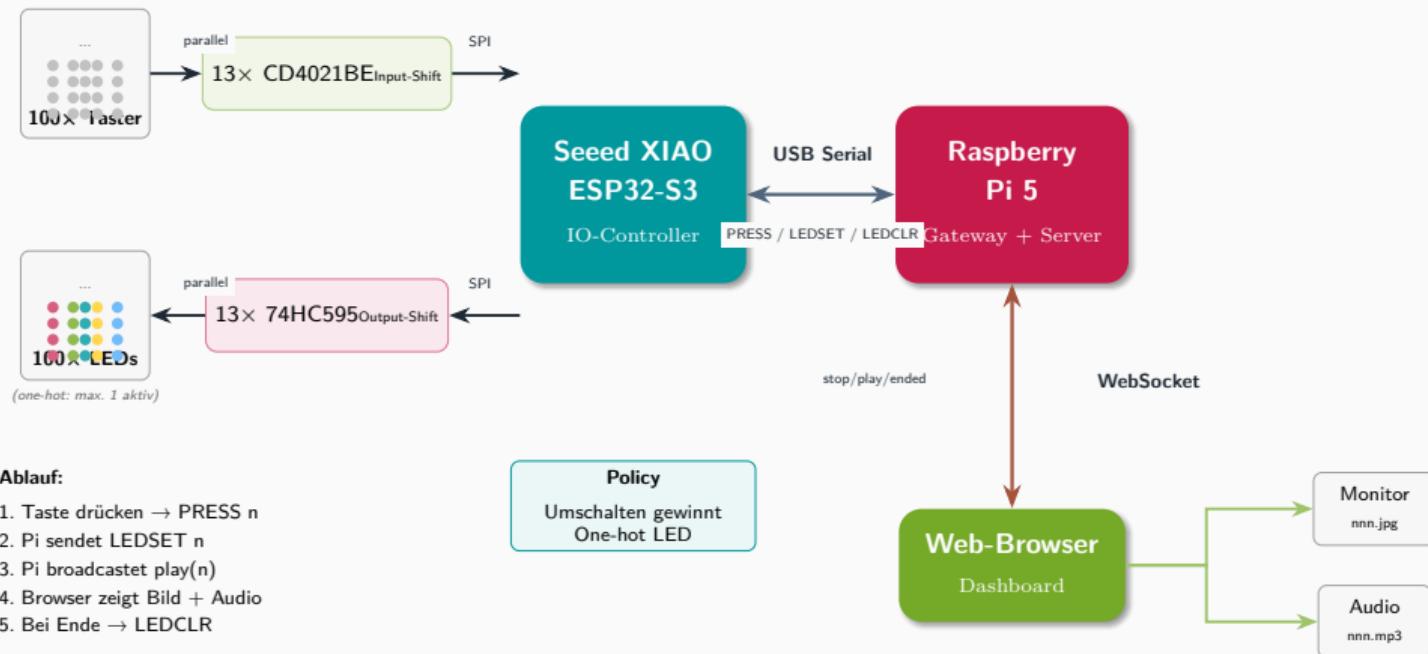
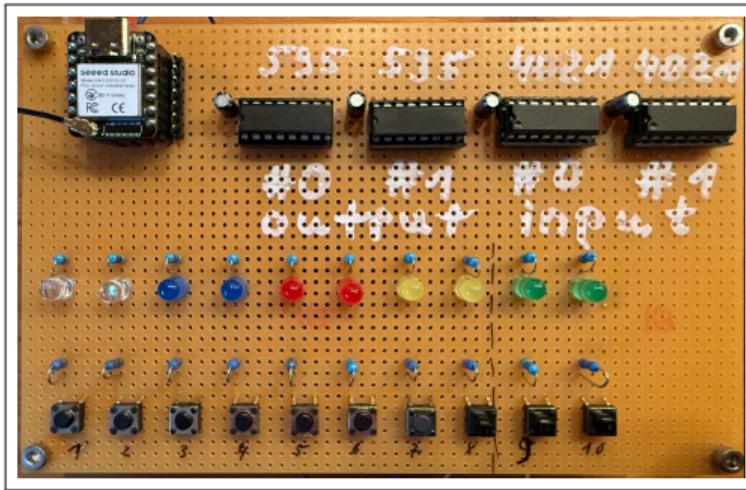


Interaktives Auswahlpanel

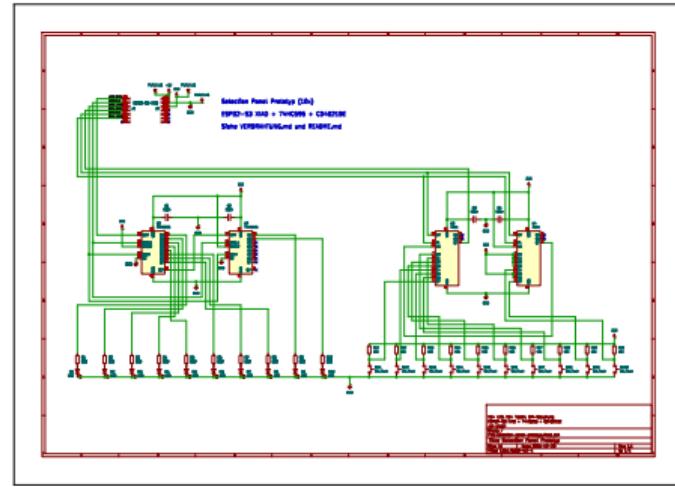
100x Taster/LED + 100x Bild/MP3



Hardware-Prototyp v2



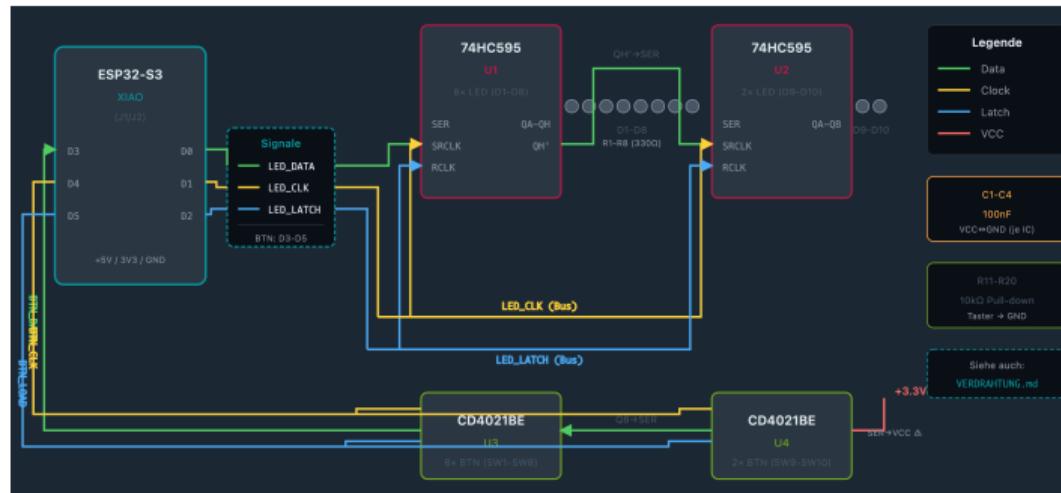
Bestückungsseite – ESP32 + ICs



KiCad – Schaltplan

Komponenten: ESP32-S3 XIAO · 2x 74HC595 · 2x CD4021BE · 10 LEDs · 10 Taster

Schieberegister-Architektur



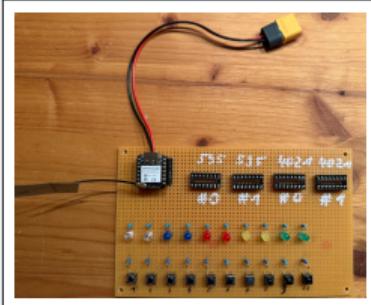
Komponenten

- **ESP32-S3 XIAO** – Mikrocontroller
- **2x 74HC595** – LED-Ausgabe
- **2x CD4021BE** – Taster-Eingabe

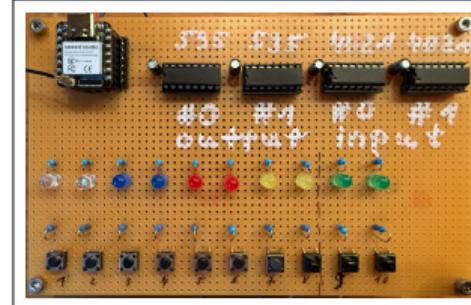
Signale (SPI-ähnlich)

- **Data** – SER/Q8 (grün)
- **Clock** – SRCLK/CLK (gelb)
- **Latch** – RCLK/P/S (blau)

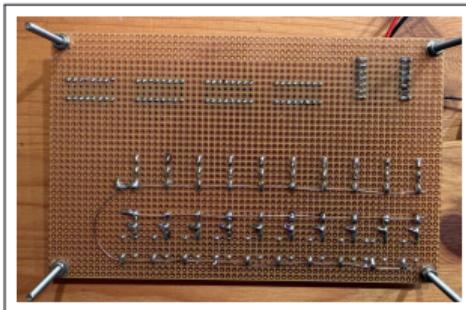
Prototyp-Evolution



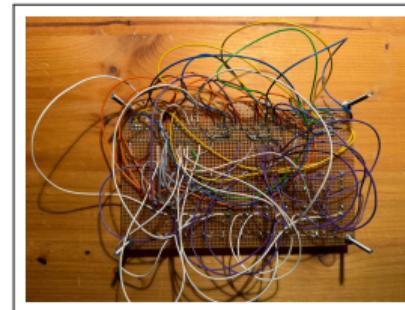
v1 Bestückung – Sockel ohne ICs



v2 Bestückung

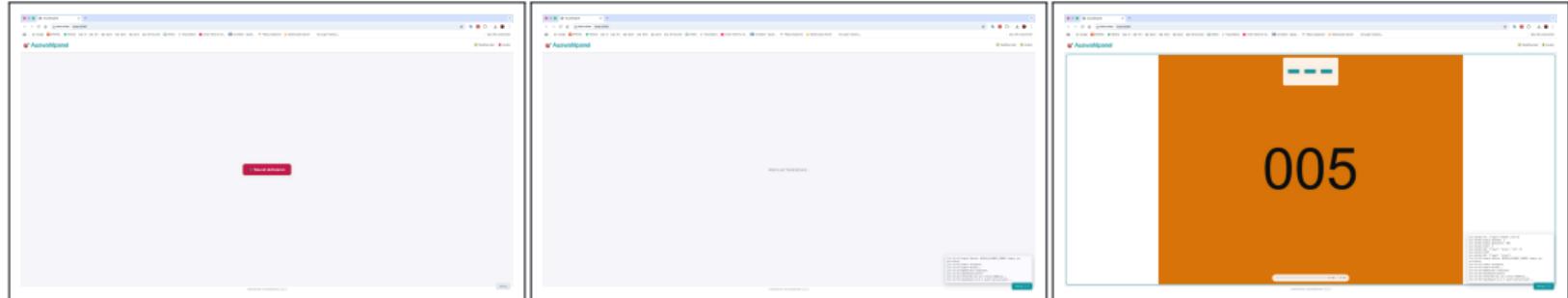


v1 Lötseite



v2 Lötseite

Web-Dashboard



Zugriff

<http://rover:8080/>

Unlock-Button

Audio freischalten

Debug-Panel

Log-Ausgabe (Ctrl+D)

Playback

ID + Bild + Progress

Was ist das Auswahlpanel?

Anwendungsfall

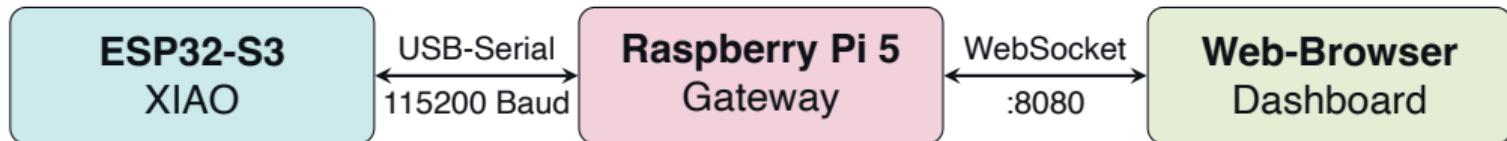
- ▶ 100 physische Taster als Eingabe
- ▶ Jeder Taster triggert ein Bild + Audio
- ▶ LED zeigt aktive Auswahl an

Policy: „Preempt – Umschalten gewinnt“

- ▶ Neuer Tastendruck → **sofortiger Wechsel**
- ▶ **One-hot LED:** Immer nur eine LED aktiv
- ▶ Nach Audio-Ende → alle LEDs aus

Merksatz: Der letzte Tastendruck zählt – kein Warten, kein Queue.

Systemarchitektur – Überblick



Komponente	Aufgabe
ESP32-S3 XIAO	100 Taster scannen, 100 LEDs steuern, Echtzeit
Raspberry Pi 5 Browser	Event-Gateway, WebSocket-Server, Media-Host Bild fullscreen anzeigen, Audio abspielen, Ende melden

Hardware – Schieberegister

Warum Schieberegister?

- ▶ 100 IOs mit 6 GPIO-Pins
- ▶ Kaskadierbar: $13 \times 8 = 104$ Kanäle
- ▶ Günstig und robust

Pinbelegung ESP32-S3

Signal	Pin	Ziel-IC
LED Data	D0	74HC595 SER
LED Clock	D1	74HC595 SRCLK
LED Latch	D2	74HC595 RCLK
Btn Data	D3	CD4021 Q8
Btn Clock	D4	CD4021 CLK
Btn Load	D5	CD4021 P/S



CD4021 Timing-Diagramm

Achtung: CD4021 hat **invertierte** Load-Logik: P/S = **HIGH** für Load!

Firmware – Dual-Core FreeRTOS

Core 0: Button-Task

- ▶ Polling alle 10 ms
- ▶ Debouncing pro Taste (50 ms)
- ▶ Events in FreeRTOS-Queue (16 Slots)

Core 1: Arduino loop()

- ▶ Serial RX/TX (Pi-Kommunikation)
- ▶ LED-Steuerung (State Machine)
- ▶ Command Parser

Vorteil: Serial-Kommunikation blockiert nicht das Button-Scanning.

Design-Prinzipien

Prinzip	Umsetzung
Non-Blocking	State Machines
Thread-Safety	Mutex-Zugriff
Determinismus	vTaskDelayUntil()
Watchdog	5s Timeout
Queue-Stats	Overflow-Counter

Serial-Protokoll

ESP32 → Raspberry Pi

Nachricht	Bedeutung
READY	Controller bereit
PRESS 042	Taster 42 gedrückt
RELEASE 042	Taster 42 losgelassen
OK	Befehl erfolgreich
PONG	Antwort auf PING

Raspberry Pi → ESP32

Befehl	Funktion
LEDSET 042	LED 42 ein (one-hot)
LEDON/LEDOFF	Additiv ein/aus
LEDCLR/LEDALL	Alle aus/ein
PING/STATUS	Test/Zustand
TEST/STOP	LED-Test
VERSION/QRESET	Info/Reset

Debugging: STATUS liefert LED-Zustand, Button-Zustand, Heap, Queue-Free, Overflow-Count.

Raspberry Pi Server

Technologie-Stack:

- ▶ **aiohttp**: Async HTTP/WebSocket-Server
- ▶ **pyserial-asyncio**: Async Serial-Kommunikation
- ▶ **Medien-Validierung**: Prüfung beim Start
- ▶ **systemd**: Autostart als Service

Datenfluss bei Tastendruck:

1. **ESP32** sendet PRESS 042
2. **Server** sendet LEDSET 042 zurück
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 (nur wenn ID noch aktuell)

Race-Condition-Schutz: Ende-Event wird nur verarbeitet, wenn ID noch aktuell.

Dashboard – Implementierung

Features v2.2.4

- ▶ **Fullscreen:** Bild füllt Viewport
- ▶ **Overlays:** ID oben, Progress unten
- ▶ **WebSocket:** Auto-Reconnect (5s)
- ▶ **Shortcuts:** Space, Ctrl+D

Farbschema

- ▶ **Arduino Teal:** Titel, Akzente
- ▶ **Raspberry Red:** Unlock-Button
- ▶ GitHub Dark: Hintergrund

Event-Handler (JavaScript)

- ▶ stop: Audio pausieren, currentTime = 0
- ▶ play: Bild + Audio laden und abspielen
- ▶ audio.onended: Ende-Event senden

Medien-Mapping

- ▶ ID 42 → /media/042.jpg
- ▶ ID 42 → /media/042.mp3

Getestete Funktionen

Firmware (ESP32)

- ▶ PING/PONG Verbindungstest
- ▶ STATUS: Heap, Queue, LEDs, Buttons
- ▶ TEST: LED-Lauflicht (non-blocking)
- ▶ LEDSET/LEDON/LEDOFF/LEDCLR

Server (Raspberry Pi)

- ▶ Serial-Kommunikation stabil
- ▶ WebSocket Broadcast funktioniert
- ▶ systemd Autostart nach Reboot

Dashboard (Browser)

- ▶ Fullscreen-Bildanzeige
- ▶ Audio mit Preempt-Verhalten
- ▶ Auto-Reconnect bei Verbindungsabbruch
- ▶ Debug-Panel mit Live-Logs

Ausstehend

- ▶ Hardware-Test mit echten Buttons
- ▶ CD4021BE + 74HC595 ICs bestellt

Live-Demo

Dashboard öffnen:

```
http://rover:8080/
```

Tastendruck simulieren:

```
curl http://rover:8080/test/play/5
```

Preempt testen (während Audio läuft):

```
curl http://rover:8080/test/play/3
```

Status abfragen:

```
curl http://rover:8080/status  
curl http://rover:8080/health
```

Ergebnis: Bild wechselt fullscreen, Audio startet, Debug-Panel zeigt Logs.

Ausblick: Vollausbau 100x



Änderungen für 100x

- ▶ 13x 74HC595 (statt 2x)
- ▶ 13x CD4021BE (statt 2x)
- ▶ PROTOTYPE_MODE = false
- ▶ Gleiche Firmware/Server

Stückliste (100x)

Bauteil	Anzahl
74HC595	13x
CD4021BE	13x
LED 5mm	100x
Taster	100x
Widerstand 220Ω	100x

Zusammenfassung

Phase	Inhalt	Status
1	Infrastruktur & Toolchain	✓
2	Hardware-Prototyp (10x)	✓
3	ESP32 Firmware v2.2.0	✓
4	Raspberry Pi Server v2.2.0	✓
5	Web-Dashboard v2.2.4	✓
6	Hardware-Integration	◀ Warte auf ICs
7	Produktivbetrieb (100x)	□

Kernbotschaft: ESP32 für Echtzeit, Pi als Gateway, Browser für UI. Skalierung 10→100 per PROTOTYPE_MODE Switch.

Übertragbarkeit: Voting-Systeme, Quiz-Panels, Museumsinstallationen.