Bachelorarbeit

**Vollautomatisierte Lüftersystem Steuerung**

vorgelegt von Ralf Hoffmann



Fakultät für Elektrotechnik und Informationstechnik

Elektro und Informationstechnik (EI)

|  |  |
| --- | --- |
| Betreuer: | Prof. Dr. Klemens Graf |
| Laufende Nummer: | 2172 |
| Bearbeitungsbeginn: | 08.11.2021 |
| Abgabetermin: | 08.07.2022 |

Ein Bild, das Text, ClipArt enthält.

Automatisch generierte Beschreibung

|  |  |
| --- | --- |
| Firma: | Robert Hoffmann - SolarStrom |
| Betreuer: | Robert Hoffmann |

Bachelorarbeit

**Vollautomatisierte Smart Home Lüftersystem Steuerung**

**- Hardware und Softwareentwicklung**

**Fully automated smart home ventilation system control**

**- hardware and software development**

vorgelegt von Ralf Hoffmann

an der

Hochschule München

Fakultät für Elektrotechnik und Informationstechnik

Elektrotechnik und Informationstechnik (EI)

|  |  |
| --- | --- |
| Betreuer: | Prof. Dr. Klemens Graf |
| Laufende Nummer: | 2172 |
| Bearbeitungsbeginn: | 08.11.2021 |
| Abgabetermin: | 08.07.2022 |

|  |  |
| --- | --- |
| Firma: | Robert Hoffmann - SolarStrom |
| Betreuer: | Robert Hoffmann |

Erklärung des Bearbeiters

Name: Hoffmann  
Vorname: Ralf

Ich erkläre hiermit, dass ich die vorliegende Bachelorarbeit selbständig verfasst und noch nicht anderweitig zu Prüfungszwecken vorgelegt habe.

Sämtliche benutzte Quellen und Hilfsmittel sind angegeben, wörtliche und sinngemäße Zitate sind als solche gekennzeichnet.

Ottobrunn, den 07.07.2022 Unterschrift

Kurzfassung

Diese Arbeit befasst sich mit der Hardware- und Softwareentwicklung einer Lüftersystem Steuerung auf KNX-Standard.

Ziel der Bachelorarbeit war es, eine vollautomatisierte Smart Home Lüftersystem-Steuerung bestehend aus HW und SW von Grund auf neu zu entwerfen und in Betrieb zu nehmen. Das Lüftersystem besteht aus 8 individuell steuerbaren Lüftern, die über einen PI-Regler stufenlos die Raumfeuchtigkeit reduziert. Implementiert wird das Projekt als KNX-Aktor.

Sämtliche Planungs-, Entwicklungs- und Integrationsaufgaben wurden dabei durchgeführt, dazu gehören:   
HW-Entwicklung des Schaltplans, Platinenlayouts, Bestückung und Inbetriebnahme   
SW-Entwicklung der KNX-Kommunikation, Regelung und Lüfteransteuerung.

Abstract

This thesis is about the hardware and software development of a ventilation system control on basis of open standard KNX.

Goal of the bachelor thesis was to develop the HW and SW of a smart home ventilation control system from scratch and bring up the system. The ventilation system consists of 8 individually controllable vents, which reduce the room humidity continuously through a pi-controller. The project is implemented as a KNX actuator.

All planning, development and integration work was accomplished and includes:

HW development of the schematic, PCB-layout, manufacturing and commissioning

SW development of the KNX communication and vent control.

Inhaltsverzeichnis

Erklärung des Bearbeiters III

Kurzfassung IV

Abstract IV

Inhaltsverzeichnis V

Abbildungsverzeichnis VIII

Tabellenverzeichnis IX

Formelverzeichnis IX

1 Einleitung 1

1.1 Smart Home 1

1.2 Ziel und Anforderung der Arbeit 2

2 Hardware 3

2.1 HW-Toolchain 3

2.2 Beschreibung des Komplettsystems 4

2.2.1 KNX – Bussystem 5

2.2.1.1 Übertragungsmedien 5

2.2.1.1.1 Twisted Pair 5

2.2.1.1.2 Powerline 6

2.2.1.1.3 Radio Frequency 7

2.2.1.1.4 IP 7

2.2.2 Lüfter 8

2.2.3 Sensor 10

2.2.4 Selbstentwickelter Aktor 10

2.2.4.1 Ansteuerprinzip 10

2.2.4.2 Schaltplan 11

2.2.4.2.1 Treiberschaltung 12

2.2.4.2.2 Selektorschaltung 15

2.2.4.2.3 Mikrocontrollerschaltung 18

2.2.4.3 Layout 19

2.2.4.3.1 Bauteilauswahl 19

2.2.4.3.2 Design 20

3 Software 23

3.1 SW-Toolchain 23

3.2 Mikrocontroller 24

3.2.1 Programmierschnittstelle 24

3.2.2 Systemarchitektur 25

3.2.3 Takteinstellungen 27

3.3 Uart 29

3.3.1 Protokoll 29

3.3.2 UART vs. USART 31

3.3.3 Implementierungsmöglichkeiten 32

3.3.3.1 Polling Methode 32

3.3.3.2 Interrupt/Callback 32

3.3.3.3 Direct Memory Access DMA 32

3.3.4 Konfiguration 32

3.4 Timer 33

3.4.1 Pulsweitenmodulation PWM 33

3.4.2 LPTIM 36

3.5 Softwarestruktur 38

3.6 Programmablauf 39

3.6.1 Initialisierung & Lüfteransteuerung 39

3.6.1.1 Funktionen 41

3.6.1.1.1 start\_all\_timers / stop\_all\_timers 41

3.6.1.1.2 reset\_all\_pwm / reset\_pwm\_not\_controlgroup 41

3.6.1.1.3 set\_all\_pwm / set\_pwm\_not\_controlgroup 42

3.6.1.1.4 toggle\_all\_gpios/toggle\_gpios\_not\_controlgroup 42

3.6.1.1.5 pi\_controller 42

3.6.2 KNX-Kommunikation 42

3.6.2.1 Telegramm Aufbau 43

3.6.2.2 UART-Statemachine 45

3.6.2.3 Funktionen knx\_receive\_telegram Bibliothek 48

3.6.2.3.1 add\_listening\_group\_address 48

3.6.2.3.2 check\_interest 48

3.6.2.3.3 check\_for\_controlbyte 48

3.6.2.3.4 extract\_data 48

3.6.2.3.5 get\_2byte\_float\_value 49

3.6.3 HAL-Treiber 50

3.6.3.1 HAL\_UART\_Receive\_IT 50

3.6.3.2 HAL\_UARTRxCpltCallback 50

3.6.3.3 HAL\_UART\_Transmit\_IT 50

3.6.3.4 HAL\_TIM\_PWM\_Start 51

3.6.3.5 HAL\_TIM\_PWM\_Stop 51

3.6.3.6 HAL\_TIM\_SET\_COMPARE 51

3.6.3.7 HAL\_LPTIM\_Counter\_Start\_IT 52

3.6.3.8 HAL\_LPTIM\_Counter\_Stop\_IT 52

3.6.3.9 HAL\_LPTIM\_AutoReloadMatchCallback 52

3.6.3.10 HAL\_GPIO\_TogglePin 52

3.7 Logging 53

3.8 Regler 55

3.8.1 Allgemein 55

3.8.2 Sinnvolle Reglertypen 56

3.8.2.1 Zweipunkt-Regler 56

3.8.2.2 P-Regler 56

3.8.2.3 PI-Regler 57

3.8.2.4 PID-Regler 58

3.8.3 Realisierung 59

3.8.3.1 Auswahl des Reglers 59

3.8.3.2 Softwaretechnische Umsetzung 60

4 Fazit 63

4.1.1 Aktueller Stand der Arbeit 63

4.1.2 To-Do Liste 63

4.1.3 Empfehlungen für Hardware 2.0 63

4.1.4 Schwierigkeiten & Probleme 64

4.1.5 Ausblick 64

5 Literaturangaben 65

6 Anhang 70

Abbildungsverzeichnis

[**Abbildung 1**: Aufbau des Gesamtsystems 4](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058526)

[**Abbildung 2**: Schalldämmlüfter inVENTer iV14-Zero [56] 8](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058527)

[**Abbildung 3**: Aufbau und Funktionsweise eines PMDC [4] 9](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058528)

[**Abbildung 4**: PWM-Signal [11] 10](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058529)

[**Abbildung 5:** Aufbau und Funktionsweise PMOS [12] 12](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058530)

[**Abbildung 6**: Treiberschaltung (Motor hier nur mit einem Ground Anschluss) 13](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058531)

[**Abbildung 7**: H-Brücke [17] 15](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058532)

[**Abbildung 8**: Selektorschaltung 16](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058533)

[**Abbildung 9**: zusammengesetzte Schaltung (EAGLE Schematic) 17](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058534)

[**Abbildung 10**: Bus Coupling Unit ohne Gehäuse [21] 19](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058535)

[**Abbildung 11**: 4-lagiger PCB-Stackup [23] 20](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058536)

[**Abbildung 12**: Designrichtlinie IPC-2152 [24] 21](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058537)

[**Abbildung 13**: 3D-Rendering der entwickelten Hardware 22](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058538)

[**Abbildung 14**: ST-Link/V2 Programmer [26] 24](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058539)

[**Abbildung 15**: Systemarchitektur [27, S.49] 26](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058540)

[**Abbildung 16**: Clock-Tree aus STM32CubeMX 27](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058541)

[**Abbildung 17**: praktischer Aufbau UART [32] 30](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058542)

[**Abbildung 18**: Aufbau UART-Protokoll [33] 30](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058543)

[**Abbildung 19**: Generation eines center aligned PWM-Signals [27, S.398] 34](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058544)

[**Abbildung 20**: Timer-Konfiguration und Pinbelegung in STM32CubeMX 36](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058545)

[**Abbildung 21**: Low-Power-Timer Interrupt [43](modifiziert) 37](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058546)

[**Abbildung 22**: Software-Stack 38](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058547)

[**Abbildung 23**: Ablauf der Process-Statemachine 39](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058548)

[**Abbildung 24**: KNX Telegramm Aufbau [43, S.8] 43](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058549)

[**Abbildung 25**: Kontrollfeld 43](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058550)

[**Abbildung 26**: Adressfeld 44](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058551)

[**Abbildung 27**: Organisation eines KNX-Systems mit Gruppenadressen [45] 44](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058552)

[**Abbildung 28**: Ablauf der UART-Statemachine 46](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058553)

[**Abbildung 29**: Datenextraktion eines 2Byte Floatwerts 49](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058554)

[**Abbildung 30**: Blockschaltbild eines geschlossenen Regelkreises 55](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058555)

[**Abbildung 31**: Blockschaltbild eines PID-Reglers in Parallelstruktur 58](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058556)

[**Abbildung 32**: Regler Sprungantworten [53] 59](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058557)

[**Abbildung 33**: Software-Implementierung des PI-Reglers 61](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058558)

[**Abbildung 34**: Zusammenhang zwischen Feuchtigkeitswert und Zustand 62](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108058559)

Tabellenverzeichnis

[**Tabelle 1**: Bootmodi des Mikrocontrollers [27, S.56] 25](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108082426)

[**Tabelle 2**: Arduino- Logging 54](file:///C:\Users\ralfh\OneDrive\Desktop\BA_Ralf_Hoffmann.docx#_Toc108082427)

Formelverzeichnis

[**Formel 1**: Spannung über die Spule 18](#_Toc108058602)

[**Formel 2**: Dutycycle [37, S.56] 35](#_Toc108058603)

[**Formel 3**: Autoreload-Register PWM [37, S.54] 35](#_Toc108058604)

[**Formel 4**: Schaltverluste eines Mosfets [40, S.2-3] 35](#_Toc108058605)

[**Formel 5**: Autoreload Register LPTIM [42] 37](#_Toc108058606)

[**Formel 6**: Regeldifferenz [48] 55](#_Toc108058607)

[**Formel 7**: P-Regler [47, S.28] 56](#_Toc108058608)

[**Formel 8**: PI-Regler [47, S.41] 57](#_Toc108058609)

[**Formel 9**: PID-Regler [47, S.43] 58](#_Toc108058610)

[**Formel 10**: Diskreter I-Anteil [48] 60](#_Toc108058611)

# Einleitung

## Smart Home

Mit der fortschreitenden technischen Entwicklung wird unser Leben immer weiter digitalisiert. Das Smartphone zum Beispiel ist mittlerweile aus unserem Alltag gar nicht mehr wegzudenken. Dieser Entwicklungsprozess färbt dabei auch auf unser Zuhause ab, wo das Smartphone zu einem zentralen Bedienelement geworden ist.

Unter Smart Home, was zu Deutsch so viel wie „intelligentes Wohnen“ heißt, versteht man die Vernetzung von mehreren verschiedenen Haushaltsgeräten in einem einheitlichen System. Diese werden dann vom einem Bedienelement zentral gesteuert. Dadurch können die teilnehmenden Geräte miteinander kommunizieren und so alltägliche Abläufe aufeinander abstimmen und automatisieren. Infolgedessen kann ein erweiterter Wohnkomfort in den Bereichen Energieverbrauch, Sicherheit und Entertainment geschaffen werden.

Die Energieeffizienz des Hauses wird gesteigert, in dem nur geheizt wird, wenn auch jemand daheim ist oder sich gerade auf dem Weg dahin befindet. Sensoren an den Fenstern erkennen, dass gerade gelüftet wird und deshalb in diesem Raum die Heizung heruntergeregelt werden muss.

Mit einem Smart Home System in Kombination mit einer Photovoltaikanlage, kann die Nutzung der produzierten erneuerbaren Energie optimiert und diese direkt verwendet werden. Um die Mittagszeit, wenn Sonnenstrom erzeugt wird, kann das Smart Home System die Verbraucher, wie etwa Spülmaschine, Waschmaschine, Wallbox, etc. stufenweise einschalten. Dadurch wird vermieden, dass Strom zurück ins Netz eingespeist und der Eigenverbrauch gesteigert wird. Ein hoher Eigenverbrauchsanteil an erneuerbarer Energie erhöht die Wirtschaftlichkeit.

Durch schaltbare Steckdosen wird beispielsweise verhindert, dass Geräte im Standby Modus weiter Energie verbrauchen. Zudem wird dadurch auch die Sicherheit erhöht, da so ein aus Versehen eingeschalteter Herd der Vergangenheit angehört.

Befindet man sich im Urlaub, lässt man zum Beispiel das Haus zur Einbruchsprävention durch eine automatisierte Rollläden- und Lichtsteuerung weiterhin belebt aussehen.

Die Sicherheit im Haus kann erheblich verbessert werden, indem man Rauchmelder so konfiguriert, dass sie bei einem Alarm auch die übrigen Rauchmelder aktivieren und zusätzlich das Licht im kompletten Haus einschalten.

Mit Multiroom-Lautsprechern lässt sich der Komfort im Bereich des Entertainments erhöhen. Im ganzen Haus wird so dasselbe Lied mit einer auf die jeweilige Zimmergröße angepassten Lautstärke wiedergegeben.

Smart Homes werden in naher Zukunft unweigerlich zur Norm werden. Die Anwendungsmöglichkeiten sind dabei nahezu unbegrenzt und Anpassungen an die Bewohnerbedürfnisse können flexibel, auch nach Erstinstallation vorgenommen werden.

## Ziel und Anforderung der Arbeit

Für die Gerätehersteller ist es schwierig eine Universallösung bereit zu stellen, da kein Haus dem anderen gleicht. Die Funktionen der Komponenten entsprechen oftmals nicht den individuellen Wünschen des Kunden. Nehmen wir zum Beispiel die Ankaufoption der Lüftersteuerung. Bei ihr lassen sich die Lüfter immer nur in Gruppen von jeweils vier Lüftern ansteuern. Außerdem lassen sich nur eine Handvoll Geschwindigkeitsstufen einstellen.

KNX-Aktoren sind meist sehr kostspielig und deshalb, kam die Idee auf eine eigene Hardware zu entwickeln, die eine deutlich flexiblere Alternative darstellen soll. Durch ihre individuelle Anpassbarkeit auf die Bedürfnisse des Nutzers, stellt sie eine universelle Lösung für KNX-Lüftersysteme dar. Das Endprodukt soll dabei die folgenden Anforderungen erfüllen:

* Individuelle Ansteuerung von 8 verschiedenen Lüftern
* Gewährleistung eines stufenlosen Lüfterbetriebs
* Dynamische Geschwindigkeitsanpassung
* Richtungswechsel nach vorgegeben Zeitintervall
* Hohe Laufruhe
* Ankopplung und Kommunikation mit KNX-Bus
* Ermöglichen von nachträglichen Anpassungen auch nach Erstinstallation

# Hardware

Dieses Kapitel gibt einen Überblick über das Gesamtsystem sowie die entwickelte Hardware und ihre Designvarianten.

## HW-Toolchain

**LTspice**

Die ersten Konzeptschaltungen wurden mit dem Simulationsprogramm LTspice verifiziert.

**EAGLE**

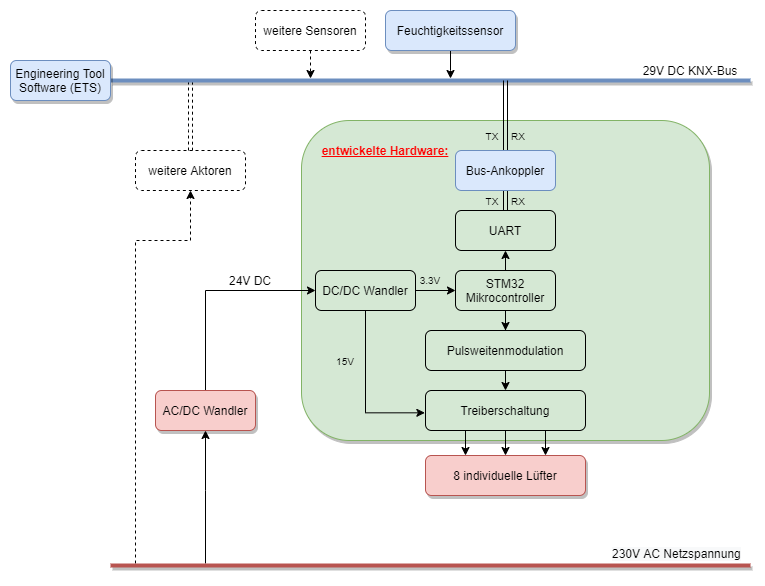
Der Schaltplan sowie das Platinenlayout wurde mit der, für Studenten kostenfreien EDA-Software Autodesk EAGLE erstellt.

**Fusion 360**

Mit der Autodesk Erweiterung Fusion 360 wurde das 3D-Modell der Platine gerendert.

## Beschreibung des Komplettsystems

In **Abbildung 1** sieht man die Darstellung des Gesamtsystems. Die Teilnehmer des Feldbusses lassen sich grundsätzlich in Sensoren und Aktoren unterteilen. Die Sensoren messen dabei eine physikalische Größe im Haus, wie in diesem Projekt die Feuchtigkeit und wandeln sie in eine elektrische Größe um. Diese Information wird dann in Form eines Telegramms an den zugehörigen Aktor weitgeleitet. Der Aktor verarbeitet den erhaltenen Wert und handelt indem er die Lüftergeschwindigkeit anpasst. Die Aktoren beziehen dabei ihre Versorgungsspannung jeweils aus dem Wechselstromnetz und kommunizieren über den KNX-Bus. Mit der Engineering Tool Software (ETS) lassen sich alle Busteilnehmer verwalten.



**Abbildung 1**: Aufbau des Gesamtsystems

### KNX – Bussystem

Der KNX-Bus ist eine weiterentwickelte Version des Europäischen Installationsbusses EIB, der gezielt für die Gebäudeautomation optimiert wurde. Dabei sind die beiden Feldbusse voll kompatibel zueinander. KNX ist mittlerweile der anerkannte internationale Standard, sowohl in Europa als auch in China und den USA. Von 450 Partnerfirmen in 45 Ländern wird eine breite Palette an Produkten auf dem KNX-Standard entwickelt und gefertigt. [1, S.4-5]

Da auf jedem Busteilnehmer ein Mikroprozessor integriert ist, wird kein zentrales Steuergerät benötigt was von Vorteil ist, weil bei einem Defekt nur das betroffene Gerät nicht funktioniert, dies aber keinen Einfluss auf die restlichen Teilnehmer hat. Dieser dezentrale Aufbau des Feldbusses ist ausgesprochen wichtig, da KNX-Systeme mit bis zu 50.000 Teilnehmern enorm groß werden können. [1, S.5]

#### Übertragungsmedien

KNX bietet unterschiedliche Übertragungsmedien an, um ein möglichst breites Spektrum an Kunden anzusprechen sowie eine nachträgliche Installation des Busses zu ermöglichen. [1, S.5]

Das KNX-System wurde auf Basis der Zweidrahtleitung entwickelt, da es sich um das meistverbreitete Übertragungsmedium handelt. Im Folgenden wird auf die verschiedenen Übertragungsmöglichkeiten kurz eingegangen.

##### Twisted Pair

Zusätzlich zu der 230V AC-Leitung wird noch eine weitere KNX-Leitung zur Datenübertragung in der Hausinstallation benötigt. Sie besteht aus einem verdrillten zweiadrigen Draht, mit dem alle Teilnehmer verbunden sind und mit einer 29V DC Steuerspannung versorgt werden. Die Bus-Teilnehmer funktionieren in einem erweiterten Spannungsbereich von 21V bis 30V fehlerfrei, womit ein reibungsloser Betrieb, auch bei Spannungseinbrüchen gewährleistet wird.

Die Datenübertragung erfolgt hierbei erdfrei beziehungsweise symmetrisch, was bedeutet, dass die Signale keinen Referenzpunkt zur Masse haben, sondern nur die Spannungsdifferenz zwischen den beiden Leitungen relevant ist. Dadurch erhält man eine bessere Störfestigkeit. Störungen auf beiden Leitungen eliminieren sich gegenseitig.

Die Teilnehmer werden mit einer Steckklemme an die Datenleitung angeschlossen und können während des Betriebs einfach entfernt werden, ohne dass die Kommunikation der übrigen Teilnehmer gestört wird.

Die Daten werden seriell und asynchron in Form von Telegrammen übertragen. Das bedeutet, sie werden Byte für Byte gesendet. Hierbei ist eine maximale Übertragungsgeschwindigkeit von bis zu 9600 Bit/s möglich. Der genaue Aufbau eines KNX-Telegramms wird im späteren Kapitel *Telegramm Aufbau* noch explizit erklärt. [1, S.6]

Telegramme können grundsätzlich nur nacheinander gesendet werden. Falls zwei   
Telegramme genau zeitgleich an dieselbe Adresse gesendet werden, kommt es zu einer Kollision. Da jeder Teilnehmer dauerhaft den Datenverkehr auf dem Bus mitliest, können Kollisionen nach dem CSMA/CA Prinzip verhindert werden (Carrier Sense Multiple Access/Collision Avoidance). [1, S.7]

Jeder Teilnehmer prüft vor dem Senden, ob der Bus frei ist. Ist dies der Fall, wird das Telegramm verschickt. Sollte der Bus im Moment besetzt sein, wird die Nachricht zunächst zurückgestellt und dann etwas später erneut gesendet. Dadurch verzögern sich die Daten im Kollisionsfall nur, gehen jedoch nicht verloren. [2]

Die Priorität des Telegramms lässt sich im jeweiligen Kontrollbyte festlegen. Im Falle gleicher Prioritäten hat die „0“ wie bei allen digitalen Systemen Vorrang vor der „1“. [1, S.7]

##### Powerline

Eine kostengünstigere Alternative stellt die KNX-Powerline dar. Hierbei müssen keine weiteren Leitungen eingezogen werden, weil die Busteilnehmer über das vorhandene 230V-Wechselstromnetz kommunizieren. Als Übertragungsmedium wird einer der   
3-Phasenleiter und der Neutralleiter als Massebezugspunkt verwendet. Zur Datenübertragung werden durch Modulation bestimmter Frequenzen, Einser und Nullen erzeugt und auf den Phasenleiter aufgeprägt.

* 105,6 kHz entspricht einer „0“
* 110 kHz entspricht einer „1“

Bei dieser Methode ist nur eine geringere Übertragungsgeschwindigkeit von  
1200 Bit/s zu erreichen. [1, S.7-8]

##### Radio Frequency

Eine weitere Möglichkeit ist die drahtlose Datenübertragung über Funk. Es bietet sich besonders dann an, wenn es schwierig oder unmöglich ist im Gebäude neue Leitungen einzuziehen wie beispielsweise in Lagerhallen. Die Aktoren werden dabei   
standardmäßig am 230V-Netz angeschlossen. Die Kommunikation wird mit   
batteriebetriebenen und daher vom Wechselstromnetz entkoppelten Hochfrequenzsensoren realisiert.

Durch Frequenzmodulation wird das niederfrequente Datensignal auf ein hochfrequentes Trägersignal aufgeprägt und in der Frequenz leicht verändert. Dies ermöglicht es das Datensignal hochfrequent zu übertragen. Der Empfänger erhält dann durch   
Demodulation das ursprüngliche Nutzsignal. Auch hier wird geprüft ober der jeweilige Funkkanal frei ist, um Kollisionen von Nachrichten zu vermeiden. [1, S.8-10]

##### IP

Die letzte Möglichkeit ist es eine KNX-Übertragung über das Netzwerk zu implementieren. Hierbei werden folgende Protokolle benötigt (Reihenfolge wie im ISO-Stack):

* User Datagram Protocol (UDP).
* Transmission Control Protocol (TCP)
* Internet Protocol (IP)
* Ethernet

Dies ist komplex und aufwendig, deshalb wird auf die IP-Realisierung nicht weiter eingegangen. Vorteil davon ist natürlich, dass man KNX-Systeme auch von außerhalb des Gebäudes parametrisieren und warten kann. [1, S.10-11]

### Lüfter

Das Lüftersystem setzt sich aus 8 Schalldämmlüftern der Marke inVENTer zusammen. Bei dem verwendeten Modell handelt es sich um den „iV14-Zero“, welcher von einem permanenterregten Gleichstrommotor (PMDC) mit einer Leistungsaufnahme von 3W betrieben wird. Angesteuert wird er mit einer Gleichspannung zwischen   
6 und 16V. Mit einem geräuscharmen Betrieb von 56 dB ist der Lüfter ideal für die Nutzung in Wohngebäuden. [3]



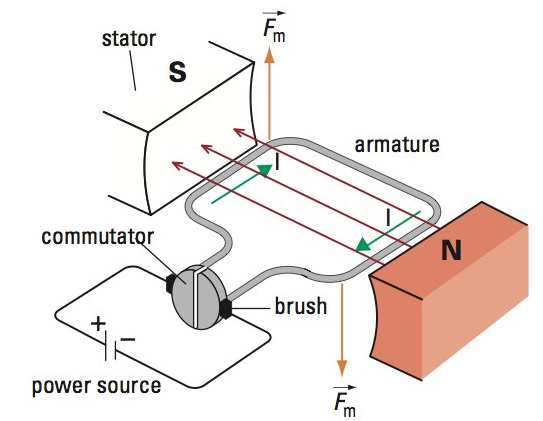
**Abbildung 2**: Schalldämmlüfter inVENTer iV14-Zero [56]

Insgesamt besitz jeder Lüfter drei Anschlüsse, einen für die positive Spannung und zwei Masseanschlüsse, durch die der Betrieb in beide Richtungen gestattet wird. Dies stellt eine Besonderheit dar, weil permanenterregte Gleichstrommotoren in der Regel nur zwei Anschlüsse besitzen.

Grundsätzlich besteht ein PMDC aus folgenden Komponenten (siehe **Abbildung 3**):

* festsitzenden Stator
* frei drehbare Anker
* Kommutator
* Kohlebürsten

Der Stator besteht hierbei namensgebend aus Permanent-Magneten. Auf dem Anker befinden sich mehrere Kupferdrahtwicklungen, die jeweils mit dem Kommutator verbunden sind. Dieser kann so einen elektrischen Strom einprägen. Die beiden Kohlebürsten sitzen auf dem rotierenden Kommutator und dienen als Schleifkontakte für die Spannungsversorgung. [5][6][7]



**Abbildung 3**: Aufbau und Funktionsweise eines PMDC [4]

Bei Anlegen einer Spannung, erzeugt die Ankerspule ein Magnetfeld und versetzt den Rotor in eine Drehbewegung. Da sich der Anker daraufhin in einer Position festzieht, muss der Kommutator die Spannung jede halbe Umdrehung mechanisch umpolen. Das kontinuierlich wechselnde Magnetfeld ermöglicht so einen fließenden Betrieb. [5][6][7]

Dieses Konzept lässt sich mit der drei Fingerregel in **Abbildung 3** verdeutlichen.

* Daumen: Richtung elektrischer Strom
* Zeigfinger: Richtung Magnetfeld
* Mittelfinger: Richtung Lorentzkraft

Spulen stellen im Gleichstromkreis einen ohmschen Widerstand dar. Erhöht sich nach dem ohmschen Gesetz die Spannung, so fließt auch ein erhöhter Strom durch die Spule. Die Folge wiederrum ist ein höheres Drehmoment und bei gleicher Last auch eine erhöhte Drehzahl. Mit der Spannung lässt sich also die Drehzahl des DC-Motors einstellen.

### Sensor

Als Sensor wurde der Raumtemperatur-/Feuchtsensor 55 der Marke MDT mit integriertem Busankoppler verwendet.

### Selbstentwickelter Aktor

Im folgenden Kapitel wird die eigens entwickelte Hardware im Detail beschrieben.

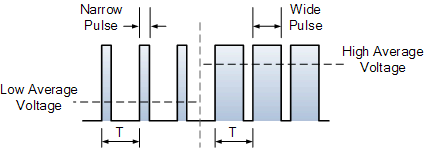
#### Ansteuerprinzip

Als Ansteuerprinzip wurde die Pulsweitenmodulation (PWM) gewählt wie im Folgenden erläutert wird.

PWM ist ein Verfahren, das es möglich macht mit einem digitalen Ausgang des Mikrocontrollers eine analoge Schaltung anzusteuern. Dabei wird vom Controller eine   
unipolare Rechteckspannung mit einem anpassbaren Tastgrad und Frequenz erzeugt.[9]

Der Dutycycle entspricht dabei dem Verhältnis zwischen der Zeit im High-Zustand und der Periode, für gewöhnlich in Prozent angegeben.

Die PWM-Frequenz gibt an wie oft pro Sekunde das Signal zwischen dem Wert 0 und der Amplitude hin und herspringt.



**Abbildung 4: PWM-Signal [11]**

Die Amplitude der Rechteckspannung entspricht dabei der Betriebsspannung und ist äquivalent zum logischen High „1“. Bei Mikrocontrollern aus der STM32 Familie beträgt sie stehts 3,3V. Der eingestellte Dutycycle ist dann direkt proportional zu der resultierenden, mittleren Spannung. Beispielsweise entspricht ein Tastgrad von 70% einer mittleren Ausgangsspannung von

.

Der Vorteil des PWM-Verfahrens in Kombination mit einem Transistor ist, dass dieser dann entweder komplett ein- oder ausgeschaltet ist. Das erhöht die Effizienz des Systems, da die meisten Verluste in Übergangszuständen zwischen an und aus generiert werden. [10][11]

Ein weiterer Vorteil der Pulsweitenmodulation ist, dass nur so viel Leistung bereitstellt, wird, wie der Verbraucher auch benötigt. [8]

Eine analoge Alternative zu PWM ist eine Transistorschaltung. Bei der, der Transistor als spannungsgesteuerter Widerstand fungiert und durch Veränderung seines Widerstands den Kollektor-Emitter-Strom steuert. Da sich der Transistorwiderstand aber seriell zur Last befindet fällt über ihn die überschüssige Spannung ab und erzeugt unnötig Wärme. [8]

Zudem brauchen Analogschaltungen eine sehr präzise Auslegung, um den gewollten Effekt zu erzielen.

Bei der Ansteuerung über ein PWM-Signal sieht der Motor immer dieselbe Spannungsamplitude. Dadurch kann er auch bei niedrigen Drehzahlen betrieben werden, ohne ins Stocken zu geraten. Das ist besonders für meinen Anwendungsfall wichtig, da die Lüfter im Standardbetrieb möglichst leise und langsam laufen sollen. [11]

#### Schaltplan

Die Hardware lässt sich in folgende Komponenten untergliedern

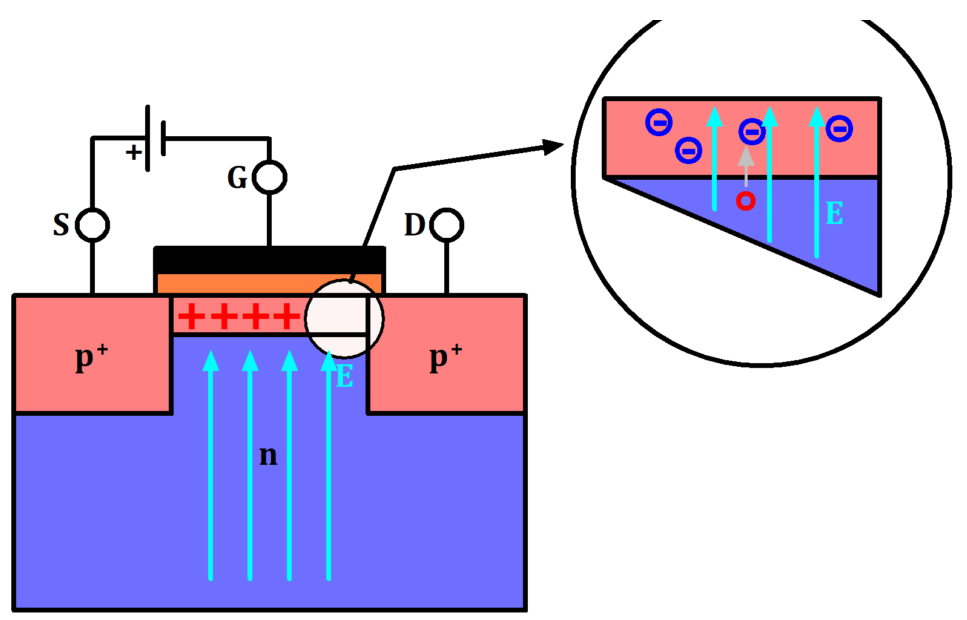
* Treiberschaltung
* Selektorschaltung
* Mikrocontroller

Durch das Auftrennen des Plus- und Minuspols am Motor wird eine bessere Übersicht, der sonst zusammenhängenden Schaltung geschaffen. Es ist anzumerken, dass für jeden Lüfter jeweils eine Treiber- und eine Selektorschaltung benötigt wird.

##### Treiberschaltung

Mit einem STM32 Mikrocontroller lassen sich, ohne zusätzliche Leitungselektronik, keine größeren Lasten betreiben. Das MCU-Signal wird deshalb durch eine Kette von Transistoren verstärkt.

Der Motor wird über einen Leistungsschalter in der Highside-Konfiguration angesteuert. Das bedeutet, dass sich der Leistungsschalter zwischen der positiven Versorgungsspannung und der Last befindet. Realisiert wird er durch einen P-Kanal Mosfet des Anreicherungstyps. [14]



**Abbildung 5:** Aufbau und Funktionsweise PMOS [12]

Solange keine Steuerspannung zwischen Gate und Source anliegt, befindet sich der Mosfet im cut-off Bereich und verhält sich wie ein Leerlauf.

Durch Anlegen einer negativen Gate-Source-Spannung bildet sich im n-dotierten Substrat ein elektrisches Feld aus, welches dafür sorgt, dass die Löcher vermehrt zur Metall-Oxidschicht gezogen werden.

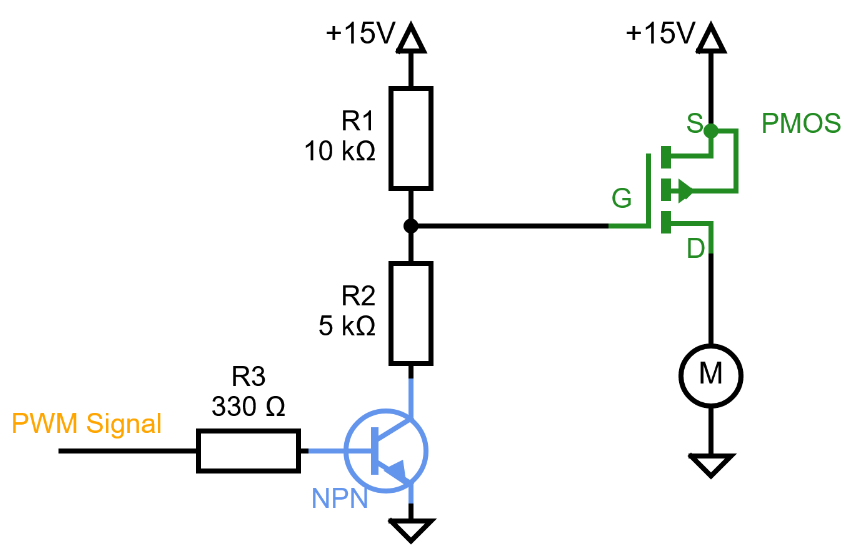
Sobald die Gate-Source-Spannung kleiner als die negative Threshold-Spannung wird (< ), ist der Zeitpunkt erreicht wo sich mehr Löcher als Elektronen (Löcherüberschuss) am Gate befinden. Nun wird von den Löchern, die sich von Source nach Drain bewegen der Ladungsfluss ermöglicht.

Durch die erneute Minderung der Gate-Source-Spannung verbreitert sich der p-Kanal weiter. Dadurch sinkt der Kanalwiderstand , die Leitfähigkeit steigt und der Drain-Source-Strom erhöht sich. [12][13][14]

Bei der NMOS Variante hingegen bildet sich durch das Anlegen einer positiven Ansteuerspannung ein elektrisches Feld aus, sodass die Elektronen aus dem p-dotierten Substrat zum Metalloxid gezogen werden und ein leitender n-Kanal entsteht. Die Gate-Source-Spannung muss hierbei größer als die positive Threshold-Spannung sein (> ). [12][13][14]

Der bipolare Logic-Level Transistor in der NPN Ausführung (siehe **Abbildung 6**) lässt sich problemlos von dem PWM-Signal des Mikrocontrollers durchschalten. Sobald die Basis-Emitter-Schwellenspannung von 0,6V erreicht wurde, fängt der NPN-Transistor an zu leiten und zieht dadurch das Gate des dahinterliegenden Feldeffekt-Transistor auf ein 15V Powerlevel. [15]

**Abbildung 6**: Treiberschaltung (Motor hier nur mit einem Ground Anschluss)



Da die Pins des Mikrocontrollers jeweils nur maximal 25mA speisen können, wird mit dem Widerstand R3 der Strom begrenzt.

Der Mosfet sollte dabei mit einer angemessenen Gate-Source-Spannung beschalten werden, um einen möglichst schnellen Übergang in den Sättigungszustand zu ermöglichen, da in diesem Zustand der Drain-Source-Widerstand gegen null geht und sich der Halbleiter wie ein Kurzschluss verhält.

Die negative Gate-Source-Spannung wird schaltungstechnisch realisiert, indem man den PMOS um 180° dreht und den Source-Pin mit der positiven Versorgungsspannung verbindet. Durch den Spannungsteiler R1 und R2 ergibt sich ein  
Gate-Potential von 5V und es resultiert:

Eine Beschaltung mit einer Gate-Source-Spannung von |10V| ist bei Mosfets üblich.

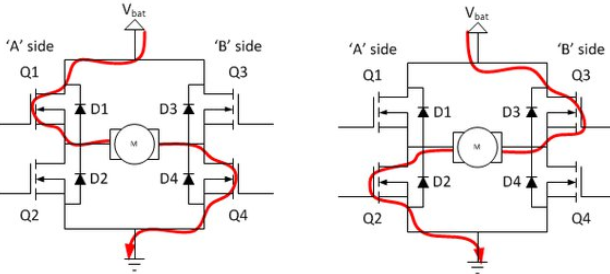
Es wurden Mosfets als Leistungsschalter gewählt, da diese durch ihren geringen Einschaltwiderstand und die schnellen Schaltzeiten ideal für Switching-Applikationen sind. Außerdem wird durch ihre extrem hohe Eingangsimpedanz der Hauptstromweg durch den Motor ermöglicht. [15]

Es wurde sich explizit für eine Highside Konfiguration entschieden, weil dadurch im ausgeschalteten Zustand wirklich die Spannungsversorgung zu der Last getrennt ist. Das bringt den Vorteil, dass sich die Windungen im Motor nicht zusätzlich erwärmen.

Der Nachteil einer Lowside Konfiguration, sprich der Leistungsschalter befindet sich zwischen Last und Masse ist nämlich, dass nur der Ground geschalten wird. [16]

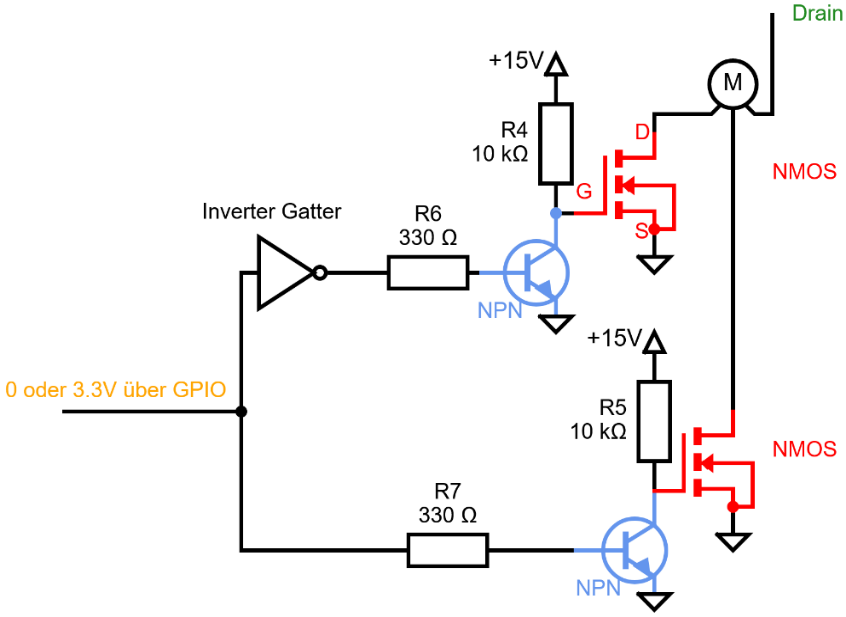
##### Selektorschaltung

Typischerweise steuert man permanent erregte Gleichstrommotoren mit einer   
H-Brücke an. Diese Schaltung besteht aus einer Anordnung von 4 Transistoren wie in **Abbildung 7** zu sehen ist. Dabei werden Q1 und Q4, sowie Q2 und Q3 immer zeitgleich geschalten. Dadurch kann der Strom von beiden Seiten durch den Motor fließen und eine Umpolung der Versorgungsspannung, um einen Richtungswechsel zu realisieren wird redundant.



**Abbildung 7**: H-Brücke [17]

Da die ausgewählten Lüfter aber einen positiven Anschluss und zwei Masseanschlüsse besitzen ist die Umsetzung mit der H-Brücke nicht möglich. Deswegen wird eine andere Schaltung/Selektorschaltung für den Wechsel der Drehrichtung benötigt.



**Abbildung 8**: Selektorschaltung

Der STM32-Mikrocontroller gibt über einen General Purpose Input Ouput Pin (GPIO) entweder 0V (logisch Low „0“) oder 3,3V (logisch High „1“) aus.

Sendet der Mikrocontroller 0V dann schaltet der NPN-Bipolartransistor nicht durch und das Gate des dahinterliegenden NMOS befindet sich auf einem positiven Spannungslevel. Nachdem der Sourcepin auf dem Massepotential hängt, befindet sich der Mosfet im leitenden Zustand (> ). Es wird so ein Stromfluss vom PMOS Drain durch den Motor und den NMOS bishin zur Masse ermöglicht.

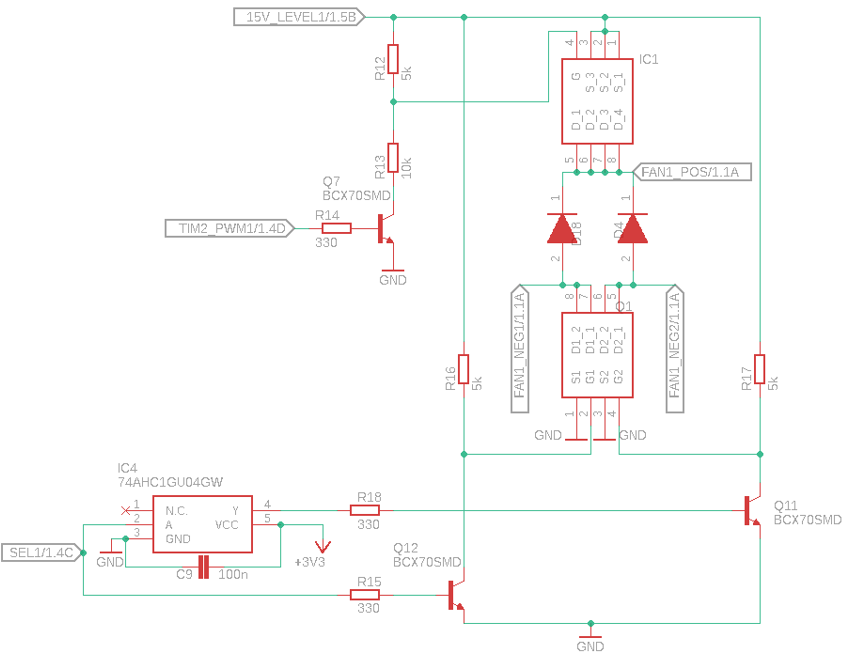
Sendet der Mikrocontroller 3,3V dann zieht der NPN-Transistor das Gate des NMOS auf Masse. Dadurch befinden sich und auf dem selben Potential.

Die Treshold-Spannung wird nicht erreicht und der N-Kanal Transistor sperrt   
(< ). Ein Stromfluss durch den Motor ist auf diesem Pfad nicht mehr möglich.

Als hardwareseitige Sicherheitslogik wurde ein Inverter Gatter hinzugefügt, damit sich immer nur einer der beiden Mosfets im leitenden Zustand befinden kann und so der Lüfter geschützt wird.

In **Abbildung 9** sieht man nochmals die zusammengesetze Schaltung. Die beiden N-Kanal Transistoren sind dabei auf einem IC integriert.

Der Schaltkreis wurde um zwei Flyback-Dioden ergänzt. Diese sind grundsätzlich immer nötig bei Switching-Applikationen in Kombination mit einer induktiven Last wie einem Motor.



**Abbildung 9**: zusammengesetzte Schaltung (EAGLE Schematic)

Nach dem Öffnen des Leistungsschalters wird der Stromfluss abrupt unterbrochen und das Magentfeld in der Spule baut sich ab. Dabei wird nach der Lenzschen Regel eine Spannung mit umgekehrter Polarität induziert, um den Stromfluss aufrecht zu erhalten.

Die Flyback-Spannung hängt dabei von der zeitlichen Stromänderung ab:

**Formel 1**: Spannung über die Spule

Da bei einer Spule der Stroms träge ist und nur mit der Konstante abbnimmt, verhält sie sich nach dem Öffnen des Schalters zeitweise wie eine Stromquelle. Aus Sicht der Indukitivtät wird der geöffnete Schalter als ein sehr großer Widerstand interpretiert. Aufgrund des hohen Drain-Source-Widerstandes des nicht leitenden Mosfets, hat bereits der kleinste Strom eine enorme Spannungsspitze über dem Transistor zur Folge. Um den Mosfet davor zu schützen werden zwei Schottky-Dioden parallel zum Motor und in Sperrrichtung zur Versorgungsspanung plaziert. Dadurch wird ein widerstandsarmer Weg für den Strom geschaffen, der dann solange durch die induktive Last geführt wird bis sich die Spule entladen hat. [18]

##### Mikrocontrollerschaltung

Eine genaue Beschreibung der MCU erfolgt im Kapitel *Mikrocontroller*.

Zu den Logic ICs und dem Mikrocontroller wurden jeweils Glättungskondensatoren hinzugefügt. Sie sorgen dafür, dass trotz elektrischem Rauschen die Spannung auf der Leiterplatte stabil bleibt. Bei Spannungseinbrüchen oder einem plötzlichen Spannungsbedarf kann der IC sie aus dem Glättungskodensator speisen. Und bei Spannungsanstiegen können die Decoupling Kodensatoren die überschüssige Energie aufnehmen. Je näher die Kondensatoren am IC plaziert sind desto besser ist der Effekt. Sie sind enorm wichtig, da Mikrocontroller und ICs empfindliche Bauteile sind und bei Spannungsschwankungen nicht korrekt funktionieren. [19]

Die Pins OSC\_IN und OSC\_OUT des Mikrocontrollers wurden auf die Leiterplatte rausgeführt. Infolgedessen lässt sich die MCU auch mit einem externen Oszillator   
takten (genaueres siehe *Takteinstellungen*).

Außerdem wurde der Boot0 Pin mit einem Pullup und Pulldown Widerstand rausgeführt. Dadurch ist es möglich den eingebettet Bootloader des Mikrocontrollers zu   
nutzen (genaueres siehe *Programmierschnittstelle*).

Für den Mikrocontroller wurde zusätzlich noch ein Reset-Button vorgesehen.

#### Layout

##### Bauteilauswahl

Als Designentscheidung wurde der Ultra Low-Power Mikrocontroller STM32L041C6T7 aus der L0-Serie des Hersteller STMicroelectronics im LQFP48 Package ausgewählt. Betrieben wird er von einem ARM Cortex M0+ Prozessor mit einer maximalen Taktrate von 32MHz. Das Auswahlkriterium war hierbei die Mindestanzahl von 8 unabhängigen PWM-Kanälen.

Um die selbst entworfene Hardware an den KNX-Bus anzukoppeln, wurde die Bus Coupling Unit (BCU) „5wg1 117-2ab12“ von Siemens aus ihrem Unterputzgehäuse entfernt und auf eine Vorrichtung auf der Leiterplatte gesteckt. Auf der BCU befindet sich ein TP-UART2 Chip, der das Twisted Pair UART in das standardmäßige   
UART-Protokoll umwandelt und dadurch dem Mikrocontroller ermöglicht über den Bus zu kommunizieren. Zwischen dem STM32 und dem Chip ist eine serielle Kommunikation mit einer Baudrate von bis zu 19200 Bit/s möglich. [20]



**Abbildung 10**: Bus Coupling Unit ohne Gehäuse [21]

Um die zur Verfügung gestellte Spannung von 24V nutzen zu können, wir diese von drei DC/DC Wandlern auf zwei verschiedene Spannungslevel heruntergestuft.

* 3,3V Betriebsspannung TSR 1-2433
* 15V Treiberspannung TSR 2-24150

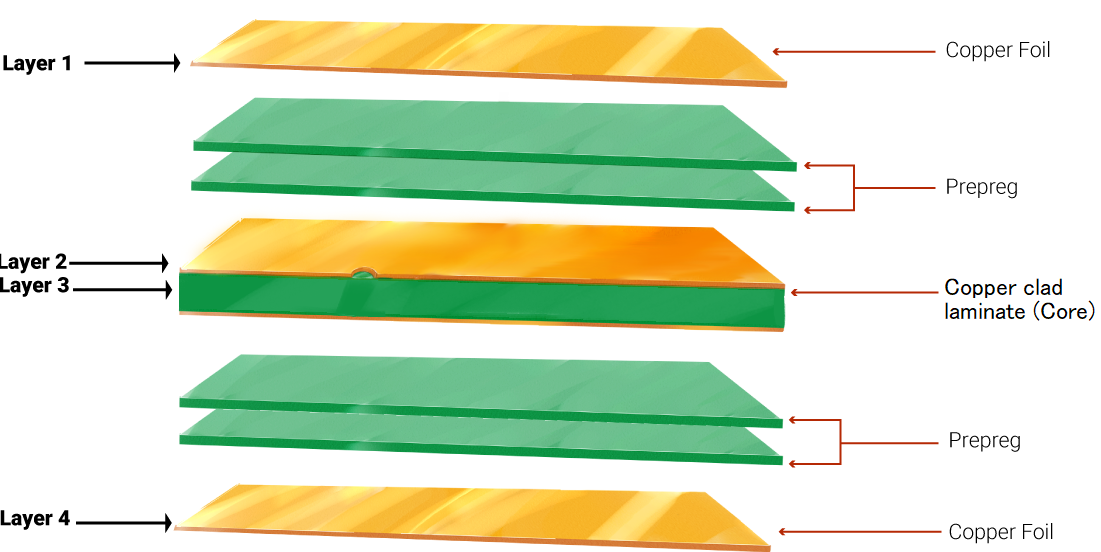
Es werden jeweils 4 Lüfter von einem separaten DC/DC Wandler versorgt. Dadurch sollen die deutlich erhöhten Anlaufströme der Motoren ermöglicht werden.

Die Anschlüsse der Spannungsversorgung und Lüfter wurden durch Through Hole Klemmleisten von Wago realisiert. Mit ihnen lässt sich ein breites Spektrum von   
Litzen und Draht in einem Durchmesserbereich von 0,08 bis 2,5 mm² klemmen.

Für die passiven Bauelemente wurde ein 0603-Package und für die Mosfets ein etwas größeres SO-8 Package gewählt, um die entstehende Wärme besser an die Platine abzugeben.

##### Design

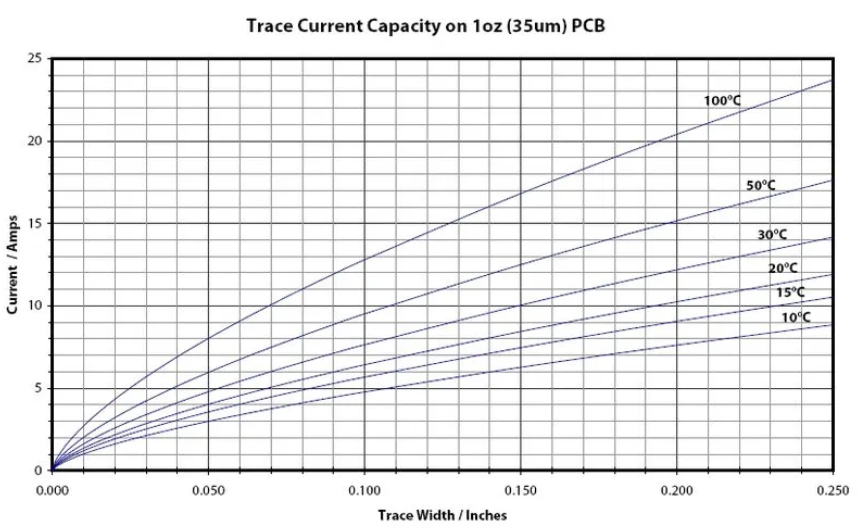
Die Leiterplatte besteht aus einem 4-lagigem PCB-Stackup. Auf den äußeren Ebenen befindet sich jeweils die Signale. Lage 2 wird als 3,3V Spannungslevel und Lage 3 als Massefläche verwendet. [22][23]



**Abbildung 11**: 4-lagiger PCB-Stackup [23]

Je mehr Strom durch eine Leiterbahn fließt, desto höher ist auch die dadurch entstehende Hitze. Um diesem Effekt entgegenzuwirken wird der Durchmesser der stromführenden Leiterbahnen erhöht.

Es wurde sich dabei an die Designrichtlinie IPC-2152 gehalten. Sie stellt den Zusammenhang zwischen Temperatur, Stromstärke und Leiterplattenbreite dar.

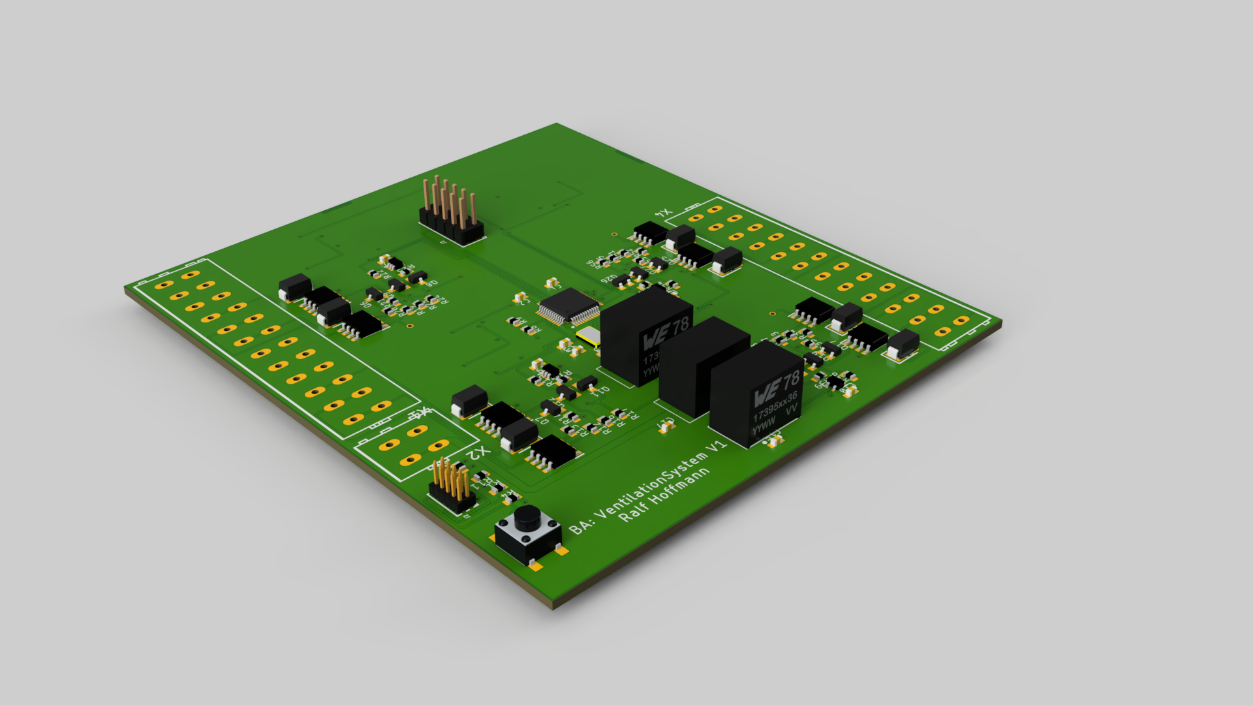


**Abbildung 12**: Designrichtlinie IPC-2152 [24]

Die stromführenden Leitungen im Design wurden mit einer Breite von 0,05 Inch (1,27mm) gewählt. Aus dem Graphen entnimmt man, dass eine Stromstärke von   
3 Ampere somit einen Temperaturanstieg von maximal 10° Celsius zur Folge hat. Angenommen wird hierbei, dass die Leiterplatte eine ungefähre Kupferdicke von 28 Gramm pro Quadratmeter aufweist, was auch dem Fertigungsstandard je Lage entspricht. [24]

Das Bestücken, der von Eurocircuits gefertigeten Leiterplatte, erfolgte im Labor für Aufbau- und Verbindungstechnik der Hochschule München per Hand und ohne Schablone.

Die Bauteile befinden sich sowohl auf der Ober- und Unterseite. Die Wago-Klemmen fehlen im Rendering, da deren CAD-Modelle für den Endverbraucher nicht freigegeben sind.



**Abbildung 13**: 3D-Rendering der entwickelten Hardware

# Software

## SW-Toolchain

**STM32CubeMX**

Mit dem graphischen Konfigurationstool STM32CubeMX lässt sich die gewünschte Pinbelegung sowie die Peripherie- und Takteinstellungen des Controllers vornehmen. Diese Basiseinstellungen werden dann auf Knopfdruck als C-Code generiert und in das Projekt geladen (Siehe **Abbildung 16** & **Abbildung 20**).

**STM32CubeIDE**

Als Entwicklungsumgebung wurde die kostenfreie STM32CubeIDE verwendet. Sie ist eine C und C++ Entwicklungsplattform auf Basis der Eclipse IDE mit   
GNU Compiler.

**Git**

Zur Versionsverwaltung wurde die OpenSource-Software Git genutzt. Mit ihr können die einzelnen Entwicklungsschritte dezentral gesichert und nachvollzogen werden. Außerdem ist es so möglich das Projekt auf andere Computer zu übertragen und zu bearbeiten.

**ETS**

Die Engineering Tool Software wird für die Verwaltung und Parametrisierung der KNX-Komponenten benötigt.

## Mikrocontroller

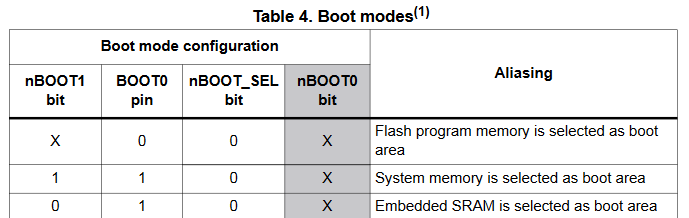
### Programmierschnittstelle

Der Mikrocontroller wird über die ARM spezifische Serial Wire Debug (SWD) Schnittstelle geflasht. Dabei werden nur die beiden Pins SWCLK und SWDIO benötigt. Mit jeder positiv registrierten Flanke auf dem SWCLK-Pin wird synchron ein Bit auf dem SWDIO-Pin empfangen oder gesendet. Es ist eine Weiterentwicklung der JTAG Programmierschnittstelle bei der 5 Pins notwendig sind. Da statt den JTAG Pins TCK und TMS, die beiden SWD Pins SWCLK und SWDIO verwendet werden, kann bei beiden Schnittstellen derselbe Stecker genutzt werden. Um den Mikrocontroller über die Schnittstellen zu programmieren, muss noch ein ST-Link Programmer zwischen MCU und PC geschalten werden. [25]



**Abbildung 14**: ST-Link/V2 Programmer [26]

Alternativ kann man den Controller auch direkt ohne zusätzliche Hardware über eine der seriellen Kommunikationsschnittstellen wie UART, CAN, I2C, etc. programmieren. Dafür wird der eingebettete Bootloader, der sich im Systemspeicher des Mikrocontrollers befindet, verwendet. Möchte man den Controller programmieren muss der Boot 0 Pin auf High gezogen werden, um zu signalisieren, dass ein neues Programm in den Flash-Speicher geschrieben werden soll. Danach muss der Boot 0 Pin auf Low gezogen sowie ein Zurücksetzen des Controllers durchgeführt werden. Nun wird der in den Flashspeicher geschriebene Code ausgeführt. [27, S.56-57]



**Tabelle 1**: Bootmodi des Mikrocontrollers [27, S.56]

Um das Programmieren der MCU zu erleichtern, wird die Variante mit dem ST-Link Programmer genutzt. Dies ist von Vorteil da so bei einer Codeänderung kein Widerstand um-gelötet werden muss.

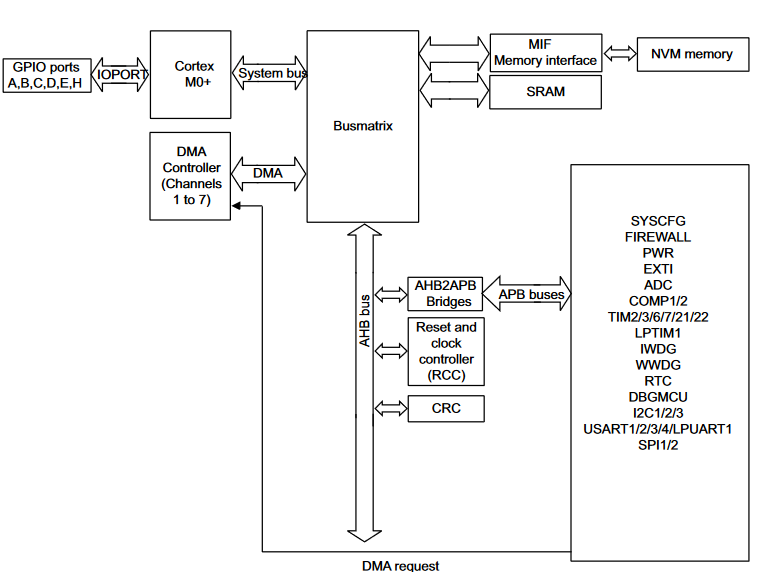
### Systemarchitektur

Der ARM Cortex M0+ Prozessor ist ein leistungsstarker und kostengünstiger 32Bit Prozessor basierend auf der Armv6-M Architektur. Er ist zudem der energieeffizienteste Prozessor von ARM.

Die Busschnittstelle des STMs beruht auf der Von-Neumann-Architektur, die sich dadurch auszeichnet, das Programm- und Datenspeicher über die gleichen Busse angesteuert werden. Gegenstück dazu ist die Harvard-Architektur bei der, der Code und die Daten auf unterschiedlichen Bussen transportiert werden. [28, S.19]

Das Herzstück des Systems ist die mehrschichtige 32Bit AHB-Bus Matrix die, die folgenden Hardwarekomponenten miteinander verbindet. Es gibt hierbei

* 2 Master
  + Den Cortex M0+ Kern
  + Und den Direct Memory Access Controller (DMA)
* Sowie 3 Slaves
  + Den internen nicht flüchtigen Speicher (Non-Volatile Memory NVM)
  + Den statischen RAM (Static Random-Access Memory SRAM)
  + Sowie die AHB (High Speed) to APB (Peripheral) Bridge



**Abbildung 15**: Systemarchitektur [27, S.49]

Der System-Bus bindet die CPU an die Busmatrix an. Sie kann so über den APB und AHB auf die Peripheriekomponenten zugreifen. Außerdem wird der System-Bus verwendet um Daten sowie allgemeine Anweisungen zu empfangen.

Der DMA-Bus bindet den DMA-Controller an die Busmatrix an. Dieser regelt den Zugriff auf die verschiedenen Speicherelemente.

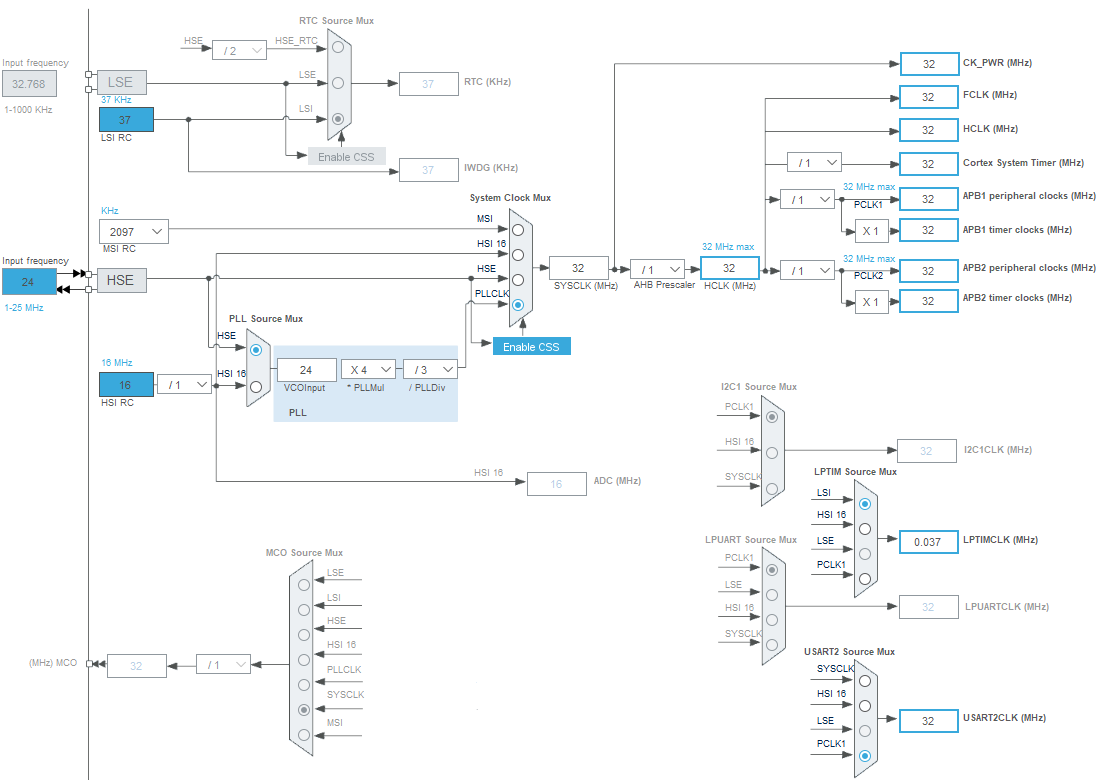
Die AHB/APB Brücke ermöglicht eine voll synchrone 32 MHz Kommunikation zwischen dem High Speed Bus und dem Peripherie-Bus.

Die Busmatrix entscheidet dabei über ein Round Robin Prinzip welcher Master Zugriff erhält. [27, S.49-50]

Der auf dem STM32L041C6T7 integrierte Nested Vector Interrupt Controller (NVIC) ermöglicht bis zu 32 verschiedene Interrupts mit 4 verschiedenen Prioritätsstufen.  
[28, S.20]

### Takteinstellungen

In **Abbildung 16** sieht man die Takteinstellungen direkt aus STM32CubeMX. Der Clock der einzelnen Peripherie Komponenten hängt davon ab wie der APB-Bus getaktet wird, auf dem sich diese befinden.



**Abbildung 16**: Clock-Tree aus STM32CubeMX

Es gibt dabei verschiedene Methoden den Takt zu erzeugen.

**Primäre Taktquellen** mit denen sich der Systemtakt bestimmen lässt:

* HSI 16 (High Speed Internal)

Der 16 MHz Takt wird intern vom einem RC-Oszillator erzeugt. [27, S.174]

* HSE (High Speed External)

Über den OSC\_IN Pin des Mikrocontroller lässt sich der Takt auch von einer externen Quelle mit bis zu 32 MHz vorgeben.

Durch die Hinzunahme des OSC\_OUT Pins kann man den Mikrocontroller auch über einen Quarz-Oszillator oder Keramik-Resonator mit bis zu 24MHz takten. [27, S.172-173]

* MSI (Multi Speed Internal Clock)

Hierbei wird der Takt von einem RC-Oszillator generiert, der sich in 7 Stufen zwischen 65,536 kHz und 4,194 MHz einstellen lässt. Er wird bei einem Reset des Controllers anfangs als Systemtakt verwendet. Außerdem kann er auch als Backup Takt konfiguriert werden, falls der externe HSE-Clock ausfällt.  
[27, S.174-175][55]

* PLL

Die Phase Locked Loop ist eine Phasenreglerschleife, bei der die lokale Frequenz abgegriffen wird und durch einen Phasenkomparator mit der Eingangsfrequenz verglichen wird. Ein spannungsgesteuerter Oszillator passt dann je nach Regelabweichung die Frequenz entsprechend an. Durch Hinzufügen eines Frequenzteilers lässt sich die Ausgangsfrequenz auf ein Vielfaches der Eingangsfrequenz erhöhen. [27, S.175]

In diesem Projekt wird die Eingangsfrequenz über einen externen 24 MHz Quarz-Oszillator vorgegeben (HSE) und dann über die PPL-Methode auf 32MHz erhöht und als Systemclock verwendet.

Grund dafür ist, dass der Quarz-Oszillator, deutlich genauer ist als der interne RC-Oszillator. Außerdem bleibt er bei Temperaturänderungen weiterhin frequenzstabil.

Nur über die PPL ist es möglich den Mikrocontroller auf der maximalen Taktfrequenz von 32 MHz zu betreiben. (Siehe **Abbildung 16**)

**Sekundäre Taktquellen** können folgende Komponenten betreiben: Realtime Clock, die UARTs und den Low-Power-Timer:

* LSE (Low Speed External)

Der Clock wird über eine externe Taktquelle zur Verfügung gestellt.  
[27, S.176]

* LSI (Low Speed Internal)

Der internen RC-Oszillator erzeugt hierbei eine Taktfrequenz von 37 KHz.  
[27, S.176]

In diesem Projekt wird als Sekundärtaktung der Low-Power Timer verwendet. Da dieser nur sehr langsame Prozesse überwacht wird er vom internen LSI-Clock getaktet. (Siehe **Abbildung 16**)

Die beiden Hauptbausteine der Software sind:

* die **UART**-Kommunikation zwischen Mikrocontroller und KNX-Busankoppler
* sowie die Lüfteransteuerung über **PWM**-Timer

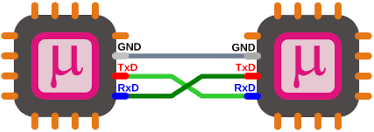
Deshalb wird Im Folgenden auf den Aufbau und die softwaretechnische Realisierung dieser Komponenten eingegangen.

## Uart

### Protokoll

Das Universal Asynchronous Receiver Transmitter Protokoll ist ein serielles Verfahren zur Datenübertragung zwischen 2 Geräten. Das bedeutet, dass die Daten nacheinander und Bit für Bit gesendet werden. Es werden hardwaretechnisch nur eine  
Transmit TX und Receive RX-Leitung benötigt.

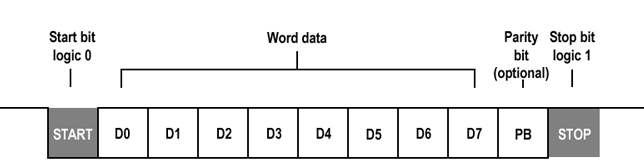
Die RX-Leitung von Gerät 1 wird dabei mit der TX-Leitung von Gerät 2 verbunden und umgekehrt. Zudem müssen die beiden Massepotentiale miteinander verbunden werden. [29][30]



**Abbildung 17**: praktischer Aufbau UART [32]

Der Vorteil der seriellen Übertragung ist der geringe Hardwareaufwand sowie der mögliche Datenaustausch über große Entfernungen. Sie ist im Vergleich zu der parallelen Übertragung unkompliziert und kosteneffizient. [31]

Ein UART Daten Paket besteht aus dem Startbit, den Nutzdaten/Payload, dem optionalen Paritätsbit und dem Stoppbit.



**Abbildung 18**: Aufbau UART-Protokoll [33]

Die TX-Leitung wird im Ausgangszustand auf High (logisch „1“) gehalten. Um eine Nachricht zu senden, wird die Leitung erstmal für einen Taktzyklus auf Low   
(logisch „0“) gezogen, was den Beginn der Message signalisiert. Dadurch fängt das Gerät auf der Empfängerleitung an das komplette Datenpaket mit der festgelegten Baudrate auszulesen. [29][30]

Die Baudrate ist dabei ein Maß für die Übertragungsgeschwindigkeit von Daten in der Einheit Symbol/Sekunde. Beim UART-Protokoll stimmt sie mit der Bitrate überein. [32]

Die Daten können dabei zwischen 5 und 8 Bit lang sein. Wird das optionale Paritätsbit weggelassen kann dementsprechend 1 Bit mehr gesendet werden. Standardmäßig wird das Bit mit dem niedrigsten Stellenwert (LSB) als erstes verschickt. Mit dem Paritätsbit lassen sich Einzelbitfehler überprüfen, das heißt, ob die Nachricht korrekt empfangen wurde [29][30]

Das Paritätsbit wird folgendermaßen erzeugt:

Die Quersumme aller Bits mit dem Wert „1“ wird genommen und das Paritätsbit so gesetzt, dass die Quersumme gerade/ungerade ist. [29]

* „0111 0000“ (Quersumme 3)
  + Bei gerader Parität muss das Paritätsbit „1“ sein
  + Bei ungerader Parität muss das Paritätsbit „0“ sein

In der Praxis wird das Paritätsbit aber meistens nicht genutzt.

Nach dem Senden des letzten Bits der Nachricht wird die Leitung mit dem Stoppbit wieder auf High gezogen, um das Ende der Nachricht zu signalisieren. Das Gerät befindet sich jetzt wieder im Ausgangszustand, bis die Leitung zu Beginn der nächsten Nachricht wieder auf Low gezogen wird.

### UART vs. USART

Der Unterschied zwischen UART und USART ist die Synchronität. Beim synchronen USART muss der Transmitter zusätzlich ein datenspezifisches Taktsignal generieren und dieses dem Empfänger schicken. Dadurch können ganze Datenblocks auf einmal gesendet werden, wodurch deutlich höhere Datenraten möglich sind. Es werden jedoch auch drei weitere Pins benötigt:

* Clock
* Ready to Send
* Clear to Send

Bei dem asynchronen UART hingegen wird der Takt lokal in beiden Geräten durch eine im Voraus festgelegte Baudrate generiert. [30]

Das UART nutzt den Standard TTL(Transistor-Transistor-Logic) Signalpegel. Eine logische „1“ entspricht 5V und eine logische „0“ entspricht 0V. Die Betriebsspannung der STM32-Familie beträgt zwar nur 3,3V, jedoch sind manche Pins wie die UART-Pins bis zu 5V spannungsresistent. Daher benötigt man keine zusätzliche Schaltung, zur Anpassung des Spannungslevels. [30]

### Implementierungsmöglichkeiten

Es gibt 3 verschiedene Methoden die UART-Kommunikation softwaretechnische zu implementieren.

#### Polling Methode

Hierbei wird so lange gewartet, bis die spezifizierte Menge an Daten erhalten wurde oder der gewählte Timeout überschritten wurde. Dieses Verfahren blockiert die Übertragung und alle weiteren CPU-Operationen in der Polling-Zeit, was Polling sehr ineffektiv macht. Es wird deshalb in der Praxis nur eingesetzt, wenn ausschließlich eine UART-Kommunikation benötigt wird. Ein weiterer Nachteil ist, dass Daten vermehrt verloren gehen wenn sie in Bursts ankommen. [34]

#### Interrupt/Callback

Die Datenübertragung findet nach dem erstmaligen Starten im Hintergrund statt. Dieses Verfahren ist nicht blockierend und die CPU kann in der Zeit andere Aufgaben erledigen. Sobald die Übertragung fertig ist, wird eine Callback Funktion aufgerufen, in der dann die Daten verarbeitet werden können. Die Callback Funktion ist die Interrupt Service Routine (ISR) des Interrupts. Im Falle von Daten-Bursts wird das Programm gegebenenfalls, durch das häufige Aufrufen der Callback-Funktion ausgebremst, jedoch gehen keine Daten verloren. Es lassen sich damit Baudraten von bis zu 115200 Bit/s realisieren. Dies ist deswegen auch die meistverbreitete Methode für   
Applikationen jeglicher Art. [34]

#### Direct Memory Access DMA

Bei DMA werden die Daten direkt von der Peripherie in den Speicher geschoben. Der DMA-Controller umgeht dabei die CPU, die deshalb auch nicht belastet wird. So können auch Baudraten von mehr als 1MBaud realisiert werden. Jedoch muss die   
DMA-Hardware die Datengröße im Voraus kennen. Da das KNX-Telegramm, aber je nach Datenlänge variiert, wurde die UART-Kommunikation über Interrupts implementiert. [34][35]

### Konfiguration

Die UART-Parameter des Mikrocontroller müssen auf die Parameter des   
KNX-Busankopplers abgestimmt sein, um eine einwandfreie Kommunikation zu gewährleisten. Die Baudrate wurde deshalb auf 19200 Bits pro Sekunde und die Datenlänge auf 8 Bit ohne Paritätsbit konfiguriert. Das Stoppbit ist 1 Bit lang. Zudem muss der NVIC Controller aktiviert werden, um nach jeder empfangen UART-Nachricht ein Interrupt auszulösen.

## Timer

Ein Hardware-Timer ist ein auf dem Mikrocontroller integriertes Zählermodul mit dem zeitspezifische Aktionen realisiert werden können. Es lassen sich zudem verschiedene Signalformen generieren. Auf dem gewählten STM32 Mikrocontroller befinden sich 3 Allzweck-Timer mit jeweils einem

* 16 Bit Autoreload-Register (ARR)
* 16 Bit Capture/Compare-Register (CCR)
* 16 Bit Prescaler (PSC)

Das Autoreload-Register stellt dabei die Zählperiode dar. Bei Erreichen des  
Autoreload-Werts wird der Timer, je nach Konfiguration entweder zurückgesetzt oder ändert die Zählrichtung.

Beim Erreichen des Werts im Capture/Compare Register wird der dazugehörige Pin gesetzt. Wurde der Betrieb im Interrupt-Modus konfiguriert, wird zusätzlich noch eine Flag im Interrupt Status Register gesetzt. So kann auch schon vor dem Erreichen des Autoreload-Werts ein Event erzeugt werden.

Der Prescaler teilt die Timerfrequenz herunter, um für die Anwendung den geeigneten Takt zur Verfügung zu stellen. [36] [27, S.371-373] [37, S.2-15; S.20]

### Pulsweitenmodulation PWM

Für die individuelle Ansteuerung der Lüfter wird jeweils ein eigener PMW-Channel benötigt. Es werden die folgende Allzweck-Timer verwendet:

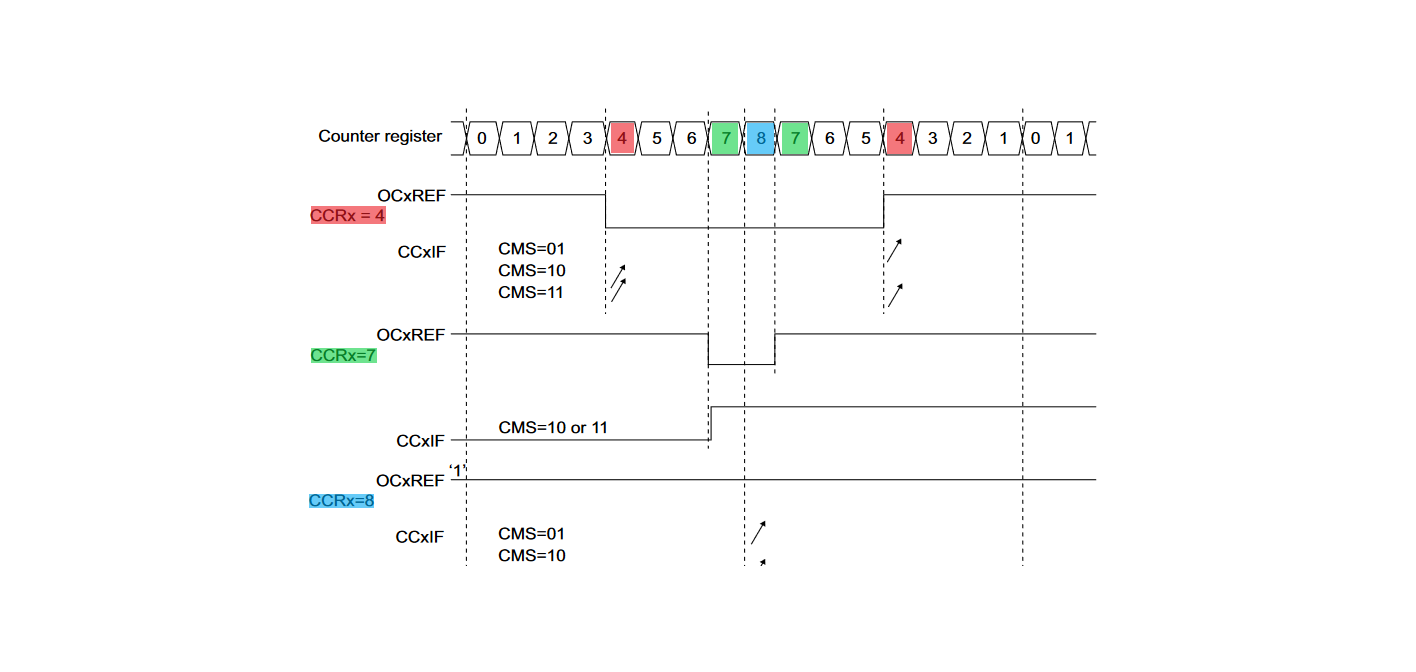
* TIM2 mit 4 Kanälen
* TIM21 mit 2 Kanälen
* TIM22 mit 2 Kanälen

Der PWM-Modus ist hierbei als center aligned gewählt worden. Das bedeutet, dass sich die Zählrichtung beim Erreichen des Autoreload-Werts umkehrt.

Das resultierende PWM-Signal wird dadurch symmetrisch über dem Überlauf des  
Timers generiert. [27, S.395-398][37, S.23-27][38]

Das Zeitdiagramm zeigt einen Timer mit einem Autoreload-Register von 8, der ein center aligned PWM-Signal erzeugt.

**Abbildung 19**: Generation eines center aligned PWM-Signals [27, S.398]



CCRx = 4

Man erkannt, dass sich das PWM-Referenzsignal OCxREF ändert sobald das Zählregister den Capture/Compare-Wert von 4 erreicht.

CCRx = 7

Das PWM-Referenzsignal OCxREF ändert wieder seinen Wert sobald das Zählregister den Capture/Compare-Wert von 7 erreicht. Zusätzlich wurde hier ein Interrupt konfiguriert, deshalb wird auch noch die Interrupt-Flag CCxIF gesetzt.

CCRx = 8

Sollte der Capture/Compare-Wert größer oder gleich dem Autoreload-Wert sein, dann wird das Referenzsignal dauerhaft auf „1“ gehalten. Dies entspricht einem Tastgrad von 100%.

Wird der Capture/Compare-Wert während einer noch laufenden Periode geändert, wird dieser erst beim nächsten negativen Überlauf des Timers übernommen, sprich in der nächsten Periode. [27, S.395-398][37, S.23-27][38]

Der Tastgrad ergibt sich aus dem Verhältnis des Capture/Compare-Werts und des Autoreload-Werts.

**Formel 2**: Dutycycle [37, S.56]

Der Autoreload-Wert errechnet sich zu

**Formel 3**: Autoreload-Register PWM [37, S.54]

Die PWM-Timer werden vom APB-Bus mit 32 MHz getaktet. Die PWM-Frequenz wurde nach ein paar Lautstärketest zu 21 kHz gewählt. Somit ergibt sich ohne Prescaler für das Autoreload-Register ein Wert von 1523.

Man hat also eine PWM-Auflösung von 32MHz/21kHz = 1524. Dies bedeutet, dass sich 1524 verschiedene Spannungslevel vom Mikrocontroller erzeugen lassen, was einer Auflösung von 10,57 Bit entspricht. [32, S.56]

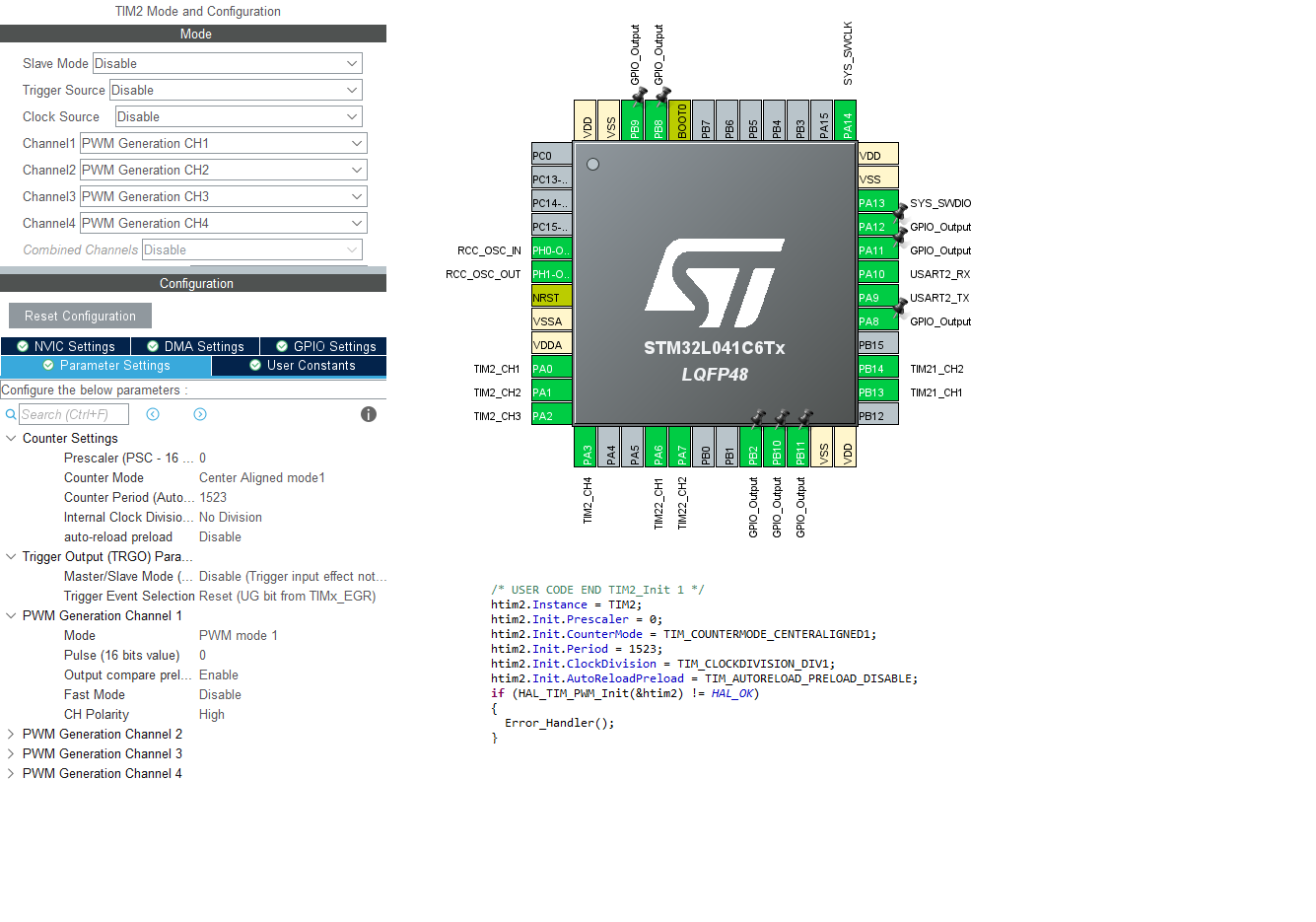
Die Schaltverluste des Mosfets sind dabei direkt proportional zu der Schaltfrequenz. [39, S.2]

**Formel 4**: Schaltverluste eines Mosfets [40, S.2-3]

Bei dem gewählten PMOS dauert ein kompletter Schaltzyklus mit Verzögerungszeiten maximal 340ns, wodurch auch noch deutlich höhere Schaltzeiten möglich wären.

Eine PWM-Frequenz von 21 kHz stellt hierbei einen guten Kompromiss aus der Lautstärke und Schaltverlusten dar.

In **Abbildung 20** sieht man die PWM-Timer-Konfiguration und die Pinbelegung des Mikrocontrollers in STM32CubeMX und einen Teil des daraus generierten Codes.



**Abbildung 20**: Timer-Konfiguration und Pinbelegung in STM32CubeMX

### LPTIM

Der noch unbelegte Low-Power-Timer LPTIM wird für das zeitliche Verhalten der Software genutzt. Es handelt sich hierbei um einen 16 Bit Timer mit begrenzter Funktionalität.

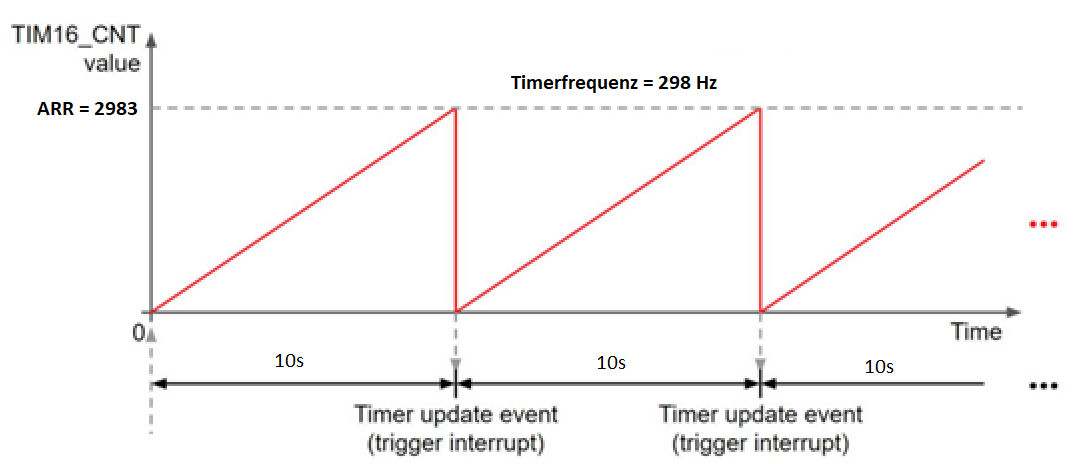
* 16 Bit Autoreload-Register (ARR)
* 16 Bit Capture/Compare-Register (CCR)
* 3 Bit Prescaler (PSC)
* Kann nur aufwärts zählen

Das Zählermodul lässt sich auch namensgebend nutzen, während sich der Mikrocontroller im Energiesparmodus befindet. [27, S.507-510; S.515] [41, S.2-8; S.14; S.17]

Der Timer wird mit dem Low Speed Internal Clock (LSI) von 37kHz getaktet. Der Prescaler wird mit dem maximalen Wert von 124 konfiguriert. Außerdem muss der NVIC Controller aktiviert werden, um jedes Mal beim Erreichen des ARR ein Interrupt auszulösen. Um das gewünschte Interrupt-Intervall von 10 Sekunden zu erzeugen, wird das Autoreload-Register folgendermaßen gesetzt:

Formel 5: Autoreload Register LPTIM [42]

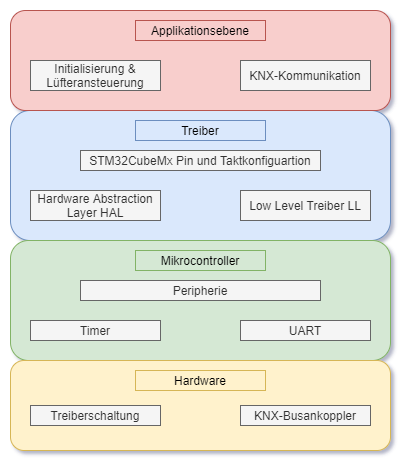
**Abbildung 21**: Low-Power-Timer Interrupt [43](modifiziert)



Es wurde sich für eine Realisierung über einen Timer Interrupt entschieden, da man so im Vergleich zu Delay-Funktionen den restlichen Code nicht blockiert.

## Softwarestruktur

Die Applikationsebene enthält die Hauptprogrammlogik der Software. Sie lässt sich in die Initialisierung & Lüftersteuerung sowie die KNX-Kommunikation aufteilen. Die Applikationen greifen dabei auf die Treiber zu.



**Abbildung 22**: Software-Stack

Mit dem graphischen Konfigurationstool STM32CubeMx und den High Level Treibern der Hardware Abstraction Layer (HAL) wird die Software von der Hardware entkoppelt und die Komplexität des Mikrocontrollers verborgen. Die HAL-Treiber stellen ein einfaches Interface für den Anwendungsentwickler zu Verfügung. Dadurch ist es möglich den Code ohne große Veränderungen auf einen anderen Controller der STM32 Familie zu portieren.

Komplementär zu den High Level HAL Treibern gibt es noch die Low Level Treiber (LL), deren Programmierschnittstellen auf Registerebene arbeiten und hauptsächlich aus Makros bestehen. Mit ihnen lassen sich eine bessere Programmlaufzeit und Speicherplatzausnutzung erzielen. Jedoch muss sich dafür ein umfangreiches Wissen über den gewählten Mikrocontroller angeeignet werden und eine nachträgliche Portierung des Codes ist aufwendig. [57]

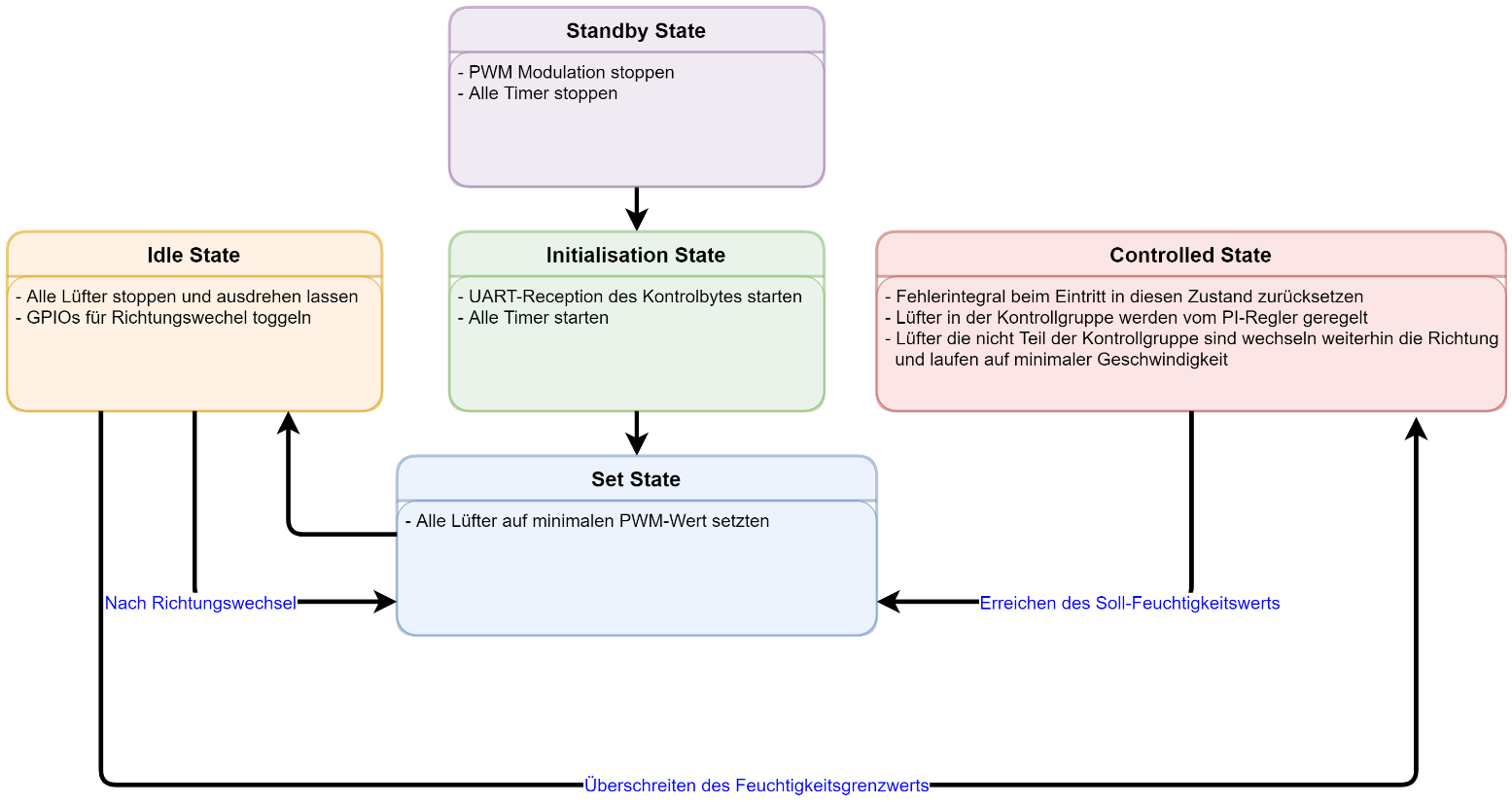
Die Peripherie-Elemente Timer und UART steuern dann die Treiberschaltung und den KNX-Busankoppler an.

Für dieses Projekt ist die Entscheidung auf die Nutzung der HAL-Treiber gefallen, da die Software vorerst auf einem Nucleo-Entwicklungsboard geschrieben wurde und dann nachträglich auf den STM32l041C6T7 portiert wurde.

## Programmablauf

### Initialisierung & Lüfteransteuerung

Die Initialisierung & Lüfteransteuerung entspricht dem übergeordneten Softwareablauf und wird in Form des Zustandsautomat statemachine\_process festgelegt. In welchem Betriebsmodus sich der Mikrocontroller gerade befindet hängt dabei von internen und externen Einflüssen ab. Der Zustandsautomat befindet sich in der gleichnamigen Source-Datei und wird dauerhaft in der Schleife der main.c aufgerufen.



**Abbildung 23**: Ablauf der Process-Statemachine

**Standby State:**

Nachdem Starten des Mikrocontrollers beginnt dieser erstmal im Standby State. Hier werden sicherheitshalber alle PWM Dutycycle auf null gesetzt (3.6.1.1.2).

Außerdem werden die Timer TIM2, TIM21 und TIM22 für die PWM-Generation gestoppt, ebenso der Low-Power-Timer LPTIM1 (3.6.1.1.1).

Danach wird bedingungslos in den Initialisation Zustand übergegangen.

Initialisation State:

Im Initialisation State werden die PWM-Timer aktiviert. Zudem wird der Low-Power-Timer im Interrupt Modus gestartet (3.6.1.1.1).

Des Weiteren wird das Empfangen des ersten Kontrollbytes mit der Funktion HAL\_UART\_Receive\_IT im Interrupt Modus gestartet (3.6.3.1).

Der Zustandsautomat wechselt dann ohne Bedingung in den Set State

Set State:

Im Set State werden alle PWM Dutycycle auf den kleinstmöglichen Wert gesetzt, um einen möglichst leisen Betrieb zu gewährleisten (3.6.1.1.3).

Idle State:

Im Idle State drehen die Lüfter mit einer konstanten Geschwindigkeit und wechseln mit dem vom Nutzer gewählten Zeitintervall kontinuierlich die Richtung. Von dem Low-Power-Timer wird in diesem Intervall immer eine Flag gesetzt, die den Richtungswechsel einleitet (hier: alle 40 Sekunden).

Es werden dann alle Lüfter angehalten (3.6.1.1.2).

Nach weiteren 20 Sekunden wird noch eine Flag gesetzt, die bestätigt, dass sich die Lüfter ausgedreht haben (3.6.3.9).

Daraufhin werden alle GPIO-Pins getoggelt (3.6.1.1.4).

Die Lüfter drehen nun in die entgegengesetzte Richtung.

Auf der Leiterplatte befindet sich eine hardwareseitige Sicherheitsschaltung, die dafür sorgt, dass der Lüfter nicht für beide Richtungen gleichzeitig angesteuert wird. Jedoch haben die Lüfter unter anderem durch Flyback-Dioden, die den Strom beim Öffnen eines Leistungsschalter weiter durch den Motor treiben, eine lange Auslaufzeit. Durch abrupte Richtungswechsel der Motoren während des Ausdrehens, wird deren Lebenserwartung durch die entstehenden Stromspitzen und resultierende Wärme stark verringert.

Es wurde außerdem sichergestellt, dass während die Lüfter noch ausdrehen kein Zustandswechsel möglich ist, umso das Softwaretiming nicht zu gefährden.

Um die Lüfter wieder anlaufen zu lassen wird nach erfolgreichem Richtungswechsel erneut der Set Zustand durchlaufen. Der Zustandsautomat wechselt also immer zwischen dem Set und Idle Zustand hin und her bis der Feuchtigkeitswert einen Grenzwert überschreitet, dann wird in den Controlled State übergegangen.

Controlled State

Im Controlled State wird die PI-Regler Schleife für alle Lüfter in der Kontrollgruppe aktiv, welche dann nur in eine Richtung laufen. Die Lüfter, die sich nicht in der Kontrollgruppe befinden, laufen weiterhin auf dem minimalen Wert mit kontinuierlichem Richtungswechsel. Wird der Feuchtigkeitssollwert erreicht oder unterschritten, werden die Lüfter über den Set Zustand wieder in ihren Ausgangszustand versetzt.

#### Funktionen

##### start\_all\_timers / stop\_all\_timers

void start\_all\_timers (void);

void stop\_all\_timers (void);

Die Funktion startet/stoppt die Allzweck-Timer zur PWM-Signalgeneration und den Low-Power-Timer im Interrupt Modus.

##### reset\_all\_pwm / reset\_pwm\_not\_controlgroup

void reset\_all\_pwm (void);

void reset\_pwm\_not\_controlgroup (void);

Die Funktion setzt den Tastgrad aller PWM-Signale auf den Wert 0. Beziehungsweise aller PWM-Signale, die sich nicht in der Kontrollgruppe befinden.

##### set\_all\_pwm / set\_pwm\_not\_controlgroup

void set\_all\_pwm (uint16\_t ccr\_value);

void set\_pwm\_not\_controlgroup (uint16\_t ccr\_value);

Parameter:

in ccr\_value neuer Capture/Compare-Registerwert

ret val -

Mit dieser Funktion lässt sich durch das Setzen des Capture/Compare-Werts der Tastgrad von allen PWM-Kanälen bestimmen bzw. von allen Kanälen, die nicht Teil der Kontrollgruppe sind.

##### toggle\_all\_gpios / toggle\_gpios\_not\_controlgroup

void toggle\_all\_gpios (void);

void toggle\_gpios\_not\_controlgroup (void);

Diese Funktion toggelt alle GPIO-Pins bzw. alle GPIO-Pins, die nicht in der Kontrollgruppe inkludiert sind. Das Toggeln eines GPIO-Pins resultiert in der Änderung der Lüfterlaufrichtung.

##### pi\_controller

uint16\_t pi\_controller (float process\_variable);

Parameter:

in process\_variable Der zu regelnde Wert

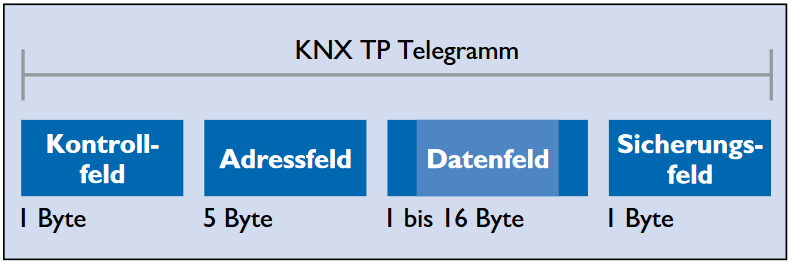
ret val - Rückgabe der neuen Stellgröße

Die Funktion enthält einen klassischen PI-Regler mit Anti-Windup und Stellgrößenbegrenzung (siehe **Abbildung 33**).

### KNX-Kommunikation

Die KNX-Kommunikation wird durch die UART-Statemachine realisiert, die im Folgenden erklärt wird. Jedoch muss zur Nachvollziehbarkeit davor der Aufbau des KNX-Telegramms geschildert werden.

#### Telegramm Aufbau



**Abbildung 24**: KNX Telegramm Aufbau [43, S.8]

Wie man in **Abbildung 24** sieht, variiert ein KNX-Telegramm je nach Payloadlänge zwischen 8 und 23 Byte.

Das Kontrollbyte wird in der ETS konfiguriert und ist in diesem Projekt stets  
„1011 1100“ was dezimal 188 entspricht.



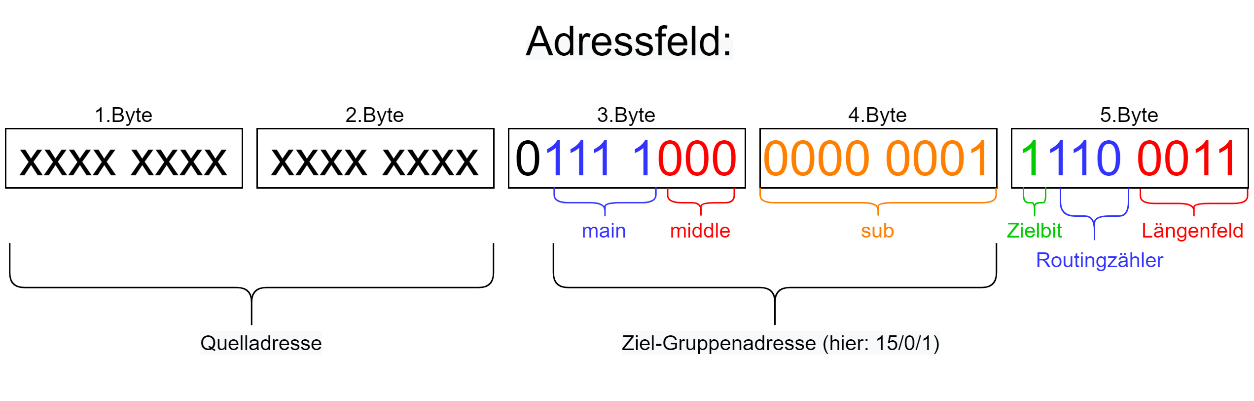
**Abbildung 25**: Kontrollfeld

Das Wiederholungsbit ist auf „1“ gesetzt was keiner Wiederholung entspricht. Im Falle eines inkorrekt empfangenen Telegramms wird dieses erneut an die Aktoren gesendet. Damit die Aktoren, die das Telegramm bereits korrekt empfangen haben, nicht nochmal auf die Nachricht reagieren wird das Wiederholungsbit auf „0“ gesetzt. [44][45]

Mit den Prioritätsbits lässt sich der Rang des Telegramms folgendermaßen festlegen:

* „00“ höchste Priorität
* „01“ Alarmfunktion
* „10“ Handfunktion
* „11“ Automatik

Die ersten zwei Bytes des Adressfelds bilden die physikalische Adresse des Senders. Da in dieser Anwendung nur auf dem Bus gelesen wird sind sie irrelevant. [44]



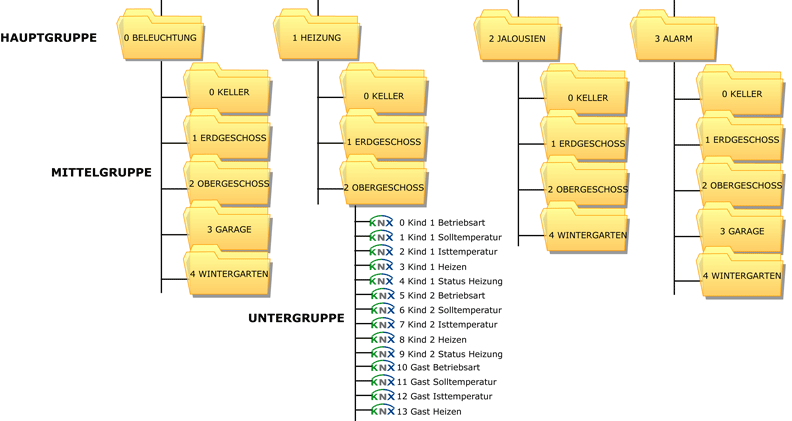
**Abbildung 26**: Adressfeld

Das Zielbit gibt den Adressentyp des Empfängers an

* „0“ physikalische Adresse
* „1“ Gruppenadresse

In der ETS wurde der Empfänger als Gruppenadresse mit einer 3 Levelstruktur konfiguriert. Das heißt sie setzt sich aus der main, middle und sub Adresse in der Form 15/0/1 zusammen.

Der einmalig vergebenen physikalischen Adressen wird dabei zusätzlich noch eine Gruppenadresse zugeordnet. Dadurch lassen sich KNX-Systeme erst richtig strukturieren. Man kann so zum Beispiel alle Heizungselemente oder ganze Stockwerke ansprechen ohne jedem Aktor ein individuelles Telegramm zu senden.[45]



**Abbildung 27**: Organisation eines KNX-Systems mit Gruppenadressen [45]

Der Routingzähler von 6 gibt an, wie oft ein Telegramm einen Koppler passieren darf. Dadurch wird verhindert, dass bei einem Verdrahtungsfehler ein Telegramm ewig lang im Bussystem kreist.

Das Längenfeld enthält die Länge der darauffolgenden Payload. Um die Anzahl der Bytes zu erhalten, muss der Wert im Längenfeld noch um eins addiert werden.

Das Sicherungsfeld enthält noch das Paritätsbit, mit dem der Empfänger überprüfen kann, ob die Nachricht korrekt empfangen wurde. [44]

Unmittelbar nach Erhalt des Telegramms muss der Aktor, noch ein Quittierungs-Byte zurücksenden. Wenn die Adresse des Telegramms mit der des Aktors übereinstimmt, dann:

* ACK „1100 1100“

sonst:

* NACK „0000 1100“

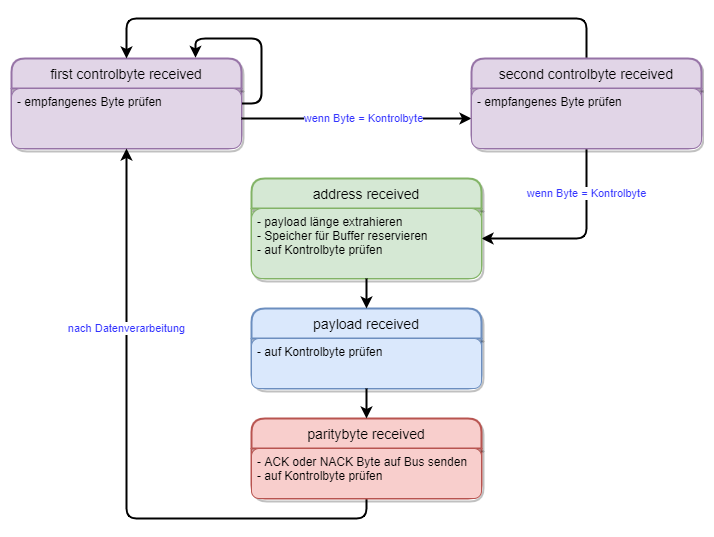
#### UART-Statemachine

Die Kommunikation im Haus wurde erstmals mit einem Arduino und einer C++ Test-Bibliothek erprobt und mitgeloggt.

Dabei kam heraus, dass das Kontrollbyte, anders als in der Theorie, immer doppelt statt einfach gesendet wird.

Der Zustandsautomat befindet sich in der statemachine\_uart.c Source-Datei und bedient sich aus Funktionen der selbstgeschrieben knx\_receive\_telegram.c Bibliothek, die auf der Arduino-Bibliothek basiert. [46]

Die UART-Reception wird einmalig im Initialisation State der Process-Statemachine gestartet. Danach wird die UART-Statemachine rekursiv über die HAL\_UARTRxCpltCallback Funktion aufgerufen (3.6.3.2).



**Abbildung 28**: Ablauf der UART-Statemachine

Um einen Zustandswechsel zu realisieren, muss am Ende eines States immer eine neue UART-Reception gestartet werden, da man sonst nicht mehr in die Callback Funktion gelangt. Nachdem die beiden korrekten Kontrollbytes empfangen wurden, wird die  
5 Byte Adresse direkt in einem Schub empfangen und die Länge der darauffolgenden  
Payload extrahiert.

In C ist ein Array ein statisches Datenkonstrukt, dessen Größe bereits beim Kompilierzeitpunkt bekannt sein muss. Deshalb muss der variierender Payloadbuffer dynamisch auf dem HEAP mit der malloc-Funktion angelegt werden.

Nach dem Erhalten der Payload und des Paritätsbyte muss noch überprüft werden, ob das empfangene Telegramm überhaupt interessant für den Aktor ist. Dafür müssen die zugeordneten Gruppenadressen des Aktors mit denen im Telegramm verglichen werden (3.6.2.3.1 & 3.6.2.3.2).

Bei Interesse wird das Quittierungsbyte auf ACK, andernfalls auf NACK gesetzt und mit HAL\_UART\_Transmit\_IT gesendet (3.6.3.3).

Erst nach der Verarbeitung der Daten in der extract\_data Funktion wird die   
UART-Reception erneut gestartet (3.6.2.3.4).

Dadurch wird verhindert, dass während der Auswertung die Buffer schon wieder beschrieben werden. Außerdem muss der reservierte Speicher auf dem Heap wieder freigegeben werden.

Es gibt zusätzlich noch einen Rücksprungmechanismus mit dem aus allen Zuständen zurückgesprungen werden kann. Dieser wurde zu Wahrung der Übersichtlichkeit nicht in **Abbildung 28** eingezeichnet. Da auf dem KNX-Bus auch Nachrichten mitten im Prozess abbrechen können, wird jeder Buffer immer mit check\_for\_controlbyte auf das Kontrollbyte geprüft, welches schließlich den Beginn eines neuen Telegramms signalisiert (3.6.2.3.3).

Sollte im geprüften Buffer ein Kontrollbyte sein, wird festgestellt an welcher Position es sich befindet und wie folgt zurückgesprungen:

* 1 Kontrollbyte an der letzten Stelle des Buffers
  + second controlbyte received Zustand
* 2 Kontrollbytes an den letzten beiden Stellen des Buffers
  + address received Zustand
* Kontrollbyte an sonstiger Stelle des Buffers
  + first controlbyte received Zustand

Durch diesen Mechanismus gehen weniger Telegramme verloren und es wird verhindert, dass dem Regler im Falle eines Telegrammabbruchs unnütze Werte übergeben werden.

#### Funktionen knx\_receive\_telegram Bibliothek

##### add\_listening\_group\_address

void add\_listening\_group\_address(char \*address);

Parameter:

in address Gruppenadresse in der Form xx/x/xxx

ret val - none

Fügt eine Gruppenadresse, auf die der Aktor reagieren soll zum globalen zweidimensionalen Array listen\_group\_addresses hinzu. Der übergebene Stringbuffer address wird dafür in Tokens zerteilt und in Integers umgewandelt.

##### check\_interest

bool check\_interest(uint8\_t \*address);

Parameter:

in address Pointer zum Buffer, der die 5Byte Adresse enthält

ret val - true bei Übereinstimmung

Sie überprüft, ob es sich um eine Gruppenadresse handelt. Außerdem wird die main, middle und subgroup aus dem übergebenen Buffer extrahiert und verglichen, ob sie mit den angelegten Adressen im Array listen\_group\_addresses übereinstimmen. Wenn dies zutrifft, wird der Rückgabewert auf true gesetzt.

##### check\_for\_controlbyte

bool check\_for\_controlbyte(uint8\_t \*buffer, uin8\_t size);

Parameter:

in buffer Pointer zum Buffer, der überprüft werden soll

in size Anzahl der Bytes im Buffer

ret val - true wenn Kontrolbyte enthalten ist

Die Funktion prüft, ob sich im übergegeben Buffer an einer beliebigen Stelle das Kontrolbyte befindet. Ist im Buffer ein Kontrolbyte wird der Rückgabewert auf true gesetzt.

##### extract\_data

void extract\_data (void);

Die Funktion befindet sich in der mainloop wird jedoch durch Abfragen von StatusFlags nur ausgeführt, wenn das Empfangen des kompletten Telegramms beendet wurde und der Aktor auch Interesse an der Nachricht hat. Nach der Datenextraktion durch die Funktion get\_2byte\_float\_value wird der reservierte HEAP-Speicher freigegeben und durch Setzen eines Flags die nächste UART-Reception gestartet.

##### get\_2byte\_float\_value

float get\_2byte\_float (uint8\_t \*payload\_buffer);

Parameter:

in payload\_buffer Pointer zum Buffer, der die Nutzdaten enthält

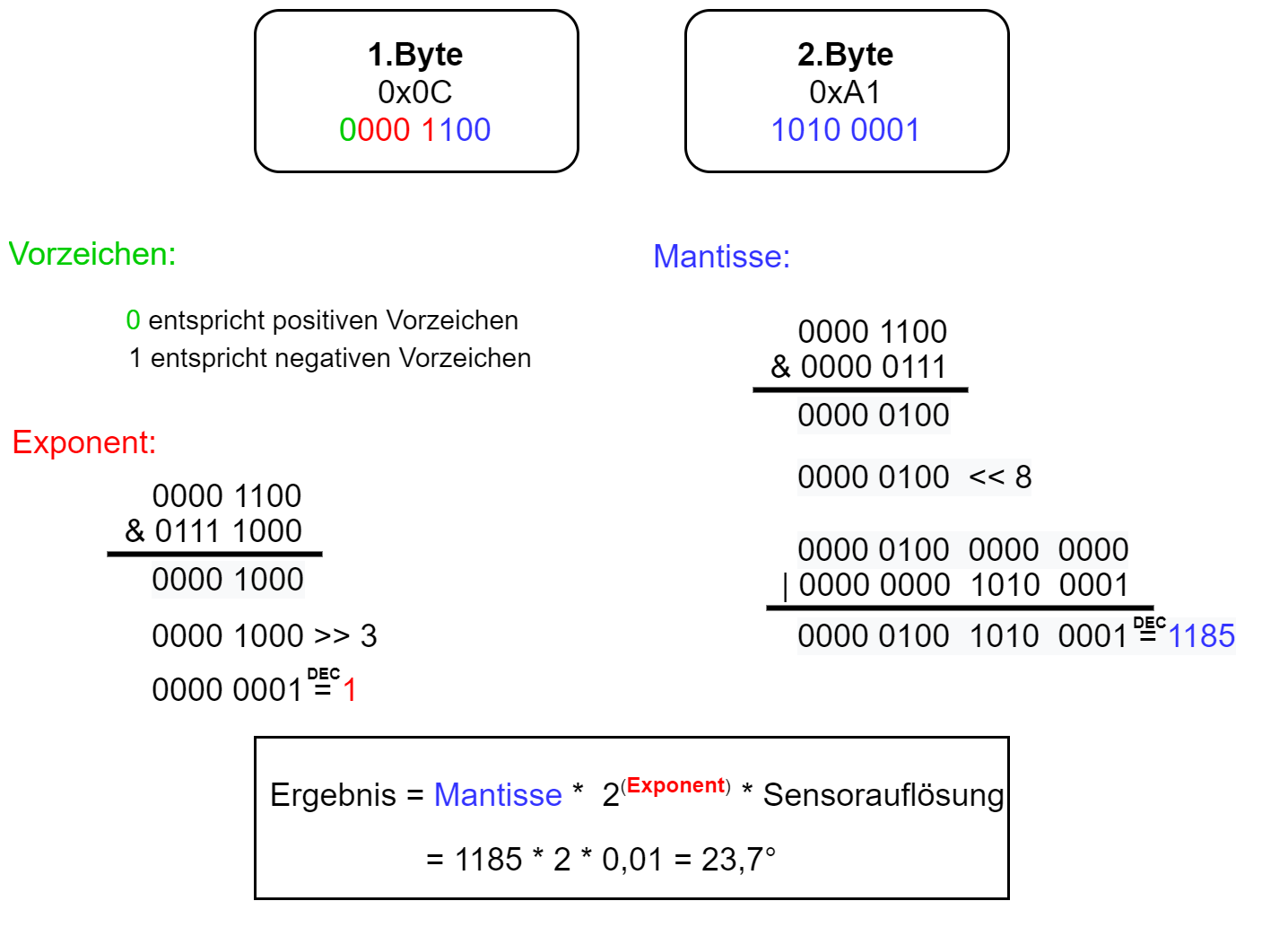
ret val - physikalischer Wert

Diese Funktion extrahiert die relevanten Daten aus dem Payloadbuffer und wandelt sie vom IEEE Format in physikalische Größen um.

Es handelt sich dabei um ein ungewöhnliches IEEE Format mit

* 1 Bit Vorzeichen
* 4 Bit Exponent
* 11 Bit Mantisse

Im Folgenden wird aus den Bytes 0x0C und 0xA1 der physikalisch Wert durch Verschiebungen und Bitmasken extrahiert (siehe *Logging*).



**Abbildung 29**: Datenextraktion eines 2Byte Floatwerts

### HAL-Treiber

Im Folgenden werden die genutzten HAL-Funktionen gezeigt und kurz erklärt.

#### HAL\_UART\_Receive\_IT

HAL\_StatusTypeDef HAL\_UART\_Receive\_IT(UART\_HandleTypeDef \* huart, uint8\_t \*pData, uint16\_t Size);

Parameter:

in huart Pointer zur UART handle Struktur aus HAL Treibern

in pData Pointer zum Datenbuffer, der beschreiben werden soll

in Size Anzahl der Bytes in pData

ret val - HAL Status

Mit dieser Funktion wird die UART-Reception im Interrupt Modus gestartet. Es wird die angegeben Anzahl an Bytes im Hintergrund und Non Blocking Modus empfangen.

#### HAL\_UARTRxCpltCallback

\_\_weak void HAL\_UART\_RxCpltCallback(UART\_HandleTypDef \*huart)

Parameter:

in huart Pointer zur UART handle Struktur aus HAL Treibern

ret val - none

Diese als schwach gekennzeichnete Funktion ist die UART Interrupt Service Routine, die nach dem vollständigen Erhalt einer UART Nachricht ausgeführt wird. Sie wurde in der main.c so überschrieben, dass sie die UART-Statemachine rekursiv aufruft.

#### HAL\_UART\_Transmit\_IT

HAL\_StatusTypeDef HAL\_UART\_Transmit\_IT(UART\_HandleTypeDef \*huart, uint8\_t \*pData, uint16\_t Size);

Parameter:

in huart Pointer zur UART handle Struktur aus HAL Treibern

in pData Pointer zum Datenbuffer, der gesendet werden soll

in Size Anzahl der Bytes in pData

ret val - HAL Status

Mit dieser Funktion wird die UART-Transmission im Interrupt Modus gestartet. Es wird die angegeben Anzahl an Bytes im Hintergrund und Non Blocking Modus verschickt.

#### HAL\_TIM\_PWM\_Start

HAL\_StatusTypeDef HAL\_TIM\_PWM\_Start(TIM\_HandleTypeDef \*htim, uint32\_t Channel);

Parameter:

in htim Pointer zur Timer handle Struktur aus HAL Treibern

in Channel Auswahl des jeweiligen Channels des Timers

ret val - HAL Status

Mit dieser Funktion wird die Generation des PWM-Signals auf dem gewählten Kanal gestartet.

#### HAL\_TIM\_PWM\_Stop

HAL\_StatusTypeDef HAL\_TIM\_PWM\_Start(TIM\_HandleTypeDef \*htim, uint32\_t Channel);

Parameter:

in htim Pointer zur Timer handle Struktur aus HAL Treibern

in Channel Auswahl des jeweiligen Channels des Timers

ret val - HAL Status

Mit dieser Funktion wird die Generation des PWM-Signals auf dem gewählten Kanal gestoppt.

#### HAL\_TIM\_SET\_COMPARE

#define \_\_HAL\_TIM\_SET\_COMPARE(TIM\_HandleTypeDef \*htim, uint32\_t Channel, uint32\_t Compare)

Parameter:

in htim Pointer zur Timer handle Struktur aus HAL Treibern

in Channel Auswahl des jeweiligen Channels des Timers

in Compare Neuer Capture Compare Registerwert

ret val - none

Mit diesem define lässt sich während der Laufzeit der Capture/Compare-Registerwert setzen und somit der DutyCycle einstellen.

#### HAL\_LPTIM\_Counter\_Start\_IT

HAL\_StatusTypeDef HAL\_LPTIM\_Counter\_Start\_IT(TIM\_HandleTypeDef \*hlptim, uint32\_t Period);

Parameter:

in hlptim Pointer zur LPTIM handle Struktur aus HAL Treibern

in Period Auswahl des Autoreload Registerwerts

ret val - HAL Status

Mit dieser Funktion wird der Low-Power-Timer im Aufwärtszähl- und Interrupt Modus gestartet.

#### HAL\_LPTIM\_Counter\_Stop\_IT

HAL\_StatusTypeDef HAL\_LPTIM\_Counter\_Stop\_IT(TIM\_HandleTypeDef \*hlptim);

Parameter:

in hlptim Pointer zur LPTIM handle Struktur aus HAL Treibern

ret val - HAL Status

Mit dieser Funktion wird der Low-Power-Timer im Interrupt Modus gestoppt.

#### HAL\_LPTIM\_AutoReloadMatchCallback

\_\_weak void HAL\_LPTIM\_AutoReloadMatchCallback(LPTIM\_HandleTypDef \*hlptim)

Parameter:

in hlptim Pointer zur LPTIM handle Struktur aus HAL Treibern

ret val - none

Diese als schwach gekennzeichnete Funktion ist die LPTIM Interrupt Service Routine, die ausgeführt wird, sobald der Timer den Wert im Autoreload-Register erreicht. Sie wurde in der main.c so überschrieben, dass alle 40 Sekunden die „flag\_switch\_direction\_demand“ gesetzt wird. Nach weiteren 20 Sekunden wird zusätzlich die „flag\_fans\_spun\_out“ gesetzt (also alle 60 Sekunden).

#### HAL\_GPIO\_TogglePin

void HAL\_GPIO\_TogglePin(GPIO\_TypeDef \*GPIOx, uint16\_t GPIO\_Pin);

Parameter:

in GPIOx GPIO Peripherie auf der sich der Pin befindet

in GPIO\_Pin GPIO Pin Nummer

ret val - none

Mit dieser Funktion wird der gewählte GPIO Pin getoggelt

## Logging

Auf der linken Seite sieht man einen Ausschnitt aus dem Arduino Logging, und rechts Screenshots von den ETS-Einstellungen. Es gibt dabei in der Praxis zwei Unterschiede zur Theorie. Zum einen wird das Kontrollbyte immer doppelt gesendet, zum anderen werden beim Senden eines 2Byte Floatwerts noch 2 Dummy-Werte   
mitgesendet.

Die ersten beiden Bytes der Payload sind die Dummy-Werte „0x00“ & „0x80“.

Die letzten beiden Bytes hingegen sind die wirklichen Nutzdaten, wie in Nachricht 1 „0x0C“ & „0xA1“.

Diese müssen dann noch aus dem IEEE-Format in physikalische Werte umgewandelt werden.

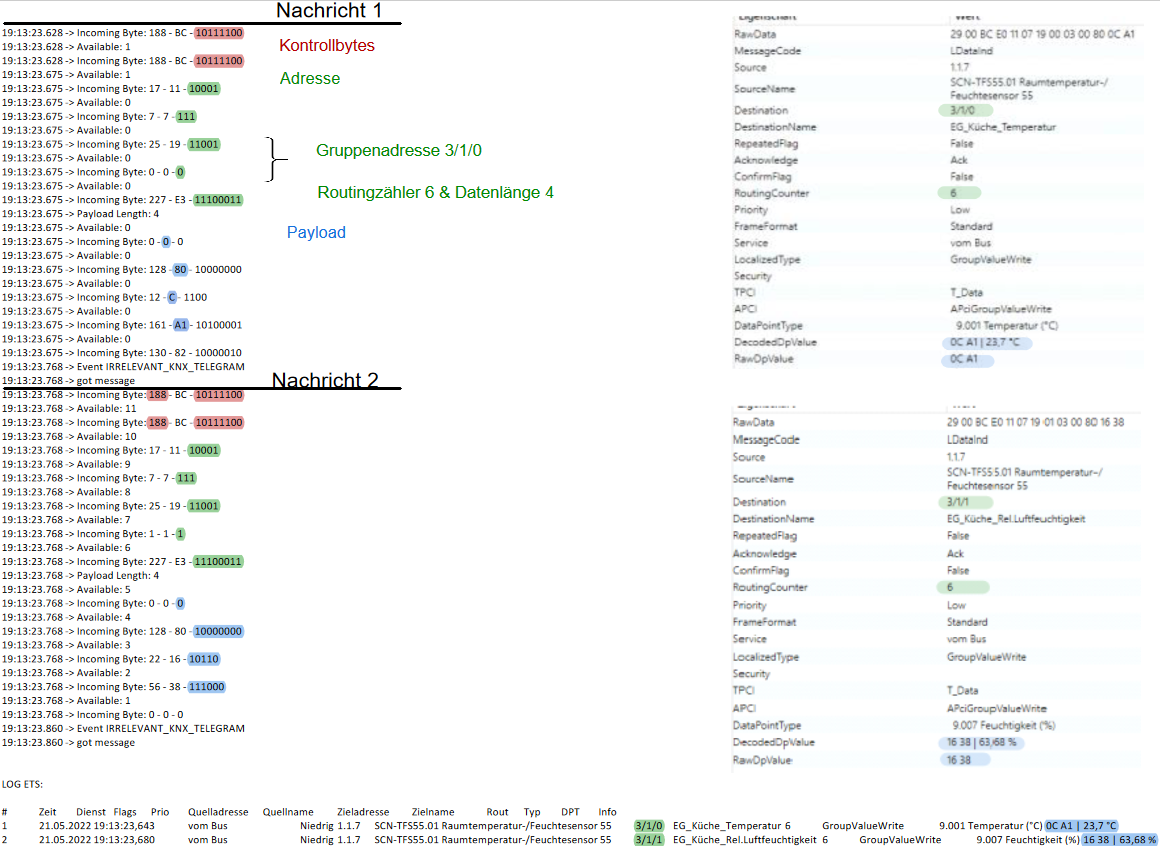
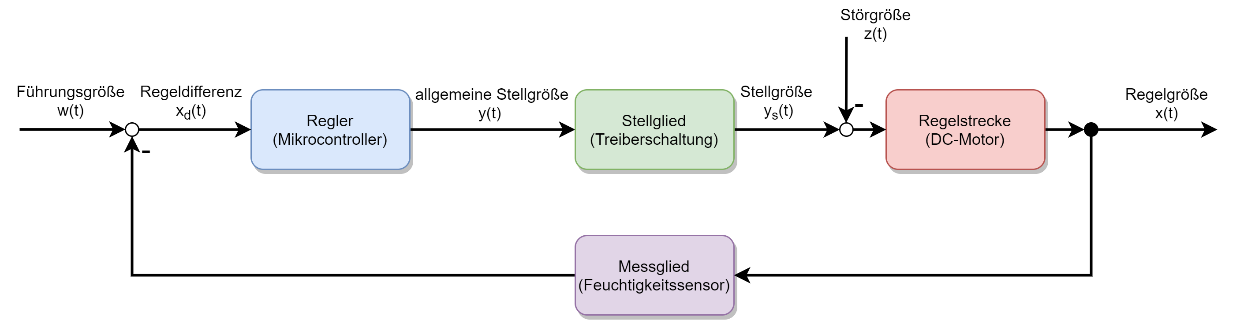


Tabelle : Arduino- Logging

## Regler

### Allgemein

Die Regelung ist ein Vorgang, bei dem die Führungsgröße dauerhaft mit der Regelgröße verglichen wird. Durch die Zugabe der Stellgröße auf die Regelstrecke, wird die Regelgröße so beeinflusst, dass die Regelabweichung möglichst minimiert oder ganz eliminiert wird. In der **Abbildung 30** sieht man einen geschlossenen Regelkreis.  
[47, S.23]



**Abbildung 30**: Blockschaltbild eines geschlossenen Regelkreises

Die Führungsgröße ist der vom Nutzer in der Software eingestellte Wunsch-Feuchtigkeitswert, den der Regler versucht einzustellen.

Die Regelgröße entspricht dem vom Feuchtigkeitssensor gemessenen Wert. Über eine Rückführung wird die Regelgröße mit der Führungsgröße verglichen. Die Regeldifferenz ergibt sich zu:

**Formel 6**: Regeldifferenz [48]

Je nach Höhe der Regelabweichung wird vom Regler die Stellgröße auf die Regelstrecke gegeben. Die Stellgröße entspricht dabei dem Tastgrad des PWM-Signals.

Durch die Rotation des Ankers im Permanentmagnetfeld wird eine Spannung zurück induziert. Man spricht hier von der Elektromotorische Kraft (kurz EMK). Die EMK reduziert dabei die wirksame Gesamtspannung und muss deshalb in Form einer Störgröße von der Stellgröße subtrahiert werden.

Grundsätzlich lassen sich Regler in zwei Typen aufteilen:

* Unstetige Regler
* Stetige Regler

Unstetige Regler können dabei nur eine begrenzte Anzahl an Stellwerten generieren.

Stetige Regler hingegen können beliebige Werte innerhalb des Stellbereichs ausgeben. [47, S.24]

### Sinnvolle Reglertypen

#### Zweipunkt-Regler

Der simpelste Regler ist der unstetige Zweipunkt-Regler, da er nur 2 verschiedene Zustände annehmen kann.

* Zustand 1 (Ist-Feuchtigkeitswert < Soll-Feuchtigkeitswert):
  + PWM-Signal mit minimalen Tastgrad wird ausgegeben
* Zustand 2 (Ist-Feuchtigkeitswert > Soll-Feuchtigkeitswert):
  + PWM-Signal mit hohem Tastgrad wird ausgegeben

Der Sollwert wird meist nicht als einziger Wert, sondern als Wertebereich übergeben. Durch den entstehenden Werte-Puffer wird verhindert, dass der Regler bei Grenzwerten ständig zwischen den Zuständen hin und herspringt.

Der Ausgabewert des Reglers lässt sich durch Hinzufügen weiterer Zustände verfeinern. Man spricht dann von einem Mehrpunkt-Regler. [49]

#### P-Regler

Beim stetigen Proportional-Regler ist namensgebend, die Stellgröße immer proportional zur Regelabweichung. An der Formel sieht man, dass nur ein Output y(t) ausgegeben wird, solange auch eine Regeldifferenz vorhanden ist. Deshalb lassen sich Störungen mit dem P-Regler nicht eliminieren und es bleibt eine dauerhafte Regelabweichung bestehen. Der Proportional-Regler reagiert dabei ohne Verzögerung auf jede Veränderung der Regelgröße. Mit dem Verstärkungsfaktor lässt sich einstellen, wie aggressiv der Regler auf den Fehler reagieren soll. Je höher der Wert desto schneller reagiert der Regler. Wird er jedoch zu hoch gewählt fängt der Regelkreis an zu schwingen. [47, S.27-29][50]

Formel : P-Regler [47, S.28]

#### PI-Regler

Für einen PI-Regler muss der P-Regler noch um einen Integralanteil ergänzt werden. Der I-Anteil summiert dabei die vergangenen Fehlerwerte über die Zeit auf. Dadurch wird vom integrierenden Anteil weiterhin eine Stellgröße ausgegeben, auch wenn die Regeldifferenz im Moment null entspricht. So kann der Steady-State Error komplett eliminiert werden. Damit keine Stellgröße mehr ausgegeben wird, muss die Regelabweichung über einen längeren Zeitraum null entsprechen.

Der P-Anteil fängt die ursprünglich auftretenden Regeldifferenz schnell ab jedoch ohne sie komplett zu beseitigen.

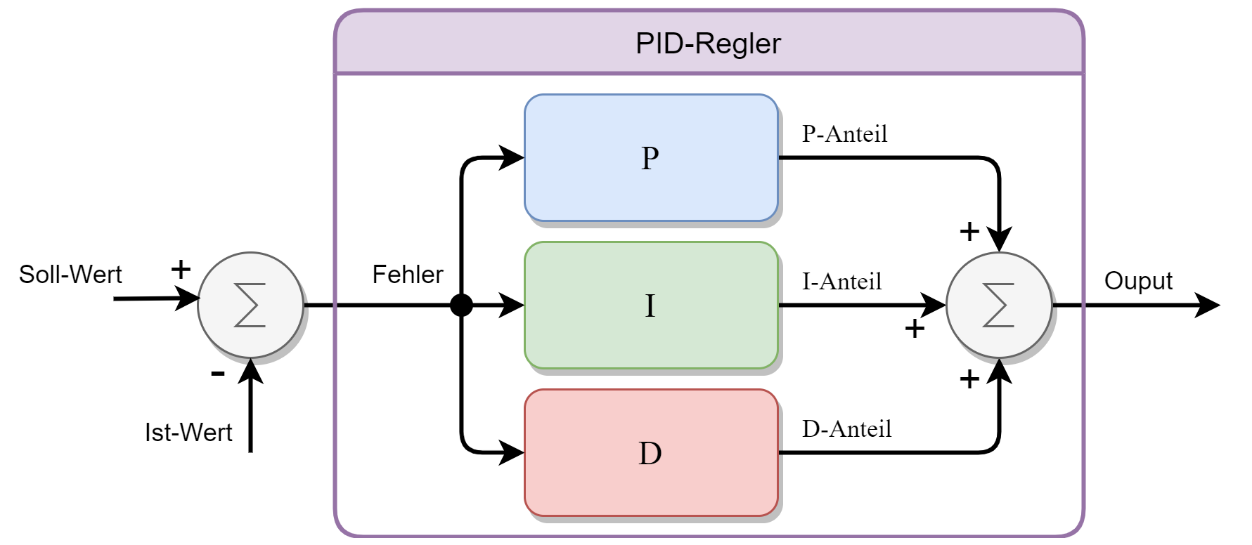
Der zusätzliche I-Anteil ist zwar langsam, dafür passt er aber langfristig die Regelgröße ohne jegliche Abweichungen an die Führungsgröße an.

Durch die Kombination der beiden Anteile erhält man den schnell und präzise arbeitenden PI-Regler. [47, S.40-41][51]

Die Nachstellzeit entspricht der Zeit, die nötig ist bis der Ausgabewert des I-Anteils dem Ausgabewert des P-Anteils folgt. Dementsprechend lässt sich durch eine Verringerung der Nachstellzeit ,der integrale Einfluss erhöhen. [49]

**Formel 8**: PI-Regler [47, S.41]

#### PID-Regler



**Abbildung 31**: Blockschaltbild eines PID-Reglers in Parallelstruktur

Durch das Hinzufügen des ableitenden Derivative-Anteils erhält man den PID-Regler. Der D-Anteil entspricht dabei der Änderungsrate der Regeldifferenz. Dadurch lässt sich die zukünftige Regelabweichung anhand der Steigung schon im Voraus abschätzen und ihr entgegenwirken. [47, S.42-44][52]

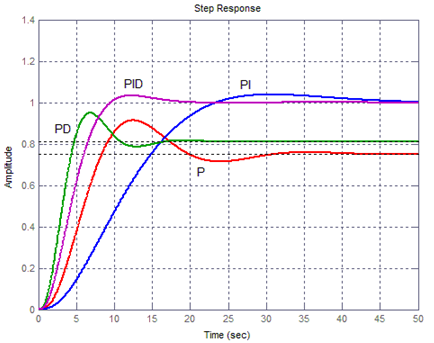
Angenommen die Regeldifferenz steigt rapide an, dann steigt aufgrund des P-Anteils auch proportional die Stellgröße an. Um jetzt das Überschwingen zu dämpfen, wird vom vorausschauenden D-Anteil schon frühzeitig eine negative Stellgröße ausgegeben. Dadurch schwingt sich der Regler schneller auf seinen Soll-Wert ein. Aufgrund des zusätzlichen D-Anteils reagiert der PID-Regler anfangs nochmal schneller auf Regelabweichungen als der Proportional-Regler.

Während der Fehler konstant ist, wird vom D-Anteil auch keine Stellgröße generiert. Der Einfluss des D-Anteils ist dabei proportional zu der Vorhaltzeit . [49]

Formel : PID-Regler [47, S.43]

In **Abbildung 32** sieht man die Sprungantworten der verschiedenen Regler.

* P-Regler schwingt über und erreicht den Soll-Wert von 1 nicht
* PI-Regler steuert deutlich langsamer aus, allerdings bleibt keine Regelabweichung
* PID-Regler reagiert und schwingt sehr schnell ein



**Abbildung 32**: Regler Sprungantworten [53]

Tendenziell addiert man zu den P, PI und PID-Formeln noch einen  
Minimums-Ausgabewert, den sogenannten bias-Wert. Der wäre in diesem Projekt der minimale Dutycycle bei dem die Lüfter noch laufen. Grund dafür ist, dass der Output nach einiger Zeit ohne Regelabweichung auf null gesetzt wird. Man möchte jedoch, dass sich die Lüfter dauerhaft drehen. [50]

### Realisierung

#### Auswahl des Reglers

Die Anforderung an dieses Projekt ist es einen stufenlosen Betrieb der Lüfter zu gewährleisten. Der Zwei-Punkt-Regler erfüllt diese Anforderung nicht. Man könnte einen Mehrstufen-Regler mit sehr vielen Zuständen entwerfen, jedoch ist es dann effizienter einen stetigen Regler herzunehmen.

Aufgrund der bleibende Regelabweichung des Proportional-Reglers kommt er für dieses Design auch nicht in Frage, da der vom Nutzer eingestellte Feuchtigkeitswert wirklich erreicht werden soll.

Es wurde sich für den PI-Regler entschieden, da ein langsames Anlaufen der Lüfter sowieso wünschenswert ist. Die erhöhte Aussteuergeschwindigkeit durch den D-Anteils ist bei diesem System nicht notwendig. Zumal der Regler über systematisches Probieren eingestellt werden muss, ist eine weitere Variable zur Parametrisierung eher kontraproduktiv. Außerdem reagiert der D-Anteil sehr empfindliche auf Störungen im Regelkreis.

#### Softwaretechnische Umsetzung

Der PI-Regler ist in der controller.c Source-Datei implementiert und wird dauerhaft von der statemachine\_process Funktion aufgerufen sobald der eingestellte Feuchtigkeitsgrenzwert überschritten wurde.

Die von der Integralkurve eingeschlossene Fläche lässt sich durch die Summe der Teilstücke approximieren.

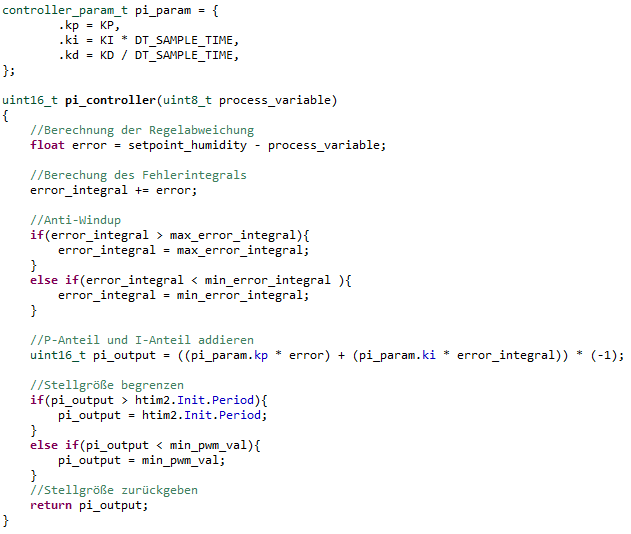
Deshalb wird zur softwareseitigen Implementierung das kontinuierliche Integral durch die Summe der Regelabweichung ersetzt. Dabei entspricht dem zeitlichen Abstand zwischen den jeweiligen Punkten. [48]

Formel : Diskreter I-Anteil [48]

Die Parameter des Reglers lassen sich direkt über die global\_config.h Header-Datei einstellen. Die konstante sample time wird dabei bereits im struct mit einbezogen. Dadurch erspart man sich eine Rechnung in der kontinuierlich aufgerufenen  
pi\_controller Funktion und verbessert so die Laufzeit.

Das Fehlerintegral soll sich nicht auf eine beachtliche Größe aufsummieren, denn dann dauert es auch entsprechend lange es wieder abzubauen

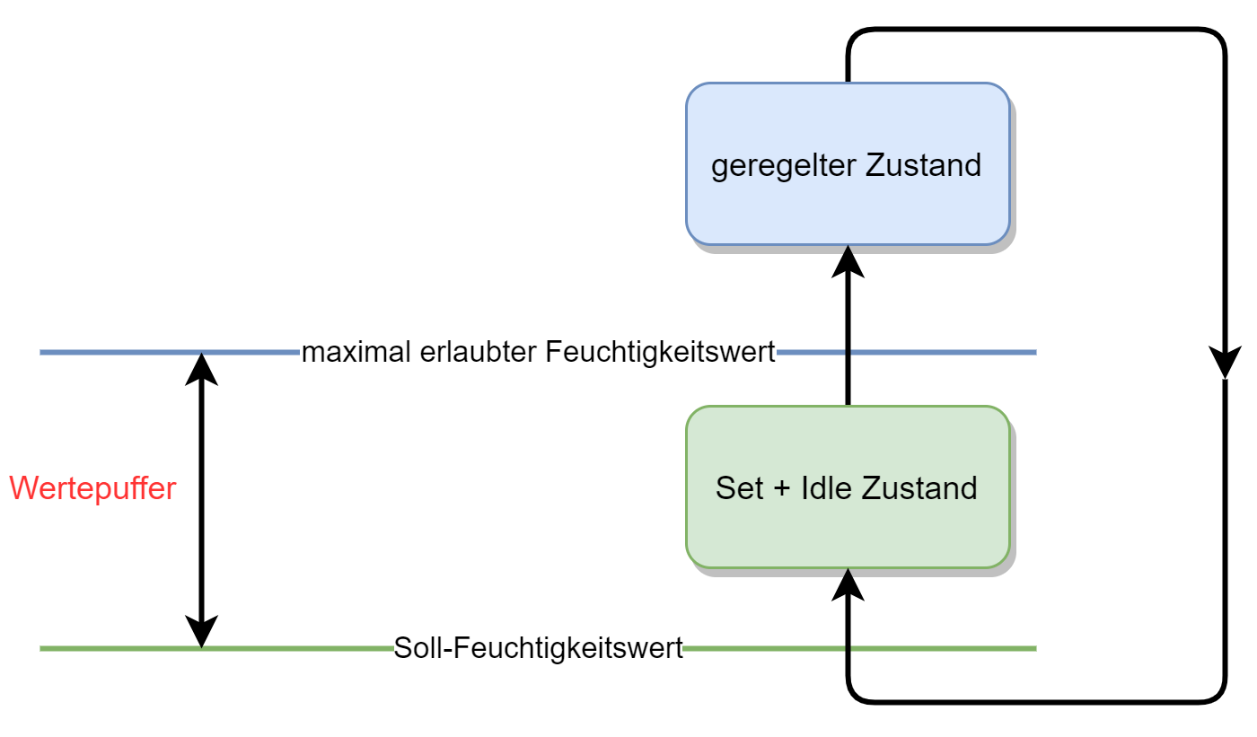
Deshalb wird das Fehlerintegral durch ein Anti-Windup Element auf einen maximalen Wert begrenzt. [54]



**Abbildung 33**: Software-Implementierung des PI-Reglers

Der Soll-Wert ist in diesem Projekt immer kleiner als der Ist-Wert, woraus stets eine negative Regelabweichung resultiert. Der Ausgabewert des Reglers muss deshalb noch mit dem Faktor (-1) korrigiert werden. Das liegt daran, dass die Stellgröße dem Tastgrad eines PWM-Signals entspricht und somit immer nur positive Werte annehmen kann. Des Weiteren wird das Stellsignal noch begrenzt und zurückgegeben.

Sobald der Soll-Wert erreicht oder unterschritten wurde, wechselt der Mikrocontroller von dem Controlled State zurück in den Set State. Da der eingestellte, maximale Feuchtigkeitswert höher ist als der Soll-Wert wird durch den dazwischen liegenden Wertepuffer dafür gesorgt, dass nicht rapide zwischen den Zuständen gesprungen werden kann.



**Abbildung 34**: Zusammenhang zwischen Feuchtigkeitswert und Zustand

Nachdem die Regeldifferenz immer negativ ist, kann sich die Integralsumme nicht von allein abbauen und wird deswegen bei jedem Eintritt in den Controlled State zurückgesetzt.

# Fazit

### Aktueller Stand der Arbeit

Die Arbeit befindet sich aktuell (Stand 01.07.2022) in einem Zustand, in dem alle Anforderungen an das Projekt erfüllt worden sind. Die Ansteuerung der einzelnen Lüfter funktioniert sowohl statisch mit kontinuierlichem Richtungswechsel, sowie dynamisch über den PI-Regler. Außerdem funktioniert die Kommunikation über den KNX-Bus. Es lassen sich die in der ETS definierten Daten über Telegramme empfangen und entschlüsseln.

### To-Do Liste

Der Feuchtigkeitswert muss noch dynamisch an die Jahreszeiten angepasst werden, in dem der jeweilige Monat vom Bus ausgelesen wird.

Es müssen noch manuelle Lüftermodi einprogrammiert werden, die dann über ein Tastsensor gestartet werden.

Außerdem muss der PI-Regler noch genau parametrisiert werden.

In der nächsten Hardwareversion werden alle bisher erfassten Erkenntnisse einfließen.

### Empfehlungen für Hardware 2.0

* Platine auf 2 Lagen reduzieren

Viele der kritischen Verbindungen sind nur durch einen Anzeigefehler in EAGLE entstanden. Dabei wurde die Verbindung im Schaltplan angezeigt jedoch nicht im Layout.

* Schraubklemmen statt Klemmleisten

Die Klemmleisten sind sehr schwergängig und ehre für größere Platine geeignet.

* Größeres Package für passiven Bauteile und Inverter ICs

Erleichtert die Fertigung und auf der Platine ist genug Platz.

* STM32 Mikrocontroller mit mehr Flashspeicher

Die HAL-Treiber haben unerwartet viel Speicherplatz belegt.

### Schwierigkeiten & Probleme

Die UART-Kommunikation hat im Testaufbau mit dem Nucleo-Entwicklungsboard als Empfänger und ESP8226 als Sender anfangs große Probleme bereitet. Das Nucleo-Board wird über einen USB-Anschluss mit Spannung versorgt und geflasht. Dabei kam es zu Komplikationen zwischen dem UART to USB Converter auf dem Board und der CubeIDE auf dem PC. Die UART-Kommunikation wurde dadurch gestört. Mit einem LogicAnalyzer konnte nachgewiesen werden, dass es auch nicht möglich war, mit dem Nucleo-Board Daten zu senden. Mit der eigens entwickelten Hardware funktioniert die Kommunikation einwandfrei, da bei ihr die UART-Pins und die Pins zum Programmieren getrennt sind. Das Arbeiten mit dem Entwicklungsboard hatte den Hintergedanken, zunächst eine funktionierende Software zu schreiben, bevor man diese auf einer gegebenenfalls noch fehlerbehafteten Hardware debuggt und so das Potential zweier Fehlerquellen hat.

Nachdem dieses Projekt meinen ersten Leiterplattenentwurf darstellt, musste ich mir dementsprechend im Vorfeld die kompletten Grundlagen zu EAGLE und dem Platinen layouten aneignen.

Ein weiteres Problem stellten die langen Lieferzeiten und der geringe Bestand von der elektronischen Bauteilen dar. Die Komponenten mussten mehrfach im Schaltplan und Layout geändert werden, da Bauteile kurzfristig abgekündigt wurden und nicht mehr verfügbar waren.

Da die Eigenentwicklung von KNX-Projekten noch nicht die breite Masse erreicht hat, war es schwierig die nötige Information für die Anbindung zu finden. Während des Projekts war es zeitweise unklar, ob überhaupt eine Kommunikation mit dem STM32 realisierbar ist. Zum jetzigen Zeitpunkt gibt es keine nützlichen Informationen über die Kombination aus STM32 und KNX im Internet.

### Ausblick

Der Prototyp wurde in der Praxis und in einem Smart Home getestet. Diese Bachelorarbeit hat alle Grundlagen geschaffen, um die nächste Prototypen Generation zu entwickeln und eine Vorserie zu starten.

Literaturangaben

Alle Internetquellen wurden am 05.07.2022 das letzte mal abgerufen

[x] Nachname, Vorname: Titel. Erscheinungsjahr, Internetadresse

[1] KNX Association: KNX Grundlagen. n/a, https://www.knx.org/wAssets/docs/downloads/Marketing/Flyers/KNX-Basics/KNX-Basics\_de.pdf

[2] Digital Guide IONOS: CSMA/CA: Definition und Erklärung des Verfahrens. 2018, https://www.ionos.de/digitalguide/server/knowhow/csmaca-carrier-sense-multiple-access-with-collision-avoidance/

[3] inVENTer GmbH: IV14-ZERO Produktbeschreibung. n/a, https://www.inventer.de/produkte/lueftung-mit-waermerueckgewinnung/schalldaemmlueftung/iv14-zero/

[4] iTecNotes: Electrical – Self Inductance in Steady State equivalent model of Brushed DC machine. n/a, https://itecnotes.com/electrical/electrical-self-inductance-in-steady-state-equivalent-model-of-brushed-dc-machine/

[5] Conrad Ratgeber: Elektromotoren >> Aufbau, Funktionsweise und Arten einfach erklärt. 2021, <https://www.conrad.de/de/ratgeber/technik-einfach-erklaert/elektromotor.html>

[6] oswos: DC-Motor. n/a, https://oswos.com/de/dc-motoren/#pll\_switcher

[7] RS Components: Leitfaden zu Gleichstrommotoren. n/a, https://de.rs-online.com/web/generalDisplay.html?id=ideen-und-tipps/dc-motoren-leitfaden

[8] Brown, Nicholas: Introduction To PWM: How Pulse Modulation Works. n/a, <https://www.kompulsa.com/introduction-pwm-pulse-width-modulation-works/>

[9] Dogan, Ibrahim: Designing Embedded Systems with 32-Bit PIC Microcontrollers and MikroC. 2014, <https://www.sciencedirect.com/topics/engineering/pulse-width-modulation>

[10] All About Circuits: Pulse Width Modulation. n/a, <https://www.allaboutcircuits.com/textbook/semiconductors/chpt-11/pulse-width-modulation/>

[11] Wayne, Storr: Pulse Width Modulation. 2014, https://www.electronics-tutorials.ws/blog/pulse-width-modulation.html

[12] Studyflix: Elektrotechnik Grundlagen Mosfet. n/a, <https://studyflix.de/elektrotechnik/mosfet-2537>

[13] Wayne, Storr: The MOSFET. 2014, <https://www.electronics-tutorials.ws/transistor/tran_6.html>

[14] Wayne, Storr: MOSFET as a Switch. 2014, <https://www.electronics-tutorials.ws/transistor/tran_7.html>

[15] Wayne, Storr: Transistor Tutorial Summary. 2014, <https://www.electronics-tutorials.ws/transistor/tran_8.html>

[16] bitsmack: Electrical Engineering Forum. 2014, <https://electronics.stackexchange.com/questions/102819/driving-dc-motors-with-mosfets-and-a-microcontroller>

[17] Yong, Aaron: Quora Forum. 2017, https://www.quora.com/What-is-an-%E2%80%9CH-bridge%E2%80%9D

[18] Maestre, Susie: How Does a Flyback Diode Work?. 2021, <https://www.circuitbread.com/ee-faq/how-does-a-flyback-diode-work>

[19] Sattel, Sam: EAGLE Academy - What are Decoupling Capacitors. n/a, <https://www.autodesk.com/products/eagle/blog/what-are-decoupling-capacitors/>

[20] Opternus Components: TP-UART2 Die Nächste Generation. n/a, <https://www.opternus.com/de/siemens/knx-chipset/tpuart2-die-naechste-generation>

[21] Opternus Components: TP-UART2 Board BTM2-PCB. n/a, https://www.opternus.com/de/siemens/entwicklungs-werkzeuge/tp-uart2-board-btm2-pcb

[22] Rayming PCB & Assembly: 4 Layer PCB Layout Tutorial, Stack-up design and Cost of manufacturing. n/a, <https://www.raypcb.com/4-layer-pcb/>

[23] Yogendrappa, Milan: How to Build a Multilayer PCB Stack-up. 2021, <https://www.protoexpress.com/blog/build-multilayer-pcb-stack-up/>

[24] Robledo, Edwin: Everything You Need to Know About Trace Width. 2021, <https://www.autodesk.com/products/fusion-360/blog/trace-width/>

[25] Segger Microcontroller Systeme GmbH: Serial Wire Debug. 2015, <https://wiki.segger.com/SWD>

[26] STMicroelectronics: ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32. n/a, <https://www.st.com/en/development-tools/st-link-v2.html>

[27] STMicroelectronics: RM0377 Reference Manual. n/a, <https://www.st.com/resource/en/reference_manual/rm0377-ultralowpower-stm32l0x1-advanced-armbased-32bit-mcus-stmicroelectronics.pdf>

[28] STMicroelectronics: STM32L041x6 Datasheet. n/a, <https://www.st.com/resource/en/datasheet/stm32l041c6.pdf>

[29] Rohde & Schwarz: UART verstehen. n/a, <https://www.rohde-schwarz.com/de/produkte/messtechnik/oszilloskope/educational-content/uart-verstehen_254524.html>

[30] Magdy, Khaled: UART Serial Communication With PIC Microcontrollers Tutorial. 2018, <https://deepbluembedded.com/uart-pic-microcontroller-tutorial/>

[31] MKS075: Difference between Serial and Parallel Transmission. n/a, https://www.geeksforgeeks.org/difference-between-serial-and-parallel-transmission/

[32] Weissgärber, Jirka: Die UART Schnittstelle. 2019, <http://www.mathe-mit-methode.com/schlaufuchs_web/elektrotechnik/mikrocontroller_lernmaterial/microcontroller_allgemein/mikrocontroller_ext_hardware/mikrocontroller_uart.html>

[33] Electric Imp Dev Center: UART Explained. n/a, <https://developer.electricimp.com/resources/uart>

[34] Controllers Tech: Receive UART data using POLL, INTERRUPT and DMA. n/a, <https://controllerstech.com/uart-receive-in-stm32/>

[35] Majerle, Tilen: STM32 tutorial: Efficiently receive UART data using DMA. 2017, <https://stm32f4-discovery.net/2017/07/stm32-tutorial-efficiently-receive-uart-data-using-dma/>

[36] Magdy, Khaled: STM32 Timers Tutorial | Hardware Timers Explained. 2020, https://deepbluembedded.com/stm32-timers-tutorial-hardware-timers-explained/

[37] STMicroelectronics: STM32L4–Timers. n/a, <https://www.st.com/content/ccc/resource/training/technical/product_training/c4/1b/56/83/3a/a1/47/64/STM32L4_WDG_TIMERS_GPTIM.pdf/files/STM32L4_WDG_TIMERS_GPTIM.pdf/jcr:content/translations/en.STM32L4_WDG_TIMERS_GPTIM.pdf>

[38] Magdy, Khaled: STM32 PWM Example – Timer PWM Mode Tutorial. 2020, <https://deepbluembedded.com/stm32-pwm-example-timer-pwm-mode-tutorial/>

[39] International Rectifier: IRF7205PbF Datasheet. n/a, <https://www.mouser.de/datasheet/2/196/Infineon_IRF7205_DataSheet_v01_01_EN-1732562.pdf>

[40] Lakkas, George: MOSFET power losses and how they affect power-supply efficiency. 2016, <https://www.ti.com/lit/an/slyt664/slyt664.pdf?ts=1656850368857&ref_url=https%253A%252F%252Fwww.google.com%252F>

[41] STMicroelectronics: STM32U5 Low-Power Timer (LPTIM). n/a, <https://www.st.com/content/ccc/resource/training/technical/product_training/group1/78/dc/d9/2b/08/16/4e/f8/STM32U5-Peripheral-LowPowerTimer_LPTIM/files/STM32U5-Peripheral-LowPowerTimer_LPTIM.pdf/_jcr_content/translations/en.STM32U5-Peripheral-LowPowerTimer_LPTIM.pdf>

[42] Cimory: Electrical Engineering Forum. 2020, <https://electronics.stackexchange.com/questions/528915/configuring-lptim-parameters-with-the-stm32l4>

[43] KNX Association: Grundlagenwissen zum KNX Standard. 2013, https://www.sius.gmbh/files/cto\_layout/PDF%20Dokumente/KNX%20Association/Grundlagenwissen\_zum\_KNX\_Standard\_German.pdf

[44] Schmidt, Matthias: Paketaufbau. 2015, <https://redaktion.knx-user-forum.de/lexikon/paketaufbau/#prettyPhoto>

[45] InfoTip Kompendium: KNX-INSTALLATIONSBUS. n/a, <https://kompendium.infotip.de/knx-installationsbus.html>

[46] Gehrig, Thorsten: Arduino EIB/KNX Interface via TP-UART. 2020, <https://github.com/thorsten-gehrig/arduino-tpuart-knx-user-forum>

[47] Samson AG: Technische Information - Regler und Regelstrecken. 2012, <https://www.samsongroup.com/document/l102de.pdf>

[48] APMonitor: Proportional Integral (PI) Control. n/a, <https://apmonitor.com/pdc/index.php/Main/ProportionalIntegralControl>

[49] JUMO: FAQ zum Thema Regler. n/a, <https://www.jumo.de/web/services/faq/controller>

[50] Cooper, Douglas: The P-Only Control Algorithm. 2015, <https://controlguru.com/the-p-only-control-algorithm/>

[51] Cooper, Douglas: Integral Action and PI Control. 2015, <https://controlguru.com/integral-action-and-pi-control/>

[52] Cooper, Douglas: PID Control and Derivative on Measurement. 2015, https://controlguru.com/pid-control-and-derivative-on-measurement/

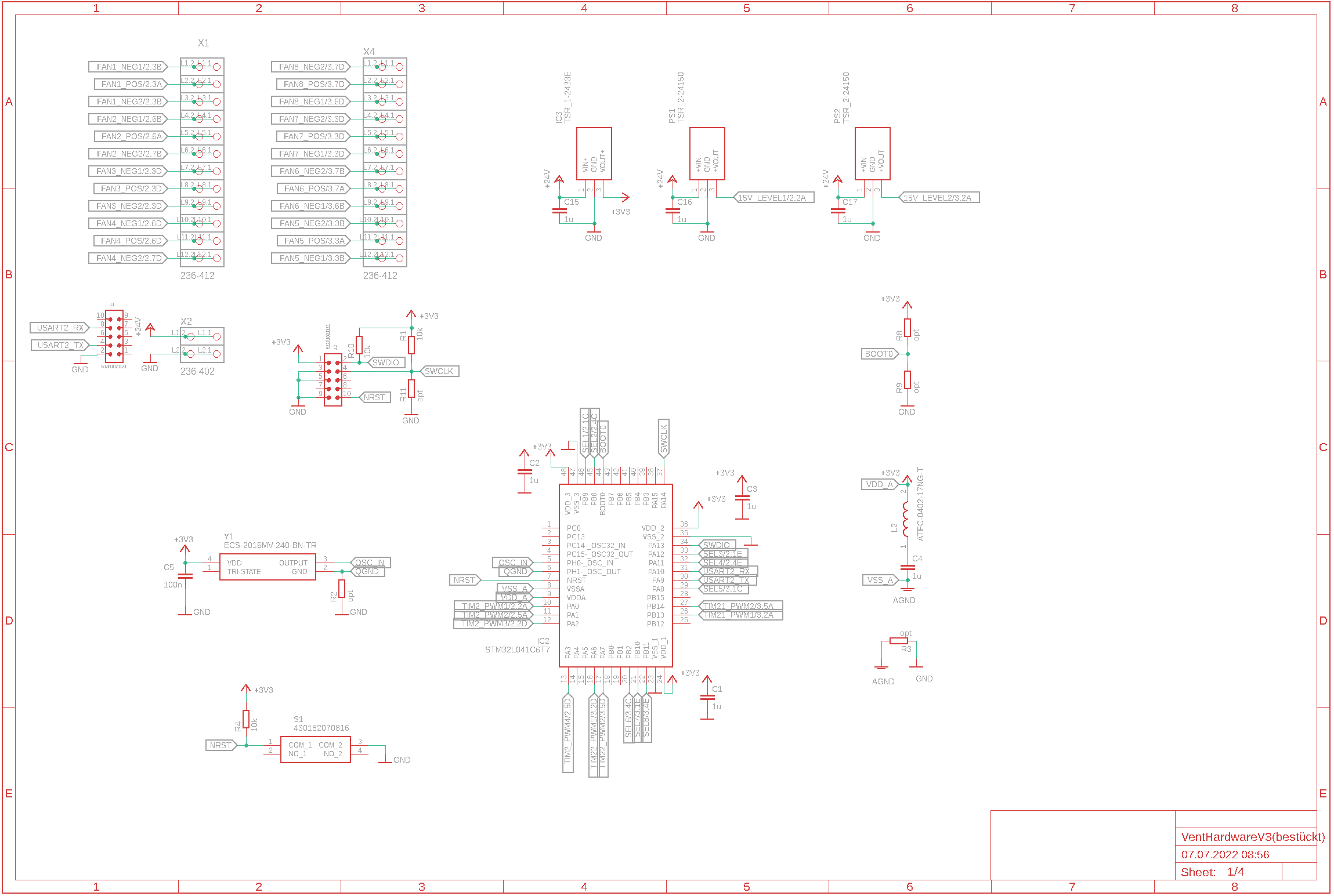
[53] SZTAKI: Example for continuous PID controller design using pole cancellation technique -Figure 2. n/a, <http://sysbook.sztaki.hu/sysbook6.php?page=97&lang=en&left=theory&right=edu>

[54] Smallwood, Jordan: PID Controllers. 2017, <https://www.arxterra.com/lecture-6-pid-controllers/#Differential_Term>

[55] Electronics Notes: PLL Phase Locked Loop Tutorial & Primer. n/a, <https://www.electronics-notes.com/articles/radio/pll-phase-locked-loop/tutorial-primer-basics.php>

[56] Luftbude: Inventer iV14 Zero Komplettset. n/a, <https://www.luftbude.de/dezentrale-lueftung/inventer/inventer-iv14-zero-komplettset-LB10006/>

[57] Magdy, Khaled: STM32 HAL Library Tutorial. 2020, https://deepbluembedded.com/stm32-hal-library-tutorial-examples/

Anhang

