

# Report Wetterstation mit einem Einplatinenrechner

Oliver Siegemund

Dezember 2023

Dokumentation der Entwicklung und Auswertung einer Wetterstation basierend auf dem daisy seed Einplatinenrechnern und analogen Sensoren.

# Contents

<b>1 Einleitung</b>	<b>3</b>
1.1 Aufgabenstellung . . . . .	3
1.2 Hardware . . . . .	4
1.2.1 Einplatinenrechner . . . . .	4
1.2.2 Sensoren . . . . .	6
1.2.3 Mess-Schaltung . . . . .	6
1.3 Software . . . . .	7
1.3.1 libDaisy . . . . .	7
<b>A Quell-Code</b>	<b>8</b>
A.1 mcu . . . . .	8
A.2 cpu . . . . .	9

# Chapter 1

## Einleitung

### 1.1 Aufgabenstellung

Du möchtest eine eigene Wetterstation bauen. Hierzu nutzt du einen Einplatinenrechner (z.B. Arduino) sowie zugehörige Sensoren.

- (a) Lege eine Schaltung zur Erfassung der beiden Messgrößen Temperatur und Luftfeuchtigkeit aus und setze diese als Hardware um.
- (b) Nehme mindestens 5 Messreihen bei unterschiedlichen Temperaturen und Luftfeuchtigkeitswerten auf (z.B. in verschiedenen Räumen oder Außenbereichen). Für jede Messreihe nehme dabei mindestens jeweils 100 Werte der beiden Messgrößen als Wiederholmessung auf.
- (c) Beschreibe allgemein die Quantisierungsabweichung der Datensätze, die mithilfe des Einplatinenrechners aufgenommen wurden und stelle diese als Funktion der Bit-Zahl dar. Ermittle diese Abweichung für deinen konkreten Anwendungsfall.
- (d) Ermittle die Messunsicherheit der Ausgangsgrößen Temperatur und Luftfeuchtigkeit auf drei verschiedene Arten – gebe als Ergebnis jeweils ein 95
  - (i) Auf Basis der statistischen Analyse Ihrer Messreihen aus Teil b).
  - (ii) Auf Basis der Informationen der Datenblätter der beiden Sensoren (a priori Wissen). Nutze die im Datenblatt vorhandene Information, um eine Bilanz der Messunsicherheit unter Nutzung des Guide to the Expression of Uncertainty in Measurement (GUM) sowie der hierin vorgesehenen Abweichungsfortpflanzung abzuleiten.
  - (iii) Über eine Messung der Spannung. Nutze deinen Einplatinenrechner um Messreihen der Spannung des Sensors sowie der zugehörigen Temperatur und Luftfeuchtigkeit aufzunehmen. Stelle auf Basis der in c)

ermittelten Quantisierungsabweichung und anderer Einflüsse der Unsicherheit der Spannungsmessung eine Bilanz der Unsicherheit unter Nutzung des Guide to the Expression of Uncertainty in Measurement (GUM) sowie der hierin vorgesehenen Abweichungsfortpflanzung für die Ausgangsgrößen Temperatur und Luftfeuchtigkeit auf. Stelle dabei das mathematische Modell der Messung als Taylor-Reihe zweiter Ordnung dar. Interpretiere und vergleiche das Ergebnis auf Basis der drei Methoden. Diskutiere die jeweiligen Vorteile und Nachteile der drei Methoden auf Basis der Ergebnisse.

Hinweise: Die Messung der Spannung kann entweder zeitgleich mit der Aufnahme der Ergebnisse von Temperatur und Spannung in Teil b) durchgeführt werden oder durch eine separate Messreihe. Die Unsicherheitsbilanzierung und Systemmodellierung müssen durch Implementierung als Programm-Code erfolgen. Die Aufgabenteile i)-iii) können unabhängig voneinander gelöst werden.

- (e) Modellierte den Wärmeübergang auf den Temperatur-Sensor aus der Umgebungsluft mithilfe einer Differentialgleichung 1. Ordnung und stelle das zeitliche Verhalten des Sensorsystems auf Basis der Sprungantwort und Impulsantwort eines PT1-Systems für jeweils ein selbst gewähltes Szenario dar. Diskutiere, welche Konsequenzen sich für die Wetterstation ergeben. Hinweise zur Thermodynamik: Der Widerstand kann als perfekter Zylinder angenommen werden. Von Interesse ist die Temperatur in seinem Kern. Der Wärmestrom  $q(t)$  ist nach dem Fourier'schen Gesetz definiert als:

$$q(t) = \lambda \frac{A}{r} (\vartheta_u(t) - \vartheta_r(t))$$

Hierbei bezeichnen  $\vartheta_u(t)$  die Umgebungstemperatur,  $\vartheta_r(t)$  die Kerntemperatur des Widerstands,  $A$  seine Außenfläche,  $r$  seinen Radius und  $\lambda$  die materialabhängige Wärmeleitfähigkeit.

Die Temperatur des Widerstands sowie der Wärmestrom hängen wie folgt zusammen:

$$\frac{1}{c_{p,R}} \int q(t) dt = \vartheta_r(t)$$

$c_{p,R}$  ist die spezifische Wärmekapazität und hängt vom Material des Widerstands ab.

Die beiden Gleichungen können zu einer DGL 1. Ordnung für  $\vartheta_r(t)$  kombiniert werden.

## 1.2 Hardware

### 1.2.1 Einplatinenrechner

Fuer die Aufgabenstellung wurde der daisy seed rev7 gewählt. Neben der Tatsache das das Board verfügbare ist, hat es sowie der verbaute

# DAISY PINOUT

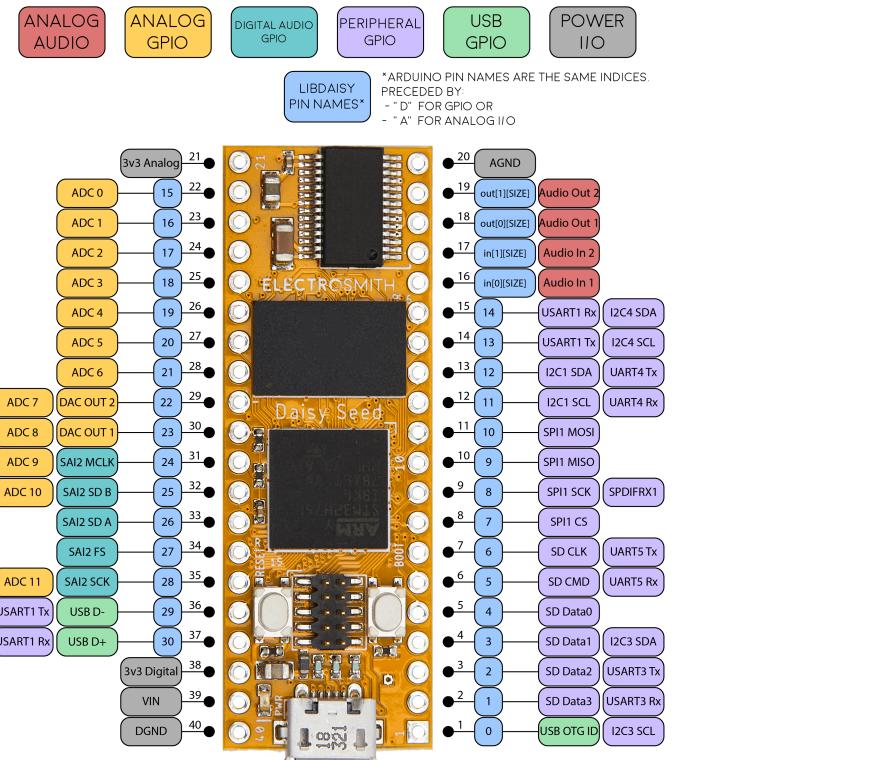


Figure 1.1: daisy seed - Pinout

STM32H750IB sehr gute Eigenschaften [3] um Analoge Signale zu erfassen.

Eigenschaften	
Analog Digital Konverter	3 Wandler Kanäle mit bis zu 18bit Auflösung auf 11 Pins
Digital Analog Konverter	2 12-bit Digital / Analog Wandler mit bis zu 1 MHz
FPU	FPU mit doppelter Präzision
RAM	64 MB SDRAM um Messwerte zu speichern (≈ 2 Mega-Samples )

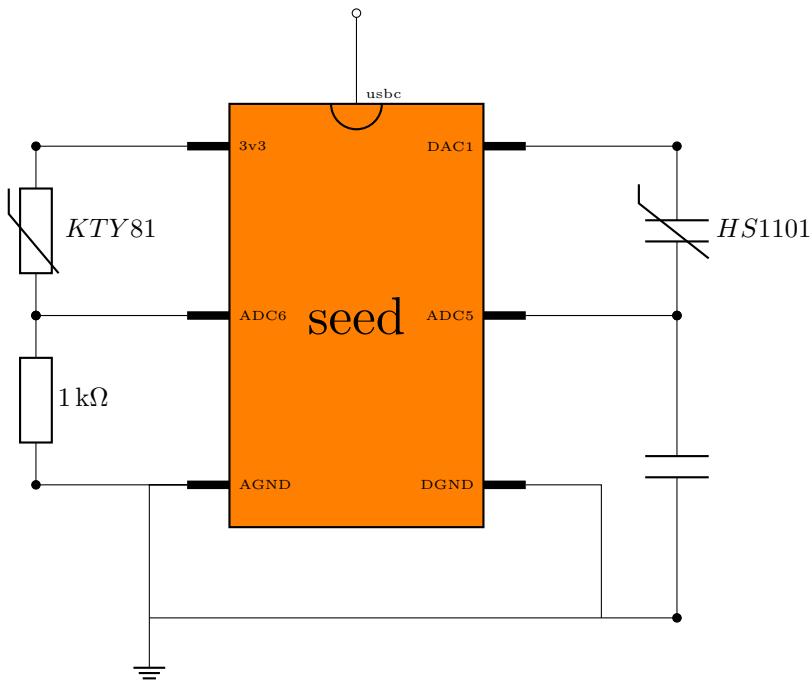
## 1.2.2 Sensoren

### Temperatur

Zur Temperaturmessung wurde ein Termistor gewählt. Genauer der KTY 81 122m mit einem Temperatur Koeffizienten  $T_C$  von 0.79 und einem Widerstand bei  $T_a = 25^\circ C$  zwischen 1000 und 1020  $\Omega$  ausweist, laut Hersteller Datenblatt. Ausgelegt ist der Sensor für einen Temperaturbereich zwischen -55 °C und +150 °C.

### Luftfeuchte

## 1.2.3 Mess-Schaltung



### Wechselspannungs Generierung

Um die unbekannte Kapazität des Luftfeuchtesensors zu messen muss eine Wechselspannung generiert werden. Dazu wird einer der Digital Analog Generierungs Kanäle des daisy seeds verwendet und eine Rechtecksignal mit  $\hat{u} \approx 1V$  und  $T \approx 100\mu s$ . Diese Werte Orientieren sich an den Referenzwerten des Handbuchs.

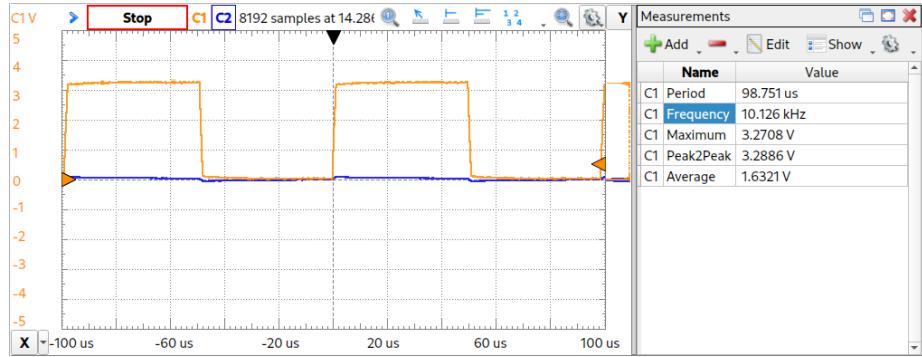


Figure 1.2: Wechselspannung

## 1.3 Software

Alle diese Aufgabe betreffender Quellcode kann unter [https://github.com/Komplementariteten/dlbaetem01\\_wetterstation](https://github.com/Komplementariteten/dlbaetem01_wetterstation) eingesehen werden. Das betrifft auch, in dem Bericht enthaltene Quellen.

### 1.3.1 libDaisy

libDaisy is a c++ api, provided by the daisy seed manufacturer Electro Smith to support common task, and provide a Unix Makefile and Visual Studio Code based Project Template. In the here used implementation a fork (<https://github.com/Komplementariteten/DaisyExamples>) of the related Examples repository is used.

# Appendix A

## Quell-Code

### A.1 mcu

```
#include <pstl/glue_numeric_defs.h>

#include "daisy_seed.h"
#include "daisysp.h"
#include <string>

#define NUMBER_OF_VALUES 1024
#define TEST_MEMORY_SIZE (NUMBER_OF_VALUES * 16)

using namespace daisy;
using namespace daisysp;

DaisySeed hw;
static size_t value_count = 0;
// 16bit adc
const float_t ADC_MAX = 0xFFFF;
const float_t VEE = 3.3;
const float_t R_1 = 1000;

void store_value(uint16_t value){
    int32_t *ram = (int32_t *)0xC0000000;
    unsigned char const *p = reinterpret_cast<unsigned char const *>(&value);
    memcpy((sizeof(uint16_t) * value_count) + ram, p, sizeof(uint16_t));
    value_count++;
}

void calculate_voltages_and_temp(){
    int32_t *ram = (int32_t *)0xC0000000;
    for (size_t i = 1; i < value_count; i++)
    {
        uint16_t result;
        unsigned char const *p = (unsigned char const *)((sizeof(uint16_t) * i) + ram);
        memcpy(&result, p, sizeof(uint16_t));
        // Calculate Voltage
        float_t v = VEE * (result / ADC_MAX);
        // Calculate Resistance
```

```

        float_t t = ((VEE - v) * R_1) / v;
        hw.Println("%7d;%2.6f;%4.6f", result, v, t);
    }
    value_count = 0;
}

int main(void)
{
    hw.Init();

    /** Configure and Initialize the DAC */
    DacHandle::Config dac_cfg;
    dac_cfg.bitdepth = DacHandle::BitDepth::BITS_12;
    dac_cfg.buff_state = DacHandle::BufferState::ENABLED;
    dac_cfg.mode = DacHandle::Mode::POLLING;
    dac_cfg.chn = DacHandle::Channel::ONE;
    hw.dac.Init(dac_cfg);

    bool dac_is_high = false;

    // Wait for Serial connection
    hw.StartLog(true);
    hw.SetLed(true);
    AdcChannelConfig adcConfig;
    adcConfig.InitSingle(hw.GetPin(21));

    hw.adc.Init(&adcConfig, 1);
    hw.adc.Start();
    while(value_count < NUMBER_OF_VALUES) {
        System::DelayUs(48);
        if(dac_is_high){
            hw.dac.WriteValue(DacHandle::Channel::ONE, 0b0);
            dac_is_high = false;
        }
        else {
            // On rising Edge
            store_value(hw.adc.Get(0));
            // 4096 ~ 3.3V => 1284 ~ 1V
            hw.dac.WriteValue(DacHandle::Channel::ONE, 1284);
            dac_is_high = true;
        }
    }
    //
    System::Delay(1000);
    // done turn of the LED
    hw.SetLed(false);
    calculate_voltages_and_temp();
}

```

## A.2 cpu

```

use std::io;
use std::io::{Read, Write};
use clap::Parser;

#[derive(Parser, Debug)]
#[command(author, version, about, long_about = None)]

```

```

struct Args{
    #[arg(short, long, required = true)]
    tty: String
}

fn main() {
    let args = Args::parse();

    println!("Opening Serial device {}!", args.tty);

    let mut port = match serialport::new(&args.tty, 115200).open() {
        Err(e) => {
            eprintln!("Failed to open \"{}\". Error: {:?}", &args.tty, e);
            ::std::process::exit(1);
        }
        Ok(p) => p
    };
    let mut serial_buf: Vec<u8> = vec![0; 1000];
    loop {
        match port.read(serial_buf.as_mut_slice()) {
            Ok(t) => io::stdout().write_all(&serial_buf[..t]).unwrap(),
            Err(ref e) if e.kind() == io::ErrorKind::TimedOut => (),
            Err(e) => eprintln!("{}: {}", e),
        }
    }
    println!("Hello, world!");
}

```

# Bibliography

- [1] Pruefungsamt (2023) - Hausarbeit, Aufgabenstellung zum Kurs: DL-BAELEM01 – Elektrische Messtechnik, Internationale Hochschule.
- [2] Electrosmith (2023) - daisy seed rev7 - Datasheet, [https://static1.squarespace.com/static/58d03fdc1b10e3bf442567b8/t/63e2eb14c0085e36812cc1e0/1675815701844/Daisy\\_Seed\\_datasheet\\_v1.0.8.pdf](https://static1.squarespace.com/static/58d03fdc1b10e3bf442567b8/t/63e2eb14c0085e36812cc1e0/1675815701844/Daisy_Seed_datasheet_v1.0.8.pdf).
- [3] STM (2023) STM32H750 - Datasheet, <https://www.st.com/resource/en/datasheet/stm32h750ib.pdf>.