Implementierung einer digitalen Raumbelegungsanzeige aus Serienkomponenten

Abschlussarbeit

zur Erlangung des akademischen Grades

Master of Science (M.Sc.)

an der

Hochschule für Technik und Wirtschaft Berlin
Fachbereich IV: Informatik, Kommunikation und Wirtschaft
Studiengang Angewandte Informatik

Erstprüfer: Prof. Dr. Alexander Huhn
Zweitprüferin: Prof. Dr. Adrianna Alexander

Eingereicht von Jonas Rupert Fabian Klein – 561228

Dienstag, 4. Februar 2025

Einleitung

Durch die Digitalisierung erlebte der heutige Einzelhandel bisher einen bedeutenden Schub. Zu den technischen Innovationen zählen u. a. digitale Kassensysteme, Barcodes (Universal Product Code, UPC) zum Einlesen von Produkten, bargeldloses Bezahlen und elektronische Preisanzeigen (Electronic Shelf Label, ESL).

Letztere werden ereignisgesteuert durch ein zentrales Warenmanagement über Funk aktualisiert. Dies spart gegenüber der manuellen Preisauszeichnung erheblich Kosten ein. Die Preisschilder bestehen dabei aus einer stromsparenden Anzeige, wie einem e-Paper-Display, einem Microcontroller und einer Batterie zur Stromversorgung.

Auch in anderen Geschäftsbereichen werden digitale Schilder eingesetzt, um auf sich ändernde Begebenheiten zu reagieren. So werden zunehmend häufiger auch im Büroumfeld an Sitzungs- und Seminar-Räumen Anzeigen eingesetzt, die Raumreservierungen darstellen, sodass Mitarbeitende spontan sehen, wann welcher Raum frei ist. Auch hier sind die angezeigten Informationen öfter aktuell, bzw. müssen nicht manuell aktualisiert werden. Ein weiterer Vorteil ist, dass externe Personen, die auf etwaige Raumreservierungen keinen Zugriff haben, wissen, vor welchem Raum sie stehen.

Kurzbeschreibung

Die Hochschule für Technik und Wirtschaft Berlin nutzt mit ausgedruckten Blättern ein kostenintensives Verfahren zur Darstellung von Raumbelegungen an den Seminarräumen. Um dieses Problem zu lösen, wurde eine digitale Raumbelegungsanzeige implementiert, die automatisch aus einem Backend die Informationen zur Raumbelegung bezieht und anzeigt.

Die Anzeige besteht aus einem Microcontrroller, betrieben mit Micropython, einem ePaper-Display zur Anzeige von Belegungsinformationen, einem Funk-Modul zum Empfang von Aufweck-Signalen zum Auslösen der Aktualisierung und einer Batterie zum autarken Betrieb unter geringer Stromaufnahme. Der Microcontroller rendert die empfangenen Informationen und stellt diese auf dem ePaper-Display dar.

Das Backend liest aus dem hochschulinternen Raummanagement-Werkzeug alle benötigten Informationen aus und stellt diese in aufbereiteter Form über eine Message-Queue dem Frontend zur Verfügung. Es werden von dem Microcontrroller Signalstärkemessungen durchgeführt, um auf über Ortung auf den anzuzeigenden Raum zu schlussfolgern.

Abstract

The University of Applied Sciences for Engineering and Economics (HTW) Berlin uses a cost-intensive method of displaying room occupancy for the seminar rooms with printed sheets. To solve this problem, a digital room occupancy display was implemented that automatically obtains and displays room occupancy information from a backend.

The display consists of a microcontroller, powered by Micropython, an ePaper display to show occupancy information, a radio module to receive wake-up signals to trigger the update and a battery for self-sufficient operation with low power consumption. The microcontroller renders the received information and displays it on the ePaper display.

The backend reads all the required information from the university's internal room management tool and makes it available to the front end in processed form via a message queue. Signal strength measurements are carried out by the microcontroller in order to draw conclusions about the room to be displayed via localization.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

ADU/C....Analog-Digital-Umsetzer/Converter
AP....Access Point
API....Application Programming Interfaces
CSMA/CA....Carrier Sense Multiple Access / Collision Avoidance
eFuse....electronic Fuse
EPD....Electrophoretic Display
FOSS....Free and Open Source Software
Fraunhofer IIS....Fraunhofer-Institut für Integrierte Schaltungen
GPIN....General Purpose Input
GPIO....General Purpose Input/Output
HTML....Hypertext Markup Language
HTTP....Hypertext Transfer Protocol
IoT....Internet of Things
JS....Javascript
JSON....Javascript Object Notation
LBS....Location Based Services
LiPo....Lithium-Polymer
LoRa(WAN)....Long Range Wide Area Network
LUT....Lookup Table
MCU....Microcontroller Unit
MOSFET....Metal-Oxide-Semiconductor Field-Effect Transistor
MOSI....Master out, Slave in
MQ....Message Queue/Queuing
MQTT....MQ Telemetry Transport
OASIS....Organization for the Advancement of Structured Information Standards
OTF....OpenType-Font
QoS....Quality of Service
REPL....read-eval-print Loop
RSSI....Received Signal Strength Indicator
ROM....Read Only Memory
RTC....Real Time Clock

RTOS.....Real-time Operating System
(H/V)SPI.....Serial Peripheral Interface
TTF.....TrueType-Font
UART.....Universal Asynchronus Remitter Transceiver
ULP (coprocessor).....Ultra-Low-Power coprocessor
VLAN.....Virtual Local Area Network, Virtual LAN
WLAN.....Wireless Local Area Network, Wireless LAN
WOR..... Wake on Radio

1 Einführung

1.1 Hintergrund der Arbeit

Diese Arbeit wurde im Zeitraum vom 1. Oktober 2024 bis zum 4. Februar 2025 durchgeführt. Sie dient der Erlangung des akademischen Grades Master of Science (M.Sc.) des Studiengangs Angewandte Informatik an der Hochschule für Technik und Wirtschaft Berlin (HTW Berlin).

1.2 Problem- und Zielstellung

Die Hochschule für Technik und Wirtschaft Berlin nutzt zur Darstellung von Raumbelegungen (Raumplan) direkt an den Seminarräumen und Laboren jeweils ein Schild aus Papier, eingefasst hinter einer transparenten Plastik-Scheibe (vgl. Abb. 1). Sobald sich eine Raumbelegung ändert, sollte dieses Papier aktualisiert bzw. ausgetauscht werden. Raumbelegungen einzelner Räume können sich theoretisch ständig und kurzfristig ändern. Die Belegung weitgehend aller Räume ändert sich jedoch mindestens zweimal im Jahr – immer zum Start eines neuen Semesters.

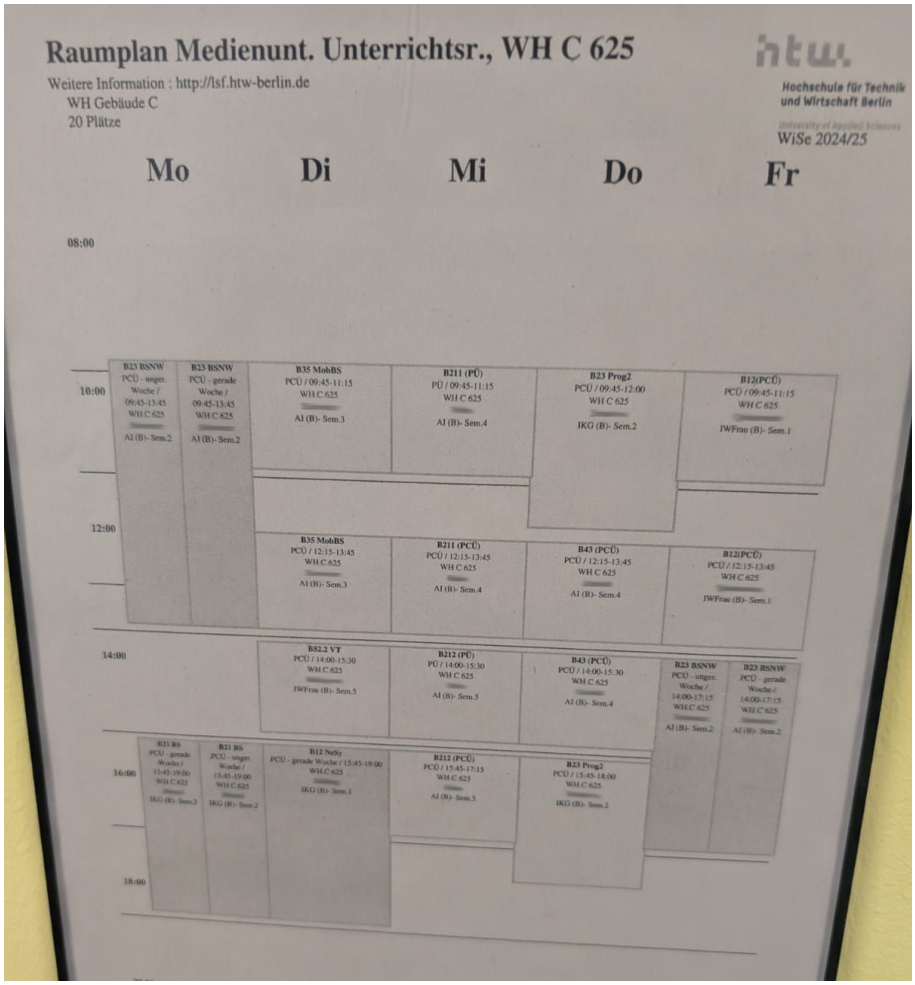


Abbildung 1: Aktueller Raumbelegungsplan der HTW Berlin des Raums WH-C-625

Das Austauschen dieser Raumschilder bindet personelle Ressourcen. Entsprechend wird das Austauschen vieler kleinerer Belegungsänderungen während des Semesters vernachlässigt.

Aufgabe dieser Arbeit ist es, eine digitale Raumbelegungsanzeige zu implementieren, die Belegungsänderungen automatisch empfängt und auf einem Display neben dem Raum anzeigt. Das neue System soll gegenüber der alten Lösung Vorteile bieten, wie:

- die Einsparung personeller Aufwendungen
- die frühzeitige Aktualisierung der Anzeige bei Belegungsänderungen

1.3 Aufbau der Arbeit

Zu Beginn werden im Kapitel *2 - Grundlagen* die Voraussetzungen vermittelt, die zum tieferen Verständnis der Arbeit erforderlich sind. Hier finden zu den verwendeten Chips, Technologien und Protokollen wie Microcontroller, EPDs und LoRaWAN nähere Erläuterungen statt.

Anschließend werden zu Kapitel *3 - Anforderungsanalyse* die Anforderungen näher beleuchtet.

In dem Kapitel *4 - Konzeption und Entwurf* werden der geplante Aufbau und die vorgesehene Kommunikations-Systematik erläutert. Hier werden auch verschiedene Optionen bzgl. des Informationsabrufs und der zu verwendenden Hardwarekomponenten diskutiert.

Kapitel *5 - Implementierung* erläutert die Implementierung des zuvor entworfenen Konzepts und die Entwicklung eines physischen Prototypen. Hier wird u. a. erklärt, wie die Komponenten physisch miteinander verbunden sind und logisch miteinander kommunizieren. Es wird insb. darauf eingegangen, wie das Rendering der Belegungsanzeige, aber auch der Abruf von Belegungsinformationen zu Räumen abläuft. Ein weiterer Gegenstand dieses Kapitels ist die Möglichkeit, die entwickelte Firmware über Funk aktualisieren zu können.

Daraufhin wird in Kapitel *6 - Validierung* der Stromverbrauch analysiert und geprüft, inwiefern die gesetzten Anforderungen erreicht wurden.

Zum Schluss werden in Kapitel *7 - Fazit* die Ergebnisse dieser Arbeit zusammengetragen und die Limitationen des entstandenen Systems näher erläutert. Zu jedem Problem werden dazu mögliche Lösungsansätze beschrieben, mit denen sich zukünftige Arbeiten befassen können.

2 Grundlagen

2.1 ESP32

ESP32 stellt eine Familie aus Microcontrollern der Firma Espressif Systems Co., Ltd. dar. Diese Microcontroller besitzen u. a. zwei CPU-Kerne mit einer Taktrate von je 240 MHz, einen Wi-Fi- und Bluetooth-Chip, eine Real Time Clock (RTC) zur genauen Zeitmessung, mehrere Analog-Digital-Umsetzer (ADU), Serial Peripheral Interfaces (SPI) und Universal Asynchronus Receiver Transmitter (UART)-Schnittstellen (vgl. Abb. 2). Aufgrund seiner Vielzahl an Komponenten eignet sich die Microcontroller Unit (MCU) ESP32 hervorragend zum vielseitigen Einsatz und der Entwicklung von Prototypen. [1]

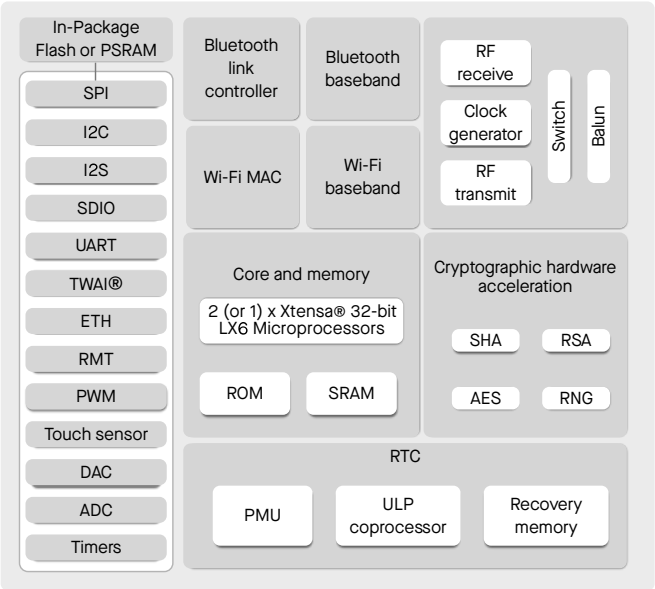


Abbildung 2: ESP32 Block Diagramm, [1]

Der ESP32 kann in unterschiedlichen „Power-Modes“ betrieben werden, in denen einige Komponenten des Chips deaktiviert sind. Im „Active“-Modus nimmt der Chip 240 mA Strom auf, während alle Komponenten aktiv sind und das Funk-Modul mit voller Sendeleistung abstrahlt. Im „Deep-sleep“-Modus bei inaktivem Ultra-Low-Power coprocessor (ULP) liegt die Stromaufnahme noch bei annähernd 10 µA. Hinzu kommen Verluste durch etwaige Spannungsregelungen, um die benötigte Betriebsspannung von 3,3 V bereitzustellen.

MCUs benötigen i.d.R. eine geregelte Betriebsspannung von 5 V oder 3,3 V. Diese Spannung steht nur in seltenen Fällen zur Verfügung und muss daher oft durch eine Spannungsregelung erreicht werden.

Der ESP32 ist für Lernzwecke und für das Prototyping auf Entwicklungsboards erhältlich. Diese Boards haben hauptsächlich eine Spannungsregelung, sodass diese z. B. über USB oder eine externe Stromversorgung betrieben werden können, und einen USB-Controller, sodass diese über USB programmiert werden können.

2.2 Micropython auf dem ESP32

Micropython ist ein Betriebssystem bzw. eine Anwendung für Microcontroller, um CPython-ähnliche Programmierung auf Microcontrollern zu ermöglichen. Die Syntax der Programmierung ist zu CPython identisch, jedoch können aufgrund fehlender Bibliotheken die meisten CPython-Programme nicht oder nicht unverändert ausgeführt werden.

Mit Micropython wird auch die für Python typische read-eval-print loop (REPL) über die serielle Konsole zur Verfügung gestellt, über die zeilenweise mit Python programmiert werden kann. Auf dem ESP32 wird auch eine sog. WebREPL zur Verfügung gestellt, die es erlaubt, über den WLAN-Stack des ESP32 durch einen Webbrowser auf der REPL zu programmieren und Dateien zu modifizieren.

Im Gegensatz zur Programmiersprache C wird durch das hohe Abstraktionsniveau von Python die Programmierung stark vereinfacht. Micropython stellt den Read Only Memory (ROM) des ESP32 als Dateisystem zur Verfügung. Micropython-Programme werden dadurch unkompiliert auf den ROM des ESP32 kopiert und können direkt interpretiert und ausgeführt werden. Es werden dabei nacheinander die Dateien boot.py und anschließend main.py im Root-Verzeichnis eingelesen und ausgeführt.

Mit der Abstraktion gehen jedoch auch einige Nachteile einher. Da Micropython erst zur Ausführung interpretiert und in Prozessorbefehle übersetzt wird, ist es wesentlich weniger performant als C oder andere Sprachen wie Rust oder TinyGo [2]. Es bestehen oft nur unzureichende Möglichkeiten, in die Heap-Verwaltung einzugreifen, um Programmabläufe zu optimieren. Auch ist z. B. der Hibernate-Modus des ESP32 nicht benutzbar.

Auf dem ESP32 wird Micropython als Anwendung neben dem Betriebssystem FreeRTOS[1] installiert. Das Betriebssystem nimmt hierbei die üblichen Aufgaben wie Multitasking oder die Steuerung von Ressourcenzugriffen wahr.

2.3 EPD

Electrophoretic Displays (EPD)[2] sind bistabile Anzeigen, die auf der Basis von reflektiertem Umgebungslicht funktionieren. Elektrophorese bezeichnet hierbei die Technologie, bei der positiv wie negativ geladene, eingefärbte Partikel in einer elektrisch nicht leitfähigen, viskosen[3] Substanz wie Öl sich zur jeweils gegensätzlich geladenen Elektrode bewegen (vgl. Abb. 3). Durch die Viskosität der Substanz können die Partikel über eine längere Zeit an ihrer Position verweilen, ohne dass diese Anzeigen mit Strom versorgt werden müssen. [3], [4]

Initial unterschiedliche Positionen der Partikel im Fluid führen zu inhomogenen Schwarzwerten auf dem Display. Dieses Problem wird dadurch adressiert, indem das Display zur Anzeige mehrere Male seine optischen Extrema wechselt und dadurch die Partikel neu anordnet. [5]

1 Ein Echtzeitbetriebssystem für Microcontroller
2 Auch unter der Bezeichnung »electronic Paper« oder »ePaper« bekannt
3 Die Viskosität gibt den Grad der Zähflüssigkeit einer Substanz an.

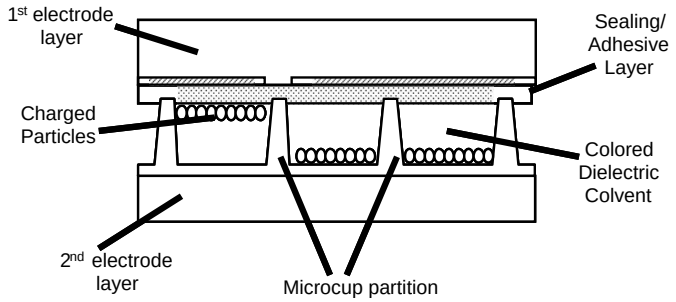


Abbildung 3: Schematische Darstellung des Querschnitts eines Microcup EPD, [1], [4]

Diese Technologie erlaubt es, im Microcontroller-Umfeld Anzeigen mit sehr geringem Energiebedarf bei sich selten ändernden Inhalten zu implementieren.

Zusätzlich ist es möglich, partielle Aktualisierungen (partial updates) des Displays durchzuführen, was – durch eine geringere Datenmenge bei der Übertragung und Verarbeitung – zu einem geringeren Stromverbrauch führt.

Das Display selbst benötigt eine vergleichsweise hohe Spannung (etwa 15 - 20 V) bei niedrigem Strom, um neue Inhalte darzustellen. Da EPDs oft in Anwendungen mit geringem Energiebedarf und geringen Stromspannungen zum Einsatz kommen, implementieren diese Displays oft eine Ladungspumpe, die elektrische Spannungen vergrößern kann. [6]

Die Aktualisierung des Bildes geschieht grundsätzlich in drei Phasen:

Phase 1: Das alte Bild wird gelöscht. Dabei wandern die schwarzen Partikel in den Hintergrund. Im alten Bild sind die Positionen der geladenen Partikel zueinander unterschiedlich, daher entsteht ein Geisterbild (Ghosting). Es wird angenommen, dass das neu anzuzeigende Bild nicht sehr unterschiedlich zum alten Bild ist. So wird im zweiten Schritt das Inverse des Bildes angezeigt, um dem Geisterbild entgegen zu wirken.

Phase 2: Je länger Partikel an ihrer Position verweilen, desto stärker nimmt ihre Mobilität ab. Dies führt auch zu Geisterbildern. Um dem entgegenzuwirken, werden die Pixel mehrmals in ihre optischen Extrema versetzt. Das ist der Moment, bei dem das Display flackert.

Phase 3: Das Zielbild wird am Ende der Puls-Sequenz erzeugt. Dabei werden verschiedene Spannungen unterschiedlich lang auf die einzelnen Elektroden der Pixel angewandt.

Diese Phasen werden durch verschiedene zeitliche Spannungsabläufe in Wellenformen beschrieben und in Umsetzungstabellen (Lookup Table, LUT) im Controller des EPD festgehalten. Durch die Modifikation dieser LUT lassen sich die einzelnen Phasen beeinflussen. [7], [8]

2.4 MQTT

MQTT (ursprüngliche MQ Telemetry Transport) ist ein Message-Queuing (MQ)-Protokoll, welches sich aufgrund seiner Leichtgewichtigkeit im IoT-Bereich besonders eignet. MQTT wird von der

Organization for the Advancement of Structured Information Standards (OASIS) entwickelt und standardisiert. Dieses Protokoll baut auf TCP als Client-Server-Anwendung auf. Teilnehmende Endpunkte (Clients) veröffentlichen dabei Informationen unter bestimmten Themen (sog. Topics) oder abonnieren diese. Hat ein Client ein Thema abonniert, bekommt dieser Client die veröffentlichten Inhalte eines anderen Clients sofort mitgeteilt. [9]

Empfängt ein Client eine Nachricht nicht, weil er nicht mit dem Server verbunden war oder das notwendige Topic nicht abonniert hat, erhält dieser Client rückwirkend keine Nachrichten. Um diesem Problem zu begegnen, können Nachrichten mit einem Retain-Flag versehen werden. Das hat zur Folge, dass der Message-Broker (Server) die entsprechenden Nachrichten solange vorhält, bis diese durch andere Nachrichten unter gleichem Topic überschrieben werden. [9]

Auf einem unzuverlässigen Übertragungskanal ist das sichere Erkennen von Informationsverlust theoretisch unmöglich. Der Erhalt einer Nachricht oder einer Bestätigung muss prinzipiell unendlich oft bestätigt werden, um einen Informationsverlust auszuschließen. Bei dem TCP-Verbindungsaufbau hingegen teilen beide Teilnehmer dem jeweils anderen ihre Sequenznummer mit. Diese Sequenznummer wird jeweils dem anderen Teilnehmer einmalig bestätigt. Die Bestätigung der zweiten Sequenznummer kann semantisch auch als Bestätigung der ersten Bestätigung angesehen werden, was dazu führt, dass beide Teilnehmer wissen, dass beide Richtungen der Kommunikation (Client>Server und Server>Client) funktionieren; dies ist nach dem dritten Weg der Fall (Drei-Wege-Handschlag). [10]

Über die Bestätigung in TCP hinaus implementiert MQTT ein Quality-of-Service-Feature (QoS), mit dem der Erhalt von Nachrichten zusätzlich bestätigt und ggf. diese erneut versendet werden können. In der niedrigsten Stufe „At most once delivery“ wird weder der Erhalt bestätigt noch erneut versucht, die Nachricht zuzustellen. Zur Stufe „At least once devlivery“ muss der Empfänger gegenüber dem Sender den Erhalt einer Nachricht bestätigen. Erhält der Sender keine Bestätigung, muss die Nachricht erneut gesendet werden. Zur höchsten Stufe „Exactly once delivery“ wird die darunterliegende Stufe um zwei Wege erweitert. Der Erhalt der Bestätigung des Empfangs der ersten Nachricht muss von dem Sender der ursprünglichen Nachricht erneut bestätigt werden. So sendet dieser eine weitere Bestätigung an den Empfänger und dieser sendet die Bestätigung der Bestätigung zurück an den ursprünglichen Sender. [9]

2.5 Node-RED

Node-RED ist ein Werkzeug der OpenJS Foundation zum Erstellen von Ereignis-getriebenen Anwendungen im Bereich des IoT anhand eines Baukastens nach dem Schema von Kontrollflussgraphen (vgl. Abb. 4).

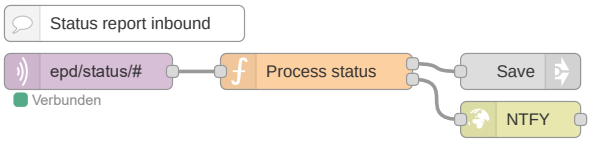


Abbildung 4: Eingehender Status-Bericht, Node-RED-Flow

Der in Abb. 4 dargestellte Kontrollfluss startet mit einem eingehenden Status-Bericht eines Clients über einen MQTT-Broker. Dieser Statusbericht wird verarbeitet und über den „Save“-Node (Knoten des Kontrollflussgraphen) in einem Unter-Fluss zur Speicherung weiterverarbeitet. Erfüllt in diesem Bericht eine bestimmte Eigenschaft eine Bedingung, wird ein HTTP-Request (Node „NTFY“) an einen NTFY-Dienst[4] übermittelt, mit dem eine Benachrichtigung an eine anwendende Person erfolgt. Die Nodes übergeben dabei zu jedem Übergang ein Nachrichten-Objekt, das eingelesen und modifiziert weitergegeben werden kann.

Zu den in der Standardausführung verfügbaren Nodes zählen u. a.:

- Ereignis-Empfänger, wie einen Timer, ein Knopf, eingehende HTTP- oder MQTT-Requests, die einen Kontrollfluss starten;
- Funktionen, um das Nachrichtenobjekt mit Javascript detaillierter zu modifizieren;
- Nodes für den Dateizugriff, um Dateien zu lesen und zu schreiben;
- Sequenz-Nodes, die den Kontrollfluss steuern, aufteilen oder zusammenführen.

2.6 Lokalisierung durch WLAN-Fingerprinting

Da viele Gebäude flächendeckend mit stationären WLAN-APs ausgestattet sind, besteht somit oft die Möglichkeit, mithilfe dieser APs in der Rolle von Funkbaken systematisch Positionsbestimmungen durchzuführen. Dies passiert gewöhnlich nach dem Verfahren der Trilateration (Entfernungsmessung zu Funkbaken), selten auch nach dem Verfahren der Triangulation (Winkelmessung zu Funkbaken).

Zur Entfernungsmessung kommt das Abstandsgesetz zum Tragen: Näherungsweise nimmt die Strahlungsintensität (die Leistung pro Fläche) des Signals mit zunehmender Entfernung zum Sender quadratisch ab. Die Signalstärke wird durch den Parameter Received Signal Strength Indication (RSSI) abgebildet.

Sind die Positionen dieser APs bekannt oder wurden RSSI-Messungen (sog. RSSI Fingerprints) an verschiedenen Punkten zuvor durchgeführt, kann durch eine Messung aktueller RSSI-Werte und dem Vergleich dieser zu bereits gemessenen Werte auf die derzeitige Position geschlussfolgert werden.

Ein Projekt im Fachbereich IV an der HTW Berlin hat sich bereits mit diesem Thema befasst. Hier wurde ein über das Web erreichbarer Dienst entwickelt, der an der HTW Berlin anhand der dortigen WLAN-APs zu gemessenen RSSI-Werten eine Vorhersage des zugehörigen Raumes abgibt.

4 NTFY ist eine FOSS-Anwendung, die auf Desktops oder Mobilgeräte Push-Benachrichtigungen ausliefern kann

3 Anforderungsanalyse

Das Projekt wurde im Umfeld des Mobile-Labs des Fachbereichs IV umgesetzt. Hier werden grundsätzlich alle Projekte in dem Umfeld von Microcontrollern auf Basis des ESP32 als MCU in Verbindung mit Micropython als Programmiersprache implementiert. So sollte die Belegungsanzeige ebenfalls auf einen ESP32 mit Micropython basieren, um das neue System in das gegenwärtige Umfeld zu integrieren. Des weiteren stand zur Anzeige ein EPD mit der Auflösung von 800×480 Pixel bei einer Bildschirmdiagonalen von 7,5 " zur Verfügung.

Dabei sollten folgende funktionale wie nicht-funktionale Anforderungen berücksichtigt werden:

- Das neue System soll gegenüber der alten Lösung weniger personelle Ressourcen binden. So soll die neue Anzeige einmalig installiert werden und anschließend über einen längeren Zeitraum autark Änderungen anzeigen.
- Die Stromversorgung soll für einen möglichst großen Zeitraum ausreichen.
- Das System soll durch Location Fingerprinting selbstständig erfassen, zu welchem Raum die jeweilige Anzeige gehört.
- Die Installation bzw. der Austausch bei Defekt oder bei ausgefallener Stromversorgung soll möglichst einfach vonstattengehen.
- Das System soll frühzeitig auf Änderungen in der Raumbelegung reagieren, sodass die angezeigten Informationen nur kurzzeitig inaktuell sind.
- Das System soll über eine Möglichkeit der Administration verfügen, über die die Anzeige konfiguriert und aktualisiert werden kann.

4 Konzeption und Entwurf

4.1 Grundlegendes Konzept

Das System sollte aus zwei Komponenten bestehen: Zum einen das Teilsystem, das neben den Eingängen der Räume platziert wird und die Belegungen anzeigt (Frontend), zum anderen das Teilsystem, das mit dem Frontend kommuniziert, es verwaltet und etwaige Informationen zur Administration vorhält (Backend).

Das Frontend sollte aus einem EPD mit der Auflösung von 800×480 Pixel zur Anzeige von Informationen, einem ESP32 als MCU und einer Batterie zur Stromversorgung bestehen. Das Backend sollte einen MQTT-Message-Broker und einen Node-RED-Server, die über einen WLAN-AP von der MCU aus dem Frontend erreichbar sind, umfassen.

Da das Frontend batteriebetrieben sein soll, sollte es sich bzw. seine Komponenten die längste Zeit in Ruhe befinden. In diesem Zustand spielt der Stromverbrauch der einzelnen Komponenten eine entscheidende Rolle.

Abruf von Belegungsinformationen und Anzeige

Die HTW Berlin verwendet zum Raumplanmanagement das Produkt „LSF“ der HIS Hochschul-Informations-System eG. Hierüber rufen Dozierende wie Studierende die Belegungsinformationen zu den Räumen der HTW über einen Webbrowser ab. Es wird keine öffentliche, dokumentierte API des LSF zum programmatischen Abruf von Raumplaninformationen angeboten. Somit müssen die benötigten Informationen durch einen HTML-Parser gewonnen werden, der die entsprechenden Raumbelegungen extrahiert. Dieses Parsing würde von dem Node-RED-Server im Backend durchgeführt werden.

Der Raumplan (wie er aus dem LSF abrufbar ist), enthält die Informationen, welche Veranstaltung zu welchem Zeitpunkt an welchem Tag der jeweiligen Woche stattfindet (vgl. Abb. 5). Eine ähnliche Ansicht wurde auch für die neue Belegungsanzeige vorgesehen. Zusätzlich sollte hier auch eine Dozierenden-Information angezeigt werden (wie es bei den alten Raumplänen der Fall war), jedoch war Platz der begrenzende Faktor. Somit wurde sich auf den Veranstaltungsnamen sowie die Uhrzeit pro Veranstaltung beschränkt (vgl. Abb. 6).

Zeit	Montag 06.01.2025	Dienstag 07.01.2025	Mittwoch 08.01.2025	Donnerstag 09.01.2025	Freitag 10.01.2025
vor 8					
8					
9					
10	B23 Betriebssysteme und Netzwerke (PCÜ)	B35 Mobile Betriebssysteme und Netzwerke (PCÜ)	B211 Drahtlose Netzwerke (PÜ)	B23 Programmierung 2 (PCÜ)	
11					
12		B35 Mobile Betriebssysteme und Netzwerke (PCÜ)	B211 Drahtlose Netzwerke (PCÜ)	B43 Verteilte Systeme (PCÜ)	
13					
14		B52.2 Verteilte Systeme (PCÜ)	B212 Internet of Things (IoT) (PÜ)	B43 Verteilte Systeme (PCÜ)	B23 Betriebssysteme und Netzwerke (PCÜ)
15					
16	B21 Betriebssysteme (PCÜ)		B212 Internet of Things (IoT) (PCÜ)	B23 Programmierung 2 (PCÜ)	
17					
18					
19					
ab 20					

Abbildung 5: Raumbelegungsplan des LSF für Raum WH C 625

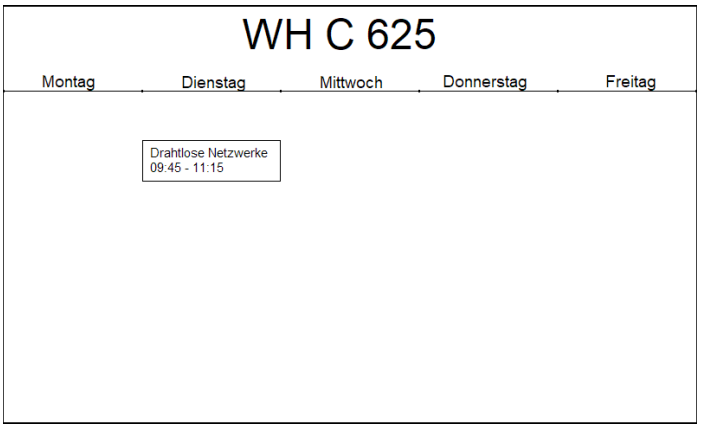


Abbildung 6: Konzept der Anzeige des neuen Systems

Das Auslösen der Aktualisierung der Anzeige

Sobald eine Raumbelegung geändert wird, sind die angezeigten Informationen im Frontend möglicherweise veraltet. Zur Aktualisierung der Anzeige soll die MCU sich mit einem WLAN-AP verbinden und das Backend nach Änderungen fragen. Sind Änderungen verfügbar, werden diese an die MCU zurückgespielt. Anschließend wird von der MCU das entsprechende Bild gerendert und angezeigt. Damit der Prozess zur Aktualisierung der Anzeige angestoßen werden kann, benötigt es ein Ereignis (Event), auf das reagiert werden kann.

Mit einem internen Event wie dem Ticken eines Timers könnte das Frontend in zeitlich regelmäßigen Abständen nach ausstehenden Änderungen bei dem Backend anfragen (Polling). Da jede Anfrage mit einem erhöhten Energieverbrauch einhergeht (aktiver WLAN-Stack), benötigt es hier eine Abwägung zwischen der Aktualität der Anzeige und der Batterielaufzeit.

Aktuellere Informationen könnten angezeigt werden, indem die Verbindung zum Backend gehalten wird. Das Backend löst dabei selbst den Prozess der Aktualisierung der Anzeige am Frontend aus (Interrupt). Das führt jedoch zu einem hohen Stromverbrauch, da der WLAN-Stack hierbei aktiv gehalten wird.

Alternativ kann durch ein anderes externes Event – wie einen Knopfdruck – der Prozess angestoßen werden. Auch Sinn ergeben kann die Möglichkeit, einen Low-Power-Wakeup-Receiver einzusetzen, der unter geringem Energiebedarf das ISM-Frequenzband abhört und bei Empfang eines bestimmten Signals den Prozess anstößt (Interrupt).

Da sich bereits parallel eine Abschluss-Thesis zu einem ähnlichen Thema, aber mit Polling von Belegungsinformationen, befasst, soll sich dieses Projekt auf den Interrupt beziehen. So soll in erster Linie eine Lösung gefunden werden, die als externes Event mit einem Wakeup-Receiver das ISM-Band auf Wakeup-Signale überwacht. Da jedoch mit dieser Möglichkeit eine erhöhte Stromaufnahme einhergeht, sollte die Möglichkeit bestehen, bei Bedarf das eingesetzte Funk-Modul zu deaktivieren und damit auf den Modus Polling umzuschalten.

Inbetriebnahme und Administration

Zur Installation soll das Frontend an dem jeweiligen Raum platziert und über einen Schalter eingeschaltet werden. Das System soll durch WLAN-RSSI-Fingerprinting erkennen, zu welchem Raum das installierte Frontend gehört. In einem fremden Projekt wurde ein Location Based Service (LBS) zur Indoor-Positionsbestimmung anhand von WLAN-RSSI-Messungen, zugänglich über eine Web-API, implementiert. Auf diese Schnittstelle kann zur Positionsbestimmung zurückgegriffen werden.

Zum Prozess der Aktualisierung der Anzeige soll sich das Frontend mit einem WLAN-AP verbinden und von dem Backend die Raumbelegungsinformationen beziehen. Das Backend sollte hierzu bereits im Vorhinein die benötigten Raumbelegungsinformationen bezogen und aufbereitet haben.

Im Normalbetrieb soll das System somit ohne Administration auskommen. Im Backend bedarf es trotzdem einer Administrationsoberfläche, z. B. um den durch WLAN-RSSI-Fingerprinting erkannten Raum manuell zu überschreiben, um die Firmware der MCU im Frontend zu aktualisieren oder um die Batterie-Ladung des Frontends zu überwachen.

Rendern des anzuzeigenden Bildes

Die Belegungsinformationen sollen von dem Backend gesammelt, nach Änderungen gefiltert und an das Frontend gesendet werden. Die dortige MCU übersetzt anschließend die empfangenen

Informationen in einen Bild-Puffer pro Änderung, sodass diese Puffer über »partial updates« direkt an das Display zur Anzeige heraus gesendet werden können.

Alternativ dazu besteht die Möglichkeit, das anzuzeigende Bild von einem Server in dem Backend rendern zu lassen und dieses anschließend auf die MCU zu übertragen. Dieses Verfahren hat jedoch einige Nachteile gegenüber dem Ansatz, die rohen Informationen auf die MCU zu übertragen und dieser das Rendern zu überlassen:

- Durch die Übertragung eines Bitmaps werden mehr Daten über WLAN übertragen. Ein Monochrombitmap bei einer Auflösung von 800 × 480 Pixel für das EPD hätte mind. eine Größe von 48 kB. Das führt zu einer längeren Sende- und Empfangs-Zeit und damit zu einem erhöhten Stromverbrauch v. a. beim Frontend.
- Zudem sind 48 kB im Microcontroller-Kontext eine große Datenmenge, die bei der Allozierung von dem benötigten Heap-Speicher zu Problemen führen kann (vgl. Kapitel *5.7-Firmware Aktualisierung*, Abschnitt *Speicherproblem*)
- Das Backend müsste hierzu aus einer weiteren Komponente bestehen, die von Node-RED angesteuert wird, um ein entsprechendes Bild zu erzeugen. Dies erhöht die Komplexität des gesamten Systems.
- Das vom Backend erzeugte Bild muss ein Monochrom-Bitmap sein. Anzuzeigende Schriftarten oder Linien müssten hierfür abgetastet werden, da diese in Form von Vektor-Informationen vorliegen. Ist die Schriftart zu klein, kann es zu einer Unterabtastung[5] kommen, wodurch die Schrift unleserlich wird bzw. Linien verschwinden. Für das Rendering auf der MCU müssten auch Schriftarten abgetastet werden, jedoch würde dies bereits zur Entwicklung und nicht erst zur Laufzeit passieren, wodurch ein größerer Einfluss auf das Resultat genommen werden kann.

Von Vorteil wäre, dass das Frontend keine Bilder selbst rendern müsste. Dadurch könnte eine höhere Flexibilität erreicht werden, da der Code auf der MCU nicht aktualisiert werden müsste, um die Art und Weise der Darstellung anzupassen. Da die Nachteile überwiegen, wurde entschieden, das Rendering mit der MCU durchzuführen.

4.2 Auswahl der Komponenten

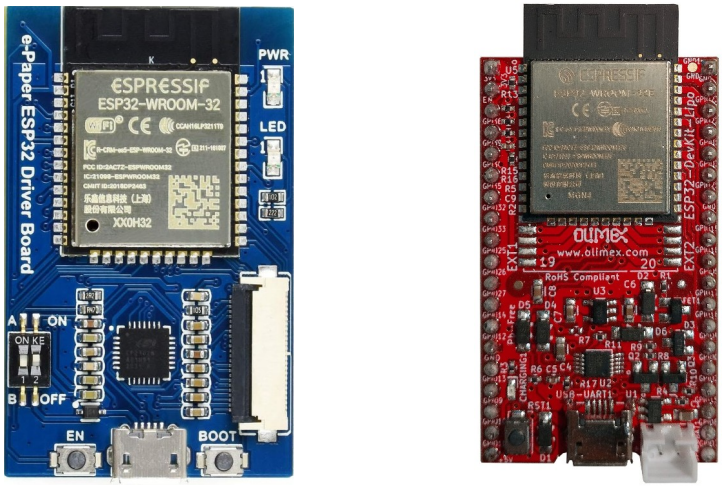
Das zu verwendende Entwicklungsboard

Der ESP32 ist mit verschiedenen Entwicklungs-Boards verfügbar. Ein naheliegendes Board ist das „e-Paper ESP32 Driver Board“ von Waveshare International Ltd. (vgl. Abb. 7, links). Es bietet eine eigens für EPDs vorgesehene Buchse, wodurch der Anschluss eines EPD sich sehr einfach gestaltet. Jedoch ist dieses Board für den Low-Power-Betrieb ungeeignet, da es von zwei Längsspannungsreglern mit einem Ruhestrom von jeweils 90 µA betrieben wird. Das EPD wird separat über den zweiten Regler betrieben; dieser wird über die gleiche Stromschiene wie der erste betrieben. Dies führt zu dem Problem, dass beide Regler verwendet werden müssen. Es gibt keine

5 Abtastung eines Signals unter zu geringer Frequenz, was zu Informationsverlust führt

Möglichkeit, den ESP32 und/oder das EPD durch einen dritten separaten Spannungsregler zu betreiben. Dazu kommt eine LED, die an der gleichen Stromschiene wie der ESP32 selbst betrieben wird. Würde die LED entfernt werden, würde im Deepsleep-Modus des ESP32 das Entwicklungsboard damit ca. 190 µA Strom aufnehmen. [11], [12]

Das „ESP32-DevKit-LiPo Board“ der Olimex Ltd. (vgl. Abb. 7, rechts) ist für den Betrieb über eine Batterie optimiert. Das Board wird mit einem Schaltregler mit einem Ruhestrom von 55 µA betrieben. Darüber hinaus besitzt dieses Board bereits einen Laderegler für eine Lithium-Polymer (LiPo)-Sekundärzelle. Zudem stellt dieses Board direkt eine Lösung zur Messung der eigenen Batteriespannung bereit. Somit kann der Ladestand der Zelle ohne weitere Hardware ermittelt werden. Da dieses Entwicklungsboard jedoch keine Treiber-Funktionalität für ein EPD bietet, muss diese durch z. B. das „e-Paper Driver HAT“ von Waveshare nachgerüstet werden. Dieses wird über das SPI an den ESP32 angeschlossen. Das Driver-HAT hat die Möglichkeit, durch einen Metall-Oxid-Halbleiter-Feldeffekttransistor (MOSFET) weitgehend stromlos geschaltet zu werden. Somit nimmt im Deepsleep-Modus des ESP32 das Board in Summe ca. 65 µA Strom auf. [13], [14]



e-Paper ESP32 Driver Board *ESP32-DevKit-LiPo Board*

Abbildung 7: Vergleich Abbildungen ESP32 Entwicklungsboards von Waveshare und Olimex

Wakeup über LoRa-WAN

„e220“ bezeichnet eine Familie aus Long Range Wide Area Network (LoRaWAN, kurz LoRa)-Modulen der Chengdu Ebyte Electronic Technology Co.,Ltd. Diese Module stellen einen Wake-on-Radio-Modus zur Verfügung, der es erlaubt, das ISM-Band bei 433 MHz bzw. 868 MHz bei einer verhältnismäßig geringen Leistungsaufnahme auf Wakeup-Signale zu überwachen. Je nach Einstellung werden bei dem Modul E220-400T30D alle 4 s auf dem Frequenzband auf das Signal gelauscht. Bei Annahmeweise einem Tastgrad[6] von $D = 1{,}25\,\%$, einem Ruhestrom von 5 µA und einem Empfangsstrom von 17.2 mA entspricht das einer zusammengesetzten Ladungsmenge von

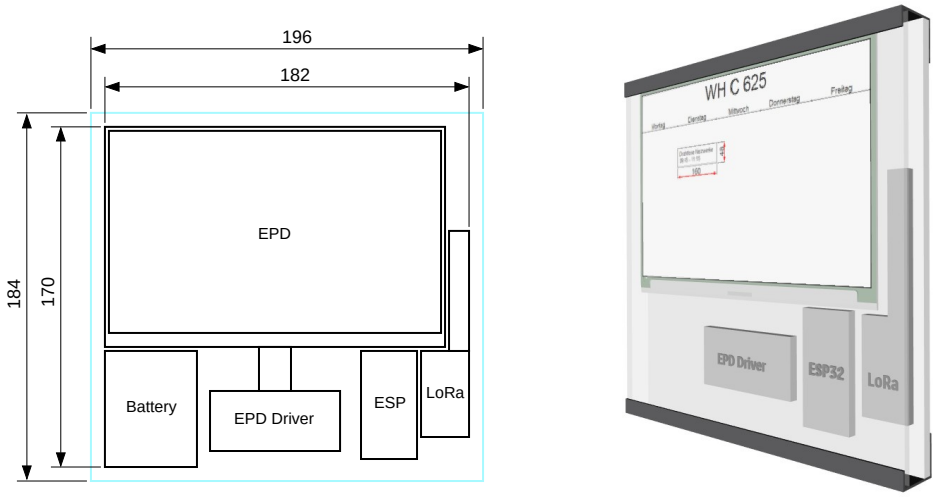
6 Verhältnis der Impulsdauer τ zur Periodendauer T. $D = \tau \div T$

rund 220 µAs über 1 s. Dies entspräche einer jährlichen Ladungsmenge von ca. 1,93 Ah (vgl. Anhang *Fehler: Verweis nicht gefunden - Fehler: Verweis nicht gefunden*). [15]

Durch die Deaktivierung des LoRa-Moduls zur vorlesungsfreien Zeit, Nachtzeit (zwischen 20 Uhr und 7 Uhr) sowie zum Wochenende lassen sich theoretisch 77 % (6.810 h/Jahr) der Energie einsparen.

4.3 Konzept eines physischen Prototypen

Für den physischen Prototypen sollten die Komponenten sich geschützt zwischen einer transparenten Polystyrol- und einer Multiplex-Platte, am Rand eingefasst von einem Aluminiumprofil, befinden (vgl. Abb. 8). Da die Antenne des LoRa-Moduls und das EPD-Driver-HAT jeweils die größte Dicke von 10 mm aufweisen, sind diese der begrenzende Faktor. Somit beträgt der Abstand zwischen den beiden Platten 10 mm.



Zeichnung in Frontalansicht *CAD-Modell in Eckansicht*

Abbildung 8: Physisches Konzept des Frontends

5 Implementierung

Das Backend besteht im Kern aus einer Node-RED-Instanz, einem MQTT-Broker und einem Dienst zur Micropython-Code-Validierung. Der MQTT-Broker dient der Interaktion zwischen den Komponenten sowie dem Frontend und stellt damit eine Middleware dar (vgl. Abb. 9). Node-RED, die Code-Validierung (Code Checker) und der MQTT-Broker wurden in eigenen Containern virtualisiert. Das Container-Netzwerk ist über das lokale WLAN erreichbar. Der WOR-Sender wird durch einen weiteren ESP32 MCU mit einem LoRa-Modul dargestellt. Auf dieser Komponente liegt jedoch ein geringeres Augenmerk.

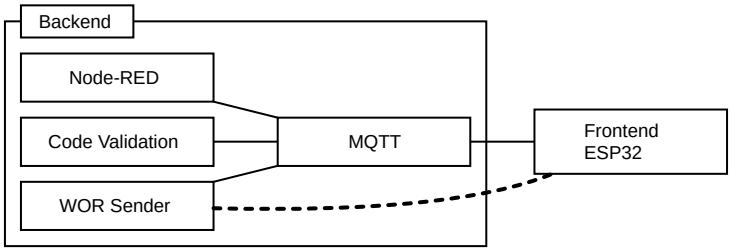


Abbildung 9: Diagramm der Interaktion zwischen den einzelnen Komponenten

5.1 Kommunikationsprotokoll Frontend <=> Backend

Zuerst wurde das Kommunikationsprotokoll zwischen dem Frontend und dem Backend entwickelt. Da sowohl Python als auch Node-RED von Haus aus zur Datenserialisierung mit der JavaScript Object Notation (JSON) arbeiten, wurde JSON als Datenformat zur Kommunikation gwählt.

Das Protokoll verwendet vier MQTT-Topics:

- `EPD/clrequest`: Die MCU meldet sich bei dem Node-RED und fragt nach einer CommandList. Als Nutzlast wird die eigene Hardware-ID übergeben.
- `EPD/clresponse`: Der Node-RED antwortet mit einer Befehlsliste. Die Befehlsliste besteht aus 1-n Befehlen mit Nutzlast, die der MCU alle notwendigen Operationen befiehlt. Zum ersten Kontakt ist die MCU dem Node-RED noch unbekannt. Hier antwortet der Node-RED (vgl. Abb. 10, links) mit dem Befehl `lora_set_addr`, um der MCU die Adresse des LoRa-Moduls mitzuteilen, die es annehmen soll. Anschließend bekommt die MCU mit dem Befehl `date` die aktuelle Uhrzeit mitgeteilt. Anhand der Uhrzeit schlussfolgert die MCU, wie lang sie höchstens im Schlafzustand verweilen soll. Geht der Befehl EPD_upd ein (vgl. Abb. 10, rechts), soll die MCU Belegungsinformationen auf dem EPD anzeigen bzw. aktualisieren.
- `EPD/lbs/rssi/[id]`: Mit dem Befehl `scan` soll die MCU ihre umgebenden APs und die zugehörigen RSSIs erfassen und das Ergebnis an dieses Topic übermitteln. Als `[id]` wird die eigene Hardware-ID eingesetzt.
- `EPD/status/[id]`: Zu diesem Topic sendet die MCU Statusinformationen. Darunter fallen Batterieladestand und etwaige Fehlermeldungen.

```json
[
  {
    "action": "lora_set_addr",
    "addh": 0,
    "addl": 0,
    "chan": 4
  },
  {
    "action": "scan"
  },
  {
    "action": "date",
    "date": "2025-01-11T19:43:56.300Z"
  }
]
```

```json
[
  {
    "action": "EPD_upd",
    "events": [
      {
        "name": "B211 Drahtlose Netzwerke (PCÜ)",
        "team": "1. Zug, 1. Gruppe",
        "time": [
          "12:15",
          "13:45"
        ],
        "wd": 2
      }
    ],
    "roomname": "WH_C_625",
    "datespan": "06.01.2025 - 12.01.2025",
    "date": "2025-01-11T20:22:28.534Z"
  }
]
```

Abbildung 10: Protokoll JSON CommandList Response

Das Backend nutzt die Hardware-ID, um das jeweilige Frontend zu identifizieren. Das Backend speichert zu jeder ID Geräteinformationen (Ladestand der Batterie, Fehlermeldung, zugeteilte LoRa-Adresse) und Raumbelegungsinformationen (Raumname, Raum-ID, Veranstaltungsliste).

Mit jedem Aufruf der Funktion wait_msg() der MQTT-Client-Bibliothek wird auf eingehende Nachrichten des MQTT-Brokers geprüft. Ist eine Nachricht eingegangen, wird diese an einen zuvor definierten Callback übergeben. Da das Konzept der Software einen synchronen Aufruf der Funktion vorsah, wurde die Bibliothek modifiziert, sodass der Aufruf der Funktion wait_msg() direkt die empfangene Nachricht zurückgibt.

5.2 Physische Ansteuerung des EPD

Im ersten Schritt wurde der EPD-Driver mit der MCU verbunden. Da zur Benutzung des WLAN-Stacks der erste ADU (ADC1) und mit ihm die Ein- und Ausgänge 32-36 und 39 nicht verwendbar sind, müssen der Anschluss des LoRa-Moduls und des EPDs sich auf die übrigen Ein- und Ausgänge verteilen. Hierbei benötigt das EPD den MISO und die Clock eines SPI und fünf weitere Logik-Eingänge und -Ausgänge (vgl. Abb. 11). Dabei müssen für das LoRa-Modul zwei Logik-Ausgänge übrig bleiben, die von der RTC der MCU gesteuert werden können (vgl. Kapitel *5.5 - Ansteuerung des LoRa-Moduls*, Abschnitt *Physischer Anschluss*).

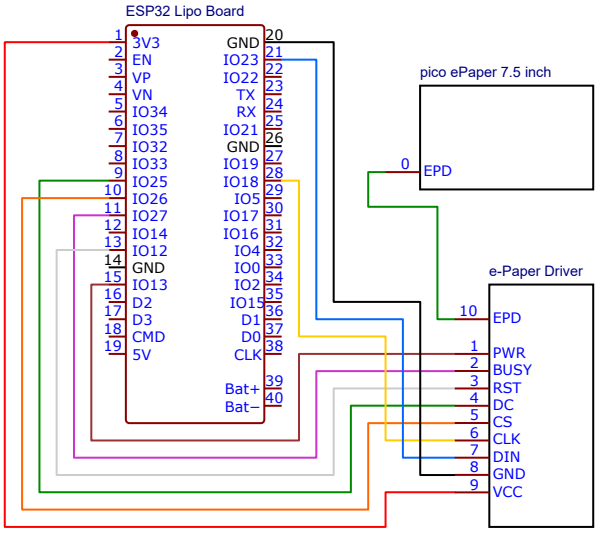


Abbildung 11: Schematischer Anschluss des EPD-Treibers an den ESP32

Zur Ansteuerung des EPD wird jedes Pixel über ein Bit dargestellt. Die Bits werden von der Bibliothek zeilenweise über die jeweilige SPI über Master out, Slave in (MOSI) an das EPD gesendet. Während des Renderings werden die Bits in einem Byte-Array als Datenstruktur vorgehalten.

Der EPD-Treiber sorgt in erster Linie dafür, dass das EPD mit einem eigenen linearen Spannungsregler versorgt wird. Darüber hinaus ermöglicht er durch einen JST-Steckverbinder den einfachen Anschluss an General Purpose Input/Output (GPIO)-Pins einer MCU.

5.3 Logische Ansteuerung des EPD

Im Folgenden wird erläutert, wie das Bild verarbeitet und vorbereitet wurde, um es erfolgreich auf dem EPD anzuzeigen.

Für alle Operationen (sowohl das Schreiben von Text, als auch das Zeichnen von Linien) wird zuvor ein Bild-Puffer in Form eines Byte-Arrays erzeugt. Jedes Byte in diesem Puffer steht für acht nebeneinanderliegende Pixel. Stehen die Bits auf 1, werden die entsprechenden Pixel schwarz gefärbt.

Logische Aufteilung des Displays

Die in Abb. 12 gezeigte Aufteilung wurde implementiert. Über dem Plan, in den ersten 96 Pixel, werden Meta-Informationen, wie die Raumnummer und der Bereich der Woche, angezeigt. Jeder Wochentag und jede Uhrzeit sind fest definierte Fenster, in denen Veranstaltungen angezeigt werden können. Da Veranstaltungen generell zur vollen Stunde und danach in einer 15-Minuten-Taktung beginnen und enden, wurden vier Fenster pro Stunde definiert. Da Veranstaltungen selten vor 8 Uhr und nach 20 Uhr stattfinden, wurde dieser Zeitraum zur Anzeige definiert.

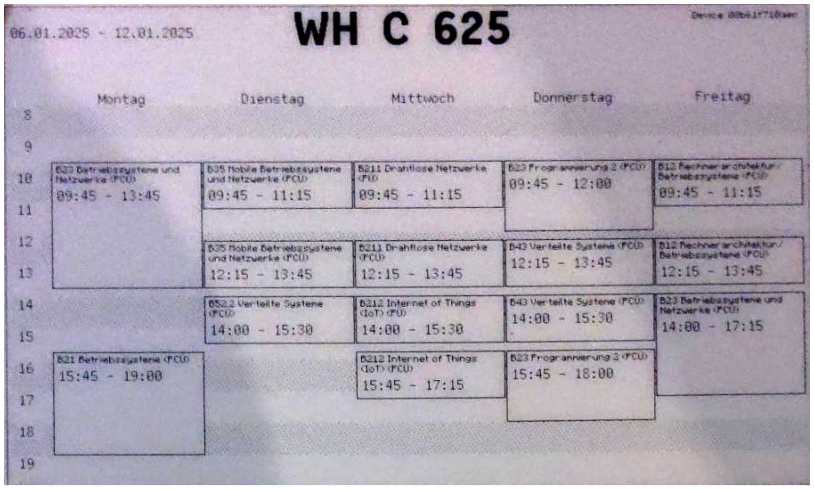


Abbildung 12: Logische Aufteilung des Raumplans auf dem EPD

Soll eine Veranstaltung angezeigt werden, wird die Nummer des Wochentags ermittelt, um anschließend mit der Funktion `get_horizontal_slot_dim()` die horizontalen Koordinaten `x1` und `x2` des entsprechenden Fensters zu ermitteln. Die Übergabe des Wertes „-1“ lässt die Funktion die Maße der Zeitspalte (0, 39) zurückgeben. Die Wochentags-Spalten sind 152 Pixel breit und beginnen mit dem Montag mit (40, 191).

Es werden die Stunden 8 - 19 dargestellt. Jede Stunde hat mit der Einteilung in Viertelstunden vier Einteilungen. Damit wird der Platz für die Veranstaltungen vertikal in 48 Blöcke unterteilt. Vertikal stellt das EPD zur Anzeige 480 Pixel zur Verfügung. Somit bestünden für jeden einzelnen Block 10 Pixel Platz. Jedoch wird im oberen Bereich des Displays Platz zur Anzeige von Meta-Informationen, wie z. B. der Raumname und die aktuelle Woche, aber auch zur Anzeige von Wochentagen, benötigt. So wird der Platz pro Block auf 8 Pixel reduziert, was zu einem Gesamtplatz von 384 Pixel führt. Somit stehen für die Meta-Informationen im oberen Bereich 96 Pixel zur Verfügung.

Die Teilschritte der Entwicklung der Raumplan-Darstellung kann dem Anhang *A - Evolution der Anzeige des Raumplans auf dem EPD* entnommen werden.

Zeichnen von Schrift

Zum Schreiben von Text in den Bild-Puffer (zur Anzeige auf dem EPD) wurde die öffentlich zugängliche Micropython-Bibliothek „micropython-font-to-py“[7] verwendet. Die Bibliothek ermöglicht es zum einen, die vektor-orientierten Schriftformate TrueType-Fonts (.ttf), wie auch OpenType-Fonts (.otf) im Vorhinein nach Python-Dateien zu rastern; zum anderen ermöglicht sie es, durch diese gerasterten Schriftarten unter geringem Ressourcen-Aufwand Text in den Bild-Puffer zu schreiben. Zum Rendern von Schrift wird der Bibliothek der anzuzeigende String mit dem Bild-Puffer als Byte-Array und der zu verwendenden Schriftart zusammen übergeben.

In ihrer Voreinstellung rastert die Bibliothek alle druckbaren ASCII-Zeichen. Sollen Zeichen anderer Zeichensätze oder weniger Zeichen übersetzt werden, können durch die Angabe von einer

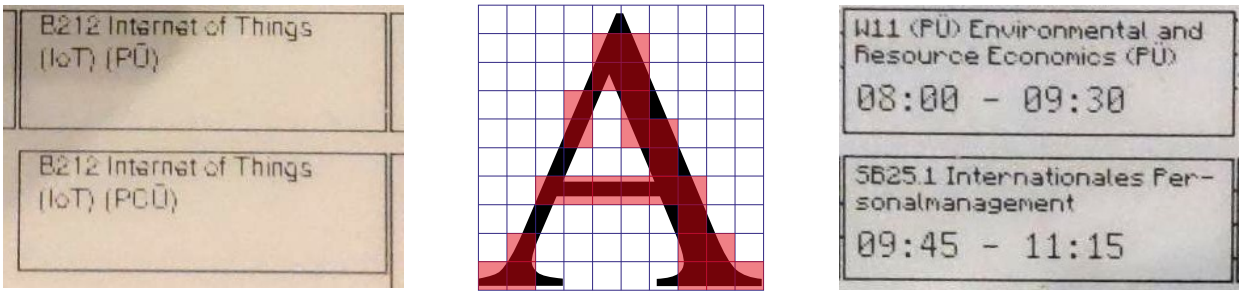
7 https://github.com/peterhinch/micropython-font-to-py, letzter Zugriff 02.02.2025

Datei mit Unicode-kodierten Zeichen über den Schalter „-k“ die zu rasternden Zeichen definiert werden (vgl. Abb. 13).

```
font_to_py.py -k charset.txt FreeSans.ttf 17 font10.py
```

Abbildung 13: Befehl zum Rastern einer Schriftart unter Angabe einer Charsets

Bei dem Rastern zu kleiner Schriftarten entsteht das Problem der Unterabtastung (vgl. Abb. 14, Bild 1). Es wird ein Raster über den Buchstaben gelegt und algorithmisch entschieden, ob der entsprechende Pixel eingefärbt wird. Ist der Buchstabe zu klein, überspringt der Algorithmus wichtige Konturen bzw. Bereiche (vgl. Bild 2) und das Ergebnis ist unleserlich. Um diesem Problem zu begegnen, wurden speziell Bitmap-Schriftarten mit einer festen Größe gewählt, die mit der Abtast-Größe übereinstimmen. Damit ist die Schrift besser lesbar (vgl. Bild 3).



1 *2* *3*

Abbildung 14: Problem der Unterabtastung

Wort- und Silben-Trennung

Die Funktion `splitlines()` zerteilt die auszugebenden Zeichenkette, unter Beachtung des zur Verfügung stehenden Platzes, in mehrere Zeilen. Dazu wird die Zeichenkette silben- und wortweise zerlegt. Die Zeile wird solange um Silben des Strings erweitert, wie Platz zur Verfügung steht. Der Platz, den die Silben und Wörter in Anspruch nehmen, werden anhand der Buchstabenbreite aus der Schriftarten-Datei errechnet. Reicht der Platz nicht mehr aus oder ist das Ende der Zeichenkette erreicht, wird die vorbereitete Zeile in die Liste der auszugebenden Zeilen übernommen.

Die Wörter werden dazu am Leerzeichen voneinander getrennt. Zur Silbentrennung der Wörter wurde die vereinfachte Regel »vc_cv« angewandt, bei der ein Wort zwischen zwei aufeinanderfolgenden Konsonanten, die von Vokalen eingeschlossen sind, getrennt wird (vgl. Abb. 15). Zur Verbesserung der Ergebnisse wurde der Buchstabe „Y“ in die Liste der Vokale aufgenommen und aus der Liste der Konsonanten entfernt. Nicht in allen Fällen ist diese Regel fehlerlos (z. B. bei Komposita wie „Demokratie“), aber sie erzeugt trotzdem überwiegend eine korrekte Silbentrennung.

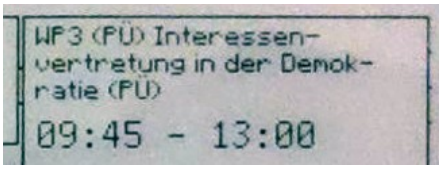


Abbildung 15: Beispiel der Silbentrennung im Raumplan

Textausrichtung

An einigen Stellen auf dem Display sollte der Text zentriert (z. B. die Wochentage) oder rechtsbündig ausgerichtet sein. Um Text auszurichten, wird zuvor der Platz definiert, der dem Text zur Verfügung steht. Die Writer-Bibliothek stellt eine Funktion zur Verfügung, mit der die Pixel-Breite eines Textes unter Angabe der verwendeten Schriftart ermittelt wird. Die Breite des Bereichs, der unbeschrieben bleibt, bildet sich aus der Differenz des zur Verfügung stehenden Platzes und der Textbreite. Zur zentrierten Ausrichtung wird der anzuzeigende Text um die Hälfte des rechts unbeschrifteten Platzes verschoben. Um den anzuzeigenden Text rechtsbündig auszurichten, wird stattdessen der unbeschriebene Platz nach vorne gesetzt.

Da die einzelnen Veranstaltungen sich gewöhnlich über eine Dauer von 90 min erstrecken, stehen pro Veranstaltung sechs Blöcke zu je acht Pixel Höhe zur Verfügung. Das entspricht einem Platz in Summe von 48 Pixel; nach Abzug der Rahmenbreite und Abstand zum Rahmen bleiben nur noch 42 Pixel. In diesem Bereich werden sowohl der Veranstaltungsname als auch die Zeitspanne der Veranstaltung angezeigt. Mit einer 10 Pixel hohen Schrift könnten so 4 Zeilen Text angezeigt werden.

Die meisten Veranstaltungen nehmen bei der entsprechenden Schriftart nur zwei Zeilen Platz ein. Darüber hinaus wird die Zeitspanne der Veranstaltung mit einer 16 Pixel hohen Schrift dargestellt, da dieser Information eine höhere Relevanz zugeschrieben wird.

Falls der Veranstaltungsname länger ist, als Platz zur Verfügung steht, werden hinten an die letzte gedruckte Zeile Auslassungspunkte angefügt (vgl. Abb. 16). Falls weniger oder mehr Platz zur Verfügung steht, weil die Veranstaltung sich über eine andere Dauer erstreckt, wird die Anzahl anzuzeigender Zeilen dynamisch angepasst. Entsprechend wird bei wenig Platz die Uhrzeit näher an die letzte Zeile des Veranstaltungsnamens herangerückt.

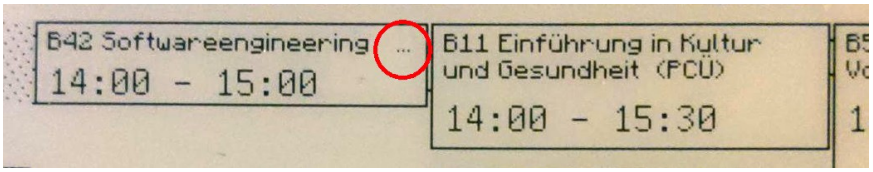


Abbildung 16: Darstellung Veranstaltungsname auf dem EPD mit Abstand zur Zeitspanne

Zeichnen von Linien

Bei dem Rendern horizontaler Linien wird im ersten Schritt der Start-Index in Pixel ermittelt (vgl. Abb. 17, Z. 224). Von diesem Index ausgehend werden der Länge nach in Richtung rechts alle Pixel abgelaufen (Z. 225). Zum Einfärben des Pixels muss die zweidimensionale Pixel-Koordinate in einen eindimensionalen Byte-Index umgerechnet werden. Um in dem Ziel-Byte den korrekten Pixel zu treffen, werden durch eine nicht-ausschließende Disjunktion das Ziel-Byte mit einem one-hot-kodierten Byte verknüpft (Z. 228).

```
224     pos = x + y * self.buf.width
225     for i in range(pos, pos+length):
226       if i < 0 or i >= 384_000:
227         continue
228       self.buf.buffer[ i//8 ] |= 0x80 >> (i%8)
```

Abbildung 17: Ziehen einer horizontalen Linie auf dem EPD, epd.py

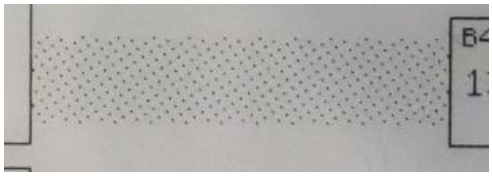
Vertikale Linien wurden auf die gleiche Art und Weise gezeichnet. Hier wurde jedoch als Schrittweite die Bildschirmbreite gewählt, sodass der nächste angesteuerte Pixel nicht neben, sondern unter dem Vorgänger-Pixel liegt.

Hervorhebung von Tabellenzeilen

Da in Tabellen zur besseren Lesbarkeit oft jede zweite Zeile eingefärbt wird, sollte auch hier jede zweite Stunde des Raumplans hervorgehoben werden. So wurde ein Muster gezeichnet, bei dem jeder 13. Pixel (Jeder Pixel-Index, der sich ganzzahlig durch 13 dividieren lässt, vgl. Abb. 18, links, Z. 278) eingefärbt wird. Dieser Divisor kontrolliert, wie sich das Muster zusammensetzt. Ist der Divisor ein Teiler der Bildschirmbreite bzw. der Breite des zu zeichnenden Teilbereichs, entstehen als Muster vertikale Linien. Mit 13 als Divisor wurde ein Muster mit Punkten, die diagonal zueinander angeordnet sind (vgl. rechts), gefunden.

```
274     pos1 = x1 + y * self.buf.width
275     pos2 = x2 + y * self.buf.width
276
277     for i in range(pos1, pos2):
278       if (i+y)%mod == 0:
279         self.buf.buffer[ i//8 ] ^= (0x80 >> (i%8))
```

EPD/EPD.py



Gezeichnete Zeile

Abbildung 18: Zeichnung der Hervorhebung von Tabellenzeilen

Das Rendering

In dem Rendering werden zuerst folgende Objekte dargestellt: Raumname, Zeitspanne der aktuell angezeigten Woche, die ID des MCU und die Wochentage über der Tabelle. Anschließend werden für alle Stunden (Stunde 8 bis Stunde 19) das Zeilenmuster gezeichnet und die Stunden selbst am Anfang der Zeile dargestellt. Abschließend werden für alle Termine der Name der jeweiligen Veranstaltung und darunter die Zeitspanne sowie die horizontalen wie vertikalen Linien gezeigt.

Anstatt partielle Aktualisierungen der Anzeige durchzuführen, werden vollständige Aktualisierungen durchgeführt. Zum einen erlaubt die Bibliothek scheinbar, nur Änderungen des Rot-Kanals durchzuführen. Zum anderen ist es erforderlich, das Display konstant mit Strom zu versorgen, da andernfalls der Bild-Puffer die angezeigten Informationen verliert und anschließend eine partielle Aktualisierung zu einem verrauschten Bild führt. Zwar nimmt die Anzeige im Sleep-Modus mit 20 µA nur wenig Strom auf, jedoch ist auch zum Sleep-Modus der Spannungsregler des Driver-HATs aktiv und nimmt weitere 90 µA Strom auf [7]. So wurde stattdessen entschieden, vollständige Aktualisierungen des EPDs durchzuführen und zum Deepsleep der MCU das EPD über den PWR-Pin auszuschalten.

5.4 Ladestandsmessung und -anzeige

Um den Ladestand der Batterie zu messen, bringt das Entwicklungsboard eine Lösung mit: Durch einen Spannungsteiler wird die Spannung der Batterie durch zwei Widerstände jeweils der Größe 470 kΩ reduziert und über den GPIN35 an den Analog-Digital-Umsetzer (ADU) der MCU übergeben (vgl. Abb. 19). Da eine LiPo-Zelle ein Spannungspotential von bis zu 4,2 V aufweist, muss diese auf den Spannungsbereich der MCU auf 3,3 V verringert werden. Durch zwei identische Widerstände in einem Spannungsteiler wird die Spannung bei vernachlässigbar geringem Stromfluss halbiert:

$$V_{out} = \frac{V_{source} \cdot R_1}{R_1 + R_2} = \frac{4.2\,v \cdot 470\,k\,\Omega}{470\,k\,\Omega + 470\,k\,\Omega} = 2.1\,v$$

Der ADU der MCU löst mit 12 Bit Genauigkeit auf, d.h. sein Wertebereich liegt bei $(0, 1, \ldots, 4094, 4095)$, wobei theoretisch ein gemessener Wert von 4095 einer Spannung von 3,3 V am Eingang des ADU entspricht. Neben der Strommessung bietet das Entwicklungsboard über den GPIN 39 (VN) auch eine Möglichkeit zu erkennen, ob das Board extern mit Strom versorgt wird.

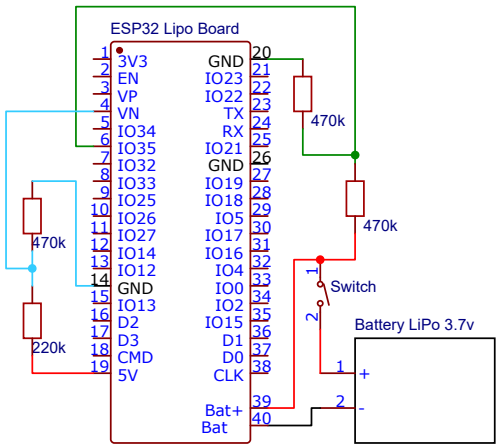


Abbildung 19: Schemata der Messung der Batteriespannung, nach [13]

Um von dem gemessenen Wert auf die Spannung zu schlussfolgern, kann der gemessene Wert durch den Quotienten aus dem größtmöglichen Wert und der größtmöglichen Spannung dividiert werden. Durch Halbierung des Quotienten wird das Ergebnis an den Bereich des Spannungsteilers angepasst. In dem Fall würde sich 620 als Quotient ergeben, um von dem gemessenen Wert des ADUs auf die Spannung der Batterie zu folgern. Jedoch weist der ADU bei kleinen wie großen Spannungen Ungenauigkeiten auf, die sich auf den Wertebereich auswirken. Eine Gegenüberstellung der von dem ADU gemessenen Werte zu der Batteriespannung (vgl. Abb. 20) ergab, dass die Ladung der Batterie mit abnehmendem Wert tatsächlich stärker abnimmt (steilerer Abfall der Funktion), als sie rechnerisch abnehmen müsste.

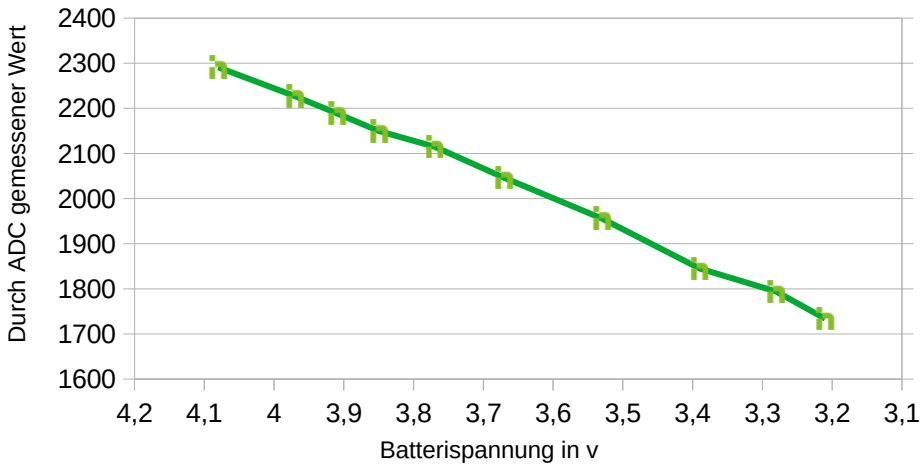


Abbildung 20: Gegenüberstellung gemessener Werte des ADU zur Batteriespannung

So wurde eine lineare Funktion (vgl. Abb. 21, Z. 42) errechnet, die die gemessene Spannung abbildet. Um Messfehlern des ADUs vorzubeugen, wird eine Serie von zehn Batteriespannungen gemessen und anschließend gemittelt (vgl. Z. 34-39).

Um von der Spannung bzw. von dem Messwert auf den Ladestand der Batterie zu schließen, wird ähnlich verfahren (vgl. Z. 43). Hier wird angenommen, dass sich die Spannung weitgehend linear zum Ladestand der Batterie verhält. Über ein lineares Gleichungssystem werden die Steigung und die Konstante der Funktion ersten Grades ermittelt. Es wurde angenommen, dass bei einer Spannung von 4,2 V die Batterie vollständig geladen ist und bei einer Spannung von 3,3 V die Batterie vollständig entladen ist.

```
34  mean_cnt = config.battery['meansby']
35  raw = 0
36  for i in range(mean_cnt):
37    raw += _bat.read()
38    sleep(0.01)
39  raw = raw//mean_cnt
40
41  # formulas to calculate voltage and percentage
42  voltage = raw / 640 + 0.5
43  perc = max(raw / 570 - 3.16, 0)
```

Abbildung 21: Messung der Batteriespannung, bat.py

Nach der Messung der Spannung wird eine Statusmeldung an das Backend abgegeben. Diese Statusmeldung beinhaltet den tatsächlich gemessenen Wert, die errechnete Spannung und den errechneten Ladestand in Prozent. Ist der kritische Ladestand (0 %) erreicht, wird eine zusätzliche Prüfung durchgeführt, ob derzeit eine externe Stromquelle existiert. Ist dies nicht der Fall, wird die Statusmeldung um den Hinweis erweitert, dass die MCU sich in den Deepsleep versetzt. Anschließend versetzt sich die MCU in den Deepsleep.

Geht eine Statusmeldung einer MCU ein, werden entsprechend die Informationen aus der Nachricht extrahiert und abgespeichert. Falls der Ladestand ein zu niedriges Niveau erreicht hat, wird ein HTTP-Request an eine NTFY-Instanz heraus gesendet (vgl. Abb. 22), wodurch eine abonnierende

Person eine Systembenachrichtigung erhalten kann. Nach der Speicherung der Ladestandsinformationen wird der Ladestand im Administrations-Dashboard angezeigt (vgl. Abb. 23, Spalte „Battery“).

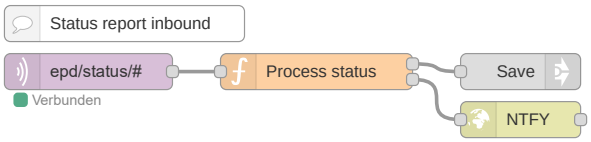


Abbildung 22: Vermerk der Batteriespannung und Benachrichtigung, Node-RED-Flow

#	EPD id	Room Name	Address	Last seen	Battery	St
1	08b61f710aec	WH_C_624	0	25.01.2025 16:28	2.54v - 0%	
2	08b61f710b9c		1	19.01.2025 19:47	0.61v - 0%	

Abbildung 23: Anzeige des ESP32 im Dashboard mit Ladestand, Node-RED-Dashboard

5.5 Ansteuerung des LoRa-Moduls

Physischer Anschluss

Das LoRa-Modul stellt zur Kommunikation eine einfache serielle Schnittstelle zur Verfügung (RXD, TXD). Über die Logik-Inputs M0, M1 wird das Modul in vier unterschiedliche Betriebs-Modi versetzt. Der Logik-Output AUX stellt die Information zur Verfügung, ob das Modul derzeit ausgelastet ist. [15]

Die Betriebs-Modi sind: »Transmission Mode« (normale Übertragung und Empfang), »WOR Transmitting Mode« (Senden von WOR-Signalen), »WOR Receiving Mode« (Empfang von WOR-Signalen) und »Deep Sleep Mode« (Das Modul befindet sich in Ruhe). Zum Deep Sleep Mode ist die Funk-Schnittstelle ausgeschaltet, aber das Modul lässt sich über die serielle Schnittstelle konfigurieren. Im WOR Receiving Mode lauscht das Modul mit einem einstellbaren Tastgrad auf WOR-Signale und zieht im entsprechenden Fall den AUX-Ausgang herunter. Dieses Signal kann die MCU dazu verwenden, aufzuwachen, um den Prozess der Aktualisierung der Anzeige anzustoßen. [15]

Um den Modus zu stellen, gibt es zwei Optionen: Das Anlegen eines Pull-up- bzw. Pull-down-Widerstandes an den Modus-Pins des LoRa-Moduls oder die Ausgabe eines High- bzw. Low-Pegels eines Logik-Ausgangs der MCU.

Da während des Betriebs Node-RED der MCU die LoRa-Adresse mitteilen soll, sodass die MCU das LoRa-Modul einstellen kann, scheidet die erste Option aus, da der Betriebsmodus sonst auf den Empfang von WOR-Signalen fixiert wäre. Bei der zweiten Option besteht jedoch das Problem, dass die Logik-Ausgänge der MCU nicht die Zustände zum Deepsleep halten, was zur Folge hat, dass das LoRa-Modul den Modus ändert, kurzzeitig den AUX-Ausgang herunterzieht und dadurch die MCU wieder aufweckt.

Die MCU besitzt eine RTC, die zum Deepsleep aktiv ist, um z. B. nach einer bestimmten Zeitspanne selbstständig aufwachen zu können. Einige Ausgänge der MCU sind mit der RTC

verbunden, sodass diese in der Lage ist, zum Deepsleep der MCU die Zustände der entsprechenden Ausgänge zu halten. [1], [13]

Somit wurden für die Modus-Eingänge des LoRa-Moduls die Ausgänge GPIO32 und GPIO33 der MCU gewählt. Darüber hinaus sollten die serielle Schnittstelle und der AUX-Ausgang des LoRa-Moduls mit schwachen Pull-up-Widerständen versehen sein (vgl. Abb. 24).

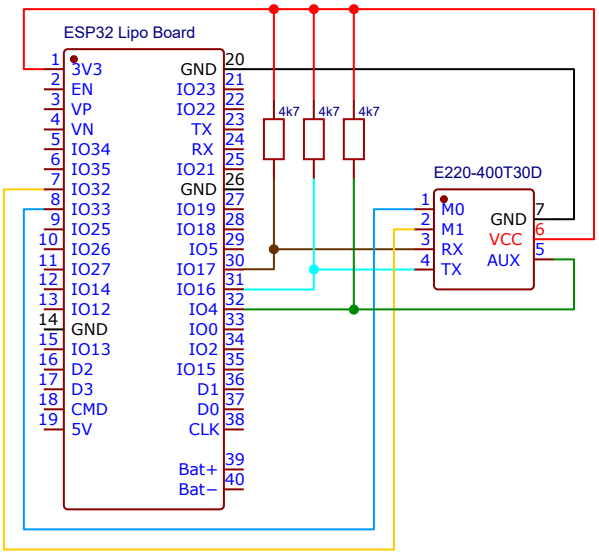


Abbildung 24: Schematischer Anschluss des LoRa-WAN-Moduls an den ESP32

Logische Ansteuerung

Das LoRa-Modul kann auf 84 verschiedenen Kanälen Geräte mit 65.536 (16 bit) verschiedenen Adressen ansprechen, wobei die letzte Adresse dem Broadcast zur Verfügung steht.

Es kann in zwei unterschiedlichen Übertragungsmodi betrieben werden. Im Modus Transparent Transmission (voreingestellt) empfangen alle Geräte bei gleichem Kanal und gleicher Adresse die gesendeten Informationen. Dabei werden alle Daten, die an die serielle Schnittstelle übergeben werden, an die Geräte übertragen, die dem Kanal des Senders entsprechen. Der Übertragungsmodus »Fixed Transmission« hingegen überträgt die Daten nur an Geräte mit einer bestimmten Ziel-Adresse. Der Kanal und die Adresse werden dabei als Präfix an die zu übertragenden Daten in die serielle Schnittstelle geschrieben. Für diese Arbeit wurde der Fixed-Transmission-Modus gewählt, da durch eine Adresse angegeben werden sollte, welches Frontend aufwachen soll.

Zur Konfiguration des LoRa-Moduls wird in den Konfigurationsmodus geschaltet (`M0 = 1, M1 = 1`), um anschließend die Konfigurationsregister über die UART-Schnittstelle zu beschreiben (vgl. Abb. 25).

Die ersten drei Byte definieren die Steuer-Informationen. Diese sagen aus, dass das Register, beginnend bei Adresse 0 (Byte 1) überschrieben (Byte 0) werden soll. Die Nutzlast mit der Länge von 8 Byte (Byte 2) folgt auf die Steuerinformationen.

Byte 3 und 4 definieren die LoRa-Adresse des Moduls. Byte 5 definiert die seriellen Parameter, wie die Übertragungsrate und Parität, durch ein Bitset. Byte 6 dient zum Einstellen von Paketgrößen

und Sendeleistung. Byte 7 definiert den zu verwendenden Kanal im Bereich von 0 bis 83. Byte 8 steuert u. a. den Transmission Mode (Transparent oder Fixed), CSMA/CA (LBT) und den WOR-Zyklus. Byte 9 und 10 werden für Übertragungsverschlüsselung verwendet.

Index	0	1	2	3	4	5	6	7	8	9	10
Byte	C0	00	08	00	00	62	00	04	43	00	00

Abbildung 25: Befehl zum Überschreiben der Konfiguration des LoRa-Moduls

Ansteuerung zur Laufzeit

Zur Erstinbetriebnahme des Frontends teilt Node-RED der MCU eine noch unbenutzte LoRa-Adresse mit. Empfängt die MCU diese Adresse, versetzt diese das LoRa-Modul in den Konfigurationsmodus und sendet eine Konfiguration (vgl. Abschnitt *Logische Ansteuerung*).

Ist die Befehlsliste durch die MCU abgearbeitet bzw. war die Befehlsliste leer, versetzt die MCU das LoRa-Modul in den WOR-Receiving-Modus (`M0 = 0, M1 = 1`) und geht selbst in den Deepsleep über (vgl. Abb. 26, Z. 18). Zuvor werden die Zustände der Logik-Ausgänge gehalten, damit das LoRa-Modul nicht direkt seinen Modus zurückwechselt (Z. 11-14). Damit die MCU durch die Signalisierung des LoRa-Moduls wieder aufwachen kann, wird zuvor ein externes Ereignis zum Aufwecken der MCU definiert (Z.7-10).

```
7  esp32.wake_on_ext0(
8    pin = config.lora['aux'],
9    level = esp32.WAKEUP_ALL_LOW
10 )
11 esp32.gpio_deep_sleep_hold(True)
12
13 m0 = Pin(config.lora['m0'], Pin.OUT, hold=True)
14 m1 = Pin(config.lora['m1'], Pin.OUT, hold=True)
15
16 print('good night')
17 sleep(1)
18 deepsleep(time*1000)
```

Abbildung 26: Betreten des Deepsleep, ds.py

Das Erwachen aus dem Deepsleep durch ein WoR hatte auf Anhieb nicht funktioniert. So wurde begonnen, die LoRa-Module in ihrer einfachsten Konfiguration zu testen. Über die REPL wurden beide Module konfiguriert und in die korrekten Modi versetzt. Auf dem Eingang der MCU, der an dem Aux-Ausgang des LoRa-Empfangs-Moduls angeschlossen ist, wurde ein Interrupt definiert, der eine Ausgabe auf der Konsole erzeugte, wenn das LoRa-Empfangs-Modul eine Nachricht empfangen hat.

Direkt zu Anfang hat sich herausgestellt, dass die Module werksseitig auf den Transparent-Tansmission-Modus eingestellt waren. Dies hatte zur Folge, dass das empfangende Modul nur bei gleich lautender Adresse zum Sender-Modul aufgeweckt werden konnte.

WOR-Sender

Die Komponente, die aus dem Backend heraus ein WOR-Signal über 433Mhz an die jeweiligen Frontends sendet, besteht aus dem identischen LoRa-Modul im WOR-Transmitting-Modus (m0=1, m1=0) und einem ESP32 (vgl. Abb. 27). Die Konfiguration dieses Moduls muss mit dem empfangenden Modul in den Paremetern „Air data rate“ und „WOR-Cycle“ übereinstimmen. Auch diese MCU wurde mit Micropython programmiert.

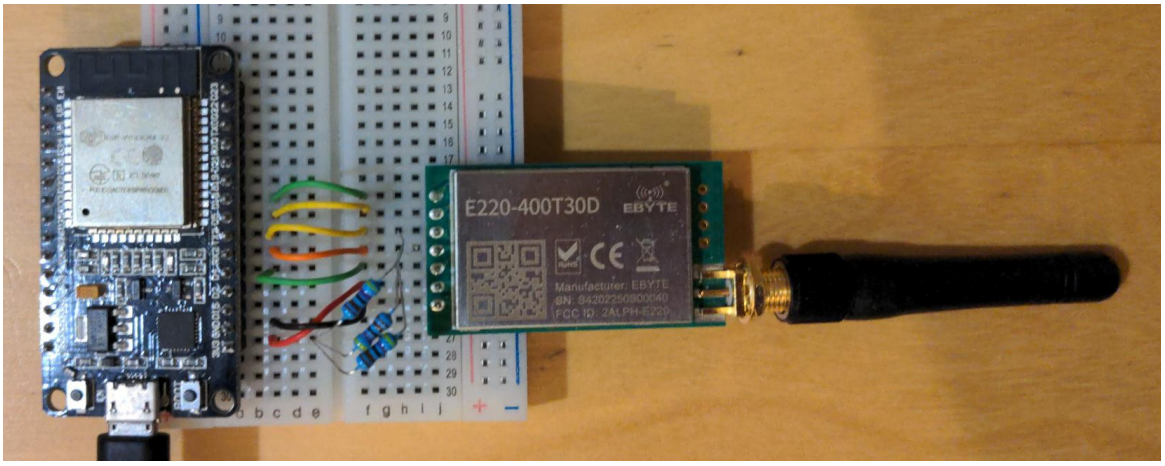


Abbildung 27: Der WOR-Sender im Backend

5.6 Erfassung anzuzeigender Raumbelegungen

Ermittlung des zugehörigen Raums

Im ersten Schritt empfängt die MCU den Befehl, ihre Umgebung nach WLAN-APs und zugehörigen RSSI-Werten abzusuchen. Anschließend teilt die MCU die daraus entstandene Liste Node-RED mit. Node-RED bereitet daraufhin einen HTTP-Request vor (vgl. Abb. 28), um bei der LBS-API in Erfahrung zu bringen, welcher Raumbezeichner am besten zu den Messungen passt (vgl. Abb. 29). Wurde nach spätestens drei Versuchen ein Bezeichner zurückgegeben, wird zum Bezeichner die Raum-ID ermittelt, wie der Raum im LSF geführt wird.

```
{
  "routers": [
    {
      "signal_strength": -48,
      "ssid": "12345678",
      "bssid": "d0:05:2a:94:d6:66"
    }
  ]
}
```

Abbildung 28: Beispiel einer Nutzlast zur Übergabe an die LBS-API

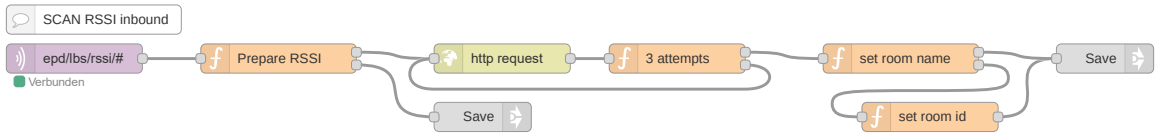


Abbildung 29: Anfordern eines Raumbezeichners mittels RSSI-Messungen, Node-RED-Flow

Konnte kein zugehöriger Raum ermittelt werden, wird eine Fehlermeldung zur jeweiligen MCU vermerkt und im Dashboard in der Geräteliste entsprechend markiert. Um die Rauminformation manuell der MCU bzw. dem Frontend zuordnen zu können, kann im Dashboard über den Knopf „set room“ ein Raum aus einer Liste ausgewählt werden. Dazu wurde über Javascript ein Click-Listener auf den Knopf gelegt, sodass im Dashboard ein anfangs unsichtbares Dropdown-Feld erscheint.

Übersetzen von Raumname nach Raum-ID

Durch die LBS API kann zu einem Satz von RSSI-Messungen ein Raum-Bezeichner (z. B. „WH_C_625“) ermittelt werden. Um Belegungsinformationen eines Raumes zu erhalten, erwartet das LSF als Identifikator Jedoch eine Raum-ID, die zum Bezeichner unterschiedlich ist (z. B. „4584“). Im LSF wird dieser Parameter „RGID“ genannt.

Zum Start des Node-RED wird einmalig die Raumliste eingelesen. Das LSF bietet keine dokumentierte öffentliche API an. So war es notwendig, die Raumliste aus dem HTML-Quellcode, der auch zur Web-Darstellung genutzt wird, zu extrahieren. Dazu wurde mit dem Streaming[8]-Parser „htmlparser2“ der HTML-Quellcode eingelesen und in eine Liste aus Schlüssel-Wert-Paaren übersetzt (RGID und Raumbezeichner).

Zu Anfang (vgl. Abb. 30, Zustand 0) werden so lange Symbole eingelesen, bis die HTML-Raumliste beginnt. Jeder Eintrag der Liste startet mit einem „div“-Container, in dem sich ein „a“-Tag befindet. Ein „a“-Tag definiert einen Link, dessen ziel im „href“-Attribut festgelegt ist. Dieses Ziel beinhaltet den Parameter RGID, der mit dem Auslesen des „a“-Tags erfasst wird (vgl. Übergang Zustand 2 - 3). Sobald der Raumname, der als Text vom „a“-Tag eingeschlossen wird, erfasst wurde (vgl. Zustand 3), wird dieser zusammen mit der RGID in der Raumliste gespeichert.

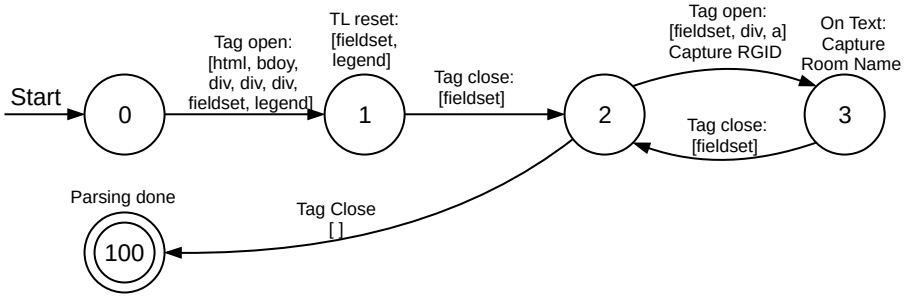


Abbildung 30: Zustandsautomat zum Parsen der Raumliste

Die Bibliothek htmlparser2 erwartet, dass die Methoden `onopentag()`, `ontext()` und `onclosetag()` der Klassen-Instanz überschrieben werden. `onopentag()` wird unter Angabe vom geöffneten Tag und der

8 Ein Parser, der ein terminales Symbol nach dem anderen einliest und direkt verarbeitet

Attribut-Liste aufgerufen, sobald der Parser einen öffnenden Tag vollständig eingelesen hat. ontext() wird unter Angabe des Inhalts aufgerufen, sobald das schließende Tag des zugehörigen öffnenden Tags eingelesen wird. Sobald das schließende Tag vollständig wurde, wird entsprechend `onclosetag()` aufgerufen.

Prüfung auf Belegungsänderungen

Alle 3 min wird zu jedem bekannten Frontend der gespeicherte Raum auf Belegungsänderungen geprüft (vgl. Abb. 31). Dazu wird zu Beginn ein HTTP-Request auf die Einstiegsseite des LSF abgesetzt, um eine Null-Sitzung[9] zu starten (vgl. Abb. 32). Das ist zwar nicht notwendig, aber es simuliert die tatsächliche Kommunikation durch einen Webbrowser mit dem LSF. Anschließend werden zur Authentisierung die Anmeldedaten an das LSF übermittelt. Das daraufhin erhaltene Sitzungs-Cookie wird anschließend zum Abruf der Raumbelegungen mit übermittelt. Enthält die Belegungsliste Änderungen gegenüber der letzten Prüfung, wird die neue Liste sowie ein Vermerk über die Aktualisierung gespeichert. Anschließend wird nach einer kurzen Verzögerung ein WOR-Signal an die entsprechende MCU herausgesendet.

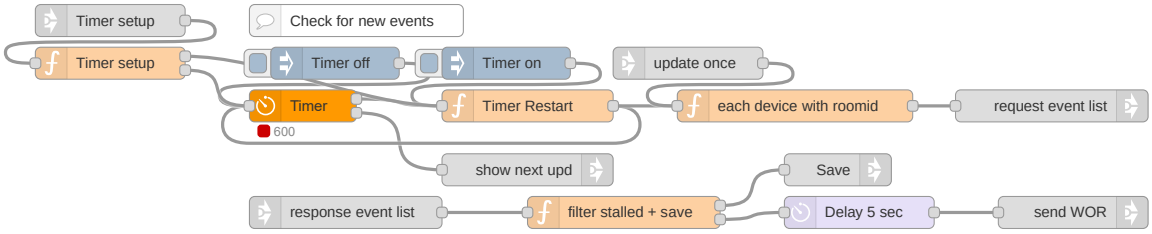


Abbildung 31: Prüfung auf verfügbare Belegungsänderungen, Node-RED-Flow

Das Extrahieren der Belegungsinformationen aus dem HTML-Quellcode erfolgt, wie im Abschnitt *Übersetzen von Raumname nach Raum-ID*, durch einen HTML-Parser. In dieser Liste fehlt jedoch jeweils die Information über den Dozierenden der Veranstaltung, da diese Information nur bei wenigen Veranstaltungen auf dem Display Platz finden würde.

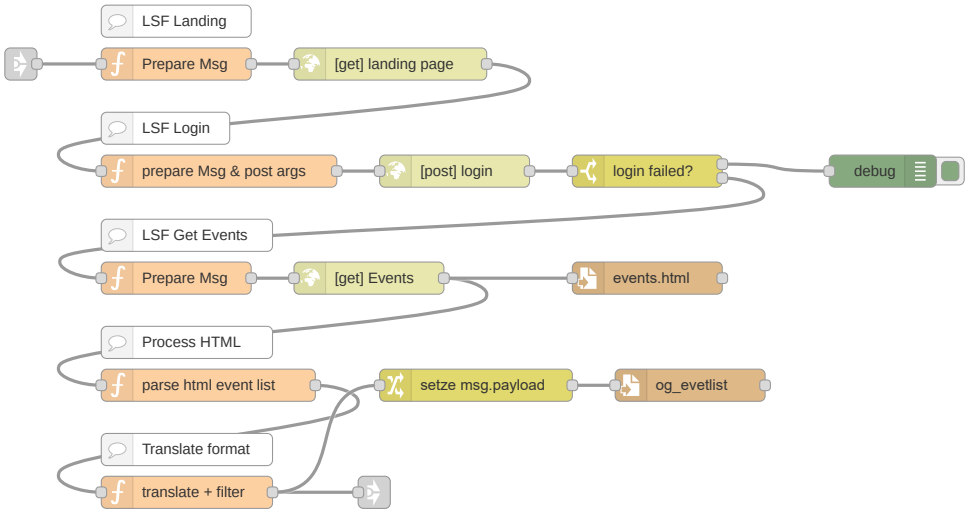


Abbildung 32: Abruf der Raumbelegungsliste, Node-RED-Flow

9 Eine hergestellte Verbindung zwischen Client und Server ohne Authentisierung des Clients

Übersetzung des Formats der Belegungen

Die Raumbelegungen werden von dem LSF sortiert nach Veranstaltungen in einer Tiefe von »drei« zurückgegben (vgl. Abb. 33). Jede Veranstaltung besteht aus mehreren Zügen bzw. Gruppen von Studierenden, die unterrichtet werden. Jeder dieser Züge besteht aus mindestens einem Termin, der durch `day_from` und `day_to` eine Zeitspanne angibt, die durch `cycle` in ihrer Wiederholung kontrolliert wird. Diese Informationen liegen im allgemeinen Format vor, das die Frage beantwortet, welche Veranstaltungen für einen bestimmten Raum für welche Züge an welchen Tagen des Semesters und zu welchen Uhrzeiten stattfinden.


```json
{
  "name": "AI (B) – Begrüßungsveranstaltung",
  "teams": [
    {
      "name": "1. Zug",
      "dates": [
        {
          "wd": "Dienstag",
          "time_from": "10:00",
          "time_to": "12:00",
          "cycle": "Einzelt.",
          "day_from": "01.10.2024",
          "day_to": "01.10.2024",
          "cancel": []
        }
      ]
    }
  ],
  "div": "Fachbereich 4: Informatik, Kommunikation und Wirtschaft"
}
```


Abbildung 33: Beispiel einer aus dem LSF extrahierten Veranstaltung mit Belegungsinformation

Dieses Format wird anschließend in ein zur Anzeige optimiertes Format übersetzt, das zu einer bestimmten Woche angibt, zu welchem Wochentag und zu welcher Uhrzeit eine Veranstaltung stattfindet (vgl. Abb. 34). Da das Ziel-Display in der Größe stark begrenzt ist, stehen pro angezeigtem Block nur wenige Zeilen Platz zur Verfügung, Informationen darzustellen. So enthält dieses Format zwar den Namen des Zuges – und kann um weitere Informationen angereichert werden – jedoch werden auf dem Ziel-Display lediglich in zwei Zeilen der Veranstaltungsname und darunter die Zeitspanne angezeigt (vgl. Abb. 12).

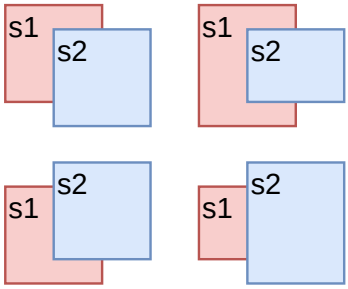

```json
{
  "name": "AI (B) - Begrüßungsveranstaltung",
  "team": "1. Zug",
  "time": [
    "10:00",
    "12:00"
  ],
  "wd": 1
}
```


Abbildung 34: Beispiel einer Raumbelegung im Zielformat

Zwischen den Zyklen der Wiederholung „Block“ und „Einzelt.“ konnte zum Stattfinden der Termine kein Unterschied erkannt werden. Während „Einzelt.“-Termine oft nur einen Tag umfassen (identisches Datum in den Feldern „day_from“ und „day_to“), finden „Block“-Termine oft an mehreren Tagen statt. In beiden Fällen spielt hier die Angabe der Zeitspanne die entscheidende Rolle.

Filterung nach dem Datum

Bei der Übernahme in das neue Format wurden die Termine der Veranstaltungen nach ihrem Stattfinden in der jeweiligen Woche gefiltert. Dazu wurde im ersten Schritt geprüft, ob die Zeitspannen der aktuellen Woche und die des Termins sich überschneiden. Das ist in genau zwei Situationen der Fall: wenn das Ende der ersten Zeitspanne zwischen Beginn und Ende der zweiten Zeitspanne liegt oder wenn das Ende der zweiten Zeitspanne zwischen Beginn und Ende der ersten Zeitspanne liegt (vgl. Abb. 35). Besteht keine Überlappung, wird dieser Termin ignoriert.



Zu betrachtende Fälle

```
55 function isDateSpansOverlap(span1, span2){
56   return (
57     span1.begin <= span2.begin
58     && span1.end >= span2.begin
59
60     || span1.begin >= span2.begin
61     && span1.begin <= span2.end
62   );
63 }
```

Implementierung in Node-RED

Abbildung 35: Prüfung auf sich überlappende Zeitbereiche

Daraufhin werden die Zyklen „ger. W.“ und „unger. W.“ der Wiederholung ausgewertet. Hat die aktuelle Kalenderwoche eine gerade Wochennummer, ist aber der Zyklus des Termins „unger. W.“ (oder umgekehrt), wird dieser Termin ignoriert.

Abschließend werden die Ausfallangaben berücksichtigt. Hierzu wird das angegebene Datum in ein Date-Objekt eingelesen und mit dem Datum des jeweiligen Wochentags verglichen. Bei Gleichheit wird dieser Termin ignoriert.

Alle anderen Termine werden in die finale Liste übernommen und gespeichert.

5.7 Firmware Aktualisierung

Um die Firmware zu aktualisieren, wurde das Projekt „Micropython OTA-Updates“ (OTA-Updater) eingebunden und modifiziert. Dieses Projekt erlaubt die Firmware-Aktualisierung einzelner mit Micropython bestückter MCUs über WLAN mit der Einbindung einer grundsätzlichen Code-Validierung. Dieses Projekt wurde u.a. um eine Möglichkeit erweitert, Firmware auf MCUs massenhaft auszurollen[10].

10 Alle vorgenommenen generellen Apassungen sind im Gitlab-Issue und PR-Request zu dem eingebundenen Projekt aufgeführt: https://gitlab.rz.htw-berlin.de/s0584373/micropython-ota-updates/-/issues/1

Funktionsweise

Steht eine neue Firmware zur Verfügung, kann ein entsprechender MCU über das Dashboard in den Firmware-Update-Modus versetzt werden. Dazu wird ein WOR-Signal an das Frontend geschickt, woraufhin die jeweilige MCU aufwacht und sich mit dem WLAN-AP und dem MQTT-Broker verbindet. Sobald die MCU das CommandList-Topic abonniert, bekommt sie den Befehl, den Firmware-Update-Modus zu starten (vgl. Abb. 36). Dies geschieht durch das Beenden des eigenen Programmflusses (vgl. Z. 173), da initial die Hauptroutine durch den OTA-Updater in der boot.py-Datei gestartet wurde. Da das Frontend über eine Batterie betrieben wird, startet die MCU nach 5 min neu und versetzt sich in den Deepsleep, wenn sie nicht innerhalb dieser Zeit einen Befehl von dem OTA-Updater erhalten hat.

```
169     if result == 'fwupdate':
170       global _timeout
171       _timeout = millis() + 5*60*1000
172       print('enter FW Update')
173       return
```

Abbildung 36: Starten des Firmware-Update Modus, main.py

Nach dem Start in den OTA-Update-Modus abonniert die MCU die Topics zum Erhalt von Befehlen und zum Erhalt der Firmware bei einem Firmware-Rollout. In diesem Modus können über das Dashboard Dateien im Speicher der MCU ausgelesen, bearbeitet, gelöscht und neu angelegt werden.

Über diesen Modus können auch Firmware-Dateien massenhaft ausgerollt werden. Über den Knopf „Firmware Bulk Rollout“ schaltet das Dashboard in ein UI um, in dem ein Firmware-Ordner hochgeladen werden kann. Nach einem Klick auf „Upload Directory“ wird das Verzeichnis vom Webbrowser zum Node-RED übertragen und zunächst für die Validierung über den MQTT-Broker propagiert (vgl. Abb. 37). Der Code-Checker empfängt den Code, führt diesen aus und erstellt pro Datei einen Bericht zur Wort-, Syntax-, Semantik- und Laufzeit-Prüfung. Diese Berichte werden über den MQTT-Broker propagiert und vom Node-RED als Response an den Webbrowser zurückgeleitet und dort angezeigt (vgl. Abb. 38). Die erste Spalte stellt die Namen bzw. Pfade der geprüften Dateien dar; die zweite Spalte gibt an, ob die geprüfte Datei den Anforderungen genügt hat; die dritte Spalte gibt im Fehlerfall die Compiler-Nachricht aus.

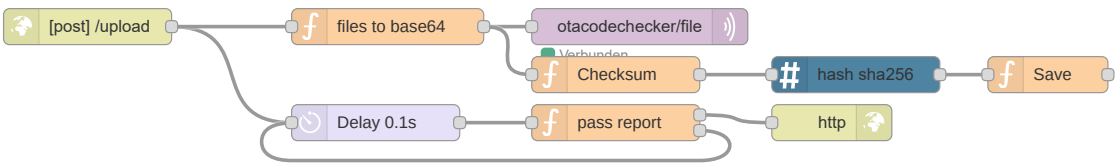


Abbildung 37: Upload eines Ordners zum massenhaften Firmware-Ausrollen, Node-RED-Flow

Name	Valid	Message
pymakr.conf	true	Syntax is correct
ota_update/simple.py	true	Syntax is correct
ota_update/helper/ wifi_manager.py	true	Syntax is correct
ota_update/helper/ status_helper.py	false	Import error: No module named 'ota_update'

Abbildung 38: Ergebnis einer Code-Validierung, Node-RED-Dashboard, Firmware Upload

Durch einen Klick auf den Knopf »Bulk Rollout« wird die Firmware zur Auslieferung an alle MCUs vorbereitet – unabhängig davon, ob eine Code-Prüfung durchgeführt wurde. Die Firmware wird erneut in base64-enkodierter Form über den MQTT-Broker propagiert, sodass die verbundenen MCUs diese herunterladen und in ihr Dateisystem kopieren können. Als Kodierung wurde base64 gewählt, da die MQTT-Bibliothek Schwierigkeiten hat, ASCII-Fremde Symbole zu übertragen und base64 von allen beteiligten Systemen implementiert wird.

Da die meisten Frontends sich im Deepsleep befinden werden, sind zu dem Zeitpunkt der Bereitstellung des Updates nur wenige Frontends online. Die Updates sollten ausgeliefert werden, sobald die MCUs sich mit dem MQTT-Broker verbinden und das entsprechende Topic abonnieren. Um dies zu realisieren, wurden die MQTT-Nachrichten mit einem Retain-Flag versehen, welches dafür sorgt, dass die Nachricht bzw. Firmware vom MQTT-Broker vorgehalten und ausgeliefert wird, sobald das Topic von einer MCU abonniert wird. Damit das gleiche Update nicht an die gleiche MCU mehrmals ausgeliefert wird, wird über die vollständige Firmware ein Hash gebildet (vgl. Abb. 37) und neben der Firmware propagiert. Die MCU speichert diesen Hash im ROM ab und prüft anhand dessen, ob sich bereits die aktuelle Firmware in dem Speicher der MCU befindet.

Sind die Hashes nicht identisch, ist die Firmware auf dem MCU veraltet und sie abonniert das Topic der ersten Firmware-Datei. Ist die Datei vollständig geschrieben, wird das Topic der nächsten Datei abonniert. Dies wird für alle Dateien wiederholt, die mit ihrem Dateipfad in den Metainformationen aufgetaucht sind. Sobald alle Dateien vollständig übertragen und in das Dateisystem geschrieben wurden, wird der Hash der neuen Firmware übernommen.

Speicherproblem

Die Firmware war in ihrer aktuellen Version mit 30 Dateien nach base64-Enkodierung ca. 190 KB groß. Durch die Dekodierung auf der MCU wurden zusätzlich 190 KB zusammenhängender Speicherplatz im Heap-Speicher benötigt. Mit der Verwendung von Micropython stehen aber nur etwa 145 KB freier Heap-Speicher der ESP32 MCU zur Verfügung. So wurde begonnen, jede Datei über ein eigenes MQTT-Topic für den Download zur Verfügung zu stellen. Die Metadatendatei, die die Information zur Anzahl der Dateien und den Hash über alle Dateien beinhaltete, wurde über ein eigenes Topic angeboten.

Die durchschnittliche Dateigröße der Firmware betrug (nach base64-Enkodierung) etwa 6 KB. Die größte der Dateien mit 70,5 KB war die Datei „htw_regular42.py“, die die HTW-Büro-Schriftart der Größe 42 pt für das EPD darstellt. Diese Datei alleine konnte aktualisiert werden. Zusammen mit den anderen Dateien trat jedoch ein Allozierungsfehler auf, da nach deren Verarbeitung der Heap-Speicher der MCU so stark fragmentiert ist, dass kein freier und ausreichend großer zusammenhängender Speicherbereich mehr zur Verfügung stand. An diesem Problem änderte sich nichts, als die Schriftart durch das Entfernen unbenutzter Zeichen auf 37 KB reduziert wurde.

Aufgrund der automatischen Speicherverwaltung der MCU durch Micropython beginnt bei der Allozierung größerer Speicherbereiche der Speicher stärker zu fragmentieren. So fragmentiert der Speicher auch stärker beim Empfangen größerer Dateien über MQTT, da hier zusätzlich deren Inhalte dekodiert werden müssen.

Um dieses Problem zu lösen, wäre normalerweise bei hardwarenaher Programmierung die ideale Herangehensweise gewesen, den benötigten Speicherplatz im Vorhinein zu allozieren und wiederzuverwenden. Jedoch abstrahiert Micropython die Speicherallozierung derart, dass dieses Vorgehen ohne weiteres nicht möglich ist.

Stattdessen wurde die MQTT-Bibliothek modifiziert (vgl. Abb. 39), sodass aus dem empfangenen Puffer höchstens 1024 Byte gleichzeitig gelesen werden (Z. 209, 214-215). Diese Zahl stellte eine Abwägung zwischen der Stärke der Fragmentierung des Heap-Speichers und der Dauer des Einlesens des Dateiinhalts dar. Zudem ist sie ein Vielfaches der Zahl 4, da die base64-Kodierung drei Byte Daten in vier Byte enkodiert. So ist es möglich, sukzessive den Dateiinhalt zu dekodieren und in die entsprechende Datei zu schreiben. Die ersten 1024 Byte wurden nach dem Auslesen dekodiert und in die entsprechende Datei überschrieben. Jedes weitere Byte wurde der Datei angehängt. Dieses Vorgehen hat das Speicherproblem behoben.

```
209     max_size = 1024 # base64, multiple of 4
210     print(f'> File size: {sz} byte')
211     print(f'Splitting up into {ceil(sz/max_size)} package(s)')
212     left_size = sz
213     while left_size > 0:
214       r = min(max_size, left_size)
215       msg = self.sock.read(r)
216       left_size -= r
217       self.cb(topic, msg, (r<max_size))
```

Abbildung 39: Modifikation der MQTT-Bibliothek, ota_update/simple.py

Problem bei der Datenübertragung über MQTT

Bei der Übertragung der Dateien trat ein weiteres Problem auf: Die Übertragung der dritten Datei endete zu früh und aus dem TCP-Socket der MQTT-Verbindung konnten keine verständlichen Informationen mehr gelesen werden.

Fälschlicherweise wurde zu früh das Topic der nächsten Datei abonniert, was dazu führte, dass der MQTT-Broker den Dateiinhalt an das Socket der MCU schickte, noch bevor das Socket vollständig ausgelesen wurde. Das führte dazu, dass der Empfangs-Puffer überlief und den Beginn erneut

beschrieb. Dieser Umstand wurde durch die Aufzeichnung und Auswertung des Netzwerkverkehrs erkannt.

5.8 Administrations-Dashboards

EPD Dashboard

Das Dashboard, worüber die einzelnen Frontends bzw. MCUs eingesehen und verwaltet werden können, ist in Abb. 40 dargestellt. Die Frontends, die dem System bekannt sind, werden in der oben abgebildeten Liste dargestellt. Darunter befinden sich Knöpfe, um diese Frontends zu steuern. Unterhalb der Knöpfe befindet sich eine Möglichkeit, das automatische Prüfen auf Aktualisierungen zu deaktivieren und das Prüf-Intervall einzustellen.

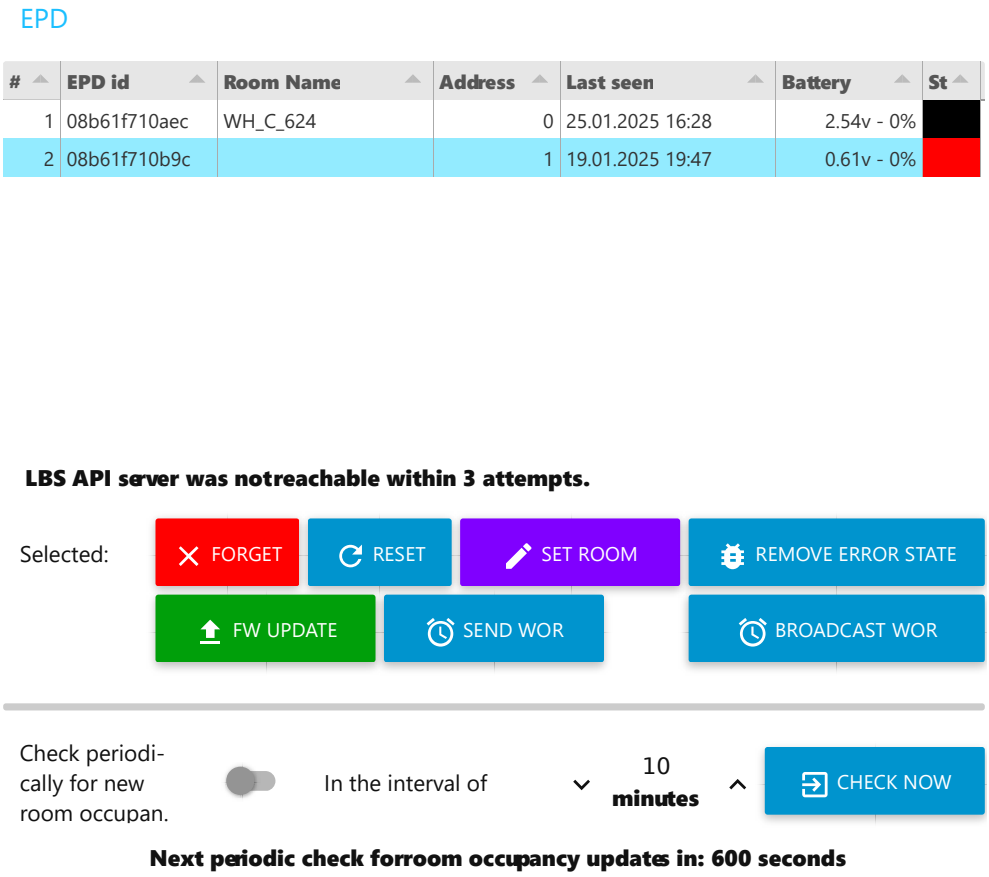


Abbildung 40: Oberfläche des Node-RED Dashboard

Ist keines der Frontends ausgewählt, sind alle Knöpfe (bis auf „Broadcast WOR“) deaktiviert. Der Knopf „Forget“ entfernt den ausgewählten Eintrag aus der Liste. Dadurch ist dieses Frontend dem Node-RED unbekannt. Verbindet sich nun dieses Frontend wieder, bekommt es einmalig die Aufforderung, eine neue Adresse seinem LoRa-Modul zuzuteilen sowie die in der Umgebung befindlichen WLAN-APs aufzuzeichnen.

Der Knopf „Reset“ entfernt den gewählten Eintrag und sendet zusätzlich ein WOR zu dem entsprechenden Frontend.

Der Knopf „Set Room“ dient dem Überschreiben der Raumzugehörigkeit. Durch den Knopf öffnet sich eine Dropdown-Liste, aus der der entsprechende Raum ausgewählt werden kann. Nach der Auswahl ändern sich der Raumname sowie die Raum-ID, sodass Node-RED zur nächsten Prüfung der Belegungsänderung die Informationen des neuen Raums abholt.

Der Knopf „Remove Error State“ leert den Fehlerspeicher des ausgewählten Eintrags. Der Inhalt des Fehlerspeichers des ausgewählten Eintrags wird unterhalb der Liste angezeigt.

Der Knopf „FW Update“ hinterlegt zum Frontend die Information, dass bei dem nächsten Start die MCU sich in den Firmware-Update-Modus schalten soll. Nach einer kurzen Verzögerung wird ein WOR an das Frontend gesendet. Das Dashboard schaltet dabei automatisch zur Update-Ansicht des OTA-Updaters um.

Während der Knopf „Send WOR“ ein WOR-Signal an das ausgewählte Frontend sendet, sendet der Knopf „Broadcast WOR“ ein WOR-Signal an alle Frontends, deren LoRa-Modul auf den korrekten Kanal lauschen. Dies kann bspw. verwendet werden, wenn aktive Frontends bestehen, die nicht in der Liste auftauchen.

Zur Anzeige der Frontend-Liste wurde ein „ui_table“-Node eingesetzt, welcher auf der JS-Bibliothek „Tabulator“ basiert. Die Einbindung in Node-RED bietet von Haus aus keine Möglichkeit, ausgewählte Einträge hervorzuheben, bzw. durch erneutes Anwählen die Auswahl wieder aufzuheben. Um dies zu ermöglichen, wurde eine „1“ in die Nachricht „ui_control“ für den Schlüssel „tabulator.selectable“ geschrieben und an den ui_table-Node gesendet. Zudem bestand auch keine Möglichkeit zu erkennen, ob durch den Klick ein Eintrag ausgewählt oder dessen Auswahl aufgehoben wurde. So wurde auf die gleiche Art und Weise je ein Callback für die Auswahl (vgl. Abb. 41) und das Aufheben der Auswahl bei dem Node registriert.

```
function (row) {
  this.send({
    ui_control: { callback: "rowselected", row: row.getData() }
  })
}
```

Abbildung 41: Node-Red UI-Table Auswahl Callback

Eine sich auf Knopfdruck öffnende Dropdown-Liste ist in Node-RED nicht vorgesehen. Dies wurde aber dadurch realisiert, dass dem Knopf eine eigene HTML-Klasse zugeordnet und in einem JS-Template für diese Klasse ein Eventlistener für das Klick-Event registriert wurde, bei dem ein Klick-Event auf das Dropdown-Steuerelement ausgelöst wird. Zudem wurde mit einer eigenen HTML-Klasse durch eine CSS-Regel das Dropdown-Steuerelement versteckt.

OTA-Update Dashboard

Im OTA-Dashboard sind alle verbundenen MCUs zu sehen. Über dieses Dashboard lassen sich auf den einzelnen MCUs Dateien neu anlegen, bearbeiten und löschen (vgl. Abb. 42). Das Dashboard

wurde um die Funktion der massenhaften Firmware-Aktualisierungen erweitert. Auf diese wird durch den Druck auf den Knopf »Firmware Bulk Rollout« umgeschaltet.

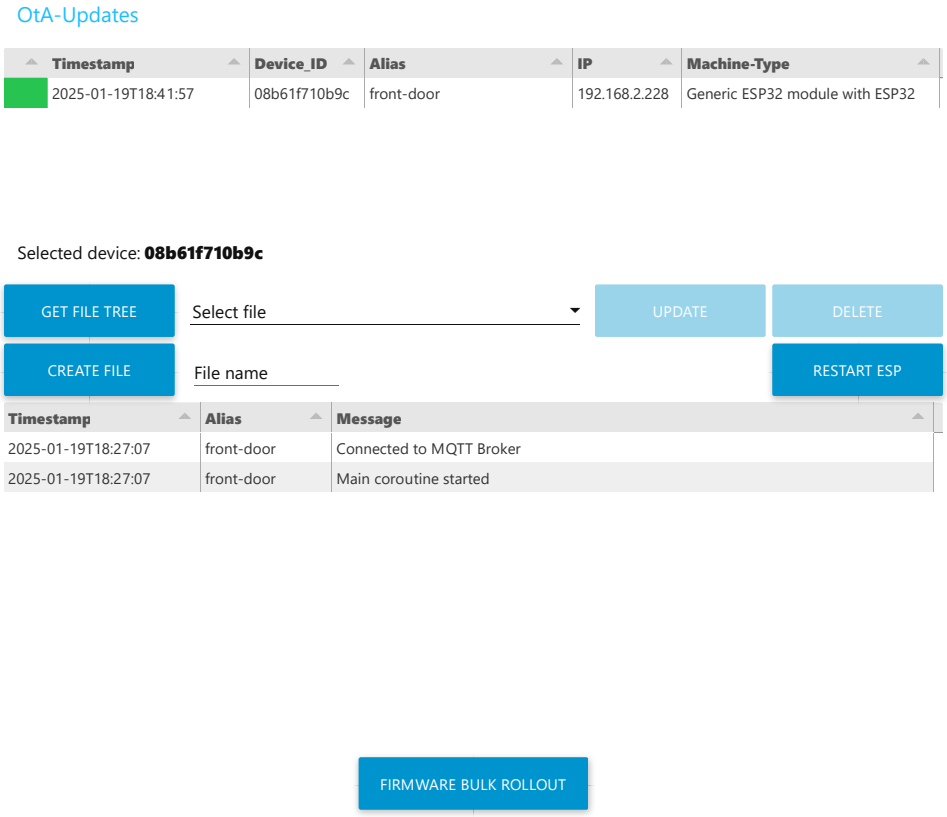


Abbildung 42: OTA-Update Dashboard

In dem UI für das massenhafte Ausrollen der Firmware (vgl. Abb. 43) kann ein Ordner mit Firmware-Dateien hochgeladen werden. Nach dem Hochladen werden diese direkt über den MQTT-Broker zur Code-Validierung weitergegeben und der Report von der Prüfung angezeigt.

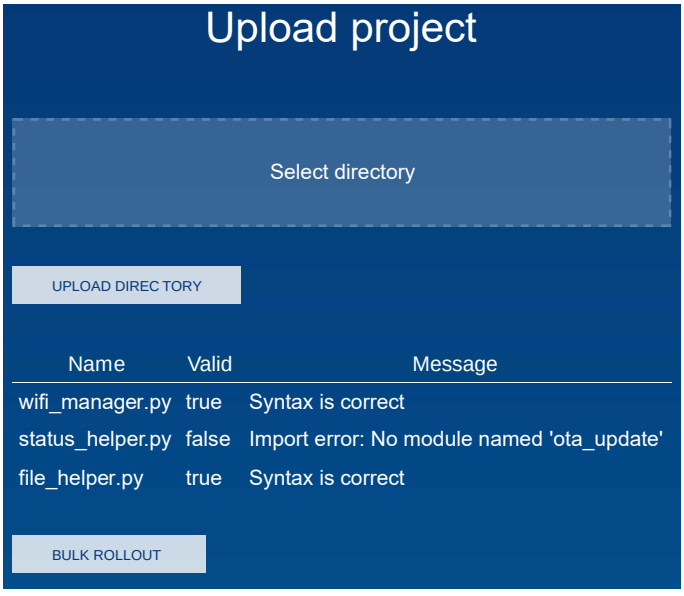


Abbildung 43: Firmware Bulk Rollout UI

Abschließend kann durch den Knopf »Bulk Rollout« die Firmware für das massenhafte Ausrollen freigegeben werden.

Speicher-Systematik

Da die Frontends nicht permanent online sind, muss das Backend alle notwendigen Informationen zu den Frontends vorhalten. Da sich mehrere Frontends gleichzeitig mit dem Backend verbinden und Befehle abrufen können, versuchen zu diesem Moment mehrere Flows gleichzeitig Informationen in der Device-List zu schreiben (z. B. Aktualisierung des Batterieladestands, Schreiben von Fehlermeldungen). So muss insb. der schreibende Zugriff auf die Frontendinformationen reglementiert sein, damit die Daten konsistent bleiben.

Um dies zu realisieren, wird ausschließlich ein Flow mit der Speicherung von Aktualisierungen in der Liste betraut (vgl. Abb. 44). Dieser Teil-Flow kann durch einen »Link-Out«-Node mit der Übergabe der Änderungen im Nachrichten-Objekt angesprochen werden. Durch eine Semaphore zum Beginn und zum Ende des kritischen Bereichs wird dafür gesorgt, dass ein Thread in seiner Ausführung an der Semaphore wartet, bis der kritische Bereich durch die zweite Semaphore wieder freigegeben ist.

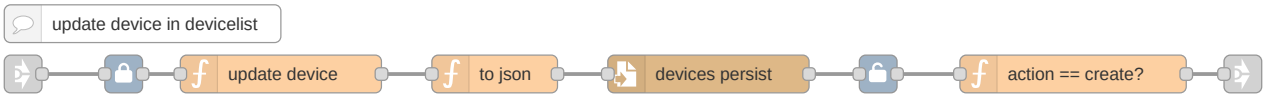


Abbildung 44: Persistierung der MCU Liste, Node-RED-Flow

5.9 Optimierung des Energiebedarfs

Da das Frontend durch eine eigene Sekundärzelle mit geringer Kapazität autark betrieben wird, musste der Energiebedarf zu jeder Zeit auf ein Minimum reduziert werden.

Energiebedarf während der Schlafphase

Der größte zeitliche Einfluss liegt in der Ruhephase, bei der das Frontend keine Operationen durchführt (Schlafphase). So sollte in erster Linie der Energiebedarf aller Komponenten während der Schlafphase auf ein Minimum reduziert werden. Da das EPD eine bistabile Anzeige ist, kann es bei Nichtbenutzung stromlos geschaltet werden, ohne dass die angezeigten Informationen verloren gehen. Das EPD und mit ihm der Treiber lassen sich stromlos schalten (bis auf vernachlässigbare Kriechströme), indem der PWR-Eingang des Treibers heruntergezogen wird. [14]

Die MCU ESP32 bietet für Ruhephasen eigene Modi an, die Komponenten der MCU deaktivieren und dadurch den Stromverbrauch deutlich reduzieren. Bei aktiver WLAN-Schnittstelle werden 100 - 240 mA Strom aufgenommen. Ist die Funk-Schnittstelle inaktiv, werden bei voller Last bis zu 68 mA Strom benötigt. Durch die Herunterregelung des Prozessortaktes lässt sich die Stromaufnahme auf 20 mA reduzieren. Zum Deepsleep-Modus der MCU sind alle Komponenten außer der RTC inaktiv. Die RTC muss aktiv bleiben, um sowohl die Modus-Ausgänge für das

LoRa-Modul zu halten, als auch auf Interrupts durch das LoRa-Modul zu lauschen. In diesem Modus nimmt die MCU noch 10 µA Strom auf. [1]

Die Periodendauer, mit der das LoRa-Modul auf eingehende WOR-Signale lauscht, kann durch die Einstellung eines WOR-Cycles modifiziert werden. Hier kann ab einer Periodendauer von 500 ms in 500-ms-Schritten die Periodendauer auf 4 s eingestellt werden. Bei einer Aussteuerung von 22 ms entspricht das somit einer Tastrate von 0,55 % (vgl. Kapitel *4.2 - Auswahl der Komponenten*, Abschnitt *Wakeup über LoRa-WAN*). [15]

Wird das Frontend ohne Wakeup-Empfänger betrieben, gibt es die Möglichkeit, auf den Polling-Betrieb umzuschalten. Hier lässt sich die Zeit bis zur nächsten Wach-Phase in der MCU durch die Angabe von Uhrzeiten und Wochentagen konfigurieren, zu denen das Frontend aufwachen und auf Raumbelegungsänderungen prüfen soll.

Energiebedarf zur Wachphase

Das Frontend mit aktiver WLAN-Schnittstelle der MCU hat zur Phase der (Prüfung auf die) Aktualisierung in der Raumbelegung (Wachphase) die größte Stromaufnahme. Da diese Phase jedoch nur einen Bruchteil der gesamten Zyklusdauer darstellt, hat diese Phase zeitlich die geringste Auswirkung auf den gesamten Strombedarf. Der Tastgrad beträgt bei einer einmaligen Aktualisierung pro Woche bei einer Dauer der Aussteuerung von 22 s $3{,}63 \cdot 10^{-5}$. Jedoch limitiert der Energiebedarf zur Wachphase des Frontends die Anzahl an Prüfungen auf Belegungsänderungen bzw. die Anzahl der Aktualisierungen der Anzeige.

So wird nach der Prüfung bzw. nach dem Empfang neuer Belegungsinformationen die MCU zurückgesetzt und bei deaktivierter Funkschnittstelle das zu zeichnende Bild gerendert und zur Anzeige auf das EPD übertragen.

5.10 Konfiguration

Durch Konfigurationsdateien auf der MCU kann das Frontend konfiguriert werden. Hier lassen sich alle wichtigen Parameter, wie Zugangsdaten zu dem zu verwendenden WLAN-AP, IP-Adressen bei statischer IP-Konfiguration, MQTT-Topic-Namen, Anschluss-Pins des LoRa-Moduls und des EPDs sowie Uhrzeiten für die Schlaf- und Wachphasen, anpassen.

Nachfolgend werden beispielhaft die Konfigurationsmöglichkeiten der Ansteuerung des LoRa-Moduls und der Dauer der Schlafphasen betrachtet:

LoRa-Modul

Das LoRa-Modul verfügt über zwei Modus-Eingänge und einen Status-Ausgang, die über die Angabe der Hardware-Pins auf Seiten des ESP32 ausgewählt werden können (vgl. Abb. 45). Zudem verfügt das LoRa-Modul über eine UART-Schnittstelle, um Konfigurationen vornehmen und Daten an andere Module senden zu können. Hier wird die ID und die Symbolrate in s der UART-Schnittstelle festgelegt, die über einen TX- und einen RX-Pin verfügt. Mit „ID 2“ werden die physischen Pins 16 und 17 genutzt.

```
62 lora = {
63   'enabled': True,
64   'uart': {
65     'id': 2,
66     'baud': 9_600
67   },
68   'm0': 33,
69   'm1': 32,
70   'aux': 4
70 }
```

Abbildung 45: Konfiguration zur Ansteuerung des LoRa-Moduls, config.py

Dauer der Schlafphase

Im Fall der Schlaf- und Wach-Phasen können die Wochentage und Uhrzeiten in UTC eingestellt werden (vgl. Abb. 46). Dazu werden die Wochentage mit nullbasierter Nummerierung, startend mit Montag, und die Uhrzeiten durch Stundennummern angegeben. Wird der MCU durch den Node-RED die aktuelle Uhrzeit mitgeteilt, berechnet sie die Dauer, bis wann sie schlafen muss, um zum nächsten Aufwach-Termin aufzuwachen.

```
89 sleepwakecycle = {
90   'enabled'    : True,
91   'enabledwdays': [0, 1, 2, 3, 4],
92   'enabledhours': [7, 12, 15]
93 }
```

Abbildung 46: Einstellung der Schlaf- und Wachphasen der MCU, config.py

Zur Berechnung der Stunden wird zunächst geprüft, ob an dem aktuellen Tag eine Stunde existiert, die noch nicht erreicht wurde (vgl. Abb. 47, Z. 131, 132). Wurde eine Stunde ermittelt (Z. 132), wird die Differenz zwischen der aktuellen und der gefundenen Stunde erfasst (wHour). Konnte keine Stunde ermittelt werden, wird die Differenz zum nächsten Tag der ersten Stunde in der Liste ermittelt (Z. 136). Wurde gesprungen, wird der aktuelle Kalendertag um eins erhöht (Z. 142). Auf diese Weise wird die Information über den Sprung an die Berechnung der Tage weitergegeben.

```
129   hourRollover = False
130   wHour = 0
131   for r in config.sleepwakecycle['enabledhours']:
132     if now[3] < r:
133       wHour = r - now[3]
134       break
135   else:
136     wHour = 24-now[3]+config.sleepwakecycle['enabledhours'][0]
137     hourRollover = True
138   wHour -= 1
139
140   now6 = now[6]
141   if hourRollover:
142     now6 += 1
```

Abbildung 47: Berechnung der Schlafzeit zur nächsten Wachphase – Stunden, main.py

Auf die gleiche Weise wird auch die Differenz in Tagen zum nächsten Tag in der Liste ermittelt (wwd, vgl. Abb. 48, Z. 145-148). Wurde kein Tag ermittelt, wird unter Berücksichtigung des möglicherweise erhöhten Kalendertages die Differenz zum Start der nächsten Woche ermittelt (Z. 150). Anschließend wird die ermittelte Stundenanzahl (wHour) auf die ermittelte Tagesanzahl (wwd) aufaddiert und um die Restminuten und -sekunden der aktuellen Uhrzeit erweitert.

```
144   wwd = 0
145   for r in config.sleepwakecycle['enabledwdays']:
146     if now6 <= r:
147       wwd = r - now6
148       break
149   else:
150     wwd = 7-now6+config.sleepwakecycle['enabledwdays'][0]
151
152   sleepseconds = (60-now[5]) + (60 - now[4])*60 + wHour*3600 + wwd*3600*24
153   return sleepseconds
```

Abbildung 48: Berechnung der Schlafzeit zur nächsten Wachphase – Tage, main.py

6 Validierung

6.1 Stromaufnahme

Erwartete Stromaufnahme – Schlafphase

Im Folgenden wird die benötigte Ladungsmenge für das LoRa-Modul berechnet.

Die Dokumentation des LoRa-Moduls gibt den regelmäßigen Stromverbrauch beim Lauschen auf eingehende Signale im „Normal“-Modus mit 17,2 mA und zum Konfigurationsmodus mit 5 µA an. Jedoch ist unbekannt, welcher Stromverbrauch während der Aussteuerung sowie welcher Tastgrad im WOR-Modus gilt. [15]

Es wurde angenommen, dass während der Aussteuerung das LoRa-Modul auch 17,2 mA Strom und während der anderen Zeit der Periode 5 µA Strom aufnimmt. Weiterhin wurde eine Impulsdauer von einem Zehntel der geringsten Periodendauer von 500 ms angenommen. Dies entspricht bei einer Periodendauer von 4 s und einem Tastgrad von 1,25 % einer regelmäßigen Stromaufnahme von 220 µA (vgl. Tab. 1).

	Zeit	Strom	Ladung
Impulsdauer	0,05 s	17,200 mA	0,860 mAs
Ruhedauer (Tastgrad 1,25 %)	3,95 s	0,005 mA	0,020 mAs
Ladungsmenge über 4 s			0,880 mAs
Ladungsmenge über 1 s			0,220 mAs
Ladungsmenge über 1 y			1926,653 mAh

Tabelle 1: Erwartete Stromaufnahme zur Schlafphase

Während der ESP32 im Deepsleep-Modus 10 µA Strom aufnimmt, beträgt die Stromaufnahme des Spannungsreglers des Entwicklungsboards im Ruhezustand 55 µA.

Somit beziehen alle aktiven Komponenten während der Schlafphase des Frontends einen Strom von 285 µA bzw. eine Ladungsmenge von rund 2,50 Ah in einem Jahr.

Erwartete Stromaufnahme – Wachphase

Zur Wachphase des Frontends sind der ESP32 und das EPD mit seinem Treiber aktiv. Aufgrund des geringen Stromverbrauchs des LoRa-Moduls wird das Modul in dieser Rechnung nicht berücksichtigt. Die angegebenen Zeiten zum Senden (TX) und Empfangen (RX) des ESP32 wurden geschätzt. Die Zeit, während das EPD aktiv ist, wurde dem Datenblatt entnommen.

Zur Wachphase bei aktivem WLAN-Modul bei voller Sendeleistung werden von dem ESP32 240 mA Strom und zum Empfang etwa 100 mA Strom aufgenommen. Da zu einer Wachphase über WLAN kurz und nur wenige Informationen gesendet werden, wird diese Stromaufnahme für 0,5 s

angenommen. Zum weiteren Empfang von Informationen wird eine Dauer von 5 s angenommen. [1], [13], [16]

Das EPD nimmt im Betrieb bei 3,3 V eine Leistung von 21,78 mW auf, was einem Strom von 6,6 mA entspricht. Der EPD-Treiber betreibt das EPD zusätzlich mit einem eigenen Längsspannungsregler. Der Spannungsregler benötigt einen Ruhestrom von 130 µA. [12], [14]

$$n = \frac{P_{out}}{P_{\text{in}}} = \frac{3{,}3\,V \cdot 6{,}6\,mA}{3{,}7\,V\left(6{,}6\,mA + 130\,\mu A\right)} = \frac{21{,}78\,mW}{24{,}901\,mW} \approx 87{,}47\,\%$$

Bei einer Batteriespannung von 3,7 V entspricht das einem Wirkungsgrad von etwa 87 %. Somit nimmt der Spannungsregler ca. 0,95 mA Strom für den eigenen Betrieb und in Summe 7,6 mA Strom auf.

Zur Wachphase mit einer Dauer von annahmeweise 20,5 s wird eine Ladung von ca. 734 mAs aufgenommen (vgl. Tab. 2).

Wachphase	Zeit	Strom	Ladung
ESP32 TX	0,5 s	240,0 mA	120 mAs
ESP32 RX	5,0 s	100,0 mA	500 mAs
EPD + HAT	15,0 s	7,6 mA	114 mAs
Ladungsmenge über 20,5 s (1 Zyklus)			734 mAs

Tabelle 2: Erwartete Stromaufnahme zur Wachphase

Erwartete Stromaufnahme in Summe

Wird in der Raumplanansicht jede Woche eine Belegungsinformation geändert (darin enthalten sind auch Veranstaltungen, die turnusweise stattfinden), entspricht das pro Jahr 52 Wachphasen. Das bedeutet eine jährliche Stromaufnahme von ca. 2507 mAh. Über die Dauer eines Jahres muss die Selbstentladung einer LiPo-Zelle mit 12 - 25 % mit berücksichtigt werden.

Gemessene Stromaufnahme – Schlafphase

Eine von 100 mV dargestellte Spannung entspricht 1 A Stromfluss in den dargestellten Messwerten in den Abbildungen in diesem Abschnitt (vgl. Abb. 49 - 53).

Im Ruhezustand des Frontends befindet sich die MCU selbst im Deepsleep-Modus. Zu dieser Zeit ist das EPD ausgeschaltet und das LoRa-Modul lauscht regelmäßig auf ein eingehendes WOR-Signal.

Innerhalb von 4 s ist das LoRa-Modul für 22 ms aktiv am Lauschen (vgl. Abb. 49). Das entspricht einem Tastgrad von 0,55 %. Während der Aussteuerung nimmt das gesamte Frontend knapp 13 mA Strom auf, die verbleibende Periodendauer etwa 150 µA. Das entspricht einer regelmäßigen, zusammengesetzten Stromaufnahme von rund 221 µA bzw. einer jährlichen Ladungsaufnahme von 1,933 Ah (vgl. Tab. 3). Bei der gemessenen Ladung spielt das LoRa-Modul zur Impulsdauer des WOR-Zyklus mit 286 µAs eine deutlich geringere Rolle, als anfangs angenommen.

	Zeit	Strom	Ladung
Impulsdauer	0,022 s	13 mA	0,286 mAs
Ruhedauer (Tastgrad 0,55 %)	3,978 s	0,15 mA	0,597 mAs
Ladungsmenge über 4 s			0,883 mAs
Ladungsmenge über 1 s			0,221 mAs
Ladungsmenge über 1 y			1933,113 mAh

Tabelle 3: Gemessene Stromaufnahme zur Schlafphase

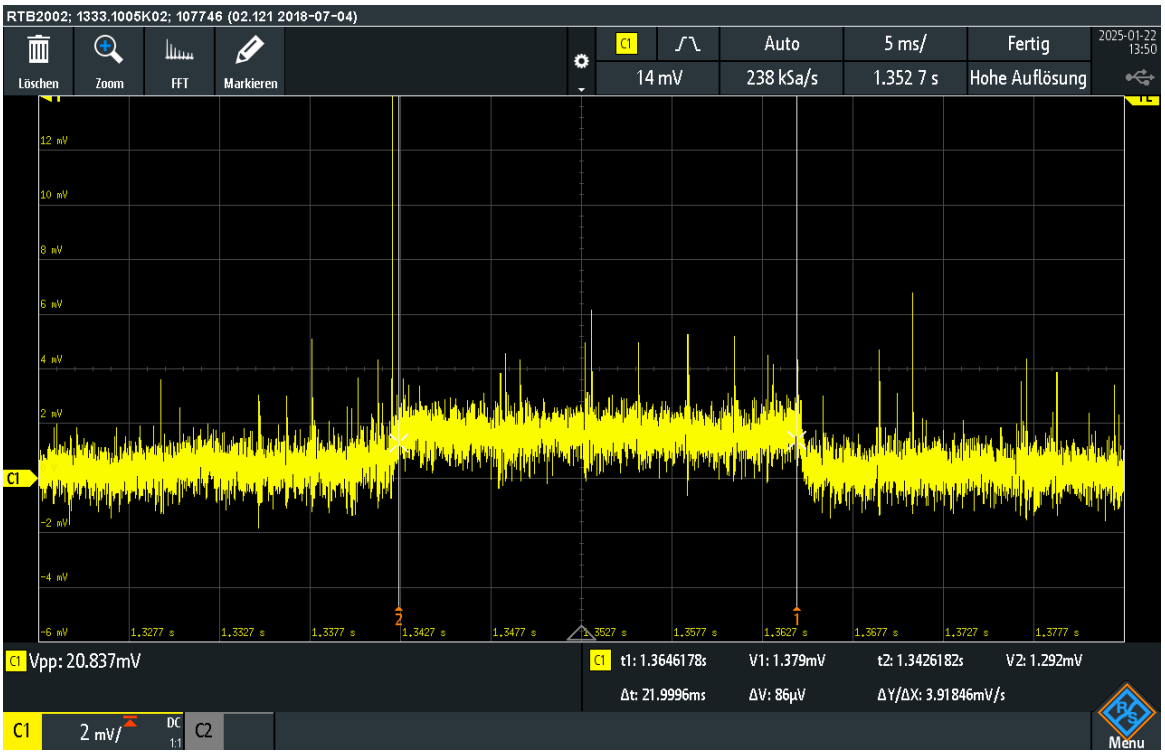


Abbildung 49: Stromaufnahme im Deepsleep während der Aussteuerung des LoRa-Moduls

Gemessene Stromaufnahme – Wachphase

Zum Prüfen und Abrufen von neuen Belegungsinformationen werden für eine Dauer von 11 s 67 mA Strom bzw. eine Ladung von 737 mAs aufgenommen (vgl. Abb. 50, Tab. 4). Dazu wird in den ersten 1,5 s bei 40 mA die MCU initialisiert. Anschließend wird über 2 s lang, bei einer Aufnahme von 115 mA, eine WLAN-Verbindung mit einem AP hergestellt und die CommandList über MQTT angefordert. In den letzten 7,5 s empfängt die MCU bei 60 mA die CommandList und versetzt sich (im Fall der leeren CommandList) nach 3 s Wartezeit in den Deepsleep-Modus.

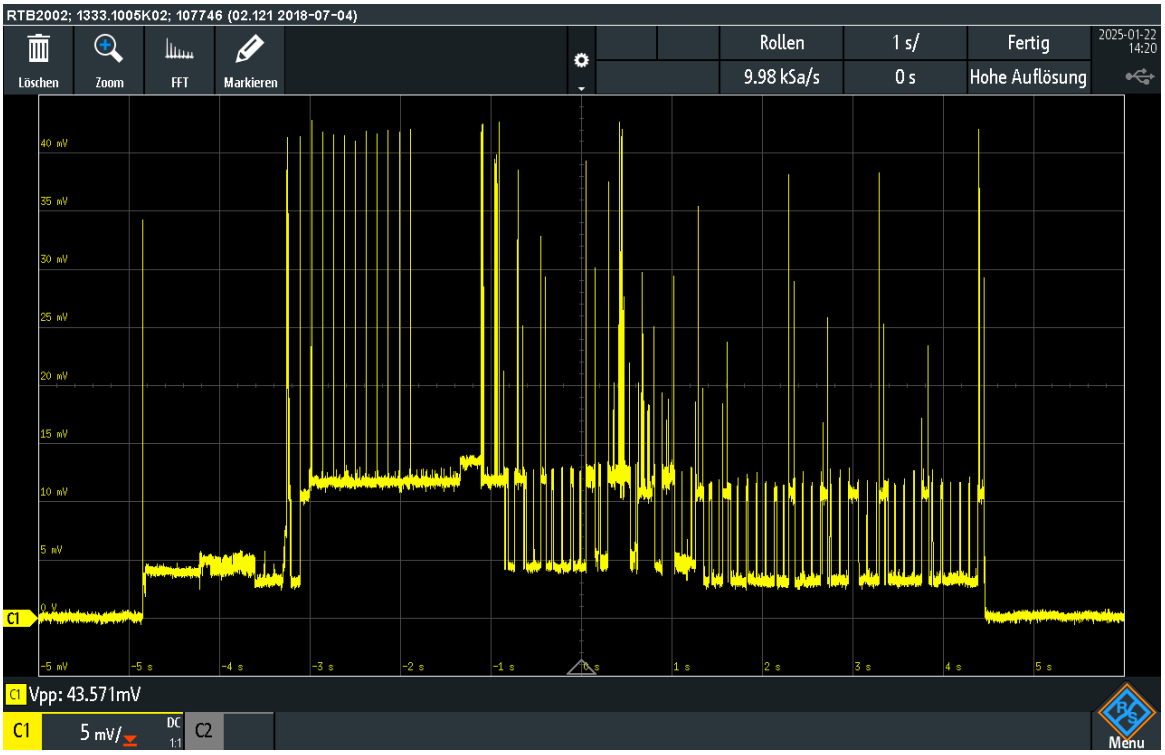
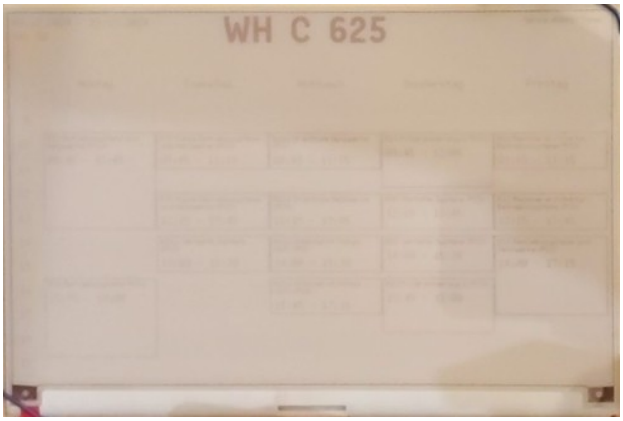


Abbildung 50: Stromaufnahme während der Abfrage neuer Belegungsinformationen

Während der Aktualisierung des EPD werden ca. 50 mA Strom aufgenommen (vgl. Abb. 53). Zur ersten Phase der Aktualisierung wird das Bild gelöscht (vgl. Abb. 51 und 53). An dem Raumnamen ist das Geisterbild zu erkennen (vgl. Abb. 51).

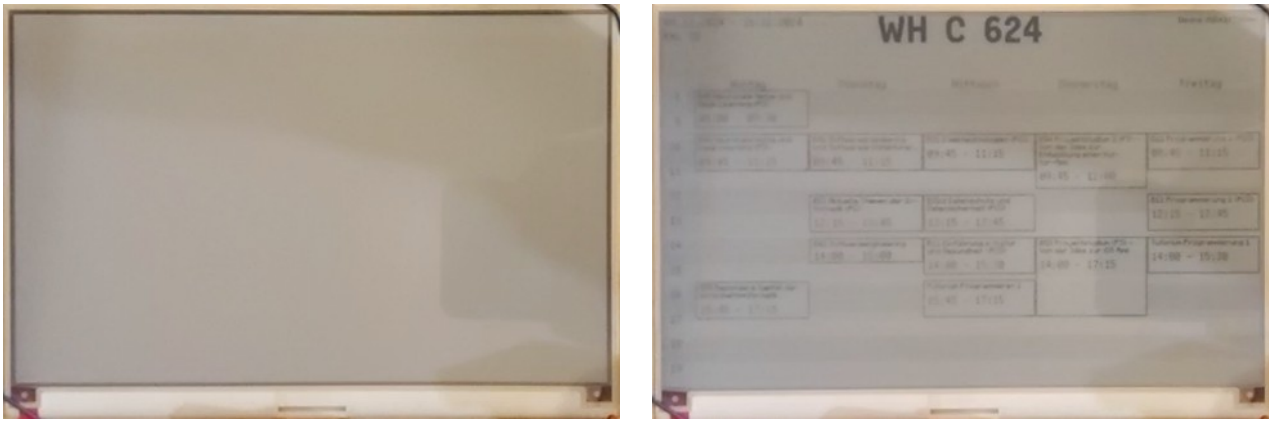


Phase 1.1

Phase 1.2

Abbildung 51: Phase 1 der EPD-Aktualisierung

Zur zweiten Phase flackert das Display, indem es mehrmals schnell hintereinander abdunkelt und aufhellt (vgl. Abb. 52 und 53). Zur dritten Phase wird das finale Bild erkennbar. Mit zunehmender Zeit nimmt der Kontrast zu – ähnlich wie der Analogfotografie bei dem Entwickeln eines Bildes (vgl. Abb. 52 und 53).



Phase 2 *Phase 3*

Abbildung 52: Phasen 2 und 3 der EPD-Aktualisierung

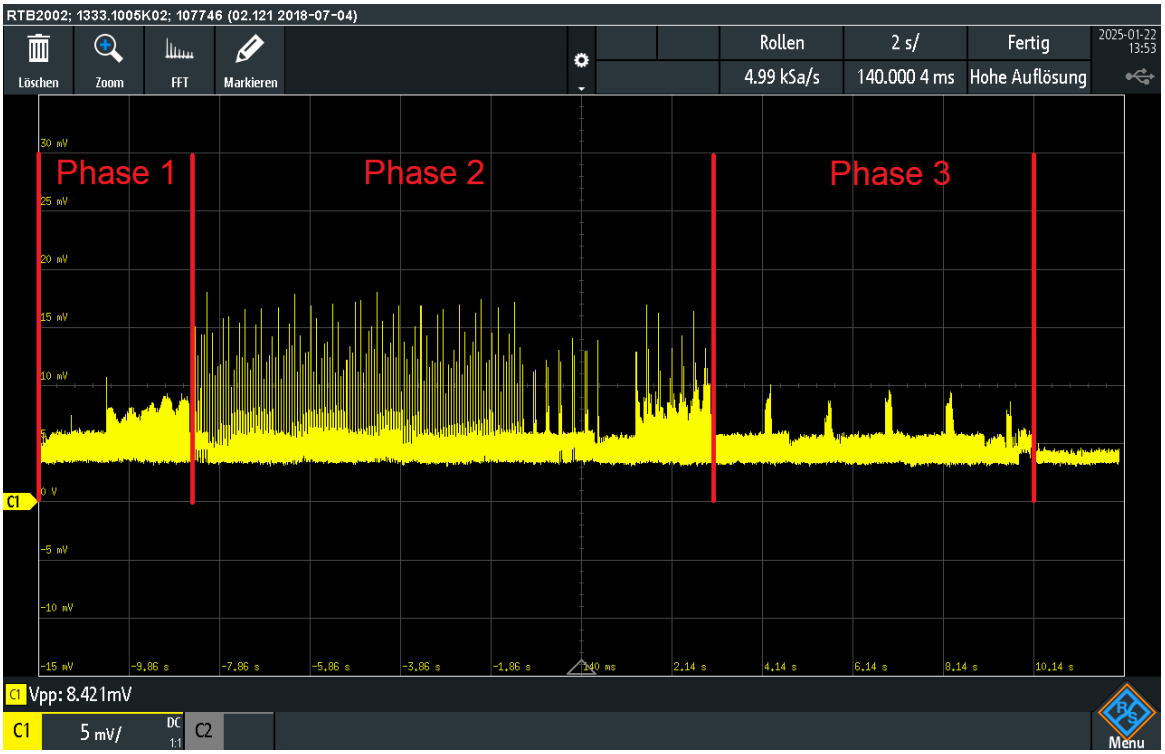


Abbildung 53: Stromaufnahme während der Aktualisierung des EPD

	Zeit	Strom	Ladung
Prüfung	11 s	67 mA	737 mAs
Anzeige	22 s	50 mA	1100 mAs

Ladungsmenge 1 Prüfung	1837 mAs
Ladungsmenge 52 Prüfungen	95524 mAs
Ladungsmenge 52 Prüfungen in mAh	26,53 mAh

Tabelle 4: Gemessene Stromaufnahme zur Wachphase

Gemessene Stromaufnahme in Summe

Bei einer Anzahl von 52 Wachphasen innerhalb eines Jahres wird eine Ladung von etwa 1940 mAh aufgenommen.

Die benötigte, gemessene und hochgerechnete Ladung für 1 Jahr bei 52 Wachphasen liegt mit 1,96 Ah, im Vergleich zum erwarteten Ladungsverbrauch von 2,5 Ah pro Jahr, deutlich niedriger. So kann unter Berücksichtigung der Selbstentladung mit einer Zelle mit 2,5 Ah Kapazität das Display über ein Jahr problemlos betrieben werden.

Jedoch ließe sich durch das Entfernen des LoRa-Moduls der Stromverbrauch um ein Drittel reduzieren. Würde zum Ausgleich die Häufigkeit des Aufwachens und Abrufens auf Aktualisierungen (durch einen Timer) auf fünfmal die Woche erhöht werden, läge der Stromverbrauch bei etwa 85 %.

6.2 Angezeigte Informationen

Auf dem EPD werden der Raumname, die Informationen zur aktuellen Woche und die Belegungsübersicht angezeigt (vgl. Abb. 12). Die Belegungsübersicht wurde oberhalb um die Angabe von Wochentage und links um Stunden erweitert.

Zu den einzelnen Belegungen werden der Veranstaltungsname sowie die Zeitspanne der Veranstaltung angezeigt (vgl. Abb. 54). Aufgrund des relativ kleinen Displays würden die Namen der Züge, die unterrichtet werden sollen sowie die der Dozierenden zusammen mit dem Veranstaltungsnamen nur in wenigen Fällen in den vorgesehenen Bereich passen.

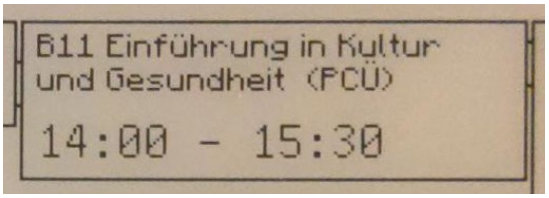


Abbildung 54: Dargestellte Informationen pro Veranstaltung auf dem EPD

6.3 Gegenüberstellung beider Systeme

Durch den Einsatz des WOR-Empfängers sind die angeschlossenen Frontends in ihrer Anzeige jeweils nur wenige Minuten nicht aktuell (je nach Taktung des Abrufs von Belegungsänderungen im Backend, vgl. Kapitel *5.8 - Administrations-Dashboards*, Abschnitt *EPD Dashboard*), wenn eine Belegungsänderung vorgenommen wird. Dadurch sind die angezeigten Informationen öfter aktuell als bei dem alten System. Das alte System erfordert, dass die Belegungsänderungen je Raum ausgedruckt und manuell an dem Türschild ausgetauscht werden.

Für den regulären Betrieb des Frontends und der Backend-Infrastruktur wird kein Personal benötigt, außer für die Inbetriebnahme und Wartung (bei der Installation an der Wand am jeweiligen Raum, bei dem Austausch bei Defekt oder leerer Batterie sowie bei der manuellen Änderung der Raumnummer). Ist der Ladestand einer Batterie im kritischen Bereich, wird eine Push-

Benachrichtigung herausgesendet. Somit muss zum Normalbetrieb das Administrations-Dashboard nicht überwacht werden.

Jedoch ist der Material-, Verarbeitungs- und Energieaufwand deutlich größer. So belaufen sich bereits die Kosten der Komponenten und des Materials eines Frontends auf über 90,00 EUR (vgl. Tab. 5), zzgl. Versand- und Produktionskosten. Zu den Betriebskosten zählen die Energie des Frontends (Laden der Batterie), der Betrieb des Backends und die o. g. Personalaufwendungen.

EPD; 800×480 Pixel; 7,5 Zoll	46,00 EUR
EPD Treiber HAT	9,60 EUR
LiPo-Batterie; 1400 mAh	5,00 EUR
ESP32 LiPo DevKit	7,45 EUR
LoRa-Modul E220 400T30D	7,00 EUR
Aluminiumprofil, Front- und Rückplatte	15,00 EUR
Σ	90,05 EUR

Tabelle 5: Kostenaufstellung der Komponenten und des Materials des Frontends

7 Fazit

7.1 Ergebnisse

An der Hochschule für Technik und Wirtschaft Berlin sollte der Prozess der Aktualisierung der Anzeige von Raumbelegungen an den Räumen optimiert werden. So sollte eine digitale Raumbelegungsanzeige implementiert werden, die die derzeit notwendigen personellen Ressourcen reduziert und früher zu einer aktuelleren Anzeige führt.

Dabei ist ein System aus mehreren Komponenten entstanden, welches permanent das hochschulinterne Raumbelegungs-Werkzeug auf Neuerungen bzgl. der Raumbelegungen überwacht und im Bedarfsfall auf einem EPD als bistabiler Anzeige, die das ursprüngliche System der Anzeige von Raumbelegungen an den Räumen ersetzt, die neuen Informationen darstellt.

Das neue System besteht an den Räumen jeweils aus einer MCU, die über ein zusätzliches LoRa-Modul ein Signal empfangen kann, das den Abruf von neuen Raumbelegungsinformationen auslöst (vgl. Abb. 55). Diese Informationen und das Signal kommen aus einem Backend, welches die Rauminformationen vorhält und Oberflächen zur Administration bietet. Über diese Oberflächen können die Raumschilder konfiguriert und deren Firmware aktualisiert werden. Die Kommunikation wird über das hiesige LAN durch einen MQTT-Broker realisiert, der alle Komponenten miteinander logisch verbindet.

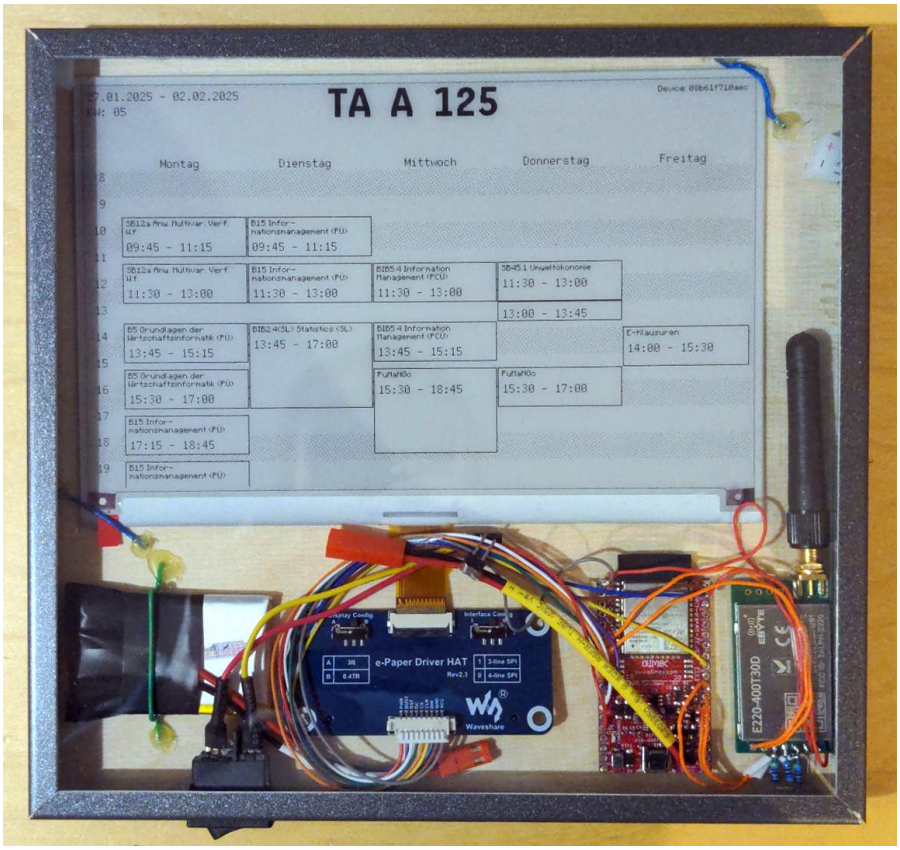


Abbildung 55: Der entstandene Prototyp

Durch RSSI Location Fingerprinting wird anhand einer bereits bestehenden API der zugehörige Raum automatisiert erfasst. Durch die Verwendung von Location Fingerprinting und einer Batterie zur autarken Stromversorgung gestaltet sich der Austausch unkompliziert.

Basierend auf Hochrechnungen reicht die Kapazität der eingesetzten Batterie ein Semester. So sollte zum produktiven Einsatz eine Batterie mit mind. 2700 mAh eingesetzt werden.

Das entstandene System kann unter verhältnismäßig geringem Personalaufwand betrieben werden und zeigt innerhalb weniger Minuten neue Raumbelegungen an. Das entstandene System ist in der Lage, das alte System abzulösen.

7.2 Limitationen und Ausblick

Verbesserung der Anzeigenlogik der Informationen

Aufgrund statischer Designentscheidungen zeigt das EPD nur Veranstaltungen, die zwischen Montag und Freitag zwischen 8:00 Uhr und 20:00 Uhr stattfinden, an. Zudem werden manche Veranstaltungsnamen aufgrund ihrer zu großen Länge nicht vollständig dargestellt. Darüber hinaus werden derzeit auch keine Dozierenden-Informationen angezeigt.

Hier besteht die Möglichkeit, die vollständige Woche darzustellen. Auch besteht die Möglichkeit, Samstage und Sonntage nur bei Bedarf anzuzeigen. Beide Möglichkeiten würden jedoch das Problem der Platzknappheit für die Veranstaltungsnamen verschärfen. Dies wiederum ließe sich dadurch verbessern, nur belegte Wochentage darzustellen oder auch Wochentage mit kurzen Veranstaltungsnamen zu verschmälern.

Die gleiche Logik lässt sich auch vertikal anwenden: Der Anfangs- und End-Bereich, zu denen keine Veranstaltungen stattfinden, könnten dynamisch ausgeblendet werden, während Stunden, zu denen längere Veranstaltungsnamen dargestellt werden müssten, erhöht würden.

Um mehr Text auf der Anzeige darstellen zu können, wurde eine einfache, aber fehleranfällige Variante der Silbentrennung implementiert. Die Silbentrennung könnte durch weitere Regeln verfeinert und damit verbessert werden. Dadurch könnten längere Veranstaltungsnamen besser angezeigt werden.

Sich überschneidende Veranstaltungen

In den Raumplänen des LSF kann es passieren, dass für den gleichen Tag, für die gleiche Uhrzeit, mehrere Veranstaltungen für einen Raum eingetragen werden (Überbelegung). Das LSF löst dieses Problem, indem die zweite Veranstaltung unter der ersten angezeigt wird.

Überbelegungen auf dem EPD werden derzeit nicht kenntlich gemacht. Stattdessen werden nacheinander beide Veranstaltungsinformationen auf den selben Bildpunkten dargestellt. Hierdurch wird die zuerst gezeichnete Veranstaltung verdeckt.

Zur Verbesserung könnten beide Veranstaltungen übereinander oder nebeneinander dargestellt werden. Das kann jedoch das bisherige Platzproblem verschärfen.

Schnelleres Reagieren auf Änderungen der Raumbelegung

Aufgrund der Verarbeitungslogik werden einerseits Änderungen an Raumbelegungen nicht direkt zum Frontend durchgereicht. Andererseits kommt es im Backend zu einem Overhead, da zu jedem Raumschild Node-RED alle 10 min (bzw. das im Backend eingestellte Intervall, vgl. Kapitel *5.8 - Administrations-Dashboards*, Abschnitt *EPD Dashboard*) die aktuellen Belegungen abruft. Im vollständigen Prozess des Abrufens werden je Raumschild drei Requests an das LSF gestellt. Bei 335 verschiedenen Räumen, die aus dem LSF abrufbar sind, entspricht das ca. 100 Requests/min, die Node-RED an das LSF übermittelt.

Zur Verbesserung könnte zum einen die Web-Sitzung von Raum zu Raum, aber auch von Intervall zu Intervall wiederverwendet werden. Somit würde die Anzahl der Requests auf 33/min verringert werden. Zum anderen könnte hier auf das Interrupt-Modell umgestellt werden. Dazu muss Node-RED im Fall einer Änderung einer Raumbelegung benachrichtigt werden. So wäre es möglich, regelmäßig innerhalb weniger Sekunden auf Änderungen von Raumbelegungen zu reagieren.

Konfiguration einzelner Frontends

Die Frontends lassen sich prinzipiell durch Konfigurationsdateien auf den MCUs anpassen. Dies erfordert jedoch, die jeweiligen Frontends in den Firmware-Update-Modus zu versetzen, um anschließend im Webbrowser die jeweiligen Konfigurations-Dateien zu bearbeiten. Durch Firmware-Rollouts werden diese Konfigurations-Dateien jedoch wieder überschrieben.

Es wäre sinnvoll, wichtige Konfigurationsoptionen (wie das Umschalten auf den Polling-Betrieb oder die Aufwachhäufigkeit der MCU) auch vom Backend mit einer eigenen Schnittstelle vornehmen zu können, ohne dass diese Konfigurationen durch ein Firmware-Update zwangsläufig überschrieben würden.

Verringerung des Stromverbrauchs

Es wurde eine größere Abweichung zwischen der gemessenen und der erwarteten Stromaufnahme bei dem Frontend festgestellt. Im Deepsleep-Modus der MCU ESP32 sollte der Chip selbst 10 µA Strom aufnehmen. In Kombination mit dem Entwicklungsboard sollte der Stromverbrauch bei etwa 65 µA liegen. Das LoRa-Modul sollte zur Ruhe-Phase 5 µA Strom benötigen. Die Kriechströme des EPD-Treibers sollten im ausgeschalteten Zustand vernachlässigbar sein.

Somit sollte der aufgenommene Strom während der Ruhephase der WOR-Periode des LoRa-Moduls insgesamt bei ca. 70 µA liegen, er lag jedoch bei 150 µA (vgl. Kapitel *6.1 - Stromaufnahme*, Abschnitt *Gemessene Stromaufnahme – Schlafphase*). Bei einer Messungenauigkeit von 5 % liegt die Differenz bei 64 %. Ein Ansatz, diesen Umstand genauer zu untersuchen, wäre, verschiedene Komponenten des Frontends zu ersetzen und erneut Strommessungen durchzuführen.

Mit einer alternativen WOR-Methode, wie dem Ultra Low Power Wake-Up Receiver »FH101RF« des Fraunhofer-Instituts für integrierte Schaltungen (Fraunhofer IIS), könnte der Energiebedarf zum Empfangen von WOR-Signalen deutlich reduziert werden. [17]

Der WOR-Empfänger kann mit einigen Empfangs-Parametern weiter konfiguriert werden, wie der Air-Data-Rate oder der Sub-Packet-Size. Möglicherweise können diese Parameter den Stromverbrauch verringern, jedoch kam es hier zu Empfangsproblemen.

Durch das Ausschalten des WOR-Empfängers zu vorlesungsfreien Zeiten, nachts und zum Wochenende kann weiterhin Energie eingespart werden. Jedoch muss hier die Energie mit berücksichtigt werden, die die MCU zum Aufwachen und Deaktivieren des WOR-Empfängers benötigt.

Wird das Frontend im Polling-Modus (anstatt eines externen Ereignisses, wie einem Wakeup-Empfänger, wird ein internes Ereignis, wie ein Timer-Interrupt, verwendet) betrieben, können die Uhrzeiten zur jeweils nächsten Wachphase angepasst werden. Jedoch beläuft sich die Angabe auf die Uhrzeiten eines Tages und die Tage einer Woche. Diese Konfigurationsmöglichkeit könnte z. B. mit der Angabe der zulässigen Wochennummern ergänzt werden, um das Frontend zu vorlesungsfreien Zeiten schlafen zu lassen.

Erweiterte Fehlerbehandlung

Für viele Fehler (z. B.: Das Frontend kann sich nicht mit dem WLAN verbinden, das LSF ist aus dem Backend heraus nicht erreichbar oder das EPD reagiert nicht auf Befehle) gibt es bisher noch keine Verfahrensweise zu Behebung und Benachrichtigung.

Kann z. B. die MCU sich nicht mit einem WLAN-AP oder im Anschluss mit dem konfigurierten MQTT-Broker verbinden, könnte die MCU über das LoRa-Modul versuchen, eine Fehlermeldung abzusetzen. Dazu würde ein weiteres LoRa-Modul im Backend benötigt werden, welches speziell auf Fehlermeldungen lauscht, um diese anschließend in der Administrationsoberfläche anzuzeigen oder Push-Benachrichtigungen herauszusenden.

Geheime Informationen sind auslesbar

Aus dem Speicher einer MCU ist über die USB-Schnittstelle gewöhnlich jederzeit die komplette Konfiguration auslesbar. Somit sind auch aus der Konfigurationsdatei die geheimen Informationen, wie die Zugangsdaten zu dem WLAN-AP oder zu dem MQTT-Broker, einsehbar.

Das konzeptuelle Problem ist, dass ein zur Authentisierung benötigtes Geheimnis nicht ohne weiteres auf dem Speicher eines öffentlich zugänglichen Chips hinterlegt werden kann, ohne dass dieser durch unbefugte Personen ausgelesen werden kann. Im Fall des ESP32 besteht die Möglichkeit, mit den Features secure-Boot und electronic Fuses (eFuse) Informationen auf dem Speicher zu schützen, jedoch bietet dies keine vollständige Sicherheit. Zudem ist dieses Feature mit Micropython noch nicht nutzbar.

Alternativ könnte der WLAN-AP ein abgeschottetes Virtual LAN (VLAN) ausstrahlen, über welches ausschließlich der MQTT-Broker erreichbar ist, mit dem sich die MCU verbindet. So würde die Möglichkeit des Missbrauchs auf ein Minimum reduziert werden.

Möglich wäre auch eine Lösung, die vollständig ohne WLAN und damit auch ohne MQTT-Broker auskommt. Die Raumbelegungen könnten stattdessen über das bereits bestehende LoRa-Modul übertragen werden.

Aktualisierung der Firmware der MCU

Aufgrund des Heap-Managements von Micropython können einmal allozierte Speicherbereiche nicht ohne Weiteres für alle Funktionen wiederverwendet werden. So werden für bestimmte Operationen auch in Schleifen immer wieder neu Speicherbereiche alloziert. Das führte dazu, dass zur Firmware-Aktualisierung der MCU der Heap-Speicher stark fragmentierte und nach einigen Iterationen kein ausreichend groß zusammenhängender Heap-Speicher mehr zur Verfügung stand. Das Problem konnte gelöst werden, indem öfter und dafür kleinere Speicher-Bereiche alloziert wurden. Jedoch fragmentiert auch bei kleineren Datenmengen der Heap-Speicher mit der Anzahl an Allozierungen. So wurde das Problem zumindest soweit abgeschwächt, dass es unter den aktuellen Bedingungen nicht mehr auftritt.

Um das Problem nachhaltig zu lösen, sollten einmal allozierte Speicherbereiche wiederverwendet werden. Das könnte durch die Erweiterung der Micropython-Umgebung durch kompilierte C-Bibliotheken erreicht werden, die selbstständig Speicherbereiche reservieren und diese wiederverwenden.

8 Quellen

[1] Espressif Systems, „ESP32 Series Datasheet“, 2024, Zugegriffen: 23. Juli 2024. [Online]. Verfügbar unter: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[2] I. Plauska, A. Liutkevičius, und A. Janavičiūtė, „Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller“, *Electronics*, Bd. 12, Nr. 1, Art. Nr. 1, Jan. 2023, doi: 10.3390/electronics12010143.

[3] S. Kholghi Eshkalak, M. Khatibzadeh, E. Kowsari, A. Chinnappan, W. A. D. M. Jayathilaka, und S. Ramakrishna, „Overview of electronic ink and methods of production for use in electronic displays“, *Opt. Laser Technol.*, Bd. 117, S. 38–51, Sep. 2019, doi: 10.1016/j.optlastec.2019.04.003.

[4] R. C. Liang, J. Hou, J. Chung, X. Wang, C. Pereira, und Y. Chen, „20.1: Microcup® Active and Passive Matrix Electrophoretic Displays by Roll-to-Roll Manufacturing Processes“, *SID Symp. Dig. Tech. Pap.*, Bd. 34, Nr. 1, S. 838–841, Mai 2003, doi: 10.1889/1.1832401.

[5] W.-C. Kao, H.-Y. Chen, Y.-H. Liu, und S.-C. Liou, „Hardware Engine for Supporting Gray-Tone Paintbrush Function on Electrophoretic Papers“, *J. Disp. Technol.*, Bd. 10, Nr. 2, S. 138–145, Feb. 2014, doi: 10.1109/JDT.2013.2289364.

[6] Silicon Laboratories Inc., „AN0063: Driving Electronic Paper Displays (E-Paper) - r1.03“, Zugegriffen: 14. Dezember 2024. [Online]. Verfügbar unter: https://www.silabs.com/documents/public/application-notes/an0063-efr32-EPD.pdf

[7] Waveshare International Ltd., „7.5inch e-Paper HAT (B) Manual“, *Waveshare Wiki*. Zugegriffen: 21. Januar 2025. [Online]. Verfügbar unter: https://www.waveshare.com/wiki/7.5inch_e-Paper_HAT_(B)_Manual#Resources

[8] W. He *u. a.*, „Driving Waveform Design of Electrophoretic Display Based on Optimized Particle Activation for a Rapid Response Speed“, *Micromachines*, Bd. 11, S. 498, Mai 2020, doi: 10.3390/mi11050498.

[9] Organization for the Advancement of Structured Information Standards, „OASIS Message Queuing Telemetry Transport (MQTT) TC“. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: https://groups.oasis-open.org/communities/tc-community-home2?CommunityKey=99c86e3a-593c-4448-b7c5-018dc7d3f2f6

[10] W. Eddy, „RFC 9293: Transmission Control Protocol (TCP)“. Internet Engineering Task Force (IETF), August 2022. Zugegriffen: 20. Januar 2025. [Online]. Verfügbar unter: https://datatracker.ietf.org/doc/html/rfc9293

[11] Waveshare International Ltd., „E-Paper ESP32 Driver Board“, *Waveshare Wiki*. Zugegriffen: 11. Januar 2025. [Online]. Verfügbar unter: https://www.waveshare.com/wiki/E-Paper_ESP32_Driver_Board

[12] Richtek Technology Corporation, „RT9193 - 300mA, Ultra-Low Noise, Ultra-Fast CMOS LDO Regulator“. 2022. [Online]. Verfügbar unter: https://www.richtek.com/assets/product_file/RT9193/DS9193-18.pdf

[13] Olimex Ltd., „ESP32-DevKit-LiPo - Open Source Hardware Board“. Zugegriffen: 19. Januar 2025. [Online]. Verfügbar unter: https://www.olimex.com/Products/IoT/ESP32/ESP32-DevKit-LiPo/open-source-hardware

[14] Waveshare International Ltd., „E-Paper Driver HAT“, *Waveshare Wiki*. Zugegriffen: 19. Januar 2025. [Online]. Verfügbar unter: https://www.waveshare.com/wiki/E-Paper_Driver_HAT

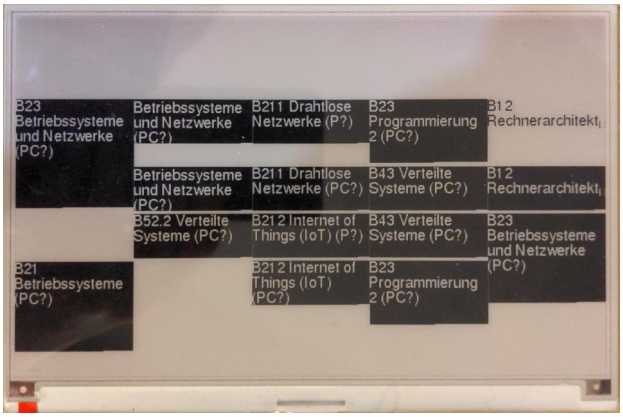
[15] Chengdu Ebyte Electronic Technology Co.,Ltd., „E220-400T30D LLCC68 LoRa Module“. Zugegriffen: 19. Januar 2025. [Online]. Verfügbar unter: https://www.cdebyte.com/products/E220-400T30D

[16] Olimex Ltd., „SY80X9AAAC - High efficient 1.5MHz, 1.5A, SOT5-23 synchronous step down regulator“. Zugegriffen: 27. Januar 2025. [Online]. Verfügbar unter: https://www.olimex.com/Products/Components/IC/SY8009A/

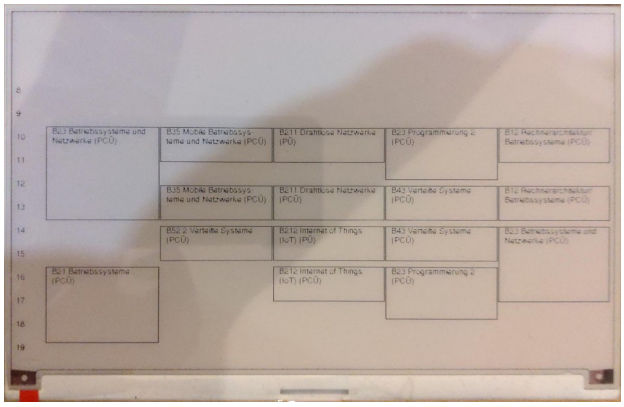
[17] Dr. Frank Oehler und Dr. Heinrich Milosiu, „RFicient®ULTRA-LOW POWER RECEIVER“. Fraunhofer-Institut für Integrierte Schaltungen, 2019. Zugegriffen: 23. Juli 2024. [Online]. Verfügbar unter: https://www.iis.fraunhofer.de/content/dam/iis/en/doc/il/ics/ic-design/Datenblaetter/WhitePaper_RFicient_2019.pdf

A Evolution der Anzeige des Raumplans auf dem EPD

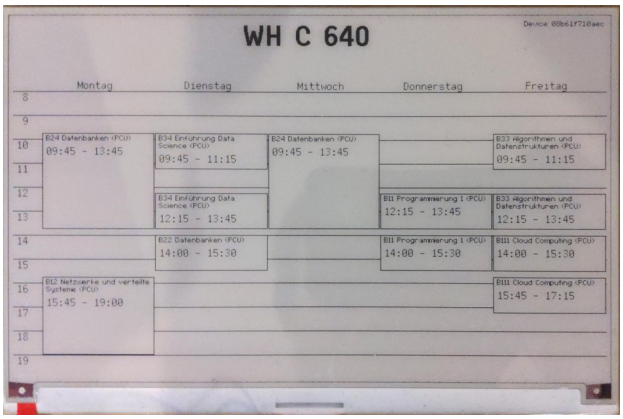
Die folgende Bilderstrecke zeigt den Fortschritt des Entwicklungsstands der Darstellung des Raumplans auf dem EPD. Das Datum unter der Abbildung gibt an, wann das jeweilige Bild entstanden ist.



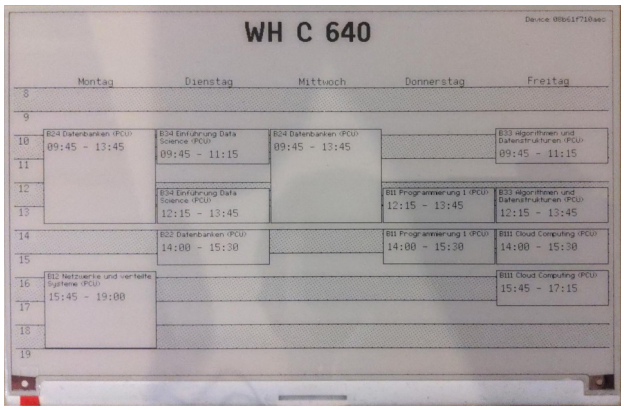
1 – 15.12.2024



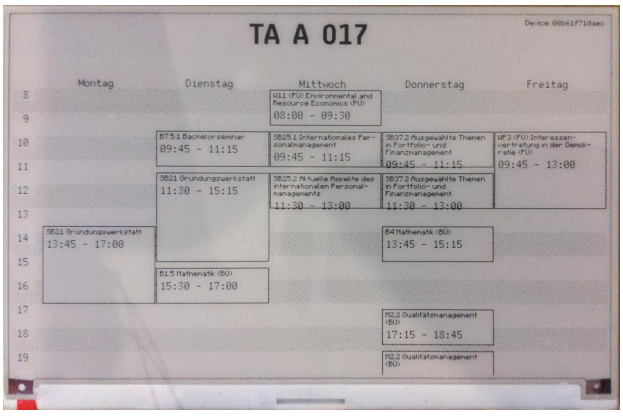
2 – 16.12.2024



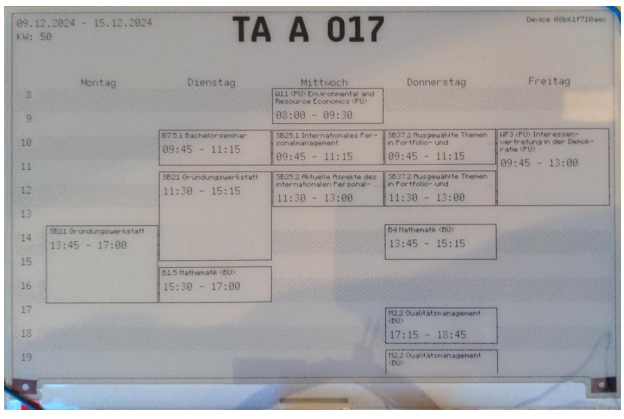
3 – 23.12.2024



4 – 23.12.2024



5 – 29.12.2024



6 – 25.01.2025

Eigenständigkeitserklärung

Ich, Jonas Klein, versichere hiermit an Eides statt durch meine Unterschrift, dass ich die vorliegende Arbeit eigenständig angefertigt, die vorliegende Arbeit nicht anderweitig für Prüfungszwecke vorgelegt, alle verwendeten Quellen angegeben und alle fremdverfassten Inhalte als solche gekennzeichnet habe.

Donnerstag, 23. Januar 2025 

Datum, Unterschrift