**Projekt-Dokumentation**

Log-Buch

Inhalt

[**04.01.2025** 8](#_Toc194514040)

[Start & Basisarchitektur des Arduino MEGA2560 8](#_Toc194514041)

[ Ziel: Entwicklung und Implementierung der grundlegenden Architektur 8](#_Toc194514042)

[ Übersicht der Module und deren Funktionen 8](#_Toc194514043)

[ Auswahl der Software & Tools (Sloeber IDE vs. Arduino IDE) 8](#_Toc194514044)

[**17.01.2025** 8](#_Toc194514045)

[GitHub und Versionsverwaltung 8](#_Toc194514046)

[Anlegen der Repos für Embedded Software, HAS und Dockerfiles 8](#_Toc194514047)

[Einrichtung von GitHub zur Verwaltung von Tasks 8](#_Toc194514048)

[**18.01.2025** 8](#_Toc194514049)

[Docker-Setup für HAS 8](#_Toc194514050)

[Auswahl von Ubuntu als Basis 8](#_Toc194514051)

[Entwicklung des Dockerfiles und Docker Compose mit Felix Latzer 8](#_Toc194514052)

[Integration von MongoDB & 3rd-Party-Plugins 8](#_Toc194514053)

[Tryouts auf verschiedenen Hosts (Raspberry Pi 4B) 8](#_Toc194514054)

[**24.01.2025** 9](#_Toc194514055)

[Meetings mit VAT & Entscheidungsfindung zur Kommunikation 9](#_Toc194514056)

[ Diskussion zur Nutzung des VAT-Slave uC 9](#_Toc194514057)

[ Entscheidung gegen EtherCAT & für Ethernetmodul von VAT 9](#_Toc194514058)

[ Anpassung des Ethernetmoduls für GET/SET-Kommunikation 9](#_Toc194514059)

[**25.01.2025** 9](#_Toc194514060)

[Meetings & Zusammenarbeit mit Felix 9](#_Toc194514061)

[ Zielsetzung der Meetings 9](#_Toc194514062)

[ Fortschritte bei der Kommunikation zwischen HAS und eSW 9](#_Toc194514063)

[ End-to-End Testing mit Sensoren, HAS & MongoDB 9](#_Toc194514064)

[**07.02.2025** 9](#_Toc194514065)

[Meetings mit Dominik 9](#_Toc194514066)

[Einführung in die Architektur 9](#_Toc194514067)

[Debugging-Ansätze und Architektur-Erweiterung 9](#_Toc194514068)

[Spikes und Tests für HighCurrent/Flyback-Modul 9](#_Toc194514069)

[**14.02.2025** 10](#_Toc194514070)

[Debugging-Methoden 10](#_Toc194514071)

[ Serial Prints 10](#_Toc194514072)

[ StackOverflow, Blogposts, LLMs 10](#_Toc194514073)

[ Austausch mit Kollegen 10](#_Toc194514074)

[**23.02.2025** 10](#_Toc194514075)

[Ergebnisse aus Debugging & Verbesserungen 10](#_Toc194514076)

[Änderungen der Kommunikationsinterfaces 10](#_Toc194514077)

[Updates für verschiedene Module 10](#_Toc194514078)

[**07.03.2025** 10](#_Toc194514079)

[Programmierung & Libraries 10](#_Toc194514080)

[Programmiersprache: C/C++ 10](#_Toc194514081)

[Externe Libraries: Auswahl & Anpassungen 10](#_Toc194514082)

[**13.03.2025** 11](#_Toc194514083)

[Code-Dokumentation mit Doxygen 11](#_Toc194514084)

[ Einrichtung auf GitHub 11](#_Toc194514085)

[ Generierung von HTML- und LaTeX-Dokumenten 11](#_Toc194514086)

[**14.03.2025** 11](#_Toc194514087)

[ Allgemeine Dokumentation 11](#_Toc194514088)

[ Strukturierte Dokumentation der Codebase 11](#_Toc194514089)

[ Zusätzliche Anleitungen & Informationen 11](#_Toc194514090)

[**15.03.2025** 11](#_Toc194514091)

[Zusammenfassung & Review aller Komponenten 11](#_Toc194514092)

[ Letzte Anpassungen und Optimierungen 11](#_Toc194514093)

[**16.03.2025** 11](#_Toc194514094)

[Abschluss & Lessons Learned 11](#_Toc194514095)

[Herausforderungen & Lösungen 11](#_Toc194514096)

[Reflexion des gesamten Projekts 11](#_Toc194514097)

[**21.03.2025** 12](#_Toc194514098)

[**29.03.2025** 12](#_Toc194514099)

# **04.01.2025**

# Start & Basisarchitektur des Arduino MEGA2560

# Ziel: Entwicklung und Implementierung der grundlegenden Architektur

# Übersicht der Module und deren Funktionen

# Auswahl der Software & Tools (Sloeber IDE vs. Arduino IDE)

# **17.01.2025**

# GitHub und Versionsverwaltung

# Anlegen der Repos für Embedded Software, HAS und Dockerfiles

# Einrichtung von GitHub zur Verwaltung von Tasks

# **18.01.2025**

# Docker-Setup für HAS

# Auswahl von Ubuntu als Basis

# Entwicklung des Dockerfiles und Docker Compose mit Felix Latzer

# Integration von MongoDB & 3rd-Party-Plugins

# Tryouts auf verschiedenen Hosts (Raspberry Pi 4B)

# **24.01.2025**

# Meetings mit VAT & Entscheidungsfindung zur Kommunikation

# Diskussion zur Nutzung des VAT-Slave uC

# Entscheidung gegen EtherCAT & für Ethernetmodul von VAT

# Anpassung des Ethernetmoduls für GET/SET-Kommunikation

# **25.01.2025**

# Meetings & Zusammenarbeit mit Felix

# Zielsetzung der Meetings

# Fortschritte bei der Kommunikation zwischen HAS und eSW

# End-to-End Testing mit Sensoren, HAS & MongoDB

# **07.02.2025**

# Meetings mit Dominik

# Einführung in die Architektur

# Debugging-Ansätze und Architektur-Erweiterung

# Spikes und Tests für HighCurrent/Flyback-Modul

# **14.02.2025**

Meetings mit Bonelli

Kommunikation zwischen Manual-Board und eSW

Erweiterungsmöglichkeiten der Architektur

Schnittstellen & verwendete Hard

**22.02.2025**

# Debugging-Methoden

# Serial Prints

# StackOverflow, Blogposts, LLMs

# Austausch mit Kollegen

# **23.02.2025**

# Ergebnisse aus Debugging & Verbesserungen

# Änderungen der Kommunikationsinterfaces

# Updates für verschiedene Module

# **07.03.2025**

# Programmierung & Libraries

# Programmiersprache: C/C++

# Externe Libraries: Auswahl & Anpassungen

# **13.03.2025**

# Code-Dokumentation mit Doxygen

# Einrichtung auf GitHub

# Generierung von HTML- und LaTeX-Dokumenten

# **14.03.2025**

# Allgemeine Dokumentation

# Strukturierte Dokumentation der Codebase

# Zusätzliche Anleitungen & Informationen

# **15.03.2025**

# Zusammenfassung & Review aller Komponenten

# Letzte Anpassungen und Optimierungen

# **16.03.2025**

# Abschluss & Lessons Learned

# Herausforderungen & Lösungen

# Reflexion des gesamten Projekts

# **21.03.2025**

Agenda Auf Gitub:

**Flyback Module**  
-Flyback check (testingbranch, main branch)  
-Interrupt für Frequenzmessung (andere Lösung)

**Flyback working**

* Kommunikation für HAS (Endpoint)

**Versuchsaufbau**

* Flyback funktionalität
* Mit TEST SOFTWARE auslesen
* Vakuum testen (Ziel Vakuum erreicht oder in der Nähe)

**Thermosensor**

* Bestellen

**Steuerung**

* Besprechung der Funktion (Kraus Naimer Schalter)

**Klemmenmodul**

* gemeinsame Lösung ( wenn ja --> Illwerke)

**VAT Komponenten**

* HAS REQUESTS GET SET (ENDPOINT)
* Fully Implement GET; SET Compound (1,2,3)
* Testing mit HAS

**Schaltplan**  
-BONELLI erklären

**DISPLAY**

* PI Lösung finden

# **29.03.2025**

**Vakuumpumpe Programm**

* Building up vacuumpump logic //not done

**Vakuumpumpe working**

* Kommunikation für HAS (Endpoint) //not done

**Zusammenfügung Flyback und Vakuumpumpe**

* Sollte Logic funktionieren ( Architektur für einen Controlling Task) //not done

**Versuchsaufbau**

* Vakuum testen (Ziel Vakuum erreicht oder in der Nähe) // Nochmal für Doku Aufnahme DONE

**Thermosensor**

* Sensoren müssen morgen bestellt werden //MONTAG BONELLI
* Bausteine SPI für Auslesung sind bestellt //DONE

**Simulation Relaissteuerung Arduino**

* LTSPICE Simulation für Relaissteuerung //not done

**Arduino ExtenderBoard**

* Erstellen des Projektes (Idee)

**Klemmenmodul**

* gemeinsame Lösung ( wenn ja --> Illwerke) // open nice to have

**VAT Komponenten**

* HAS REQUESTS GET SET (ENDPOINT) //DONE
* Fully Implement GET; SET Compound (1,2,3) //not done
* Testing mit HAS // DONE 90%

**DUTYCYCLE Implementation Flyback**

* Zuzüglich unserer Frequenzeinstellung sollen wir auch den DutyCycle einstellen //DONE
* Test mit HAS //DONE

**Treiber austesten**

* Nicht möglich, Probleme mit STU klären

# **05.04.2025**

* Tryout mit Pfeiffer Vaccum Gerät um Drücke zu überprüfen.
  + Falsche einstellungen in VAT Software, diese durch Parameter in UI konfiguriert
    - Franic fragt bei Rolf nach, Probleme mit mapping/skalierung
* Aufbau und tryout von vacControl, sowohl als Code-Review mit Franic
* Bugfixes in ReportSystem und bugfixes von Compound getter/setter

# **17.04.2025**

* Verbesserung Logging und Printing, Konzept des LogManagers, Vorbereitung für Integration
* Hilfe für Franic mit VacControl, Endpoints, Debugging, Fehlersuche
* Versuchsaufbau mit beiden Controllern
  + Nach langen Einsätzen und unter starker Belastung steigt die Temperatur des EthernetShields Controller auf knapp 60°, daher verwenden wir Kühlkörper.
    - Nach Dauerbetrieb von 09:00 bis 18:00
  + (Passive Kühlung) um dem Problem entgegenzuwirken
  + Mit Fluke TiS20 Messung von beiden EthernetShield’s
* Optimierung der Speichernutztung unter Verwendung von Flashstrings um SRAM zu schonen, Optimierung des SerialMenu, analyze mittels:
  + Bessere loglevel, weniger spam in der seriellen Konsole

Ein Bild, das Text, Screenshot, Electric Blue (Farbe) enthält.

KI-generierte Inhalte können fehlerhaft sein.Arduino\_MEGA2560\_W5100\_EthernetShield Elegoo\_MEGA2560\_W5100\_EthernetShield

Ein Bild, das Text, Screenshot, Electric Blue (Farbe), Majorelle Blue enthält.

KI-generierte Inhalte können fehlerhaft sein.

# **19.04.2025**

* Ausarbeitung des Header Files und cpp files für den LogManager, erste Tests mit verschiedenen SD-Karten und Formaten
  + SD-Karte mit 32GB FAT32
  + SD-Karte mit 2GB FAT16
* Fehlersuche bei LogManager Libraries, probleme mit pinbelegung und SPI (gleichzeitiges benutzten von Ethernet und SD-Card nicht möglich)
  + Switchen zwischen den Teilnehmern im Bus, verwendung von pinMode und richtigen timings…
* Anbringen von passiver Kühlung auf IC für EthernetShield

Ein Bild, das Elektronik, Elektrisches Bauelement, Elektronisches Bauteil, passives Bauelement enthält.

KI-generierte Inhalte können fehlerhaft sein.

# **25.04.2025**

* Nach Franic’s Anruf bei VAT, talk mit Embedded Entwickler richitge Range für Drucksensor eingestellt.
  + API Endpoints Funktion des VacControls getestet
  + Task für VacControl getestet
  + Code optimiereung
* Weitere Test von MicroSD Karten für Logging

# **26.04.2025**

* Testaufbau mit Franic, testen neue Range von Sensor.
  + PI- Regler Einstellungen vorgenommen um Pressure Control mode zu optimieren.
    - Anstatt Downstream Mode wurde nun der Upstream Mode verwendet und 2 x Learning Mode ausgeführt um Daten im CSV zu sammeln.
    - Eventuelle Absprache oder Austausch mit VAT bezüglich generierter Daten und Verhalen.
* Finalisierung des VacControl und kleiner Bugfixes, Codereview
* Anpassung des FlyBackVacControlTask
  + erhöhung der max. Erlaubten Stacksize um overflow zu verhindern
* StackMonitorTask Anpassungen für max. Grenzwerte des Stacks
* Anpassungen API-Endpoints, sowie aufräumen von alten Debug Messages usw.
* Bugfix und einige Tryouts im logManager, Probleme mit SPI Zugriff bzw setzten der States der betroffenen Ports

# **Architekturübersicht – FFRESW (Farnsworth Fusion Reactor Embedded Software)**

**Einführung**

FFRESW ist die Embedded Software des Farnsworth Fusionsreaktors. Sie basiert auf C++ und läuft auf einem Arduino MEGA2560. Aufgrund der Plattform wurde C++ gewählt, da diese mit .ino-Dateien und der Arduino-IDE kompatibel ist. Der objektorientierte Aufbau erleichtert Wartung, Erweiterbarkeit und Modularisierung.

Die Software ist in **Core Libraries** (eigene Entwicklungen) und **externe Libraries** unterteilt.

**1. Core Libraries (eigene Entwicklungen)**

**1.1 comModule**

Das comModule bietet Zugriff auf die verschiedenen Schnittstellen des Systems:

* **ETH** (Ethernet): Kommunikation mit dem HAS (Hardware Access Service) und dem VAT µC.
* **I2C**: Besonders wichtig zur Auswertung der Temperaturdaten.
* **SER** (Serielle Schnittstelle)
* **SPI**: Zentral für das Ethernet-Shield und den LogManager. Darüber wird gesteuert, wer wann senden darf.

Diese Library wird genutzt, um Daten zu senden, zu empfangen sowie Konfigurationen zu übertragen.

**1.2 calcModule**

Das calcModule enthält verschiedene Funktionen zur Datenverarbeitung:

* Umrechnung von Sensorwerten und Kammerdaten
* Finalisierung, Parsing und Bearbeitung von Rohdaten
* Berechnung abgeleiteter Werte

**1.3 sensModule**

Das sensModule stellt die Verbindung zu den Sensor-Libraries **Temperature** und **Pressure** her. Es übernimmt:

* Überwachung und Prüfung von Sensordaten
* Abstraktion zur einfachen Einbindung neuer Sensoren
* Einheitliches Interface zur Sensordatenerfassung

**1.4 jsonModule**

Ein integraler Bestandteil der Core Libraries, da die gesamte Kommunikation über JSON erfolgt:

* Gestaltung und Verwaltung von JSON-Objekten und -Strings
* Validierung der Datenstruktur
* Sicherheitsmechanismen zur Gewährleistung valider Daten

**1.5 logManager**

Ermöglicht das kontinuierliche Logging auf der SD-Karte im Ethernet-Shield:

* Speicherung strukturierter Logdaten
* Optimierungen zur Umgehung bekannter SPI-Probleme
* Interaktion mit dem serialMenu zur Protokollierung serieller Ausgaben

*Hinweis: [Hier können zukünftig Links oder Referenzen eingefügt werden]*

**1.6 serialMenu**

Bietet sichere und strukturierte Kommunikation über die serielle Schnittstelle:

* Thread-Sicherheit durch Mutexes (FreeRTOS)
* Steuerung der Ausgabe über boolesche Flags
* Möglichkeit zur direkten Speicherung von Nachrichten im LogManager

**1.7 reportSystem**

Umfasst zentrale Funktionen zur Systemüberwachung:

* Stack Guard und Stackoverflow-Detection
* Speicherüberwachung (RAM, EEPROM)
* Kontrolle aller Kommunikationswege
* Threshold-Kontrolle, Stackdump-Funktionalität
* Fehlerpersistenz im EEPROM

**1.8 timeModule**

Verwaltet die Systemzeit auf dem Arduino MEGA2560:

* Abruf der aktuellen UTC-Zeit via Ethernet (als JSON)
* Lokale Speicherung in einer DateTimeStruct
* Eigenständige Inkrementierung und Formatierung
* Nutzung in allen gesendeten JSON-Daten

**1.9 ptrUtils**

Eine Header-only-Library zur Verwaltung von Pointern:

* Überprüfung und Validierung
* Speicher- und Zugriffskontrolle

**1.10 vacControl**

*Beschreibung folgt.*

**1.11 flyBack**

*Beschreibung folgt.*

**2. Externe Libraries**

Folgende Libraries wurden importiert, um Funktionalität, Stabilität und Kompatibilität zu erweitern:

* **ArduinoJson** – Verarbeitung von JSON-Daten
* **Arduino STL** – Nutzung von Standard-Template-Elementen (z. B. vector, string)
* **SD** – Zugriff auf die SD-Karte
* **SPI** – Kommunikation mit SPI-Geräten
* **Ethernet** – Netzwerkkommunikation
* **ErriezMemoryUsage** – Überwachung des Speichers
* **FreeRTOS** – Betriebssystem für Multitasking, Zeitsynchronisierung etc.
* **Frt.h** – Objektorientierter Wrapper für FreeRTOS

**3. Entscheidungsgrundlage & Architekturprinzipien**

Die Entscheidung für C++ basiert auf den technischen Gegebenheiten des Arduino MEGA2560 und der nativen Unterstützung von .ino-Dateien. Das objektorientierte Design erleichtert:

* Die modulare Erweiterbarkeit
* Die klare Trennung der Funktionalitäten
* Die langfristige Wartung und Dokumentation

**4. Erweiterbarkeit**

Diese Struktur ist so konzipiert, dass sie leicht ergänzt und gepflegt werden kann:

* Neue Module können analog zu den bestehenden Sections hinzugefügt werden
* Platzhalter für vacControl und flyBack ermöglichen spätere Ergänzungen
* Alle Module können mit Quellcode-Links, Diagrammen oder Doxygen-Kommentaren ergänzt werden

Beschreibung der Anwendung der Libraries:

Ich habe mich für C++ entschieden, da wir aufgrund des Arduino MEGA2560 bereits mit C++/.ino arbeiten. Objektorientierter Ansatz beim Aufbau.

Das **comModule** bietet Zugriff auf die verschiedenen Schnittstellen auf unserem System, vor allem das Ethernet-Modul, mit dem wir als eSW mit dem HAS (Hardware Access Service) und dem VAT µC kommunizieren. Diese Library wird verwendet, um Daten zu senden und zu empfangen sowie Konfigurationen zu versenden oder zu erhalten. Durch das **comModule** arbeiten wir auch mit dem I2C-Modul, was besonders wichtig für die Auswertung der Daten aus dem Temperatursensor ist. **SPI** ist sowohl für den **logManager** als auch das Ethernet-Shield wichtig, da wir dort die Teilnehmer tracken und steuern, wer wann senden darf.

Das **calcModule** bietet verschiedenste Funktionen, um wichtige Werte der Vakuumkammer oder Daten von Sensoren umzurechnen, zu finalisieren, zu parsen oder zu bearbeiten und umzuwandeln.

Das **sensModule** bietet Zugriff auf die Libraries **Temperature** und **Pressure**, welche im **sensModule** überwacht und geprüft werden. Des Weiteren dient die Abstraktion dazu, ein einfaches Interface zu schaffen und schnell neue Sensoren einzubinden, sollte dies notwendig sein.

Das **jsonModule** ist ein integraler Bestandteil der Core-Libraries, da wir das JSON-Format verwenden, um Daten zu versenden oder zu empfangen. Diese Library übernimmt die wichtigsten Grundaufgaben in der Gestaltung der JSON-Strings/Objekte und hat selbst einige Sicherheitsmechanismen, um valide JSONs zu garantieren.

Der **logManager** bietet die Möglichkeit, Logfiles zu erstellen und diese kontinuierlich auf der verwendeten SD-Karte im Ethernet-Shield zu speichern. Diese Library beinhaltet einige Tweaks, um Probleme mit dem SPI-Bus zu umgehen und einen guten Workflow zu garantieren.

[HIER LINKS EINFÜGEN]  
Des Weiteren sind einige Funktionen des **logManager** Bestandteil des **serialMenus**, was es uns erlaubt, Serial-Messages direkt in einem strukturierten Format abzuspeichern.

Das **serialMenu** bietet einige Funktionen, um sicheres Drucken via Serial zu garantieren. Hier haben wir Wert auf Sicherheit gelegt; die Funktionen verwenden Mutexes, um Probleme beim Drucken mit mehreren Tasks innerhalb des RTOS zu verhindern. Des Weiteren erfolgt die Speicherung der Messages durch das Setzen eines Boolean-Flags, um den **logManager** zu aktivieren.

Das **reportSystem** hat verschiedene Funktionen, um die Integrität des Systems zu gewährleisten, vom **StackGuard** und **StackOverflow**-Detection bis hin zur Speicherüberwachung von allen relevanten Speichern. Des Weiteren werden alle Kommunikationswege überwacht und kontrolliert sowie alle anderen Sensoren. Es erfolgt eine Kontrolle von Thresholds, das Abrufen von Stack-Dumps und das Speichern von Fehlern im EEPROM.

Das **timeModule** verwaltet die Zeit auf dem Arduino MEGA2560. Im Setup holen wir uns mit dem Ethernet-Modul den JSON-String mit der aktuellen Zeit in UTC. Danach speichern wir diesen lokal auf dem Arduino in einem DateTime-Struct, inkrementieren die Zeit selbstständig, formatieren diese und verwenden sie dann in unseren erstellten JSON-Strings, die wir auch versenden.

Die **ptrUtils** (Pointer Utilities) ist eine Header-only-Library, die wir verwenden, um Pointer zu überprüfen, zu überwachen und zu managen.

Flyback -> // NOCH SCHREIBEN

vacControl -> // NOCH SCHREIBEN

**Hardwareübersicht FFRESW**

**Arduino MEGA 2560 R3**

Für das Projekt wurde der **Arduino MEGA 2560 R3** gewählt. Dieser Mikrocontroller zeichnet sich durch eine hohe Anzahl an digitalen und analogen Ports aus, die eine flexible Anbindung verschiedener Peripheriegeräte ermöglichen. Mit 54 digitalen I/O-Pins, 16 analogen Eingängen und 4 UARTs bietet der MEGA 2560 eine exzellente Basis für die Erweiterung des Systems und den Anschluss zahlreicher Sensoren und Aktoren. Die Leistungsfähigkeit dieses Mikrocontrollers ist für den Preis hervorragend und stellt sicher, dass auch komplexe Aufgaben in Echtzeit verarbeitet werden können.

Ein weiterer großer Vorteil des **Arduino MEGA 2560 R3** ist die umfangreiche Unterstützung durch die Arduino-Community. Es gibt unzählige Tutorials, Foren und Libraries, die eine schnelle Implementierung und Fehlerbehebung ermöglichen. Besonders wichtig war uns, dass jeder im Team die Embedded Software (eSW) verstehen und bei Bedarf auch selbstständig erweitern oder anpassen kann. Für Programmierer mit weniger Erfahrung bietet die breite Zugänglichkeit des Systems eine ideale Grundlage. Im Vergleich zu anderen Mikrocontrollern, wie beispielsweise den ST Mikrocontrollern, die ebenfalls in Betracht gezogen wurden, hat der Arduino MEGA 2560 den Vorteil einer niedrigeren Einstiegshürde und einer weitaus größeren Entwicklergemeinschaft.

**Ethernetshield W5100 von AZ Delivery**

Für die Netzwerkkommunikation wurde das **Ethernetshield W5100** von **AZ Delivery** verwendet. Das Shield nutzt den **W5100 Ethernet-Chip**, der eine zuverlässige und leistungsstarke Möglichkeit bietet, das Arduino-Board mit einem Netzwerk zu verbinden. Besonders hervorzuheben ist die einfache Integration und Handhabung des Ethernet-Shields, das mit einer Standardbibliothek von Arduino direkt kompatibel ist. Dies erleichterte die Implementierung der Netzwerkkommunikation erheblich.

Das **Ethernetshield W5100** verfügt über einen eingebauten **MicroSD-Kartenleser**, der für die Speicherung von Logdateien und anderen Daten verwendet werden kann. Diese Funktion war für das Projekt besonders relevant, da wir eine Lösung benötigten, die sowohl die Kommunikation über Ethernet als auch die Speicherung von Daten auf einer SD-Karte ermöglicht. Das Shield unterstützt TCP/IP-Verbindungen und ermöglicht die schnelle Übertragung von Daten zwischen dem Arduino-Board und einem externen Server oder anderen Geräten im Netzwerk.

Ein weiterer Vorteil des **W5100 Chips** ist, dass er im Vergleich zu anderen Ethernet-Modulen eine höhere Stabilität und geringere Latenz bietet, was für unsere Anforderungen an die Netzwerkkommunikation von entscheidender Bedeutung war.

**Zusammenfassung der Entscheidung:**

Die Wahl des **Arduino MEGA 2560 R3** und des **Ethernetshield W5100** von **AZ Delivery** wurde maßgeblich durch die folgenden Faktoren beeinflusst:

* **Arduino MEGA 2560 R3**:
  + Große Anzahl an digitalen und analogen I/O-Pins
  + Kostengünstig und leistungsstark für das Preis-Leistungs-Verhältnis
  + Umfangreiche Arduino-Community und Libraries
  + Einfache Implementierung für Entwickler mit unterschiedlichem Erfahrungsstand
* **Ethernetshield W5100 von AZ Delivery**:
  + Integrierter W5100 Ethernet-Chip für stabile Netzwerkverbindungen
  + MicroSD-Kartenleser für Datenspeicherung
  + Einfache Integration mit bestehenden Arduino-Libraries

Durch diese Wahl konnten wir die Hardwareanforderungen effizient abdecken und gleichzeitig eine Entwicklungsumgebung wählen, die für das gesamte Team zugänglich und gut dokumentiert ist.

**Limitationen durch Hardware und bekannte SW Bottlenecks**

Durch die Verwendung des **Arduino MEGA2560** ergeben sich bestimmte Einschränkungen, insbesondere in der Kommunikation mit dem **VAT uC** zur Steuerung der Ventile. Die API bietet Endpunkte mit Getter- und Setter-Funktionen, um Werte zu senden oder abzurufen. Aufgrund von Hardware- und Softwarearchitektur-Limitationen war es jedoch nicht möglich, kontinuierliche Datenströme von den Sensoren des **VAT uC** bereitzustellen. Stattdessen arbeiten wir im **FFRESW**-Projekt über das **EthernetModule** mit den **Compound1**, **Compound2** und **Compound3**-Befehlen, was unsere Möglichkeiten in dieser Hinsicht etwas einschränkt.

Durch die Verbindung des **VAT uC** via Ethernet über Port 503 kann jedoch der **Hardware Access Service (HAS)** konstant Daten streamen. Diese Daten können dann in einer Datenbank gespeichert und über **Grafana** visualisiert werden. Dieser Datenstrom erfolgt allerdings nicht über den **Arduino MEGA2560**, sondern direkt über den **VAT uC**.

Zusätzlich ergeben sich durch die Verwendung von **FreeRTOS** und dem OOP-Wraper **frt.h** Speicher- und **SRAM**-Limitationen. Um möglichen Problemen aufgrund dieser Begrenzungen entgegenzuwirken, haben wir einen **StackMonitorTask** implementiert. Dieser überwacht die anderen Tasks und verhindert so potenzielle **Stackoverflow**-Fehler.

Des Weiteren habe ich mir die Tools der GNU Compiler Collection zu nutzen gemacht und diese verwendet, um etwaige große Speicherallokationen in dem ELF Binary festzustellen. Danach habe ich diese dann behoben/entfernt/optimiert.

**FlowDiagramme/DoxyGen**

Für die Flowchars bzw Diagramme wurde das OpenSource Projekt Sourcetrail verwendet. Mit diesem Tool kann man C/C++ Projekte (und andere Sprachen) relativ einfach Indexieren, Scannen, Parsen und verarbeiten. Das Tool ermöglichte es uns schlussendlich auch Fehler bzw schlechte Designs zu finden da wir damit interaktiv den Flow des Programmes analysieren können. Des Weiteren haben wir uns dazu entschieden die hier generierten Diagramme zu verwenden, da diese Qualitativ Hochwertig sind und einfach anzupassen.

Leider Original wurde geclosed aber andere maintainer derzeit:

Neuer Fork -> <https://github.com/petermost/Sourcetrail>

Original -> <https://github.com/CoatiSoftware/Sourcetrail>

**Erklärung Zusammenspiel HAS, ESW und VAT**

Mittels Ethernets sind der HardwareAccessService kurz HAS (also Raspberry Pi), die Embedded Software kurz eSW (also Arduino Mega2560) und der VAT (uC der Firma VAT) verbunden. Datenaustausch findet mittels TCP/IP Frames statt und die Nachrichten sind in JSON formatiert. Diese JSON-Strings/Objekte werden dann auch in der Datenbank (MongoDB) von dem HAS gespeichert. Da der VAT uC ein anderes Datenformat erwaretet, sendet die eSW das erwartet Format an den VAT uC und wandelt die Antworten dann passend um, sodass der HAS diese auch verstehen kann. Neben Ethernet werden auch SPI für das Ethernetshield, den MicroSD-Katenleser verwendet. Auch I2C kommt für die Temperatursensoren zum Einsatz. Der VAT uC verwendet EtherCAT um mit dem Drucksensor und der Valve zu kommunizieren.

Auf dem Rasperrypi läuft ein Dockercontainer mit Ubunutu und einigen Konfigurationen, diese Repo ist auf Github gehostet und der HAS ist ein Submodule von diesem Repo.

Auf dem Arduino Mega2560 läuft ein FreeRTOS mit einer OOP Library namens frt.h, die einige Dinge vereinfacht und optimiert.

Das Betriebssystem auf dem VAT uC ist zum jetzigen Zeitpunkt nicht bekannt. Muss noch herausfinden?????????

**Toolchain**

Da wir ein recht umfangreiches Embedded Software Projekt geplant hatten und sowohl die alte als auch die neue Arduino IDE nicht passen für unsere Anforderungen war, habe ich mich dazu entschlossen einen eigene Toolchain aufzustellen.

Als erstes bin ich hier auf die Programmino IDE gestoßen und habe diese getestet. Da diese einige Bugs hatte und auch ansonsten nicht gerade Optimal war (Kostentechnisch und Support) habe ich mich nicht für diese IDE entschieden.

<https://programino.com/>

Im zweiten Anlauf bin ich dann auf die Open Source IDE Sloeber gestoßen, welche eine kleine Community hatte und einige freiheiten bietet. Diese IDE baut auf Eclypse auf welche nicht Opmtimal ist aber im großen und ganzen wesentlich besser ist als die von Arduino.

<https://eclipse.baeyens.it/index.shtml>

Bei dem Compiler habe ich den avr-gcc\7.3.0-atmel3.6.1-arduino7 verwendet.