# ZAMusiclight

Echtzeitfähige, bespielbare LED-Installation.
Verantwortlich: Thomas Kolb aka. cfr34k <m-zam@tkolb.de>

Dies soll ein Deko-Objekt für das ZAM selbst werden. Derzeit besteht es aus einer LED-Leiste, die im Couch-Bereich hängt und über das Netzwerk live angesteuert werden kann.

Aktuell zeigt die Leiste einige von mir fest einprogrammierte Animationen an. Wenn Musik in der Couchecke abgespielt wird, wird diese auf den LEDs visualisiert. Prinzipell ist es aber für alle, die wollen, möglich, eigene Animation zu programmieren. Doku dazu folgt in Kürze.


- Schnellstart
- Aufbau und Beispielskript
- Technische Details

# Schnellstart

- Repository klonen: `git clone https://github.com/zam-haus/zamusiclight-scripts.git`

  oder aktuelles Zip-Archiv herunterladen
- Beispielskript *rgbw_sinus.py* als eigenes Skript kopieren
- Skript im ZAM ausprobieren:
  - Netzteil für die LED-Leiste (links an der LED-Leiste im Acrylgehäuse) einstecken. Bitte eine Minute warten, bis der ESP32 fertig gebootet hat. Sonst funktioniert die Leerlaufanimation nicht.
  - Falls der Musikrechner (*turntable*) gerade Musik abspielt, muss Musiclight vorher beendet werden. Dazu gibt es dort auf dem Desktop ein Skript (*stop-musiclight.sh*). Dieses einfach doppelklicken. Die Musiclight-Animation stoppt, wenn es funktioniert hat.
- Dein Skript funktioniert und du willst es mit anderen teilen? Dann schick einen Pull

  Request auf Github oder sende es an m-zam@tkolb.de (dann nehme ich es in’s Repo auf).

# Aufbau und Beispielskript

## Aufbau und Adressierung der LEDs

Das folgende Bild zeigt den aktuellen Aufbau im ZAM mit der Adressierung der LEDs.

ZAMusiclight_v1.jpg

Wie man sieht, ist der Strip in zwei logische Strips aufgeteilt, die jeweils aus 150 LEDs bestehen und „von unten nach oben“ adressiert werden. Dies erlaubt die einfachere Programmierung von symmetrischen Animationen.

## Beispielskript

Auf meinem Gitea gibt es jetzt ein in Python geschriebenes Beispielskript. Das Repository enthält außerdem eine Programmbibliothek, die das Netzwerkprotokoll abstrahiert und einfach verwendbar macht. Wer sich nicht mit der Git-Versionsverwaltung herumschlagen will, kann den Repository-Inhalt auch direkt herunterladen (der Link zeigt jeweils auf die neueste Version).

Als Inspiration können evtl. auch die Skripte in einem älteren Repo dienen, die das gleiche Protokoll verwenden, aber für einen anderen Aufbau geschrieben sind.

ℹ Die ZAMusiclight-spezifischen Skripte sind jetzt in ein eigenes Github-Repo umgezogen! Bitte aktualisiert eure Git-Remotes. Falls ihr selbst ein Skript geschrieben habt und dieses gerne veröffentlichen würdet, schickt gerne einen Pull-Request :-)

# Technische Details

# Hardware

## LED-Leiste

Die LED-Leiste besteht aus SK6812-LEDs, die einzeln angesteuert werden können. Jede LED enthält vier LED-Chips in verschiedenen Farben: rot, grün, blau und (neutral-)weiß. Es gibt auch Varianten mit warm- und kaltweiß zu kaufen, die neutralweiße hat jedoch den Vorteil, dass die Farbtemperatur der weißen LED ziemlich gut dem Sonnenlicht entspricht und sich damit gut zur Farbmischung eignet.

Die Leiste ist 5m lang und enthält 300 LEDs.

## Mikrocontroller

Zur Ansteuerung der LED-Leiste kommt ein ESP32 zum Einsatz. Dabei handelt es sich um einen netzwerkfähigen Dual-Core-Mikrocontroller. Netzwerkfähig heißt in diesem Fall, dass er WLAN, Bluetooth Low Energy und auch Ethernet unterstützt. Außerdem hat er genug Rechenleistung, um auch auf dem Mikrocontroller direkt Animationen für die LEDs berechnen zu können.

Die Firmware ist in C++ mit dem Arduino-Framework und der PlatformIO-IDE programmiert. Der Code kann auf meinem Gitea heruntergeladen werden (der Git-Branch `bnb` enthalt die für's ZAM relevanten Anpassungen).

# Ansteuerung im Netzwerk

## Adresse

Die LED-Leiste ist unter der Adresse **zamusiclight.im.zam.haus** im ZAM-Netzwerk erreichbar.

## Dienste

Folgende Dienste laufen auf dem ESP32:

- Ein HTTP-Server, über den die fest einprogrammierten Funktionen gesteuert werden können (Animationswechsel, Anzeige einfacher Farben) =>

  http://zamusiclight.im.zam.haus/
- Ein UDP-Server, der die Echtzeitansteuerung ermöglicht. Dieser wird weiter unten genauer beschrieben.

⚠ **Wichtig: Eine Ansteuerung über UDP „gewinnt“ immer gegenüber den Einstellungen auf dem HTTP-Server.** Sobald ein Kommando über UDP empfangen wurde, wird die interne Animation unterbrochen und das „Live-Bild“ angezeigt. Die interne Animation wird fortgesetzt, wenn für 3 Sekunden kein UDP-Paket empfangen wurde.

# Der HTTP-Server

Der ESP32 stellt per HTTP eine sehr spartanische Startseite bereit, auf der einige vordefinierte Animationen umgeschaltet werden können. Da die Firmware ursprünglich für einen anderen Aufbau geschrieben wurde, sind manche Animationen noch nicht an den Aufbau im ZAM angepasst, also bitte nicht wundern, wenn mal etwas komisch aussieht.

Zusätzlich bietet der HTTP-Server eine API an, die z.B. das Setzen von Farben für die ganze Leiste ermöglicht, z.B. setzt folgendes ein hübsches Frühlingsgrün:

http://zamusiclight.im.zam.haus/api/color?color=20800010

Es gibt auch eine API für Textanzeige, die aber am ZAM-Aufbau nicht sinnvoll umsetzbar ist:

http://zamusiclight.im.zam.haus/api/text?text=Hallo%20ZAM!&color=20800010

# Echtzeit-Ansteuerung über UDP

Der ESP32 wartet auf **Port 2703** auf Kommandos.

Ein Kommando ist wie folgt aufgebaut:

| **Byte** | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|---|
| **Bedeutung** | Kommandotyp | Strip-Index | LED-Index | Rot-Wert | Grün-Wert | Blau-Wert | Weiß-Wert |

Es gibt folgende Kommandos zum Setzen bzw. Verändern von Farben:

- **Farbe setzen** (Kommandotyp **0x00**): setzt die gegebene LED direkt auf die gegebene Farbe.
- **Farbe überblenden** (Kommandotyp **0x01**): blendet die LED linear auf die gegebene Farbe über.

- **Farbe aufhellen** (Kommandotyp **0x02**): addiert die gegebene Farbe auf die aktuelle Farbe der LED.

Außerdem gibt es folgende Steuerbefehle. Für diese werden die Index-Bytes und Farbwerte speziell interpretiert:

- **Überblendgeschwindigkeit setzen** (Kommandotyp **0x03**): Setzt die Schrittweite pro Frame bei der Überblendung. Diese ist im Rot-Wert (Byte 3) angegeben, alle anderen Bytes werden ignoriert. Schrittweite 1 ist am langsamsten, Schrittweite 255 setzt die Farbe direkt.
- **Frame beenden** (Kommandotyp **0xFE**): Schließt den aktuellen Frame nach diesem Paket ab. Sollte das Paket noch weitere Kommandos enthalten, werden diese noch verarbeitet, es werden aber keine weiteren Pakete für diesen Frame gelesen. Die Bytes 1 bis 6 werden ignoriert.
- **Bestätigung anfordern** (Kommandotyp **0xFF**): Fordert eine Antwort vom ESP an. Die Antwort ist ein UDP-Paket mit 2 Bytes: das erste ist der Rot-Wert aus dem Kommando, das zweite der Grün-Wert. Dieses Kommando ist nützlich, um eine Flusskontrolle umsetzen zu können, indem in den beiden Bytes ein Zähler mitgeschickt wird. Hinkt dieser Zähler in den Antworten hinterher, weiß man, dass der ESP die Pakete nicht schnell genug verarbeiten kann. Dann sollte die Senderate gedrosselt werden.

In einem UDP-Paket können bis zu 210 Kommandos aneinandergereiht werden. Bei mehr Kommandos müsste das UDP-Paket fragmentiert werden, was der ESP32 nicht unterstützt. Pro Frame verarbeitet der ESP32 jedoch bis zu 3 UDP-Pakete.