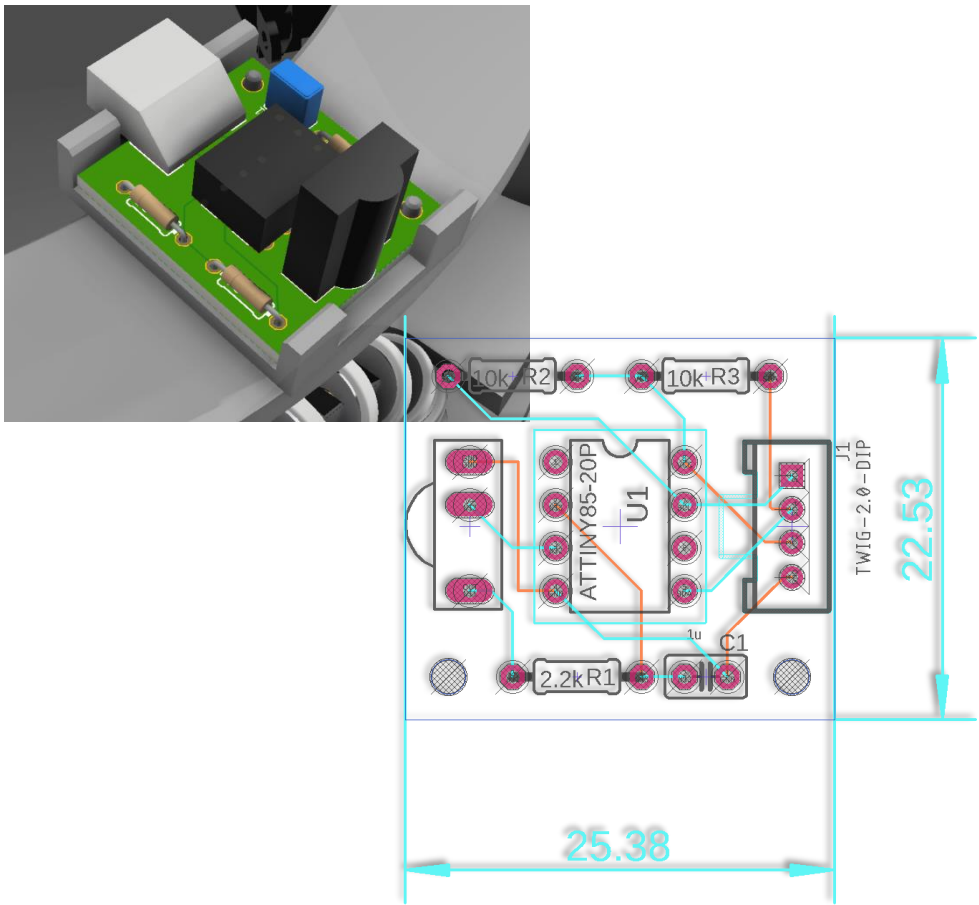
HiTechnik IR-Ball auslesen mit

ATTiny/ TSOP 31240,

Datenübertragung per I2C



Genutzt wird ein ATtiny85 *(8-pin DIP package)* mit 8MHz internem Takt.

https://www.reichelt.de/mcu-attiny-avr-risc-8-kb-20-mhz-pdip-8-attiny-85-20-pu-p69299.html?&trstct=pos_0

Abbildung 1: ATtiny85, DIL8

Pinbelegung des ATtiny85:

Die Zahlen geben die Pins der Arduino-IDE an z.B.

```
pinMode(0,OUTPUT);
analogWrite(1, 128);
analogRead(A3);
digitalWrite(4, HIGH);
```

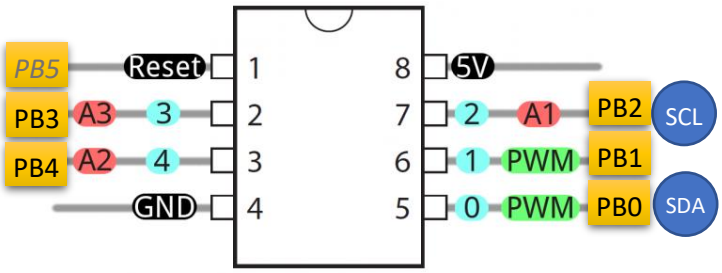


Abbildung 2: Pinbelegung ATtiny85

Der Tiny wird mit dem Sparkfun Tiny Programmer programmiert:

!!ACHTUNG: I2C kann nicht direkt auf dem Programmer getestet werden, weil die Programmer-LED auf PIN 5 / Arduino-PIN 0 liegt und so für I2C geblockt ist!

Ausführliche Installations-Infos:
https://learn.sparkfun.com/tutorials/tiny-avr-programmer-hookup-guide

Als Treiber muss im Windows-Gerätemanager installiert werden:
https://cdn.sparkfun.com/datasheets/Dev/AVR/usbtinyisp_libusb_1.2.6.0.zip
(*die „zadig.exe“ auf Sparkfun macht Probleme, besser nicht nutzen*)

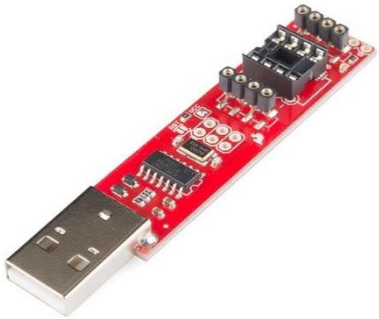


Abbildung 3: Tiny Programmer

Programmiert wird mit der Arduino-IDE. Setup wie folgt:

1. ATtiny-Boardverwalter-URL:
Im Menüpunkt **„Datei > Voreinstellungen“** wird in Zeile „ Zusätzliche Bordverwalter-URLs“ dieser Link angegeben:
https://raw.githubusercontent.com/damellis/attiny/ide-1.6.x-boards-manager/package_damellis_attiny_index.json

2. Board einstellen:
Anschließend im Menüpunkt **„Werkzeuge > Board > Boardverwalter…“** die Hardwaretreiber für den ATtiny laden. Ist dies geschehen, kann der Tiny eingestellt werden (Siehe Abbildung rechts)

3. Programmer einstellen:
Dann muss noch Programmer ausgewählt werden unter **„Werkzeuge > Programmer > USBtinyISP“**

4. Bootloader brennen:
Ab Werk ist der ATtiny auf 1MHz interne Taktfrequenz eingestellt. Hier muss der 8MHz Bootloader gebrannt werden, ***sonst funktioniert die I2C-Kommunikation nicht.***

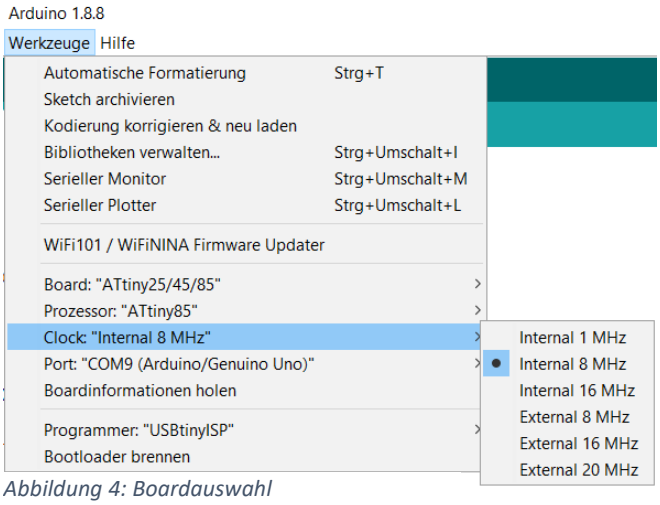


Abbildung 4: Boardauswahl

I2C-Bibliothek für den ATtiny85

KRITISCH: Für die I2C-Kommunikation existieren mehrere Alternative Bibliotheken. Hier muss auf die Anforderungen geachtet werden: Der ATtiny soll als I2C-Slave erst auf Anfrage des I2C-Masters antworten. Dazu muss er den Befehl „**TinyWire.onRequest**“ verstehen.

Es kommt daher nur die folgende Bibliothek zum Einsatz: https://github.com/rambo/TinyWire

UND NICHT: ~~https://github.com/lucullusTheOnly/TinyWire~~

Wird versehentlich die Lucullus-Bibliothek zuerst geladen, und anschließend die Rambo-Variante, so gibt es Überschneidungen, und der Tiny verhält sich nicht vorhersagbar.

Von der Github-Seite wird das Repository „**tinywire-rollback**“ als Zip heruntergeladen und ausgepackt. Der Ordner „**tinywireS**“, wird hier hin verschoben:

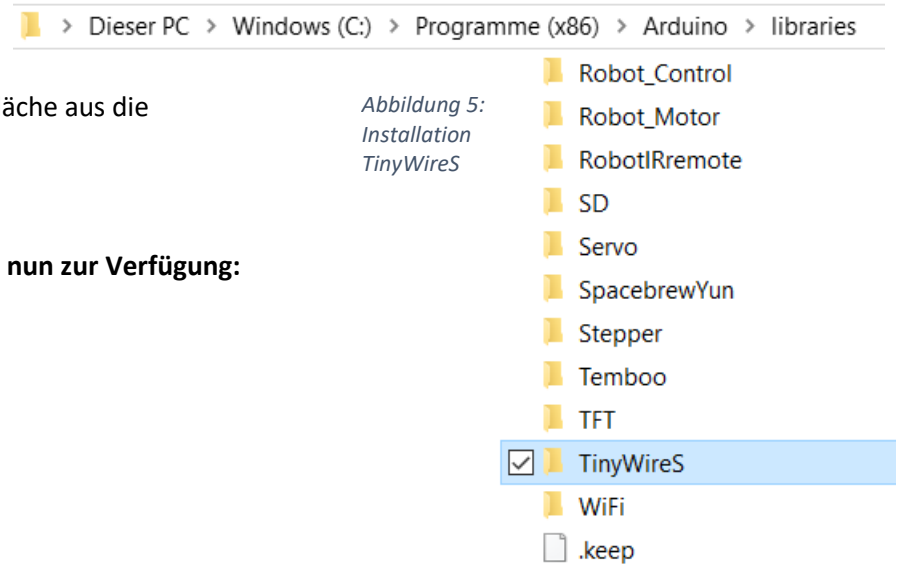


Abbildung 5: Installation TinyWireS

Anschließend können von der Arduino-Oberfläche aus die Beispiel-Dateien geladen werden.

Die folgenden Befehle für den ATtiny stehen nun zur Verfügung:
http://highlowtech.org/?p=1695

```
pinMode()
digitalWrite()
digitalRead()
analogRead()
analogWrite()
shiftOut()
pulseIn()
millis()
micros()
delay()
delayMicroseconds()
SoftwareSerial
```

Zusätzlich noch die Befehle der TinyWireS-Bibliothek:

```
TinyWireS.begin(I2C_Adresse_des_Tinys);

TinyWireS.onReceive(empfangs_Eventhandler_Funktion);
einByteWert_volatile = TinyWireS.receive();

TinyWireS.onRequest(sendeanforderungs_Eventhandler_Funktion);
TinyWireS.send(einByteWert_volatile);

TinyWireS_stop_check();

TinyWireS.end();
```

Spezifikationen HiTechnic-Ball:

Es existieren 4 verschiedene Modi von Mode A bis Mode D. Jeder Modus generiert eine unterschiedliche Wellenform. Der Ba ll wird für den Robocup auf Mode A verwendet.

Quelle:

http://www.drgraeme.net/DrGraeme-free-NXT-G-tutorials/Ch108/SoccerGenIINXTG/Soccer%20Ball/RCJ-05_Use_Discussion.htm

Abbildung 6: Hitechnic-Soccerball

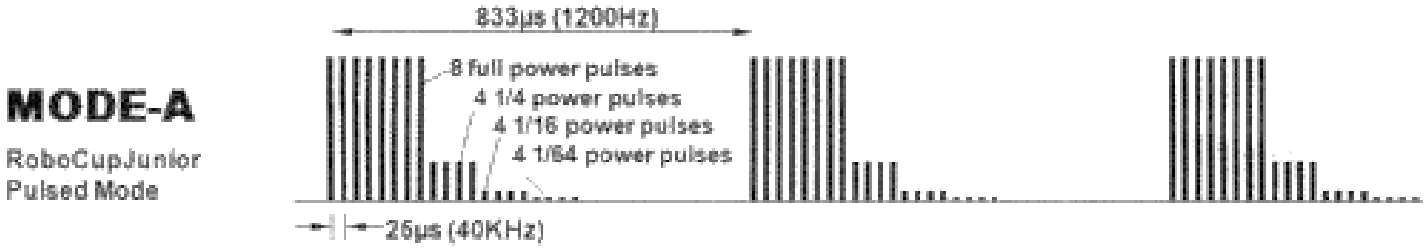


Abbildung 7: Ausschnitt Hitechnic-Dokumentation

Demodulation des 40 kHz-Signals durch den TSOP:

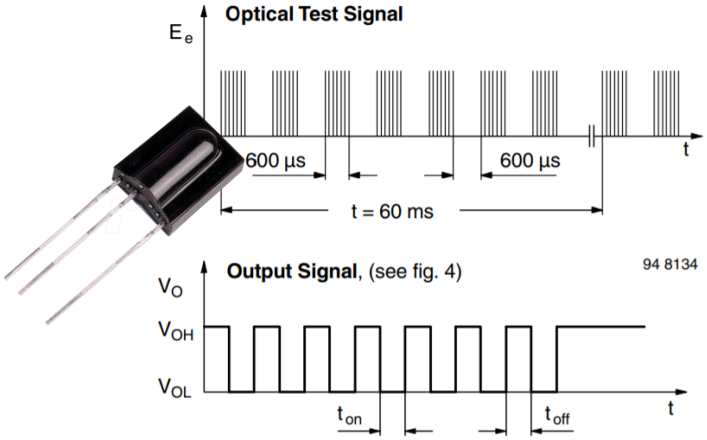


Abbildung 8: Ausschnitt TSOP-Datenblatt

Um das Signal zu demodulieren, wird über einen 40kHz-TSOP gefiltert, z.B. diesen hier:

TSOP31240 40kHz

https://www.reichelt.de/ir-empfaenger-module-tsop31240-40khz-tsop-31240-p107211.html?&trstct=pos_4

Da der TSOP invertiert, sieht obiges Hitechnik-Soccerball-Signal dann so aus:

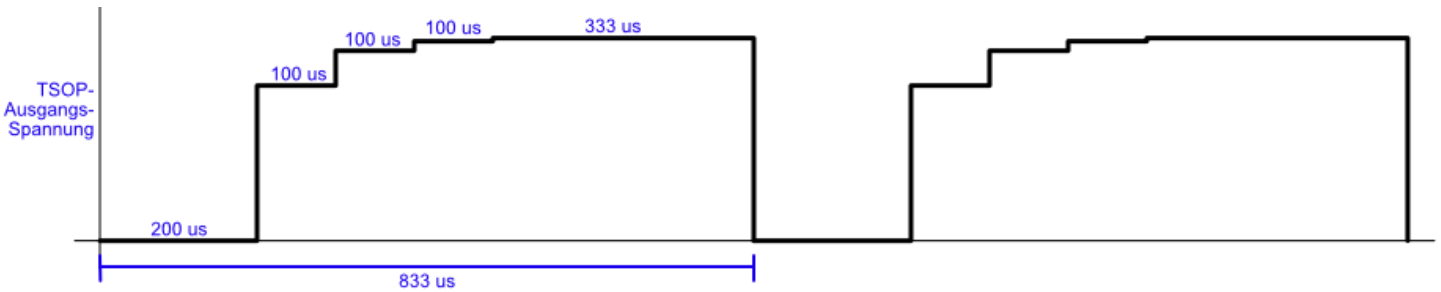


Abbildung 9: Ausgangssignal TSOP

Der durchschnittliche Duty-Cycle des demodulierten PWM-Signals liegt bei ca. 1-(200/833) = 100%-24% = 76%!

Aufbau der Testschaltung

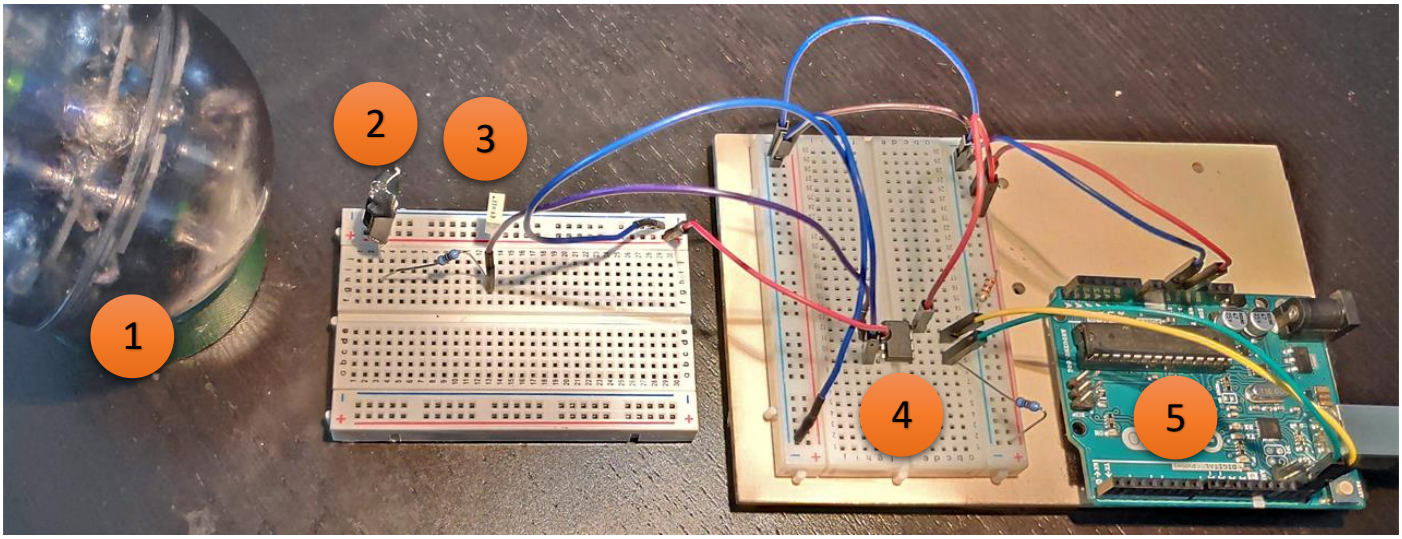


Abbildung 10: Breadboard-Setup

1. Hitechnic-Soccerball
2. TSOP, mit Isolierband abgeklebt, um den Sichtwinkel einzugrenzen
3. Ausgangssignal wird durch parallelgeschaltetes RC-Glied geglättet: 100nF, 2 kHz
4. ATtiny85 (I2C-Slave)
5. Arduino Uno (I2C-Master)

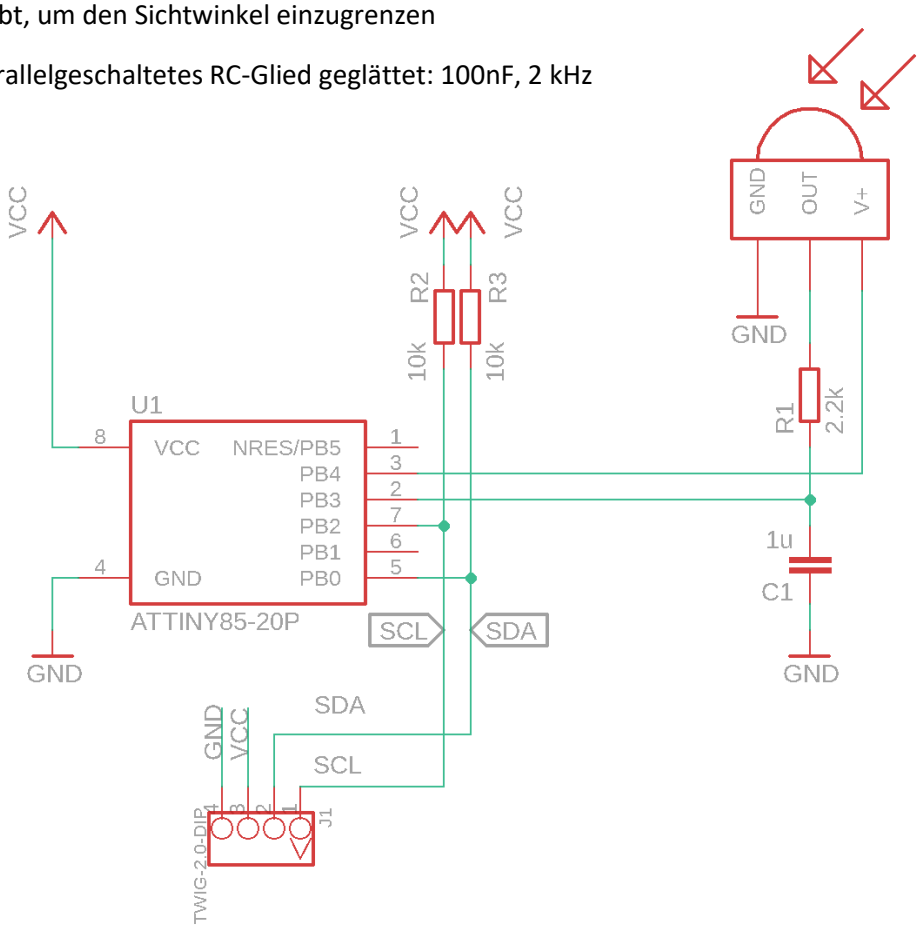


Abbildung 11: EAGLE-Schaltplan

Der Ausgang des TSOP wird über einen Tiefpass geglättet. Dieser berechnet sich nach der Grenzfrequenzformel

$$C_{Mindestgröße} = \frac{1}{2\pi \cdot f_{Grenzfrequenz} \cdot R}$$

Setzt man für R einen Wert von 2.2kOhm ein, so ergibt sich die Mindestgröße für den Glättungskondensator auf 60nF. Daher wird ein etwas größerer Kondensator mit 100nF eingesetzt.

Arduino-Code für den Uno:

```
#include <Wire.h>


void setup(){
  Wire.begin();
  Serial.begin(9600);
}

void loop(){
  Wire.requestFrom(0x4, 1);
  while (Wire.available()){
    int i = Wire.read();
    Serial.println(i);
  }
  delay(200);
}
```

Arduino-Code für den ATtiny85, Version 1:

```
#include <TinyWireS.h>

volatile byte analogWert = 0;
long deltaMikros = 0;
long altMikros = 0;
boolean messenJaNein = true;

void requestEvent(){
  TinyWireS.send(analogWert);
}

void setup(){
  TinyWireS.begin(0x4);
  TinyWireS.onRequest(requestEvent);
  pinMode(A3, INPUT);
  pinMode(A2, OUTPUT);
}

void loop(){
  deltaMikros = micros() - altMikros;
  if (messenJaNein == true) {
    if (deltaMikros <= 100000) {
      digitalWrite(A2, HIGH);
      analogWert = (byte) analogRead(A3);
    }
    if (deltaMikros > 100000) {
      altMikros = altMikros + deltaMikros;
      messenJaNein = false;
    }
  }
  if (messenJaNein == false) {
    if (deltaMikros <= 100000) {
      digitalWrite(A2, LOW);
    }
    if (deltaMikros > 100000) {
      altMikros = altMikros + deltaMikros;
      messenJaNein = true;
    }
  }
  TinyWireS_stop_check();
}
```

Erläuterung:

Der TSOP wird jeweils für 100ms angeschaltet, indem der Pin PB4 (Arduino-Pin A2, bzw. 4) auf Digital High geschaltet wird. Laut Datenblatt des TSOP werden 3mA Versorgungsstrom benötigt, was weit unterhalb der Grenzstromstärke des Tiny liegt.

Nach diesen 100ms wird er für 100ms ausgeschaltet, indem der PB4 auf LOW gesetzt wird.

Der maximal mögliche Analogspannungswert liegt bei 76%*1024 = 778. Geht man weiterhin von einem Eingangswiderstand des ATtiny von ca. 10k aus, verringert sich der maximale ADC-Wert durch den vorgeschalteten 2.2k-Widerstand auf 622. Zudem wird die maximale Helligkeit am TSOP niemals erreicht.

Daher ist es nicht notwendig, den 10Bit-Wert des AnalogReads an A3 des Tiny auf 255 zu limitieren.

Arduino-Code für den ATtiny85, Version 2:

```
#include <TinyWireS.h>

volatile byte analogwert = 0;
volatile long messwert = 0;
volatile long durchlauf = 1;
long deltaMikros = 0;
long altMikros = 0;
boolean messenJaNein = true;

void requestEvent() {
  TinyWireS.send(analogwert);
}

void setup() {
  TinyWireS.begin(0x4);
  TinyWireS.onRequest(requestEvent);
  pinMode(A3, INPUT);
  pinMode(A2, OUTPUT);
}

void loop() {
  deltaMikros = micros() - altMikros;
  if (messenJaNein == true) {
    if (deltaMikros <= 2*833) {
      messwert = messwert +  analogRead(A3);
      durchlauf = durchlauf + 1;
    }
    if (deltaMikros > 2*833) {
      analogwert = messwert/durchlauf;
      altMikros = altMikros + deltaMikros;
      messenJaNein = false;
    }
  }
  if (messenJaNein == false) {
    if (deltaMikros <= 10000) {
      digitalWrite(A2, LOW);
    }
    if (deltaMikros > 10000) {
      digitalWrite(A2, HIGH);
      altMikros = altMikros + deltaMikros;
      messwert = 0;
      durchlauf = 1;
      messenJaNein = true;
    }
  }
  TinyWireS_stop_check();
}
```