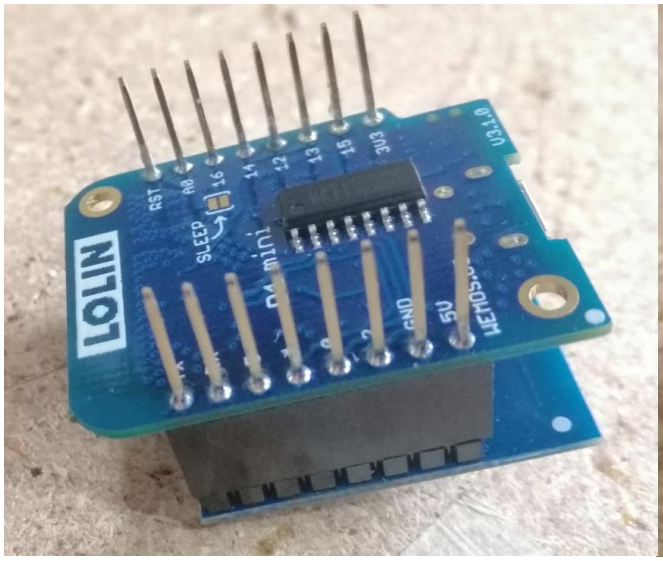
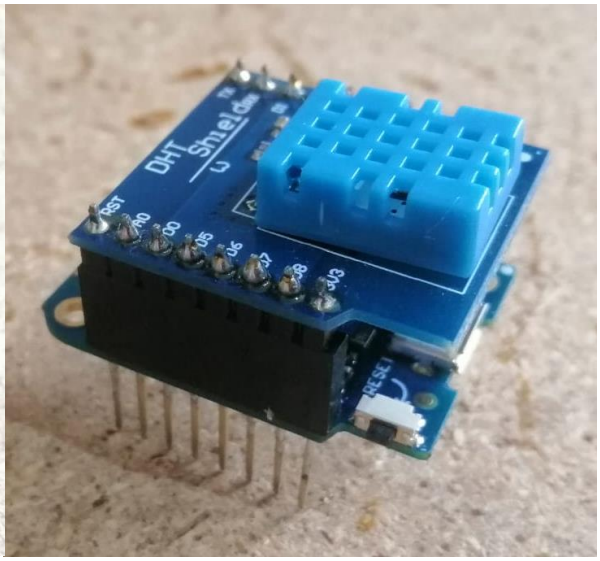
Übungsblatt 6: ESP8266 & M5Stack

Für Code und Videos siehe Ordner „Code“ und „Anhang“

Aufgabe 1.1





Anmerkungen

- In PlatformIO muss „WEMOS D1 mini pro“ oder „WEMOSE D1 mini light“ ausgewählt werden („WEMOS D1 MINI ESP32“ ist inkorrekt, Mikrocontroller kann dann nicht erkannt werden)
- Im Code kann per DHT dht(Dht_type, Dht_PIN) ein DHT Objekt erzeugt werden
  - ➔ Dht_type in unserem Fall: DHT **11**
  - ➔ Dht_PIN: in unserem Fall: PIN **D4** (kann **nicht** geändert werden: http://www.esp8266learning.com/wemos-dh11-shield-example.php)

Aufgabe 1.2

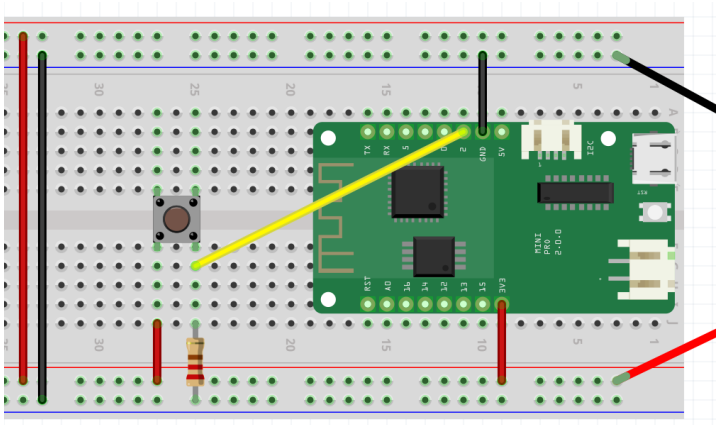
- Passwort, SSID wurden in einer extra Datei „secret.h“ angelegt (außerdem habe ich die IP-Adresse vor dem Pushen auf Github gelöscht)
- Code für Client und Server wurde größtenteils aus den gegebenen Codebeispielen hergenommen
- Wichtig: für den WEMOS D1 Mini muss `<ESP8266WiFi.h>` inkludiert werden
- Temperatur und Luftfeuchtigkeit werden alle fünf Sekunden von dem Sensor gemessen und dann per Wifi an den PC gesendet
- Gesendet werden die Daten per print() Methode des Clients, welcher man einen String/ die Message übergeben kann. Der String, den ich als Message übergebe, besteht dabei aus dem Float-Wert der Temperatur (in °C) und dem Float-Wert der Luftfeuchtigkeit (in %), wobei beide durch ein Leerzeichen getrennt sind, um sie auf Server-Seite wieder korrekt auslesen zu können.
- Auf der Server-Seite empfangen wir die Daten als String. Indem ich einfach die Daten am Leerzeichen teile, erhalte ich meine zwei einzelnen Messwerte
- Zusätzlich habe ich die **win10toast** Python-Library installiert, um mir Toast-Messages am PC ausgeben zu lassen, wenn …
  - ➔ … Luftfeuchtigkeit zu hoch ist (über 60%): die Meldung informiert mich dann, dass mal wieder gelüftet werden sollte

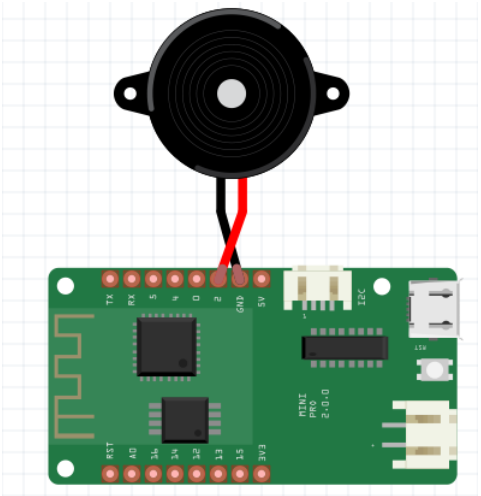
➔ … Temperatur zu hoch ist (über 25°C): die Meldung informiert mich dann, dass ich bei der Hitze genügend trinken sollte

- Win10toast Library: https://pypi.org/project/win10toast/

Aufgabe 2: WiFi Türklingel

- Der WEMOS, bei dem sich der Button für die Klingel befindet, dient als Client, der WEMOS, der mit dem Lautsprecher verbunden ist, als Server
- Einen WEMOS (der mit Lautsprecher) versorge ich mit Strom über USB an meinem Laptop, den anderen mit dem SBC-POW-BB Breadboard Power-supplier (Wichtig: WEMOS mit 3.3V versorgen!)
- Die zwei Schaltungen (siehe Bilder weiter unten) waren schnell und einfach umgesetzt: (links der Wemos mit Button, rechts der Wemos mit Lautsprecher)
  ➔ Anmerkung: in den Bildern unten wurde ein WEMOS d1 mini Pro verwendet (da ich den WEMOS d1 mini light nicht finden konnte). Die digitalen PINS sind hier etwas anders beschriftet.
- Code für die Wifi Connection größtenteils aus dem gegebenen Sample Code übernommen
- Passwort, SSID und IP sind geheim und in die extra Datei „secret.h“ ausgelagert, welche nicht mit auf Github gepushed wird
- Ich habe die Klingel im Code so implementiert, dass immer ein gleich langer Ton (per tone(PIN, frequency, duration) Methode) abgespielt wird, unabhängig wie lange die Klingel gedrückt wird. D.h. drückt jemand auf die Klingel und hält diese gedrückt, so ertönt trotzdem nur einmal das Klingelgeräusch.



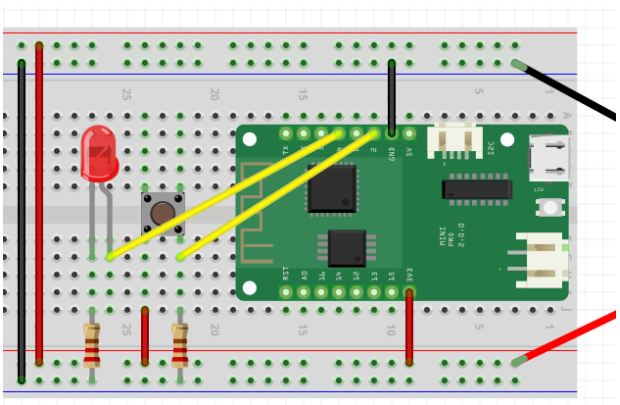


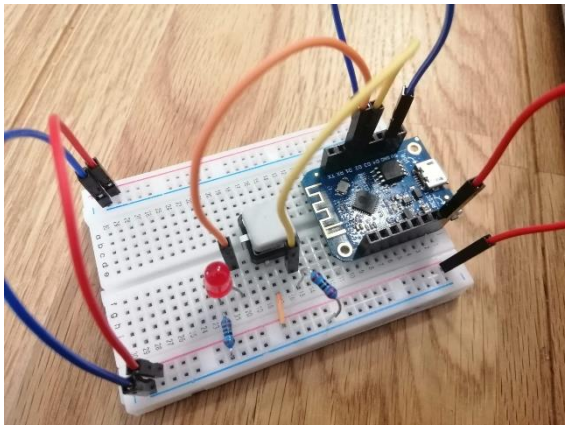
Aufgabe 3: WiFi Telegraph

- Schaltung für beide WEMOS identisch (siehe Bilder unten)
- Auch der Code für beide ist nahezu identisch. Einziger Unterschied: ein WEMOS dient als Client (verbindet sich mit Server), der andere als Server (initiiert Verbindung, wartet auf Clients, die sich verbinden). Welcher WEMOS welche Funktion übernimmt ist egal, da beide sowohl Daten senden, als auch empfangen können.
- Es werden immer nur Daten an den jeweils anderen WEMOS geschickt, wenn sich der Status des Buttons ändert. D.h. sobald man auf den Button klickt wird einmal ein HIGH Signal an den anderen WEMOS geschickt. Hält man den Button weiterhin gedrückt, wird nichts

gesendet. Lässt man dann den Button los, wird einmalig LOW gesendet. Dann wird wieder solange nichts gesendet, bis der Button wieder gedrückt werden würde.

➔ Auf diese Weise kann verhindert werden, dass in jeder Loop Iteration Daten gesendet werden, die den Zustand der LED aber eigentlich nicht verändern





Aufgabe 4: M5Stack

Aufgabe 4.1

Der M5Stack wurde zuerst nicht von PlatformIO als Port erkannt. Um dieses Problem zu lösen musste der entsprechende Treiber neu installiert werden

➔ siehe Schritt 2 auf: https://docs.m5stack.com/#/en/quick_start/m5core/m5stack_core_get_started_Arduino_Windows

Ist dies erledigt, dann einfach nur ein neues Projekt auf PlatformIO erstellen, das passende Board auswählen (in meinem Fall ist das ein **M5Stack Fire**) und die zugehörige Library (M5Stack) für das Projekt installieren. Der Code für das HelloWorld Programm war ja bereits gegeben, diesen einfach nur auf den M5Stack laden (per USB Kabel mit Laptop verbinden) und dann sollte folgendes erscheinen:

➔ Anmerkung: der M5Stack kann mit einem Doppelklick auf die „An-/Aus- Taste“ wieder ausgeschalten werden. Bei nur einem Klick wird er nur resettet.

Aufgabe 4.2

Benötigte Funktionen:

- *M5.Lcd.setTextColor(YELLOW)*; ab diesem Aufruf ist die Schriftfarbe gelb
- *M5.Lcd.clear(BLACK)*; Display clearen
- *M5.Lcd.setTextSize(2);* ab diesem Aufruf ist die Textgröße 2
- *M5.Lcd.println("Button example");* Text auf dem Display anzeigen
- *M5.Lcd.setCursor(3, 35);* Position des Cursors und damit der nächsten Elemente setzen (absolute Werte, d.h. man kann einen Text auch über den anderen schreiben wenn der Cursor auf dieselbe Position gesetzt wird)
  - ➔ 3 = x-Wert, 35 = y-Wert
  - ➔ Negative Werte ebenfalls möglich
- Button Funktionen: https://github.com/m5stack/m5-docs/blob/master/docs/en/api/button.md

Probleme:

- Ansatz 1: Wenn man mit delay(1) arbeitet, kann es sein, dass die Zeit nicht ganz korrekt ist/ dass eine „echte Sekunde“ eigentlich schneller vergeht, als eine Sekunde der Stoppuhr, da man quasi 1ms das Programm pausiert und hinzu kommt dann noch die Zeit, die das Programm an sich braucht, um die Befehle auszuführen. Angenommen das Programm laggt also mal/ macht irgendwelche aufwendigen Berechnungen, dann entspricht die Zeit auf der Stoppuhr nicht der tatsächlichen Zeit
  - ➔ Hat denk ich mal nur einen sehr sehr minimalen Einfluss, allerdings habe ich mich trotzdem für den anderen Ansatz entschieden
- Ansatz 2: in jedem Durchlauf die vergangene Zeit zur letzten loop-Iteration messen und die Differenz auf die Zeit der Stoppuhr addieren. Angenommen mit diesem Ansatz stoppt/ hängt das Programm mal für 5ms, dann werden halt die 5ms auf der Anzeige der Stoppuhr übersprungen, allerdings entspricht die gemessene Zeit in der nächsten Iteration dann wieder der tatsächlichen Zeit. Mit dem anderen Ansatz wäre die gestoppte Zeit dann um 5ms falsch.

- Anfangs hat die Zeit-Anzeige immer geflimmert, da ich bei jedem Update der Anzeige, das Display gecleared (*M5.Lcd.clear()*) habe.

  - ➔ Konnte ich leider nicht auf einem Video festhalten, da durch die Framerate der Kamera dieses Flimmern nicht aufgenommen werden konnte

  Lösung des Problems: nicht immer das ganze Display clearen, sondern einfach den Text „überschreiben“. Am besten funktioniert das, wenn man den Text mit einer Hintergrundfarbe belegt, sodass im nächsten Display update einfach nur der Text durch das gefüllte Rechteck überschrieben wird. Also anstatt nur die Textfarbe zu setzen mit

  M5.Lcd.setTextColor(WHITE);

  muss einfach nur eine Hintergrundfarbe mit festlegt werden:

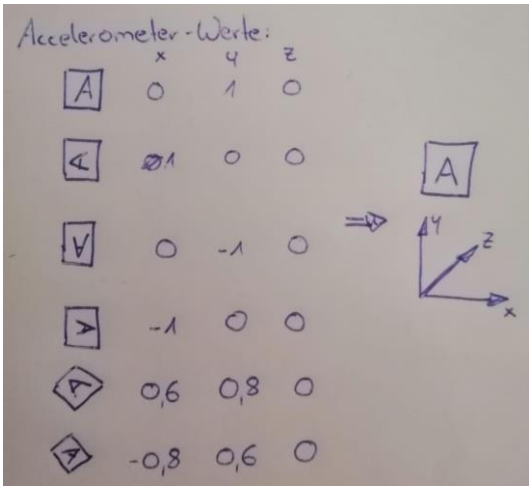
  M5.Lcd.setTextColor(WHITE, BLACK);

Aufgabe 4.3: Wasserwaage

- Infos zum eingebauten Accelerometer: https://github.com/m5stack/m5-docs/blob/master/docs/en/api/imu.md

Vorgehen:

- Zuerst habe ich versucht, einen Kreis genau in die Mitte des Displays zu platzieren. Dazu kann man mithilfe von *M5.Lcd.height()/width()* die Breite und Höhe des Displays bestimmen (in meinem Fall 320*240px) bestimmen und dann den Mittelpunkt des Displays bestimmen (indem man durch 2 teilt), an welchem man anschließend mit *M5.Lcd.fillCircle(MIDDLE_X, MIDDLE_Y, 10, WHITE)* den Kreis zeichnen kann.
- Anschließend habe ich versucht, die Werte des eingebauten Accelerometers zu lesen. Hier gab es erstmal das Problem „M5 has no class ‚IMU'", wenn ich versucht habe per *M5.IMU.Init();* auf die Inertial Measurement Unit (IMU) zuzugreifen. Das Problem ist behoben, indem man *#define M5STACK_MPU6886* noch vor *#include <M5Stack.h>* einfügt.
- Nachdem nun auf die IMU zugegriffen und die Accelerometer-Daten per *M5.IMU.getAccelData(&accX, &accY, &accZ);* ausgelesen werden können, habe ich mir erstmal einen Überblick verschafft, welche Achsenwerte sich wie verändern, wenn man den M5 bewegt:



- Wenn der M5 gerade (oder auf dem Kopf) steht, ist der x-Wert = 0, wenn er 90° steht, dann ist der Wert 1 bzw. -1. Um eine Wasserwaage zu implementieren reicht uns also der Wert der x-Achse aus.
- Unser Display ist insgesamt 320 Pixel breit, d.h. (ausgehend davon, dass sich der Kreis bei gerader Oberfläche zentral befindet) wir können uns in beide Richtungen um 160 Pixel bewegen.
- Die x-Werte reichen von -1 (wenn nach rechts gekippt) bis 1 (wenn nach links gekippt) (auch höhere Werte möglich, wenn das Gerät schnell bewegt wird, aber das ignoriere ich, da beim Messen mit der Wasserwaage das Gerät sehr still halten). Das heißt, für alle 0.01 Einheiten für den x-Wert müssen wir uns um 1.6 Pixel in eine Richtung bewegen.
- Flackern des Displays konnte ich hier nicht ganz verhindern, allerdings deutlich verbessern, indem ich nicht in jedem Loop-Durchlauf wieder die clear() Methode aufrufe (und so das ganze Display update), sondern immer den alten Kreis mit der Hintergrundfarbe „übermalen" lasse und dann den neuen Kreis zeichne.
- Steht der M5Stack auf einer geraden Oberfläche, d.h. der x-Wert des Acc ist 0, dann wird der Kreis eingefärbt.
  ➔ Ich habe hier zusätzlich eine „Deadzone" von 2 Pixeln in jede Richtung mit eingebaut, d.h. wenn sich der Kreis um +- 2 Pixel von der Mitte des Displays befindet, wird er

trotzdem noch eingefärbt. Das habe ich gemacht, weil die Messwerte für die x-Achse immer minimal schwanken und somit der Kreis immer zwischen weiß und rot umherspringt. So ist das ganze etwas „smoother“.