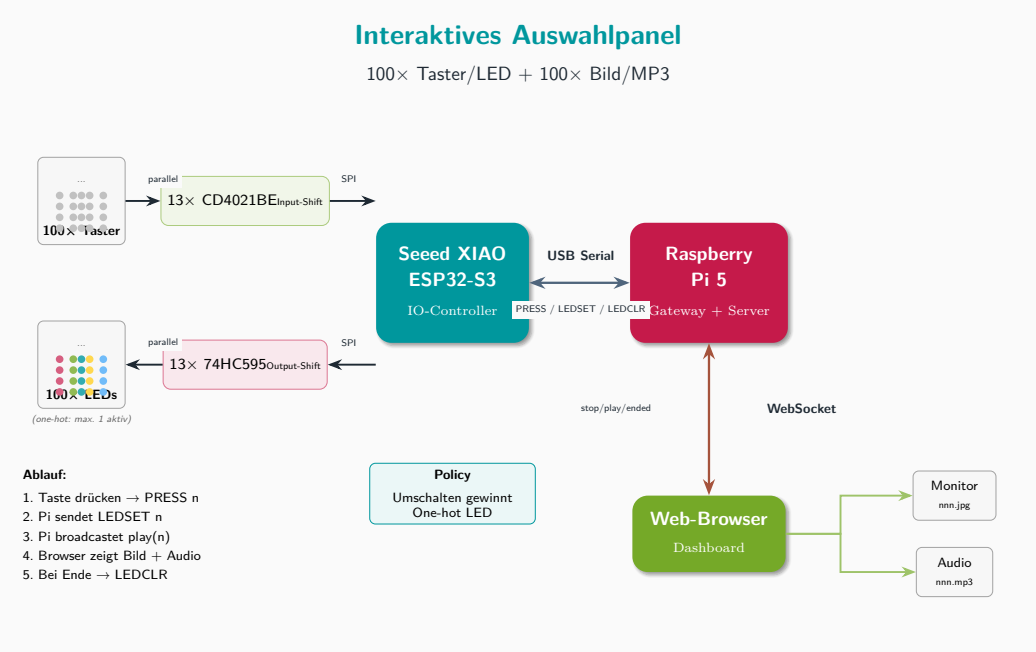
Interaktives Auswahlpanel

100× Taster/LED + 100× Bild/MP3

„Drücken – Leuchten – Abspielen"

ESP32S3 • Raspberry Pi 5 • Web-Dashboard



Technologie-Stack

CD4021BE • 74HC595 • PlatformIO • Python/aiohttp • WebSocket • HTML/JS • systemd

Stand: 27. Dezember 2025
Auswahlpanel-Projekt

Jan Unger
Version: 2.2.5

Inhaltsverzeichnis

1 Spezifikation **1**
- 1.1 Glossar . . . 1
- 1.2 Policy . . . 1
- 1.3 Pinbelegung ESP32-S3 XIAO . . . 2
- 1.4 CD4021BE vs. 74HC165 . . . 2
- 1.5 Verdrahtungsregeln . . . 3
- 1.6 Serial-Protokoll (ESP32 ↔ Pi) . . . 3
- 1.7 WebSocket-Protokoll (Pi ↔ Browser) . . . 4
- 1.8 HTTP-Endpoints . . . 4
- 1.9 Medien-Konvention . . . 4
- 1.10 Akzeptanztests . . . 5

2 Hardware **5**
- 2.1 Stueckliste (Vollausbau 100×) . . . 5
- 2.2 Prototyp-Stueckliste (10×) . . . 7
- 2.3 Kaskadierung . . . 7
- 2.4 IC-Pinbelegung . . . 8
- 2.5 Wichtige Regeln . . . 10
- 2.6 Stromversorgung . . . 10

3 Firmware **10**
- 3.1 Ueberblick . . . 11
- 3.2 Projektstruktur . . . 11
- 3.3 Konfiguration . . . 11
- 3.4 Pinbelegung . . . 12
- 3.5 Schieberegister-Treiber . . . 13
- 3.6 Build und Upload . . . 13
- 3.7 Debugging . . . 14
- 3.8 Features . . . 14

4 Server **14**
- 4.1 Ueberblick . . . 15
- 4.2 Verzeichnisstruktur . . . 15
- 4.3 Installation . . . 15
- 4.4 Konfiguration . . . 16
- 4.5 Autostart mit systemd . . . 16
- 4.6 HTTP-Endpoints . . . 18

4.7 Datenfluss bei Tastendruck . . . 18
4.8 Deployment (Mac → Pi) . . . 18
4.9 Features . . . 19

5 Dashboard **19**
5.1 Ueberblick . . . 19
5.2 Dateistruktur . . . 20
5.3 Farbschema . . . 20
5.4 WebSocket-Protokoll . . . 20
5.5 Audio-Unlock . . . 21
5.6 Tastenkuerzel . . . 21
5.7 Status-Indikatoren . . . 22
5.8 Debug-Panel . . . 22
5.9 Responsive Design . . . 22
5.10 Accessibility . . . 23
5.11 Features . . . 23

6 Befehlsreferenz **23**
6.1 Deployment (Mac → Pi) . . . 23
6.2 Server-Steuerung . . . 24
6.3 Dashboard testen . . . 25
6.4 Firmware flashen . . . 25
6.5 Serial-Debugging . . . 25
6.6 Medien verwalten . . . 26
6.7 Quick Reference . . . 26

7 Betriebshandbuch **27**
7.1 Schnellstart . . . 27
7.2 Hardware-Debugging . . . 27
7.3 Browser-Debugging . . . 28
7.4 Server-Debugging . . . 29
7.5 End-to-End Tests . . . 29
7.6 Deployment-Checkliste . . . 30
7.7 Nuetzliche Einzeiler . . . 30

8 Implementierungsplan **31**
8.1 Uebersicht . . . 31
8.2 Phase 1–5: Abgeschlossen . . . 31
8.3 Phase 6: Integration (aktiv) . . . 32

8.4 Phase 7: Produktivbetrieb 33
8.5 Zeitschaetzung 33
8.6 Naechste Schritte 34

9 Aenderungshistorie **34**

1 Spezifikation

Dieses Kapitel definiert die verbindlichen Schnittstellen und Regeln des Auswahlpanels. Es dient als *Single Source of Truth* – alle anderen Dokumente referenzieren hierher.

1.1 Glossar

Tabelle 1: Fachbegriffe und ihre Bedeutung

Begriff	Erklaerung
One-hot	Genau ein Bit ist aktiv, alle anderen sind aus
Preempt	Neue Aktion unterbricht sofort die laufende
Race-Condition	Timing-Problem, wenn zwei Ereignisse fast gleichzeitig auftreten
FreeRTOS	Echtzeit-Betriebssystem fuer Mikrocontroller
Mutex	Sperre, die gleichzeitigen Zugriff auf Ressourcen verhindert
CMOS	Chip-Technologie – Eingaenge nie unbeschaltet lassen
Pull-Up	Widerstand zieht Signal auf HIGH, Taster zieht auf LOW
Kaskadierung	ICs in Reihe schalten, um mehr Ein-/Ausgaenge zu erhalten
DIP-16	IC-Gehaeuse mit 16 Pins in zwei Reihen

1.2 Policy

1.2.1 One-hot LED

Zu jedem Zeitpunkt leuchtet **maximal eine LED**. Wenn wir uns die Strombilanz ansehen, wird der Vorteil klar: Maximalstrom betraegt $1 \times 20\,\text{mA}$ statt $100 \times 20\,\text{mA} = 2\,\text{A}$.

Tabelle 2: LED-Steuerbefehle

Befehl	Wirkung
`LEDSET n`	LED n an, alle anderen aus
`LEDCLR`	Alle LEDs aus

1.2.2 Preempt (Umschalten gewinnt)

Jeder neue Tastendruck unterbricht sofort die aktuelle Wiedergabe. Die Sequenz laeuft wie folgt ab:

1. Pi $\rightarrow$ ESP32: `LEDSET n`

2. Pi → Browser: `{"type":"stop"}`
3. Pi → Browser: `{"type":"play","id":n}`

Nach Audio-Ende meldet der Browser `{"type":"ended","id":n}`. Der Pi sendet `LEDCLR` **nur wenn** id = current_id – das schuetzt vor Race-Conditions.

Race-Condition-Schutz

Drueckt der Benutzer kurz vor Audio-Ende einen neuen Taster, darf das Ende des alten Audios die neue LED nicht ausschalten. Deshalb vergleichen wir immer die ID.

1.3 Pinbelegung ESP32-S3 XIAO

Tabelle 3: GPIO-Zuordnung des ESP32-S3 XIAO

Signal	Pin	Ziel-IC	Funktion
LED Data	`D0`	74HC595 SER	Serielle Daten fuer LEDs
LED Clock	`D1`	74HC595 SRCLK	Takt fuer Schieberegister
LED Latch	`D2`	74HC595 RCLK	Ausgabe freigeben
Btn Data	`D3`	CD4021 Q8	Serielle Daten von Tastern
Btn Clock	`D4`	CD4021 CLK	Takt zum Auslesen
Btn Load	`D5`	CD4021 P/S	Taster-Zustaende einlesen

1.4 CD4021BE vs. 74HC165

Wir verwenden den CD4021BE statt des 74HC165. Doch Vorsicht: Die Load-Logik ist invertiert!

Tabelle 4: Vergleich der Eingangs-Schieberegister

Aspekt	74HC165	CD4021BE
Load-Signal	LOW	**HIGH**
Shift-Signal	HIGH	LOW
DIP-Verfuegbarkeit	Schwer	Gut

Firmware-Anpassung

Die invertierte Load-Logik ist in der Firmware beruecksichtigt. Beim Wechsel von 74HC165 zu CD4021BE muss `P/S = HIGH` zum Laden gesetzt werden.

1.5 Verdrahtungsregeln

Tabelle 5: Kritische Verdrahtungsregeln

IC	Pin	Regel	Grund
CD4021 (letzter)	Pin 11 (SER)	→ VCC	CMOS-Eingaenge nie floaten
74HC595 (letzter)	Pin 9 (QH')	offen OK	Ausgang treibt aktiv
74HC595 (alle)	Pin 10 (SRCLR)	→ VCC	Clear deaktiviert
74HC595 (alle)	Pin 13 (OE)	→ GND	Outputs aktiviert

1.6 Serial-Protokoll (ESP32 ↔ Pi)

Die Kommunikation erfolgt mit 115 200 baud, ASCII-kodiert und Newline-terminiert.

1.6.1 ESP32 → Pi

Tabelle 6: Nachrichten vom ESP32 zum Raspberry Pi

Nachricht	Bedeutung
`READY`	Controller bereit
`PRESS nnn`	Taster gedrueckt (000–099)
`RELEASE nnn`	Taster losgelassen
`OK`	Befehl erfolgreich
`PONG`	Antwort auf PING

1.6.2 Pi → ESP32

Tabelle 7: Befehle vom Raspberry Pi zum ESP32

Befehl	Funktion
`LEDSET nnn`	LED ein (one-hot)
`LEDON/LEDOFF nnn`	LED additiv ein/aus
`LEDCLR/LEDALL`	Alle aus/ein
`PING/STATUS`	Test/Zustand
`TEST/STOP`	LED-Lauflicht
`VERSION/QRESET`	Info/Reset

1.7 WebSocket-Protokoll (Pi ↔ Browser)

Endpoint: `ws://<pi>:8080/ws`

Tabelle 8: WebSocket-Nachrichten

Richtung	Nachricht
Pi → Browser	`{"type":"stop"}`
Pi → Browser	`{"type":"play","id":42}`
Browser → Pi	`{"type":"ended","id":42}`

1.8 HTTP-Endpoints

Tabelle 9: REST-Schnittstellen des Servers

Pfad	Funktion
`/`	Dashboard
`/ws`	WebSocket
`/media/*`	Bilder/Audio
`/test/play/{id}`	Wiedergabe testen
`/status`	Server-Status (JSON)
`/health`	Health-Check

1.9 Medien-Konvention

Jede ID (0–99) korrespondiert mit einem Bild und einer Audio-Datei:

Tabelle 10: Dateinamenskonvention

ID	Bild	Audio
42	`media/042.jpg`	`media/042.mp3`

IDs sind intern 0–99, im Protokoll jedoch zero-padded (000–099).

1.10 Akzeptanztests

Tabelle 11: Testfaelle fuer die Abnahme

Test	Erwartung
Preempt	Neuer Taster unterbricht sofort
One-hot	Nur eine LED leuchtet
Ende	Nach Audio: alle LEDs aus
Race	LED bleibt an wenn neue ID aktiv
Debounce	Nur ein Event pro Tastendruck

2 Hardware

Dieses Kapitel beschreibt die Komponenten und deren Verdrahtung. Fuer Pinbelegung und Policy verweisen wir auf section 1.

2.1 Stueckliste (Vollausbau $100\times$)

2.1.1 Recheneinheit

Tabelle 12: Recheneinheit und Peripherie

Anzahl	Komponente	Anmerkung
$1\times$	Raspberry Pi 5	Gateway + Media-Server
$1\times$	Netzteil USB-C 5V/5A	
$1\times$	HDMI-Monitor	Dashboard
$1\times$	Aktivlautsprecher	Klinke/HDMI/USB

2.1.2 IO-Controller

Tabelle 13: Mikrocontroller

Anzahl	Komponente	Anmerkung
1×	Seeed XIAO ESP32-S3	Dual-Core, USB-C
1×	USB-C Kabel	XIAO ↔ Pi

2.1.3 Eingaenge (CD4021BE)

Tabelle 14: Taster-Eingangsstufe

Anzahl	Komponente	Anmerkung
100×	Taster 6×6 mm	Tact-Switch
13×	CD4021BE DIP-16	8 Inputs pro IC
100×	Widerstand 10 kΩ	Pull-Up
13×	Kondensator 100 nF	Abblockkondensator

2.1.4 Ausgaenge (74HC595)

Tabelle 15: LED-Ausgangsstufe

Anzahl	Komponente	Anmerkung
100×	LED 5 mm	Farbe nach Wahl
13×	74HC595 DIP-16	8 Outputs pro IC
100×	Widerstand 330 Ω	Vorwiderstand @ 5V
13×	Kondensator 100 nF	Abblockkondensator

2.1.5 Verdrahtung

Tabelle 16: Verdrahtungsmaterial

Anzahl	Komponente
1×	Lochrasterplatine 160×100 mm
26×	DIP-16 Sockel (optional)
1×	Dupont-Jumper Set
1×	Litze 0,14–0,25 mm^2

2.2 Prototyp-Stueckliste (10×)

Fuer erste Tests genuegt eine reduzierte Bestueckung:

Tabelle 17: Komponenten fuer den Prototyp

Anzahl	Komponente
1×	ESP32-S3 XIAO
2×	74HC595
2×	CD4021BE
10×	LED 5 mm
10×	Taster 6 × 6 mm
10×	Widerstand 330 Ω
10×	Widerstand 10 kΩ
4×	Kondensator 100 nF

2.3 Kaskadierung

Kaskadierung bedeutet: ICs in Reihe schalten. Der serielle Ausgang eines ICs geht in den seriellen Eingang des naechsten.

2.3.1 74HC595 (LED-Ausgabe)

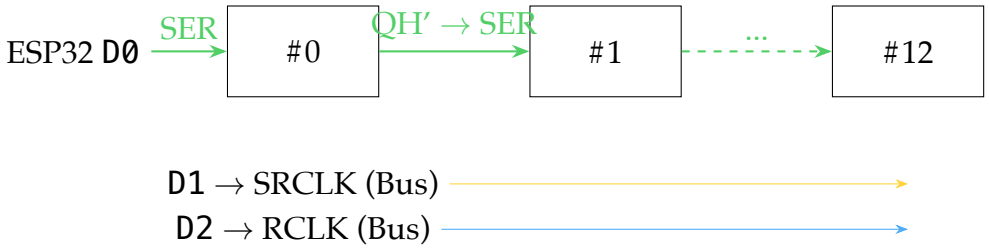


Abbildung 1: Kaskadierung der 74HC595 Ausgangsregister

2.3.2 CD4021BE (Taster-Eingabe)

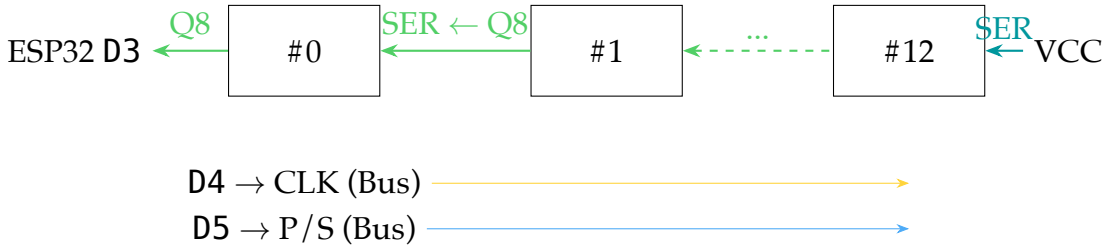


Abbildung 2: Kaskadierung der CD4021BE Eingangsregister

Wichtig: SER am letzten IC

Der SER-Eingang des letzten CD4021BE muss auf VCC gelegt werden. CMOS-Eingaenge duerfen nie floaten – sonst entstehen zufaellige Trigger.

2.4 IC-Pinbelegung

2.4.1 74HC595 (DIP-16)

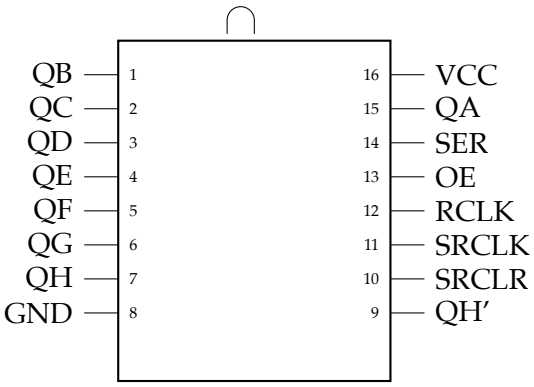


Abbildung 3: Pinbelegung 74HC595

Tabelle 18: Anschlussbelegung 74HC595

Pin	Verbindung
14 (SER)	← `D0` oder vorheriger QH′
11 (SRCLK)	← `D1` (Bus)
12 (RCLK)	← `D2` (Bus)
9 (QH′)	→ naechster SER oder offen
10 (SRCLR)	→ VCC
13 (OE)	→ GND

2.4.2 CD4021BE (DIP-16)

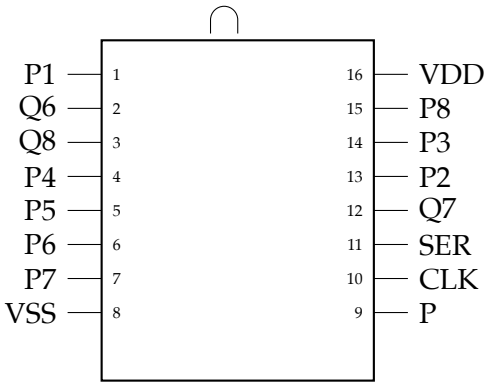


Abbildung 4: Pinbelegung CD4021BE

Tabelle 19: Anschlussbelegung CD4021BE

Pin	Verbindung
3 (Q8)	→ `D3` oder vorheriger SER
10 (CLK)	← `D4` (Bus)
9 (P/S)	← `D5` (Bus)
11 (SER)	← naechster Q8 oder → **VCC**

P/S = HIGH fuer Load

Der CD4021BE laedt bei **P/S = HIGH** – invertiert zum 74HC165!

2.5 Wichtige Regeln

Tabelle 20: Kritische Hardware-Regeln

Regel	Grund
CD4021 letzter IC: Pin 11 → VCC	CMOS-Eingaenge nie floaten
74HC595 letzter IC: Pin 9 offen OK	Ausgang treibt aktiv
100 nF an jedem VCC / VDD	Spannungsspitzen filtern
Pull-Up 10 kΩ an jedem Taster	Definierter Pegel wenn offen
Unbenutzte CD4021-Inputs → VCC	Keine zufaelligen Trigger

2.6 Stromversorgung

Tabelle 21: Stromaufnahme nach Betriebsart

Betriebsart	Stromaufnahme
One-hot (1 LED)	max. 20 mA
Alle LEDs (Test)	max. 2 A

Empfehlung

5V vom Pi-Netzteil abzweigen. Bei 3,3V-Betrieb direkt vom ESP32 3V3-Pin versorgen.

3 Firmware

Dieses Kapitel beschreibt die Dual-Core-Firmware fuer den IO-Controller. Fuer Pinbelegung und Protokoll verweisen wir auf section 1.

Tabelle 22: Firmware-Metadaten

Eigenschaft	Wert
Board	Seeed XIAO ESP32-S3
Framework	Arduino + FreeRTOS
Version	2.2.5

3.1 Ueberblick

Die Firmware nutzt **FreeRTOS** – ein Echtzeit-Betriebssystem, das mehrere Tasks parallel ausfuehrt. Der ESP32-S3 besitzt zwei CPU-Kerne: Core 0 scannt die Taster, Core 1 verarbeitet Befehle.

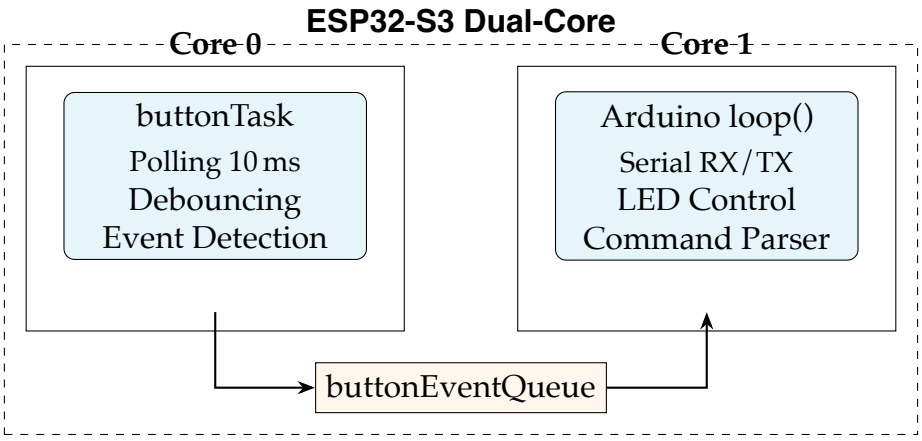


Abbildung 5: Dual-Core-Architektur der Firmware

Warum zwei Kerne?

Serial-Kommunikation kann blockieren. Laeuft das Button-Scanning auf einem separaten Kern, gehen keine Events verloren.

3.2 Projektstruktur

```
button_panel_firmware/
|-- platformio.ini          # Build-Konfiguration
|-- include/
|   |-- config.h            # Pins, Timing, Skalierung
|   +-- shift_register.h    # Hardware-Abstraktion
+-- src/
    |-- main.cpp            # Setup, Loop, Tasks
    +-- shift_register.cpp  # Schieberegister-Treiber
```

3.3 Konfiguration

Die Datei `config.h` definiert alle Hardware-Parameter:

```
// Skalierung: true = 10 Kanaele (Prototyp), false = 100 Kanaele
#define PROTOTYPE_MODE

#ifdef PROTOTYPE_MODE
    constexpr uint8_t NUM_SHIFT_REGS = 2;   // 2 ICs pro Kette
    constexpr uint8_t NUM_LEDS = 10;
    constexpr uint8_t NUM_BUTTONS = 10;
#else
    constexpr uint8_t NUM_SHIFT_REGS = 13;  // 13 ICs pro Kette
    constexpr uint8_t NUM_LEDS = 100;
    constexpr uint8_t NUM_BUTTONS = 100;
#endif

// Timing
constexpr uint32_t DEBOUNCE_TIME_MS = 50;   // Entprellzeit
constexpr uint32_t BUTTON_POLL_MS = 10;     // Abtastintervall
constexpr uint32_t WATCHDOG_TIMEOUT_MS = 5000;
```

Tabelle 23: Timing-Parameter erklaert

Parameter	Wert	Erklaerung
Debouncing	50 ms	Mechanische Taster prellen – der Kontakt schwingt kurz, bevor er stabil ist. Diese Wartezeit filtert Stoerimpulse.
Polling	10 ms	Abtastintervall fuer Tasterzustaende
Watchdog	5 s	Timer, der das System neu startet, wenn es haengt

3.4 Pinbelegung

Tabelle 24: GPIO-Zuordnung des ESP32-S3 XIAO

Signal	Pin	Ziel-IC	Funktion
LED Data	D0	74HC595 SER	Serielle Daten fuer LEDs
LED Clock	D1	74HC595 SRCLK	Takt fuer Datenuebernahme
LED Latch	D2	74HC595 RCLK	Ausgabe an LEDs freigeben
Btn Data	D3	CD4021 Q8	Serielle Daten von Tastern
Btn Clock	D4	CD4021 CLK	Takt zum Auslesen
Btn Load	D5	CD4021 P/S	Taster-Zustaende einlesen

Invertierte Load-Logik

Der CD4021BE verwendet `P/S = HIGH` fuer Parallel Load – invertiert zum bekannteren 74HC165.

3.5 Schieberegister-Treiber

3.5.1 LED-Ausgabe (74HC595)

```
class OutputShiftRegister {
public:
    void setOnly(int8_t index);  // One-hot: nur diese LED an
    void clear();                // Alle LEDs aus
    void update();               // Buffer an Hardware senden
};
```

One-hot bedeutet: Genau ein Bit ist aktiv, alle anderen sind aus. Bei `setOnly(5)` leuchtet nur LED 5.

3.5.2 Taster-Eingabe (CD4021)

```
class InputShiftRegister {
public:
    void update();                      // Taster einlesen
    bool getInput(uint8_t index) const; // Zustand abfragen
};
```

Beide Treiber schuetzen ihren Hardware-Zugriff mit einem **Mutex** – einer Sperre, die verhindert, dass zwei Tasks gleichzeitig auf dieselbe Ressource zugreifen.

3.6 Build und Upload

```
pio run                      # Kompilieren
pio run -t upload            # Flashen
pio device monitor           # Serial Monitor (115200 Baud)
pio run -t upload -t monitor # Flash + Monitor
```

3.7 Debugging

Tabelle 25: Haeufige Firmware-Fehler

Meldung	Ursache	Loesung
`ERROR LED init failed`	Mutex-Erstellung fehlgeschlagen	RAM pruefen
`ERROR Queue create failed`	Nicht genug Heap	Stack-Groessen reduzieren
Keine Button-Events	P/S-Logik falsch	HIGH fuer Load verwenden

3.8 Features

- Dual-Core: Button-Scan auf Core 0, Serial auf Core 1
- Mutex-geschuetzter Hardware-Zugriff
- Non-blocking LED-Test (State Machine)
- Watchdog mit 5 s Timeout
- `STATUS` liefert Heap, Queue, LED-Maske
- `PROTOTYPE_MODE` fuer Skalierung $10 \leftrightarrow 100$

4 Server

Dieses Kapitel beschreibt den Python-Server als Bruecke zwischen ESP32 und Browser. Fuer das Protokoll verweisen wir auf section 1.

Tabelle 26: Server-Metadaten

Eigenschaft	Wert
Plattform	Raspberry Pi 5, Pi OS Lite
Python	3.11+
Version	2.2.5

4.1 Ueberblick

Der Server empfaengt Tastendruck-Events vom ESP32 per USB-Serial und verteilt sie per **WebSocket** an alle Browser. WebSocket ist eine bidirektionale Verbindung – der Server kann jederzeit Nachrichten senden, ohne dass der Client fragt.

Abbildung 6: Kommunikationsarchitektur

Der Server nutzt **aiohttp** – eine Python-Bibliothek fuer asynchrone HTTP/WebSocket-Server. Asynchron bedeutet: Der Server wartet nicht blockierend auf Antworten, sondern bearbeitet mehrere Anfragen parallel.

4.2 Verzeichnisstruktur

```
/home/pi/selection-panel/
|-- venv/                    # Isolierte Python-Umgebung
|-- server.py                # Hauptserver
|-- selection-panel.service  # Autostart-Konfiguration
|-- static/                  # Dashboard-Dateien
|   |-- index.html
|   |-- styles.css
|   +-- app.js
+-- media/                   # Bild/Audio-Dateien
    |-- 000.jpg  000.mp3
    +-- ...
```

Virtual Environment (venv)

Eine isolierte Python-Umgebung. Pakete werden nur hier installiert, nicht systemweit – das verhindert Versionskonflikte.

4.3 Installation

4.3.1 Python-Umgebung

```
# Umgebung erstellen
```

```
python3 -m venv ~/selection-panel/venv

# Pakete installieren
~/selection-panel/venv/bin/pip install aiohttp pyserial-asyncio
```

pyserial-asyncio ist ein asynchroner Serial-Treiber – er liest vom ESP32, ohne den Server zu blockieren.

4.3.2 Serial-Port ermitteln

```
ls -l /dev/serial/by-id/
```

Den Pfad tragen wir in `server.py` ein:

```
SERIAL_PORT = "/dev/serial/by-id/usb-Espressif_..."
```

4.4 Konfiguration

```
# Skalierung: True = 10 Medien, False = 100 Medien
PROTOTYPE_MODE = True
NUM_MEDIA = 10 if PROTOTYPE_MODE else 100

# Serial
SERIAL_PORT = "/dev/serial/by-id/usb-Espressif_..."
SERIAL_BAUD = 115200

# HTTP
HTTP_HOST = "0.0.0.0"  # Alle Interfaces
HTTP_PORT = 8080
```

4.5 Autostart mit systemd

systemd ist der Service-Manager von Linux. Er startet den Server automatisch beim Hochfahren und startet ihn bei Absturz neu.

4.5.1 Service-Datei

Datei: `/etc/systemd/system/selection-panel.service`

```
[Unit]
Description=Interaktives Auswahlpanel
After=network.target

[Service]
Type=simple
User=pi
WorkingDirectory=/home/pi/selection-panel
ExecStart=/home/pi/selection-panel/venv/bin/python server.py
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target
```

4.5.2 Aktivieren

```
sudo systemctl daemon-reload
sudo systemctl enable --now selection-panel.service
```

4.5.3 Status pruefen

```
sudo systemctl status selection-panel.service
journalctl -u selection-panel.service -f  # Live-Logs
```

4.6 HTTP-Endpoints

Tabelle 27: REST-Schnittstellen des Servers

Pfad	Funktion
`/`	Dashboard (index.html)
`/ws`	WebSocket-Verbindung
`/media/*`	Bilder und Audio
`/test/play/{id}`	Wiedergabe simulieren
`/status`	Server-Status (JSON)
`/health`	Health-Check

4.7 Datenfluss bei Tastendruck

Wenn wir uns den Ablauf eines Tastendrucks ansehen, erkennen wir die Preempt-Policy in Aktion:

1. ESP32 sendet `PRESS 042`
2. Server sendet `LEDSET 042` zurueck
3. Server broadcastet `{"type":"stop"}` an alle Browser
4. Server broadcastet `{"type":"play","id":42}`
5. Browser meldet `{"type":"ended","id":42}` nach Audio-Ende
6. Server sendet `LEDCLR`

Race-Condition-Schutz

Wenn zwei Tasten schnell hintereinander kommen, koennte ein `ended`-Event zur falschen ID gehoeren. Der Server prueft deshalb: Ist die ID noch aktuell? Nur dann werden die LEDs geloescht.

4.8 Deployment (Mac → Pi)

```
rsync -avz --exclude='venv' --exclude='.git' \
    ./selection-panel/ pi@rover:/home/pi/selection-panel/

ssh pi@rover 'sudo systemctl restart selection-panel.service'
```

4.9 Features

- Asynchroner Serial-Reader
- WebSocket Broadcast an alle Clients
- Preempt-Policy mit Race-Condition-Schutz
- Medien-Validierung beim Start
- Auto-Reconnect bei Serial-Abbruch
- `PROTOTYPE_MODE` fuer Skalierung $10 \leftrightarrow 100$

5 Dashboard

Dieses Kapitel beschreibt das Web-Frontend fuer die Vollbild-Anzeige. Fuer das WebSocket-Protokoll verweisen wir auf section 1.

Tabelle 28: Dashboard-Metadaten

Eigenschaft	Wert
Framework	Vanilla JavaScript
Farbschema	Arduino Teal + Raspberry Pi Red
Version	2.2.5

5.1 Ueberblick

Das Dashboard ist eine **Single-Page-Application (SPA)** – eine Webanwendung, die nur einmal geladen wird. Alle Updates kommen per WebSocket, ohne Seitenneuladen.

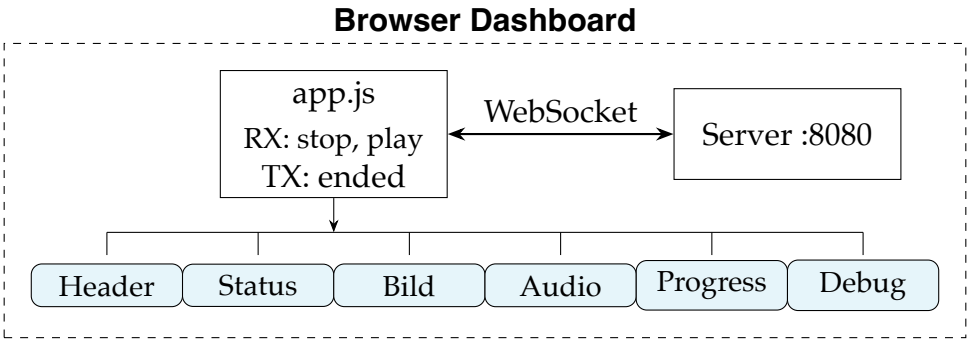


Abbildung 7: Dashboard-Architektur

5.2 Dateistruktur

```
static/
|-- index.html      # Hauptseite
|-- styles.css      # Styling mit Design Tokens
+-- app.js          # WebSocket-Client
```

Design Tokens

Zentral definierte Farbwerte als CSS-Variablen. Aendert man `--arduino-teal`, aendert sich die Farbe ueberall.

5.3 Farbschema

Tabelle 29: Farbpalette des Dashboards

Farbe	Hex	Verwendung
Arduino Teal	`#00979D`	Titel, Links, Akzente
Raspberry Red	`#C51A4A`	Unlock-Button, Warnungen
Success Green	`#75A928`	Status: Verbunden
GitHub Dark	`#0D1117`	Hintergrund

5.4 WebSocket-Protokoll

5.4.1 Empfangen (Server → Browser)

Tabelle 30: Eingehende WebSocket-Nachrichten

Nachricht	Aktion
`{"type":"stop"}`	Audio stoppen, Bild ausblenden
`{"type":"play","id":42}`	Bild 042.jpg laden, Audio 042.mp3 abspielen

5.4.2 Senden (Browser → Server)

Tabelle 31: Ausgehende WebSocket-Nachrichten

Nachricht	Ausloeser
`{"type":"ended","id":42}`	Audio fertig abgespielt

5.4.3 Auto-Reconnect

```
ws.onclose = () => {
    setTimeout(connectWebSocket, 5000);  // Nach 5 s neu verbinden
};
```

5.5 Audio-Unlock

Browser blockieren automatische Audio-Wiedergabe – die sogenannte **Autoplay Policy**. Erst nach einer Nutzer-Interaktion darf Audio starten.

Unsere Loesung: Der Unlock-Button spielt ein stilles Audio ab. Danach ist Audio entsperrt.

```
function unlockAudio() {
    audio.src = 'data:audio/wav;base64,...';  // Stilles WAV
    audio.play().then(() => {
        state.audioUnlocked = true;
        unlockButton.style.display = 'none';
    });
}
```

5.6 Tastenkuerzel

Tabelle 32: Keyboard-Shortcuts

Taste	Funktion
`Space`	Play/Pause
`Ctrl+D`	Debug-Panel ein/aus

5.7 Status-Indikatoren

Zwei Punkte oben rechts zeigen den Verbindungsstatus:

Tabelle 33: Status-Indikatoren im Dashboard

Indikator	Gruen	Rot
WebSocket	Verbunden	Getrennt
Audio	Entsperrt	Gesperrt

5.8 Debug-Panel

Das Debug-Panel (`Ctrl+D`) zeigt alle Events:

```
[14:32:05] WS: {"type":"play","id":5}
[14:32:05] Audio gestartet: 005.mp3
[14:32:12] Audio beendet
[14:32:12] TX: {"type":"ended","id":5}
```

5.9 Responsive Design

Tabelle 34: Breakpoints fuer verschiedene Bildschirmgroessen

Breakpoint	Ziel
< 768 px	Mobile
768–1199 px	Tablet
1200–1919 px	Desktop
≥ 1920 px	Full HD / 4K

5.10 Accessibility

Tabelle 35: Barrierefreiheits-Features

Feature	CSS-Query
Reduzierte Animationen	`prefers-reduced-motion`
Hoher Kontrast	`prefers-contrast: high`
Light Mode	`prefers-color-scheme: light`
iPhone Notch	`env(safe-area-inset-*)`

5.11 Features

- WebSocket mit Auto-Reconnect
- Audio-Unlock fuer Autoplay-Policy
- Fortschrittsanzeige mit Zeitanzeige
- Status-Indikatoren (WebSocket, Audio)
- Debug-Panel mit Log-History
- Responsive Design (Mobile → 4K)
- Dark/Light Theme Support
- Keyboard-Shortcuts

6 Befehlsreferenz

Dieses Kapitel fasst alle Deployment-, Build- und Test-Befehle zusammen. Fuer das Protokoll verweisen wir auf section 1.

6.1 Deployment (Mac → Pi)

rsync synchronisiert Dateien zwischen Rechnern – nur geaenderte Dateien werden uebertragen.

```
cd ./selection-panel

rsync -avz --delete \
```

```
    --exclude='button_panel_firmware' \
    --exclude='venv' \
    --exclude='.git' \
    --exclude='__pycache__' \
    . pi@rover:/home/pi/selection-panel/
```

Tabelle 36: rsync-Optionen

Flag	Bedeutung
-a	Archiv-Modus (Rechte, Zeiten erhalten)
-v	Verbose (zeigt Dateien)
-z	Komprimiert Uebertragung
--delete	Loescht Dateien auf Ziel, die lokal nicht existieren

6.2 Server-Steuerung

systemd ist der Service-Manager von Linux. Er startet Dienste automatisch und ueberwacht sie.

```
# Starten + Logs verfolgen
ssh pi@rover 'sudo systemctl restart selection-panel && \
              journalctl -u selection-panel -f'

# Nur starten/stoppen
ssh pi@rover 'sudo systemctl start selection-panel'
ssh pi@rover 'sudo systemctl stop selection-panel'

# Status pruefen
ssh pi@rover 'sudo systemctl status selection-panel'
```

journalctl zeigt systemd-Logs:

```
# Live-Logs
journalctl -u selection-panel -f

# Letzte 50 Zeilen
journalctl -u selection-panel -n 50
```

```
# Letzte Stunde
journalctl -u selection-panel --since "1 hour ago"
```

6.3 Dashboard testen

```
# Browser oeffnen (Mac)
open http://rover:8080/

# Status (JSON)
curl http://rover:8080/status

# Health-Check
curl http://rover:8080/health

# Wiedergabe simulieren
curl http://rover:8080/test/play/5
curl http://rover:8080/test/stop
```

6.4 Firmware flashen

PlatformIO ist ein Build-System fuer Embedded-Entwicklung.

```
cd button_panel_firmware

pio run                      # Kompilieren
pio run -t upload            # Flashen
pio device monitor           # Serial-Monitor (115200 Baud)
pio run -t upload -t monitor # Flash + Monitor
```

6.5 Serial-Debugging

screen ist ein Terminal-Multiplexer – er verbindet sich mit seriellen Geraeten.

```
# Server stoppen (gibt Port frei)
ssh pi@rover 'sudo systemctl stop selection-panel'
```

```
# Serial-Port finden
ssh pi@rover 'ls -l /dev/serial/by-id/'

# Verbinden
ssh pi@rover
screen /dev/serial/by-id/usb-Espressif_... 115200
```

Screen beenden

Tastenkombination: `Ctrl+A`, dann `K`, dann `Y`

Fuer verfuegbare Befehle siehe section 1.6.

6.6 Medien verwalten

```
# Test-Medien generieren
./generate_test_media.sh      # 10 Stueck (Prototyp)
./generate_test_media.sh 100  # 100 Stueck (Produktion)

# Medien auf Pi pruefen
ssh pi@rover 'ls ~/selection-panel/media/*.jpg | wc -l'
ssh pi@rover 'ls ~/selection-panel/media/*.mp3 | wc -l'
```

6.7 Quick Reference

Tabelle 37: Haeufig verwendete Befehle

Aktion	Befehl
Deploy	`rsync -avz --delete --exclude=... . pi@rover:...`
Server starten	`ssh pi@rover 'sudo systemctl restart selection-panel'`
Dashboard	`open http://rover:8080/`
Test-Play	`curl http://rover:8080/test/play/5`
Firmware	`cd button_panel_firmware && pio run -t upload`
Serial-Test	`screen /dev/serial/by-id/usb-Espressif* 115200`

7 Betriebshandbuch

Dieses Kapitel enthaelt Debugging-Checklisten und Testprozeduren. Fuer die Befehlsreferenz verweisen wir auf section 6.

7.1 Schnellstart

```
open http://rover:8080/              # Dashboard
curl http://rover:8080/status        # Status
curl http://rover:8080/test/play/5   # Test
```

7.2 Hardware-Debugging

7.2.1 CD4021BE (Buttons)

Tabelle 38: Fehlersuche CD4021BE

Symptom	Ursache	Loesung
Keine Events	P/S-Logik invertiert	HIGH = Load, LOW = Shift
Falsche Bits	Kaskadierung falsch	Q8 → SER pruefen
Instabil	Fehlende Kondensatoren	100 nF an VDD/VSS
Zufaellige Trigger	SER floatet (letzter IC)	Pin 11 → VCC

7.2.2 74HC595 (LEDs)

Tabelle 39: Fehlersuche 74HC595

Symptom	Ursache	Loesung
Keine LEDs	OE nicht auf GND	Pin 13 → GND
Alle LEDs an	SRCLR auf LOW	Pin 10 → VCC
Falsche LED	Kaskadierung	QH' → SER pruefen

7.2.3 ESP32

Tabelle 40: Fehlersuche ESP32

Symptom	Ursache	Loesung
Keine Antwort	USB-Port belegt	Server stoppen
Kein Upload	Falscher Port	`ls /dev/serial/by-id/`
Reboot-Schleife	Watchdog	Firmware pruefen

7.3 Browser-Debugging

7.3.1 Kein Ton

Wenn wir keinen Ton hoeren, pruefen wir folgende Punkte:

1. „Sound aktivieren“-Button klicken (Autoplay-Sperre des Browsers)
2. Audio-Status muss **gruen** werden
3. DevTools → Console auf Fehler pruefen

7.3.2 WebSocket-Verbindung

Die Verbindung laesst sich in den DevTools pruefen: Network → WS → Messages.

Tabelle 41: Status-Indikatoren im Dashboard

Status-Punkt	Gruen	Rot
WebSocket	Verbunden	Getrennt
Audio	Entsperrt	Gesperrt

7.3.3 Tastenkuerzel

Tabelle 42: Dashboard-Shortcuts

Taste	Funktion
`Space`	Play / Pause
`Ctrl+D`	Debug-Panel

7.4 Server-Debugging

```
# Logs live
journalctl -u selection-panel -f

# Letzte 10 Minuten
journalctl -u selection-panel --since "10 min ago"

# Serial-Port pruefen
ssh pi@rover 'ls -l /dev/serial/by-id/'

# Benutzerrechte
ssh pi@rover 'groups pi'  # dialout muss enthalten sein
```

7.5 End-to-End Tests

7.5.1 Preempt-Test

1. Button 5 druecken → LED 5 an, Audio laeuft
2. Nach 1 s: Button 3 druecken
3. **Erwartet:** Audio 5 stoppt sofort, LED 3 an, Audio 3 laeuft

7.5.2 One-hot Test

1. `curl .../test/play/5`
2. `curl .../test/play/3`
3. **Erwartet:** Nur LED 3 leuchtet

7.5.3 Ende-Test

1. Audio abspielen lassen
2. Warten bis Ende
3. **Erwartet:** Alle LEDs aus

7.5.4 Race-Condition Test

Dieser Test prueft den Schutz vor Timing-Problemen:

1. Button 3 druecken, Audio laeuft
2. Kurz vor Ende: Button 5 druecken
3. Audio 3 endet
4. **Erwartet:** LED 5 bleibt an (nicht aus!)

Kritischer Test

Wenn LED 5 nach Ende von Audio 3 ausgeht, liegt ein Race-Condition-Bug vor. Der Server muss die ID vergleichen, bevor er `LEDCLR` sendet.

7.5.5 Stresstest

```
for i in {1..200}; do
    curl -s "http://rover:8080/test/play/$((RANDOM % 10))"
    sleep 0.2
done
```

7.6 Deployment-Checkliste

- ☐ Firmware geflasht
- ☐ `server.py` auf Pi kopiert
- ☐ venv mit Paketen erstellt
- ☐ Serial-Port eingetragen
- ☐ systemd-Service aktiviert
- ☐ Medien vorhanden (000–099)
- ☐ Reboot-Test bestanden
- ☐ Preempt-Test bestanden

7.7 Nuetzliche Einzeiler

```
# Medien zaehlen
ls ~/selection-panel/media/*.jpg | wc -l

# Service-Status
systemctl is-active selection-panel

# Firewall oeffnen
sudo ufw allow 8080/tcp

# USB-Regeln neu laden
sudo udevadm control --reload-rules && sudo udevadm trigger
```

8 Implementierungsplan

Dieses Kapitel gibt einen Ueberblick ueber die Projektphasen und deren Status. Fuer technische Details verweisen wir auf section 1, fuer Tests auf section 7.

8.1 Uebersicht

Tabelle 43: Phasenuebersicht des Auswahlpanel-Projekts

Phase	Bezeichnung	Status
1	Infrastruktur	**abgeschlossen**
2	Hardware-Prototyp	**abgeschlossen**
3	ESP32 Firmware	**abgeschlossen**
4	Raspberry Pi Server	**abgeschlossen**
5	Web-Dashboard	**abgeschlossen**
6	Integration & Test	**aktiv**
7	Produktivbetrieb	ausstehend

8.2 Phase 1–5: Abgeschlossen

Die ersten fuenf Phasen haben wir erfolgreich abgeschlossen. Jede Phase endete mit einem klar definierten Meilenstein:

Tabelle 44: Erreichte Meilensteine

Phase	Meilenstein	Beschreibung	Status
1	M1	Toolchain funktioniert	✓
2	M2	Taster → LED reagiert	✓
3	M3	`PRESS` → `LEDSET` funktioniert	✓
4	M4	WebSocket-Verbindung steht	✓
5	M5	Browser zeigt Bild + Audio	✓

8.3 Phase 6: Integration (aktiv)

In Phase 6 skalieren wir das System vom Prototyp ($10\times$) auf die volle Ausbaustufe ($100\times$).

8.3.1 Hardware skalieren

Tabelle 45: Hardware-Aufgaben in Phase 6

Aufgabe	Status
$13\times$ CD4021BE kaskadieren (Eingaenge)	○
$13\times$ 74HC595 kaskadieren (Ausgaenge)	○
Durchgangspruefung aller Verbindungen	○

8.3.2 Medien

Fuer den Vollausbau benoetigen wir 100 Medien-Sets, jeweils bestehend aus einem Bild und einer Audio-Datei:

Tabelle 46: Medien-Status

Aufgabe	Status
Test-Medien (000–009)	✓
Produktiv-Medien (000–099)	○

8.3.3 Akzeptanztests

Bevor wir Phase 6 abschliessen, muessen alle Akzeptanztests bestanden sein:

Tabelle 47: Akzeptanztests fuer Meilenstein M6

Test	Status
Preempt-Verhalten (via `curl`)	✓
Ende → `LEDCLR`	✓
Hardware End-to-End (100 Kanaele)	○
Stresstest (200× Wiederholung)	○

Meilenstein M6

Phase 6 gilt als abgeschlossen, sobald alle Tests in table 47 bestanden sind. ✓

8.4 Phase 7: Produktivbetrieb

In der letzten Phase geht es um Stabilitaet im Dauerbetrieb:

Tabelle 48: Aufgaben fuer den Produktivbetrieb

Aufgabe	Status
systemd-Service aktiviert	✓
Reboot-Test	○
24 h Dauertest	○

Meilenstein M7

Das System laeuft 24 Stunden ohne manuellen Eingriff. ✓

8.5 Zeitschaetzung

Tabelle 49: Geschaetzter Aufwand je Phase

Phase	Aufwand	Status
1–5	6–9 Tage	abgeschlossen
6	2–3 Tage	aktiv
7	0,5 Tage	ausstehend

8.6 Naechste Schritte

Wenn wir die Integration abschliessen wollen, stehen folgende Aufgaben an:

1. Hardware auf $100\times$ skalieren (Schieberegister kaskadieren)
2. 100 Medien-Sets erstellen (Bilder + Audio)
3. End-to-End-Tests mit vollstaendiger Hardware
4. 24 h Dauertest zur Freigabe fuer den Produktivbetrieb

9 Aenderungshistorie

Dieses Kapitel dokumentiert alle wesentlichen Aenderungen am Projekt. Das Format orientiert sich am Standard *Keep a Changelog*.

[2.2.5] – 2025-12-27

Geaendert

- **Dokumentation:** Bloch-Standard angewendet – Fachbegriffe erklaert, Redundanzen entfernt
- `SPEC.md` als Single Source of Truth fuer das Glossar etabliert
- `README.md` fungiert als Hub ohne Duplikate
- Alle Dokumente um ca. 40 % gekuerzt

[2.2.4] – 2025-12-27

Hinzugefuegt

- Praesentation (LaTeX Beamer) mit allen Projektbildern
- CD4021 Timing-Diagramm
- Vollausbau-Blockdiagramm ($100\times$)

Korrigiert

- Verdrahtungsregeln ergaenzt: `SER` $\rightarrow$ `VCC` am letzten IC

[2.2.3] – 2025-12-26

Korrigiert

- Pinbelegung korrigiert: `D4`/`D5` statt `D6`/`D7`
- ID-Schema auf 0-basiert vereinheitlicht (000–099)
- `SPEC.md` als Single Source of Truth

[2.2.0] – 2025-12-25

Hinzugefuegt Phase 5 abgeschlossen – das Web-Dashboard ist nun funktionsfaehig:

- Test-Endpoints: `/test/play/{id}`, `/status`, `/health`
- Debug-Panel mit Live-Logs
- Auto-Reconnect bei WebSocket-Abbruch
- Test-Medien-Generator
- Queue-Overflow-Statistik (`QRESET`)

Geaendert

- Server nutzt Python Virtual Environment
- Dashboard mit explizitem Button „Sound aktivieren“
- `PROTOTYPE_MODE`-Switch fuer die Skalierung zwischen 10 und 100 Kanaelen

[2.1.0] – 2025-12-25

Hinzugefuegt Die ESP32-Firmware erhielt einen Dual-Core-Ausbau:

- **Dual-Core FreeRTOS:** Button-Scan auf Core 0, Serial-Kommunikation auf Core 1
- **Watchdog:** 5 s Timeout mit automatischem Reset
- **Mutex:** Thread-sicherer Hardware-Zugriff
- Neue Befehle: `TEST`, `STOP`, `VERSION`, `STATUS`

Behoben

- Race-Condition bei schnellen Tastendrucken beseitigt

[2.0.0] – 2025-12-25

Geaendert Wechsel des Eingangs-Schieberegisters:

- **CD4021BE** statt 74HC165 (bessere DIP-Verfuegbarkeit)
- Separate Pins: `D0`–`D2` fuer Output, `D3`–`D5` fuer Input
- Load-Logik invertiert: P/S = HIGH zum Laden

Entfernt

- 74HC165-Unterstuetzung (nicht mehr gepflegt)

[1.0.0] – 2025-12-22

Hinzugefuegt Initiale Projektstruktur mit folgenden Komponenten:

- 74HC595 + 74HC165 Schieberegister
- Serial-Protokoll (`PRESS`, `LEDSET`)
- aiohttp-Server mit WebSocket
- Einfaches Dashboard