Faculty of Electrical Engineering
Department of Measurement

Smart e-ink information badge

Master's thesis

Petr Procházka

Open Informatics
Computer Engineering

Supervisor: Ing. Vladimír Janíček, Ph.D.
March 2025

Abstract

In recent years, electronic paper, as a display technology, has become a compelling choice for low-power devices. This thesis explores its use for independent, wirelessly rewritable conference screens. The introductory part introduces e-paper technology. Subsequently, an analysis of the smart e-paper screen market is performed. In the next part, a suitable hardware platform and components for the control gateway and the e-paper screen itself are selected. Subsequently, a software architecture and a suitable method of control and data transfer are designed. This is followed by a practical part of the work, where the prototype design process, including the 3D printed housing, is described in detail. The firmware development for both the control gateway and the e-paper screen is also described. The final part describes the prototype testing and comparison of the achieved results with commercial products.

Keywords: ESP32, Electronic paper, Wireless Communication, ESP-NOW, Web Server

Abstrakt

V posledních letech se elektronický papír, jakožto zobrazovací technologie, stal přesvědčivou volbou pro zařízení s nízkou spotřebou energie. Tato diplomová práce zkoumá jeho použití pro nezávislé, bezdrátově přepisovatelné konferenční obrazovky. V úvodní části je představena e-paper technologie. Následně je provedena analýza trhu chytrých e-paper obrazovek. V další části je vybrána vhodná hardwarová platforma a komponenty pro ovládací bránu i samotnou e-paper obrazovku. Následně je navržena softwarová architektura a vhodný způsob ovládání a přenosu dat. Poté následuje praktická část práce, kde je podrobně popsán proces návrhu prototypu včetně pouzdra vytištěného na 3D tiskárně. Rovněž je popsán vývoj firmwaru jak pro ovládací bránu, tak pro e-paper obrazovku. V poslední části je popsáno testování prototypu a porovnání dosažených výsledků s komerčními zařízeními.

Klíčová slova: ESP32, Elektronický papír, Bezdrátová komunikace, ESP-NOW, Webový server

Acknowledgements: I would like to thank my supervisor Ing. Vladimír Janíček, Ph.D. for his professional guidance, help and advice in the preparation of this thesis. I would also like to thank my family for their support not only during the writing of this thesis, but especially throughout my studies. Finally, I would like to thank my girlfriend Natálie for her unwavering support and trust even in difficult times.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The demand for flexible, energy-efficient, and wirelessly updatable display systems has increased significantly in recent years, particularly in the context of conferences, offices, and public spaces. Traditional paper-based or LCD information badges suffer from limitations such as poor visibility in direct sunlight and significant energy consumption. As a response to these limitations, electronic paper technology presents an attractive alternative due to its low power consumption, high readability, and capability to retain displayed content without continuous power supply.

The motivation for this work emerged from practical challenges observed during a medical conference, where printed name badges for panelists had to be repeatedly reprinted throughout the day due to changes in the lineup. A similar need for dynamic, real-time updates also arises in academic and business environments, such as during meetings, where participants and agenda items may change frequently. These cases illustrate a broader need for adaptable, user friendly identification and signage solutions that can respond quickly to changing circumstances without the need to manually reprint.

This thesis focuses on the design and implementation of a wirelessly updatable electronic information badge based on e-ink display technology. The badge is intended to serve as a low-power, user friendly solution for information presentation in dynamic environments like conferences or exhibitions. The main objective of this work is to develop a functional prototype of the badge, including both the embedded firmware and server side logic, and to evaluate its performance and usability. The practical implementation is complemented by an analysis of the proposed solution and selected commercial alternatives.

Chapter 2

Electronic paper technology

Electronic paper (also known as e-paper or electronic ink) is a display technology that mimics the appearance of ink on classical paper. Unlike traditional displays such as LCD or OLED, e-paper reflects ambient light instead of emitting its own, making its readability perfect in direct sunlight and comfortable for prolonged viewing when used as e-book reader. The display consumes power only when redrawing, making it suitable for low-power applications. The display retains an image without continuous power which makes it particularly well suited for battery-powered embedded devices. This chapter provides an overview of the principles, characteristics, and applications of e-paper technology.

2.1 Principle of operation

The most widely used form of electronic paper is the electrophoretic display (EPD) technology, commercialized primarily by the company E Ink, which was co-founded in 1997 by MIT undergraduates Mr. Comiskey and Mr. Albert.[1] The core mechanism involves microcapsules or microcups filled with positively charged white particles and negatively charged black particles suspended in a clear fluid. When a voltage is applied to electrodes above or below the capsule, the particles move toward the surface or away from it, making the pixel appear either white or black depending on the polarity.

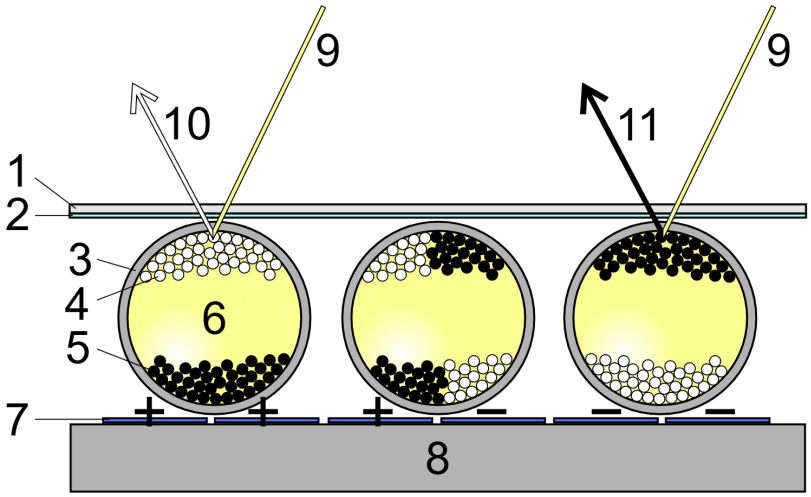

Figure 2.1: Schematic side view of an electrophoretic e-paper display.[2]

The structure of an electrophoretic e-paper display is illustrated in Figure 2.1. The display consists of multiple layers: the upper layer (1), transparent electrode layer (2), and microcapsules (3) filled with positively charged white pigments (4) and negatively charged black pigments (5) suspended in transparent oil (6). Below these are the electrode pixel layer (7) and the bottom supporting layer (8). When voltage is applied, the pigments move within the capsules, changing the visible color to white (10) or black (11) depending on the direction of light (9) and the electric field.

Some newer versions of e-paper can also display limited colors by incorporating colored particles or filters. Other technologies, such as electrochromic displays, cholesteric liquid crystals, or electrowetting, exist but are less common in commercial badge applications due to lower stability or higher power requirements.

2.2 Characteristics and advantages

In addition to low power consumption and high readability, electronic paper displays offer several other features that make them attractive for embedded systems. One notable advantage is their wide viewing angle, which results from the use of pigment-based imaging rather than emissive pixels, which ensures consistent visibility from various positions without contrast degradation. Another key feature is bistability, which allows the display to retain an image without power, further reducing energy usage during idle states. E-paper modules are also thin, lightweight, and available in flexible formats.

2.3 Limitations and challenges

Despite its numerous advantages, electronic paper technology also presents several limitations that constrain its broader application. One of the downsides is the slow refresh rate, which makes e-paper unsuitable for displaying dynamic content like video. Another challenge is ghosting, where remains of a previously displayed image may persist after an update. This is present especially in displays that rely primarily on partial refresh techniques, which redraw only specific part of the screen in faster time. Ghosting is resolved by full refresh, which is considerably longer in contrast with partial refresh.

Although color variants of e-paper exist, they tend to suffer from low saturation, narrow color range, and even slower update speeds compared to monochrome versions. Additionally, while some flexible models are available, most commercially used e-paper modules are relatively fragile and require careful handling to prevent mechanical damage.

2.4 Manufacturers and common modules

The dominant supplier of electrophoretic displays is E Ink Holdings, which licenses its technology to various manufacturers such as Waveshare, Pervasive Displays, and Visionect. These companies produce modules of various sizes, typically ranging from 1.54" to 7.5", with communication interfaces such as SPI or I^2C. Controllers such as the SSD1675, UC8151, or IL3897 are commonly used to manage the waveform driving needed for particle movement.

2.5 Driving e-paper displays

E-paper displays are driven by specialized display controller ICs that interpret data and generate waveforms to update the image. The microcontroller communicates with the display using SPI or another serial interface to send pixel data and update commands.

Each update requires a specific waveform, defined in a look-up table (LUT), which determines how voltages are applied to move the particles. Full refresh cycles are slower but remove ghosting, while partial refreshes are faster and more power efficient but can lead to visual artifacts.

2.6 Applications of e-paper technology

E-paper displays are used across a wide range of applications where low power consumption, high readability, and image stability are essential. In consumer electronics, they are best known from e-book readers like the Kindle[3] and digital writing tablets such as reMarkable[4]. In retail, electronic shelf labels enable dynamic pricing with minimal energy use. E-paper also supports education by providing eye-friendly displays for digital learning, and in offices, it powers meeting room signs showing real-time schedules. In healthcare, they are used for patient room and information signs. Public transport systems employ e-paper for bus stop and station signage. In innovative applications, BMW's iX Flow concept car uses E Ink for a color changing vehicle surface[5], while Reviver's digital license plates on figure 2.2 demonstrate its use in connected automotive infrastructure.

Figure 2.2: Reviver's digital license plate.[6]

2.7 Summary

Electronic paper consumes power only while refreshing and remains readable in direct sunlight. It is a technology that excels in applications requiring low power consumption and persistent image retention. Its reflective, non emissive behaviour makes it particularly easy on the eyes, supporting comfortable longterm viewing without the strain associated with backlit displays. Real world applications span from e-book readers and digital writing tablets, to retail shelf labels, meeting room signs, logistics tags,

healthcare signage and public transport displays. Although it suffers from limitations in refresh speed and color rendering, e-paper remains a appealing alternative to traditional displays in scenarios with low interactivity and the need for long battery life.

Chapter 3

Market analysis

This chapter provides an overview of the current market landscape for smart e-paper badges, with a focus on identifying key application areas, customer segments, and existing products. The analysis aims to identify both the strengths and weaknesses of current solutions in order to highlight opportunities for improvement.

3.1 Target market

The primary target market for smart E-Ink badges consists of events and conferences where the displayed information may change frequently. In such scenarios, repeatedly printing new paper badges or signs becomes inefficient and complicated. A reusable badge system with remotely updatable content provides a practical and flexible solution.

Beyond individual badges, the same technology can be applied in the form of wall mounted information signs used in meeting rooms, offices, or public spaces. These can dynamically display schedules, room availability, directional signs, or other context dependent information.

A key design goal of the system is full autonomy and independence from third party services. The device is intended to operate without the need for continuous internet connectivity or reliance on external platforms. It should be self powered and capable of receiving updates via a local wireless interface like Bluetooth or Wi-Fi, through a custom web based management portal. This approach minimizes operational dependencies and simplifies deployment and usage on the event.

3.2 Competitive landscape

The market for E-Ink display solutions encompasses a variety of products tailored for dynamic signage and nameplate applications. Below is an analysis of selected products relevant to smart badge and signage solutions.

3.2.1 Infsoft E-Ink display beacons

Infsoft offers a series of E-Ink display beacons primarily used for conference room scheduling and digital signage in smart building environments.[1] The 7.5 inch version of the display has resolution of 800 × 480 pixels. The display supports tri-color modes and includes an RGB LED indicator for additional visual signaling.

For power, it relies on six CR2477 batteries, offering a long lifespan under low frequency update conditions. The unit is priced at €75, with the mounting kit available for an additional €25.

Importantly, the system depends on proprietary Infsoft infrastructure. To enable beacon communication, the user must purchase an Infsoft Locator Dongle, priced at approximately €90. Furthermore, all device management and content updates are handled via the Infsoft LocAware platform, which is a cloud based solution. While LocAware offers a centralized interface for large scale deployment and monitoring, this setup introduces dependency on external infrastructure and continuous internet connectivity, making it less suitable for standalone or privacy sensitive deployments.

Although the hardware looks promising, its integration model is designed primarily for use cases such as conference room scheduling and smart office installations, rather than flexible, self hosted environments.

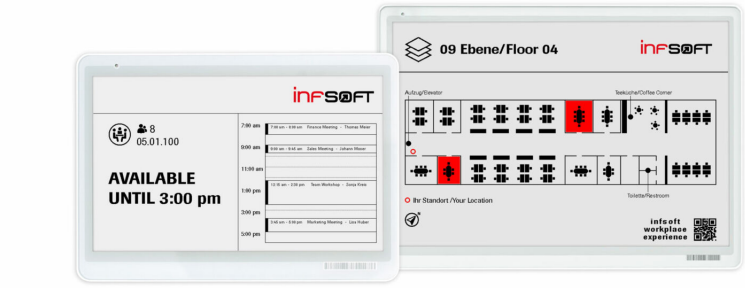

Figure 3.1: Infsoft E-Ink Display Beacon[7]

3.2.2 Good Display ET0750

Good Display offers the ET0750 series, a 7.5 inch E-Ink display designed for applications such as electronic shelf labels, signage, and logistics.[2] The screen features a resolution of 800 × 480 pixels and supports both three and four color modes (black, white, red, and optionally yellow).

The device is powered by three CR2450 batteries, offering a potential battery life of up to seven years. The unit is priced at $50. It supports BLE 5.0 communication using a private protocol, and can be updated via a mobile app, supported only on Android, or a dedicated gateway.

The system is intended to operate within Good Display's infrastructure. Device management and updates require either a smartphone app or integration through a proprietary gateway. The manufacturer also offers options such as NFC, local cloud

[1] https://www.infsoft.com/hardware/infrastructure-hardware/infsoft-e-ink-display-beacons

[2] https://www.good-display.com/product/558.html

Figure 3.2: GoodDisplay ET0750-44B[8]

servers, and API based integration for enterprise use. Although the solution is flexible and capable of high volume updates, its reliance on specific backend systems and hardware makes it more suitable for centralized, managed deployments rather than standalone or fully offline scenarios.

3.2.3 TapirX 7.5

TapirX provides a 7.5 inch E-Ink display module with a resolution of 800×480 pixels.[3] Its main purpose is to work as a meeting room scheduling device. The device is equipped with a 6500 mAh Li-Po battery, claiming a battery life of over 3 years on a single charge. It utilizes Wi-Fi for periodic wireless updates and features USB-C port for charging. The unit price is €313 including VAT.

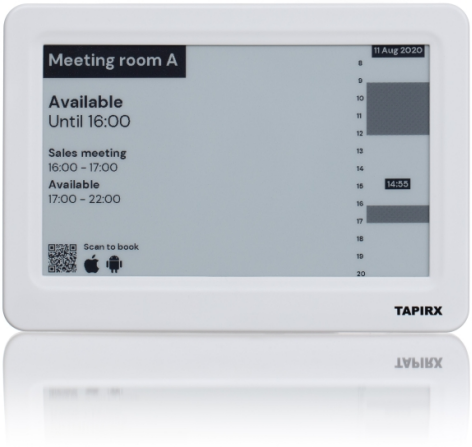

Figure 3.3: TapirX 7.5 scheduling device.[9]

Device management and updates require smartphone app, which is available both on iOS and Android.

Similarly to Infsoft E-Ink beacon, its integration model is designed primarily for use cases such as conference room scheduling and smart office installations.

[3] `https://www.tapirx.com/product/tapirx-75-1`

3.2.4 Taiden HCS E-Ink nameplate

Taiden offers a 7.5 inch dual screen E-Ink nameplate designed for conference environments.[4] It features a resolution of 800 × 480 pixels and a tri color display. The device is powered by a 2300 mAh battery, lasting up to two years with daily updates, and includes BLE 4.2 and LED indicators.

Figure 3.4: Taiden HCS-1085 conference nameplate.[10]

Integration is limited to the proprietary Taiden system, with full functionality requiring the HCS-1085T adapter for central control. However, it is a closed system not suited for standalone or customizable use. Furthermore, pricing is not publicly listed on the manufacturer's website.

3.2.5 DSPPA D7642

The DSPPA D7642 is a dual sided 7.5 inch E-Ink nameplate created for use in conference and meeting environments.[5] Each side offers an 800 × 480 pixel tri color display (black, white, red). The device communicates via ZigBee 2.4G and Bluetooth v4.2, with a range of up to 100 meters in open areas and a battery life of 3–5 years.

The system requires a Bluetooth gateway for operation and is managed through a proprietary mobile application, available on both iOS and Android platforms. The unit is priced at $223 on Alibaba. In the case of gateway, the price is not listed.

3.3 Summary of analysed products

The compared e-paper display solutions each serve specific use cases, ranging from meeting room information signage to conference nameplates. Infsoft and TapirX target room scheduling use cases with polished hardware, though both rely on cloud infrastructure, Infsoft through the LocAware platform and TapirX via its mobile apps. Good Display's ET0750 offers the most flexible hardware at the lowest cost, but still depends on a proprietary gateway or Android app, limiting standalone use. Taiden's HCS nameplate has a dual screen format and smooth appearance, but is fully tied

[4] https://www.taiden.com/product_list/15.html
[5] https://www.dsppacs.com/products/e-ink-screen-meeting-nameplate/

Figure 3.5: DSPPA D7642 conference nameplate.[11]

to its proprietary ecosystem and central control adapter. DSPPA's D7642 balances display quality and dual screen functionality with broader app support, but it too requires a Bluetooth gateway and lacks transparency in infrastructure costs. Overall, while all options offer display capabilities, most are not suitable for fully independent or offline deployments. It is also worth mentioning that some products cannot be easily purchased, but only an inquiry form is available.

Product	**Protocol**	**Control Method**	**Primary Use Case**	**Price**
Infsoft Beacon	BLE	Web portal (LocAware platform)	Room scheduling, digital signage	€165 (including gateway)
Good Display ET0750	BLE	Android app or proprietary gateway	Shelf labels, signage	$50
TapirX 7.5	Wi-Fi	Mobile app (iOS & Android)	Room scheduling	€313
Taiden HCS	BLE	Central control via HCS-1085T adapter	Conference nameplate	Not listed
DSPPA D7642	BLE, ZigBee	Mobile app (iOS & Android)	Conference nameplate	$223 (gateway not listed)

Table 3.1: Comparison of 7.5" E-Ink Displays

Chapter 4

Hardware platform

This chapter presents the hardware platform and necessary peripherals for the e-paper badge system. The system requires a microcontroller (MCU) for each client device to manage the e-paper display and local functionalities, as well as a dedicated unit for the gateway, which hosts the web server used to manage and update badge content. The hardware must support Wi-Fi connectivity for the web interface and a lightweight, energy-efficient communication protocol such as Bluetooth Low Energy (BLE) for data exchange between devices. A suitable e-paper display panel of optimal size and a battery are also essential components of the system. The battery ensures flexible and independent operation without the need for a constant power supply during events and conferences.

4.1 Suitable MCU

For this project, an ideal MCU would support the following communication protocols:

- **SPI**: Required to interface with the e-paper display. SPI is commonly supported by most low-power microcontrollers, making it a non-restrictive requirement.
- **Wi-Fi**: Microcontroller can act as a standalone access point and host a web server or simple captive portal, providing app-free interaction with any smartphone through a standard web interface.
- **BLE**: Enables efficient, low-energy local communication between badges.

The key parameters for a suitable MCU for a prototype of the smart e-paper badge and the gateway are: wireless connectivity, easy programmability, affordability and community support. There are a number of manufacturers on the Wi-Fi microcontroller market. The STM32Wx series from STMicroelectronics[12] is worth mentioning, which primarily focuses on short-range modules with support for BLE (Bluetooth Low Energy), Thread, ZigBee, or longer-range modules with support for LoRaWAN and Sigfox protocols. Another manufacturer is NXP Semiconductors with a similar focus to STMicroelectronics. Microchip Technology offers the PIC32MZ-W1 series of microcontrollers[13]. Nordic nRF52[14] series and Texas Instruments low power 2.4 GHz MCUs[15] are also possible. Last but not least, Espressif Systems has the ESP32 product on the market. These are just a few examples of Wi-Fi microcontrollers that are commonly used in various applications. There are many other MCUs with similar functions on the market today.

After evaluating several microcontrollers, the ESP32 was chosen as the platform of choice for both e-paper client and the gateway. The primary reason for choosing the ESP32 was the wide availability of affordable development kits, which made it an ideal choice for fast prototyping. It supports all necessary protocols mentioned before and offers strong community support, extensive development resources, and a good balance between performance and energy efficiency.

4.1.1 ESP32

ESP32 is a microcontroller manufactured by Espressif Systems since 2016.[1] It is a low-cost system on low power, high price and high performance. The chip is manufactured by the Taiwanese company TSMC.

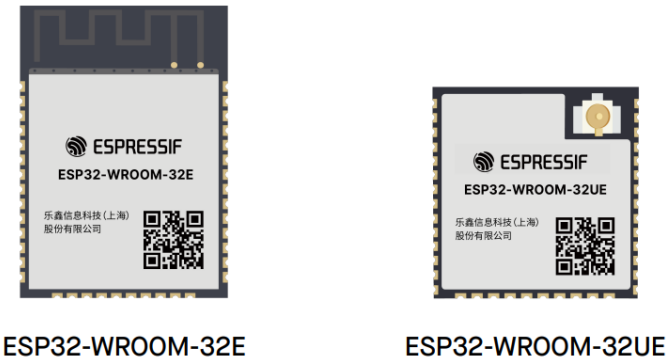

Figure 4.1: ESP32 SoC with PCB antenna on the left and with external antenna connector on the right.[16]

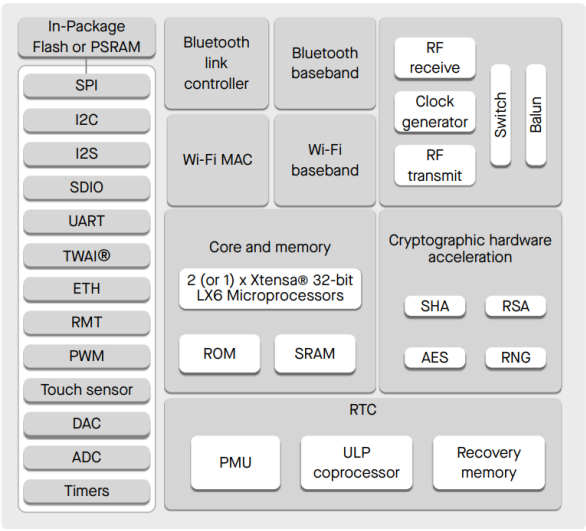

Figure 4.2: ESP32 Functional Block Diagram[17]

The ESP32 is the successor to the ESP8266 microcontroller. One of the key features of the ESP32 SoC, in most of its series, is a dual-core processor that enables efficient multitasking and high performance. The processor is based on the 32-bit Xtensa LX6

[1] https://www.st.com/en/microcontrollers-microprocessors/stm32-wireless-mcus.html

architecture and can operate at a clock speed of up to 240 MHz. This microcontroller is available in many variants with slightly different parameters, such as the size of the flash memory. There are even series of ESP32 with RISC-V architecture.

The ESP32 also has built-in Wi-Fi and Bluetooth, which makes it easy to connect to wireless networks and other devices. The Wi-Fi module supports the 2.4 GHz bands and is compatible with the 802.11 b/g/n standards. The Bluetooth module supports classic Bluetooth in 4.2 version and Bluetooth Low Energy (BLE), enabling a wide range of applications. The ESP32 functional block diagram is shown on image 4.2.

The ESP32 SoC module is full of on-chip peripherals. It has 34 programmable GPIOs. For serial communications it provides three UART controllers, two I^2C masters/slaves and four SPI interfaces, plus two I^2S interfaces for audio. Analog I/O includes two 8-bit DAC and up to 18 channels of 12-bit SAR ADC. Last but not least, the SoC has Ethernet MAC interface with dedicated DMA, motor PWM and LED PWM up to 16 channels.[17]

4.1.2 ESPink board

ESPink[2] board was chosen as a suitable ESP32 board to create a prototype of the smart e-paper badge. ESPink provides a dedicated ePaper connector whose power supply rail is gated by a transistor on GPIO2, eliminating any sleep-mode current draw. A built-in, low-power programmer simplifies development. Board has JST-PH connector for LiPo battery, which is chargeable over onboard USB-C. Battery voltage can be monitored through a divider on GPIO34's ADC input. Numerous unused ESP32 GPIOs are exposed for custom extensions.

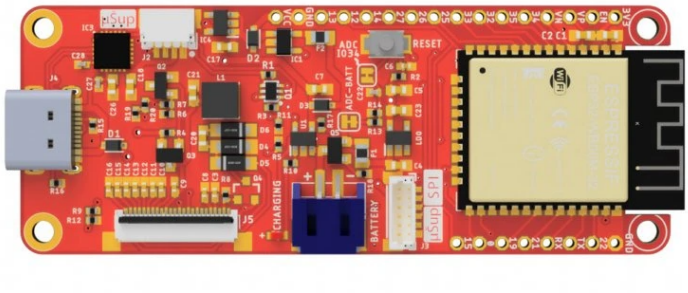

Figure 4.3: ESPink development board.[18]

4.2 E-Ink panel

Electrophoretic (EPD) technology was selected for its low power consumption, high contrast and persistent image retention when unpowered. To ensure good visibility from distance a larger panel size was targeted. After evaluating available modules, a 7.5-inch display was found to offer the optimal balance between viewing area, flexibility and price.

Good Display's 7.5-inch GDEY075T7 panel[3] was chosen to meet these requirements. This monochrome (black and white) EPD features an 800x480 pixel resolution, deliv-

[2] https://github.com/LaskaKit/ESPink

[3] https://www.good-display.com/product/396.html

Figure 4.4: Good Display GDEY075T7 7.5".[19]

ering sharp and readable text. The module is driven by the UC8179 IC and interfaces via an SPI bus routed through a flat flexible cable, providing a compact connection. Operating from a 3.3 V supply and rated for 0 °C to 50 °C ambient temperature. Full refresh of the display takes around 3 seconds, fast refresh 1.5 seconds and partial refresh 0.34 seconds. The refresh consumes 26.4 mW of energy. The GDEY075T7 combines competitive pricing and the proven quality of Dalian Good Display's product line.

Chapter 5

Software architecture

This chapter presents the high level software architecture that enables a self-contained gateway to discover, manage, and update battery powered e-paper badges. It begins by outlining the functional and physical requirements that shape the design and then justifies the selection of protocols and interfaces with respect to energy constraints, usability, and independence from external infrastructure. The remaining sections trace the flow of data from a browser based control page through the gateway firmware to the badge and highlight the responsibilities and interactions of each major component. Together these elements provide a foundation for the detailed implementation and prototype evaluation that follow in later chapters.

5.1 Protocol selection

The Smart E-Ink Badge project aims to provide event staff with a completely independent way to edit the text shown on e-paper badges. The architecture must therefore rely solely on chosen hardware that can run from a battery for at least 24 hours. Provide a control page through which organisers can rewrite any badge at run time, allow new badges to be registered on-site without reflashing firmware, and continue to operate even when external Internet connection, electricity or other local infrastructure is not available. Those constraints guided 3 key architectural choices.

5.1.1 Local AP with webserver

Running the gateway as its own access point lets any phone or laptop connect instantly. No on-site Wi-Fi, captive portal or custom mobile application is needed. Bluetooth low energy would reduce gateway power consumption, but demands a native app and is inconsistently supported. NFC and USB would force organisers to reach every badge physically, breaking the remote rewrite goal. Local AP therefore trades a little extra power consumption on the gateway, easily covered either by a larger battery or by connecting the gateway directly into laptop via USB.

5.1.2 REST over HTTP with JSON

Badge updates are triggered by people and therefore occur only occasionally. Most of the time the control channel is idle and wakes only when staff decide to change what appears on a badge. In this low duty cycle setting the connection overhead of ordinary

HTTP is negligible and the protocol's universality makes integration easy. Because the gateway already runs an HTTP listener to serve the HTML control page, adding a REST interface on the same socket reuses code, avoids extra background services, and matches the familiar request and response pattern of every web browser. HTTP is the standard for application communication on the Internet. Every operating system ships client libraries, diagnostic tools such as `curl` are available by default, and a wide range of guides, frameworks, and security middleware already exists. The protocol also scales smoothly.

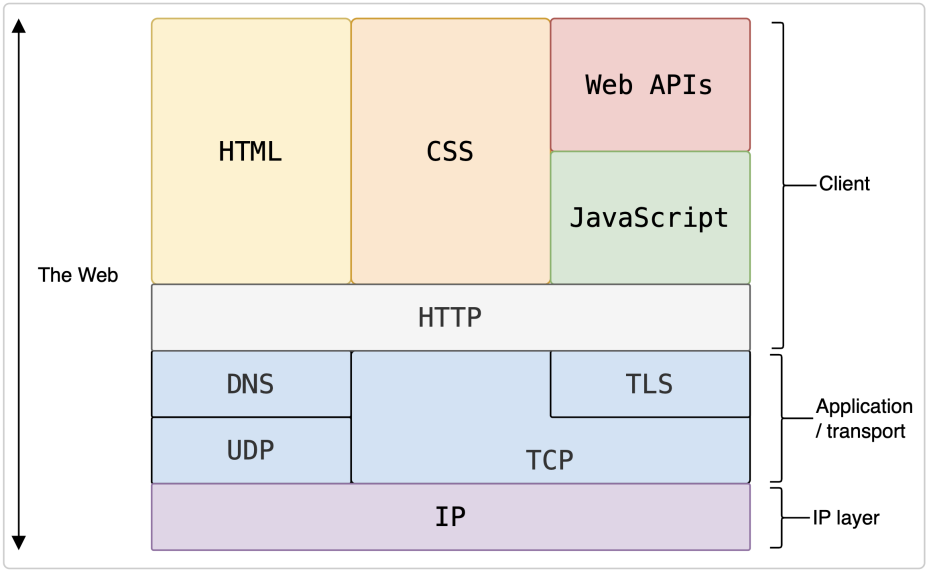

Figure 5.1: Simplified protocol stack of a modern web client.[20]

Figure 5.1 shows that in a browser based workflow the first protocol to leave the tab sandbox is always HTTP. HTTP rides on top of TCP or, in the case of HTTP 3, on QUIC over UDP[21], and both rest on IP. Security and naming layers such as TLS and DNS are handled by the operating system so the firmware does not need to manage them. Higher level browser features like HTML, CSS, JavaScript, and Web APIs are by design limited to sending or receiving HTTP resources. Other options such as WebSocket or MQTT carried in WebSocket still begin with an HTTP upgrade or tunnel. Using plain REST over HTTP therefore follows the browser's native path and keeps the gateway firmware simple because no extra broker or upgrade logic is required.

Binary formats like Protocol Buffers save only a few dozen bytes at our small payloads, so JSON remains the simplest and most transparent option.

5.1.3 Gateway-badge communication

Choosing the over the air link between the gateway and a badge required evaluating established short range protocols. Conventional options such as infrastructure mode Wi-Fi and Bluetooth Low Energy both impose a multi step connection handshake before a single user byte can be transferred. Wi-Fi must complete association, authentication, DHCP, and TCP setup which keeps the radio active for a considerable time and demands additional code for IP and TLS. BLE reduces the packet size but

still requires advertisement scanning, connection establishment, and GATT discovery which adds tens of milliseconds of signalling.

To overcome these limits the design adopts ESP-NOW, a vendor specific protocol built into the ESP32. ESP-NOW achieves a significantly lower overhead by operating directly at the data link layer, which reduces the five layers of the OSI model to only one. Application data are placed in a vendor specific action frame and broadcast or unicast to a peer without any prior connection. Each payload can carry up to 250 bytes, because the length field of the action frame is only 1 byte. A unicast frame with its acknowledge finishes in roughly five hundred microseconds at the default one megabit rate so the transceiver is enabled for a fraction of the time required by the alternatives.

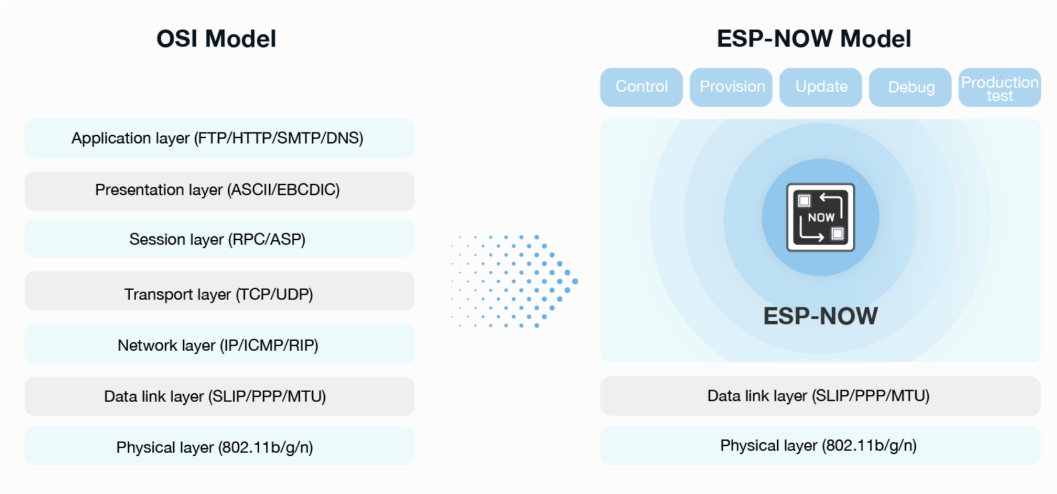

Figure 5.2: OSI model vs. ESP-NOW.[22]

The peer table embedded in the ESP32 silicon supports 20 in open mode or 6 when encrypted mode is used, which is sufficient for the expected conference deployment. Furthermore, using broadcast and storing the target MAC address into payload removes this limitation. With a transmit power of up to +21dBm, even devices that are 200 meters apart from each other in an open space can be controlled easily. The protocol therefore delivers the shortest airtime, the lowest energy cost, and a secure transmission path while reusing the existing 2.4 GHz radio on the ESP32, making it the most efficient choice for battery powered badges that require only occasional updates.[23]

These 3 choices minimise badge battery drain, maximise organiser convenience and eliminate dependencies on outside networks or power, fully aligning the implementation with the project's design objectives.

5.2 Dataflow pipeline

Figure 5.3 shows the end-to-end dataflow. A user connects a phone or laptop to the ESP32 gateway's local Wi-Fi access point and loads the control page served by the embedded web server. Through this page the user can register new badge addresses or submit updated content. Each interaction produces an HTTP POST request that reaches the web server which validates the JSON payload and stores or queues it as required. When an update is queued the gateway immediately encapsulates the payload

in an ESP-NOW vendor specific action frame and unicasts it to the target badge whose MAC address was supplied by the user. The badge receives frame and extracts the content. It then refreshes the e-paper display and returns a short acknowledge so the gateway can report success back to the browser. After the transfer the badge returns to sleep and the gateway resumes listening for further HTTP requests. The entire path therefore consists of three distinct hops: browser to gateway over Wi-Fi using standard HTTP, gateway to badge over ESP-NOW using a connectionless data-link frame, and badge back to gateway with a minimal acknowledge, all of which take place without relying on external infrastructure or keeping any radio active longer than necessary.

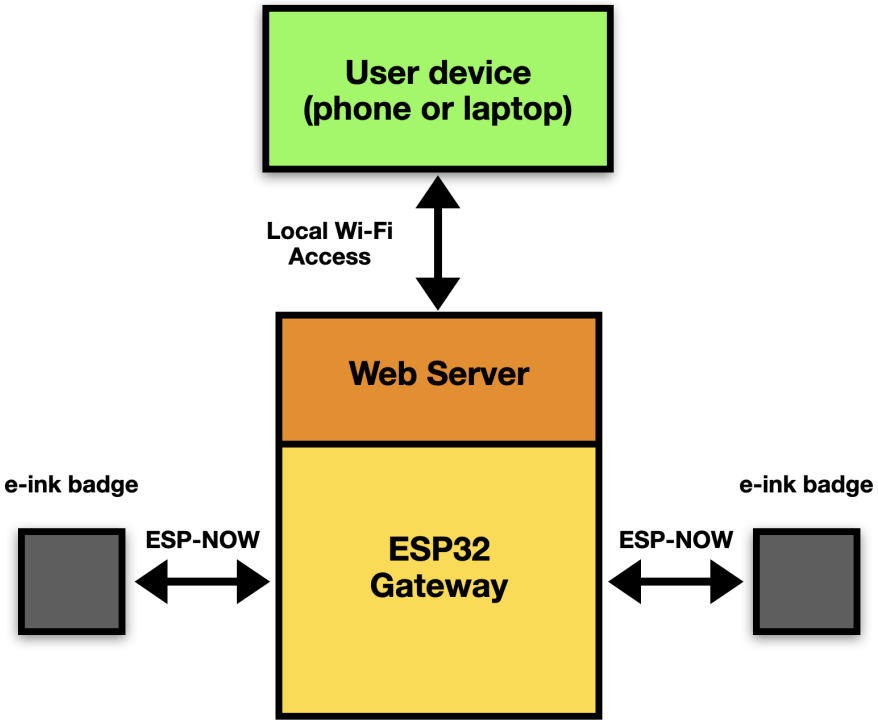

Figure 5.3: Simplified SW schematic.

5.3 Summary

This chapter defined the software architecture that turns a single ESP32 gateway into a self reliant control hub for battery powered e-paper badges. It first translated functional goals such as 24 hour battery life and zero external infrastructure into specific design constraints. Guided by those constraints it selected three complementary technologies. A local Wi-Fi access point with an embedded web server gives staff an immediate and universal control channel. RESTful HTTP with JSON uses the existing listener and the browser's native request response pattern while keeping the firmware simple. ESP-NOW then carries each update to a badge with minimal airtime and overhead. The chapter justified each choice by comparing it with familiar alternatives. It concluded with a step by step description of the data flow from a browser request through the gateway to the badge and back. Together these elements form a solid framework that meets the project objectives.

Chapter 6

Prototype realization

This chapter describes the assembly of a prototype e-ink conference badge and a gateway which hosts the webserver, tracing the process from the initial concept to a fully functional device housed in a protective enclosure. It details the technologies employed as well as the components and materials selected.

6.1 Gateway

As stated in Chapter 4, the ESP32 was chosen as the gateway's microcontroller. For the prototype an Espink board was used. Its main advantage is the integrated JST-PH connector for a Li-Po battery and an on-board charging circuit. The board also features a USB-C connector, which is widely adopted and standardized. The PCB footprint measures 75 mm × 28.3 mm.

6.1.1 Gateway enclosure

To protect the gateway and make it easier to handle during testing, an enclosure was designed in Autodesk Fusion 360. The lid is a press-fit that snaps onto the bottom half, eliminating the need for screws or adhesive.

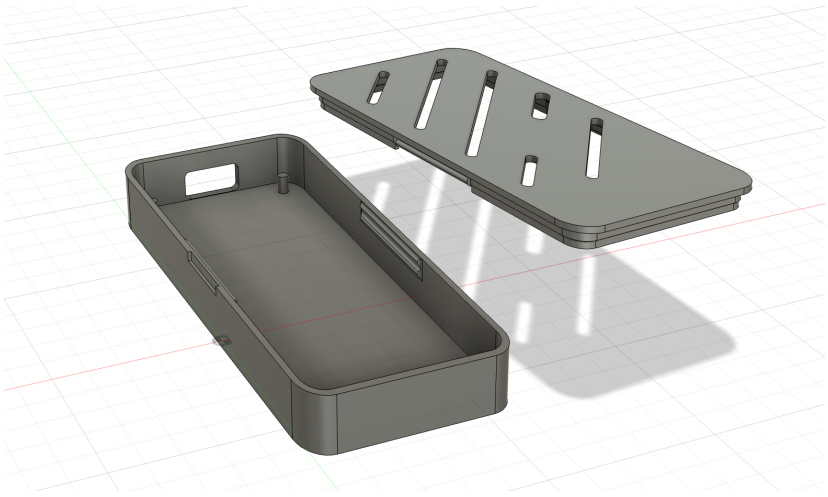

Figure 6.1: Gateway enclosure model in Fusion 360.

The 3D model was exported from Fusion 360 as an STL file and imported into

PrusaSlicer, which converts the geometry into G-code—machine instructions that specify toolpaths, temperatures, and estimated print time. G-code is the standard for additive manufacturing and CNC machining.

The enclosure was printed in PLA (polylactic acid), a cost-effective and beginner-friendly filament that excels at detailed prints and rapid prototypes where high mechanical, chemical, or thermal resistance is not required. Printing was carried out on an Original Prusa MINI+ from the Czech company Prusa Research.

6.2 E-Ink badge

The schematic 6.2 shows a compact system built around an ESPink ESP32 module with an integrated power-management circuit. A bidirectional SPI bus connects the ESP32 to the e-ink display, while a Li-Po battery, plugged into the board's JST-PH socket, supplies power through the same management circuitry, enabling charging, regulation and protection. A USB-C port serves both as an alternate power source for battery charging and as a USB-to-serial interface for programming and debugging.

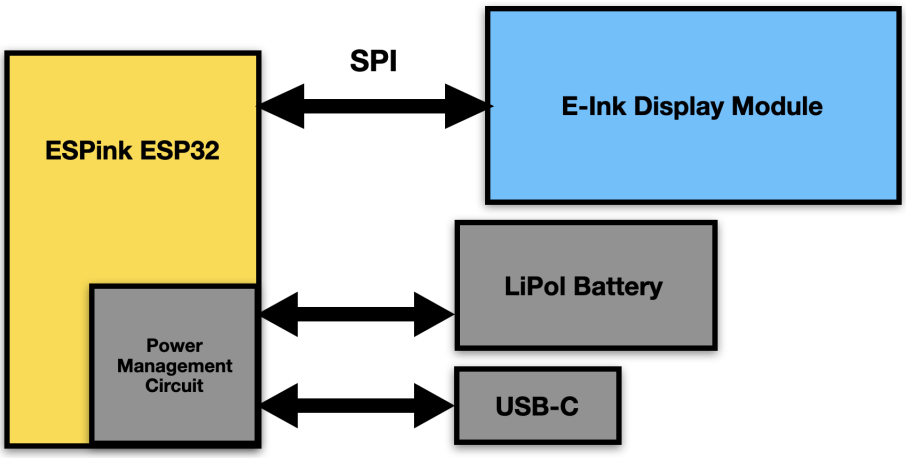

Figure 6.2: Simplified badge schematic.

6.2.1 EPD panel connection

Connector on image 6.3 *J5* is a 24-pin flat-flex cable socket (AFC24-S24FIC-00) that links the ESPink board to the e-ink display.

Pins 11–24 form the display's SPI and control bundle: `MOSI` (IO23), `CLK` (IO18), `CS` (IO5), `DC` (IO17), `RST` (IO16), `BUSY` (IO4) and the timing lines `GDR`/`RESF`. Logic power arrives on pins 9–10 as `EPD_3V3` (decoupled by C9), while pins 3–4 supply the gate-driver bias rails `PREVGH`/`PREVGL` (filtered by C10–C16). All other even-numbered contacts and the shell are grounded for low-impedance return and shielding. In one ribbon, connector *J5* thus delivers the display's data, control, logic power and bias voltages.

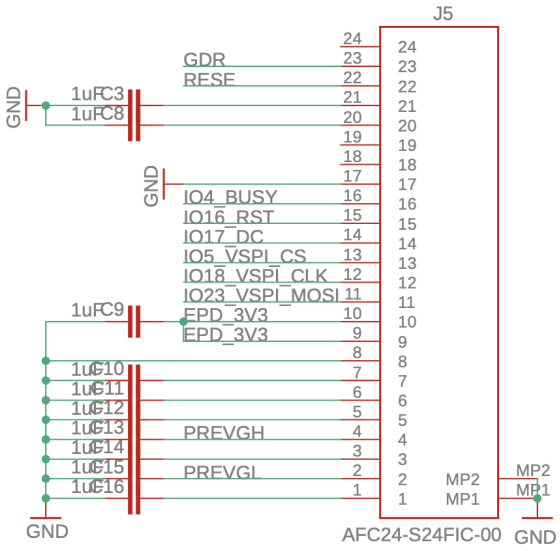

Figure 6.3: EPD connector.[18]

6.2.2 Power management circuit

A Li-Po battery connects through a resettable polyfuse to the main system rail $V+$, which can also be powered from the USB bus. Charging is handled by a **TP4054** linear Li-ion charger, while the regulated logic supply is generated by a **HT7833** low-dropout regulator that provides 3.3 V.

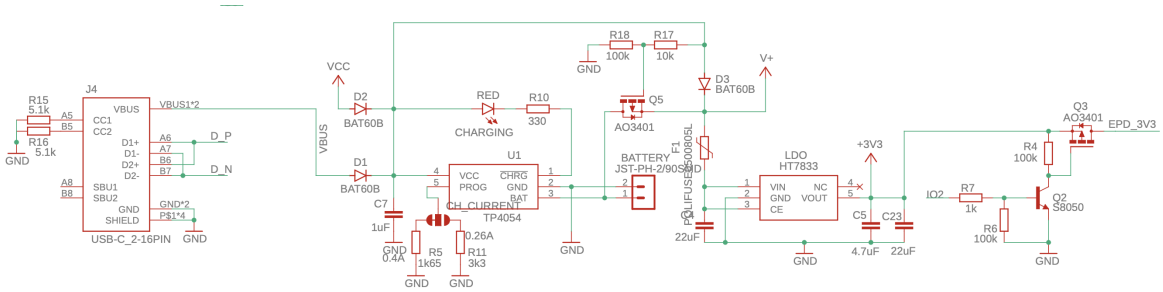

Figure 6.4: Power Management Circuit.[18]

A P-channel MOSFET implements an ideal diode power path so the badge can run from USB even when the battery is full, and a second MOSFET gates the `EPD_3V3` rail under firmware control, disconnecting the e-paper display to minimise quiescent drain. Together, these core devices enable safe charging, seamless USB/battery switchover, and software-controlled power gating, giving the prototype long runtime with minimal external parts.

6.2.3 Badge enclosure

Because the prototype is meant for desk demonstrations, the enclosure on figure 6.5 is a simple two-piece box that frames the display but omits holes or mounting tabs. Modelled in Autodesk Fusion 360, it consists of a shallow tray for the PCB and battery

and a thin front bezel with a large window that leaves the e-ink screen fully exposed. A continuous lip on the bezel mates with a matching groove on the tray, creating a firm press-fit joint, no screws, glue or clips are required, yet the parts can still be split apart for service.

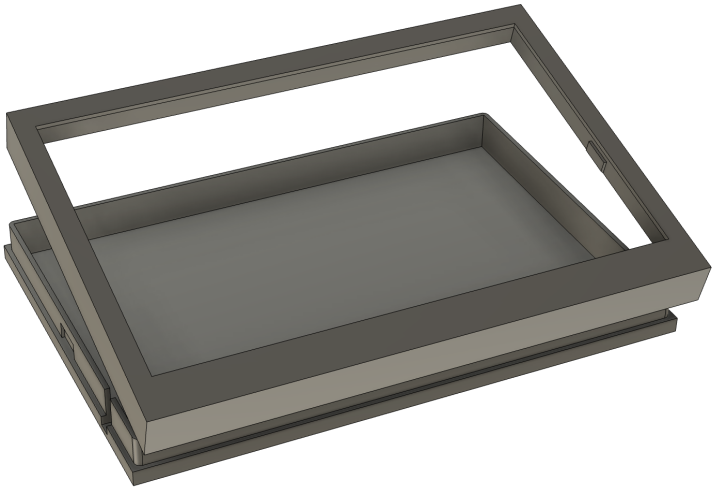

Figure 6.5: E-Ink badge enclosure.

The CAD model was exported as an STL file and converted to G-code in PrusaSlicer. The enclosure was printed in PET-G, chosen for its higher impact and heat resistance compared with PLA. Printing was also carried out on an Original Prusa MINI+, yielding a stiff yet lightweight housing that protects the electronics during handling and transport.

6.2.4 Assembly

Putting the prototype together proved straightforward. The e-paper display was first seated in the four printed corner supports of the front bezel. A thin fixing plate was then laid over the panel to lock it in place. A notch along the plate's edge allowed the display's FFC to snake out and mate with the ESPink board, which was put onto standoffs on top of the same plate. A 2800 mA h Li-Po cell was adhered to the fixing plate beside the ESPink and plugged into its JST-PH header. For development and charging, a USB-C lead was attached to the board before the enclosure was closed. Finally, the rear cover was pressed onto the front frame, completing the press-fit shell. A small cut-out in the back panel lets the USB-C connector remain accessible when needed.

Figure 6.6: Assembled prototype.

Chapter 7

Firmware for badge

Programming the firmware that drives the e-ink badge is built on Espressif's open-source ESP-IDF (Espressif IoT Development Framework). It provides a self-sufficient SDK for any generic application development on these platforms, using programming languages such as C and C++. ESP-IDF currently powers millions of devices in the field, and enables building a variety of network-connected products, ranging from simple light bulbs and toys to big appliances and industrial devices.[24]

All development work for this thesis was carried out in Visual Studio Code using Espressif's official ESP-IDF Extension. The extension bundles the compiler tool-chain, CMake presets, menuconfig UI, serial monitor, tracing tools and an OpenOCD-based JTAG debugger into a single, cross-platform add-on, so the entire build–flash–debug cycle can be executed inside a familiar IDE.

Compared with higher level frameworks such as the Arduino core for ESP32 or PlatformIO's ESP32 platform, ESP-IDF trades a steeper learning curve for much finer control. Arduino abstracts most hardware details behind a simplified API, ideal for rapid prototypes but limiting when low-power modes, custom linker scripts or dual-core task pinning become important.

7.1 Module layout

To keep the firmware maintainable the project is decomposed into five self-contained modules, each delivered as a `.c`/`.cpp` translation unit accompanied by a header that publishes its public API:

- `battery.c` / `battery.h` — wraps the ADC, performs on-chip calibration and converts raw counts into millivolt readings that the rest of the code consumes as a simple floating-point value.
- `wifi.c` / `wifi.h` — brings up the station interface, selects channel 1 and initialises ESP-NOW so higher layers can receive packets.
- `display.cpp` / `display.h` — drives the 7.5" E-Ink panel through CalEPD, decodes incoming JSON or binary logo frames, renders text with font auto-selection, shows a QR-code helper screen and overlays live battery voltage.
- `text_decode_utils.c` / `text_decode_utils.h` — exposes a single helper that strips Czech diacritics from UTF-8 strings so names always render with the built-in bitmap fonts.

- `main.cpp` — it wires the modules together, registers the ISR-level ESP-NOW callback, spawns a worker task fed through a FreeRTOS queue to handle incoming data from the gateway.

This partitioning isolates concerns such as power measurement, networking and rendering, keeps each component logically seperated, and makes the firmware's call graph obvious.

7.1.1 Configuration

```
// Init Wi-Fi and ESP-NOW
wifi_sta_init()

// Create queue for incoming packets
espnow_queue = xQueueCreate(30, sizeof(espnow_evt_t))
assert(espnow_queue)
// Register RX callback function
ESP_ERROR_CHECK(esp_now_register_recv_cb(esp_now_recv_callback))

// Start the worker to handle queue contents
xTaskCreatePinnedToCore(
  espnow_worker_task,
  'espnow_worker',
  4096 /* stack size */,
  NULL,
  2 /* priority */,
  NULL,
  tskNO_AFFINITY
)

// Init ADC for battery measurement
adc_init()

// Power for E-Ink display
gpio_set_direction(GPIO_NUM_2, GPIO_MODE_OUTPUT)
gpio_set_level(GPIO_NUM_2, 1)

// Display default screen
display_start_screen()

```

Figure 7.1: Initial configuration in app_main.

The sequence on figure 7.1 executes at the start of `app_main()`. First of all, `wifi_sta_init()` brings the radio up in Station mode and performs ESP-NOW initialization, giving the badge a connection-less backend. A FreeRTOS queue capable of buffering inbound frames is created, and the ISR-level ESP-NOW receive callback is registered. Every packet landing in the Wi-Fi driver will hence be copied into the queue for deferred processing. Next, a dedicated worker task (`espnow_worker_task`) is spawned. Pinning is left at `tskNO_AFFINITY`, letting the scheduler pick either core on a dual-core ESP32. The analogue front-end is then initialised by `adc_init()`, which configures ADC1-CH6, loads eFuse V_{ref} data and if available, enables line-fitting calibration so subsequent voltage readings are accurate. Finally, GPIO 2, wired to the e-ink panel's power rail is driven high, the driver stack is initialised and `display_start_screen()` renders the default QR-code helper screen along with a live battery readout. From this point the system enters its event-driven loop where the worker task handles all incoming messages and refreshes the display on demand.

7.1.2 Initial screen

The badge's power-on screen is assembled by the function `display_start_screen()` with help from three complementary libraries:

- **CalEPD** — a C++ wrapper around Espressif's SPI driver that exposes the 7.5" e-ink panel through the familiar Adafruit GFX API. The call `display.init(false)` resets the controller, allocates the 480 × 800-pixel framebuffer and selects full-update waveform data. A subsequent `display.setRotation(2)` flips the image so that the badge is in landscape orientation.[25]
- **Adafruit GFX** — inherited by CalEPD, it supplies all drawing primitives used inside the QR callback: `fillScreen()` clears the background, text is positioned with `setTextColor()` and `setCursor()`, while each QR module is painted by a tight `fillRect()` loop.
- **qrcode** — Espressif's QR helper macro `ESP_QRCODE_CONFIG_DEFAULT()` fills a configuration structure that is customised with medium error-correction level and a user-supplied `qr_eink_display` callback. When `esp_qrcode_generate()` is executed, the library streams the QR bitmap line-by-line to that callback.

```
1  void display_start_screen(void)
2  {
3      ESP_LOGI(TAG, "CalEPD version %s", CALEPD_VERSION);
4      display.init(false);
5      display.setRotation(2);
6      display.setTextColor(EPD_BLACK);
7
8      // Configure the QR code parameters.
9      esp_qrcode_config_t cfg = ESP_QRCODE_CONFIG_DEFAULT();
10     cfg.display_func = qr_eink_display;        // Use our custom display callback
11     cfg.qrcode_ecc_level = ESP_QRCODE_ECC_MED; // Set medium error correction level
12
13     // Define the Wi-Fi credentials using the standard QR code format.
14     const char *qrText = "WIFI:T:WPA;S:" EXAMPLE_ESP_WIFI_SSID ";P:" EXAMPLE_ESP_WIFI_PASS ";;";
15     // Generate and display the QR code.
16     esp_err_t ret = esp_qrcode_generate(&cfg, qrText);
17     if (ret == ESP_OK)
18     {
19         ESP_LOGI(TAG, "QR code generated and displayed on the EInk.\n");
20     }
21     else
22     {
23         ESP_LOGE(TAG, "Failed to generate QR code. Error: %d\n", ret);
24     }
25 }
```

Figure 7.2: Initial screen setup.

The initial screen prints a concise three-step guide so a first-time user can bring the badge to life without opening any manual. First, it invites the user to connect a phone or laptop to the badge gateway's Wi-Fi network, showing the SSID and password in clear text and, for convenience, embedding the same credentials in a QR code that most camera apps will recognise instantly. Second, it instructs the user to open a browser and navigate to the built-in configuration site, reachable either by the badge's numeric IP address or by its mDNS host name suffixed with *.local*. Third, once the page loads, the user is asked to register the physical badge by entering the MAC address that the firmware renders right below the instructions.

Figure 7.3: Badge with initial screen.

7.1.3 ESP-NOW message receive

When a frame arrives, the interrupt routine `esp_now_recv_callback()` runs first. It throws away any packet that is too large, packs the sender's MAC address, length, and data into a small structure, and puts that structure on a FreeRTOS queue with `xQueueSendFromISR()`. That task is `espnow_worker_task()`. The task then waits on `xQueueReceive()`, and each time it pulls a message from the queue it calls the display function to parse the data and update the e-ink screen. This split keeps the interrupt code fast while letting the slower screen update run safely in normal task context.

7.1.4 Display function

The `display_message_data()` is responsible for all display updates that arrive over ESP-NOW. First it decides whether the incoming bytes represent a JSON command or a chunk of raw image data. For JSON it copies the payload into a temporary buffer, turns it into an object, and backs out immediately if the text cannot be parsed. A message that contains only the keyword "clear" erases the screen, resets the logo buffer, and returns. Otherwise the code pulls out the first name, last name, and any extra line of text, strips diacritics so every character is supported by the built-in fonts.

With the display supply rail enabled it paints a white background, chooses fonts that fit each line within the panel width, and prints the lines centred one below another, the extra line is placed near the bottom in a fixed, bold typeface. Before refreshing the panel it also measures the battery voltage and writes the value in small text at the top-left corner. All temporary strings and the JSON object are then released to avoid memory leaks, the screen is refreshed, and the power rail is switched off.

```c
static void esp_now_recv_callback(const esp_now_recv_info_t *info, const uint8_t *data, int len)
{
    if (len > ESPNOW_MAX_PAYLOAD)
        return;

    espnow_evt_t evt;
    memcpy(evt.mac, info->src_addr, ESP_NOW_ETH_ALEN);
    evt.len = len;
    memcpy(evt.data, data, len);

    BaseType_t xHigherPrioTaskWoken = pdFALSE;
    xQueueSendFromISR(espnow_queue, &evt, &xHigherPrioTaskWoken);

    // If the queue send woke a higher-prio task, request a context-switch
    if (xHigherPrioTaskWoken)
        portYIELD_FROM_ISR();
}
// Task that handles incoming messages from the queue
static void espnow_worker_task(void *arg)
{
    espnow_evt_t evt;

    while (true)
    {
        if (xQueueReceive(espnow_queue, &evt, portMAX_DELAY) == pdTRUE)
        {
            display_message_data(evt.data, evt.len);
        }
    }
}
```

Figure 7.4: ESP-NOW receive callback and worker task.

If the data does not look like JSON it is treated as part of a monochrome bitmap that will become a full-screen logo. Each chunk is copied into a rolling buffer as long as the total size stays within limits. A length check prevents buffer overflow and, if the limit would be exceeded, the partial image is discarded with an error message. When the final chunk arrives and the buffer holds a complete frame, the routine powers the panel, clears it to white, draws the bitmap at the origin, refreshes the display, and finally turns the power back off. The logo buffer pointer is then reset so the next transfer can start from zero. In this way the function handles both quick text updates and larger image uploads.

7.1.5 Battery voltage measurement

The battery line is fed to the ADC through a fixed resistor divider made of 1 MΩ and 1.3 MΩ resistors. This scales the cell voltage by about 1 / 1.77, keeping even a fully-charged LiPol pack inside the ESP32's 3.3 V input limit. Because both resistors are in the mega-ohm range the divider draws less than 2 µA, so it does not affect the badge's standby life. A 100 nF capacitor ties the centre node to ground, that smooths noise and lets the ADC sample a clean, low-impedance signal.

The battery-monitor code begins by defining which calibration method to try, which ADC channel is wired to the cell, and what resistor-divider ratio converts pin voltage to true pack voltage.

During start-up it creates a one-shot ADC driver instance for the first ADC unit, assigns the chosen channel, and applies a 12 dB attenuation so the full battery range fits inside the converter's input window. It then attempts to enable the line-fitting calibration scheme: if the eFuse on the chip contains reference-voltage data, a software helper is installed that can translate raw counts into millivolts with much better accuracy than the default slope. Success or failure is noted only in a Boolean flag. The rest of the firmware continues to work even on uncalibrated parts. A companion routine

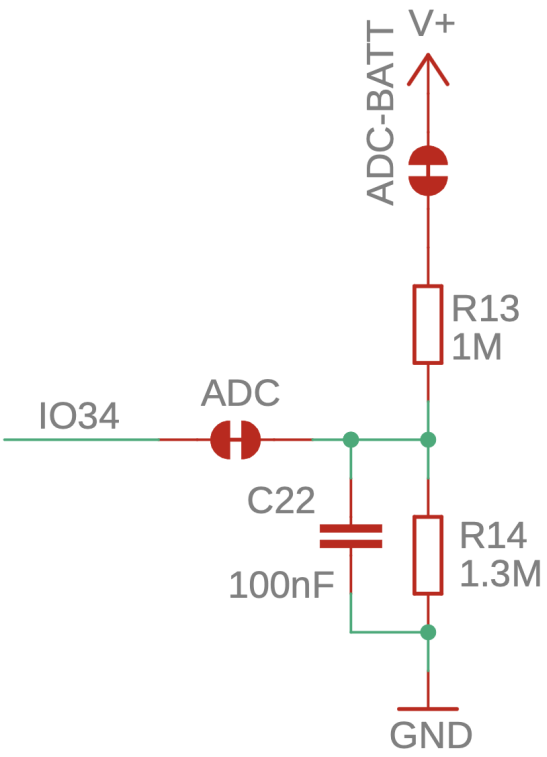

Figure 7.5: Voltage divider on IO34.

exists to tear down the calibration handle later, keeping the driver free of memory leaks.

When the firmware needs a reading it commands the ADC to perform a single conversion on the battery channel and captures the raw 12-bit result. If calibration was set up earlier the raw code is immediately turned into a calibrated voltage value. Because the battery is connected through a resistor divider, that pin voltage is multiplied by the pre-computed ratio to recover the actual cell voltage. The function logs both the unscaled sample and the corrected voltage, then returns the latter in millivolts. Should calibration be unavailable, the routine simply returns zero so higher layers can decide how to handle the missing data.

Chapter 8

Firmware for gateway

The gateway firmware for the smart e-paper infrastructure is built, like the badge firmware described in the previous chapter, with Espressif's ESP-IDF SDK and it is written, built, flashed, and debugged entirely from inside Visual Studio Code using Espressif's official IDF extension.

8.1 Module layout

The project is, like in the case of the badge firmware, divided into several translation units.

- `main.c` - application entry point. It initialises NVS and calls `wifi_init_softap()` to setup the access point.
- `wifi.c` / `wifi.h` - It configures the radio as a 2.4 GHz access point, starts the mDNS responder, launches the web server with a single call to `start_webserver()`, initialises ESP-NOW.
- `webserver.c` / `webserver.h` - built around ESP-IDF's lightweight HTTP daemon. The module registers URI handlers for a handful of endpoints. It serves the embedded single-page application, accepts POST requests that contain raw frames or text data, and queues finished images for the radio task. Only one public symbol is exported: `httpd_handle_t start_webserver(void);`
- `webcontent.html` - the complete browser front-end. Written in plain HTML, CSS and JavaScript, it renders the badge editor GUI, rescales user images to fit the E-Ink badge, and issues AJAX calls to the REST endpoints declared in `webserver.h`. Keeping the file as static HTML avoids extra build time dependencies such as Node or TypeScript.
- `html_gen.sh` - a build helper that converts the front-end html file into a C include file: `xxd -i webcontent.html > web_content.h`
- `web_content.h` - auto-generated output from the script above. It holds two symbols, `webcontent_html[]` and `webcontent_html_len`, allowing the index handler in `webserver.c` to serve the page with a single call to `httpd_resp_send()`.

8.1.1 Configuration

First, `nvs_flash_init()` opens the on-chip key–value store, if the partition is full or from an older SDK version it is erased and re-initialised so subsequent reads and writes succeed. The global ESP-IDF event loop is then created with function call `esp_event_loop_create_default()`, giving the application a place where all Wi-Fi, IP and ESP-NOW events will later be handled.

Next, the radio is brought up by `wifi_init_softap()` . In sets up a 2.4 GHz access point named `Meetink Controller`, enables the built-in DHCP server and registers a Wi-Fi event handler that logs every station connect or disconnect. A common function called `connect_handler()` will run once an IP address is available.

```
1 void app_main(void)
2 {
3     static httpd_handle_t server = NULL;
4
5     // Initialize NVS
6     esp_err_t ret = nvs_flash_init();
7     if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
8     {
9         ESP_ERROR_CHECK(nvs_flash_erase());
10        ret = nvs_flash_init();
11    }
12    ESP_ERROR_CHECK(ret);
13    ESP_LOGI(TAG, "Starting in Access Point mode");
14    // Init AP
15    wifi_init_softap();
16    // Init ESP-NOW and register callbacks
17    init_esp_now();
18    // When IP assigned, start webserver
19    ESP_ERROR_CHECK(esp_event_handler_register(IP_EVENT,
20                                               IP_EVENT_AP_STAIPASSIGNED,
21                                               &connect_handler,
22                                               &server));
23 }
```

Figure 8.1: Initial configuration in app_main.

Right after Wi-Fi starts, `init_esp_now()`, registers a send callback and adds exactly one peer record whose MAC is the universal broadcast address. From this moment the gateway can push 250-byte datagrams to every badge in range without using TCP or UDP. A FreeRTOS queue is created to hold outbound chunks and a worker task, `send_logo_task`, is spawned with default stack size.

Higher-level services are only started after networking is confirmed. When connect handler fires it first calls `start_webserver()`, which launches ESP-IDF's HTTP daemon, registers every REST endpoints and begins serving the front-end that was embedded in `web_content.h`. Immediately afterward `init_mdns()` advertises the host under the label `meetink.local`, enabling zero-configuration discovery from browsers.

Once these steps complete `app_main()` returns and the firmware becomes entirely event-driven. Each HTTP POST received at `/upload` is parsed by `webserver.c`, passed through the encoder, split into radio-sized pieces and enqueued. The worker task drains the queue and feeds the chunks to `esp_now_send()`.

8.1.2 Webserver code

The gateway's webserver is a thin wrapper around ESP-IDF's native HTTP daemon. A build-time shell script converts the single-page `webcontent.html` file into

a C array, so the landing page lives in flash and can be served with one zero-copy `httpd_resp_send()` call, eliminating the need for SPIFFS or FAT.

During that first GET the firmware replaces a placeholder in the HTML template with a fragment generated on the fly that lists every badge stored in NVS, effectively doing template rendering with `snprintf()` and a small scratch buffer. Each badge block in the rendered page contains a text form, a file input and four buttons, all wired to JavaScript functions that hit REST endpoints exported by the same C module. Endpoints that accept structured data (`/addmac`, `/sendtext`, `/deletemac`, `/clearbadge`) expect a JSON body and rely on the tiny `cJSON` library, avoiding the overhead of a full HTTP parser.

```
async function renderAndThreshold(file, canvas) {
    const img = new Image();                          // create <img> holder
    img.src = URL.createObjectURL(file);              // point to local blob URL
    await img.decode();                               // wait for decode

    const ctx = canvas.getContext('2d');              // 2-D drawing context
    ctx.fillStyle = 'white';                          // white background
    ctx.fillRect(0, 0, EINK_W, EINK_H);               // clear canvas

    const scale = Math.min(EINK_W / img.width,        // preserve aspect
                           EINK_H / img.height, 1);
    const dw = Math.round(img.width * scale);         // dest width
    const dh = Math.round(img.height * scale);        // dest height
    const dx = (EINK_W - dw) >> 1;                    // x offset
    const dy = (EINK_H - dh) >> 1;                    // y offset
    ctx.drawImage(img, dx, dy, dw, dh);               // draw scaled image

    const data = ctx.getImageData(0, 0,               // RGBA buffer
                                  EINK_W, EINK_H).data;
    for (let i = 0; i < data.length; i += 4) {        // iterate pixels
        const avg = (data[i] + data[i+1] + data[i+2]) / 3; // grayscale
        const v = avg < 128 ? 0 : 255;                // threshold
        data[i] = data[i+1] = data[i+2] = v;          // set RGB
    }
    ctx.putImageData(new ImageData(data, EINK_W, EINK_H),0, 0);
}

```

Figure 8.2: JavaScript image convert function.

The browser script converts any images into bitmaps sized to match the resolution of the badge. The conversion is shown in the figure 8.3. When a file is chosen `renderAndThreshold()` decodes it, scales and centres it on the canvas, then converts every pixel to pure black or white using a fixed 128-level threshold. A helper `canvasToBytes()` walks that monochrome canvas and packs eight successive pixels into one bit of a `Uint8Array`.

Pressing *Send logo* runs `uploadLogo()`, which prepends the target MAC address, wraps header plus bitmap in a single binary `Blob`, and posts it to `/sendlogo`. A callback pops up a success or failure alert. Lighter actions (`/addmac`, `/sendtext`, `/deletemac`, `/clearbadge`) use `fetch()` with small JSON bodies that the firmware parses with `cJSON`. On `DOMContentLoaded` the code scans every badge block injected by the server, attaches a `change` listener so a preview appears as soon as an image is picked, and enables the *Send logo* button only after a valid preview exists. Offloading all image processing to the browser keeps the gateway lean and ensures that only the pre-packed radio payload crosses the network.

Upon receipt the handler reads the first line to learn which badge should receive the frame. A FreeRTOS queue decouples HTTP latency from radio latency. The handler

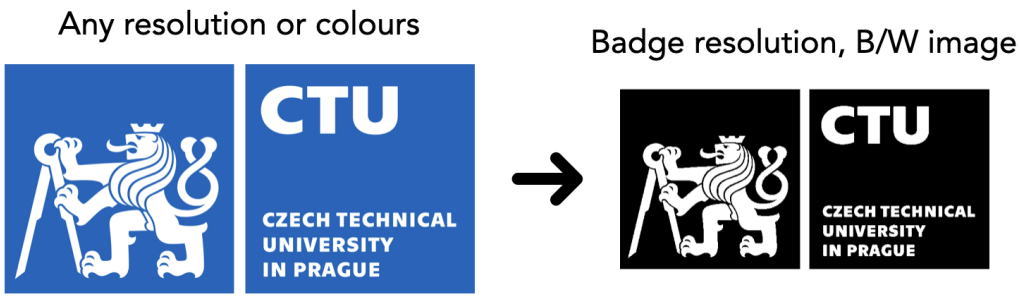

Figure 8.3: JavaScript image conversion and rescaling.

enqueues a pointer to the frame and returns HTTP 200 immediately, while a dedicated worker task slices the buffer into 250-byte chunks and feeds them to `esp_now_send()`.

8.1.3 Webserver UI

The page opens with a centred card headed `Badge Editor`, rendered on a diagonal blue gradient that spans the full viewport. Inside the card a pale-blue "surface" panel groups all controls that belong to a single badge. The panel shows the badge's MAC address in bold at the top, immediately anchoring the context for every command that follows. On the left a vertical form accepts first name, last name and an optional line of additional info, each field styled as a rounded white input against the blue background.

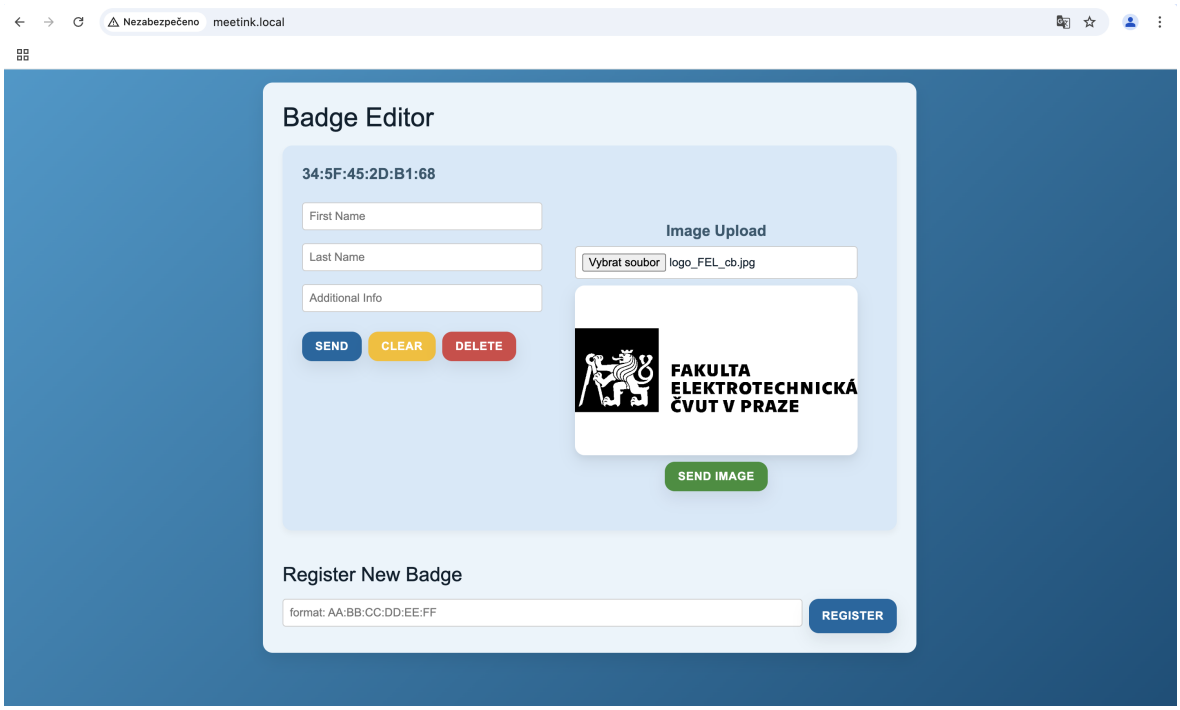

Figure 8.4: Webserver interface on PC.

Below the text fields three colour-coded buttons provide the common actions: blue *SEND* pushes the text frame, amber *CLEAR* wipes the e-ink screen and red *DELETE*

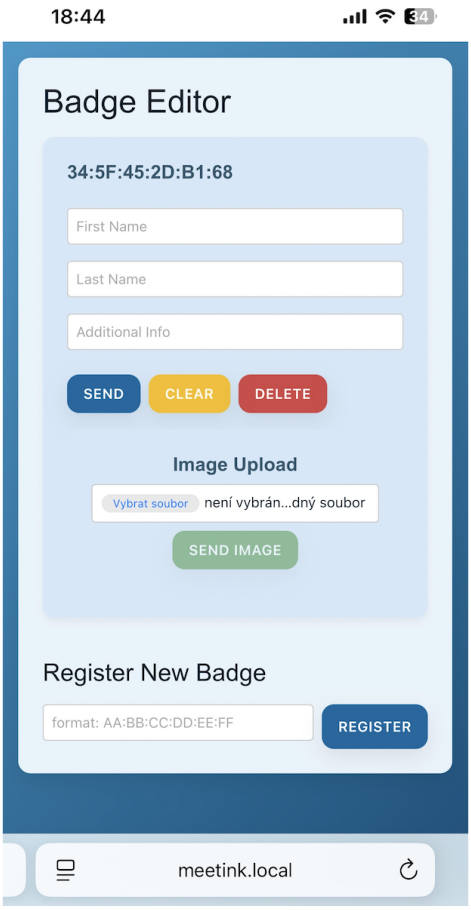

Figure 8.5: Webserver interface on a smartphone.

removes the badge from the list. The right-hand column carries the image workflow: a file input labelled *Image Upload* picks any common file type, the chosen picture is previewed on a light card with subtle shadow, and a green *SEND IMAGE* button appears once the thumbnail is ready. All buttons use bold uppercase text, rounded 12 px corners and a consistent drop shadow so they pop from the background and share a uniform style.

Below the badge block a second card titled *Register New Badge* lets the operator enter a new MAC address and press a *REGISTER* button. Generous padding, 16 px gaps and a single sans-serif font keep the layout airy and readable even on narrow screens and CSS variables ensure the same colour scheme is reused sitewide.

8.1.4 ESP-NOW image send

When the browser posts to `/sendlogo` it sends a MAC address followed by the packed bitmap. The MAC string is parsed into six bytes with `parse_mac()`, and malformed addresses trigger a HTTP 400 error.

Next, the remaining bytes are streamed directly into the global `logo_buf` buffer while the handler continues to guard against timeouts and short reads. As soon as the last byte lands the function replies “200 OK” so the client connection can close without waiting for radio traffic. Before returning it allocates a tiny heap block that holds the peer MAC and payload length and launches `send_logo_task()` with priority 5. The

```c
static void send_logo_task(void *arg)
{
    struct
    {
        uint8_t addr[6];       /* destination MAC */
        size_t  len;           /* payload length */
    } *p = arg;

    if (!p) { vTaskDelete(NULL); }          /* guard against NULL pointer */

    size_t total  = p->len;                 /* bytes to send in total    */
    size_t offset = 0;                      /* current position in buffer*/
    int    pid    = 0;                      /* running chunk index       */

    while (offset < total)                  /* loop until whole payload sent */
    {
        /* choose next slice, capped at ESP-NOW MTU (250 B) */
        size_t chunk = (total - offset > ESP_NOW_MAX_DATA_LEN)
                       ? ESP_NOW_MAX_DATA_LEN
                       : (total - offset);

        /* transmit the slice and check status */
        esp_err_t err = esp_now_send(p->addr, logo_buf + offset, chunk);
        if (err != ESP_OK)
        {
            ESP_LOGE(TAG, "ESP-NOW pkt %d failed: %d", pid, err); /* log failures */
        }

        offset += chunk;                    /* advance offset            */
        pid++;                              /* increment packet counter  */
        vTaskDelay(pdMS_TO_TICKS(10));      /* brief pause to share airtime */
    }

    /* print completion message */
    ESP_LOGI(TAG, "All %d chunks sent to %02X:%02X:%02X:%02X:%02X:%02X",
             pid, p->addr[0], p->addr[1], p->addr[2],
             p->addr[3], p->addr[4], p->addr[5]);

    free(p);                                /* release argument buffer   */
    vTaskDelete(NULL);                      /* delete self, free stack   */
}

```

Figure 8.6: ESP-NOW send image task.

task iterates over the bitmap, chopping it into 250 byte chunks, calls `esp_now_send()` for each piece, logs any failure, and inserts a 10 ms delay between packets to avoid airtime bursts. After the final chunk it prints a summary line, frees its argument structure, and erases itself with `vTaskDelete(NULL)`.

8.1.5 ESP-NOW text send

The handler `sendtext_post_handler()` begins by allocating a heap buffer large enough to hold the entire HTTP body. It then calls `httpd_req_recv()` to pull the raw bytes from the socket, adds a NUL terminator, and converts the result into a cJSON tree. The routine extracts the key `mac` and validates its colon-separated format with `sscanf()`, turning the six hex pairs into a `uint8_t[6]` array. Optional strings `first_name`, `last_name` and `additional_info` are fetched next, and the handler rejects the request if every field is empty so that the radio never transmits a pointless packet. Logging lines record the destination MAC together with the three strings for debugging over the serial console. A fresh cJSON object is created, the three fields are inserted, and `cJSON_PrintUnformatted()` flattens it into a compact null-terminated buffer ready for transmission. The payload travels to the badge with `esp_now_send()`, whose return code is converted to human-readable text via `esp_err_to_name()` and placed in a tiny JSON reply to the browser. Finally the function sets sends the response, frees all temporary buffers, and returns the status of the HTTP write back to the webserver.

8.1.6 Storing badges to memory

At startup the firmware scans its NVS namespace for keys that match the pattern `mac_#` and builds an list of registered badge addresses. For every entry it emits a short HTML fragment—label, three text fields, file input, preview canvas and the row of action buttons—and concatenates the fragments into one string. When the main page template is served that string is simply substituted for the placeholder in the html page, giving the browser a populated list of registered badges.

Adding a badge is handled by the `/addmac` endpoint. The browser posts a JSON body with a single `mac` field, the handler validates the six colon separated hex bytes, rejects duplicates, and commits the address into NVS under the next free key. It then invalidates the cached MAC list so the next page load will render the block with the new badge in place.

Removal follows the reverse path. The `/deletemac` handler checks that the requested address exists, erases the corresponding NVS key, compacts the remaining entries to keep key indices contiguous. Because both handlers reply with a bare `200 OK` the client can simply refresh after success. The page will regenerate and the updated list appears instantly. Storing MACs in flash rather than RAM ensures persistence across reboots, while caching the rendered block avoids slow string assembly for every request.

8.1.7 Badge clear

The browser issues a `POST /clearbadge` containing a single JSON key — `"mac"` with the target badge address. The handler reads the body into a buffer, parses it with cJSON and immediately rejects requests that lack a valid colon separated MAC. After converting the string into six binary bytes it builds a miniature JSON object `{"clear":"1"}` and flattens it into an unformatted byte buffer. That buffer is sent to the badge over ESP-NOW, signalling the badge firmware to blank its e-ink display. The handler then returns a JSON reply whose single field `"status"` contains the human-readable result of `esp_now_send()`, so the frontend can show success or error.

Chapter 9

Prototype testing

This chapter outlines the prototype testing process, covering the system's start-up routine and the integration of additional displays. It reports measurements of transmission range and endurance, and concludes with an assessment of the system's overall performance.

9.1 Setup of the system

Firmware needs to be uploaded to both the gateway and the badges. Gateway and at least one client badge are needed to run the system. Up to 20 badges can be connected to the system.

9.1.1 Gateway

For the correct functionality of the gateway, it is necessary to connect it using USB to power. Optionally, it is possible to connect the gateway using the JST-PH port to the battery. Once powered, it automatically launches a Wi-Fi access point named `Meet Ink Controller`.

Figure 9.1: Gateway connected to power.

9.1.2 Client badge

Badge already contains a battery, but can also be connected using a USB cable to power. Badge is placed in 3D printed legs for stability and good readability. On the badge, the initial screen with setup info is displayed, as it was shown on figure 7.2. After placing the badge in a suitable place, the setup is completed.

Figure 9.2: Badge placed on table.

9.2 Connecting new client

The next step involves establishing a connection to the designated Wi-Fi access point and then loading the web-server page. This connection can be made through any device equipped with Wi-Fi capabilities and a web browser, such as a smartphone, laptop, or tablet.

9.2.1 Wi-Fi access

Wi-Fi access can be obtained manually with the credentials displayed on the initial screen or automatically by scanning the QR code presented on the screen.

9.2.2 Opening webserver

After a successful Wi-Fi connection and browser launch, the web server can be reached in two ways: either by entering its HTTP IPv4 address directly or by using its ".local" hostname though the latter may not be supported on all devices. Both addresses are displayed on the initial screen.

9.3 Adding a new badge

Once the web server has loaded, the badge must be registered both as a client stored in non-volatile flash memory and as an ESP-NOW peer to enable proper communication. This one time procedure is simple: the badge's address, shown on the initial screen, is entered into the text field on the web-server form, illustrated in figure 9.3.

A successfully registered badge is then listed, as illustrated in figure 9.4.

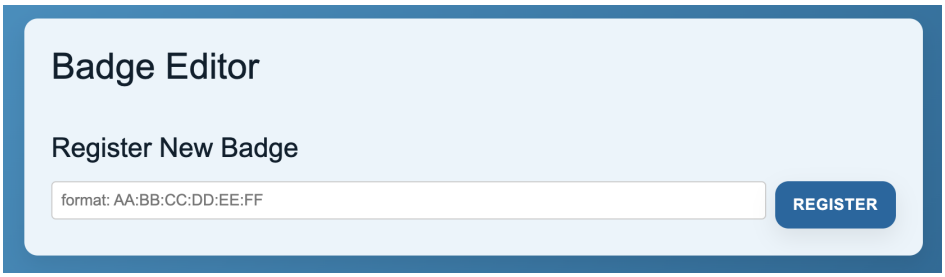

Figure 9.3: Badge registration form.

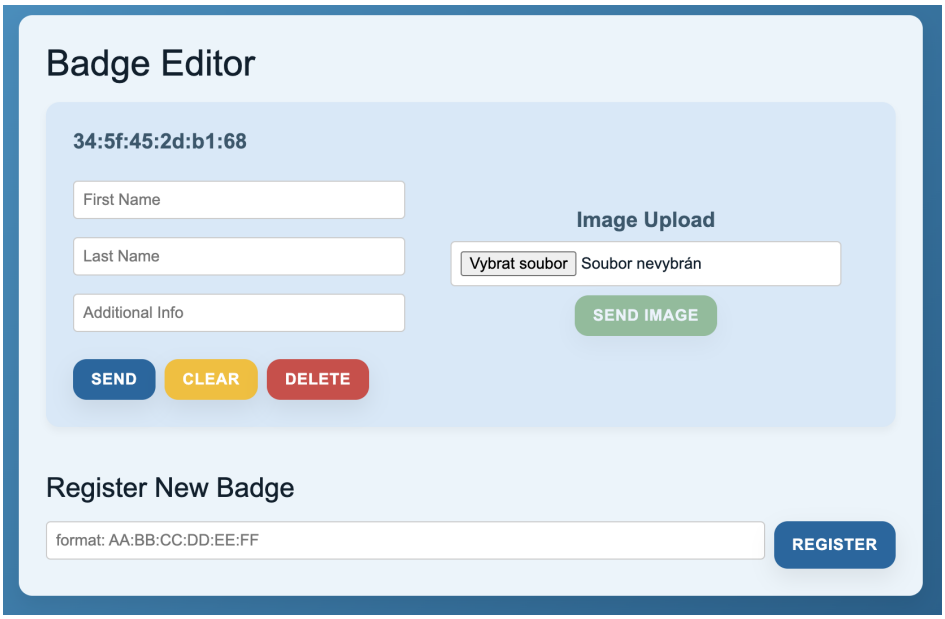

Figure 9.4: Successfully registered badge.

9.4 Removing badge

A badge can be removed from the list by selecting the Delete button and confirming the browser pop-up. This action erases the address from flash memory and simultaneously deregisters the peer from ESP-NOW communication.

9.5 Testing in real environment

9.5.1 Scenario A: Meeting room

In the first test scenario, measurements were conducted in a large meeting room measuring approximately 30 m × 6 m. The environment was not radio quiet, several other Wi-Fi transceivers operated nearby introducing background interference. For the trial, the gateway and the badge were positioned at opposite ends of the room in a distance of about 25 m.

During this trial the system underwent an extensive functional stress test: over several sessions, dozens of transmissions comprising both text packets and compressed images were dispatched to the client badge while separating the gateway and badge at distances ranging from roughly 10 m to the full 30 m span of the room. Traffic origi-

Figure 9.5: Meeting room in scenario A.

nated from a representative mix of devices, including Android and iOS smartphones as well as laptops running macOS and Windows. Across this entire matrix of distances, payload types, and operating platforms, every send command was acknowledged by the badge without loss or retransmission, confirming stable end to end performance under typical indoor interference conditions.

9.5.2 Scenario B: Long distance

Scenario B repeated the indoor trials under typical background Wi-Fi interference, but with the gateway and badge separated by approximately 70 m in clear line of sight, unobstructed by any physical barriers. Despite the increased distance, every packet sent from Android and iOS smartphones as well as macOS and Windows laptops was delivered and acknowledged without loss, confirming full reliability across all tested platforms.

9.5.3 Battery life

A Lithium polymer battery rated at 4000 mAh and 3.7 V powered the badge for more than 30 hours while the ESP32 radio continuously listened for ESP-NOW traffic which is already adequate for the target use. Endurance can be increased either by turning the badge off during idle periods or by introducing a wake window scheme with `esp_now_set_wake_window()` so the chip sleeps between periodic wake ups whose duration equals the chosen window size.[26] This strategy requires the application layer to maintain time alignment between the gateway and each badge because the client cannot initiate pulls and the gateway transmits at irregular moments.

9.6 Communication range of ESP-NOW

The maximum distance over which an ESP-NOW link remains reliable depends on several radio-frequency factors: the selected physical layer data rate, the transmit power limit, antenna gain, and the amount of indoor or outdoor obstructions.

9.6.1 Retransmission and Reliability

ESP-NOW supports two delivery styles, *unicast* and *broadcast*, each with its own retransmission behaviour.

Unicast Frames

1. **Acknowledgement.** Every unicast packet requests a MAC-layer ACK from the peer.
2. **Automatic retries.** If the sender does not see an ACK, it retries up to seven times (count configurable in the ESP-IDF API), using a random back off to limit collisions.
3. **Duplicate handling.** Because retries occur below the application, an upper layer can occasionally receive duplicates.

Broadcast Frames

1. **One time delivery.** Broadcast packets (`ff:ff:ff:ff:ff:ff`) never request ACKs.
2. **No hardware retries.** The radio sends the frame once, if a device misses it, recovery must be arranged at the application level, for example by repeating the message or polling recipients with unicast.

9.6.2 Outdoor test using broadcast frames

To quantify the reliability of pure *broadcast* traffic, an experiment was carried out by Chris Greening using ESP32 with PCB antennas[27]. The measurements were collected outside in park with some obstacles like trees or bushes. All data were sent as broadcast frames, so the radio performed no automatic retries or acknowledgements.

Results. Figure 9.6 reveals that, without retransmission, packet loss rises rapidly with distance. Beyond 350 m every broadcast frame was lost. The data underscore the importance of acknowledgements and retries for links exceeding roughly 100 m. While occasional low loss bursts are possible out to 300 m, the mean performance is insufficient for reliable messaging. In practical deployments, either

1. Fall back to unicast mode (which provides hardware ACK and up to seven automatic retries), or
2. Introduce an application level redundancy scheme (e.g. repeated broadcasts with sequence checking),

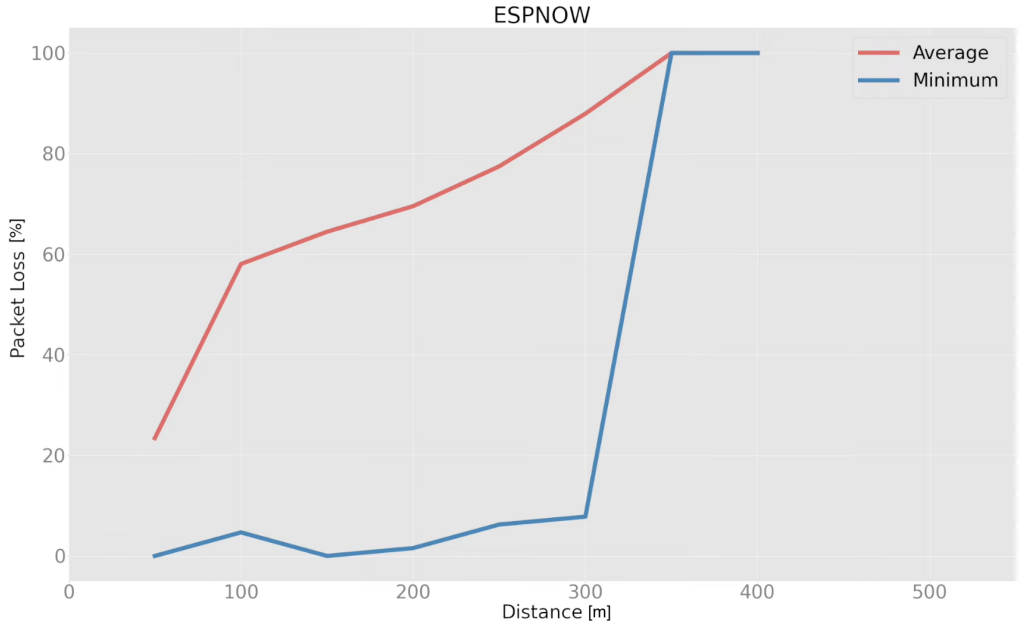

Figure 9.6: Packet loss versus distance for outdoor broadcast traffic.[27]

9.6.3 Long range test using ESP-NOW-LR

ESP-NOW-LR is Espressif's "low-rate" variant of ESP-NOW. Because `ESP-NOW-LR` lowers the data rate and widens the modulation index, it is expected to perform better under weak signal conditions than the default mode. To evaluate its practical benefit, a second test was performed in the park.

Results. Figure 9.7 indicates that LR mode improves the best case delivery rate at extended range. Compared with the default, LR mode pushes the *occasional* successful broadcast well past 300 m.

With standard ESP32 modules with PCB antennas, ESP-NOW typically reaches 50–100 m indoors and 200–300 m outdoors. ESP-NOW-LR push links to near 600 m range in open areas. Unicast traffic benefits from built in ACKs and retries, broadcast traffic is faster but requires higher layer redundancy if guaranteed delivery is needed.

9.7 Summary

The chapter explained prototype preparation testing and evaluation. After flashing firmware the USB powered gateway creates a WiFi access point and up to 20 e-paper badges are added through a browser form that stores each address in flash memory and pairs it as an ESP NOW peer. Badges can also be removed through the same interface with a confirmation dialog. Indoor trials in a 30 m x 6 m meeting room showed zero packet loss for text and compressed images over distances up to 30 m using Android and iOS phones plus macOS and Windows laptops. A second experiment placed gateway and badge about 70 m apart in clear line of sight and again every transmission was acknowledged. The text then compares ESP NOW unicast which includes hardware acknowledgments and up to seven retries with broadcast which lacks automatic recovery. Outdoor broadcast measurements reveal rising loss beyond about 100 m and complete

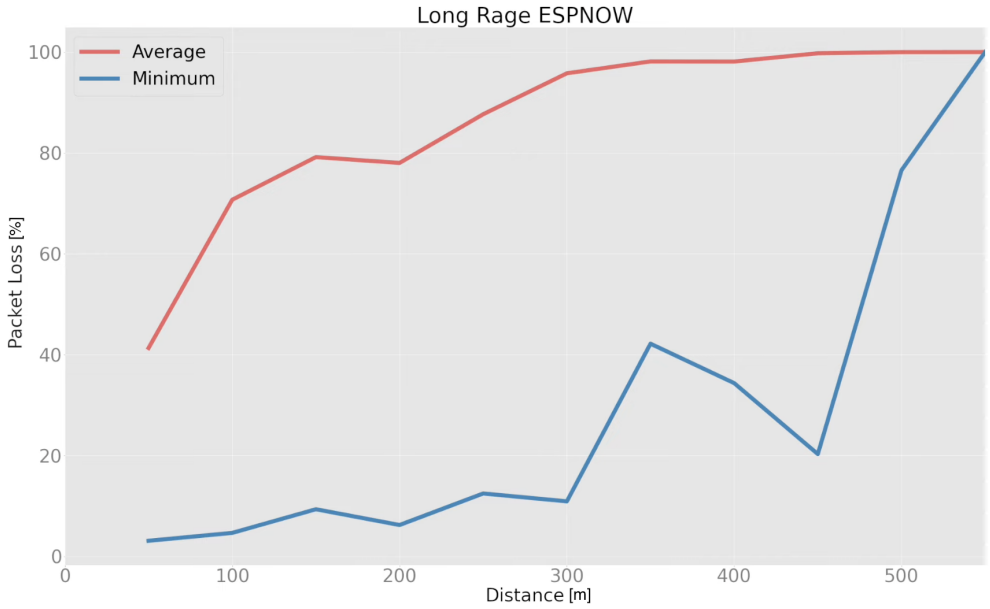

Figure 9.7: Packet-loss versus distance for ESP-NOW-LR outdoor broadcast traffic.[27]

failure past 350 m. Using ESP-NOW-LR mode occasional successful broadcasts were observed near 600 m.

Chapter 10

Evaluation of achieved results

This chapter summarizes the results achieved and compares them with commercially available products. At the same time, the costs for building a prototype of the badge and the gateway are calculated.

10.1 Prototype production costs

Table 10.1 summarizes the costs of individual items. The total cost of the badge prototype parts is approximately $67.00. The cost of the gateway itself was $17.00. The price could be reduced even further if, instead of the ESP32 development kit, the ESP32 module itself were used, which would be mounted directly on the custom designed printed circuit board. It would also be possible to save on the purchase of e-ink panels. Buying them in small quantities at retail is disadvantageous. It would be more profitable to negotiate larger deliveries directly with the manufacturer.

Table 10.1: Price of used components for badge

Item	Seller	Price [$]
GDEY075T7 7.5" display	Good Display	39,00
ESPink ESP32 board	Laskakit	17,00
GeB LiPol Battery 4000mAh	Ebay	9,00
3D printed case	Own	2,00
	Total	**67,00**

The total price does not include development costs, which would significantly increase the total price. The reason is that prototype production is quite time-consuming and it would be difficult to adequately estimate a specific amount.

10.2 Comparison with commercial solutions

In the following section, the e-paper badge prototype is compared with commercially available products. Individual aspects are compared and the advantages and disadvantages of the test prototype are summarized.

10.2.1 Comparison in terms of required features

Although it is a prototype, its price is comparable and in some cases even significantly cheaper than the commercial variants.

The badge prototype is a battery powered device with a badge that exposes its own Wi-Fi access point and lightweight web server. Any standard browser can connect to this interface to upload new images or text, so the badge updates without internet connectivity, mobile apps, or third-party servers. Because the control logic and power source are both on the device, it meets all three project requirements: autonomous operation, zero proprietary infrastructure, and remote rewriting.

Infsoft's 7.5-inch E-Ink beacon content updates pass through the LocAware cloud platform, and the beacon will not communicate without an additional Locator Dongle. The need for both the dongle and a cloud platform breaks the standalone and open infrastructure criteria, although the cloud does permit remote screen changes.

Good Display's ET0750 uses a BLE 5.0 protocol and can only be configured through an Android application or a dedicated gateway sold by the manufacturer. Furthermore, the ET0750 rewrite process is not clearly defined. So there is no certainty, if the updates can be done remotely and what is the maximum range.

Taiden's HCS-1085 nameplate combines two 7.5-inch tri-colour displays. All management traffic must flow through the HCS-1085T adapter via BLE 4.2. The adapter, in turn, connects to Taiden's closed backend. Without that adapter the badge cannot be updated, so it fails the stand alone and open infrastructure requirements. Furthermore, the price is not listed on manufacturer's site.

DSPPA's D7642 provides dual 7.5-inch screens and can only be configured through a proprietary application and needs a Bluetooth gateway.

In summary, the prototype is the only device in this group that operates with no external hardware, no internet link, and no vendor software while still allowing wireless rewrites. Each commercial alternative relies on at least one proprietary component that fails the standalone and open infrastructure goals.

10.2.2 Comparison in terms of available features

The test prototype is built on the popular ESP32 platform, which has a powerful processor and the ability to communicate via Wi-Fi and Bluetooth. This makes it possible to perform more complex operations directly on the device and add additional functions over time. The device is designed from the beginning to be open, unlike commercially available products. The code is open-source.

The analysed products typically have their own application with limited functionality. And within a few years, due to phone operating system updates, the application may stop working. Choosing to control via a web server provides a more permanent solution that can also be used on laptops or other platforms. Another problem is the language barrier. Even though the applications are in some cases translated into English, the translation is not perfect.

The prototype's self hosted architecture leaves space for additional functions. Potential improvements include a browser based layout editor that lets event staff design custom screen templates, a timer that cycles through multiple pages on a fixed schedule, and buttons for on badge voting and feedback collection. Because all logic runs locally on the badge's microcontroller and the configuration interface is already web

based, each new feature can be deployed without altering the hardware or introducing any external services.

Chapter 11

Conclusion

The aim of the diploma thesis was to conduct a market analysis of smart e-paper screens and use the knowledge gained to design own prototype. It was required to create a prototype of the control gateway and the e-paper screen itself. The aim was also to select suitable protocols and technologies for the operation of the entire system. Furthermore, firmware for the control gateway and the e-paper screen was to be developed. Finally, the aim was to test the prototype and compare it with commercial products.

First, the basic principle of electronic paper technology was described. In the market analysis, a total of 5 devices were examined, among which devices were focused on both meeting rooms and name tags at conferences. Individual parameters were compared and it turned out that the common denominator of commercial devices is the closure of their systems and dependence on proprietary applications.

In the prototype design, an affordable and user popular ESP32 microcontroller was selected, which has Wi-Fi connectivity and sufficient performance. Furthermore, a suitable e-paper panel was selected. A suitable data processing and transmission process was presented. The control gateway functions as a Wi-Fi access point and runs a web server that is used to control e-paper screens. They can be added and removed using a web page. Data transfer to the screens is via the ESP-NOW protocol. A 3D case was designed for the test prototype. The cost of parts needed to build the prototype was less than $70.

The prototype was tested in continuous operation in various scenarios and at various distances. Individual functions were used and their correct functionality was verified. Compared to commercial devices, the prototype has several advantages, including openness due to the use of a universal web server, the relatively low cost of the prototype, the powerful ESP32 chip, and the ability to overwrite screens from various devices that support the Wi-Fi protocol.

The use of a powerful ESP32 microcontroller allows for future enhancements to the prototype with additional functions and more complex rendering. Future extensions could include dynamic changes to the screen layout using a web server, periodic changes to the content of the screens, or a voting button.

Bibliography

[1] Alec KleinStaff. „A New Printing Technology Sets Off a High-Stakes Race“. In: *The Wall Street Journal* (Jan. 2000). Accessed: 2025-04-16. URL: https://www.wsj.com/articles/SB946939872703897050.

[2] FREEscanRIP. *Electronic paper (Side view of Electrophoretic display)*. Accessed: 2025-04-18. Feb. 2014. URL: https://upload.wikimedia.org/wikipedia/commons/3/3a/Electronic_paper_%28Side_view_of_Electrophoretic_display%29_in_svg.svg.

[3] Amazon. *Kindle family*. Accessed: 2025-05-20. URL: https://www.amazon.com/Amazon-Kindle-Ereader-Family/b?ie=UTF8&node=6669702011.

[4] reMarkable. *reMarkable digital notebook*. Accessed: 2025-05-20. URL: https://remarkable.com/.

[5] E INK HOLDINGS INC. *Applications*. Apr. 2025. URL: https://www.eink.com/application.

[6] Reviver INC. *Digital License Plate*. Apr. 2025. URL: https://reviver.com/rplate/.

[7] infsoft GmbH. *E-Ink Display Beacon Collage*. Apr. 2025. URL: https://www.infsoft.com/wp-content/uploads/infsoft-infrastructure-hardware-e-ink-collage.jpg.

[8] LTD. DALIAN GOOD DISPLAY CO. *GoodDisplay ET0750-44B Image*. Apr. 2025. URL: https://img201.yun300.cn/repository/image/ba50a3ec-27d2-4fc2-8335-36ddc15457fe.jpg_640xaf.jpg?tenantId=160096&viewType=1&k=1734319076000.

[9] quality cases s.r.o. *TapirX 7.5 Image*. Apr. 2025. URL: https://www.tapirx.com/static/img/ec/products/1/0/1000/1.jpg.

[10] Ltd. Shenzhen TAIDEN Industrial Co. *Taiden HCS-1085 Image*. Apr. 2025. URL: https://www.taiden.com/upload/goodsgallery/2024-07/6690dadf0a3fc.png.

[11] Ltd. Guangzhou DSPPA Audio Co. *DSPPA D7642 Image*. Apr. 2025. URL: https://www.dsppacs.com/uploads/image/20221214/11/paperless-board_1670990283.jpg.

[12] STMicroelectronics. *STM32 wireless MCUs*. Accessed: 2025-04-25. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32-wireless-mcus.html.

[13] Microchip. *PIC32MZ W1 and WFI32 Family*. Accessed: 2025-04-25. URL: https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors/32-bit-mcus/pic32-32-bit-mcus/pic32mz-w1.

[14] Nordic Semiconductor ASA. *nRF52 Series*. Dec. 2024. URL: https://docs.nordicsemi.com/category/nrf-52-series.

[15] Texas Instruments Incorporated. *Low-power 2.4-GHz products*. Dec. 2024. URL: https://www.ti.com/wireless-connectivity/low-power-2-4-ghz/products.html.

[16] Espressif. *ESP32-WROOM-32E datasheet*. Accessed: 2025-04-25. URL: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf.

[17] Espressif. *ESP32 datasheet*. Accessed: 2025-04-25. URL: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[18] LaskaKit Github repository. *ESPink development board*. Accessed: 2025-04-25. URL: https://github.com/LaskaKit/ESPink.

[19] LTD. DALIAN GOOD DISPLAY CO. *E-ink screen 7.5 inch electronic paper display*. Accessed: 2025-04-25. URL: https://www.good-display.com/product/396.html.

[20] Mozilla foundation. *Protocol stack image*. Accessed: 2025-04-29. URL: https://mdn.github.io/shared-assets/images/diagrams/http/overview/http-layers.svg.

[21] Cloudflare. *What is HTTP/3*. Accessed: 2025-04-29. URL: https://www.cloudflare.com/learning/performance/what-is-http3/.

[22] Espressif. *OSI Model vs ESP-NOW model*. Accessed: 2025-04-29. URL: https://www.espressif.com/sites/all/themes/espressif/images/esp-now/model-en.png.

[23] Espressif. *ESP-NOW API Reference*. Accessed: 2025-04-29. URL: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/esp_now.html.

[24] Espressif. *Official IoT Development Framework*. Accessed: 2025-05-08. URL: https://www.espressif.com/en/products/sdks/esp-idf.

[25] martinberlin. *Epaper ESP-IDF component with GFX capabilities and multi SPI support*. Accessed: 2025-05-08. URL: https://github.com/martinberlin/CalEPD.

[26] Espressif. *Connectionless Modules Power-saving*. Accessed: 2025-05-08. URL: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-guides/wifi.html#connectionless-module-power-save.

[27] Chris Greening. *ESP-Now Range Test*. Accessed: 2025-05-08. URL: https://youtu.be/oz0a7Ur7nko?si=NiFAsEGsvR17VabI.

Appendix A

Acronym List

ADC	Analog to Digital Converter
AP	Access Point
API	Application Programming Interface
BLE	Bluetooth Low Energy
CAD	Computer-Aided Design
CNC	Computerized Numerical Control
CSS	Cascading Style Sheets
DAC	Digital to Analog Converter
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DMA	Direct Memory Access
EPD	Electrophoretic Display
FFC	Flexible Flat Cable
GPIO	General Purpose Input/Output
GATT	Generic Attribute Profile
HTTP	Hypertext Transfer Protocol
HTML	Hypertext Markup Language
IC	Integrated Circuit
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation

JTAG Joint Test Action Group

LCD Liquid Crystal Display

LED Light Emitting Diode

LUT Look-Up Table

MAC Media Access Control

MCU Microcontroller Unit

MQTT Message Queuing Telemetry Transport

mDNS Multicast DNS

NFC Near Field Communication

NVS Non-Volatile Storage

OLED Organic Light Emitting Diode

PCB Printed Circuit Board

PWM Pulse Width Modulation

QUIC Quick UDP Internet Connections

RAM Random Access Memory

REST Representational State Transfer

SDK Software Development Kit

SAR Successive Approximation Register

SoC System on Chip

SPI Serial Peripheral Interface

STL Standard Triangle Language

TCP Transmission Control Protocol

TLS Transport Layer Security

UDP User Datagram Protocol

UART Universal Asynchronous Receiver–Transmitter

USB Universal Serial Bus

VAT Value Added Tax