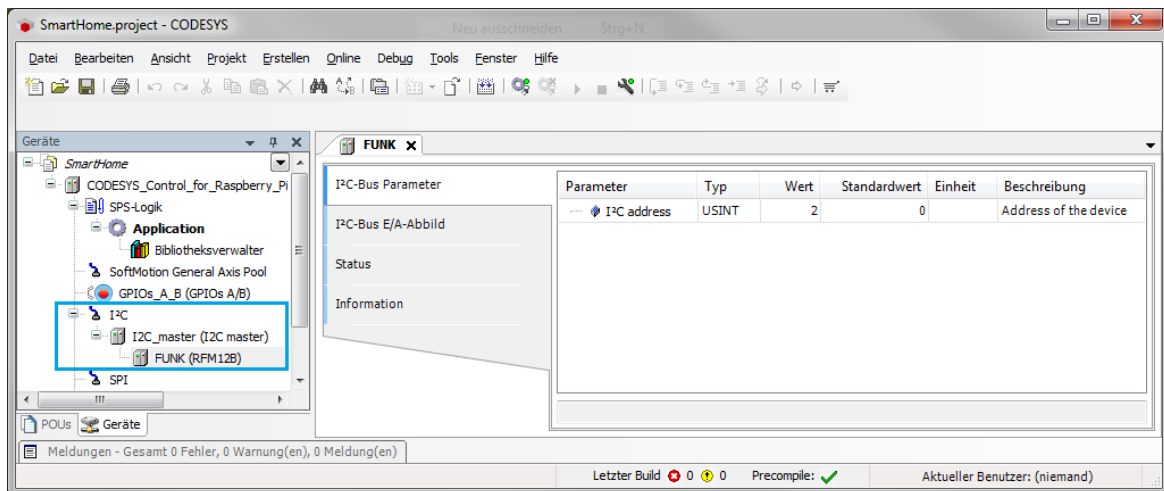


Abbildung 4.13: Projektinformationen - Arduino Funk Bibliothek

Abbildung 4.14: I²C-Slave FUNK (RFM12B)

Als Kommunikationspartner des Arduino Funk wurden 8 Funk-Steuermodule sowie 8 Funksensoren spezifiziert. Über das Funkmodul erhält der Arduino die Empfangspakete mit entsprechender Absenderadresse. Die Empfangsdaten werden folgend per I²C-Bus zum RasPi übertragen. Entsprechend der Absenderadresse werden die Empfangsdaten in der Methode *afterReadInputs* der betreffenden ARRAY-Struktur `nodeOUT[1..8]` bzw. `nodeSENSOR[1..8]` zugewiesen. In entgegengesetzter Richtung werden Steuerbefehle der Steuermodule über die ARRAY-Struktur `nodeIN[1..8]` in der Methode *beforeWriteOutputs* ausgewertet und mit Empfängeradresse zum Arduino übertragen. Die Datenverarbeitung der Methoden wird in Abbildung 4.15 als Ablaufdiagramm dargestellt. Die Variablen `nodeOut[]` bzw. `nodeIn[]` beschreiben die Ein-/Ausgangsdaten des Funk IO-Moduls. Die Sensordaten des Funk Sensor-Moduls werden in der Variable `nodeSENSOR[]` gespeichert. Die Daten-Pakete zur I²C-Übertragung sind in Tabelle 4.1 abgebildet. Die Byte-Zuweisung geht daraus hervor. Die jeweiligen Bytes des Eingangspuffers (16bit) der Variablen `control` und `state` beschreiben den Zustand der Ausgänge bzw. Eingänge des IO-Moduls. Die gleichnamige `control`-Variable des Ausgangspuffers (16Bit) beschreibt die Steueranweisung als Eingang des IO-Moduls. Eine logische 1 führt dabei einen Ausgang-Zustandswechsel des IO-Moduls aus (Setzbefehl aus CoDeSys für IO-Modul 1: `FUNK.nodeIN[1].control := 1`).

Tabelle 4.1: I²C Ein-/Ausgangspuffer

Eingangspuffer [BYTE]	0	1	2	3	4	5	6	7
Steuermodul [1..8]	Adresse [1 - 8]		control		state		Reserve	
Sensormodul [1..8]	Adresse [11 - 18]		Temperatur		Feuchtigkeit		Helligkeit	

Ausgangspuffer [BYTE]	0	1	2	3	4	5
Steuermodul [1..8]	Adresse [1 - 8]		control		Reserve	

Die erstellte Bibliothek *RFM12B.library* wird im CoDeSys-Projekt „SmartHome“ über den Reiter *Tools* → *Bibliothek-Repository* installiert. Die dazugehörige Gerätebeschreibung *RFM12B.devdesc* wird über *Tools* → *Geräte-Repository* hinzugefügt. Folgend lässt sich der I²C-Slave FUNK vom Gerätetyp RFM12B im Gerätebaum des Projekts hinzufügen. Hierfür wird über die I²C-Eigenschaft im Gerätebaum zunächst der I²C-Master angehängt und mit dem Slave ausgestattet (Abb. 4.14).

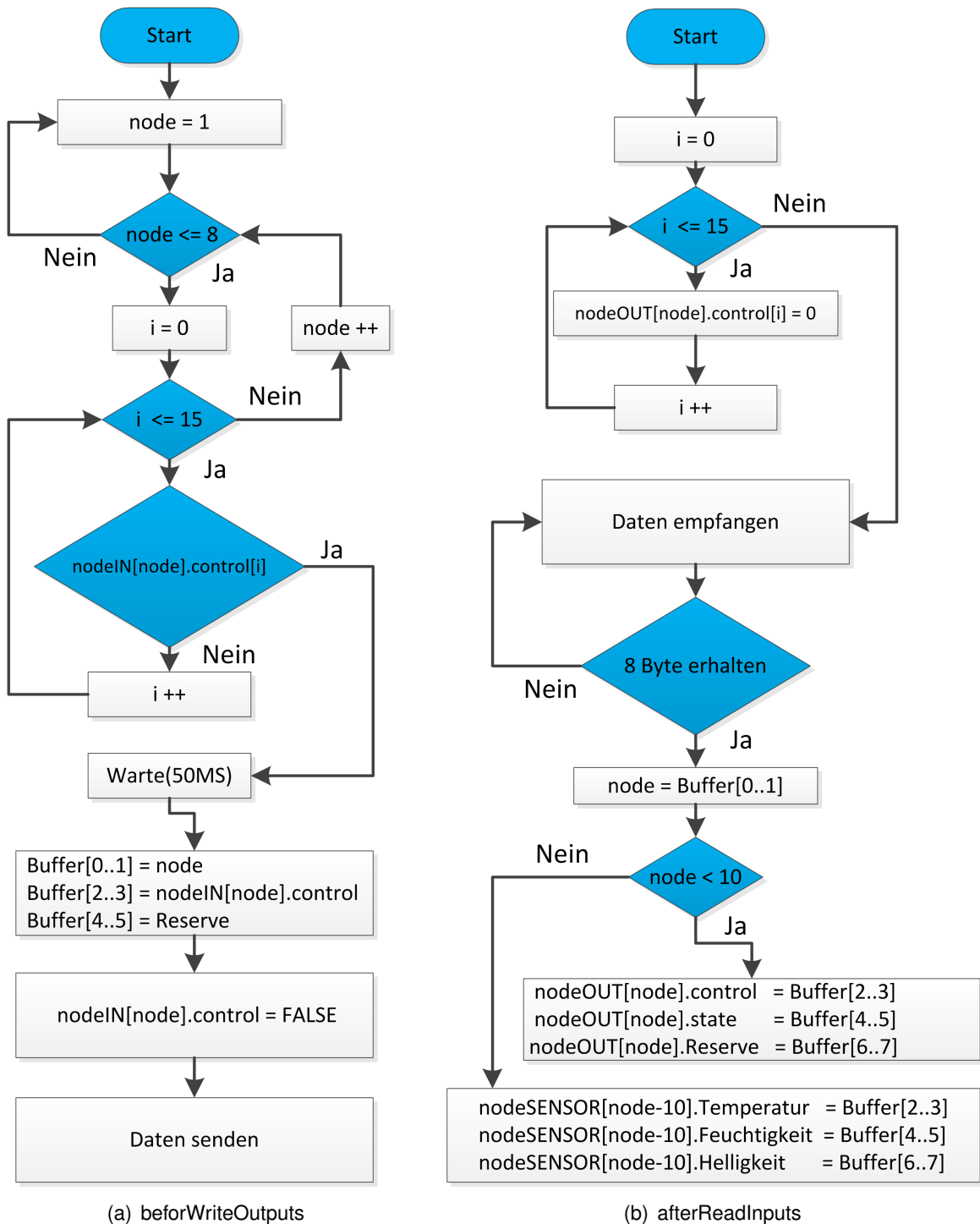


Abbildung 4.15: Arduino Funk – Bibliotheksmethoden

4.1.2.4 Sketch - Arduino Funk

In der Hardware-Umsetzung der Smart Home Zentrale wird ein Arduino Nano als Funk-Controller des Funk-Modules RFM12B eingesetzt (4.1.1.5). Er hat die Aufgabe Empfangsdaten des Funk-Modules auszuwerten und per I²C-Bus an den Raspberry bzw. CoDeSys zu übertragen. Der Arduino fungiert in seiner Rolle dabei als I²C-Slave. Folgend wird der Programmablauf bzw. Sketch des Arduinos erläutert.

Dank der umfangreichen Bibliothek des Arduinos wird die spezielle Bibliothek `<RFM12B.h>` des Funk-Modules inkludiert. Für die Nutzung des I²C-Buses wird die Standard-Bibliothek `<Wire.h>` verwendet. In folgendem Code-Ausschnitt (Abb. 4.16) ist die Initialisierung der Bibliotheks-Umgebung abgebildet.

Für die Verwendung der RFM12B-Bibliothek wird zuvor die Variable `radio` vom Typ `RFM12B` instanziiert. Die verwendeten Bibliotheks-Funktionen werden im Anhang A.4.1 im Einzelnen erläutert. Mit der Initialisierung des Funk-Moduls wird dessen Adresse mit `NODEID=31` sowie die Netzwerk-Adresse mit `NETWORKID=212` festgelegt. Mit der Definitions-Variable `RF12_868MHZ` wird die Übertragungs-Frequenz auf `868MHz` gesetzt. Mit der Funktion `Encrypt()` wird die Verschlüsselungs-Option der Übertragungsdaten genutzt. Empfangsdaten sind dadurch nur mit dem `KEY="ABCDABCDABCDABCD"` zu entschlüsseln.

Zur Aktivierung des I²C-Bus wird die Funktion `Wire.begin()` ausgeführt. Durch Angabe des Übergabe-Parameters `I2C_ADDRESS = 2` wird die Teilnahme am Bus-System als Slave definiert. Wichtige Definition als I²C-Slave ist die Angabe der Receive- bzw. Request-Handler. Deren Übergabeparameter benennt die durch den I²C-Master initiierten Sketch-Funktionen, welche im Sketch definiert werden und eingehende bzw. ausgehende Bus-Daten organisieren. Initiiert der I²C-Master eine Datenübertragung zum Arduino wird nach vorliegender Initialisierung die Funktion `receiveEvent()` ausgeführt. Fordert der Master Daten an, wird die Funktion `request()` ausgeführt.

```
// Funkmodul initialisieren
radio.Initialize(NODEID, RF12_868MHZ, NETWORKID);

// Daten zur Übertragung verschlüsseln
radio.Encrypt(KEY);

// I2C Initialisierung
Wire.begin(I2C_ADDRESS);           // I2C-Slave Init.
Wire.onReceive(receiveEvent);     // Bus-Receive-Handler
Wire.onRequest(requestEvent);     // Bus-Request-Handler
```

Abbildung 4.16: Code-Ausschnitt – Initialisierung

In folgender Abbildung 4.17 wird der Arduino-Sketch bzw. dessen Funktionen anhand von Struktogrammen dargestellt. Die klassische Grundfunktion `void setup()`, die zum Start des Arduinos ausgeführt wird, sorgt für die Initialisierung des Mikrocontrollers. Die kontinuierlich ausführende Hauptfunktion `void loop()` beschreibt die Organisation der erhaltenen Funkdaten sowie deren Weiterleitung in den I²C-Ausgangspuffer. Durch Setzen der I²C-Flanke wird der Ausgangspuffer freigegeben und bei folgender Initiierung der Funktion `void requestEvent()` auf den Bus gelegt. Die eingehenden Master-Daten werden durch die Funktion `void receiveEvent()` im Eingangspuffer gespeichert und abschließend die Sende-Flanke gesetzt. Die gesetzte Sende-Flanke sorgt in der `void loop()`-Routine für die Sendung der Pufferdaten über das Funkmodul RFM12B.

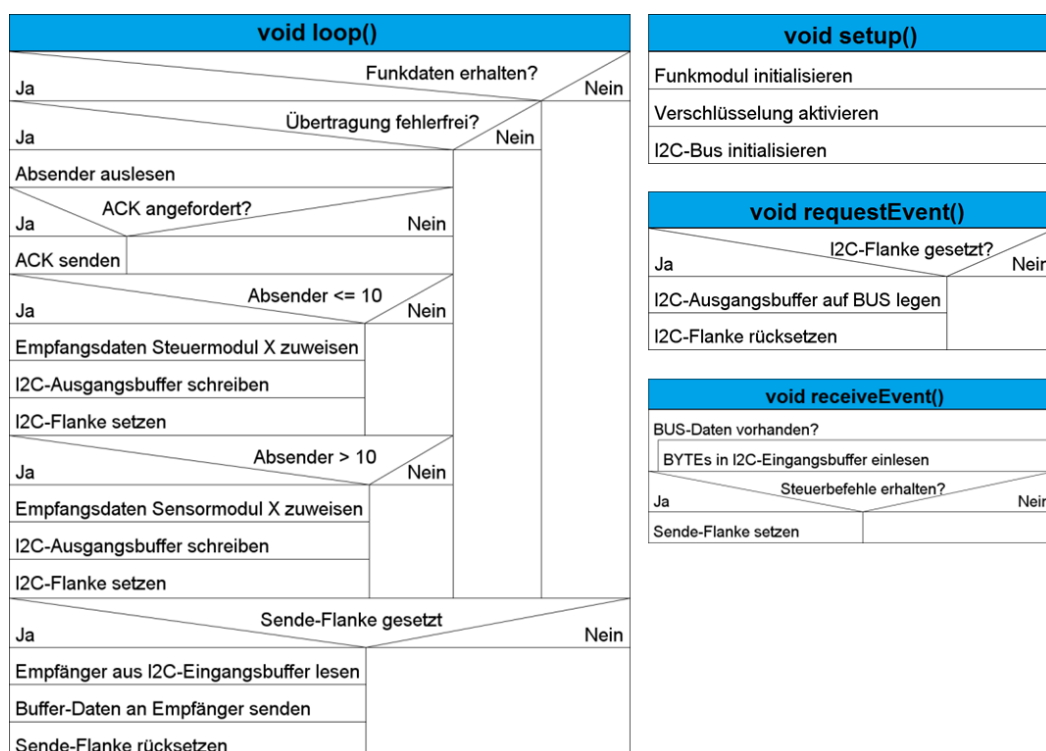


Abbildung 4.17: Struktogramm – Arduino Funk Controller

4.1.2.5 CoDeSys - Softwarestruktur

Die Programmierung des Smart Home Systems wird mit der in CoDeSys V3 nutzbaren, objektorientierten Programmierung (OOP) umgesetzt. Die OOP bietet eine modulare Organisation sämtlicher Smart Home Eigenschaften. Funktionale Zusammenhänge werden dabei in Klassen strukturiert, die entsprechende Methoden und Eigenschaften bereitstellen. Der große Nutzen der OOP liegt in der Wiederverwendbarkeit der einzelnen Klasseninstanzen bzw. -objekte. Im Folgenden werden die einzelnen Klassen vorgestellt deren Zusammenhang im abschließenden Klassendiagramm ersichtlich ist.

Klassen - INTERFACES

Eine weitere Stärke der OOP-Struktur ist die Datenkapselung der einzelnen Klassen. Durch die Möglichkeit der Schnittstellendefinition, dem Erstellen von sogenannten INTERFACES, können Methoden und Eigenschaften bereits im Vorfeld deklariert werden. In folgender Abbildung 4.18 sind die erstellten Interfaces ILicht, ISensor und IRollo dargestellt. Die Methoden und Eigenschaften werden dabei lediglich deklariert. Eigenschaften können dabei sowohl als Setter als auch als Getter deklariert werden, und legen so die Schreib- bzw. Leseberechtigung fest. Durch Implementierung in eine Klasse erhalten sie ihre Funktion und müssen dementsprechend definiert werden. Die Schnittstellen werden in ihrer verwandten Klasse vorgestellt.

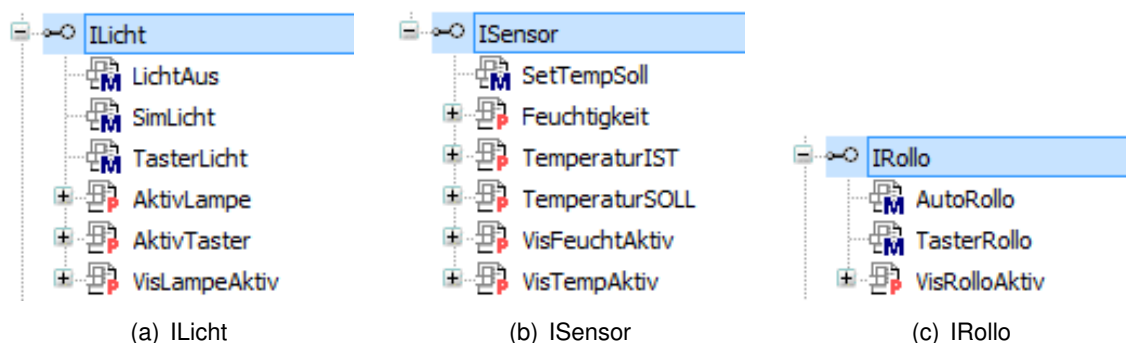


Abbildung 4.18: Klassen - Interfaces

Klasse - KL_LICHT

Die Klasse `KL_LICHT` ist für die Lichtschaltung zuständig. Mit Aufruf der Aktions-Methode `Taster` wird die Ausgangseigenschaft `Out` als Steuerbefehl ein- bzw. ausgeschaltet. Die Eigenschaft `Zustand` wird mit der Schaltbestätigung verknüpft und signalisiert den realen Lampenzustand. Folgend ist die Deklaration sowie die strukturelle Darstellung der Klasse in CoDeSys dargestellt.

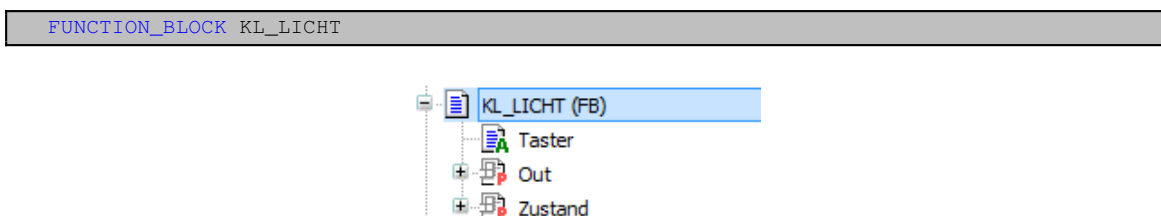


Abbildung 4.19: Klasse `KL_LICHT`

Klasse - KL_ROLLO

Mit der Klasse KL_ROLLO wird eine Rollsteuerung realisiert. Über die Aktions-Methode `Taster` wird das Rollo im halbautomatischen Betrieb abwechselnd entweder ganz auf- oder zugefahren bzw. im aktiven Betrieb gestoppt. Die Motoransteuerungen sind über die Ausgangseigenschaften `MRollo_AUF` und `MRollo_ZU` beschrieben. Die Eigenschaft `Position` beschreibt den aktuellen Zustand des Rollladens mit 0 im geöffneten bis 100 im geschlossenen Zustand. Mit der Eigenschaft `Zeit_AufZu` wird die Zeit in ms für den Schließ- bzw. Öffnungsvorgang vorgegeben (Standardwert:10s). Mit `Zeit_Nachlauf` wird die Nachlaufzeit am Endschalter in ms konfiguriert (Standardwert:1s). Die Nachlaufzeit sorgt für eine fortlaufende Kalibrierung an den Endpositionen. Bei Initialisierung wird eine Öffnung des Rollladens in der doppelten Öffnungszeit durchgeführt und sorgt für den Initialisierungszustand im geöffneten Rollo-Zustand. Die Zeiten sind über die Konfigurations-Visualisierung änderbar. Über die Methode `SetMode` mit Übergabeparameter vom Typ `DUT_MODE_ROLLO` (AUTOAUF, AUTOZU, AUTO50) können die Positionen 0, 50 und 100 angefahren werden. Aktive Steuerungen werden dabei gestoppt und folgend der neue Steuerbefehl ausgeführt. In folgender Abbildung sind die Klassendeklaration sowie die Baumstruktur in CoDeSys dargestellt. Die Rollsteuerung wird mit dem Aktivitätsdiagramm in Abbildung 4.21 dargestellt.

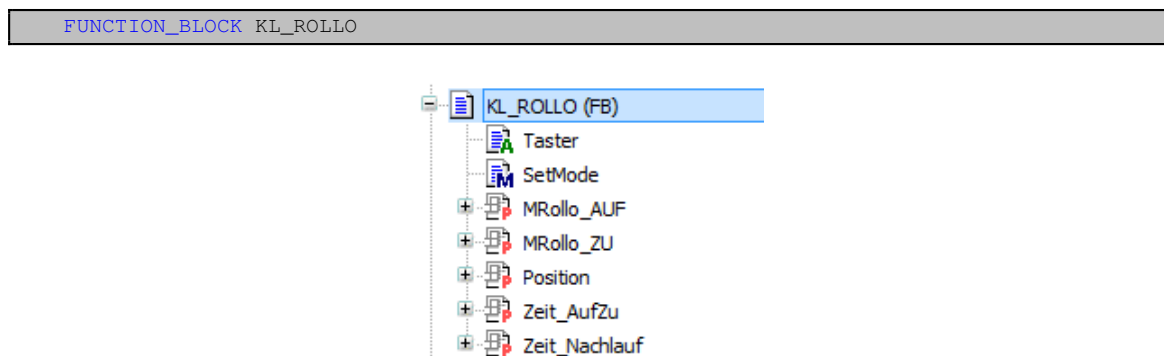


Abbildung 4.20: Klasse KL_ROLLO

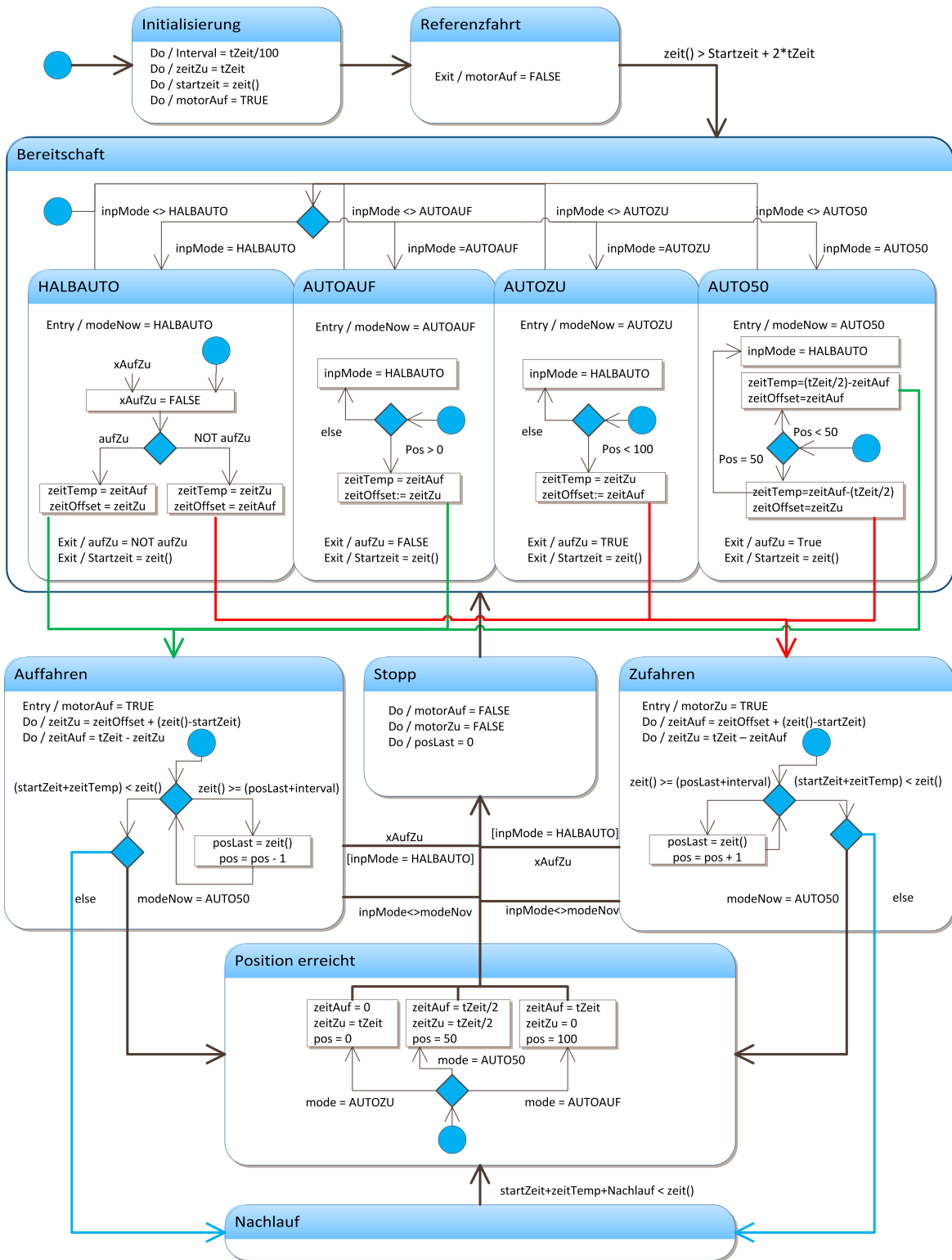


Abbildung 4.21: Aktivitätsdiagramm – Klasse KL_ROLLO

Klasse - KL_RAUMKONFIG

Die Visualisierungsseite *Konfiguration* ermöglicht dem Anwender die Ein- und Ausgänge des entsprechenden Moduls für Licht- bzw. Rollosteuern oder den Sensor eines Raumes zu konfigurieren. Die Konfiguration der Ein-/Ausgänge wird persistent gespeichert und steht somit auch bei eventuellem Absturz des Systems weiter zur Verfügung. Die Klasse KL_RAUMKONFIG ist für die Auswertung der Konfigurationsdaten zuständig und stellt über deren Eigenschaften die entsprechenden Ein- und Ausgangs-Referenzen der Raumsteuerung zur Verfügung. Der Datentyp REFERENCE ist einem Zeiger ähnlich, allerdings wird der referenzierte Datentyp (bspw. REFERENCE TO BOOL) im Speicher direkt beeinflusst. Über die Referenz kann sowohl geschrieben als auch gelesen werden. Mit dem Operator `__ISVALIDREF` kann überprüft werden, ob es sich bei der Referenz-Variable um einen gültigen Wert handelt. Die Eigenschaften `Rollo_IN`, `Rollo_AUF` und `Rollo_ZU` liefern die konfigurierten BOOL-Referenzen zur Rollosteuern. Die Eigenschaften `Licht_IN1`, `Licht_IN2`, `Licht_STATUS` und `Licht_OUT` liefern die BOOL-Referenzen zur Lichtsteuerung. Über die Eigenschaften `RolloKonf_OK` und `LichtKonf_OK` wird die erfolgte Konfiguration ausgegeben. Die Eingangseigenschaft `Konfiguration` vom Typ `DUT_VAR_RAUM` liefert die Konfigurationsdaten der Konfigurations-Visualisierung. Folgend sind der Klassenbaum von CoDeSys sowie die Klassendeklaration dargestellt.

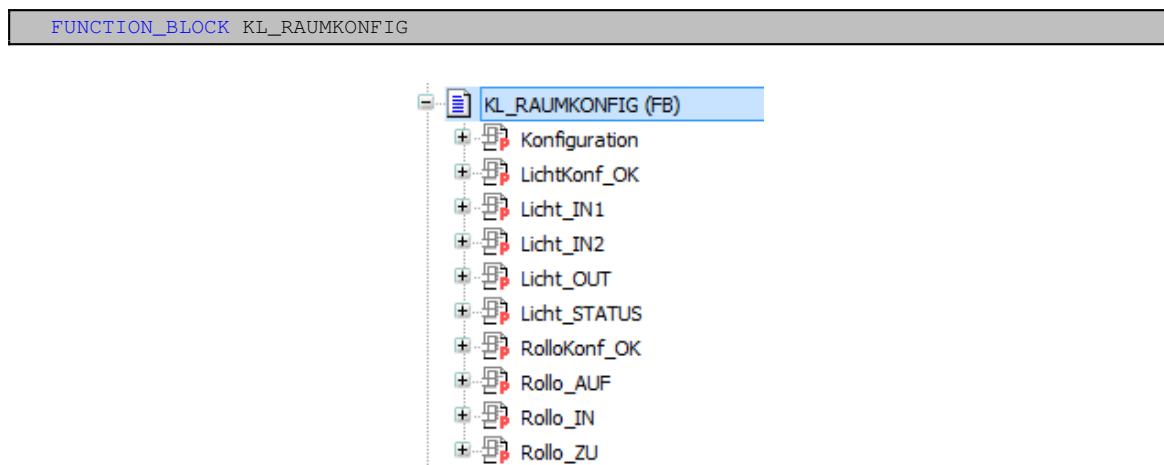


Abbildung 4.22: Klasse KL_RAUMKONFIG

Klasse - KL_RAUM_TYP1

Die Klasse KL_RAUM_TYP1 beschreibt einen Standard-Raum mit lediglich einer Lampenschaltung. Die Klasse beinhaltet jeweils ein Objekt der Klasse *KL_Licht* als Lichtsteuerung sowie ein Objekt der Klasse *KL_RaumKonfig*, welche die durch den Anwender konfigurierten Ein- bzw. Ausgänge der Lampenschaltung bereitstellen. Die erfolgte Konfiguration wird über die Eigenschaft `VisLampeAktiv` ausgegeben, welche für die Aktivierung der Lampenvisualisierung genutzt wird. Über die Aktions-Methode `TasterLicht` wird das Licht ein-

bzw. ausgeschaltet. Die Aktivität der Ausführung wird dabei über die Eigenschaft `Aktiv` für einen Programmlauf ausgegeben. Eine spezielle Lichtansteuerung über `SimLicht` unterbindet die Ausgabe der Aktivität. Mit der Methode `LichtAus` wird die Beleuchtung ausgeschaltet. Die Methode `RaumLicht` ist für die Auswertung der Raumkonfigurationsdaten und Ansteuerung der Licht-Instanz zuständig. Durch den Zugriffsschutz `PROTECTED` ist der Zugriff auf die Methode nur für Verwandtschaften gestattet. Die besagten Methoden werden über das Interface `ILicht` bereitgestellt und wie folgt implementiert.

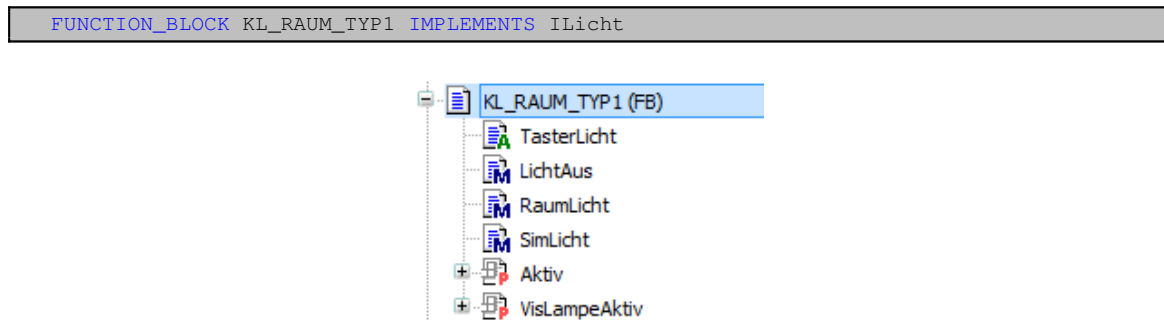


Abbildung 4.23: Klasse `KL_RAUM_TYP1`

Klasse - `KL_RAUM_TYP2`

Die Klasse `KL_RAUM_TYP2` ist ein Erbe der Klasse `KL_RAUM_TYP1` und erweitert die Raumeigenschaften um die Visualisierung von Sensordaten sowie insbesondere um die Steuereigenschaft der Temperatur-Sollwerte. Die Methode `SetTempSoll` ist für die Aktualisierung der Temperatur-Sollwerte nach dem Modus-Wochenprogramm zuständig. Das Wochenprogramm der Sollwerte je Modus kann über die entsprechende Visualisierung der Sollwerte konfiguriert werden. Die Klasse stellt die Eigenschaft-Zugänge zu `Feuchtigkeit`, `TemperaturSoll` sowie `TemperaturIST` zur Verfügung. Die Eigenschaften `VisFeuchtAktiv` und `VisTempAktiv` geben Auskunft über die erfolgte Konfiguration des Sensors und aktivieren die Sensor-Visualisierung. Die nach außen geschützte Methode `RaumSensor` (`PROTECTED`) ist für die Auswertung der Sensordaten zuständig. Die erforderlichen Konfigurationsdaten des betreffenden Sensors erhält die Klasse über die Raumkonfigurationsdaten der Mutterklasse. Auch die Funktionalität der Lichtansteuerung erhält die Klasse durch ihre Mutterklasse und wird intern mit dem `SUPER`-Aufruf der Mutterklassen-Methode `RaumLicht` aktiviert. Folgend ist die Klassendeklaration als Ableitung der Mutterklasse `KL_RAUM_TYP1` mit Interface-Implementierung `ISensor` sowie die Klassen-Bestandteile der `CoDeSys`-Ansicht abgebildet:

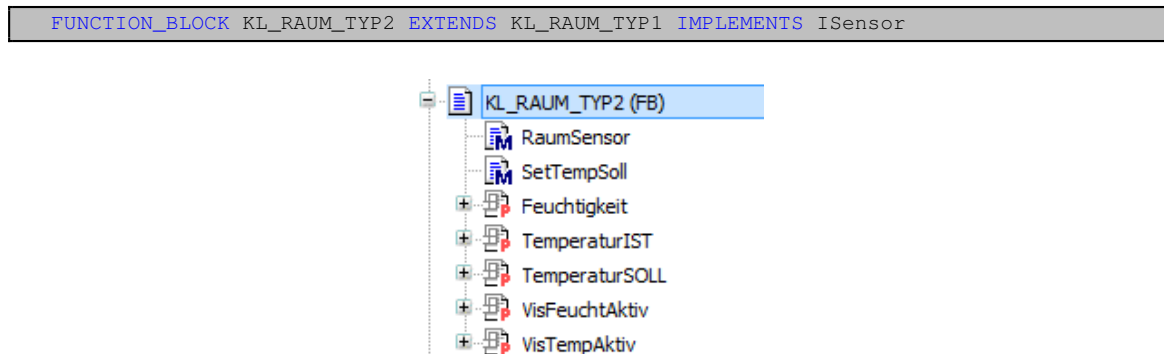


Abbildung 4.24: Klasse KL_RAUM_TYP2

Klasse - KL_RAUM_TYP3

Die Klasse `KL_RAUM_TYP3` ist ein Erbe der Klasse `KL_RAUM_TYP2` und erweitert die Raumfunktionalitäten um die Ansteuerung eines Rollos. Die Methoden und Eigenschaften werden mit dem Interface `IRollo` implementiert. Für die Rollofunktionalität sorgt ein Objekt der Klasse `KL_ROLLO`. Mit der Methode `TasterRollo` kann das Rollo auf-/zugesteuert bzw. gestoppt werden. Die Methode `AutoRollo` steuert das Rollo-Objekt im Automatikbetrieb an. Durch den Übergabeparameter vom Typ `DUT_MODE_ROLLO` (AUTOAUF, AUTOZU, AUTO50) erhält der Automatikbetrieb seinen Auftrag. Die Eigenschaft `VisRolloAktiv` aktiviert die Rollovisualisierung. Für die interne Auswertung der Rollokonfiguration ist die Methode `RaumRollo` zuständig und nach außen mit `PROTECTED` geschützt. Die Mutterfunktionen werden wiederum intern durch den `SUPER`-Aufruf der Methoden `RaumLicht` und `RaumSensor` aktiviert. In Abbildung 4.25 wird die CoDeSys-Struktur sowie Deklaration der Klasse abgebildet.

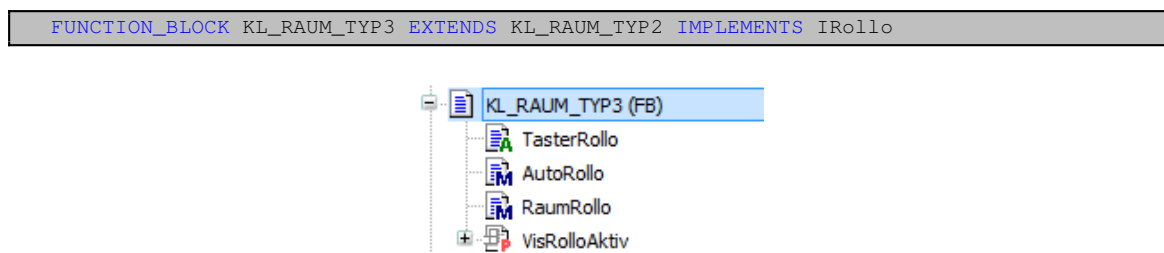


Abbildung 4.25: Klasse KL_RAUM_TYP3

Klasse - KL_MODEKONFIG

Über die Visualisierungsseite Mode-Konfiguration erfolgt die Konfiguration der Smart Home Modi `HAND`, `SLEEP`, `SECURE` sowie `SMART`. Die Konfiguration wird dabei durch aktivieren bzw. deaktivieren der speziellen Checkboxes durchgeführt. Die Konfigurationsdaten

werden dabei persistent gespeichert. Die Klasse `KL_MODEKONFIG` ist für die Auswertung der Konfigurationsdaten zuständig. Die Eingangseigenschaften `ModeToHand`, `ModeToSleep`, und `ModeToSecure` liefern die jeweiligen Raum-Konfigurationen mit Datentyp `DUT_ModeRoomConf`. Die spezielle Simulations-Konfiguration mit Nacht- und Tagschaltung wird über die Eingangseigenschaft `ModeToSmart` mit Datentyp `DUT_ModeSimConf` eingelesen. Das Ergebnis der Auswertung wird in der Globalen Variablenliste `GVL_Intern` mit den Variablen `rolloModeConfig` und `lichtModeConfig` abgelegt.

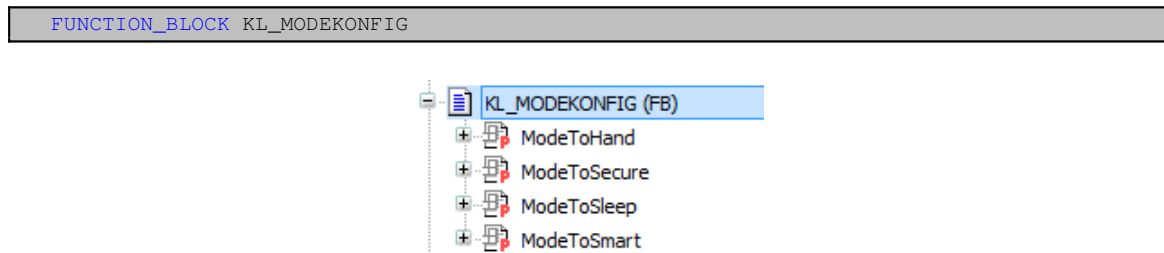


Abbildung 4.26: Klasse `KL_MODEKONFIG`

Klasse - `KL_SIMULATION`

Die Klasse `KL_SIMULATION` ist für eine Raumsimulation im Modus SMART zuständig. Je nach Simulations-Konfiguration werden Licht- und Rollosteuern automatisch ausgeführt. Mit der Methode `INIT` wird die Initialisierung der Klasse durchgeführt. Für die Lichtsimulation werden ein Belichtungszeitraum sowie eine maximale Belichtungsdauer über die Visualisierung konfiguriert (Standard: Anfang: `20Uhr`, Ende: `23Uhr`, Max. Dauer: `20min`). Die zugriffsgeschützte (`PRIVATE`) Methode `Lichtschaltung` generiert mittels Zufallsgenerator eine Zufallszahl zwischen 0 und 999. Im selben Verhältnis zum Belichtungszeitraum wird die Lichtsteuerung über die Ausgangseigenschaft `LichtSimulation` ausgegeben. Nach demselben Prinzip wird mit den privaten Methoden `RolloNachtSchaltung` und `RolloTagSchaltung` die Steuerung des Rollos über die Ausgangseigenschaft `RolloSimulation` mit Datentyp `DUT_MODE_ROLLO` ausgegeben. Nach jeder Ansteuerung wird eine neue Zufallszahl für den Folgetag generiert und sorgt so für ein wechselndes Simulationsprogramm. Folgend die Übersicht der Klassenstruktur in CoDeSys sowie deren Deklaration.

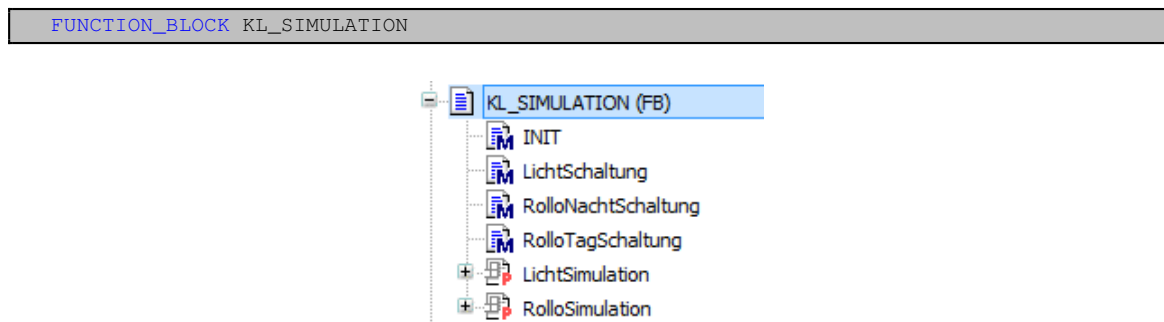


Abbildung 4.27: Klasse KL_SIMULATION

Klasse - KL_WOHNUNG

Die Klasse KL_WOHNUNG stellt die zentrale Klasse der Smart Home Umgebung dar und vereinigt sämtliche Räume zu einer Wohnung. Sie beinhaltet fünf Instanzen vom Klassentyp KL_RAUM_TYP3 (WZ, SZ, KUECHE, BAD und BUERO) sowie eine Instanz FLUR vom Klassentyp KL_RAUM_TYP2 ohne Rollostuerung. Je nach Wohnungsmodell können die entsprechenden Rauminstanzen angepasst werden. Die Ansteuerung der Raumfunktionen wird dabei durch die Instanziierung der jeweiligen Interfaces (ILicht, IRollo, ISensor) realisiert. Die Klasse definiert die Smart Home Modi HAND, SLEEP, SECURE und SMART und sorgt für die entsprechende Ansteuerung der Räume. Die Modi werden über die Hauptvisualisierung angewählt und können je nach Bedarf vom Anwender über die Mode-Konfiguration-Visualisierung konfiguriert werden. Mit der Methode `LampenAus` werden sämtliche Lampen der Räume ausgeschaltet. Die Methode `SetAlarm` mit STRING-Übergabeparameter setzt den Ausgang Nr.8 der Zentrale aktiv. Der STRING als Auslöserinformation wird dabei visualisiert. `QuitAlarm` sorgt für ein Rücksetzen des Alarms. Die Hauptvisualisierung bietet die Möglichkeit alle Rollos gemeinsam zu steuern. Durch Aufruf der Methode `SetRollo`s mit Übergabe des entsprechenden DUT_MODE_ROLLO (AUTOAUF, AUTOZU und AUTO50) werden sämtliche Rollo-Instanzen angesteuert. Für die Ansteuerung der konfigurierten Rollo-Stellungen bei Mode-Wechsel ist die Methode `SetRollo_ModeConfig` zuständig. (Der Übergabeparameter `rolloModeConfig` mit sämtlichen Mode-Konfigurationen als Inhalt wird von der Klasse KL_MODEKONFIG in der globalen Variablenliste `GVL_Intern` bereitgestellt.) Die Methoden `SetMode_HAND`, `SetMode_SLEEP`, `SetMode_SMART` und `SetMode_SECURE` sorgen für die entsprechende Mode-Initialisierung und schalten entsprechend den MODE. Für die Simulation der Raumfunktionen im Modus Smart werden sechs KL_SIMULATION-Instanzen verwendet (`rSim[6]`). Das Ablaufdiagramm der Klasse KL_WOHNUNG ist in [Abbildung 4.29](#) dargestellt.

Folgend werden die einzelnen Modi in ihrer Standard-Konfiguration vorgestellt:

- Im initialisierten Mode-Zustand **HAND** wird der Herd-Ausgang (Ausgang Nr. 7 der Zen-

trale) aktiv. Die Temperatur-Sollwerte entsprechen 18°C . Im Wechsel werden sämtliche Rollos geöffnet.

- Mit dem Wechsel in den Modus **SLEEP** werden die Rollos zugefahren. Der Herd-Anschluss wird deaktiviert. Sämtliche Lichter werden ausgeschaltet. Die Soll-Temperaturen werden auf 16°C gesenkt. Ein automatischer Mode-Wechsel zu HAND bei Tasterbetätigung steht in der Konfiguration zur Verfügung (GVL_Intern: sleepAktWechsel).
- Im Mode **SECURE** werden die Tasterschaltungen überwacht und gegebenenfalls wird Alarm ausgelöst. Der entsprechende Raum wird dabei in der Visualisierung ausgegeben. Der Herd-Anschluss und sämtliche Lampen werden ausgeschaltet. Rollos werden auf 50% gesetzt. Die Soll-Temperatur wird auf 15°C gesenkt.
- Mit Mode **SMART** werden die Raum-Simulationen aktiviert. Im Wechsel werden Herd und Lampen ausgeschaltet. Die Tasterüberwachung ist aktiv. Zwischen 07:00 und 10:00 Uhr werden Rollos zufällig aufgefahren. Zwischen 17:00 und 21:00 werden sie auf 50% gesetzt. Alle Lampen werden unterschiedlich zwischen 20:00 und 23:00 Uhr einmalig für eine Dauer von 20min . zufällig eingeschaltet. Die Soll-Temperaturen entsprechen denen der SECURE-Konfiguration. Zur Demonstration des Modus kann die Uhrzeit mit unterschiedlichen Geschwindigkeiten simuliert werden.

Mode-Wechsel sind nur bei inaktivem Alarm möglich.

Die CoDeSys-Struktur der Klasse KL_WOHNUNG ist in folgender Abbildung dargestellt.

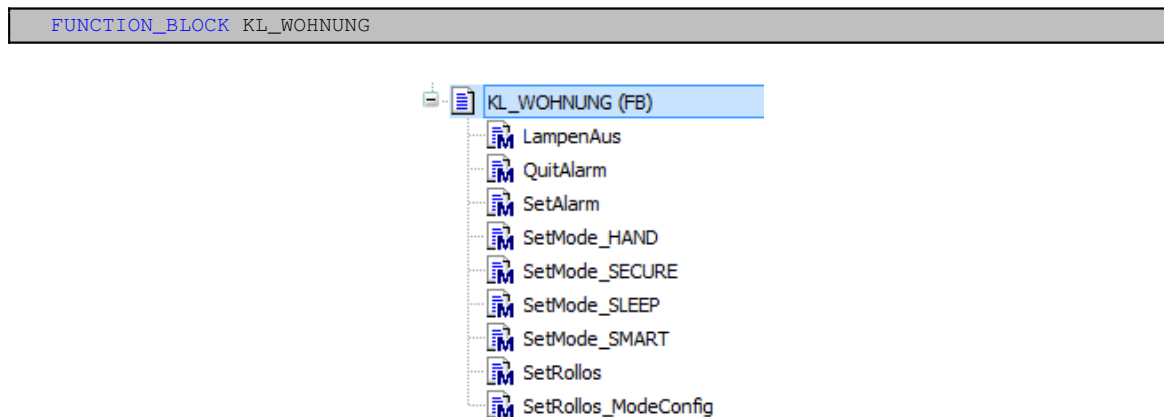


Abbildung 4.28: Klasse KL_WOHNUNG

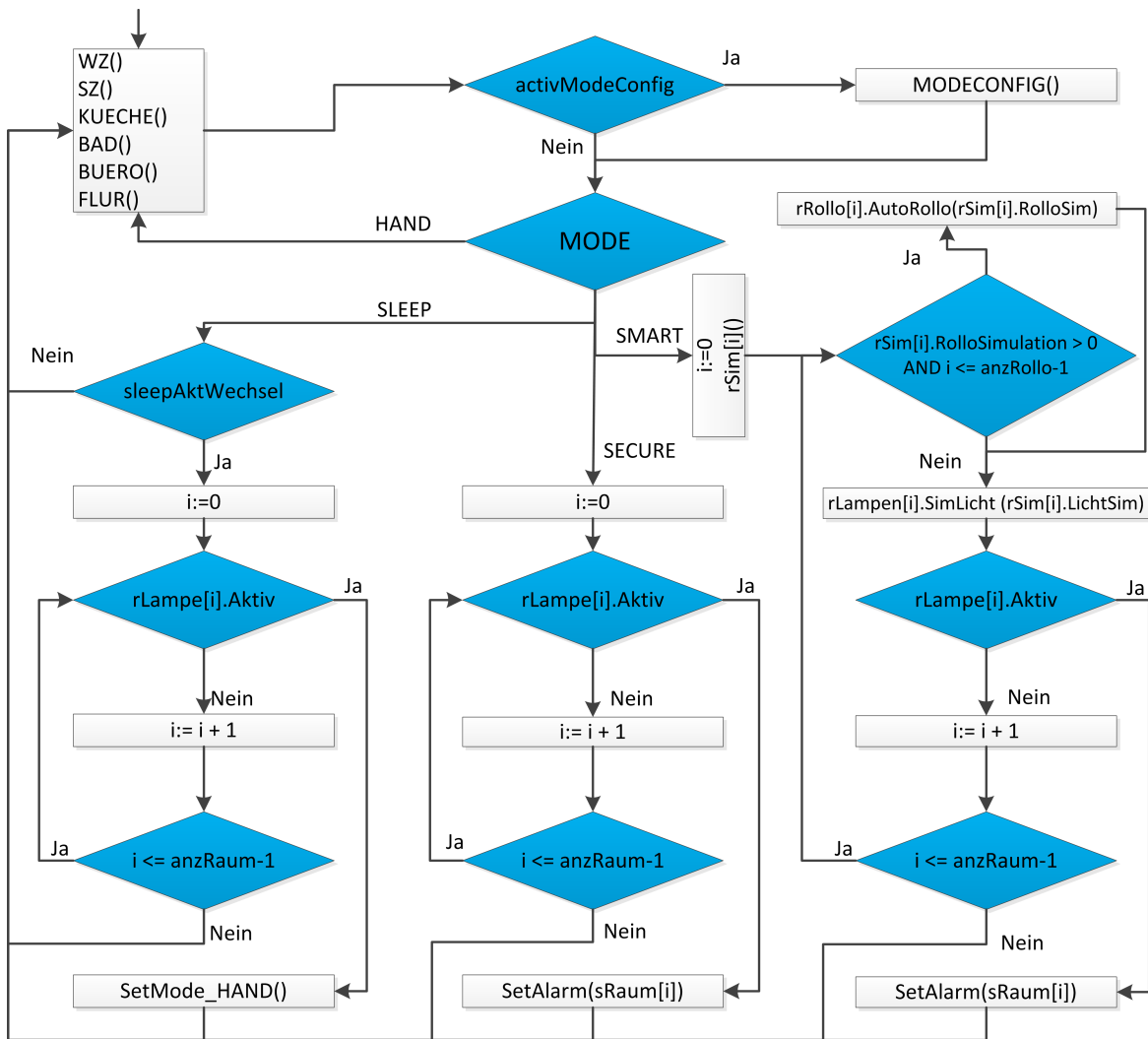


Abbildung 4.29: Ablaufdiagramm – Klasse KL_WOHNUNG

UML-Klassendiagramm

In folgender Abbildung 4.30 werden die Klassen bzw. Schnittstellen-Beziehungen anhand des UML³-Klassendiagrammes dargestellt. Die Instanziierungen zur Realisierung der Smart Home Umgebung gehen aus den Klassen-Kompositionen hervor.

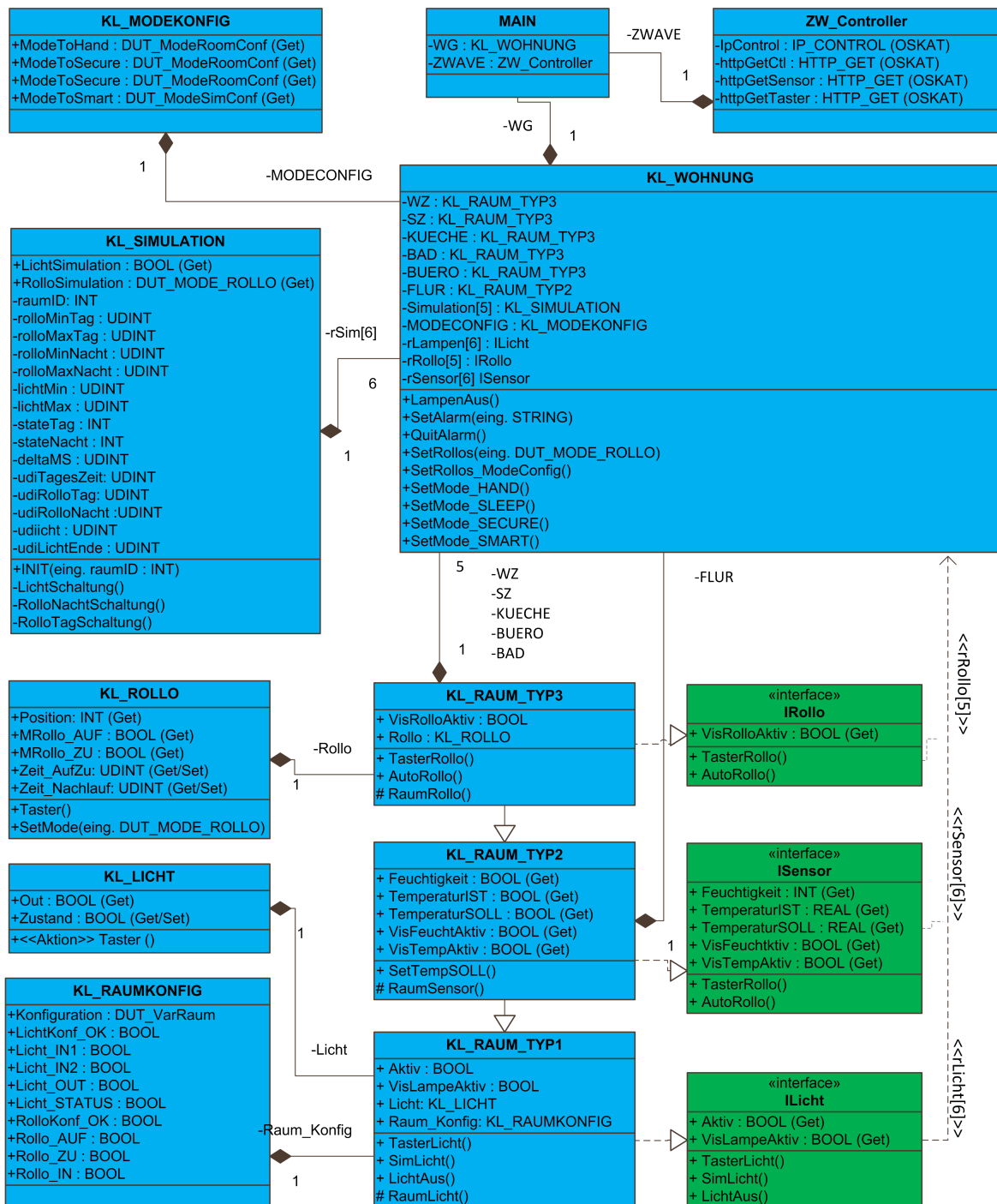


Abbildung 4.30: UML-Diagramm der Smart Home Umgebung

³Vereinheitlichte Modellierungssprache (Unified Modeling Language)

4.1.2.6 CoDeSys - Webserver

Für den Zugriff auf die Visualisierung zur Steuerung des Smart Home Systems wird der in CoDeSys integrierte Webserver konfiguriert. Über den *VisualizationManager* im Gerätebaum der CoDeSys-Übersicht wird eine sogenannte *Web-Visualisierung* hinzugefügt. Die Anwahl der Web-Visualisierung ist in folgender Abbildung 4.31 dargestellt. Der Browser kommuniziert per Java-Skript mit Unterstützung der [HTML5-Canvas](#) mit dem Webserver. Die Visualisierung kann dadurch von nahezu allen Browsern, wie auch iOS und Android geladen werden. Mit *Startvisualisierung* wird die Startseite der Visualisierung festgelegt. Die *.htm-Datei* wird mit *home* benannt. Der Zugriff aus dem Intranet erfolgt somit durch die Adresse 192.168.1.3:8080/home.htm. Die Aktualisierungsrate der Visualisierung wird auf *200ms* gesetzt. Für den *Standard-Kommunikationspuffer* soll eine Größe von *50kB* ausreichen. Mit der Skalierungsoption *Isotropisch* wird die Visualisierung automatisch proportional zur Größe des Browser skaliert.

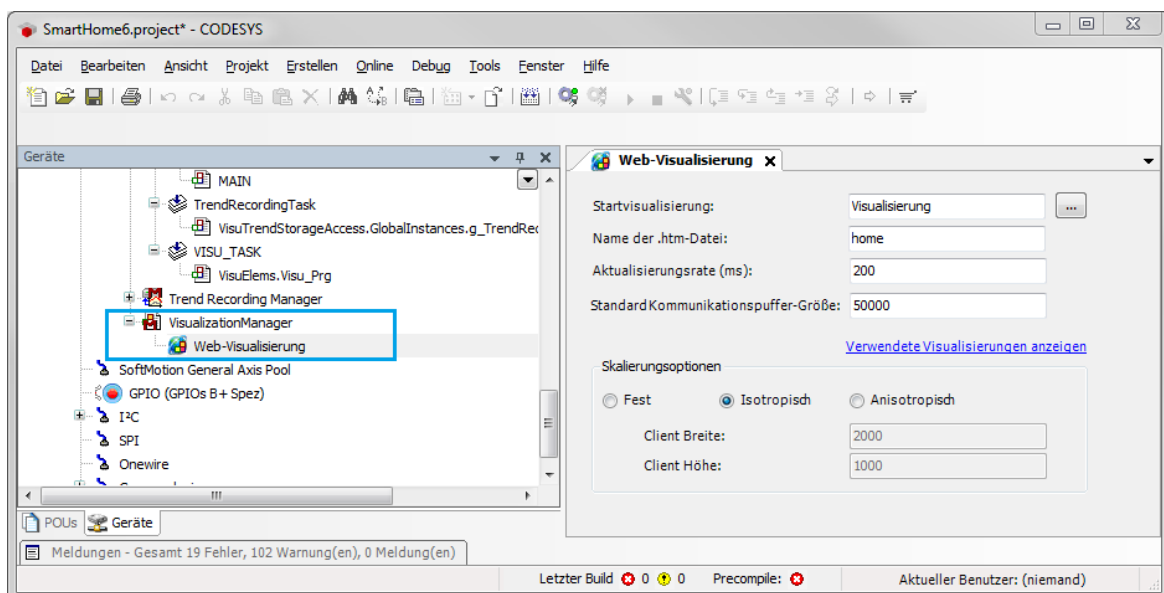


Abbildung 4.31: CoDeSys - Webserver

4.1.2.7 CoDeSys - Visualisierung

Folgend werden die Haupt-Visualisierungsseiten der Smart Home Umgebung vorgestellt. Die einzelnen Seiten bestehen aus mehreren Visu-Objekten die einmalig erstellt und durch das Visu-Werkzeug *Frame* mehrfach instanziiert und mit entsprechenden Variablen-Strukturen referenziert und erhalten somit ihre Funktion.

Hauptseite Smart Home

Das Visu-Objekt „*Smarthome*“ stellt die Hauptseite der Visualisierung dar und wird mit dem Klassentyp KL_WOHNUNG referenziert. Folgende Abbildung 4.32 stellt ein unvollständig konfiguriertes Smart Home System im Mode *HAND* dar. Im linken Steuerbereich werden die Klassenfunktionen bedient. Die Buttons Temperaturverläufe, Konfiguration sowie Konfiguration im Modi-Bereich führen einen Seitenwechsel durch. Im Wohnungsbereich werden die einzelnen Raumfunktionen angesteuert. Der Raum Wohnzimmer (WZ) visualisiert mit dunkelrotem Rollobutton aktuell ein geschlossenes Rollo sowie eine aktive Beleuchtung. Im Küchenbereich ist ein aktiver Schließvorgang des Rollos dargestellt. Der aktive Herdanschluss ist über die grüne Herd-Umrandung erkennbar. Durch die fehlende Konfiguration des Raumes BUERO, fehlen sämtliche funktionale Visualisierungselemente. Je nach Sensormodul werden die Temperatur-Sollwerte (weiß), die Temperatur-Istwerte (grün) und die Luftfeuchtigkeit (türkis) dargestellt.



Abbildung 4.32: Visu – Hauptseite Smart Home

Seite Konfiguration

Über den Button Konfiguration auf der Hauptseite erfolgt der Wechsel zur Konfigurations-Seite. Im linken Bereich werden die einzelnen Räume selektiert. Als Beispiel wird in folgender Abbildung 4.34 die Konfigurations-Möglichkeit des Raumes *Wohnzimmer* dargestellt. Für die *Lichtschaltung* lassen sich zwei Eingangsmöglichkeiten sowie ein Ausgang konfigurieren. Die zweifachen Eingänge ermöglichen die Nutzung von Eingangs-Anschlüssen unterschiedlicher Module. Im ersten Scrollbalken wird der Modul-Typ (ZENTRALE, FUNK oder ZWAVE) angewählt. Je nach Auswahl steht im zweiten Scrollbalken die Modul-Adresse bzw. die Ein- oder Ausgangs-Nr. zur Auswahl. Bei Modul-Typ FUNK erfolgt die Ein- bzw. Ausgangskonfiguration mit dem dritten Scrollbalken. Nach demselben Prinzip werden der Eingang sowie die Ausgänge zur Rolloststeuerung konfiguriert.

Konfiguration: Wohnzimmer

Räume

- Wohnzimmer
- Schlafzimmer
- Küche
- Bad
- Büro
- Flur

Lichtschaltung:

FUNK	Modul 1	Eingang 1	Input 1
FREI	Modul 1	Eingang 1	Input 2
ZENTRALE	Ausgang 1		Output

Rolloschaltung:

FREI	Modul 1	Eingang 1	Input
FREI	Modul 1	Ausgang 1	AUF
FREI	Modul 1	Ausgang 1	ZU

Auf/Zu -Zeit: 10000 MS Nachlaufzeit: 1000 MS

Sensor:

Z-Wave	Modul 1	Input
--------	---------	-------

OK

Abbildung 4.33: Visu – Seite Konfiguration

Für die Eingabe der Zeitwerte der Rolloschaltung wird eine sogenannte Faceplate-Technik angewandt, die eine angenehme Eingabe per Touchpanel ermöglicht. Bei Berührung der Zeit-Werte öffnet sich ein Dialog-Fenster, welches mit den entsprechenden Zeit-Werten re-

ferenziert ist. Über die Pfeil-Optionen wird der Wert in 200ms-Schritten verändert und mit dem Button *Schließen* übernommen.



Abbildung 4.34: Dialog – Zeiteingabe

Seite Modi-Konfiguration

Über den Button Konfiguration im Modebereich auf der Hauptseite erfolgt der Wechsel zur Mode-Konfigurations-Seite. Sämtliche Rollo- bzw. Lichtsteuerungen bei Wechsel in einen bestimmten Mode können über sogenannte Checkboxes für jeden Raum individuell konfiguriert werden. Die einzelnen Checkbox-Gruppen der einzelnen Modi beliefern die KL_MODEKONFIG-Instanz der Raumklasse mit den entsprechenden Konfigurationsdaten. Mit dem OK-Button gelangt man auf die Hauptseite zurück.

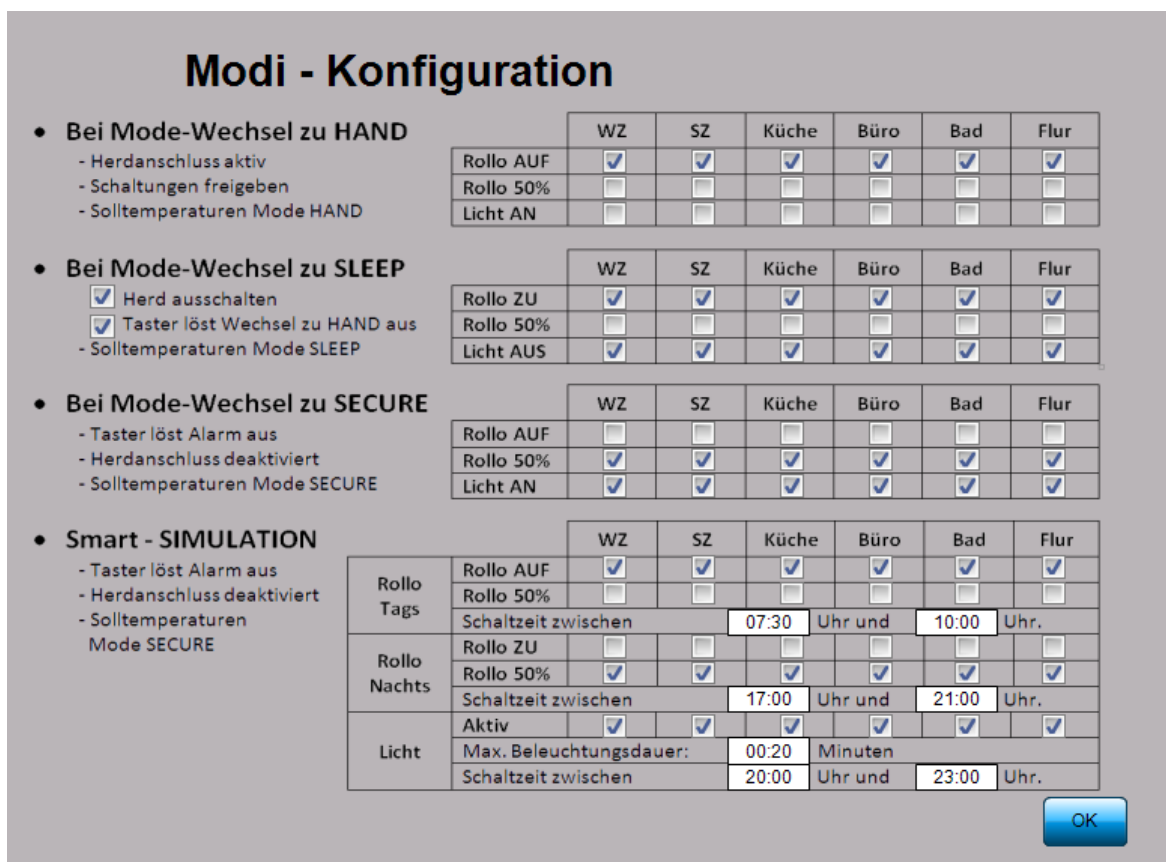


Abbildung 4.35: Visu – Modi Konfiguration

Für die Eingabe von Dauer bzw. der Uhrzeit werden die abgebildeten Dialoge aus Abbildung A.6 verwendet. Die Dauer lässt sich in 5 *min*-Schritten festlegen. Die Angabe der Uhrzeit erfolgt in 15 *min*-Schritten. Die Variablen sowie die Fenster-Überschrift werden entsprechend referenziert.

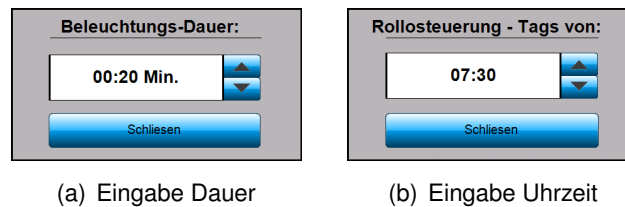


Abbildung 4.36: Dialog – Eingabe

Seite Temperaturverläufe

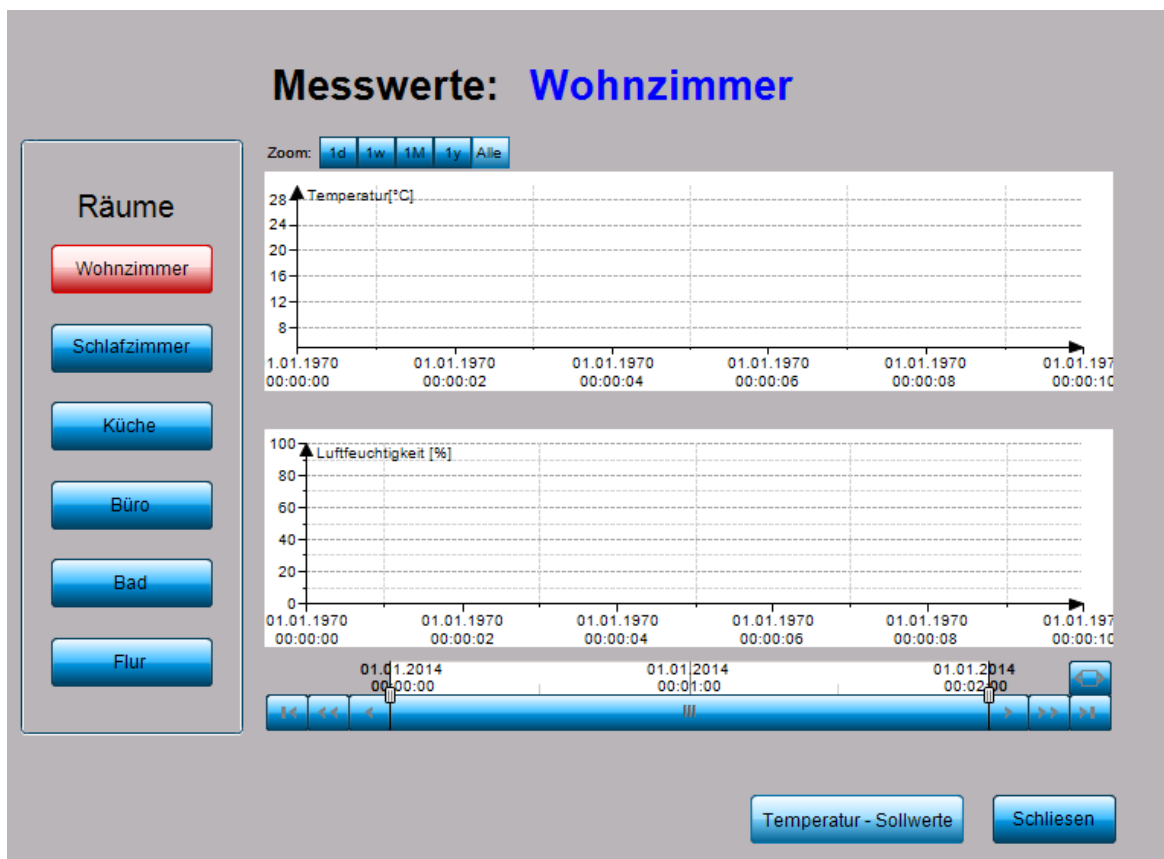


Abbildung 4.37: Visu – Temperaturverläufe

Über den Button Temperaturverläufe der Smart Home Seite erfolgt der Aufruf zur Messdaten-Übersicht. Auf der linken Seite stehen die Räume zur Auswahl zur Verfügung. Sofern die Sensorik konfiguriert wurde, werden die vorhandenen Messdaten über ein Kurvendiagramm visualisiert. Über den Button *Schließen* wird die Seite geschlossen. Mit dem Button *Soll-Temperatur* wird das Wochenprogramm der Sollwerte aufgerufen. Die Visualisierung der Messdaten wird dabei mit dem Visu-Element *Trend* in CoDeSys realisiert. Die Messdaten werden entsprechend dem *TrendRecordingTask* auf der SD-Karte des Raspberry archiviert.

Seite Temperatur-Sollwerte

Die Seite stellt das Wochenprogramm der einzelnen Modi zur Ansteuerung der Temperatur-Sollwerte dar (Abb. 4.38). Bei Berührung eines Temperaturwertes öffnet sich ein Dialog-Fenster das die Eingabe der Soll-Temperaturwerte des bestimmten Wochentages ermöglicht. Die Temperatur-Werte lassen sich in 0.1° Grad-Schritten einstellen. Durch Schließen des Fensters werden die Daten im Wochenprogramm übernommen.

Temperatur - Sollwerte

Wohnzimmer	Mo.	Di.	Mi.	Do.	Fr.	Sa.	So.	Feiertag
Hand	18.0 °C	18.0 °C	18.0 °C	18.0 °C	19.0 °C	19.0 °C	18.0 °C	19.0 °C
Sleep	16.0 °C	16.0 °C	16.0 °C	16.0 °C	17.0 °C	17.0 °C	16.0 °C	17.0 °C
Secure	15.0 °C	15.0 °C	15.0 °C	15.0 °C	16.0 °C	16.0 °C	15.0 °C	16.0 °C

Schlafzimmer	Mo.	Di.	Mi.	Do.	Fr.	Sa.	So.	Feiertag
Hand	18.0 °C	18.0 °C	18.0 °C	18.0 °C	19.0 °C	19.0 °C	18.0 °C	19.0 °C
Sleep	16.0 °C	16.0 °C	16.0 °C	16.0 °C	17.0 °C	17.0 °C	16.0 °C	17.0 °C
Secure	15.0 °C	15.0 °C	15.0 °C	15.0 °C	16.0 °C	16.0 °C	15.0 °C	16.0 °C

Küche	Mo.	Di.	Mi.	Do.	Fr.	Sa.	So.	Feiertag
Hand	18.0 °C	18.0 °C	18.0 °C	18.0 °C	19.0 °C	19.0 °C	18.0 °C	19.0 °C
Sleep	16.0 °C	16.0 °C	16.0 °C	16.0 °C	17.0 °C	17.0 °C	16.0 °C	17.0 °C
Secure	15.0 °C	15.0 °C	15.0 °C	15.0 °C	16.0 °C	16.0 °C	15.0 °C	16.0 °C

Bad	Mo.	Di.	Mi.	Do.	Fr.	Sa.	So.	Feiertag
Hand	18.0 °C	18.0 °C	18.0 °C	18.0 °C	19.0 °C	19.0 °C	18.0 °C	19.0 °C
Sleep	16.0 °C	16.0 °C	16.0 °C	16.0 °C	17.0 °C	17.0 °C	16.0 °C	17.0 °C
Secure	15.0 °C	15.0 °C	15.0 °C	15.0 °C	16.0 °C	16.0 °C	15.0 °C	16.0 °C

Büro	Mo.	Di.	Mi.	Do.	Fr.	Sa.	So.	Feiertag
Hand	18.0 °C	18.0 °C	18.0 °C	18.0 °C	19.0 °C	19.0 °C	18.0 °C	19.0 °C
Sleep	16.0 °C	16.0 °C	16.0 °C	16.0 °C	17.0 °C	17.0 °C	16.0 °C	17.0 °C
Secure	15.0 °C	15.0 °C	15.0 °C	15.0 °C	16.0 °C	16.0 °C	15.0 °C	16.0 °C

Flur	Mo.	Di.	Mi.	Do.	Fr.	Sa.	So.	Feiertag
Hand	18.0 °C	18.0 °C	18.0 °C	18.0 °C	19.0 °C	19.0 °C	18.0 °C	19.0 °C
Sleep	16.0 °C	16.0 °C	16.0 °C	16.0 °C	17.0 °C	17.0 °C	16.0 °C	17.0 °C
Secure	15.0 °C	15.0 °C	15.0 °C	15.0 °C	16.0 °C	16.0 °C	15.0 °C	16.0 °C

Abbildung 4.38: Visu – Temperatur-Sollwerte

4.1.3 Ergebnis

Nach erfolgreicher Testphase der Smart Home Zentrale wurde der Schaltungsaufbau mit EAGLE, zur Herstellung der Platine, realisiert. Der Schaltplan mit Geräteliste und Platinenlayout befindet sich im Anhang A.1. Für das reine Modul wurde ein Warenwert für Hard- und Software von €137,28 eingesetzt. Folgend ist das Ergebnis der Smart Home Zentrale abgebildet:

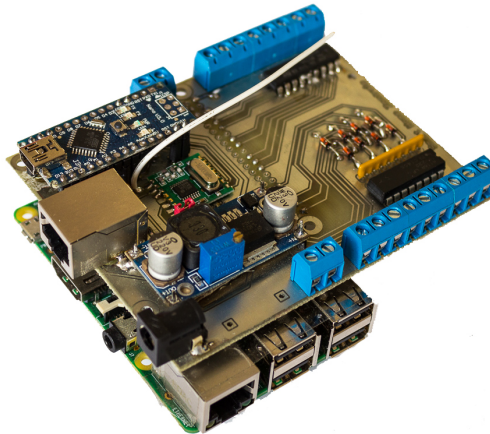


Abbildung 4.39: Smart Home Zentrale

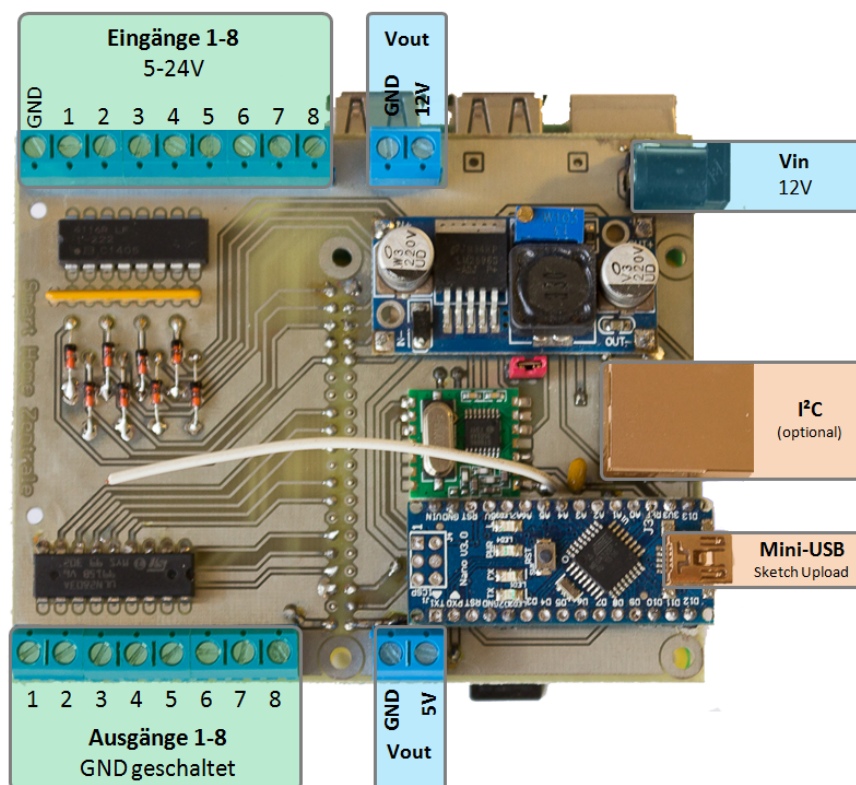


Abbildung 4.40: Anschlüsse

4.2 Funk IO-Modul

4.2.1 Hardware

Zur Erweiterung der Smart Home Zentrale wurde ein Funk IO-Modul mit 16 Eingängen sowie 16 Ausgängen zur Ansteuerung von Relais spezifiziert. Herzstück des Moduls bildet der unter Punkt 2.2 vorgestellte Mikrocontroller *Arduino* in der Bauweise *UNO*. Zur Funkübertragung wird die bereits unter Punkt 4.1.1.5 entwickelte Nutzung des RFM12B-Funkmodules eingesetzt. Ergebnis ist ein Aufsatz des Arduino UNO, der die Spezifikation des IO-Moduls erfüllt.

4.2.1.1 Spannungsversorgung

Die Spannungsversorgung des Arduino UNO Mikrocontroller lässt einen empfohlenen Spannungsbereich von 6 – 12V zu. Als gemeinsame Speisung erfolgt die Spannungsversorgung über eine entsprechende Buchse auf der Platine, welche den Arduino über den VIN-Eingang mit Spannung versorgt. Es wird ein AC-DC Netzteil mit 12V und 2A verwendet. Für Bauteile unterschiedlicher Spannungen besitzt der Arduino jeweils einen geregelten Ausgang mit 5V sowie 3.3V. [22]

4.2.1.2 IO-Erweiterung

Das IO-Modul wurde mit 16 Eingängen sowie 16 Ausgängen spezifiziert. Da der Arduino lediglich über 14 digitale Ein-/Ausgänge verfügt, wird der I²C-Baustein MCP23017-E als Port-Erweiterung eingesetzt. Der MCP23017 Portexpander verfügt über 2 Ports (GPA und GPB) mit jeweils 8 digitalen Ein-/Ausgängen (Abb. 4.41). Die Spannungsversorgung erfolgt über VDD (5V) sowie VSS (GND). Über die Anschlüsse A0, A1 und A2 wird die I²C-Adresse festgelegt. Somit sind maximal 8 Expander-Bausteine im selben I²C Bus adressierbar. Für das IO-Modul werden zwei MCP2317-Bausteine eingesetzt, welche die gesamten 32 notwendigen IO-Signale des Moduls liefern. Der RESET-Eingang (negative Logik) ermöglicht eine externe Neuinitialisierung des Bausteins. Da dies nicht verwendet wird, wird der Eingang auf Spannung gelegt. Die Anschlüsse SCL bzw. SDA stellen den Zugang des I²C-Buses dar. Die Interrupt-Ausgänge INTA bzw. INTB bieten die Möglichkeit eine Interrupt-Routine des verwendeten Mikrocontrollers bei Eingangsänderung auszulösen. [1][24]

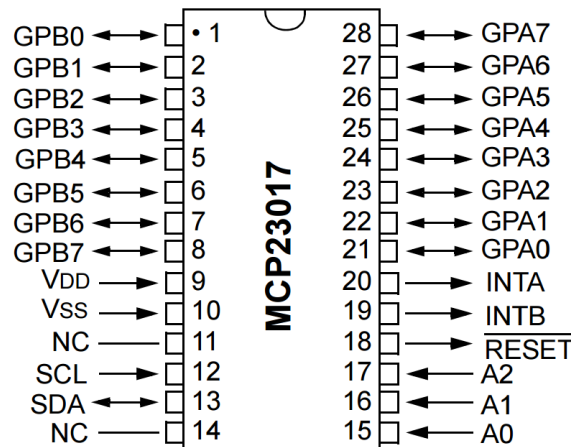


Abbildung 4.41: Portexpander MCP23017 [24]

4.2.1.3 Ein-/Ausgänge

Ähnlich der Eingangsspezifikation der Zentrale soll auch das IO-Modul über spannungstolerante Eingänge für einen Spannungsbereich von 5 bis 24V verfügen. Wie zuvor unter IO-Erweiterung (4.2.1.2) erläutert, werden die Eingänge über den MCP23017 Portexpander detektiert. Der Portexpander definiert, bei einer Betriebsspannung von 5V, Eingangsspannungen zwischen 0 und 1V als LOW-Pegel sowie Spannungen zwischen 4 und 5V als HIGH-Pegel [24]. Für die Eingangspegelwandlung wird eine Zener-Diode mit einer Durchbruchspannung von 4.7V gewählt. In folgenden Abbildungen sind das PSpice-Modell (Abb.4.42) sowie das Ergebnis der Simulation (Abb.4.43) mit den Widerständen $R1 = 2.2k\Omega$ und $R2 = 47k\Omega$ abgebildet. Die maximale Verlustspannung bei einer maximal definierten Eingangsspannung von 24V beträgt 170mA.

Die als Ausgang genutzten Ports des MCP23017 liefern eine Ausgangsspannung von 5V und sind maximal mit einem Strom von 25mA belastbar. Zur Verstärkung der Ausgänge wird wiederum der ULN2803A-Baustein eingesetzt (Punkt 4.1.1.3). Er ermöglicht bei einer maximalen Strombelastbarkeit von 500mA je Ausgang, die Ansteuerung von 12V-Relais und sorgt durch die integrierte Freilaufdiode für eine Kompensation der eventuellen Induktions-Rückspannungen. [24]

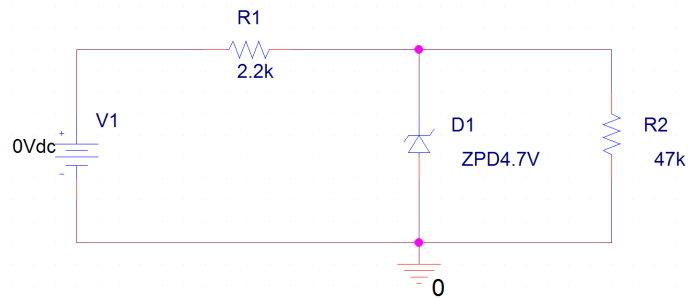


Abbildung 4.42: PSpice Model mit Zener-Diode ZPD4.7V

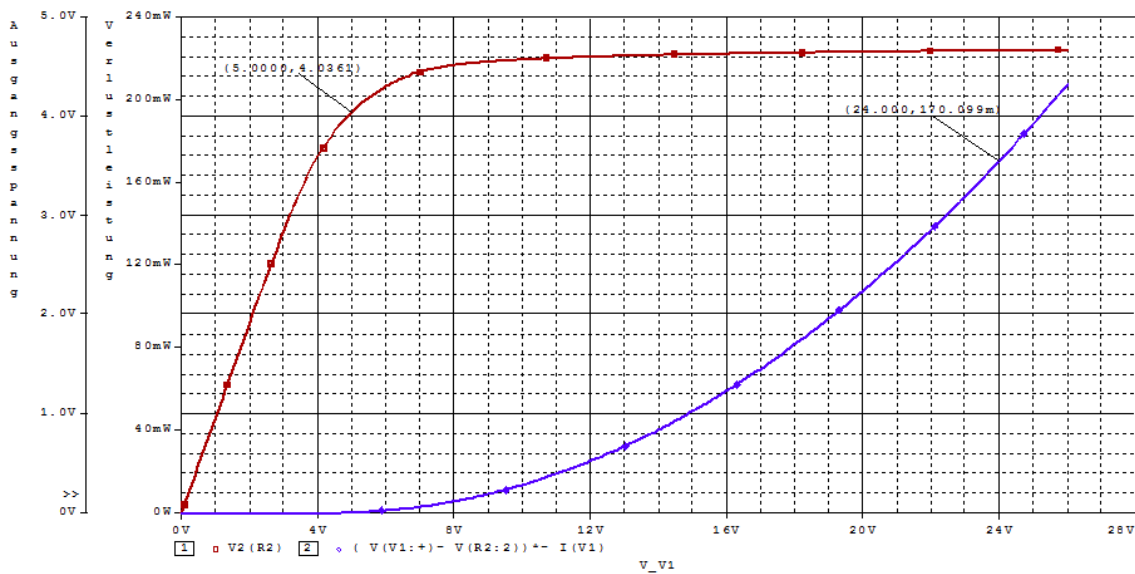


Abbildung 4.43: PSpice Simulation – Pegelwandlung 4.7V

4.2.2 Software

Für die Umsetzung des Arduino-Sketches werden wiederum die Arduino-Bibliotheken `<RFM12B.h>` sowie `<Wire.h>` zur Nutzung des I²C-Buses implementiert. Die Anwendung des RFM12B-Modules wurde bereits zur Umsetzung des Funk-Controllers der Smart Home Zentrale (siehe Seite 46) vorgestellt und wird analog dazu mit Funkadresse `NODEID=1` eingesetzt. Im Gegensatz zum Funk-Controller wird der I²C-Bus folgend als Master zur Ansteuerung der Portexpander `MCP23017` verwendet. In folgendem Code-Ausschnitt in Abbildung 4.44 ist die Initialisierung des Portexpanders 1 über I²C abgebildet.

Mit der Funktion `Wire.begin()` wird der I²C-Bus als Master-Teilnehmer initialisiert. Mit der Initialisierung des *MCP23017*-Bausteins werden dessen Port-Eigenschaften als Eingang bzw. Ausgang definiert. Die Bibliothek-Funktion `Wire.beginTransmission()` initiiert mit einer 7-Bit-Adressierung als Übergabeparameter den entsprechenden Slave zur Kommunikation. Der Portexpander 1 wurde hardwareseitig mit der Adressierung $A0 = A1 = A2 = \text{LOW}$ (GND) festgelegt. Bei Betrachtung der *Control Byte* -Definition des Bausteins (Abb. 4.45) ergibt sich durch die fixen Bits, die durch die Baustein-Eigenschaften festgelegt sind, eine Adresse von 0x20 (Portexpander 2 = 0x21). Mit dem Schreibzugriff `Wire.write(0x00)` wird das Port-Initialisierungsregister A des Portexpanders selektiert und folgend werden mit `Wire.write(0xFF)` alle 8 Ein-/Ausgänge als Eingänge definiert. Mit `Wire.endTransmission()` wird die Port-Initialisierung beendet. Im folgenden Teil des Code-Ausschnittes wird durch 0x01 das Port-Initialisierungsregister B mit 0x00 als Ausgang definiert. [1][24]

```
// I2C-Bus aktivieren
Wire.begin();

// Portexpander MCP23017-1 Port A initialisieren
Wire.beginTransmission(addressMCP1); // I2C-Adresse (0x20)
Wire.write(ODIRA); // IO-Register A (0x00)
Wire.write(0xFF); // All pins to input
Wire.endTransmission();

// Portexpander MCP23017-1 Port B initialisieren
Wire.beginTransmission(addressMCP1); // I2C-Adresse (0x20)
Wire.write(ODIRB); // IO-Register B (0x01)
Wire.write(0x00); // All pins to output
Wire.endTransmission();
```

Abbildung 4.44: MCP23017 – Initialisierung

Das Setzen bzw. Auslesen der entsprechenden Port- Ein-/Ausgänge des Portexpanders, soll durch folgendes Code-Beispiel (Abb. 4.46) erläutert werden. Mit `Wire.beginTransmission()` wird der entsprechende Baustein angewählt. Die `write`-Methode versendet die Adresse des Port A (0x12), was quasi als Zeiger arbeitet. Durch den Aufruf der `requestFrom()`-Methode mit entsprechender Adresse wird der Status des Port A mit einer Größe von 1Byte (8Bit) abgefragt und folgend mit `Wire.read()` in die Variable `inpA` eingelesen. [1]

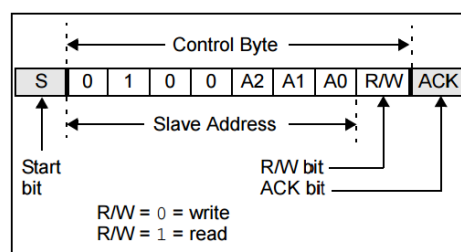


Abbildung 4.45: MCP23017 – Adressierung [24]

Zur Ansteuerung der Ausgänge wird analog zur Eingangsabfrage das Portregister B (0x13) des Portexpanders selektiert. Mit folgender Übertragung des binären Wertes 11111111 werden beispielhaft sämtliche Port-Ausgänge auf HIGH gesetzt.

```

// Eingänge MCP1 abfragen
Wire.beginTransmission(addressMCP1);
Wire.write(GPIOA);           // Register A (0x12)
Wire.endTransmission();
Wire.requestFrom(addressMCP1,1);
inpA = Wire.read();
...
// Ausgänge setzen
Wire.beginTransmission(addressMCP1);
Wire.write(GPIOB);           // Register B (0x13)
Wire.write(11111111);
Wire.endTransmission();
...

```

Abbildung 4.46: MCP23017 – Register-Zugriff

Im vorangegangenen Code-Auszug wurde der Port A in die BYTE-Variablen `inpA` eingelesen. Das Bit 0 entspricht dabei dem Status des Eingangs GPA0 bzw. Bit 7 dem von GPA7 (Abbildung 4.41). Da die Leiterbahn-Planung in EAGLE eine Umsortierung der Reihenfolge erfordert, wird eine Korrektur durchgeführt, wodurch die aufsteigende Reihenfolge der tatsächlichen Hardware-Anschlüsse des Moduls entspricht. Eine Sortierung wird mit den bitweisen Zugriffsmethoden von Arduino vollzogen. Folgendes Beispiel liest Bit 3 aus der Variable `inpA` und schreibt sie an die zweite Bit-Stelle der Variable `out`:

```
bitWrite(out, 1, bitRead(inpA, 3))
```

Folgendes Struktogramm in Abbildung 4.47 stellt den funktionellen Programmablauf des Arduino-Sketches dar. Mit `void loop()` erfolgt die Initialisierung des Funkmoduls mit aktiver Verschlüsselung, sowie der Portexpander MCP23017 mittels des I²C-Bus. Mit `void loop()` werden die fehlerfreien Empfangsdaten des Funkmoduls kontinuierlich ausgewertet. Für eine ACK-Anforderung wird dem entsprechenden Absender ein ACK-Signal als Bestätigung gesendet. Bei enthaltenem Steuer-Bit wird der entsprechende Ausgang gesetzt bzw. rückgesetzt. Eine Änderung der Ausgänge bringt eine Sendung der aktuellen Port-Zustände zur Zentrale mit sich. Die Eingänge der Portexpander werden im Intervall von 400ms auf Aktivität überprüft. Bei vorliegendem Steuerbefehl wird das aktuelle Abbild der Port-Zustände an die Zentrale gesendet.

Tabelle 4.2: Funk – Datenstruktur

BYTE	0	1	2	3
Sendestruktur	INT statusOUT		INT steuerOUT	
Empfangsstruktur	INT steuerIN		INT resIN	

Im Anhang A.4.1 wird unter anderem die Sendefunktion (`Send(nodeid, sendbuf, sendLen, groupid)`) des Funkmoduls beschrieben. In der Anwendung des IO-Moduls wird hierbei die `nodeid=31` des Empfängers (Zentrale), sowie mit `groupid` die Adresse des Funk-Netzwerkes 212 verwendet. Mit dem Parameter `sendBuf` wird die Daten-Sendestruktur aus Tabelle 4.2 als Abbild der Zustände übergeben. Die Variable `statusOUT` (16Bit) beschreibt die 8 Ausgänge des PortB des Portexpanders 1 sowie die des PortA des Portexpanders 2. Die Eingänge des PortA (Portexpanders 1) und PortB (Portexpander 2) werden mit der Strukturvariable `steuerOUT` (16Bit) beschrieben. Der Parameter `sendLen` gibt mit `32Bit` die Bitlänge der Informations-Daten an. Mit der Empfangsstruktur werden über die Funktion `data()`, mit `steuerIN` (16Bit) die Steuerbefehle der Port-Ausgänge ausgewertet.

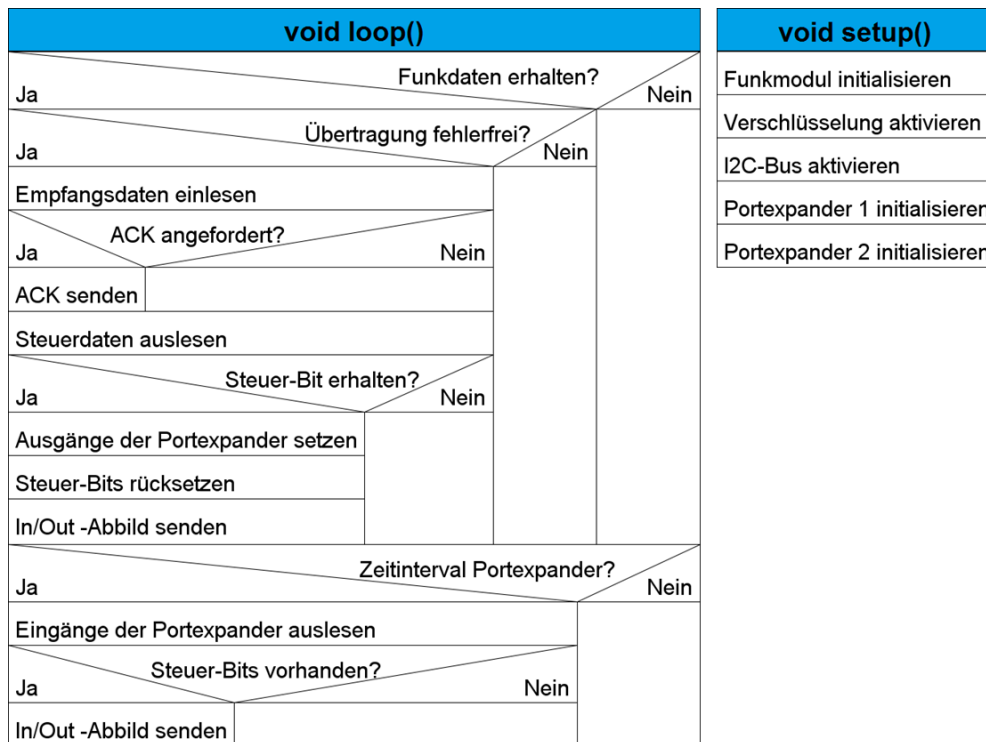


Abbildung 4.47: Struktogramm – Funk IO-Modul

4.2.3 Ergebnis

Nach erfolgreicher Testphase des Testaufbaues des Funk-IO-Moduls wird das Platinenlayout mittels EAGLE entworfen. Der Schaltplan mit Geräteliste und Platinenlayout befindet sich im Anhang A.2. Zur Erstellung des Moduls wurde ein Warenwert von €58.84 benötigt. Folgend ist das Ergebnis des Funk IO-Moduls abgebildet:

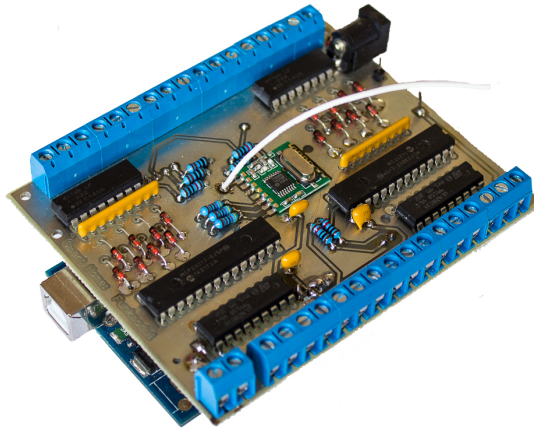


Abbildung 4.48: Funk IO-Modul

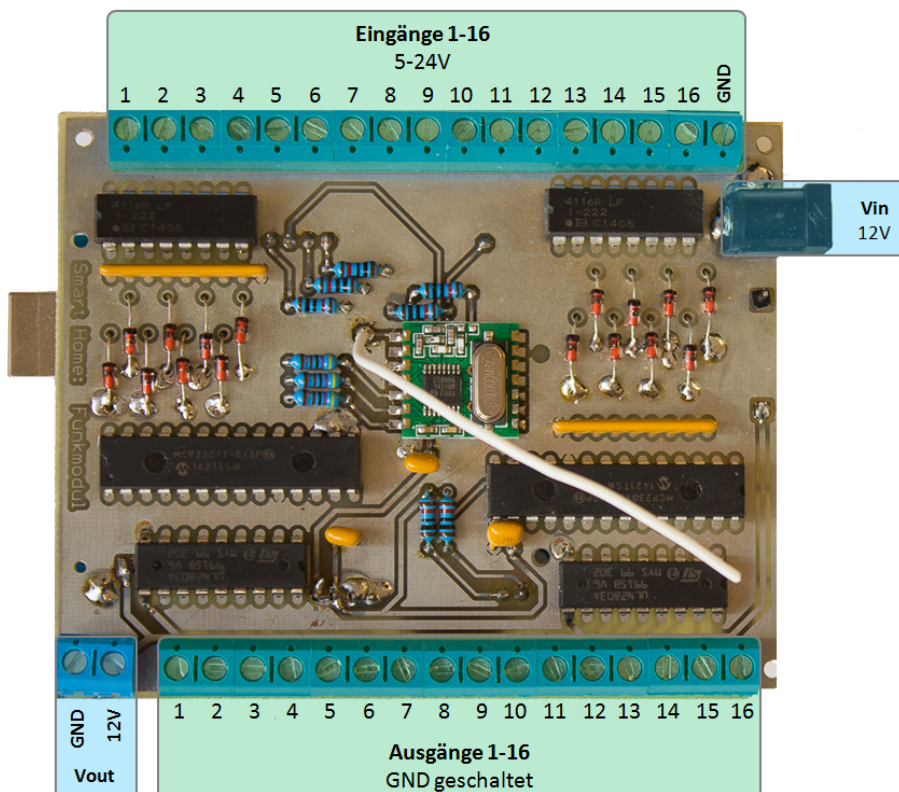


Abbildung 4.49: Anschlüsse

4.3 Funk Sensor-Modul

Es wurde ein Sensor-Modul zur Messung von Temperatur, Luftfeuchtigkeit sowie Helligkeit spezifiziert. Das Modul soll mit Batterieversorgung eine Lebensdauer von min. einem Jahr erreichen.

4.3.1 Hardware

Herzstück des Sensor-Moduls bildet der Arduino Mikrocontroller in der Bauversion Mini Pro (Abb.4.50). Der Mini Pro mit minimaler Ausstattung ist als einzige Arduinoversion für eine Spannungsversorgung von $3.3V$ und $8MHz$ Taktfrequenz erhältlich und bildet somit nach der Formel der Verlustleistung $P = V_{cc}^2 * f_T$ eine energieeffiziente Grundlage. Wie der Arduino Uno ist er mit dem CHIP ATmega328 bestückt. Durch die kompakte Version verzichtet er gänzlich auf Steckverbinder und verfügt über keinen USB-Port. Zur Programmierung muss ein USB-TTL⁴-Kabel eingesetzt werden. Besagtes Kabel beinhaltet einen sogenannten FTDI⁵-Konverter, der das USB-Signal auf den TTL-Ausgang konvertiert. Der TTL-Ausgang besteht dabei aus 4 Ausgängen (Rot:Vout, Schwarz:GND, Weiss:RX-Singal, Grün: TX-Signal). Die gleichnamige, serielle Schnittstelle des Arduino Mini mit Receive- (RX) und Transmit-Signal (TX) werden dabei über Kreuz verbunden.

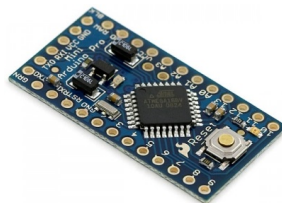


Abbildung 4.50: Arduino Mini Pro

4.3.1.1 Spannungsversorgung

Als Spannungsversorgung werden vier handelsübliche Batterien vom Typ Mignon AA eingesetzt. Eine 4-fach-Batteriehalterung mit Reihenschaltung der Batterien ergibt eine Gesamtspannung von $6V$ mit einer Kapazität von $2500mAh$. Bezogen auf die angestrebte Lebensdauer von einem Jahr ergibt sich eine mittlere, maximale Stromaufnahme von $\frac{2500mAh}{365*24h} = 285\mu A$. Der Betrieb des ATmega 328P des Arduino Mini Pro verspricht im Schlafmodus eine maximale Stromaufnahme von $2.6\mu A$ [23], wodurch die Einhaltung der mittlere, maximale Stromaufnahme durch geeignete Intervall-Sensormessung erreicht werden sollte.

⁴Transistor-Transistor-Logik

⁵Future Technology Devices International

4.3.1.2 Adressierung

Laut Spezifikation der Smart Home Zentrale, soll die Zentrale um 8 Sensor-Module erweiterbar sein. Durch den Einsatz von sogenannten **DIP⁶**-Schaltern soll eine manuelle Vergabe der Adresse ermöglicht werden. Es werden 3 DIP-Schalter verwendet, die je nach Schalterstellung die binäre Adresse von 0 bis 7 darstellen. Die Schalter werden über einen digitalen Ausgang des Arduinos versorgt. Deren Schalterstellung wird über drei digitale Eingänge erkannt. Der Aufbau der Schaltung ist im Anhang [A.7](#) zu entnehmen.

4.3.1.3 Sensoren

DHT22 Temperatur- und Feuchtigkeitssensor

Zur Messung der Temperatur und Feuchtigkeit wird der kostengünstige, digitale Messsensor DHT22 eingesetzt. Der Sensor verfügt über einen kapazitiven Feuchtigkeitfühler sowie einen Thermistor und misst die umliegende Luft. Die Messdaten werden über den Ausgangs-Pin DATA als digitales Signal ausgegeben. Der DHT22 weist eine Messgenauigkeit der Temperaturmessung bei $\pm 0.5^{\circ}\text{C}$ sowie der Feuchtigkeitsmessung bei $\pm 2\% \text{H}$ auf. Der Sensor wird mit einer Betriebsspannung von $3.3 - 6\text{V}$ gespeist. Die Stromaufnahme beträgt im Messbetrieb max. 1.5mA . [\[25\]](#)

Fotodiode

Die Helligkeit wird anhand einer einfachen Fotodiode gemessen. Es wird die Fotodiode vom Typ PFW351 eingesetzt. Diese wird über den VCC-Ausgang (3.3V) des Arduinos versorgt und über den analogen Eingang A0 ausgewertet. In Abhängigkeit zur Lichtstärke verändert die Diode ihren Widerstand. Zur Spannungsteilung wurde zudem ein Widerstand von $R = 10\text{k}\Omega$ in Reihenschaltung zur Diode eingesetzt. Die Schaltung ist im Anhang [A.3](#) dargestellt.

4.3.2 Software

4.3.2.1 TTL-Treiber einrichten

Wie bereits erwähnt ist für die Beschreibung des *Arduino Mini Pro* ein USB-TTL-Kabel notwendig. Das verwendete Kabel konvertiert dabei die Signale in beide Richtungen mittels des sogenannten **FTDI**-Chip. Auf dessen Webauftritt (ftdichip.com) sind Treibersoftware für die gängigen Betriebssysteme erhältlich. Nach erfolgreicher Installation des Treibers wird der per USB-TTL-Kabel verbundene Arduino Mini auf dem Entwicklungsrechner erkannt.

⁶Bauform parallel angeordneter Anschlussreihen(Dual in-line package)

Für eine erfolgreiche Datenübertragung muss der Treiber entsprechend konfiguriert werden. Hierfür wird der erkannte *USB Serial Port* im Geräte-Manager selektiert und dessen Eigenschaften bearbeitet. Unter dem Reiter Anschlussinstellungen werden die seriellen Übertragungseigenschaften des Arduinos nach Abb. 4.51 konfiguriert (8 Datenbits, keine Parität, 1 Stoppbit). Die verwendete Baudrate muss dabei mit der im Arduino-Sketch definierten Baudrate übereinstimmen. [4]

Die Übertragung des Sketches aus der Arduino IDE erfordert zusätzlich eine manuelle Synchronisation. Hierfür werden der Upload-Button in der IDE sowie der Reset-Button auf dem Arduino-Board gleichzeitig gedrückt. Sobald das „Hochladen“ in der Entwicklungsumgebung signalisiert wird, wird der Reset-Button losgelassen.

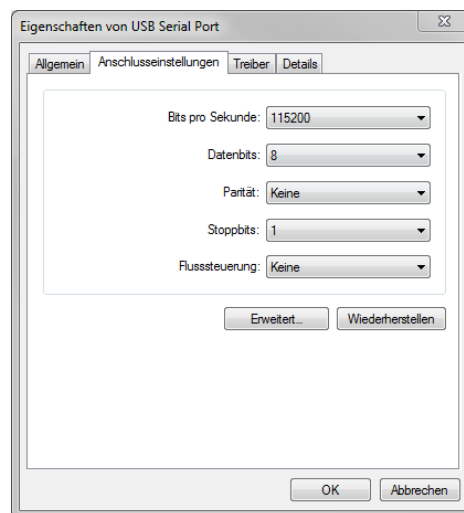


Abbildung 4.51: FDTI Treiber Konfiguration

4.3.2.2 Arduino Sketch

In folgendem Struktogramm (Abb. 4.57) ist der Programmablauf des Arduino-Sketches abgebildet. Mit der Initialisierung des Arduinos (`void setup()`) wird die Funkadresse des Moduls einmalig bestimmt. Aus der Addition der DIP-Schalter ergeben sich dabei die Funkadressen 11 bis 18 (Die ersten 10 Adressen sind für die möglichen IO-Module reserviert!). Des Weiteren wird das Funkmodul initialisiert, sowie dessen Verschlüsselung aktiviert.

Die Funktion `void loop()` beschreibt die Ablauf-Routine des Arduinos. Zur digitalen Auswertung des DHT22-Sensors wird die zur Verfügung gestellte Bibliothek `dht.h` eingebunden. Eventuelle Übertragungsfehler des Sensors werden durch wiederholte Versuche

ausgemerzt (maximal 3 Versuche). Die Messdaten werden folgend der Sende-Struktur-Variable zugewiesen. Messdaten zur Helligkeit stehen am analogen Eingang A0 bereit. Durch den [ADC⁷](#), mit einer Auflösung von $10Bit$, wird die Helligkeit mit einem Wertebereich von 0 bis 1023 beschrieben.

Zur Sendung der Daten wird das Funkmodul zunächst aufgeweckt. Nach erfolgter Datensendung wird mit der Wartefunktion (`bool waitForAck()`) maximal $200ms$ auf eine ACK-Bestätigung des Empfängers abgewartet. Das Sensor-Modul führt maximal 3 Sendeversuche für eine erfolgreiche ACK-Bestätigung durch. Anschließend wird das Funkmodul sowie der Arduino in den Schlafmodus versetzt. Die unterschiedlichen Schlafmodi, insbesondere deren Energieeffizient werden folgend explizit untersucht.

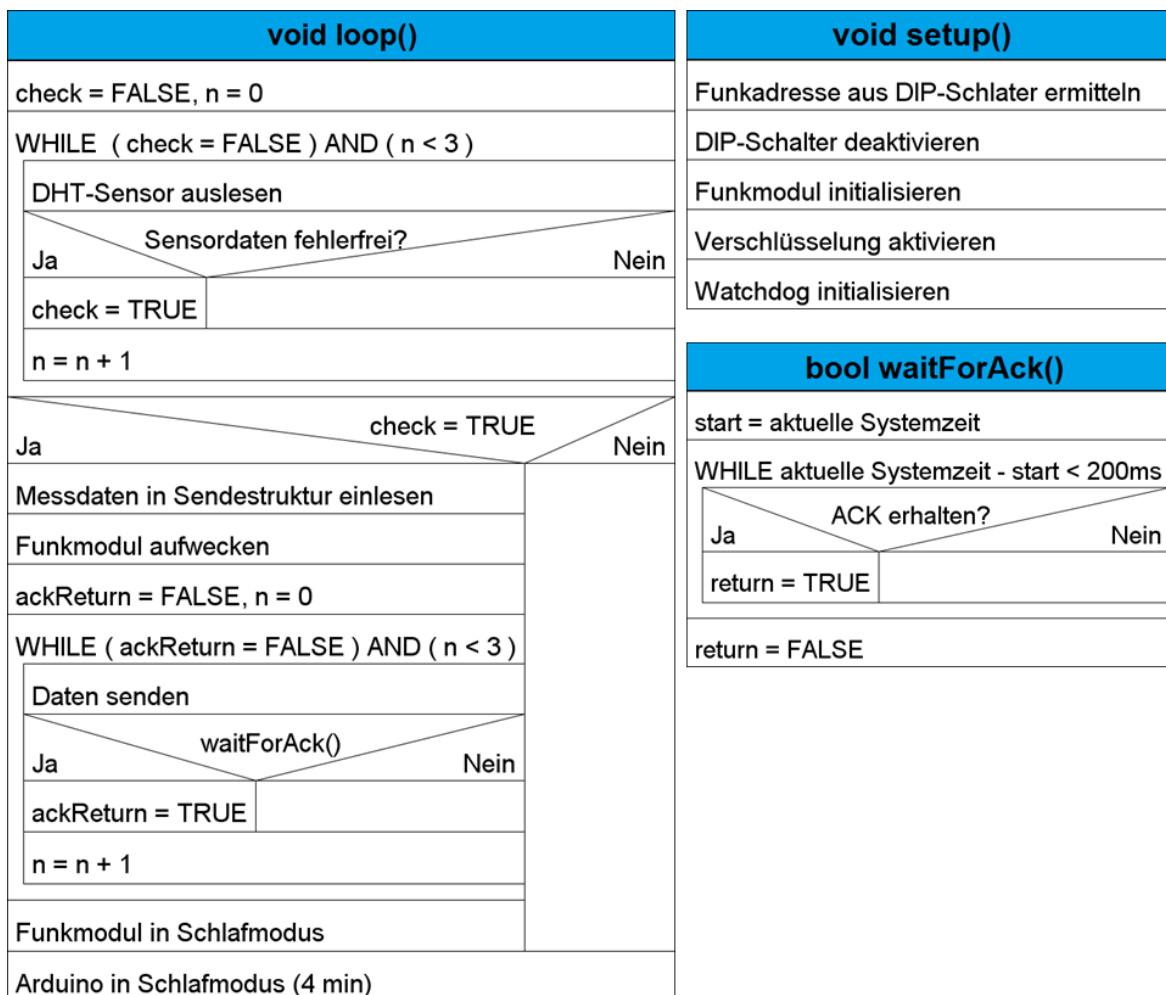


Abbildung 4.52: Struktogramm – Funk Sensor

⁷Analog Digital Converter

4.3.2.3 Schlafmodi

Um eine möglichst lange Lebensdauer zu erreichen, wird der Arduino in den Schlafmodus versetzt. Dabei werden die Standard-Bibliotheken `avr/sleep.h` und `avr/wdt.h` verwendet. Die Bibliothek `avr/sleep.h` liefert Funktionen zur Auswahl der zur Verfügung stehenden Schlafmodi. Der Mikro-Chip ATmega unterscheidet dabei die Modi Standby, Power-Down und Power-Save. Die Schlafmodi unterscheiden sich dabei in den im Schlafzustand zur Verfügung stehenden Timerfunktionen bzw. aktiven Oszillatoren des Micro-Chips. Der Schlafmodus Power-Down, von Arduino als *tiefster Schlaf* bezeichnet, verspricht durch die minimalsten Aktivitäten den energiesparendsten Betrieb. Zur Aufweckung aus dem Schlafmodus wird der Watchdog-Timer mit der Bibliothek `avr/wdt.h` eingesetzt.

In Abbildung 4.53 ist die Initialisierungs-Methode des Watchdogs dargestellt, welche einmalig bei Initialisierung des Arduinos ausgeführt wird. Über das MCU Status Register (Abb. 4.54) wird die Watchdog-Reset-Flanke mit Bit WDRF rückgesetzt. Folgend wird die Konfiguration des *Prescalers* über das WDTCSR-Register (Abb. 4.55) mit Bits WDCE und WDE aktiviert. Mit der Konfiguration des *Prescalers* durch WDP3, WDP2, WDP1 und WDP0 wird die Schlafzeit des Arduinos festgelegt. Die maximalen Zählzyklen von 1024K (1,0,0,1) ermöglichen dabei eine maximal kontinuierliche Schlafzeit von 8sek [23].

```
void watchdog() {
  MCUSR = MCUSR & B11110111; // Reset flag ausschalten, WDRF .
  WDTCSR = WDTCSR | B00011000; // Prescaler Konfiguration aktivieren
  WDTCSR = B00100001; // Prescaler auf 1024k => ergibt ca. 8 Sekunde
  WDTCSR = WDTCSR | B01000000; // Watchdog Interrupt einschalten
  MCUSR = MCUSR & B11110111;
}
```

Abbildung 4.53: Code-Ausschnitt – Watchdog Timer

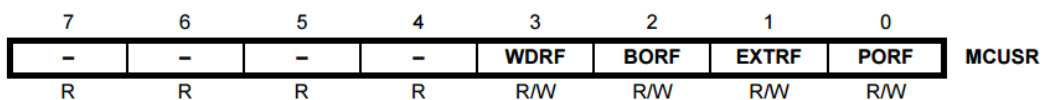


Abbildung 4.54: MCU Status Register [23]

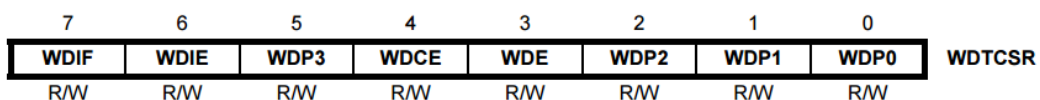


Abbildung 4.55: Watchdog Timer Control Register [23]

Mit dem Ziel einen möglichst kleinen Energieverbrauch zu erreichen, wird sowohl das Funkmodul als auch der Arduino in den Schlafmodus versetzt. Die bereits erwähnte Konfigu-

ration des *Prescalers* lässt lediglich eine Schlafzeit von *8sek* zu, wodurch die angestrebte Lebenszeit von mindestens einem Jahr nicht erreichen werden kann. In folgender Abbildung 4.56 ist die Funktion für den Schlafmodus dargestellt. Mit der Bibliothek-Funktion `set_sleep_mode()` wird der spezielle Schlafmodus initialisiert. Durch Aktivierung des Schlafmodus (`sleep_enable()`) und folgender Anwahl mit `sleep_mode()` wird der Mikrocontroller in den Schlafmodus versetzt. Mit einem Übergabeparameter von `xZeit = 30` wird der Schlafzyklus, bedingt durch die Schleife, 30 mal wiederholt, was zu einer Schlafzeit von *4min* führt.

```
void sleepModus(int xZeit) {
    // Sleep Mode setzen (SLEEP_MODE_PWR_SAVE, SLEEP_MODE_PWR_DOWN, SLEEP_MODE_STANDBY)
    set_sleep_mode(SLEEP_MODE_PWR_DOWN);

    for(int i=0; i < xZeit; i++) {
        sleep_enable();           // sleep mode einschalten
        sleep_mode();            // in den sleep mode gehen
        sleep_disable();         // sleep mode ausschalten nach dem Erwachen
    }
}
```

Abbildung 4.56: Code-Ausschnitt – Sleep-Mode

In Folgender Tabelle 4.3 sind die realen Stromverbräuche der einzelnen Schlafmodi aufgeführt. Im aktiven Betrieb wird die höchste Stromaufnahme um ca. 30.18mA gemessen. Die Unterschiede der einzelnen Schlafmodi bewegen sich dabei im Mikroampere-Bereich. Es ist deutlich zu erkennen, dass die angestrebte mittlere Stromaufnahme $< 285\mu\text{A}$ nicht zu erreichen ist. Der Mikrocontroller verspricht zwar im Schlafmodus eine Stromaufnahme von wenigen Mikroampere, das Problem liegt allerdings an der aktiven Betriebsleuchte. Die Leuchtdiode lässt sich softwaretechnisch nicht deaktivieren. Eine gewöhnliche Leuchtdiode hat einen Stromverbrauch von ca. 15mA . Ein Auslöten der Leuchtdiode würde den Stromverbrauch deutlich vermindern und das Ziel theoretisch erfüllen lassen. Eine solch drastische Manipulation des Arduinos wird an dieser Stelle nicht durchgeführt.

Tabelle 4.3: Arduino – Sleepmodi

Schlafmodus (Typ)	Stromaufnahme bei aktive Mode	Stromaufnahme bei deaktivierten Mode
Power Save Mode	13.59 mA	30.17 mA
Standby Mode	13.35 mA	30.18 mA
Power Down Mode	13.10 mA	30.17 mA

Funk-Übertragungsablauf

Zur Überprüfung der fehlerfreien Funkübertragung wird die Sende-Ausgabe des Sensors mit den Empfangs-Daten des Funk-Empfängers der Smart Home Zentrale betrachtet. Der Arduino Mini des Sensors und der Arduino Nano der Smart Home Zentrale werden dabei mit USB-Verbindung zum Entwicklungsrechner betrieben. Über den *Seriellen Monitor* der Arduino IDE lassen sich die Aktivitäten durch spezielle Print-Ausgaben der einzelnen Umgebungen beobachten. In folgender Abbildung 4.57 mit COM12-Verbindung des Sensors und COM10-Verbindung des Empfängers, sind die Ausgaben abgebildet. Bei Initialisierung der Arduinos wird einmalig der Name sowie dessen Funk-Adresse ausgegeben. Die Reserve-Variable wird zur Überprüfung der Sendungen verwendet und fortlaufend inkrementiert. Der Sensor (COM12) sendet die Messdaten an die Funkadresse 31 und wartet auf eine ACK-Bestätigung. Der Empfänger (COM10) erhält Daten von Adresse 14 und Bestätigt dessen Aufforderung mit einem ACK-Signal. Durch den sofortigen Erhalt des ACK-Signals, fährt der Sensor in den Schlafmodus.

```

COM12
-----
Funk-Sensor 868MHZ:  Adresse:
14

DHT22, OK,
OUT ---> Reserve      : 1
OUT ---> Temperatur   : 192
OUT ---> Feuchtigkeit: 23
OUT ---> Helligkeit   : 899
- waiting for ACK...ok!

DHT22, OK,
OUT ---> Reserve      : 2
OUT ---> Temperatur   : 193
OUT ---> Feuchtigkeit: 23
OUT ---> Helligkeit   : 900
- waiting for ACK...ok!

DHT22, OK,
OUT ---> Reserve      : 3
OUT ---> Temperatur   : 192
OUT ---> Feuchtigkeit: 23
OUT ---> Helligkeit   : 893
- waiting for ACK...ok!

COM10
-----
Funk-Steuermodul 868MHZ:  Adresse:
31

from Node 14
- ACK sent!
IN <--- Res : 1
IN <--- Temp : 192
IN <--- Luft  : 23
IN <--- Hell : 899

from Node 14
- ACK sent!
IN <--- Res : 2
IN <--- Temp : 193
IN <--- Luft  : 23
IN <--- Hell : 900

from Node 14
- ACK sent!
IN <--- Res : 3
IN <--- Temp : 192
IN <--- Luft  : 23
IN <--- Hell : 893

from Node 14
- ACK sent!
IN <--- Res : 4
IN <--- Temp : 192
IN <--- Luft  : 23
IN <--- Hell : 893
  
```

Abbildung 4.57: Funkübertragung – Sensor - Zentrale

Das fertige Funk Sensor-Modul ist in Anhang A.3 samt Geräteliste und Schaltplan abgebildet. Zur Erstellung des Moduls wird ein Wareneinsatz von €25,56 eingesetzt.

4.4 Z-Wave Controller

4.4.1 Hardware

Die Smart Home Umgebung wurde mit einer Z-Wave Anbindung spezifiziert. Wie bereits unter den technischen Grundlagen erläutert, wird für die Z-Wave Anbindung ein weiterer Raspberry Pi als Z-Wave-Server mit dem Funkmodul *Razberry* eingesetzt. Das Z-Wave Modul *Razberry* besitzt einen GPIO-Stecker und wird direkt auf den GPIO-Anschluss (PIN 1-10) des Raspberry aufgesetzt. Der Z-Wave-Controller bzw. dessen Z-Wave-Endgeräte werden über die Ethernet-Verbindung von der Smart Home Zentrale gesteuert.

4.4.2 Software

4.4.2.1 Z-Wave Controller einrichten

Zur Steuerung des Z-Wave-Netzwerkes wird die Controller-Software *Z-Way* von *Z-Wave.me* eingesetzt. Die *Z-Way*-Software wurde für die Linux-Distribution *Raspian* des Raspberry Pi entwickelt. Das Betriebssystem *Raspian* ist über die Homepage von Raspberry (www.raspberrypi.org) als SD-Karten-Image frei erhältlich. Der Z-Wave Controller wird nach folgenden Schritten eingerichtet. Die Software-Tools *Win32 Disk Imager* sowie *PuTTY* sind bereits aus der Installation der *CoDeSys*-Laufzeitumgebung bekannt.

1. Das *Raspian*-Image wird am Entwicklungsrechner per *Win32 Disk Imager* auf die SD-Karte des Raspberry Pi geladen.
2. Raspberry Pi mit SD-Karte und LAN-Verbindung starten bzw. mit Spannungsversorgung verbinden.
3. Die dynamische IP-Adresse des Raspberry ermitteln. (`arp/a` über Kommandozeile des Entwicklungsrechners).
4. Einwahl mit entsprechender IP-Adresse über *Telnet Client PuTTY*.
5. Benutzerdaten eingeben (`login as: pi, password: raspberry`).
6. Installation der *Z-Way* Software mit:

```
wget -q -O - razberry.z-wave.me/install | sudo bash
```
7. Raspberry mit `sudo reboot` neu laden. [20]

4.4.2.2 Z-Wave Netzwerk konfigurieren

Nach erfolgreicher Installation des Z-Wave Controllers erfolgt die Konfiguration der Z-Way-Serversoftware. Die Konfiguration erfolgt dabei über die grafische Oberfläche mit Zugriff aus dem Intranet (Abb.4.58). Das Z-Wave Netzwerk wird in folgenden Schritten konfiguriert.

1. Die Einwahl auf dem Z-Wave-Controller erfolgt mit IP-Adresse:8083 (Abb.4.58)
2. Über den Reiter *Expert UI* erreicht man die Konfigurations-Übersicht des Z-Wave Netzwerkes.
3. Mit *Network* → *Control* → *Start Inclusion* werden erkannte Z-Wave-Geräte dem Netzwerk hinzugefügt. Zur Erkennung der Teilnehmer muss für sämtliche Z-Wave-Endgeräte der Inklusions-Modus aktiviert werden.
4. Nach erfolgter Inklusion sind sämtliche Geräteeigenschaften der Netzwerk-Umgebung über den Reiter *Configuration* einsehbar und gegebenenfalls konfigurierbar. Über die Konfiguration-Ansichts des Z-Wave-Gerätes lassen sich die Geräte-Id sowie die zur Verfügung stehenden Kommando-Klassen zur externen Ansteuerung ermitteln. Auch Hinweise zur Aktivierung des Inklusions-Mode des jeweiligen Gerätes sind daraus ersichtlich.
5. Über den Reiter *Control* lassen sich sämtliche Geräte testweise steuern. In Abbildung 4.59 werden die Sensoreigenschaften des Multisensors beispielsweise ausgelesen.

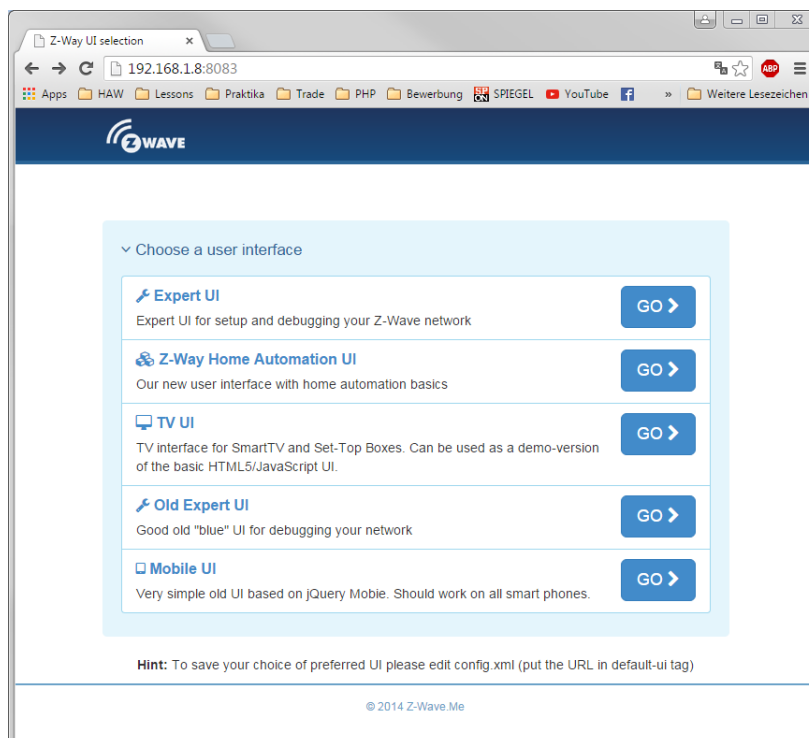


Abbildung 4.58: GUI – Z-Way-Software

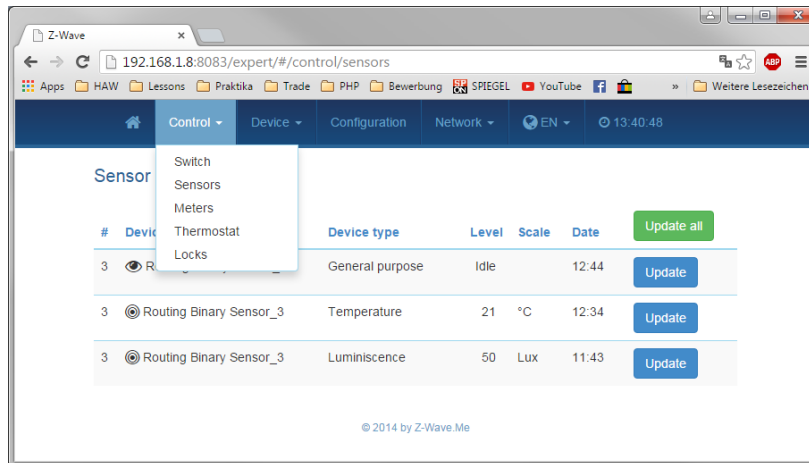


Abbildung 4.59: GUI – Z-Way Control

4.4.2.3 JSON - CommandClass

Nach erfolgreicher Inklusion der Z-Wave-Geräte wird die externe Ansteuerung der JSON-API des Z-Wave-Servers untersucht. Die Geräte-IDs, sowie die zur Verfügung stehenden Kommando-Klassen der verwendeten Geräte werden über deren *Configuration*-Seite des Servers ausgemacht. Abbildung 4.60 zeigt die Konfigurations-Eigenschaften der Z-Wave-Steckdose. Die indirekte Ansteuerung der Geräte über die JSON API wird über den Browser des Entwicklungsrechners getestet.

Der grundsätzliche Zugriff der API erfolgt über die Adresse

`http://IP-Z-Way:8083/ZWaveAPI/Run/devices[x].instances[y]...`, gefolgt von der Kommandoklasse (x =Geräte-ID, y =Instanz-ID). Die *Instanz-ID* inkrementiert, je nach Anzahl der inkludierten Geräte vom selben Typ. [33]

Schaltbare Steckdose

Als Akteur-Beispiel wird die schaltbare Steckdose von MÜWI mit Geräte-ID-Nr. 4 eingesetzt. Zur Ansteuerung wird die Kommandoklasse *Basic* angewandt. Mit der Kommando-Methode `Basic.Set()` und Übergabeparameter 0 bzw. 255 kann die Steckdose über den Browser gesteuert werden. Der Zustand der Steckdose wird auf dem Z-Way-Server gespeichert und kann mit dem Aufruf `Basic.data.level.value` erfragt werden. Der Rückgabewert von 0 bzw. 255 gibt folgend Auskunft über den aktuellen Zustand. Die Steckdose von MÜWI verfügt zudem über einen lokalen Taster am Gerät, der lokales Schalten ermöglicht. Dessen Funktionalität kann über die Geräte-Eigenschaft des Z-Way-Servers konfiguriert werden. Ein lokal gesteuerter Zustandswechsel muss mit der Methode `Basic.Get()` auf dem Server

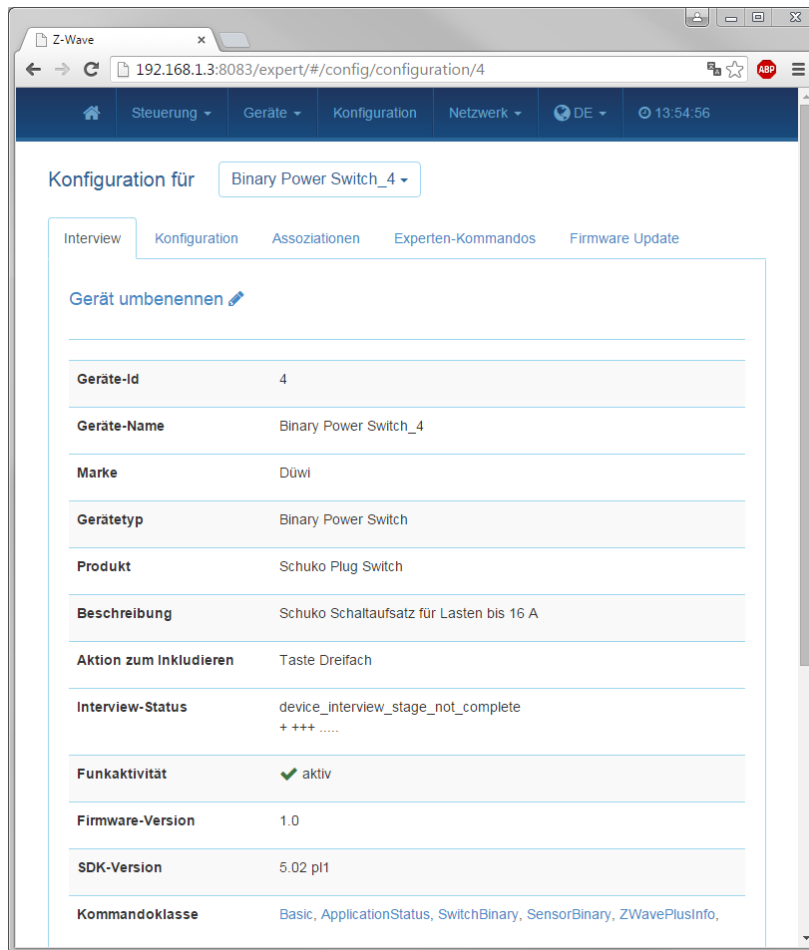


Abbildung 4.60: Z-Way – Configuration

aktualisiert werden. Die Verwendung des Tasters wird in der Umsetzung deaktiviert und erspart das kontinuierliche Aktualisieren des Zustandes.

Multisensor

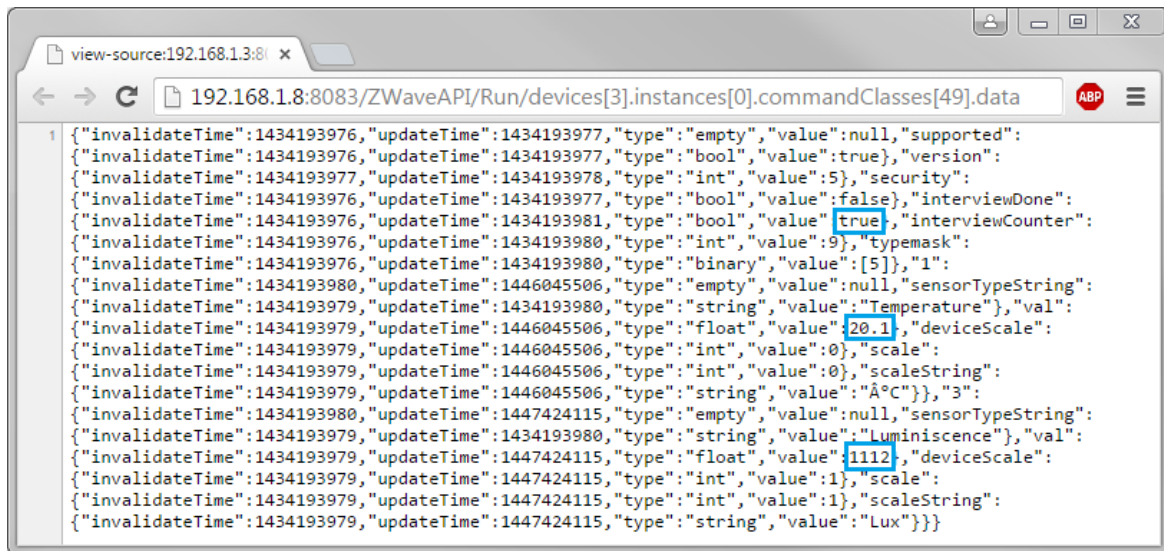
Der Multisensor liefert Messungen von Temperatur und Helligkeit und erkennt Bewegungen durch den optischen Sensor. Über die Geräte-Konfiguration des Sensors lassen sich die Messintervalle festlegen. Zwischen den Intervallen befindet sich der Sensor im Schlafmodus. Um eine wartungsfreie Verwendung zu garantieren, sollten die Messintervalle größer 5min gewählt werden, da kurze Messintervalle die Batterie-Lebensdauer wesentlich verkürzen. Die Nutzung der Bewegungserkennung ist dadurch für aktive Lichtansteuerungen nicht zu gebrauchen und könnte optional lediglich zur Alarmüberwachung genutzt werden.

Zur Auslesung der Messwerte auf dem Z-Way-Server steht eine Vielzahl von Kommando-

Klassen zur Verfügung, die einzelnes Auslesen der Messwerte ermöglichen. Da ein mehrfacher Aufruf nicht sinnvoll erscheint, wird eine einmalige Ausgabe sämtlicher Eigenschaften der betreffenden Kommando-Klasse bevorzugt. In Abbildung 4.61 sind die Rückgabewerte folgender Kommando-Klasse-Ergänzung abgebildet:

```
.../devices[3].instances[0].commandClasses[49].data
```

Die gekennzeichneten Werte geben die Messwerte wieder und sollen für deren Verwendung in CoDeSys gefiltert werden (Bewegungssensor = TRUE, Temperatur = 20.1, Helligkeit = 1112).



```
1 [{"invalidateTime":1434193976,"updateTime":1434193977,"type":"empty","value":null,"supported":
{"invalidateTime":1434193976,"updateTime":1434193977,"type":"bool","value":true},"version":
{"invalidateTime":1434193977,"updateTime":1434193978,"type":"int","value":5},"security":
{"invalidateTime":1434193976,"updateTime":1434193977,"type":"bool","value":false},"interviewDone":
{"invalidateTime":1434193976,"updateTime":1434193981,"type":"bool","value":true},"interviewCounter":
{"invalidateTime":1434193976,"updateTime":1434193980,"type":"int","value":9},"typemask":
{"invalidateTime":1434193976,"updateTime":1434193980,"type":"binary","value":[5]},"1":
{"invalidateTime":1434193980,"updateTime":1446045506,"type":"empty","value":null,"sensorTypeString":
{"invalidateTime":1434193979,"updateTime":1434193980,"type":"string","value":"Temperature"},"val":
{"invalidateTime":1434193979,"updateTime":1446045506,"type":"float","value":20.1},"deviceScale":
{"invalidateTime":1434193979,"updateTime":1446045506,"type":"int","value":0},"scale":
{"invalidateTime":1434193979,"updateTime":1446045506,"type":"int","value":0},"scaleString":
{"invalidateTime":1434193979,"updateTime":1446045506,"type":"string","value":"Å°C"},"3":
{"invalidateTime":1434193980,"updateTime":1447424115,"type":"empty","value":null,"sensorTypeString":
{"invalidateTime":1434193979,"updateTime":1434193980,"type":"string","value":"Luminiscence"},"val":
{"invalidateTime":1434193979,"updateTime":1447424115,"type":"float","value":1112},"deviceScale":
{"invalidateTime":1434193979,"updateTime":1447424115,"type":"int","value":1},"scale":
{"invalidateTime":1434193979,"updateTime":1447424115,"type":"int","value":1},"scaleString":
{"invalidateTime":1434193979,"updateTime":1447424115,"type":"string","value":"Lux"}}}]
```

Abbildung 4.61: Z-Way – Multisensor-Return

Doppel-Taster

Es wird der Doppel-Taster von zwave.me eingesetzt. Hinter den Doppel-Wippen des Tasters verbergen sich 4 einzelne Taster. Der Inclusions-Mode wird durch gleichzeitige Betätigung aller Taster und darauffolgender Betätigung des rechten, oberen Tasters bei Led-Signal aktiviert. Über die Konfigurations-Beschreibung des Z-Way-Servers wird die Geräte-ID-Nr. 2 festgestellt. Es stehen unterschiedliche Konfigurations-Möglichkeiten der Taster zur Auswahl. Die Taster können bei Bedarf nur zum Ausschalten bzw. Einschalten konfiguriert werden, oder auch als abhängige Gruppenschaltung fungieren. Alle 4 Taster werden dabei als unabhängige Taster konfiguriert und bieten somit den größtmöglichen Verwendungsfreiraum.

Für die externe Auswertung der Tastersignale wurde folgende Zugriffs-Methode analysiert:

```
.../devices[2].instances[0].commandClasses[91].data.currentScene
```

In folgender Abbildung 4.62 ist der Rückgabe-String bei zuletzt betätigtem Taster 4 abgebildet. Über die `value`-Angabe kann die letzte Taster-Betätigung zugeordnet werden (Taster1:1, Taster2:2, Taster3:5, Taster4:6). Aus der Angabe der `updateTime` lassen sich wiederholte Betätigungen desselben Tasters ausmachen.

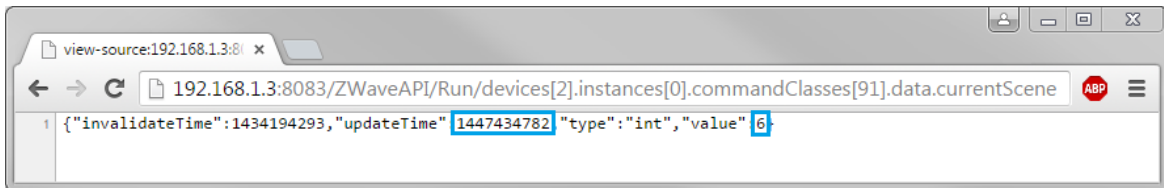


Abbildung 4.62: Z-Way – Taster3-Return

4.4.2.4 CoDeSys - Ansteuerung

Die bereits analysierte, externe Ansteuerung der Z-Wave-Geräte über die JSON-Schnittstelle des Z-Wave-Servers soll folgend über die CoDeSys-Umgebung realisiert werden. Die Adressen-Eingabe bzw. deren Ausführung über einen Browser ist ein sogenannter [HTTP⁸](#)-Request; eine GET-Methode die eine bestimmte Ressource mittels [TCP⁹/IP¹⁰](#)-Protokoll am Web-Server anfordert.

Für den HTTP-Request wird die lizenzfreie Network-Bibliothek von Oscat (www.oscat.de) bedient, welche auch über den CoDeSys-Store zur Verfügung gestellt wird. Die Bibliothek verfügt über unterschiedlichste Übertragungs-Bausteine, sowie über den speziell zur Ausführung eines HTTP-Request nötigen Funktionsbaustein `HTTP_GET`. Die Anwendung bzw. Parameterbeschreibung der folgend genannten Funktionsbausteine wird im Anhang [A.4.2](#) erläutert. Die Ausführung beschränkt sich dabei lediglich auf deren Anwendung. Spezifische Informationen sind der Bibliotheks-Dokumentation zu entnehmen (im Bibliotheks-Paket enthalten).

Funktionsbaustein `IP_CONTROL`

Zur zentralen Koordination des Socket-Zugriffes zur Ethernet-Kommunikation ist der Funktionsbaustein `IP_CONTROL` der Oscat-Bibliothek zuständig. `IP_CONTROL` ist eine zentrale

⁸Hypertext-Übertragungsprotokoll (Hypertext Transfer Protocol)

⁹Übertragungssteuerungsprotokoll (Transmission Control Protocol)

¹⁰Internet Protokoll (Internet Protocol)

Organisationseinheit, die als sogenannter Wrapper fungiert. Dieser ermöglicht eine neutrale Nutzung der Ethernet-Kommunikation und wird, je nach Anwendung der Ethernet-Verbindung, um anwendungsspezifische Funktionsbausteine ergänzt. Der HTTP-Request wird, wie bereits erwähnt, mit der Baustein-Ergänzung *HTTP_GET* umgesetzt.

Funktionsbaustein *HTTP_GET*

Der Funktionsbaustein *HTTP_GET* wird für die Ansteuerung der Z-Wave-Geräte modifiziert. Der Standard-Baustein führt mittels des TCP/IP-Protokolls ein HTTP-Request aus. Bei erfolgreicher Antwort des Webservers wird die Antwort mit HTML -Header und -Body-Information mit separaten Informationen des Body-Beginns sowie Body-Endes mittels Zeichen-Positionsangabe als String ausgegeben.

Es wird die Eingangs-Variable *TYP* (INT) hinzugefügt. Je nach Eingangskonfiguration wird die Request-Antwort nach den relevanten Informationen gefiltert (1:Steckdose, 2:Taster, 3:Multisensor). Die zuvor analysierten *URL*¹¹-Adressen (Punkt 4.4.2.3) zur Ansteuerung der Z-Wave-Geräte werden über die Variable *URL_DATA* des Funktionsbausteins *HTTP_GET* eingelesen. Je nach *TYP*-Konfiguration erfolgt die Ausgabe über die String-Variable *STATUS*.

Folgender Code-Ausschnitt (4.63) soll die Filterung am Beispiel der Taster-Abfrage (*TYP*=2) verdeutlichen. Der Wert als Information des zuletzt betätigten Tasters wird an Stelle 273 analysiert. Die besagte Stelle, bestehend aus einem Zahl-Zeichen, wird als String in die Variable *output* konvertiert. Um mehrmalige Betätigungen zu erkennen, wird die Aktualisierungszeit mit *uptime* an Stelle 241 bis 250 gefiltert und mit der letzten Filterung verglichen. Bei Änderung der Zeit erfolgt die *STATUS*-Ausgabe mit der Zeichenausgabe „6“ (aus Abb. 4.62 „6“ = Taster 3).

```

IF HTTP_STATUS = '200 OK' THEN // Anfrage erfolgreich
  CASE TYP OF
    ...
    2: // Body-Auswertung fuer Tastererkennung
      output := OSCAT_BASIC.BUFFER_TO_STRING(ADR(R_BUF.BUFFER),R_BUF.SIZE, 273, 273);
      uptime := OSCAT_BASIC.BUFFER_TO_STRING(ADR(R_BUF.BUFFER),R_BUF.SIZE, 241, 250);

      // Falls 6 => Taster 3
      IF output = '6' THEN
        IF uptime = uptimeLast[0] THEN STATUS:= '';
        ELSE STATUS:= '6'; uptimeLast[0]:= uptime; END_IF
        state:= 50;
        DONE:= TRUE;

      // Falls 2 => Taster 2
      ELSIF output = '2' THEN
        ...
      END_CASE
    END_IF
  
```

Abbildung 4.63: Code-Ausschnitt – Requestfilter

¹¹einheitlicher Ressourcenanzeiger (Uniform Resource Locator)

Funktionsbaustein ZW_Controller

Der Funktionsbaustein *ZW_Controller* ist der Haupt-Baustein zur Integrierung der Z-Wave-Technologie in das Smart Home System und wird im MAIN-Baustein des Smart Home-Projektes instanziiert. Durch Anwendung der vorgestellten OScat-Bausteine wird ein Datenaustausch mit dem Z-Wave-Server realisiert. Zur Ansteuerung der Z-Wave-Geräte wird die Strukturvariable *ZWAVE* in der globalen Variablenliste *GVL_Main* angelegt. Die mögliche Anzahl bzw. Instanzen der einzelnen Geräte werden konform zu den anderen Modul-Entwicklungen mit maximal 8 festgelegt. Messwerte des Multisensors werden zyklisch im 5min-Intervall in der *ZWAVE*-Variable aktualisiert. Taster-Aktivitäten werden alle 400ms überprüft und stehen für die Weiterverarbeitung bereit.

Folgende Daten-Zugriffe sollen den Struktur-Zugriff der Variable *ZWAVE* verdeutlichen:

- Z-Wave Steckdose 3 umschalten
`ZWAVE.Steckdose[2] := TRUE`
- Z-Wave Steckdosenzustand 3 auslesen
`BOOL ZWAVE.SteckdoseStatus[2]`
- Z-Wave Doppeltaster 8: Taster3
`BOOL ZWAVE.Taster[7][2]`
- Z-Wave Multisensor 4: Temperatur
`REAL ZWAVE.Multisensor[3].Temperatur`
- Z-Wave Multisensor 4: Helligkeit
`UINT ZWAVE.Multisensor[3].Helligkeit`

5 Résumé

5.1 Bewertung

Im Rahmen dieser Arbeit ist mit kostengünstigen Mitteln eine solide Hard- und Softwarelösung eines Smart Home Systems entstanden. Die Grenzen des sehr breit gefächerten Themenbereiches „Smart Home“ wurden dabei bei Weitem nicht erreicht. Die Art und Weise der Umsetzung bietet allerdings schier grenzenlose Erweiterungsmöglichkeiten.

Mit der Web-Visualisierung von CoDeSys wurde eine zentrale Steuerung einer Beispiel-Wohnung geschaffen, welche besonders durch ihre Modi-Ansteuerung der Vision des „Internet of Things“ einen Schritt näher kommt. Durch die Unterstützung der gängigen Browser, sowie insbesondere der Systeme Android und iOS, kann der Zugriff auf die Visualisierung von fast allen Tablet-PCs erfolgen. In dieser Arbeit wurde der Zugriff auf das Intranet beschränkt. Eine entsprechende Konfiguration des Routers würde den Zugriff aus dem Internet ermöglichen. Aufgrund der komplexen Netzwerk-Topologie der Hochschule, insbesondere dessen Sicherheitsstandard, lies sich dies allerdings nicht realisieren.

Durch die objektorientierte Softwarestrukturierung wurden die Vorteile der Programmieroberfläche CoDeSys voll ausgekostet. Sie bietet eine einfache Anpassungen an gegebene Wohnungssituationen, sowie einen modularen Einsatz für weitere eventuelle Entwicklungen.

Sämtliche externe Anschlüsse der entworfenen Module können vom Anwender individuell über die Visualisierung konfiguriert werden. Der Einsatz der Module kann somit bestmöglich nach der gegebenen Elektro-Installation erfolgen und kommt besonders der Integration in bestehenden Installationen zugute.

Die Einbindung der Z-Wave Technologie schuf die Nutzung von klassischen Z-Wave-Geräten der Smart Home Industrie. Zur Funkübertragung zwischen eigenen Entwicklungen konnte die Technik, aufgrund des proprietären Funkprotokolls zwar nicht eingesetzt werden, trotzdem hat ihre Einbindung als zentrale Steuerung eine große Erweiterungsmöglichkeit des Systems geschaffen. Die Standardisierung des Protokolls und die gegebene Kompatibilität sämtlicher Z-Wave Produkte erwies sich im Vorfeld als starker Mehrwert der Technologie. Die Umsetzung der Ansteuerung der unterschiedlichen Z-Wave-Geräte zeigte allerdings

den Umfang der Standardisierung. Er bietet eine Vielzahl von möglichen Geräteklassen mit unterschiedlichsten Kommandoklassen, die je nach Hersteller individuell ihre Funktion erhalten. Im Rahmen der Arbeit wurde die Ansteuerung von lediglich drei Geräten umgesetzt. Eine Umsetzung sämtlicher Geräteklassen ohne betreffende Hardware ist schwer realisierbar.

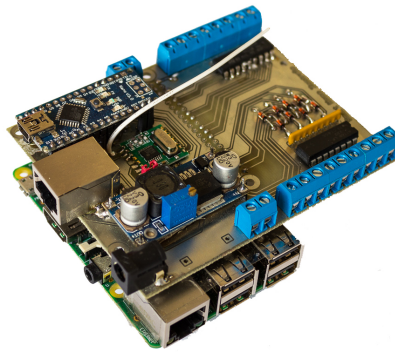
Bei der Entwicklung des Funk-Sensors wurde das Ziel, eine Lebensdauer von einem Jahr zu erreichen, nicht erfüllt. Bei rund 4-fachem Stromverbrauch nach aktueller Entwicklung muss ein Batteriewechsel alle 3 Monate erfolgen. Für den Einsatz in einer Smart Home Umgebung ist dieser hohe Wartungsaufwand nicht hinnehmbar. Bessere Ergebnisse würde die Umsetzung mit reinem ATmel-Chip anstatt des Arduinocontrollers liefern.

5.2 Aussichten

Das entwickelte System bietet eine solide Grundlage für eine smarte Haussteuerung. Das Augenmerk lag dabei vorwiegend auf der instinktiven Bedienung der Visualisierung, der Schaltbarkeit von bestehenden Schaltungskreisen, sowie eine modulare Erweiterbarkeit. Die individuellen Erweiterungsinteressen sind dabei fast grenzenlos. Es gibt zahlreiche Z-Wave-Geräte die derzeit nicht unterstützt werden wie bspw. Rolllansteuerungen, Jalousien, Fenster- und Türkontakte oder Heizungsventile, um nur einige zu nennen. Heizungsventile könnten dabei mit der Solltemperatur der Modi angesteuert werden. Weitere, wichtiger Mehrwert eines Smart Home Systems wär die Visualisierung und Archivierung von Energiedaten durch geeignete Messsensoren für den Leistungs- und Wasserverbrauch. Auch Meldungen eines vollen Postfachs oder Füllstände von Lebensmitteln könnten in Zukunft per Smartphone erfolgen. Das sollen hierbei lediglich Beispiele für die Vernetzung von möglichen Gegenständen der Wohnungsumgebung sein, die den Gedanken des „Internet of Things“ in Zukunft immer mehr erfüllen.

A Anhang

A.1 Smart Home Zentrale

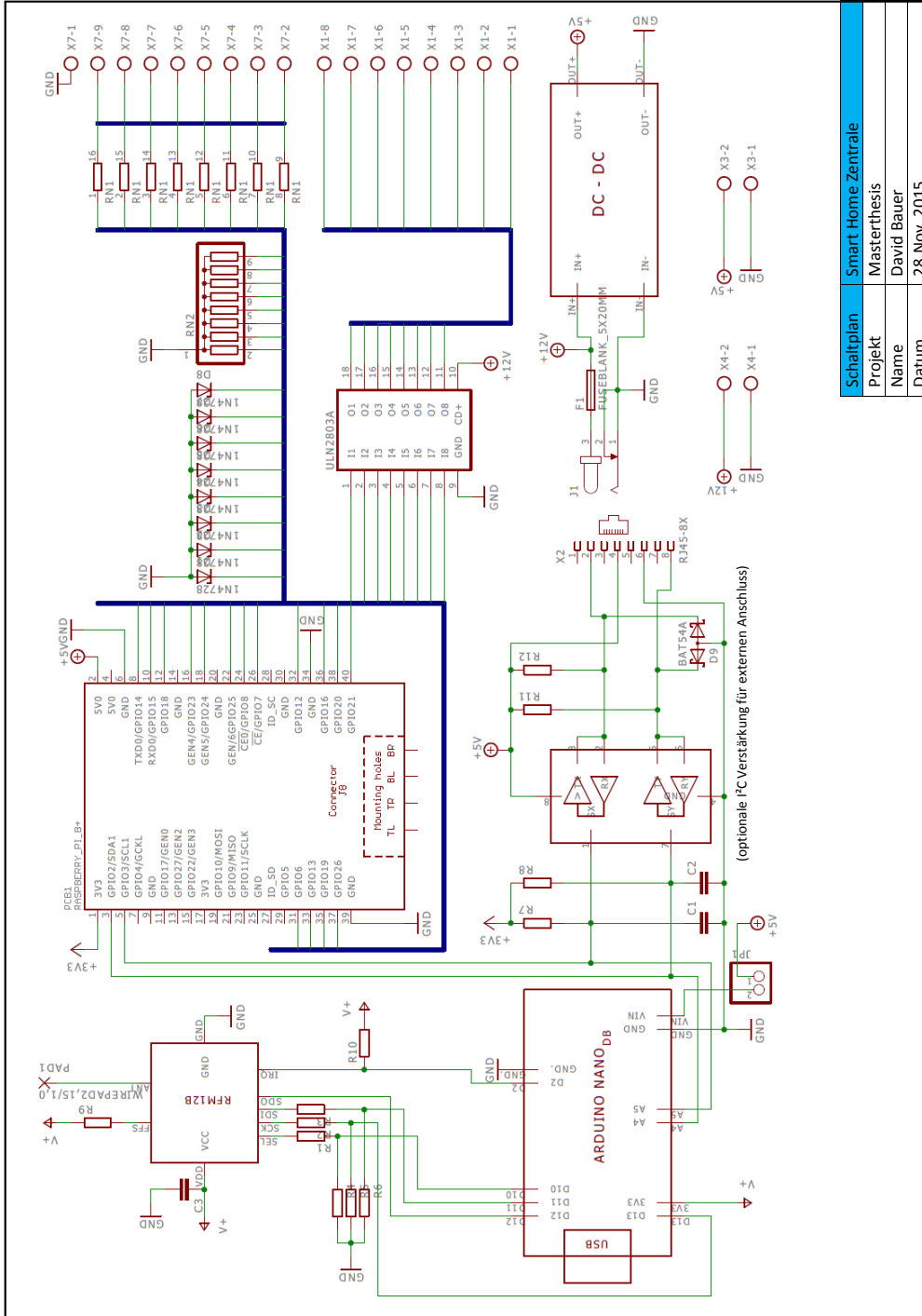


A.1.1 Geräteliste

Tabelle A.1: Geräteliste

Bauteil	Beschreibung	St.	€/St.	Preis
Raspberry Pi	Modell B+, 700MHz, 500MB RAM	1	29,50	29,50
Arduino Nano	ATmega 328P	1	16,90	16,90
RFM12B	Funkmodul RFM12B-868-S Sende-/Empfangsmodul	1	4,35	4,35
Z-Diode ZPD3.3V	Zener-Diode 500mW 3.3V	8	0,20	1,60
Bourns R-Netzwerk 8x2.2kΩ	Widerstand-Netzwerk radial bedrahtet DIP-16, 8W	1	1,10	1,10
Bourns R-Netzwerk 8x47kΩ	Widerstand-Netzwerk radial bedrahtet SIP-9, 8W	1	1,20	1,20
ULN2803A	Darlington Transistor Netzwerk, DIL-18	1	0,38	0,38
Kondensator 100nF	Keramik, 16V, 10% Toleranz	1	0,21	0,21
Widerstand 10 kΩ	Metall, 0.6W, 1% Toleranz	9	0,13	1,17
Widerstand 4.7 kΩ	Metall, 0.6W, 1% Toleranz	3	0,13	0,39
LM2596 DC-DC-Wandler	Spannungswandler 1.23-30V, 3A, einstellbar	1	5,03	5,03
RJ45 Buchse	STP, Einbau horizontal	1	2,09	2,09
Schraubkontakt 1.5mm ²	5mm Raster, 2 Anschlüsse	3	0,31	0,93
Schraubkontakt 1.5mm ²	5mm Raster, 3 Anschlüsse	4	0,36	1,44
Netzteil	12V/DC, 2.5A	1	14,90	14,90
Photoplatine (beidseitig)	80x100mm, 1,5mm, 35μ	1	2,20	2,20
Enwicklungsmaterial	Natriumhydroxid, Entwickler, Zinn	1	3,00	3,00
Buchsenleiste RM 2.54	Präzision, gerade, RM 2.54mm, 2x20	1	1,60	1,60
Buchsenleiste RM 2.54	Präzision, Gerade, RM 2.54mm, 1x20	1	0,86	0,86
Stiftleiste RM 2.54	Gerade, RM 2.54mm, 1x10	1	0,22	0,22
Jumper RM 2.54	Kurzschlussbrücke, RM 2.54mm	1	0,15	0,15
Niedervolt Buchse	Einbau horizontal, 6mm	1	0,66	0,66
CoDeSys Lizenz	CoDeSys Control for Raspberry Pi SL	1	37,50	37,50
Micro-SD-Karte	Speicherkapazität 8GB	1	9,90	9,90
GESAMT				137,28€

A.1.2 Schaltplan



Schaltplan	Smart Home Zentrale
Projekt	Masterthesis
Name	David Bauer
Datum	28.Nov. 2015

Abbildung A.1: Schaltplan

A.1.3 Platinenlayout

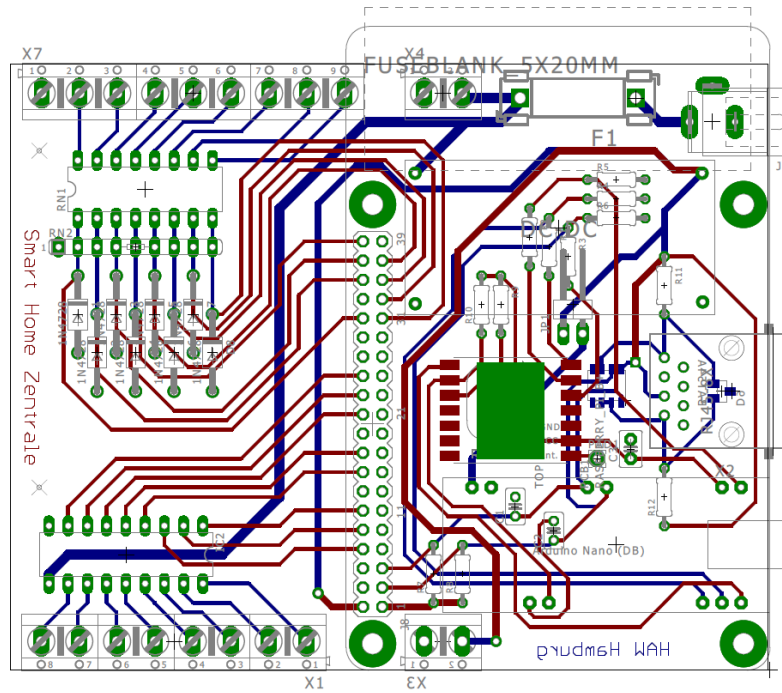
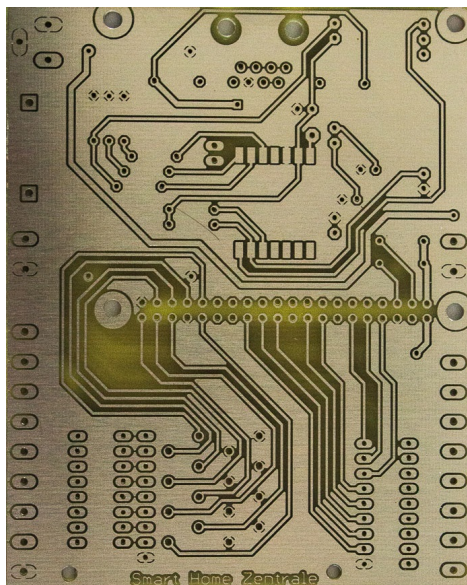
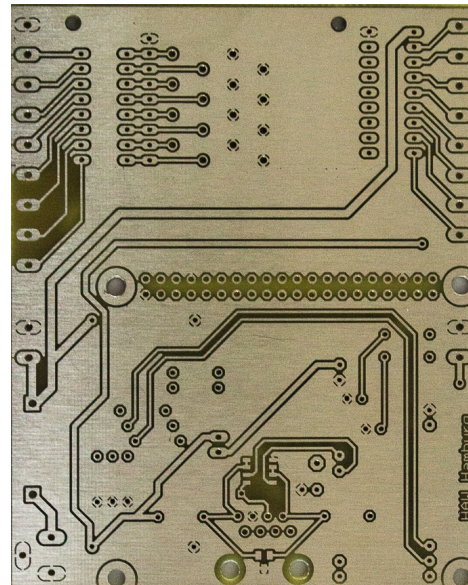


Abbildung A.2: Platinenlayout



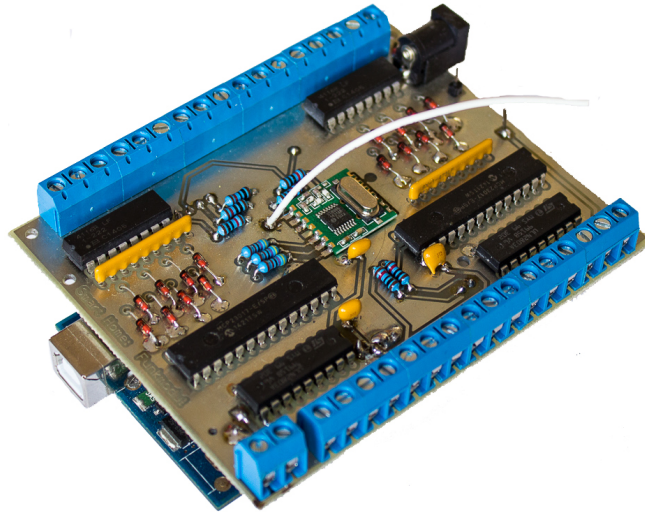
(a) vorne



(b) hinten

Abbildung A.3: Platine

A.2 Funk IO-Modul

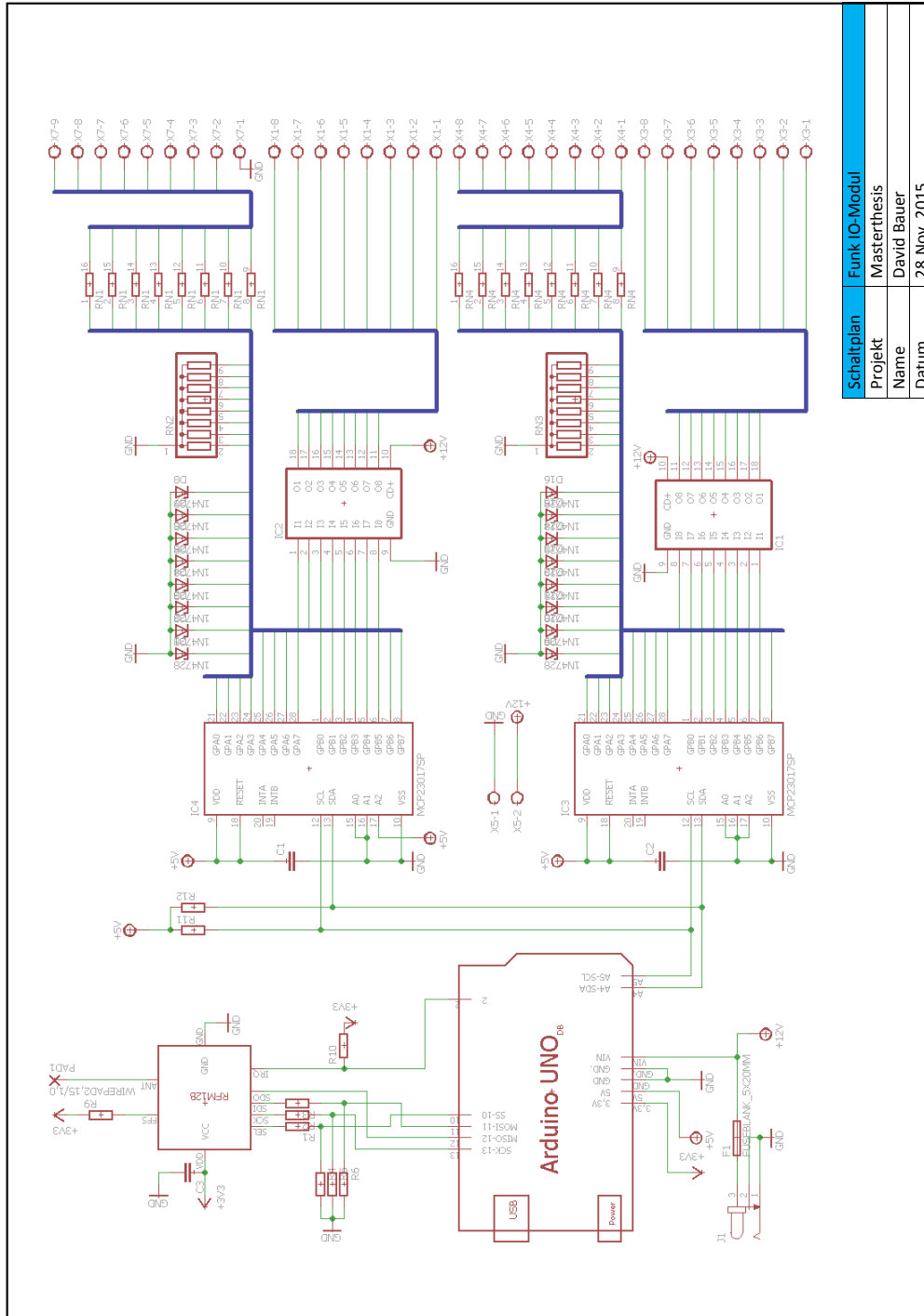


A.2.1 Geräteliste

Tabelle A.2: Geräteliste

Bauteil	Beschreibung	St.	€/St.	Preis
Arduino UNO	ATmega 328P	1	13,90	13,90
RFM12B	Funkmodul RFM12B-868-S Sende-/Empfangsmodul	1	4,35	4,35
Z-Diode ZPD4.7V	Zener-Diode 500mW 4.7V	16	0,22	3,52
Bourns R-Netzwerk 8x2.2kΩ	Widerstand-Netzwerk radial bedrahtet DIP-16, 8W	2	1,10	2,20
Bourns R-Netzwerk 8x47kΩ	Widerstand-Netzwerk radial bedrahtet SIP-9, 8W	2	1,20	2,40
ULN2803A	Darlington Transistor Netzwerk, DIL-18	2	0,38	0,76
MCP23017-E/SP	I/O-Expander, Linear-IC, SPDIP-28, I2C	2	1,93	3,86
Kondensator 100nF	Keramik, 16V, 10% Toleranz	3	0,21	0,63
Widerstand 10 kΩ	Metall, 0.6W, 1% Toleranz	7	0,13	0,91
Widerstand 4.7 kΩ	Metall, 0.6W, 1% Toleranz	3	0,13	0,39
Schraubkontakt 1.5mm ²	5mm Raster, 2 Anschlüsse	4	0,31	1,24
Schraubkontakt 1.5mm ²	5mm Raster, 3 Anschlüsse	9	0,36	3,24
Netzteil	12V/DC, 2.5A	1	14,90	14,90
Photoplatine (beidseitig)	80x100mm, 1,5mm, 35μ	1	2,20	2,20
Enwicklungsmaterial	Natriumhydroxid, Entwickler, Zinn	1	3,00	3,00
Stiftleiste RM 2.54	Gerade, RM 2.54mm, 1x20	2	0,34	0,68
Niedervolt Buchse	Einbau horizontal, 6mm	1	0,66	0,66
GESAMT				58,84 €

A.2.2 Schaltplan



Schaltplan	Funk IO-Modul
Projekt	Masterthesis
Name	David Bauer
Datum	28.Nov. 2015

Abbildung A.4: Schaltplan

A.2.3 Platinenlayout

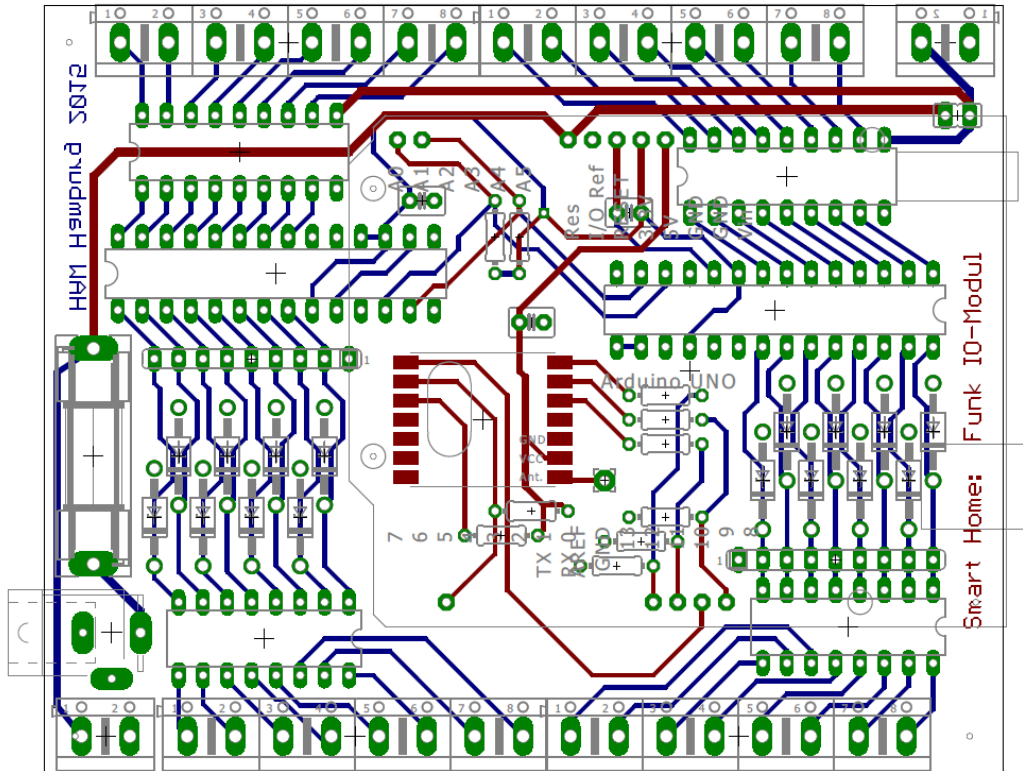
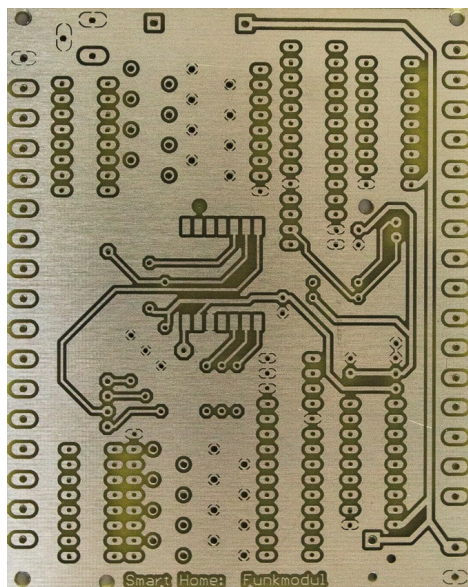
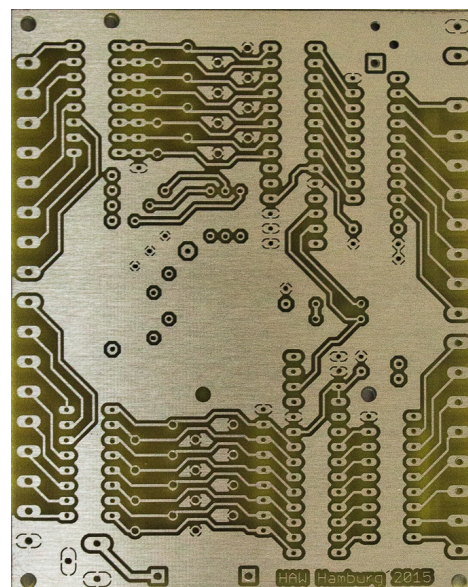


Abbildung A.5: Platinenlayout



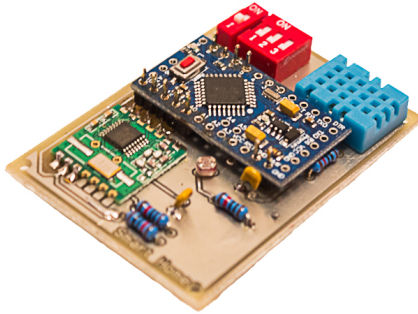
(a) vorne



(b) hinten

Abbildung A.6: Platine

A.3 Funk Sensor-Modul

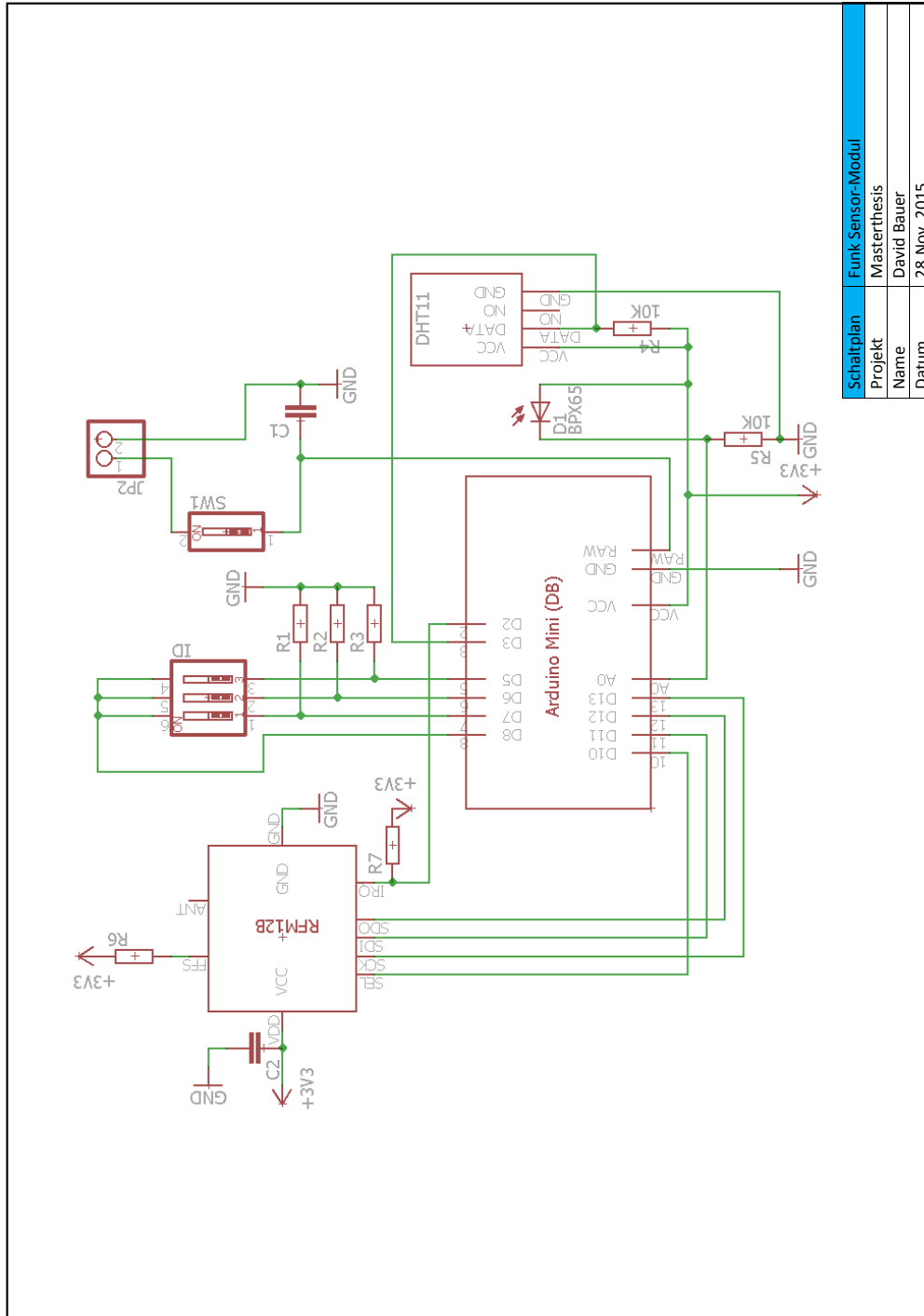


A.3.1 Geräteliste

Tabelle A.3: Geräteliste

Bauteil	Beschreibung	St.	€/St.	Preis
Arduino Mini Pro	ATmega 328P	1	6,39	6,39
RFM12B	Funkmodul RFM12B-868-S Sende-/Empfangsmodul	1	4,35	4,35
DHT22 Sensor	Digital Aasong, Temp. und Feuchtigkeit	1	4,20	4,20
Fotodiode PFW351	3mm, max100V, max50mW, 0,6M Ω	1	0,95	0,95
DIP-Schalter DS-01	DIP-Schalter, Polzahl 1	1	0,78	0,78
DIP-Schalter DS-03	DIP-Schalter, Polzahl 3	1	0,95	0,95
Kondensator 100nF	Keramik, 16V, 10% Toleranz	2	0,42	0,84
Widerstand 10 k Ω	Metall, 0.6W, 1% Toleranz	7	0,13	0,91
Photoplatine (beidseitig)	80x100mm, 1,5mm, 35 μ	1	2,20	2,20
Entwicklungsmaterial	Natriumhydroxid, Entwickler, Zinn	1	3,00	3,00
Stiftleiste RM 2.54	Gerade, RM 2.54mm, 1x20	1	0,34	0,34
Batteriehalter	4-fach, 4xMignon-Batterien	1	0,40	0,40
Druckknopfanschluss	Passend für 9V-Batterie	1	0,25	0,25
GESAMT				25,56 €

A.3.2 Schaltplan



Schaltplan	Funk Sensor-Modul
Projekt	Masterthesis
Name	David Bauer
Datum	28.Nov. 2015

Abbildung A.7: Schaltplan

A.3.3 Platinenlayout

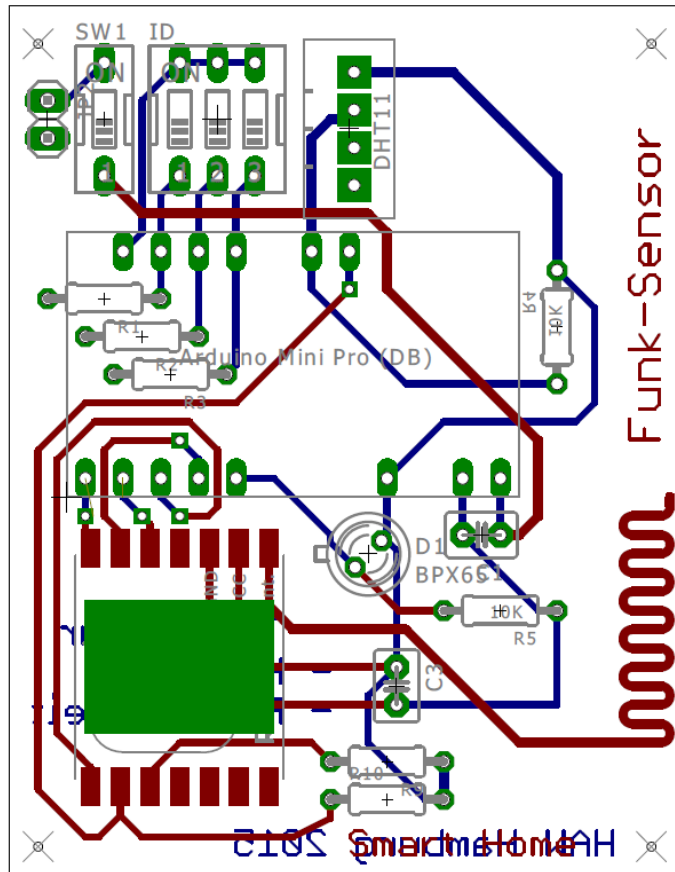


Abbildung A.8: Platinenlayout

A.4 Bibliotheken

A.4.1 Arduino Bibliotheken

Wire.h

Bibliothek: Wire.h	
Arduino Standard Bibliothek für I ² C-Bus	
Bibliothek - Funktionen	Beschreibung
<code>Wire.begin(uint8_t id)</code>	I ² C-Slave initialisieren. Übergabeparameter: id = I ² C-Slave-Adresse
<code>Wire.onReceive(handler)</code>	Eingangs-Handler des I ² C Slave wird festgelegt. handler: Funktion die bei Dateneingang aufgerufen wird. Datenerhalt wird extern durch I ² C-Master bestimmt!
<code>Wire.onRequest(handler)</code>	Ausgangs-Handler des I ² C Slave wird festgelegt. handler: Funktion die bei Datenausgang aufgerufen wird. Datenausgabe wird extern durch I ² C-Master bestimmt!
<code>Wire.write(uint8_t* data, uint8_t length)</code>	Daten auf dem Bus schreiben nach Request durch den Master. Wird im Rumpf des onRequest-handler ausgeführt. Übergabeparameter: - data: BYTE-Array mit Sendedaten - length: Anzahl der zu sendenden BYTE
<code>Wire.available()</code>	Aktuellen Dateneingang abfragen. TRUE = BYTE erhalten
<code>Wire.read()</code>	Erhaltenes BYTE auslesen. RETURN = BYTE

RFM12B.h

Bibliothek: RFM12B.h	
Lizenz: http://opensource.org/licenses/mit-license.php	
Bibliothek - Funktionen	Beschreibung
<code>Initialize(uint8_t nodeId, uint8_t freqBand, uint8_t groupID)</code>	Initialisierungs-Funktion des RFM12B Moduls. Übergabeparameter: - <code>nodeid</code> : Adresse des Funkmoduls (Zwischen 0 und 31) (Adresse 0 fungiert als Broadcast-Adresse) - <code>freqBand</code> : Funkfrequenz Bibliothek-Definitionen: RFM12_433MHZ = 433 Mhz RFM12_868MHZ = 869 Mhz RFM12_915MHZ = 915 Mhz (USA) - <code>groupID</code> : Netzwerkadresse (zwischen 0 und 255)
<code>Encrypt(const uint8_t key)</code>	Aktivierung der Datenverschlüsselung. Der Übergabeparameter Key vom Typ <code>uint8_t</code> legt den Schlüssel zur Entschlüsselung fest.
<code>Send(uint8_t nodeId, const void* sendbuf, uint8_t sendLen, bool requestACK)</code>	Sende-Funktion zur Übertragung der Daten. Übergabeparameter: - <code>nodeid</code> : Adresse des Empfängers - <code>sendbuf</code> : Sendepuffer - <code>sendLen</code> : Länge des Sendepuffer (Anzahl Bytes) - <code>requestACK</code> : Empfangsbestätigung anfordern: TRUE = Ja
<code>Data()</code>	Return = Empfangsdaten
<code>ResceiveComplete()</code>	TRUE = Daten empfangen
<code>CRCPass()</code>	TRUE = Daten fehlerfrei erhalten
<code>ACKRequest()</code>	TRUE = ACK wurde angefordert
<code>sendACK()</code>	ACK-Bestätigung senden
<code>ACKReceived(uint8_t id)</code>	TRUE = ACK von Absender "id" erhalten

A.4.2 Oscat Network Bibliothek

Oscat Netlib212 (CoDeSys)

Bibliothek: Oscat Netlib212 (CoDeSys)	
Lizenz: Opensource (www.oscat.de)	
Bibliothek - Funktionen	Beschreibung
<p>IP_CONTROL</p> <p>Typ: Funktionsbaustein</p> <p>IN_OUT: IP_C: IP_C S_BUF: NETWORK_BUFFER R_BUF: NETWORK_BUFFER</p> <p>Input: IP: DWORD PORT: WORD TIME_OUT: TIME</p>	<p>Funktionsbaustein zur Koordination der Ethernet-Kommunikation. Wird je nach Ethernet-Anwendung mit anwendungsspezifischen Funktionsbausteinen über die IN_OUT-Variablen ergänzt bzw. vernetzt. Übergabeparamter:</p> <ul style="list-style-type: none"> - IP_C: Parametrierungsdaten - S_BUF: Sendedaten - R_BUF: Empfangsdaten - IP: kodierte IP - PORT: Portnummer der IP Adresse - TIME_OUT: Überwachungszeit
<p>HTTP_GET (Modifiziert)</p> <p>Typ: Funktionsbaustein</p> <p>IN_OUT: URL_DATA: URL IP_C: IP_C S_BUF: NETWORK_BUFFER R_BUF: NETWORK_BUFFER</p> <p>INPUT: GET: BOOL TYP: INT UNLOCK_BUF: BOOL</p> <p>OUTPUT: STATUS: STRING DONE: BOOL</p>	<p>Der modifizierte Funktionsbaustein führt einen HTTP-Request aus. Die Lizenz-Beschreibung von Oscat erlaubt eine Modifizierung der Bausteine für den individuellen Anwendungsbereich. Vorliegende Bausteinbeschreibung betrifft lediglich die Anwendung zur Kommunikation mit dem Z-Wave-Server und beschränkt sich dabei auf die verwendeten Parameter. Übergabeparameter:</p> <ul style="list-style-type: none"> - URL_DATA: URL-Adresse (mit STRING_TO_URL zu Datentyp URL konvertiert) - IP_C: Parametrierungsdaten - S_BUF: Sendedaten - R_BUF: Empfangsdaten - GET: TRUE startet den HTTP-Request - TYP: 0: Return-Auswertung für Z-Wave-Steckdose 1: Return-Auswertung für Z-Wave-Sensor 2: Return-Auswertung für Z-Wave-DoppelTaster - STATUS: Ausgabe entsprechend TYP-Konfiguration - DONE: TRUE = HTTP-Request erfolgreich beendet
<p>READ_HTTP</p>	<p>READ_HTTP wird vom Funktionsbaustein HTTP_GET verwendet und ist für die Auswertung des HTTP-Headers zuständig.</p>

Abkürzungsverzeichnis

ACK Zustimmung (Acknowledgment)

ADC Analog Digital Converter

DC Gleichstrom (Direct Current)

DIP Bauform parallel angeordneter Anschlussreihen (Dual in-line package)

DynDNS dynamische Aktualisierung der Domäne (Dynamic Domain Name System)

EEPROM elektrisch, löschbarer, programmierbarer Nur-Lese-Speicher (electrically erasable programmable read-only Memory)

EIB Europäischer Installationsbus

FSK Frequenzumtastung (Frequency Shift Keying)

FTDI Future Technology Devices International

GPIO Allzweckeingabe/-ausgabe (General Purpose Input/Output)

GUI Grafische Benutzerschnittstelle (Graphical User Interface)

HTML Hypertext Auszeichnungssprache (Hypertext Markup Language)

HTTP Hypertext-Übertragungsprotokoll (Hypertext Transfer Protocol)

ICs Integrierte Schaltkreise (Integrated Circuit)

IDE Integrierte Entwicklungsumgebung (Integrated Development Environment)

IoT Internet der Dinge (Internet of Things)

IP Internet Protokoll (Internet Protocol)

I²C Serieller Datenbus, I-Quadrat-C (Inter-Integrated-Bus)

JSON Kompaktes Datenformat zum Zweck des Datenaustausches zwischen Anwendungen (JavaScript Object Notation)

LSB Niedrigwertige Bit (least significant bit)

MOSI Master Out Slave In

- MISO** Master In Slave Out
- MSB** Höchstwertige Bit (most significant bit)
- NACK** Ablehnung (Negativ-Acknowledgment)
- OOP** Objektorientierte Programmierung
- OSGi** Open Services Gateway initiative
- PIN** Metallischer Anschlussstift auf einer Leiterplatte
- PWM** Pulsweitenmodulation (Verzeichnisdienst von Microsoft Windows Server)
- SCL** Serielltes Taktsignal (Serial Clock)
- SCLK** Serielltes Taktsignal (Serial Clock)
- SDA** Serielltes Datensignal (Serial Data)
- SPI** Synchron, serieller Datenbus (Serial Periphel Interface)
- SSH** Verschlüsseltes Netzwerkprotokoll (Secure Shell)
- SSL** Hybrides Verschlüsselungsprotokoll (Secure Socket Layer)
- TCP** Übertragungssteuerungsprotokoll (Transmission Control Protocol)
- TTL** Transistor-Transistor-Logik
- TWI** Konform zu I²C-Bus (Two Wire Interface)
- UART** Serielle Schnittstelle (Universal Asynchron Receiver Transmitter)
- UML** Vereinheitlichte Modellierungssprache (Unified Modeling Language)
- URL** einheitlicher Ressourcenanzeiger (Uniform Resource Locator)
- USB** Universelle Bus Schnittstelle (Universal Serial Bus)
- W-LAN** Drahtloses, lokales Netzwerk (Wireless, Local Area Network)

Literaturverzeichnis

- [1] ERIK BARTMANN: *Die elektronische Welt mit Arduino entdecken*
O'Reilly Verlag, 2. Auflage 2014
- [2] ERIK BARTMANN: *Die elektronische Welt mit Raspberry Pi entdecken*
O'Reilly Verlag, 1. Auflage 2013
- [3] DAVID KUSHNER: *The Making of Arduino*
<http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino> (besucht am 01.08.2015)
- [4] ARDUINO: *Homepage*
www.arduino.cc (besucht am 01.08.2015)
- [5] RASPBERRY: *Homepage*
www.raspberrypi.org (besucht am 05.08.2015)
- [6] MARTIN BELAM: *The Raspberry Pi*
www.theguardian.com/commentisfree/2012/feb/29/raspberry-pi-childrens-programming (besucht am 05.08.2015)
- [7] KARIN ZÜHLKE: *Farnell zeigt den Raspberry Pi Nachwuchs*
www.elektroniknet.de/distribution/design-in/artikel/95026/ (besucht am 05.08.2015)
- [8] ELEKTRONIK KOMPEDIUM: *Raspberry Pi*
www.elektronik-kompedium.de/sites/raspberry-pi/2002061.htm (besucht am 05.08.2015)
- [9] RASPBERRY: *Raspberry Pi Model B+*
www.raspberrypi.org/documentation/hardware/raspberrypi/
(besucht am 05.08.2015)
- [10] RASPBERRY PI LAB: *GPIO Pins Model B+*
www.raspilab.wordpress.com/2014/08/06/gpio-pins-bei-modell-b-ab-juli-2014/
(besucht am 10.08.2015)

- [11] NXP SEMICONDUCTORS: *I²C-bus specification and user manual*
http://www.nxp.com/documents/user_manual/UM10204.pdf
Rev.6-4 April 2014 (download am 15.07.2015)
- [12] TELOS SYSTEMENTWICKLUNG GMBH: *I²C-Bus*
www.i2c-bus.org (besucht am 25.07.2015)
- [13] HELMUT BÄHRING: *Anwendungsorientierte Mikroprozessoren*
Springer-Verlag, 4. Auflage 2010
- [14] MICROCOMPUTERTECHNIK: *SPI-Serial Peripheral Interface*
www.mct.de/faq/spi.html (besucht am 12.08.2015)
- [15] Z-WAVE ALLIANZ (EU): *Homepage*

[urlwww.zwave.de](http://www.zwave.de) (besucht am 22.08.2015)
- [16] TEXAS INSTRUMENT: *ULN2803 Darlington Array*
www.ti.com/lit/ds/symlink/uln2803a.pdf (besucht am 28.08.2015)
- [17] TEXAS INSTRUMENT: *Step-Down Regulator LM2596:*
www.ti.com/lit/ds/symlink/lm2596.pdf (besucht am 28.08.2015)
- [18] HOPERF ELECTRONIC: *Funkmodul RFM12B*
www.rcscomponents.kiev.ua/datasheets/RFM12B.pdf
(besucht am 28.08.2015)
- [19] CODESYS: *CoDeSys Control for Raspberry Pi SL*
<http://store.codesys.com/codesys-control-for-raspberry-pi-sl.html>
(besucht am 28.08.2015)
- [20] Z-WAVE.ME: FUNKMODUL RAZBERRY
<http://www.raspberry.z-wave.me> (besucht am 20.08.2015)
- [21] ELEKTRO.NET: *Funkkommunikation mit Smart Home*
<http://www.elektro.net/38666/z-wave/> (besucht am 18.07.2015)
- [22] ARDUINO: *Arduino UNO*
www.arduino.cc/en/Main/ArduinoBoardUno (besucht am 26.08.2015)
- [23] ATMEL: *ATmega328P*
<http://www.atmel.com/images/doc8161.pdf>
(besucht am 11.10.2015)
- [24] MICROCHIP: *16-Bit I/O Expander MCP23017*
<http://ww1.microchip.com/downloads/en/DeviceDoc/21952b.pdf>
(besucht am 04.09.2015)

- [25] AOSONG ELECTRONIC Co.: *Sensor DHT22*
<https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf> (besucht am 20.09.2015)
- [26] SMART HOME WELT
<http://smarthomewelt.de/> (besucht am 14.08.2015)
- [27] MARK WEISER: *The Computer for the 21th Century*
www.ics.uci.edu/~corps/phaseii/Weiser-Computer21stCentury-SciAm.pdf (besucht am 28.08.2015)
- [28] ZIGBEE ALLIANCE: *Homepage*
www.zigbee.org/ (besucht am 28.08.2015)
- [29] ENOCEAN: *Homepage*
www.enocean.com (besucht am 28.08.2015)
- [30] EQ-3: *RWE und HomeMatic*
www.eq-3.de (besucht am 28.08.2015)
- [31] INNOQ: *Einführung in die Heimautomatisierung mit openHAB*
www.innoq.com/de/articles/2014/11/welten-verbinden
(besucht am 28.08.2015)
- [32] FHEM: *Homepage*
<http://fhem.de/> (besucht am 28.08.2015)
- [33] Z-WAVE.ME: *Z-Way Developers Dokumentation*
<http://rasberry.z-wave.me/docs/zwayDev.pdf>
(besucht am 28.08.2015)

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 27. November 2015

Ort, Datum

Unterschrift