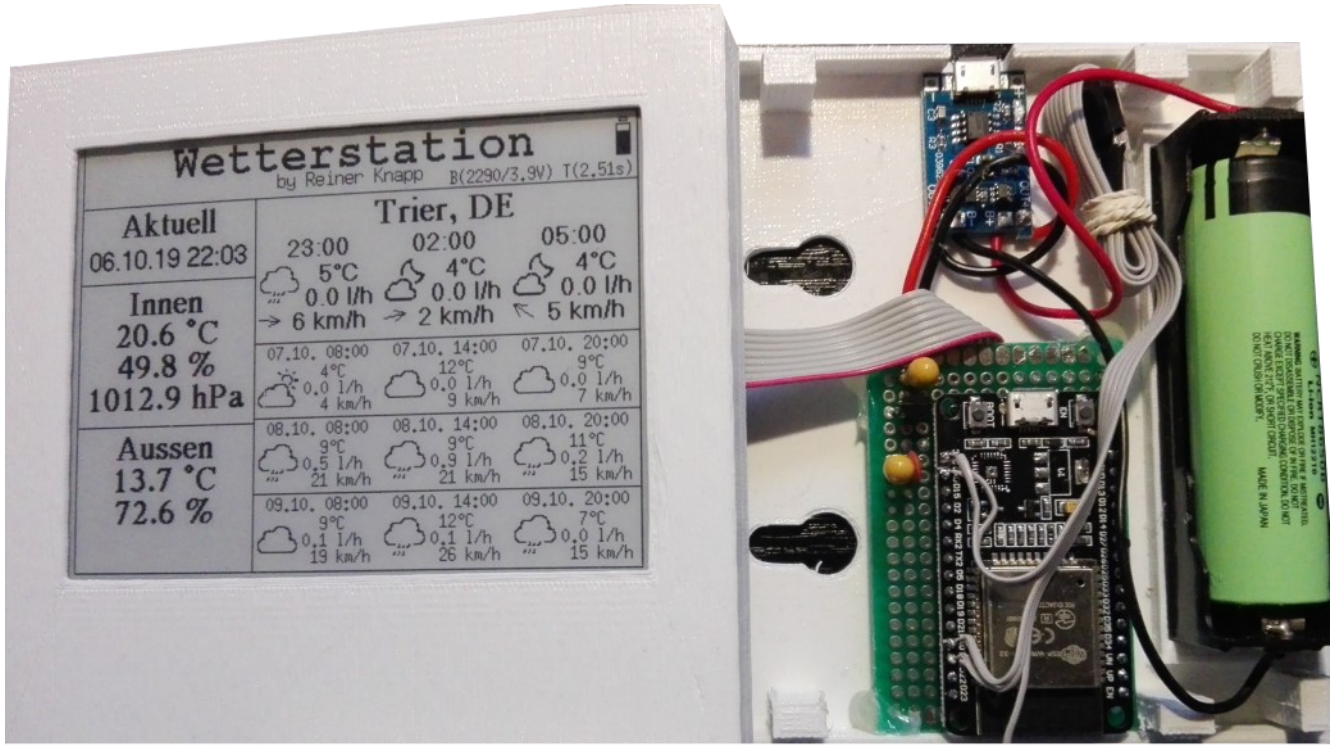
Baubericht einer Wetterstation



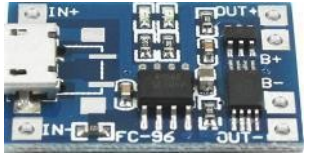
Reiner Knapp

PiAndMore 12

16. November 2019

Die Hardware

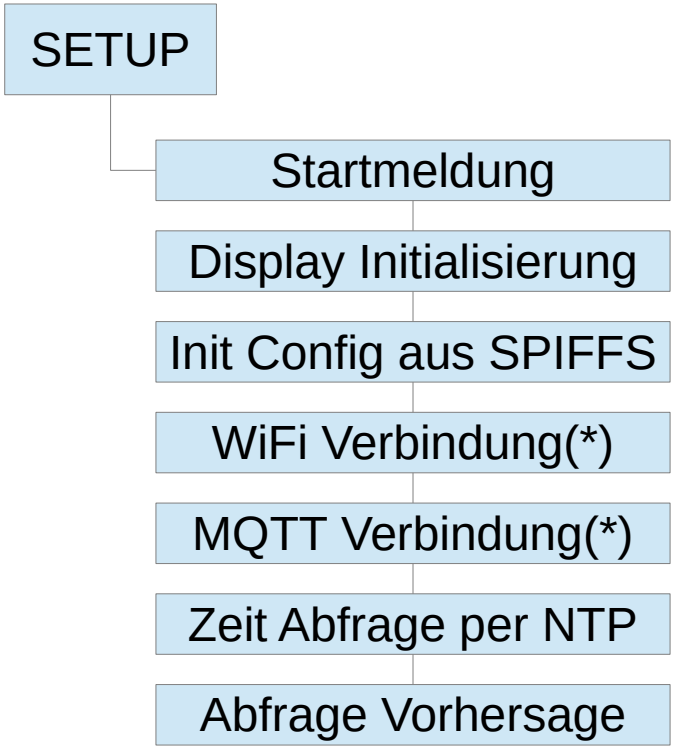
- ESP32 DevKit Board (WLAN, 520 KB RAM)
- Panasonic NCR18650 Li-Ion Akku
  (3,6V / 3400 mAh => 12,2Wh)
- TP4056 LiIon Ladeboard mit Schutzfunktionen
- MSP1700 Spannungsregler 3.3 V
  (DropOut: <350mV; avg. 250mA; max. 550mA)
- WaveShare 4,3“ E-Ink Display (400x300px)
- Gehäuse aus dem 3D Drucker

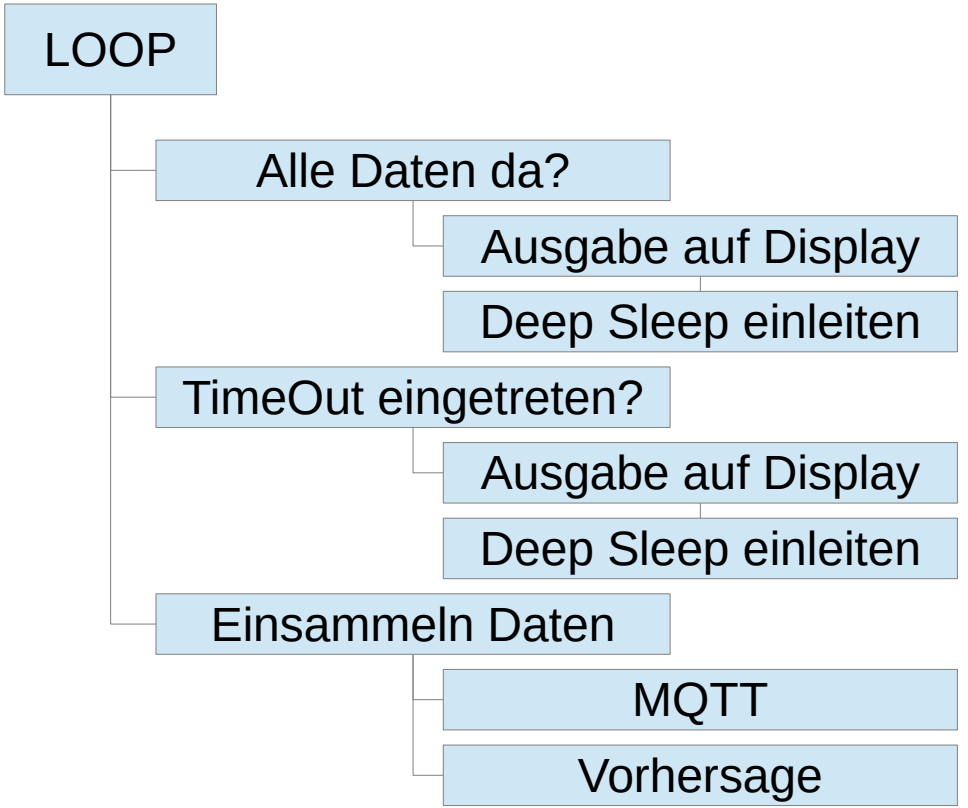


Die Quellen

- Arduino IDE (1.8.9) als Programmierumgebung
  Standard Arduino Libraries für WIFI, SPIFFS… zusätzlich:
  MQTT: PubSubClient
  E-Ink Display: GxEPD2 und U8g2_for_Adafruit_GFX
- Wetter Vorhersage Daten von OpenWeatherMap
  URL: https://openweathermap.org/
- Wetter Symbole von https://peter.build/weather-underground-icons/
- Aktuelle Werte von lokalem MQTT Server
- Grundlagen Batterie Kapazität (Dave Jones, EEVblog #140):
  https://www.youtube.com/watch?v=R8hTQXqURB4

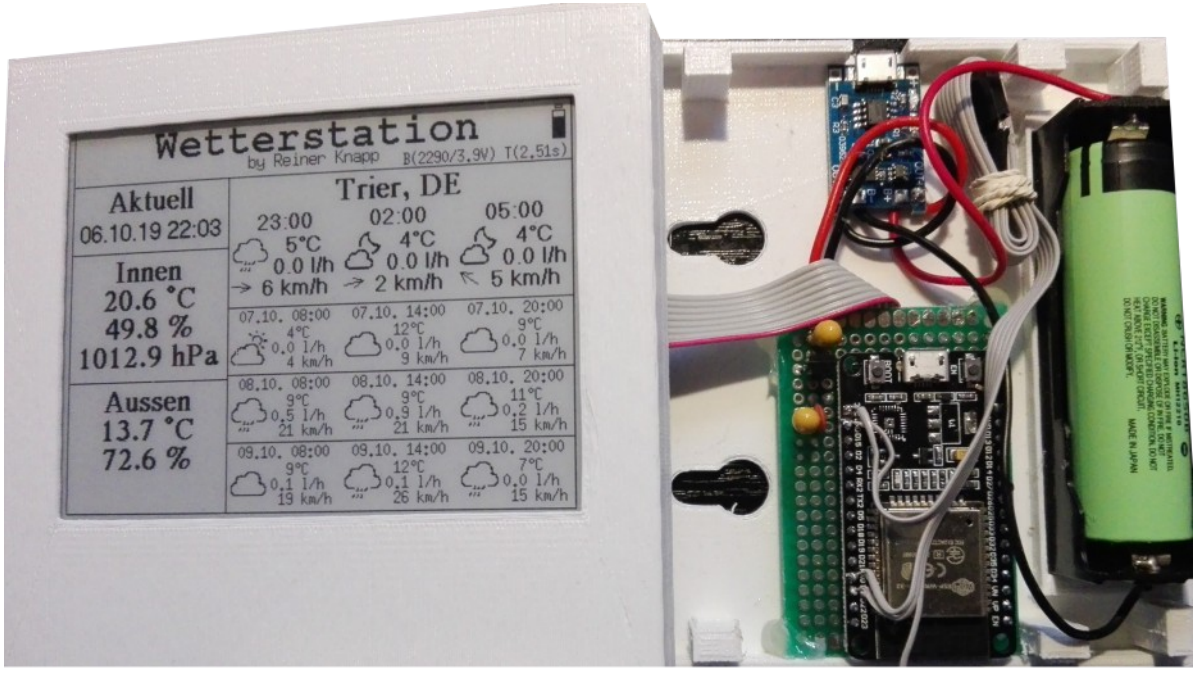
Ablaufplan der Software





(*) Timeout Vorkehrungen treffen!

Das Ergebnis



Alles gut, oder?

Wie hoch ist eigentlich der Stromverbrauch?

„Lust auf etwas Realität?“ fragte die Vernunft

„Sicher!“ antwortete die Torheit

„Nimm Schnaps mit!“ sagte die Erfahrung

Wie misst man Strom?

Multimeter:

- ✔ Günstig
- ✗ Träge Messung / ungeeignet für dynamische Lasten
- ✗ Ungenaue Strommessung bei sehr kleinen Werten

Oszilloskop mit Shunt Widerstand:

- ✔ Aufzeichnung des Spannungsabfalls am Shunt Widerstand
- ✔ Integral Funktion ermittelt Stromverbrauch aus Spannungsverlauf
- ✔ Geeignet für dynamische Lasten
- ✗ Teurer in der Anschaffung

Wie misst man Strom?

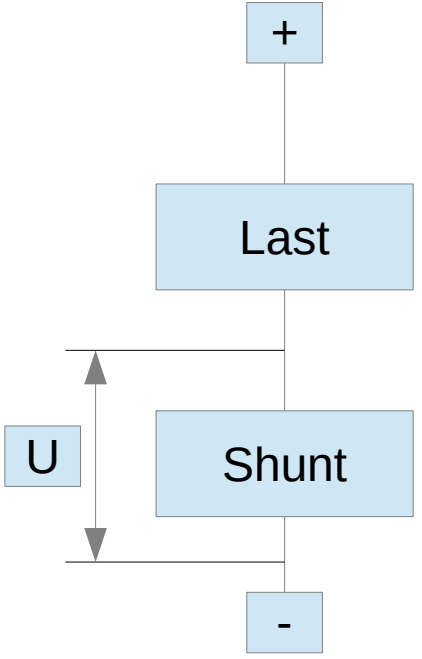
Ohmsches Gesetz: R = U/I

Gegeben:

- Shuntwiderstand: 0,5 Ω
- Spannungsabfall am Shunt durch Messung

Stromstärke ergibt sich durch: I = U/R

Beispiel: U = 50mV => I = 0,05 V/0,5 Ω = 0,1 A



Stromverbrauch im Deep Sleep

- ESP32 (laut Datenblatt)
  Deep-Sleep: 10 uA
  Hibernation: 5 uA
- WaveShare E-Ink Display (laut Datenblatt):
  Deep Sleep: 5 uA
- TP4056 Li-Ion Ladeboard (laut Datenblatt):
  TP4056 Standby Mode: 2,5 - 6 uA
  DW01 (Batterie Schutz): 6 uA
- MCP1700 Spannungsregler (laut Datenblatt):
  Input Quiescent Current: 1,6 - 4 uA
- Diverses (z.B. Widerstände; Leistungsverlust Spannungsreglung): unbekannt

Zu erwartender Stromverbrauch: ca. 26 uA + X (Diverses)

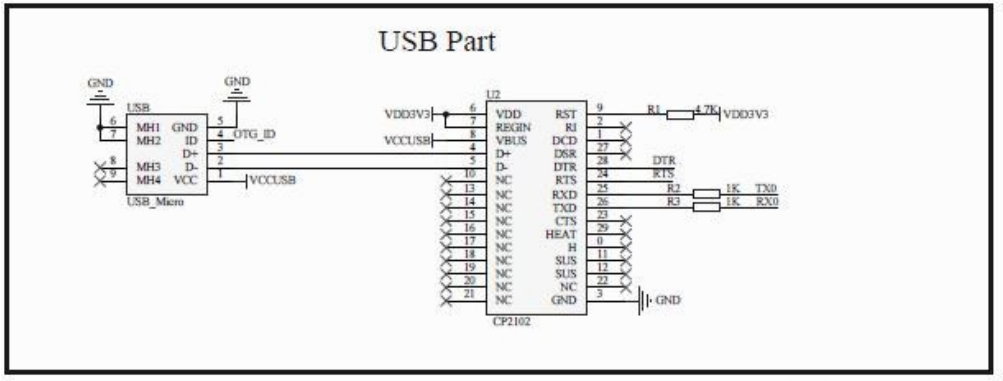
Tatsächlich gemessen: 11,8 mA => **WTF$_{(1)}$ => nur ca. 12 Tage Laufzeit!**

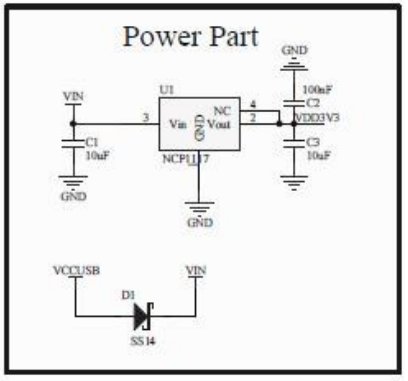
Stromverbrauch im Deep Sleep

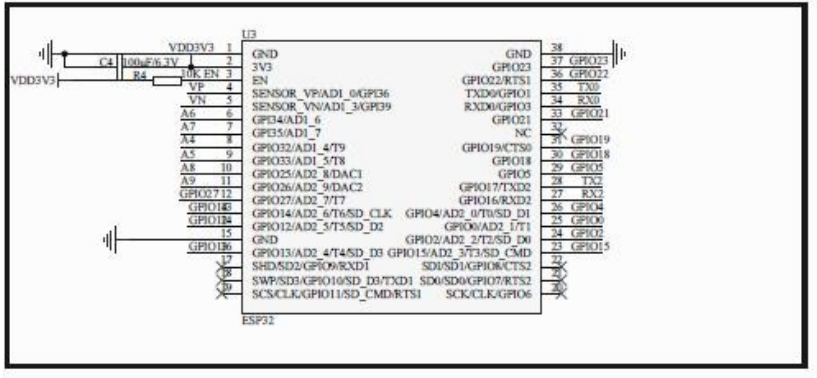
Google Suche:
„ESP32 DEVKITV1 Schematics“

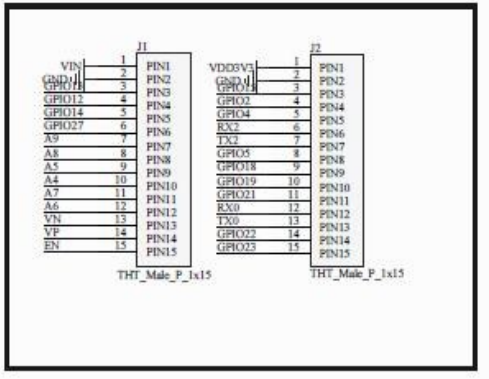
https://github.com/playelek/pinout-doit-32devkitv1

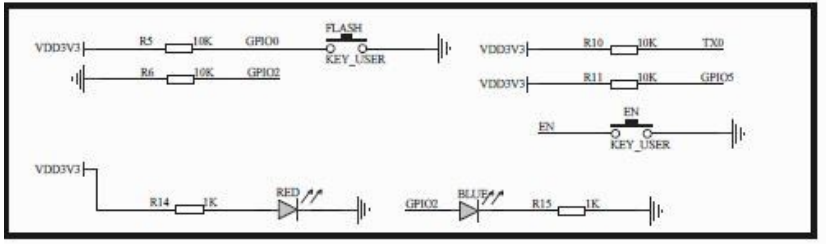
- **USB Part:**
  USB Mikro Buchse
  CP2102 USB-to-UART Bridge
- **Power Part:**
  AMS1117 3.3V Regulator
  Schutzdiode (Polarität)
- **ESP32 Part**
  Pull-Up EN-Pin
- **Sonstiges**
  Anschluss-Pinleisten
  Div. Pull Widerstände
  Div. Kondensatoren
  Power On LED (VDD, rot)
  Indikator LED (GPIO2, blau)
  Reset Beschaltung CP2102

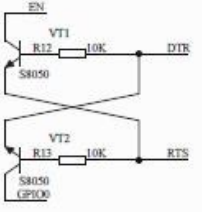








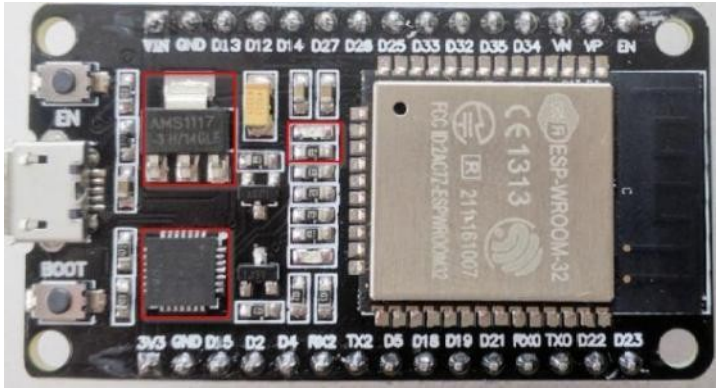


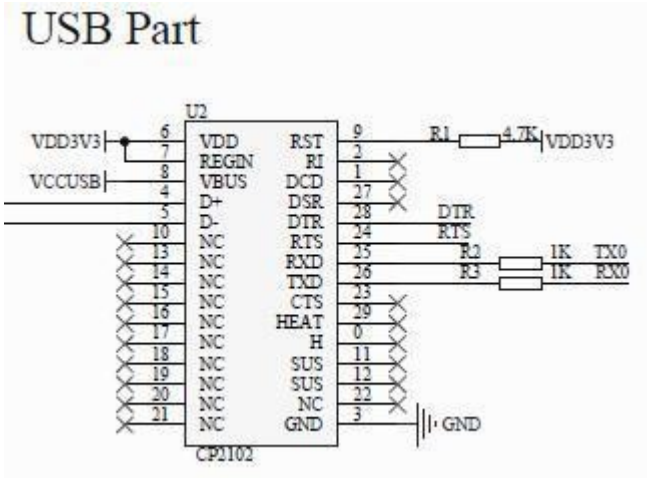


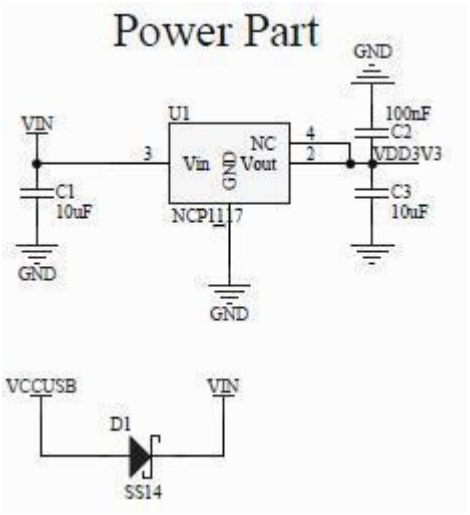
Stromverbrauch im Deep Sleep

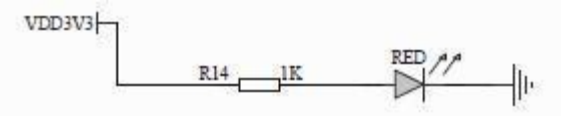
Woher kommt der hohe Deep Sleep Verbrauch des ESP32 DevKit Boards?

- Ein Blick in den Schaltplan





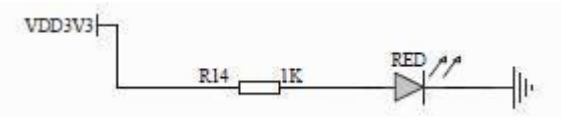




Stromverbrauch im Deep Sleep

Die Auswirkungen der zusätzlichen Komponenten

Die rote LED zur Anzeige der Versorgungsspannung:



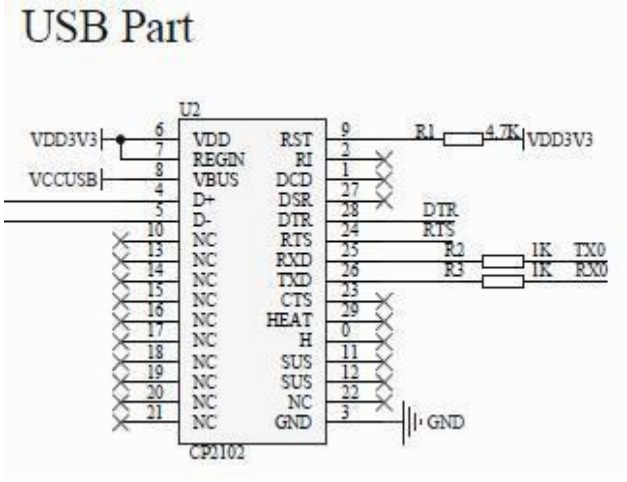
I = U/R (U=3,3V-1,9V=1,4V) = 1,4 mA

Der Spannungsregulator AMS1117:

Quiescent Current	AMS1117-1.5/-1.8/-2.5/-2.85/-3.3/-5.0	$(V_{IN} - V_{OUT}) = 1.5V$		**5**	**11**	mA

Stromverbrauch im Deep Sleep

Die Auswirkungen der zusätzlichen Komponenten



CP2102 USB-to-UART Bridge:

Verbrauch im Suspend (Datenblatt): 330 uA (!)

=> <1 mA gemessen …

... mit vorheriger USB Host Verbindung

=> Aber 5 mA ohne USB Verbindung => **WTF$_{(2)}$???**

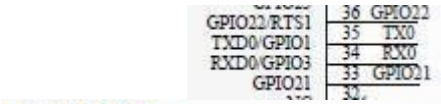
- Suspend setzt offenbar einmalige USB Verbindung zum Host voraus
- Software Upload über FTDI Adapter => CP2102 ausgelötet

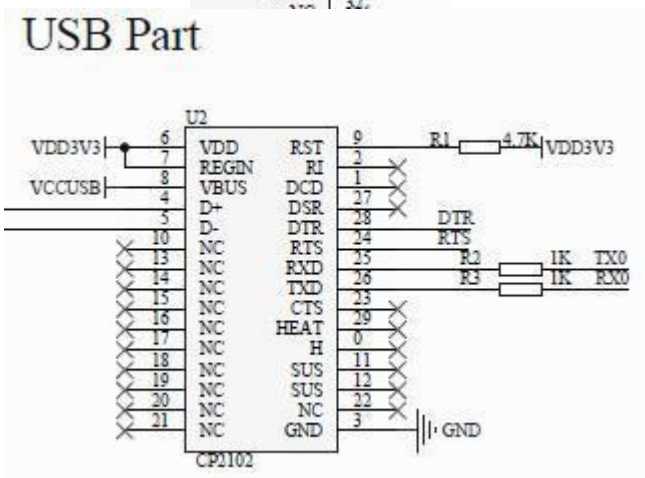
Verbrauch im Deep Sleep nun gemessen: **ca. 40 mA** => **WTF$_{(3)}$???**

Stromverbrauch im Deep Sleep

Die „Nebenwirkung“ der fehlenden CP2102 USB-to-UART Bridge:

(Sehr) kurzer Auszug aus dem Datenblatt:
„… with integrated … pull-up resistors“





- Offener RXD Pin am ESP32 verhindert den Deep Sleep, manchmal!
  - ➔ Lösung 1: Pull-Up Widerstand einlöten
  - ➔ Lösung 2: UART per Software deaktivieren

```
#include "soc/periph_defs.h"
#include "driver/periph_ctrl.h"
periph_module_disable(PERIPH_UART0_MODULE);
```

- FunFact am Rande: Dem TXD Pin wurde ein PullUp spendiert :-)

Stromverbrauch im Deep Sleep

Stromverbrauch Deep Sleep gemessen mit Oszilloskop:

- Shunt: 100Ω
  Spannung: 6,8mV
  => I = U/R
  => I = 0,0068/100
  => I = 0,000068A
  => **I = 68uA**

Stromverbrauch bei Aktivität

Stromverbrauch gemessen mit Oszilloskop (erste Software Version):

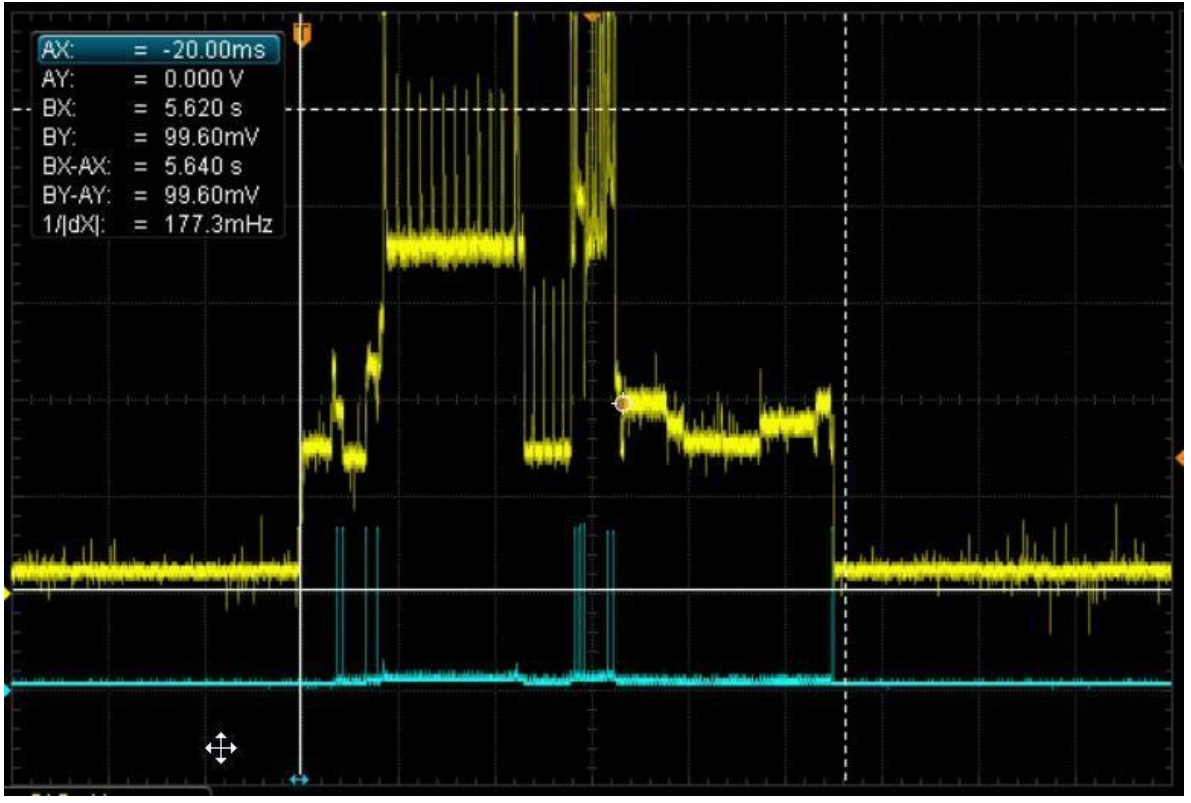
- 684mVs / 0,5Ω
  => 1368 mAs
- 1368mAs/3600
  => 0,38 mAh
- Laufzeit: 99 Tage

Geht da noch was?

Stromverbrauch bei Aktivität

Identifizieren der Programmabschnitte durch ein Signalflag (240MHz)

PreSETUP (Arduino)	**400ms**
Serielle Ausgabe	<50ms
Display Init	**240ms**
SPIFFS und Config	110ms
WiFi Verbindung	2180ms
NTP Zeit abfragen	**<50ms**
MQTT Connect	**<50ms**
Abfrage Vorhersage	**240ms**
Ende SETUP/LOOP	**50ms**
Displayausgabe	2270ms
Gesamt	5640ms



Stromverbrauch bei Aktivität

Vergleich der Zeiten bei verschiedenen ESP32 Geschwindigkeiten

240 MHz

PreSETUP (Arduino)	**400ms**
Serielle Ausgabe	<50ms
Display Init	**240ms**
SPIFFS und Config	110ms
WiFi Verbindung (DHCP)	2180ms
NTP Zeit abfragen	**<50ms**
MQTT Connect	**<50ms**
Abfrage Vorhersage	**240ms**
Ende SETUP/LOOP	**50ms**
Displayausgabe	2270ms
Gesamt	5640ms

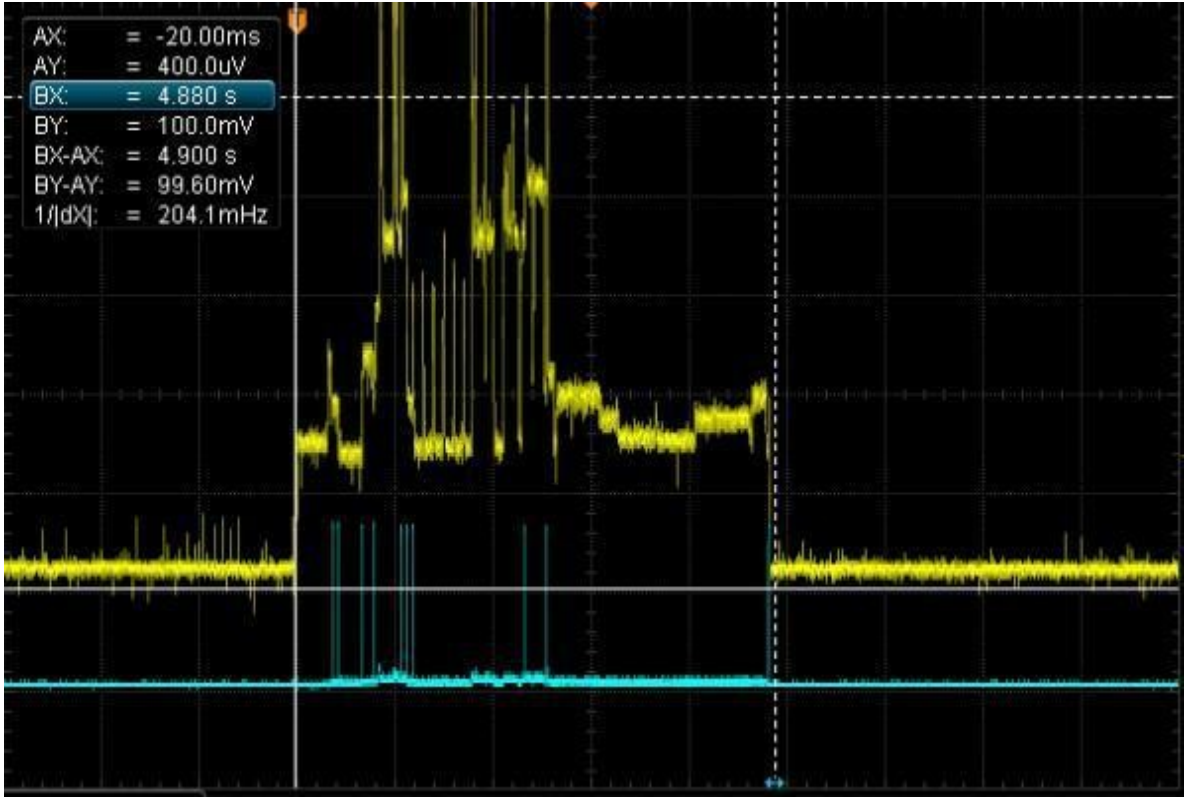
80 MHz

PreSETUP (Arduino)	**450ms**
Serielle Ausgabe	<50ms
Display Init	**240ms**
SPIFFS und Config	310ms
WiFi Verbindung (DHCP)	2090ms
NTP Zeit abfragen	**<50ms**
MQTT Connect	**<50ms**
Abfrage Vorhersage	**380ms**
Ende SETUP/LOOP	**370ms**
Displayausgabe	3220ms
Gesamt	7210ms

Stromverbrauch bei Aktivität

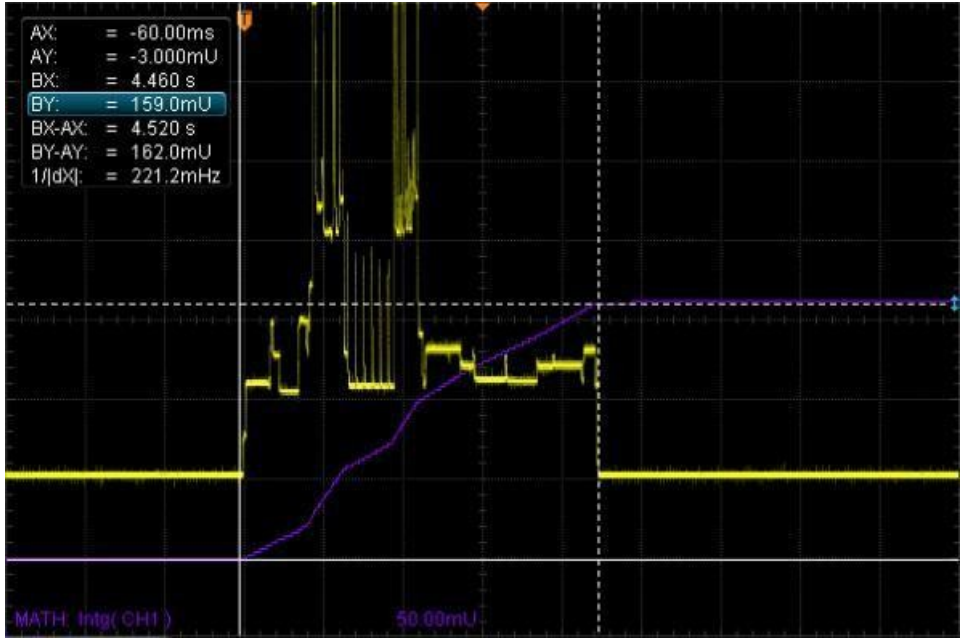
Nach Optimierung der Programmabschnitte

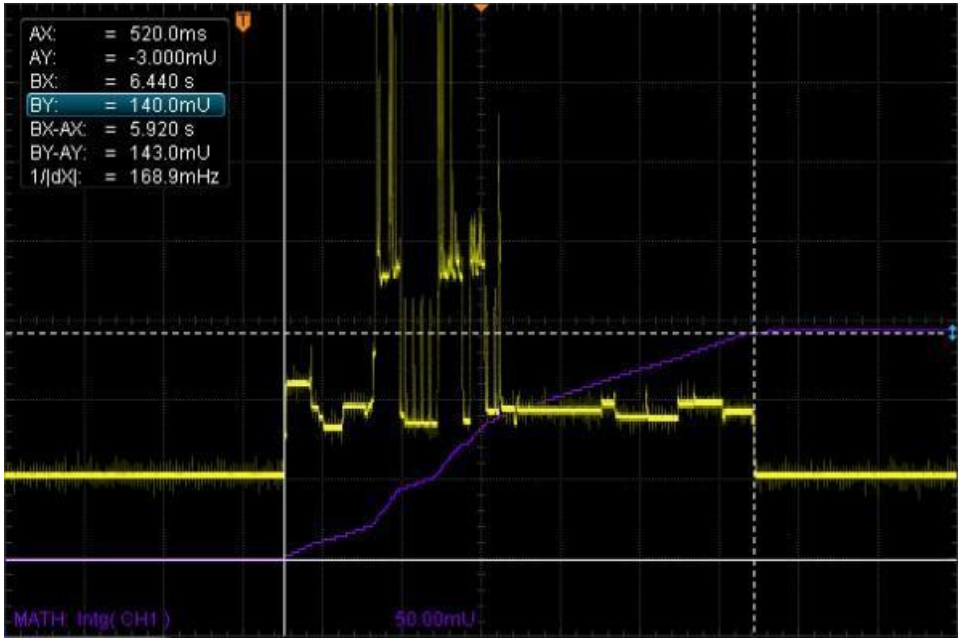
PreSETUP (Arduino)	**450ms**
Serielle Ausgabe	<50ms
Display Init	**240ms**
SPIFFS und Config	110ms
WiFi Verbindung (IP)	390ms
NTP Zeit abfragen	**<50ms**
MQTT Connect	**<50ms**
Abfrage Vorhersage	**1140ms**
Ende SETUP/LOOP	**200ms**
Displayausgabe	2270ms
Gesamt	4900ms



Stromverbrauch bei Aktivität

240MHz vs. 80MHz





324 mAs (0,09 mAh) => ca. 360 Tage 286 mAs (0,08 mAh) => ca. 400 Tage

=> 80MHz empfehlenswert, wegen ungewisser Dauer externer Abrufe

Reduzierung des Stromverbrauch

Tipps zur Hardware

- Mikrocontroller passend zur Anwendung aussuchen!
  Was brauche ich, und was nicht?
- Nicht nur der Mikrocontroller benötigt Strom (z.B. TP4056, CP2102, AMS1117)
  Datenblätter ansehen und Schaltung verstehen!
- Alles, was leuchtet, blinkt und piept braucht Strom! => Vermeiden

Tipps zur Software

- Unnötige Ausgaben und Delays vermeiden.
- ESP32/8266 WiFi Verbindung mit fester IP Adresse geht schneller
- Bei Datenabruf aus Internet schnellen DNS Server benutzen
- Bei ungewissen externen Wartezeiten lieber „langsam“ warten lassen
- Endlosschleifen vermeiden durch definierte Timeouts (MQTT, WIFI, Web-Daten)
- Dinge, auf die im „Hintergrund“ gewartet werden kann, zuerst anstoßen (z.B. MQTT)

Baubericht einer Wetterstation

Fragen?

Vielen Dank für die Aufmerksamkeit!