

Protokoll PingPong

Kilian Hornischer & Niklas Maderbacher

11.11.2025

Contents

PingPong mit LED-Streifen und Fußtaster für Tag der offenen Tür	1
Aufgabenstellung	1
Vorgegebene Bauteile	1
WS2811 LED Strip	2
Spannungskonverter	2
Level Shifter 3.3V -> 5V	3
Fußtaster	3
Schaltung	5
Code	5
Includes	5
Pin definitionen und Anzahl an LEDs am LED Strip angeben	5
Zustandsvariablen	5
Setup	6
Endzonen Handler	7
Ansteuern der LED	9
Loop	11

PingPong mit LED-Streifen und Fußtaster für Tag der offenen Tür

Aufgabenstellung

Entwicklung eines PingPong Spieles für den Tag der offenen Tür.

Vorgegebene Bauteile

- WS2811 LED Strip
- 12V 5A Power supply AC/DC
- Buzzer
- Spannungskonverter 12V -> 5V
- ESP8266MOD
- Level Shifter 3.3V -> 5V
- Fußtaster

WS2811 LED Strip



3-Lead 12V RGB LED Strip (WS2811 + 3x5050LED)

Figure 1: LED Strip Aufbau

Der LED Strip besteht aus mehreren Segmenten, wobei ein Segment mit drei LEDs bestückt ist. Man kann dabei nur das Segment als ganzes ansteuern, näheres bei der Implementierung

Spannungskonverter

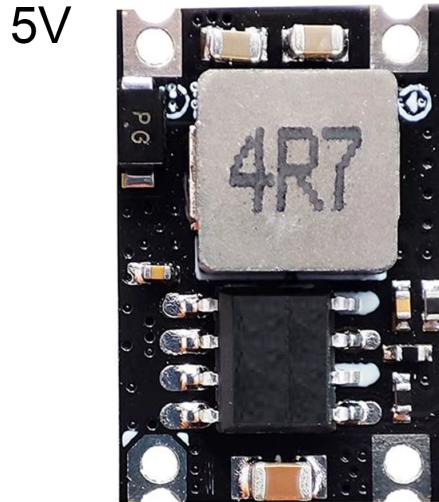


Figure 2: Spannungskonverter

Der Spannungskonverter, wie der Name bereits sagt, ändert die Spannung von 12V auf 5V ab, da das Netzteil 12V liefert, womit der LED Strip arbeitet, und 5V womit der ESP arbeitet.

Level Shifter 3.3V -> 5V

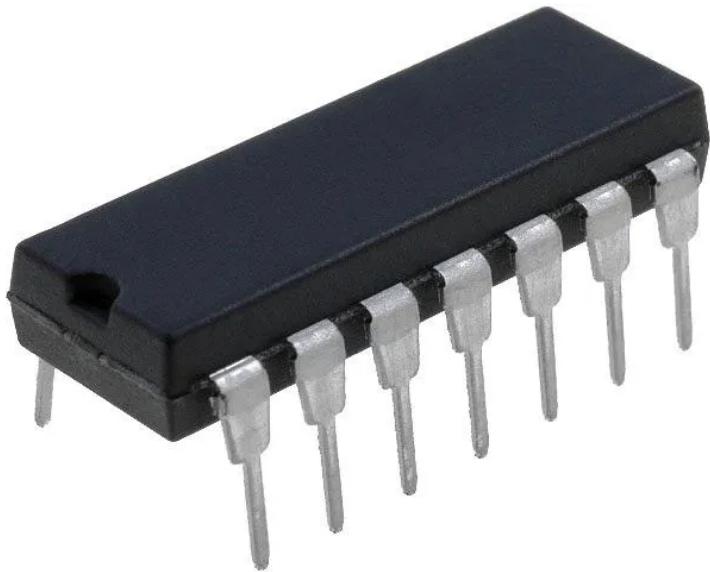


Figure 3: Level Shifter

Da der ESP nur eine Spannung von 3.3V ausgibt und der LED Strip nur mit einer Spannung von 5V arbeiten kann, wird ein Level Shifter benötigt, um den LED Strip ansteuern zu können

Fußtaster



Figure 4: Fußtaster

Der Fußtaster (3PDT Footswitch) ändert seinen Zustand erst **nach** dem Drücken und funktioniert daher nicht wie ein herkömmlicher Taster, der **während** er gedrückt wird HIGH ausgibt, sondern ändert mit dem drücken zwischen HIGH und LOW.

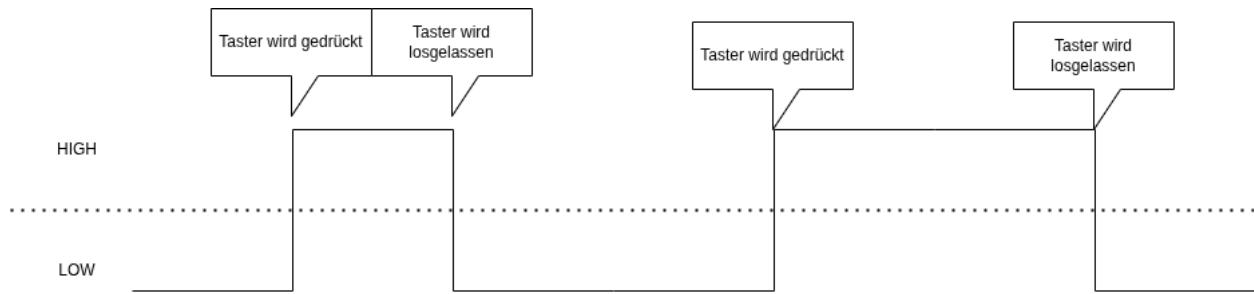


Figure 5: normaler Taster

Wie man in der Grafik sehen kann, ist das Signal so lange HIGH, wie der Taster betätigt wird.

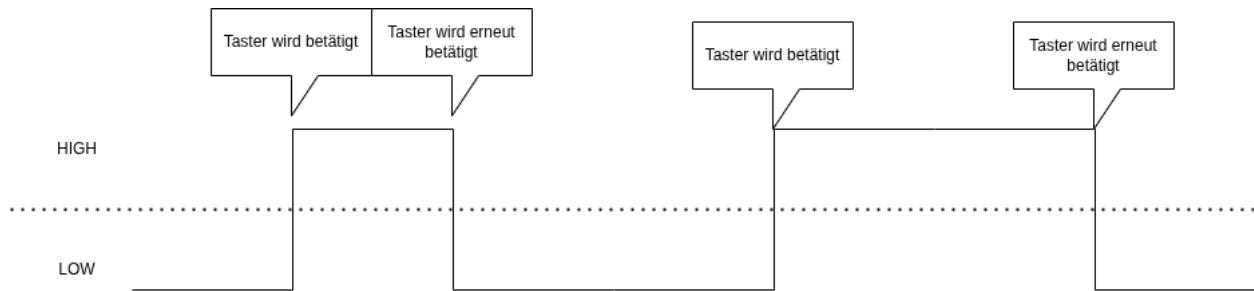


Figure 6: Fußtaster

Wie man in der Grafik sehen kann, ändert das Signal nach dem drücken auf HIGH und wird erst mit erneutem drücken wieder LOW.

Schaltung

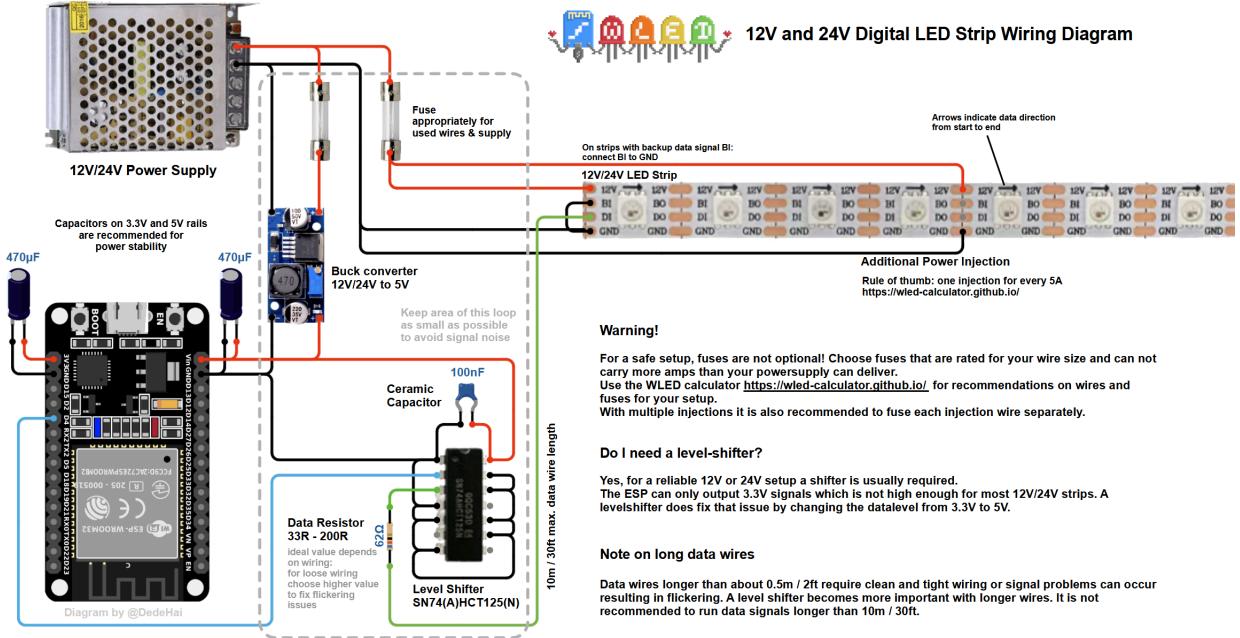


Figure 7: Schaltplan

Wichtig hierbei zu beachten ist, dass der Fußtaster hierbei nicht enthalten ist, wobei dieser einfach am ESP angeschlossen werden kann.

Code

Wichtig hierbei zu beachten ist, dass der Code nicht vollständig ist und nur teilweise funktioniert.

Includes

```
#include <Arduino.h>
#include <FastLED.h>
```

Arduino.h bindet die grundlegenden Arduino-Funktionen und -Definitionen ein (z. B. digitalWrite(), pinMode(), millis()), damit dein Sketch mit der Arduino-Plattform funktioniert. FastLED.h lädt die FastLED-Bibliothek, die speziell zur einfachen Ansteuerung und Kontrolle von adressierbaren LEDs (z. B. WS2812, SK6812) entwickelt wurde.

Pin definitionen und Anzahl an LEDs am LED Strip angeben

```
#define NUM_LEDS 60
#define DATA_PIN D1
#define SWITCH_PIN D2
```

Zustandsvariablen

```
CRGB leds[NUM_LEDS];

unsigned long previousMillis = 0;
const long interval = 50;
```

```

unsigned long currentMillis = 0;

int currentPosition = 0;
bool reverseDirection = false;

bool running = true;

bool currentSwitchState = HIGH;
bool oldSwitchState = LOW;

const int TIMING_WINDOW_SIZE = 5;
const unsigned long TIMING_WINDOW_START = 500;
const unsigned long TIMING_WINDOW_END = 2000;
unsigned long timingWindowStartTime = 0;
bool inTimingWindow = false;
bool timingWindowActive = false;

enum LEDState
{
    RUNLIGHT_ACTIVE
};
LEDState currentState = RUNLIGHT_ACTIVE;



- CRGB leds[NUM_LEDS]: Array mit allen LEDs
- previousMillis, interval, currentMillis: implementieren ein nicht-blockierendes Timing-Schema
- currentPosition: Welches LED gerade angesteuert wird
- reverseDirection: Ob LED Strip nach vorne oder hinten läuft
- running: Ob das Lauflicht läuft
- currentSwitchState und oldSwitchState zur Flankenerkennung bezüglich des Fußtasters
- TIMING_WINDOW_SIZE: Wie groß die Endzone sein soll
- TIMING_WINDOW_START, TIMING_WINDOW_END: Zeitfenster, in dem der User den Fußtaster betätigen kann, damit das Lauflicht die Richtung ändert
- timingWindowStartTime, inTimingWindow, timingWindowActive: Verfolgen den Zustand der Endzone

```

Setup

```

void setup()
{
    Serial.begin(115200);
    FastLED.addLeds<WS2811, DATA_PIN, GRB>(leds, NUM_LEDS);
    FastLED.setBrightness(50);

    pinMode(SWITCH_PIN, INPUT_PULLUP);

    currentSwitchState = digitalRead(SWITCH_PIN);

    FastLED.clear();
    FastLED.show();
}



- FastLED.addLeds<WS2811, DATA_PIN, GRB>(leds, NUM_LEDS): konfiguriert den LED Strip
  - Protokoll WS2811
  - Farbreihenfolge GRB
- FastLED.setBrightness(50): Reduzierung der Helligkeit
- pinMode(SWITCH_PIN, INPUT_PULLUP):

```

Endzonen Handler

```
void checkTimingWindow()
{
    currentSwitchState = digitalRead(SWITCH_PIN);

    if (running && !timingWindowActive)
    {
        bool nearStart = (!reverseDirection && currentPosition >= NUM_LEDS - TIMING_WINDOW_SIZE);
        bool nearEnd = (reverseDirection && currentPosition < TIMING_WINDOW_SIZE);

        if (nearStart || nearEnd)
        {
            timingWindowActive = true;
            inTimingWindow = false;
            timingWindowStartTime = currentMillis;
        }
    }

    if (timingWindowActive)
    {
        unsigned long windowElapsed = currentMillis - timingWindowStartTime;

        if (windowElapsed >= TIMING_WINDOW_START && windowElapsed <= TIMING_WINDOW_END)
        {
            if (!inTimingWindow)
            {
                inTimingWindow = true;
                Serial.println(">>> TIMING FENSTER JETZT AKTIV - BUTTON DRÜCKEN! <<<");
            }

            if (windowElapsed % 200 < 100)
            {
                for (int i = 0; i < TIMING_WINDOW_SIZE; i++)
                {
                    if (!reverseDirection)
                    {
                        leds[NUM_LEDS - 1 - i] = CRGB::Yellow;
                    }
                    else
                    {
                        leds[i] = CRGB::Yellow;
                    }
                }
                FastLED.show();
            }
        }
        if (currentSwitchState == LOW && oldSwitchState == HIGH)
        {
            reverseDirection = !reverseDirection;
            oldSwitchState = LOW;

            if (reverseDirection)
            {

```

```

        currentPosition = NUM_LEDS - 1;
    }
    else
    {
        currentPosition = 0;
    }

    timingWindowActive = false;
    inTimingWindow = false;
    running = true;

    FastLED.clear();
    return;
}
}
else if (windowElapsed > TIMING_WINDOW_END)
{
    timingWindowActive = false;
    inTimingWindow = false;
    running = false;

    for (int i = 0; i < NUM_LEDS; i++)
    {
        leds[i] = CRGB::Red;
    }
    FastLED.show();
    delay(1000);
    FastLED.clear();
    FastLED.show();
}
}

if (!running && !timingWindowActive && currentSwitchState == LOW && oldSwitchState == HIGH)
{
    reverseDirection = false;
    currentPosition = 0;
    running = true;
    oldSwitchState = LOW;

    Serial.println("!!! Neustart vom Anfang !!!");
}

if (currentSwitchState == HIGH)
{
    oldSwitchState = HIGH;
}
}

```

Die Funktion liest zuerst den aktuellen Zustand des Tasters ein und prüft, ob das Lauflicht gerade läuft und noch kein Timing-Fenster aktiv ist; befindet sich die aktuelle LED-Position nahe dem definierten Anfangs- oder Endbereich, wird das Timing-Fenster aktiviert, die Startzeit gespeichert und eine Aufforderung über die serielle Schnittstelle ausgegeben. Sobald ein Timing-Fenster aktiv ist, berechnet die Routine die verstrichene Zeit seit Aktivierung und erkennt, ob diese Zeit innerhalb des gültigen Bereichs liegt; beim erstmaligen Eintritt in dieses gültige Intervall wird `inTimingWindow` gesetzt und eine zusätzliche Meldung ausgegeben.

Innerhalb des gültigen Zeitbereichs werden die LEDs der Fensterzone periodisch gelb zum Blinken gesetzt, um visuelles Feedback zu geben, und es wird auf eine Flanke des Tasters (HIGH → LOW) geprüft; trifft ein solcher Tastendruck zu, wird die Laufrichtung umgeschaltet, die Position ans jeweilige Ende gesetzt, das Fenster deaktiviert, die Laufvariable `running` auf true belassen, die visuellen Hinweise gelöscht und die Funktion vorzeitig verlassen. Überschreitet die verstrichene Zeit das Ende des Fensters, so wird das Lauflicht gestoppt, das Fenster deaktiviert, eine Fehlermeldung ausgegeben und kurz ein rotes Failure-Signal am Streifen gezeigt, bevor die LEDs wieder gelöscht werden. Wenn das Lauflicht gestoppt ist und kein Fenster aktiv ist, löst ein erneuter Tastendruck einen Neustart vom Anfang aus (Richtung zurückgesetzt, Position 0, `running = true`) aus. Abschließend wird der alte Schalterzustand auf HIGH gesetzt, sobald der Taster nicht gedrückt ist, damit künftige LOW-Flanken korrekt erkannt werden.

Ansteuern der LED

```
void light_led()
{
    switch (currentState)
    {
        case RUNLIGHT_ACTIVE:
            if (currentMillis - previousMillis >= interval && running && !timingWindowActive)
            {
                FastLED.clear();

                // Main LED
                leds[currentPosition] = CRGB::Red;

                // Trail effect
                for (int i = 1; i <= 3; i++)
                {
                    int trailPos = currentPosition - (reverseDirection ? -i : i);
                    if (trailPos >= 0 && trailPos < NUM_LEDS)
                    {
                        leds[trailPos] = CRGB(255 / (i * 2), 0, 0);
                    }
                }

                // Highlight timing window area
                if (running)
                {
                    for (int i = 0; i < TIMING_WINDOW_SIZE; i++)
                    {
                        if (!reverseDirection && currentPosition >= NUM_LEDS - TIMING_WINDOW_SIZE)
                        {
                            leds[NUM_LEDS - 1 - i] = CRGB::Blue; // Blue for upcoming timing window
                        }
                        else if (reverseDirection && currentPosition < TIMING_WINDOW_SIZE)
                        {
                            leds[i] = CRGB::Blue; // Blue for upcoming timing window
                        }
                    }
                }
            }

            FastLED.show();

            if (currentPosition % 10 == 0)
```

```

{
    Serial.print("LED Position: ");
    Serial.print(currentPosition);
    Serial.print(" | Richtung: ");
    Serial.print(reverseDirection ? "Rückwärts" : "Vorwärts");
    Serial.print(" | Laufend: ");
    Serial.print(running ? "Ja" : "Nein");
    Serial.print(" | Timing Fenster: ");
    Serial.println(timingWindowActive ? "Aktiv" : "Inaktiv");
}

// Move position
if (!reverseDirection)
{
    currentPosition++;
    if (currentPosition >= NUM_LEDS)
    {
        currentPosition = NUM_LEDS; // Set to end to trigger timing window
    }
}
else
{
    currentPosition--;
    if (currentPosition < 0)
    {
        currentPosition = -1; // Set to before start to trigger timing window
    }
}

previousMillis = currentMillis;
}
break;
}
}

```

Die Funktion `light_led()` steuert das eigentliche Lauflicht und wird ständig in der Hauptschleife aufgerufen. Sie überprüft zuerst, ob der aktuelle Zustand `RUNLIGHT_ACTIVE` aktiv ist und ob seit dem letzten Aufruf genügend Zeit (`interval`) vergangen ist; nur dann, und wenn das Lauflicht läuft und kein Timing-Fenster aktiv ist, wird der Inhalt ausgeführt. Zunächst werden alle LEDs gelöscht, danach wird die LED an der aktuellen Position rot eingeschaltet, wodurch die Hauptleuchtposition des Lauflichts entsteht. Anschließend wird ein Nachleuchteffekt erzeugt: drei LEDs hinter der Hauptposition (abhängig von der Laufrichtung) werden schwächer rot eingefärbt, wodurch der Eindruck einer Bewegung entsteht. Danach prüft die Funktion, ob sich das Lauflicht nahe dem Timing-Fenster befindet; in diesem Fall werden die betreffenden LEDs blau markiert, um den Spieler visuell darauf hinzuweisen, dass bald ein Zeitfenster folgt, in dem ein Tastendruck erforderlich ist. Nach dem Setzen aller Farben werden die neuen Werte mit `FastLED.show()` an den LED-Streifen übertragen. Alle zehn Positionen werden zusätzlich über die serielle Schnittstelle ein kurzer Statusbericht ausgegeben, der die aktuelle Position, Richtung, den Laufzustand und den Status des Timing-Fensters zeigt. Schließlich wird die Position der leuchtenden LED um eins in die entsprechende Richtung verschoben – vorwärts oder rückwärts, je nach `reverseDirection`; erreicht die Position dabei das Ende des Streifens oder fällt sie unter null, wird sie auf einen speziellen Wert (`NUM_LEDS` oder `-1`) gesetzt, um das Timing-Fenster im nächsten Schritt auszulösen. Zum Schluss wird `previousMillis` aktualisiert, um das Intervall für den nächsten Schritt neu zu starten.

Loop

```
void loop()
{
    currentMillis = millis();

    checkTimingWindow();
    light_led();
}
```

In der `loop` Funktion werden mit jedem Durchlauf die aktuellen Millisekunden neu gesetzt und die `checkTimingWindow` Funktion, zum überprüfen der Endzone und die `light_led` Funktion, um die LEDs zum leuchten zu bringen.