Projekt Gewitterwarner

Zusammengestellt von DL6OAA für einen Vortrag im H39-Workshop

https://www.qsl.net/dh1stf/blitzbilder/images/blitz-stroke.jpg

Wolkentypen und ihre Feldkurven:

Nach Kilinski (Lehrbuch der Luftelektrizität) kann man Wolken drei ungefähre Typen von Ladungsverteilungen zuordnen:

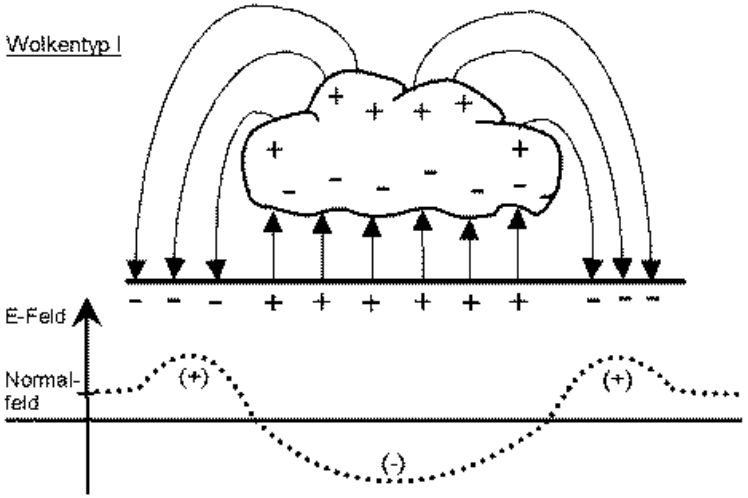
Typ I: die positiv-polare Wolke trägt im oberen Teil der Wolke die positiven Ladungen und entsprechend im unteren Teil die Negativen;

Typ II: die negativ-polare Wolke ist entsprechend andersherum gepolt, also oben negativ und unten positiv;

Typ III: die tripolare Wolke, also eine positiv-polare Wolke mit einem eng begrenzten positiven Zentrum inmitten der negativen Ladungen an der Unterseite.

In den Skizzen sieht man die entstehende Registrierkurve beim Vorüberziehen der Wolken unterschiedlichen Typs:

Dem letztgenannten, tripolaren Wolkentyp gehören auch meistens die Gewitterwolken an. Jedoch muss man sich immer vor Augen halten, dass dies nur grobe Modelle der Wirklichkeit sind. Bei Gewitterwolken hat sich allerdings gezeigt, dass der Aufbau, also dass der obere Teil des "Doms" positiv ist und der untere negativ, bei so gut wie jedem Gewitter gleich ist. Dies hängt mit den Aufladungsvorgängen in der Wolke zusammen. Damit kann man z.B. für die Gewittervorhersage Aussagen treffen, wie die Feldkurve im groben aussehen müsste um das Heranziehen eines Gewitters zu belegen.
https://www.qsl.net/dh1stf/



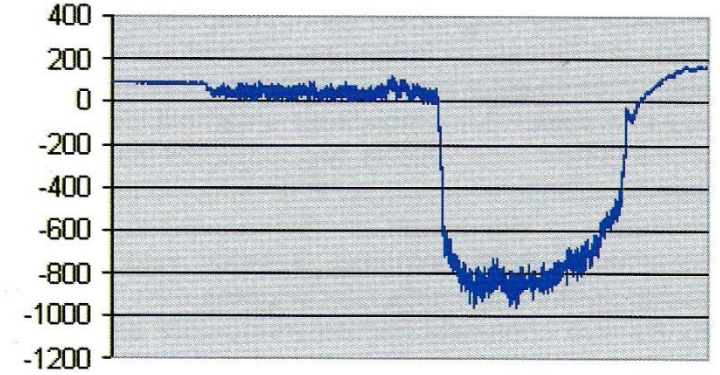
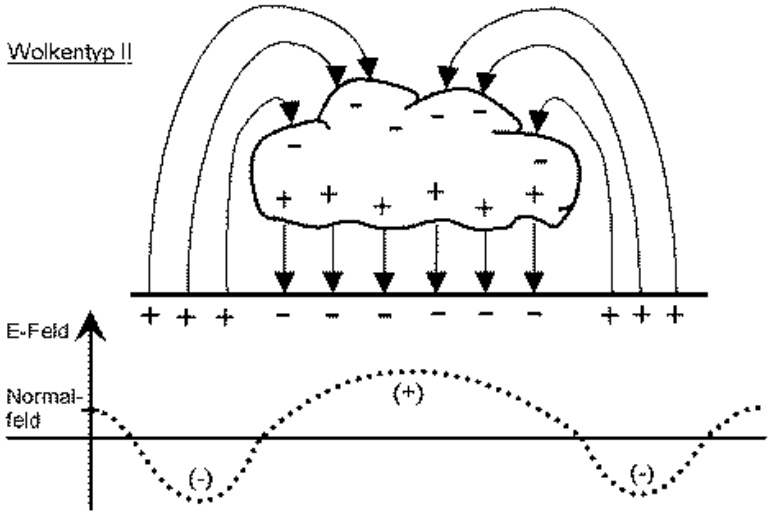


Abb. oben: In diesem Digramm vom 23.11.2001, das einen Zeitabschnitt von 8.10 bis 14.00 Uhr wiedergibt, ist eine Feldkurve zu sehen, die dem Wolkentyp I in der Abbiildung links entspricht. Die meteorologischen Zusammenhänge wurden jedoch noch nicht abschließend untersucht.



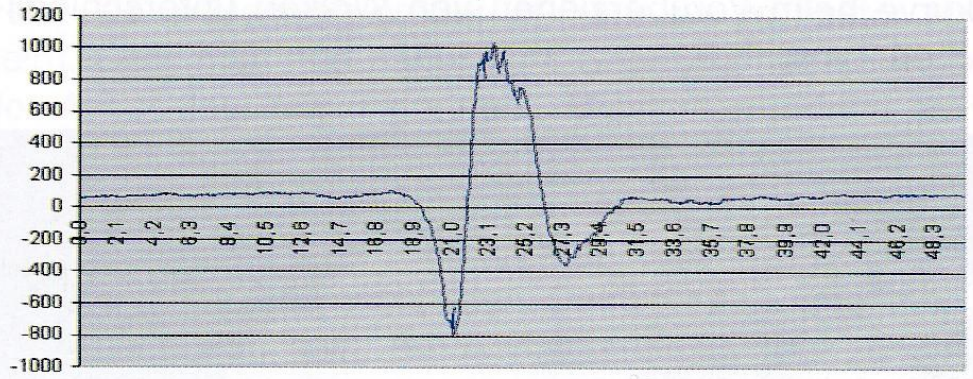
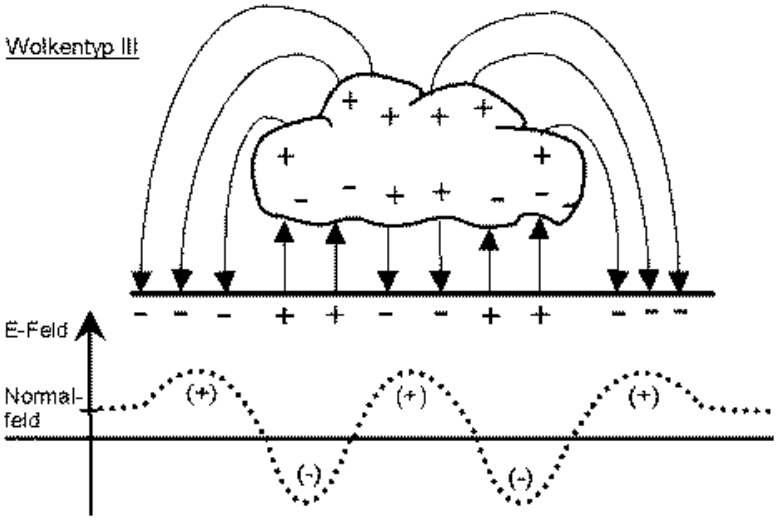
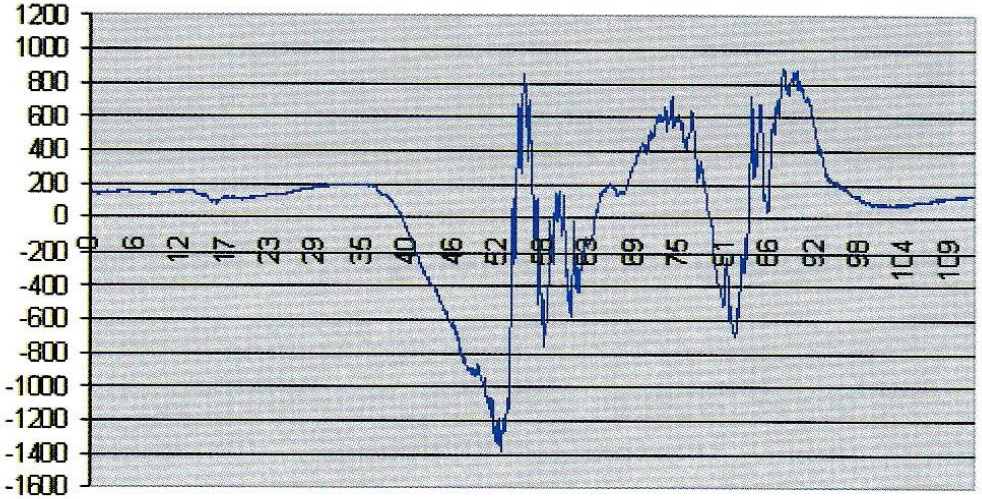
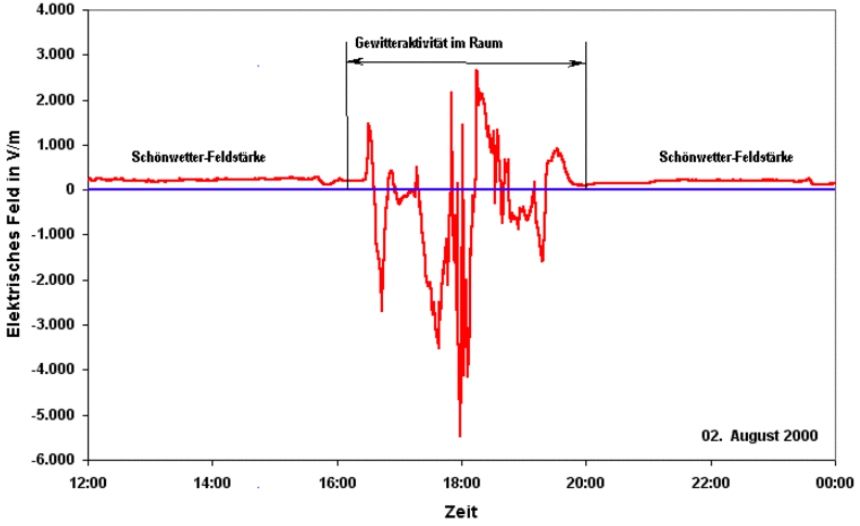


Abb. oben: Im EXCEL-Diagramm ist eine vom Autor aufgenommene Kurve vom 24.11.2001 ab 5.25 Uhr zu sehen, welche recht eindeutig dem Wolkentyp II entspricht.



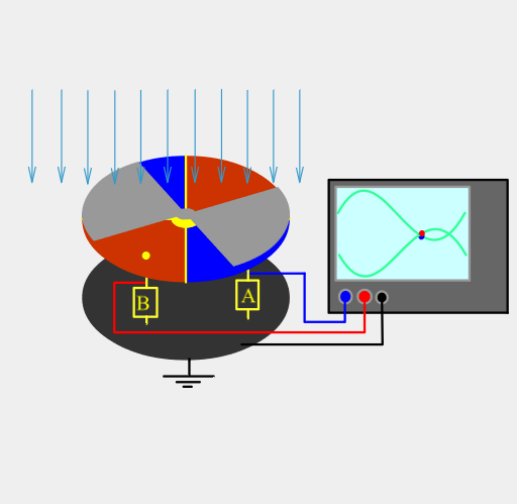


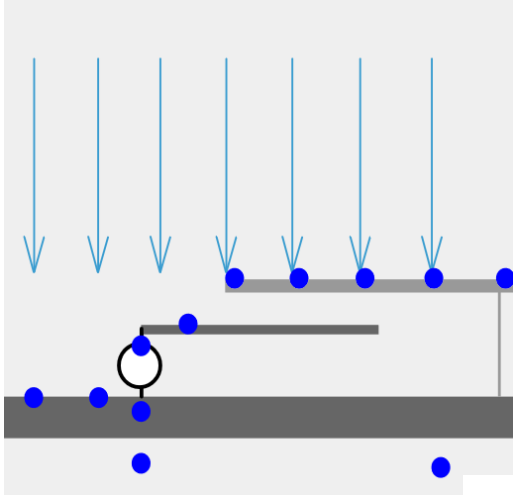
Elektrostatisches Feld gemessen mit der Feldmühle



Messungen elektrischer Felder mit einer Feldmühle (Animation):

https://www.leifiphysik.de/elektrizitaetslehre/ladungen-elektrisches-feld/ausblick/feldmuehle





Stefan Kneifel, DH1STF

Ein elektrisches Feld influenziert Flächenladungen auf der Erde.

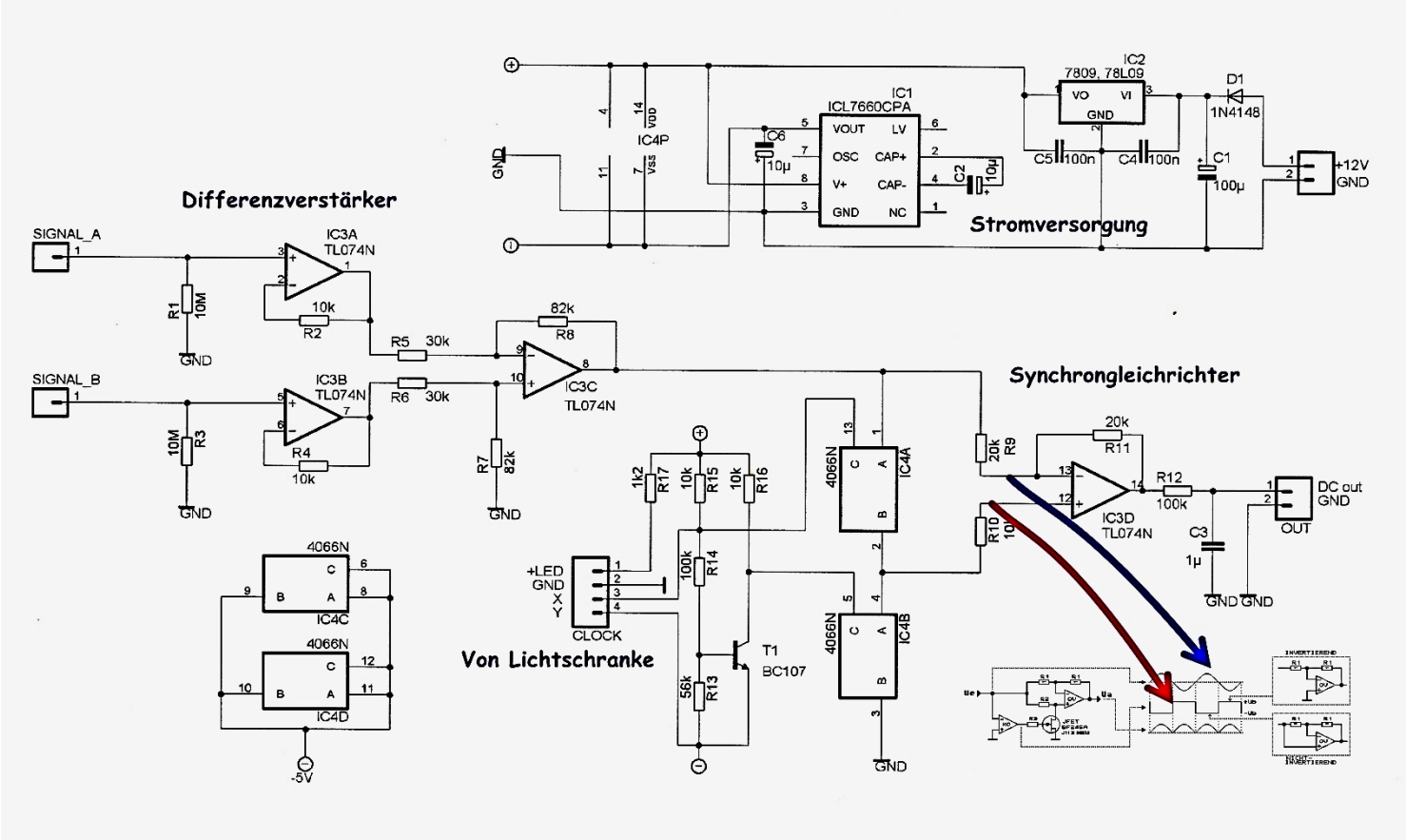
Legt man eine Metallplatte über die Erdoberfläche und erdet sie, so treten auf ihr ebenfalls Flächenladungen auf - je nach Richtung des Feldes positive oder negative.

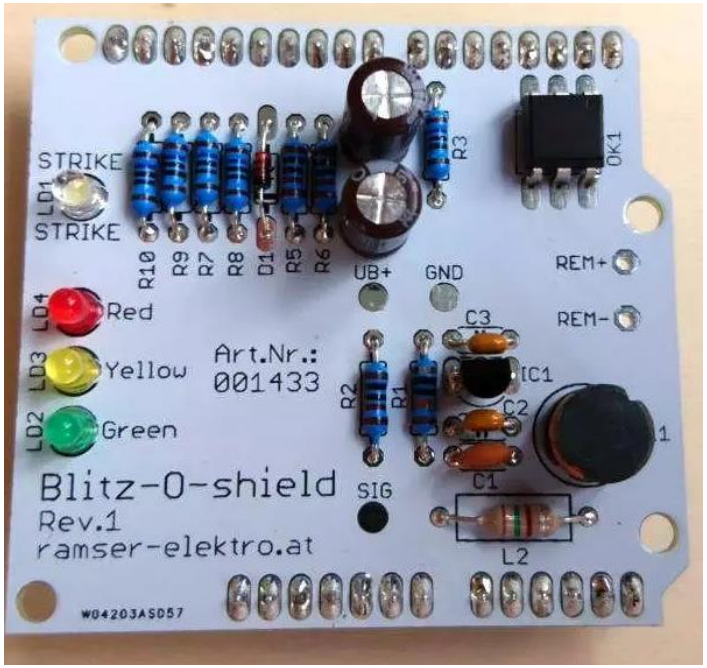
Wenn diese Platte mit einer anderen geerdeten Platte überdeckt wird, so treten die Ladungen auf der oberen Platte auf und die Ladungen auf der unteren Platte fließen zur Erde ab und können mittels eines Messgeräts bestimmt werden.

https://www.elektronik-labor.de/Elo/Feldmuehle.html

Stefan Kneifel (DH1STF) hat sein Projekt „Eigenbau-Feldmühle“ in den AATiS-Praxisheften 11 und 12 veröffentlicht. Der AATiS-Bausatz AS529 (Auswerteelektronik für die Feldmühle) ist leider vergriffen.

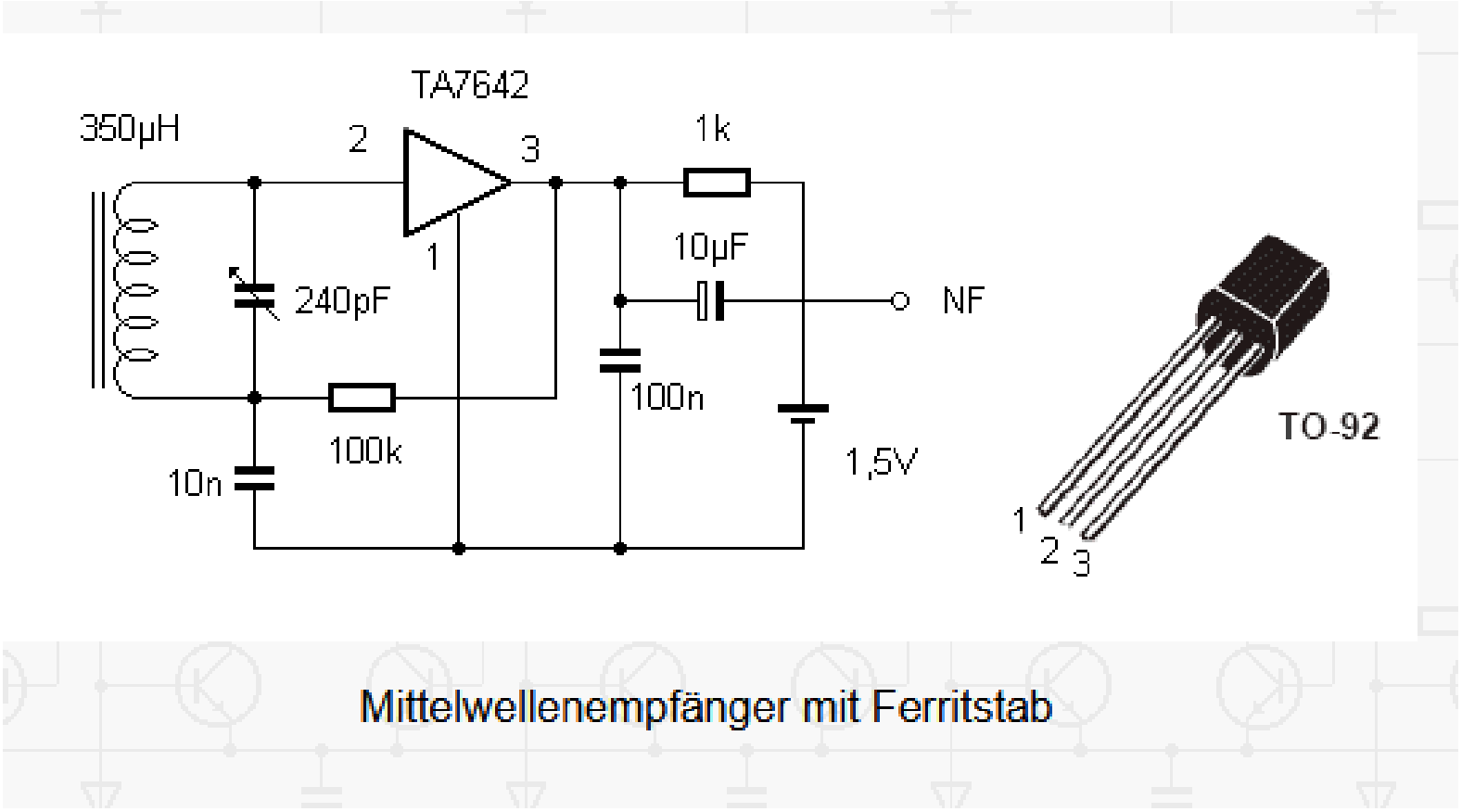
Feldmühle im Kennedy Space Center in Florida





https://www.elektronik-labor.de/Elo/TA7642.html
https://www.aeq-web.com/ramser-blitz-o-shield-arduino-blitzdetektor/
https://de.elv.com/elv-gewitterwarner-gw1-komplettbausatz-130942

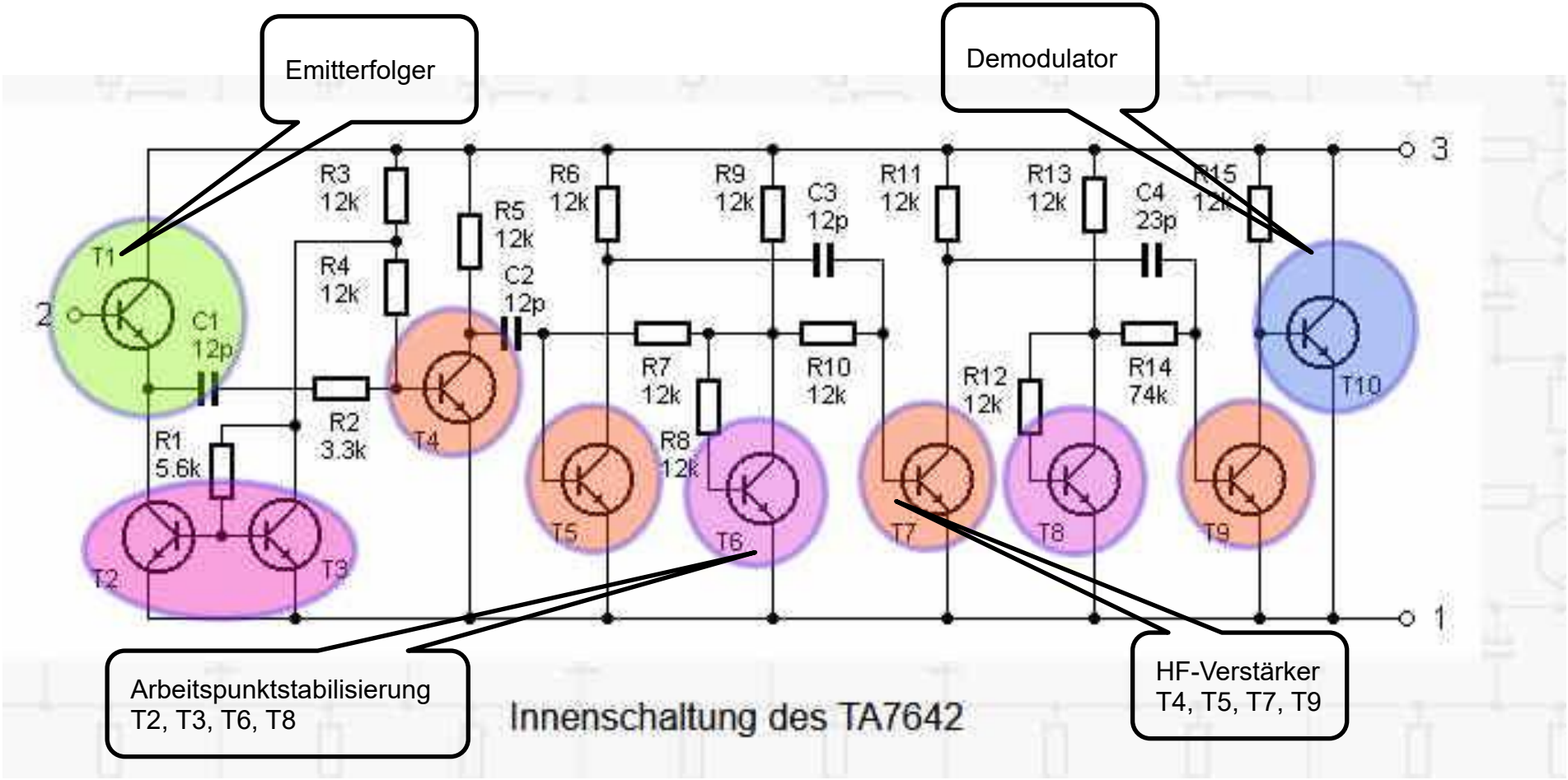
Das MW-Radio mit dem IC TA7642



Mittelwellenempfänger mit Ferritstab

Der integrierte Mittelwellen-Empfängerbaustein ZN414 der Firma Feranti wurde später durch den MK484 ersetzt und ist inzwischen als **TA7642** erhältlich.

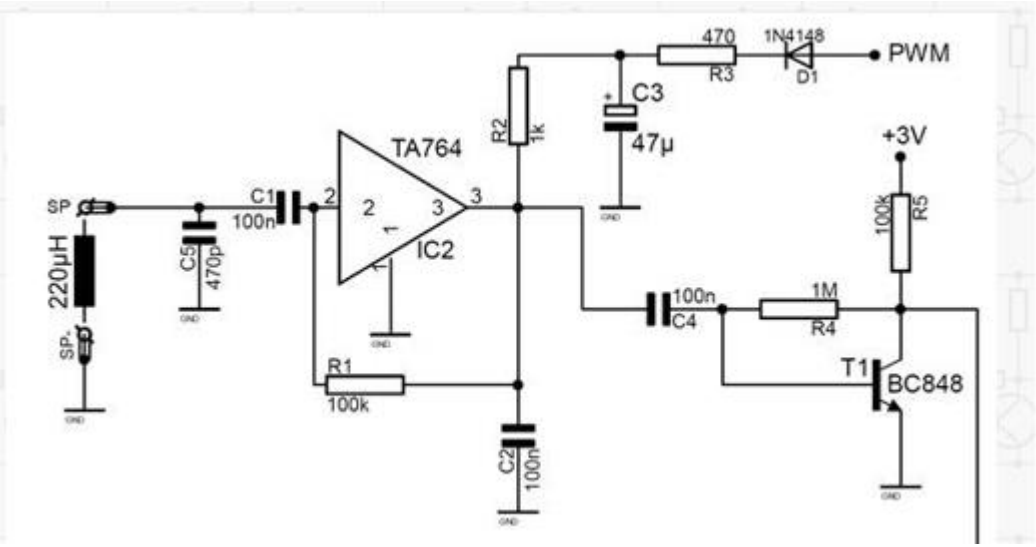
Kainka (Franzis Verlag) Gewitterwarner
Verwendet als Detektor den MW-Baustein TA7642 (ca. 2€)



Schaltungsbeschreibung

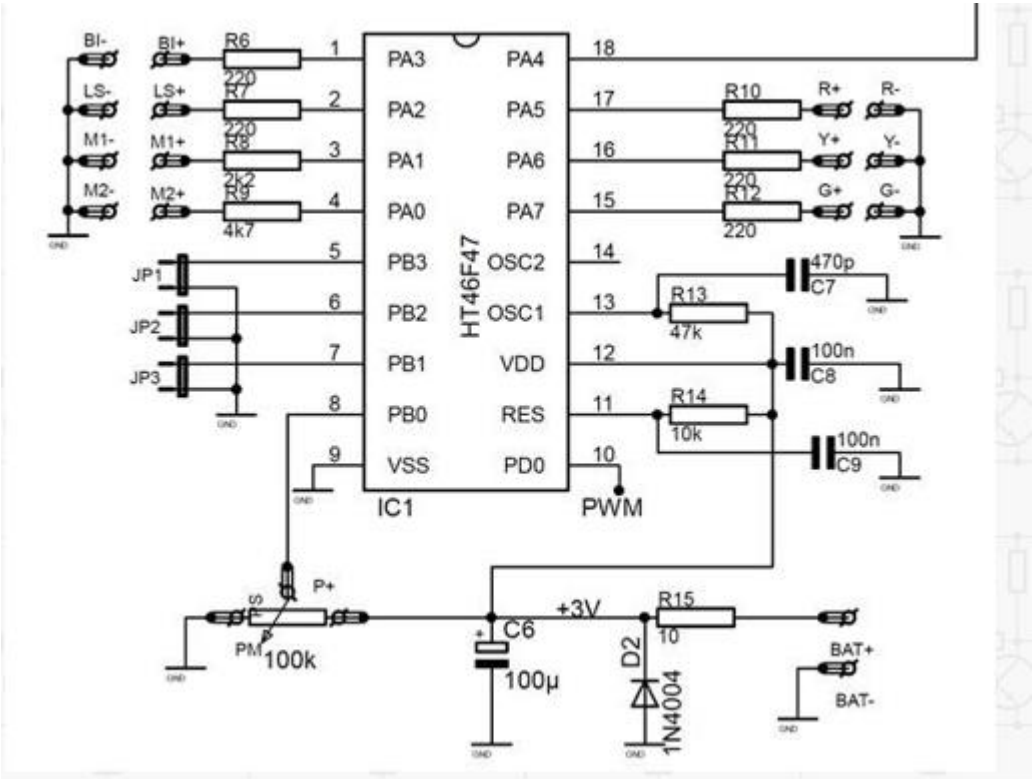
Der Gewitterwarner verwendet einen integrierten AM-Empfänger TA7642 zusammen mit einer fest auf 500 kHz abgestimmten Mini-Ferritantenne in Form einer Festinduktivität. Das IC besitzt eine sehr große Verstärkung, die in weiten Grenzen vom Betriebsstrom abhängt. Die Versorgungsspannung für den Empfänger wird daher vom Mikrocontroller über ein geglättetes PWM-Signal geliefert. So kann die Verstärkung bei der Initialisierung der Schaltung passend eingestellt werden. Zugleich werden dabei Unterschiede in der Batteriespannung ausgeglichen.

Das NF-Signal des Empfängers wird mit einem Transistor ca. 100-fach verstärkt. Das verstärkte Signal gelangt dann an den Eingang PA4 des Mikrocontrollers, der als Impulszählereingang verwendet wird. Der Ruhepegel am Kollektor von T1 liegt bei ca. 0,6 V und damit deutlich unter der Auslöseschwelle des Zählers von ca. 1,5 V. Damit werden verstärkte Impulse ausgewertet, die mindestens 1 V über dem Ruhepegel liegen.



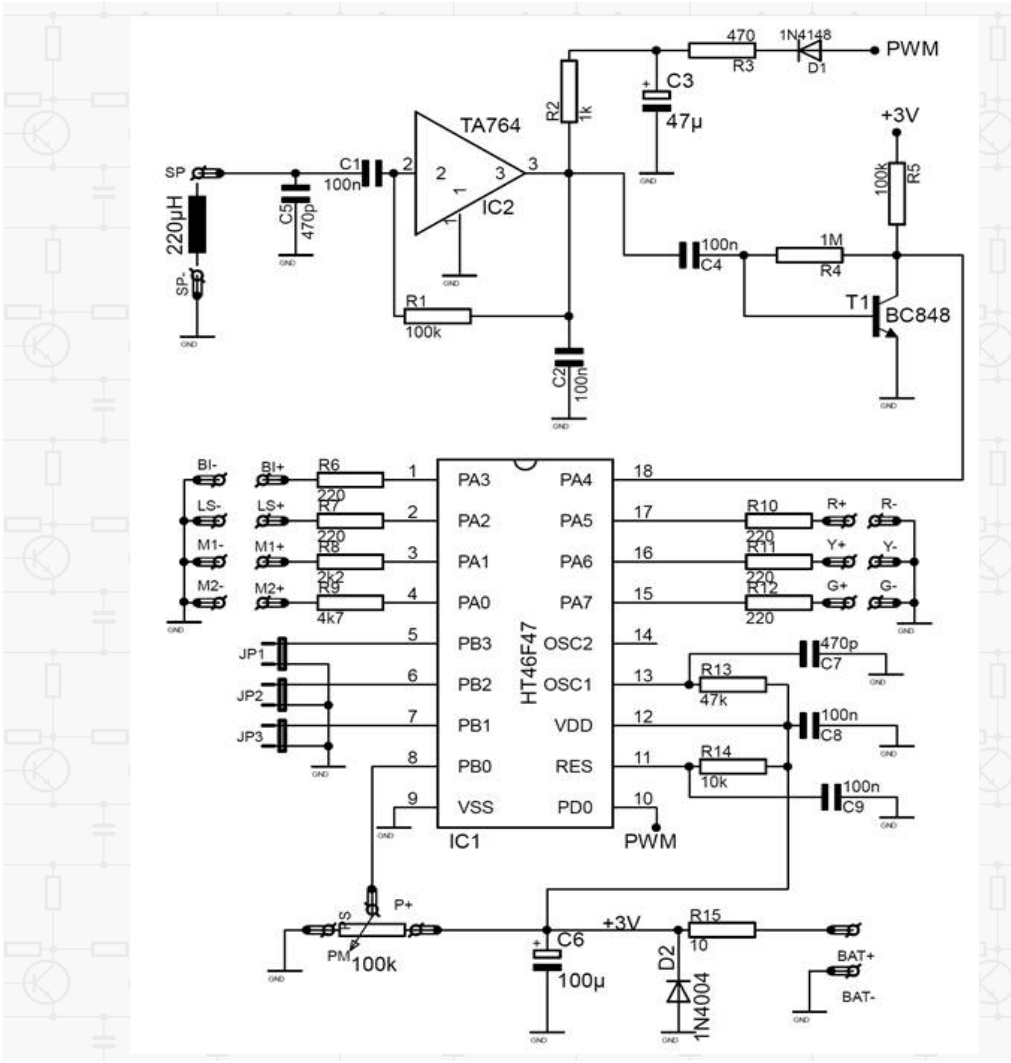
Der 500-kHz-Empfänger empfängt immer ein Grundrauschen, das teilweise vom atmosphärischen Rauschen stammt und teilweise vom Eigenrauschen des TA7642. Gewitterimpulse sollen gewertet werden, wenn sie deutlich aus diesem Grundrauschen herausragen. Dies gelingt durch eine automatische Einstellung der Verstärkung. Nach dem Neustart erhöht der Controller seine PWM-Ausgangsspannung langsam und kontinuierlich, bis das Grundrauschen den Zähler anspricht. Dann wird die PWM-Spannung um einen gewissen Betrag verringert. Damit ist die Gesamtverstärkung optimal eingestellt. Die Initialisierung dauert länger, wenn die Batteriespannung geringer wird und gelingt noch bis herab auf 2,6 V.

Alle LED-Ausgänge des Mikrocontrollers besitzen Vorwiderstände von 220 Ω. Der Lautsprecher-Ausgang hat ebenfalls einen Vorwiderstand von 220 Ω, sodass statt des Piezo-Schallwandlers auch ein dynamischer Lautsprecher angeschossen werden kann. Zusätzlich gibt es an PA0 und PA1 Ausgänge zum Anschluss eines Drehspulmesswerks. Die Widerstände 2,2 kΩ und 4,7 kΩ bilden einen einfachen Digital-Analogwandler mit einer Auflösung von zwei Bit, also vier Anzeigestufen. Alle Ausgänge werden über die Firmware des Mikrocontrollers in Abhängigkeit von der Anzahl der empfangenen Impulse gesteuert. Die Eingänge an den Jumpern JP1 bis JP3 arbeiten mit internen Pullup-Widerständen und haben im Ruhezustand einen High-Zustand. Der Poti-Eingang ist hochohmig und darf nicht frei bleiben.

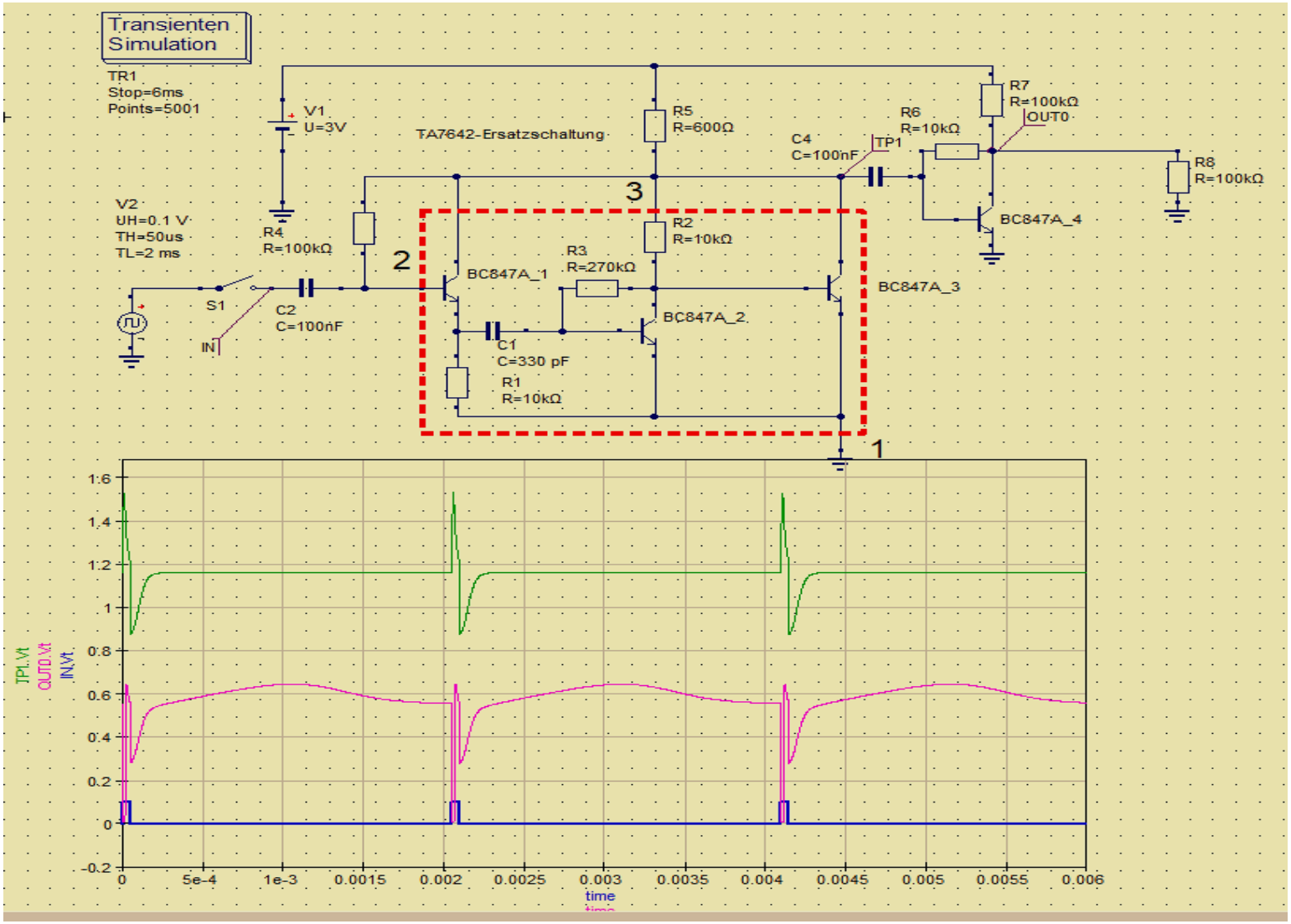


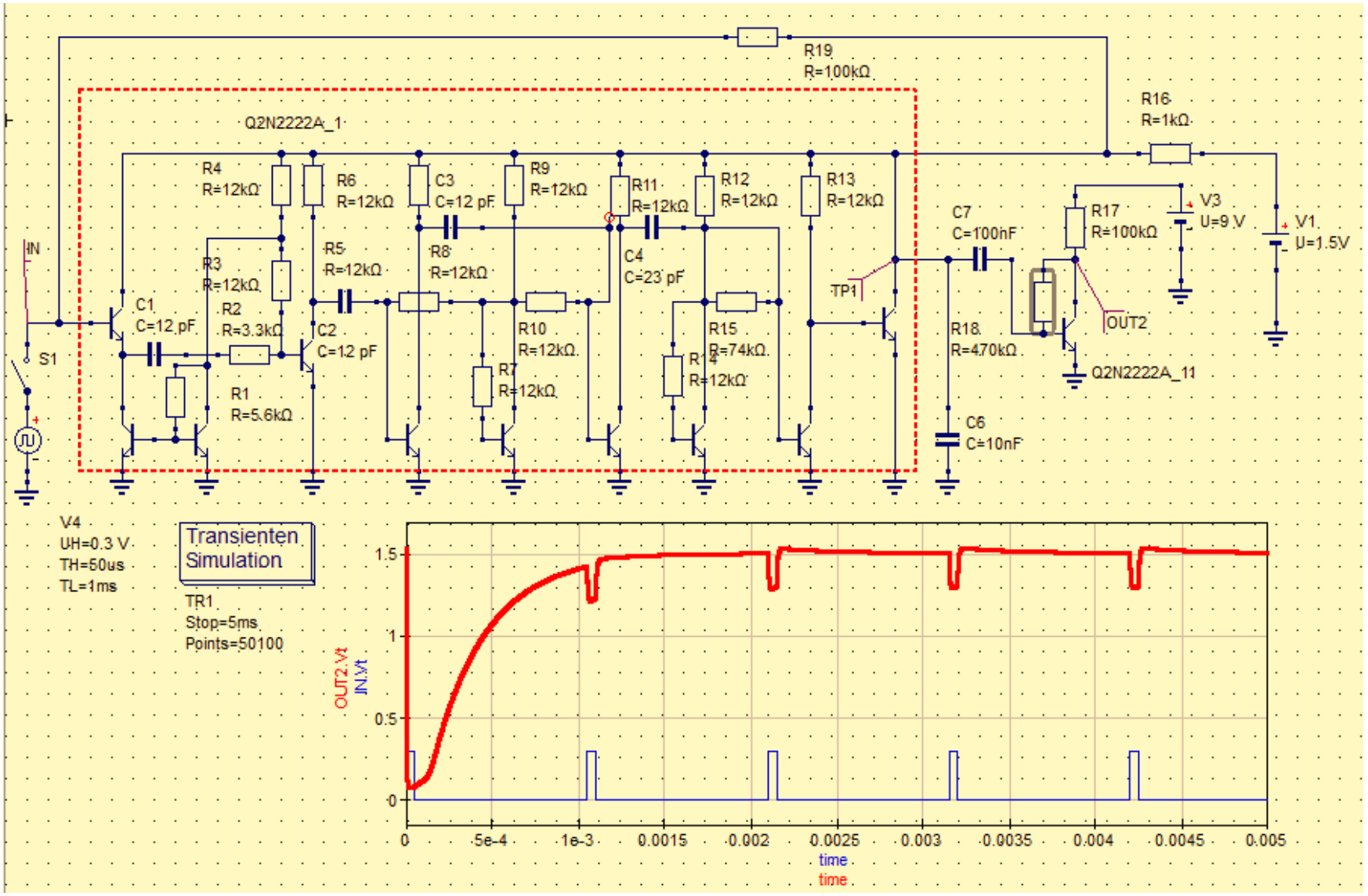
Die Batteriespannung gelangt über einen Schutzwiderstand von 10 Ω an den Mikrocontroller. Eine antiparallele Diode schützt die Schaltung gegen Falschpolung. Im Gegensatz zu einem Verpolungsschutz mit einer Diode in Reihe zur Batterie hat man im Ruhezustand nur einen sehr geringen Spannungsabfall von 10 mV, weil der Ruhestrom etwa 1 mA beträgt. Wenn eine der LEDs leuchtet, erhöht sich der Spannungsabfall am Schutzwiderstand auf etwa 50 mV.

https://www.elektronik-labor.de/Lernpakete/Gewitter.html


470
R3
1N4148
D1
PWM
C3
47µ
R2
1k
TA764
+3V
SP
C1
100n
C5
470p
220µH
SP-
IC2
100k
R5
100n
C4
1M
R4
T1
BC848
R1
100k
C2
100n
BI-
BI+
R6
220
LS-
LS+
R7
220
M1-
M1+
R8
2k2
M2-
M2+
R9
4k7
PA3
PA2
PA1
PA0
PB3
PB2
PB1
PB0
VSS
PA4
PA5
PA6
PA7
OSC2
OSC1
VDD
RES
PD0
HT46F47
IC1
R10
220
R+
R-
R11
220
Y+
Y-
R12
220
G+
G-
JP1
JP2
JP3
470p
C7
R13
47k
100n
C8
R14
10k
100n
C9
PWM
P+
P-
PM
100k
C6
100µ
+3V
R15
10
D2
1N4004
BAT+
BAT-

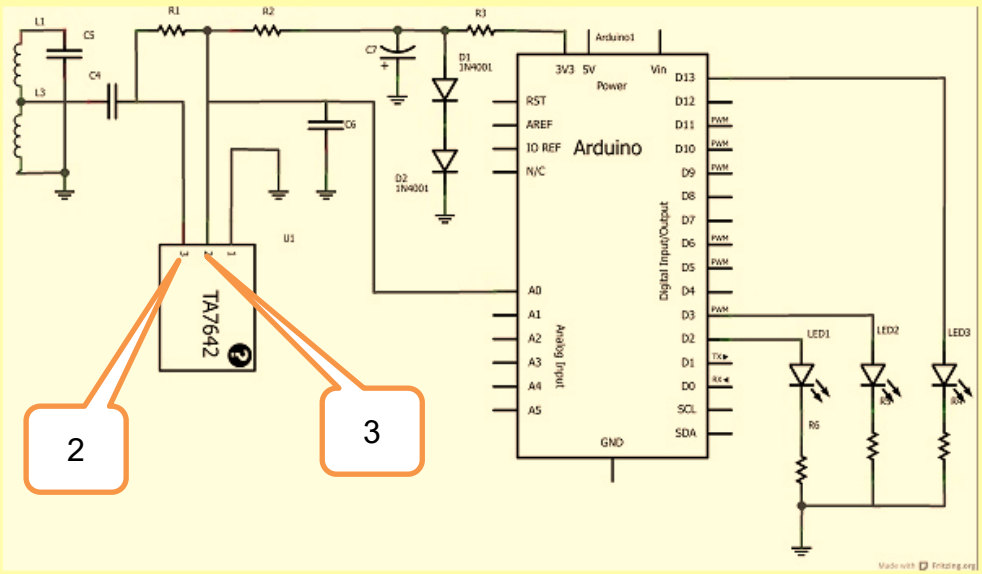
Gewitterwarner in der Simulation (mit Ersatzschaltung des TA7642):




R19
R=100kΩ
R16
R=1kΩ
Q2N2222A_1
R4
R=12kΩ
R6
R=12kΩ
C3
C=12 pF
R9
R=12kΩ
R11
R=12kΩ
R12
R=12kΩ
R13
R=12kΩ
C7
C=100nF
R17
R=100kΩ
V3
U=9 V
V1
U=1.5V
IN
R3
R=12kΩ
R5
R=12kΩ
R8
R=12kΩ
C4
C=23 pF
C1
C=12 pF
R2
R=3.3kΩ
C2
C=12 pF
R10
R=12kΩ
R15
R=74kΩ
TP1
R18
R=470kΩ
OUT2
S1
R1
R=5.6kΩ
R7
R=12kΩ
R14
R=12kΩ
Q2N2222A_11
C6
C=10nF
V4
UH=0.3 V
TH=50us
TL=1ms
Transienten
Simulation
TR1
Stop=5ms
Points=50100
OUT2.Vt
IN.Vt
1.5
1
0.5
0
0
5e-4
1e-3
0.0015
0.002
0.0025
0.003
0.0035
0.004
0.0045
0.005
time
time

Hier findet man eine Arduino-Version (Gewitterwarner mit dem TA7642):

https://fkainka.de/blitzduino-blitzwarner/

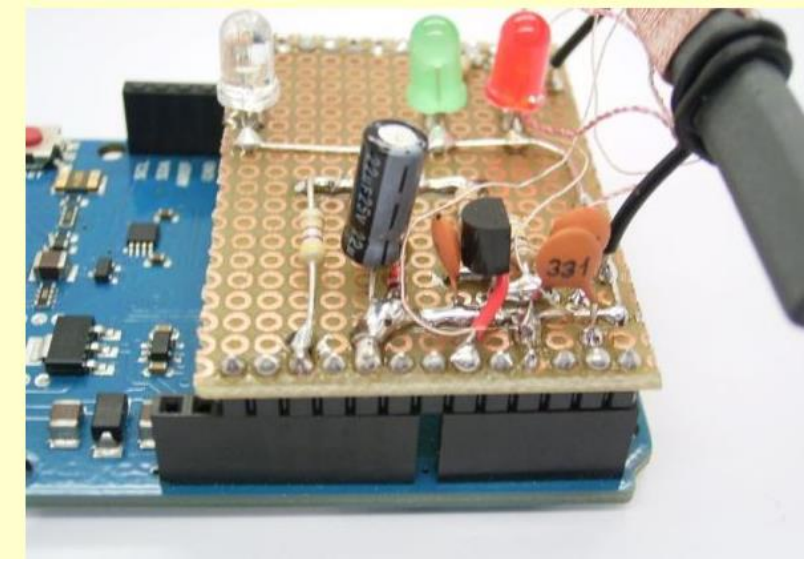


C4=100nF; C5=330pF; C6=100nF; C7=22µF;

L1=250µH; L3=50µH;

LED: Pin13= Weiß; Pin2= Grün, Pin3 = Rot;

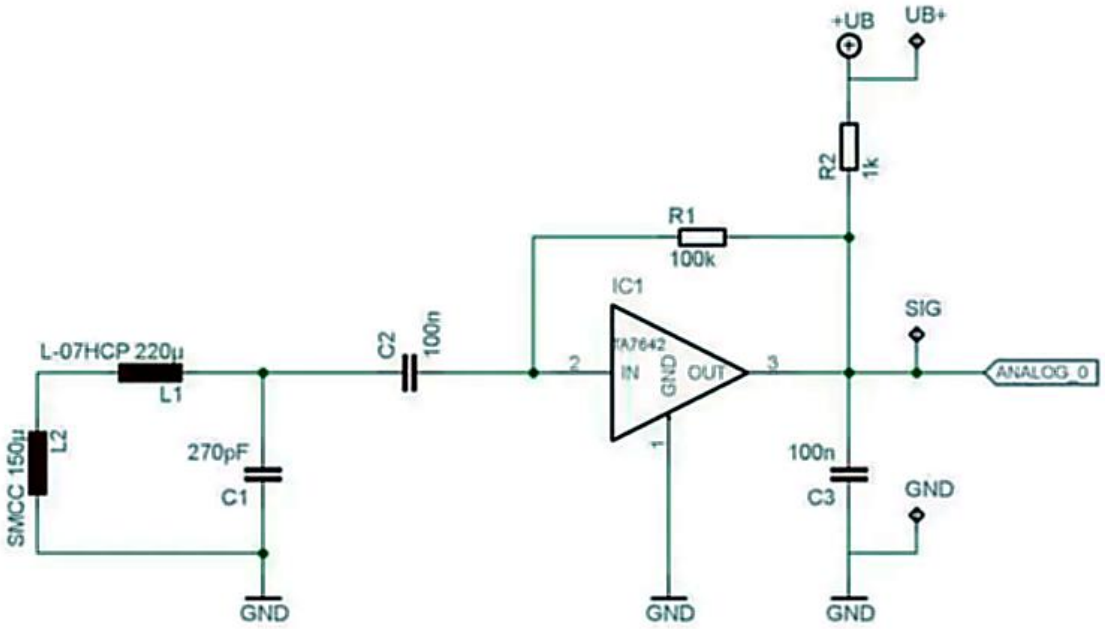
R1=100kΩ; R2=680 Ω; R3=470 Ω; R4=220 Ω; R5=220 Ω; R6=220 Ω;



Ramser-Elektro: Gewitterwarner Arduino Shield

(https://www.ramser-elektro.at/blitz-o-shield-der-gewitter-detektor/) Verwendet als Detektor auch den MW-Baustein TA7642

Funktionsweise::
Die Funkwellen der Blitze werden vom TA7642 empfangen, vorverstärkt, demoduliert und verstärkt.



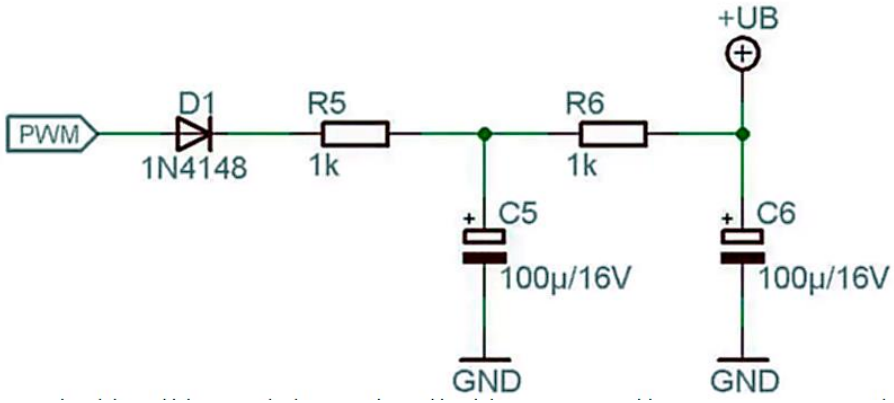
Danach wird der Ausgang auf den Analogeingang A0 des Arduino geführt.

Dies ergibt pro Blitzentladung einen „Spannungeinbruch“ der Spannung am A0 Arduino Eingang, welche sich auswerten lässt.

Umso mehr „Spannungeinbrüche“ am Analogeingang, umso mehr Entladungen, umso näher das Gewitter.
Die Amplitude des Signals ist in diesem Projekt egal!

Beim Blitzwarner des Elektroniklabors (Kainka) wurde der normale Logiklevel des Portpins gewählt, um einen Interrupt auszulösen.
Bei meinem Blitzdetektor wird der ADC des Arduino verwendet.
Dadurch kann die PWM Dynamisierung für den AGC besser angepasst werden!

Die Verstärkung des TA7642 wird mittels geglätteter Spannung eines PWM Signals geregelt.



Die Funktionsweise der Software (Arduino-Sketch im Anhang):

Bei Start wird die Spannung des TA7642 so lange erhöht, bis eine gewisse Grundspannung am TA7642 anliegt.
Dies dient dazu, die Exemplarstreuung auszugleichen.

Anschließend wird zyklisch ein laufender Mittelwert gebildet, um Störeinflüssen entgegenzuwirken.

Trifft ein Blitzevent auf (Spannung bricht an A0 ein), so wird eine Countervariable erhöht.

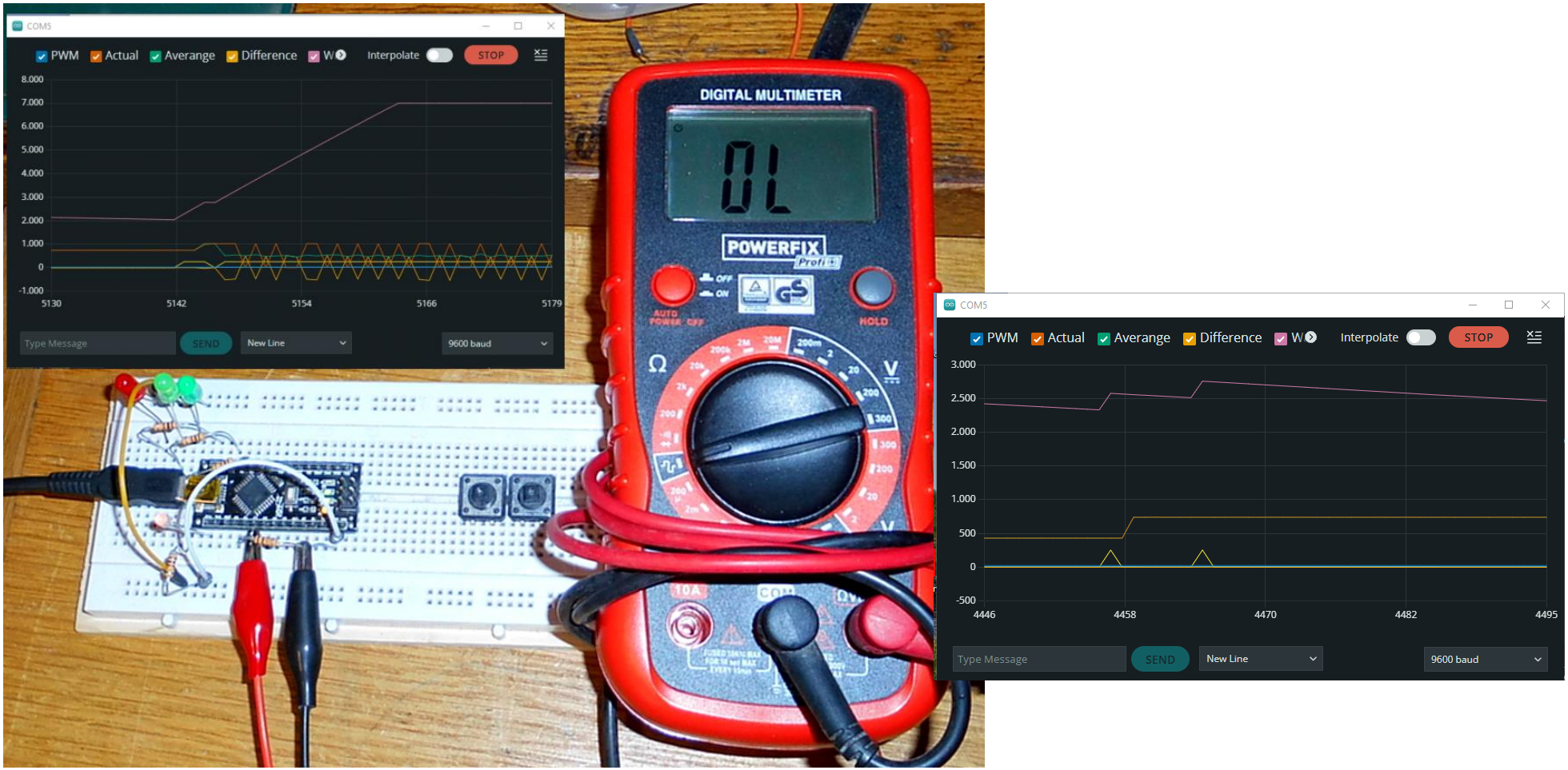
Nach einer gewissen Zeit wird der Wert natürlich wieder dekrementiert.
Abhängig vom Wert, werden die LEDs angesteuert.

Erkennungsalgorithmus:
Bei jedem erkannten Blitz (max.8 pro Sekunde) wird ein **Interrupt** ausgelöst, und ein „Strikecount“ hochzählt.
Der „Strikecount“ wird jede Sekunde auf ein „Level“ aufsummiert.
Jede Sekunde wird von diesem „Level“ ein Anteil (Decay) abgezogen.
Umso höher das „Level“, umso mehr Blitze sind „in der Nähe“ d.h. umso näher ist ein Gewitter.

Ab FW Revision 8: Werden mehr als 8 Blitze pro Sekunde detektiert, ist von (starken) Umgebungsstörungen auszugehen. Darum wird der PWM-Wert gesenkt.

https://www.ramser-elektro.at/potzblitz-der-gewitterwarner/

Mit einem Digitalvoltmeter kann die Funktionsweise der Software getestet werden. Die Anschlüsse des DVM werden an die Arduino-Eingänge A0 und GND angeschlossen. Ist der Ohm-Mesbereich auf einen hohen Widerstandswert eingestellt, ist die Spannung an A0 gering – je kleiner der Ohm-Messbereich, umso höher wird die Spannung an A0. Wird am DVM zwischen den Messbereichen gewechselt, entspricht das einem Spannungseinbruch → Blitz-Event.
Man sieht im Seriellen Plotter dann, wie durch dieses Ereignis die Warnstufe nach oben korrigiert wird. Mit einem Rechtecksignal steigt die Warnstufe kontinuierlich an bis zum Grenzwert von 7000.



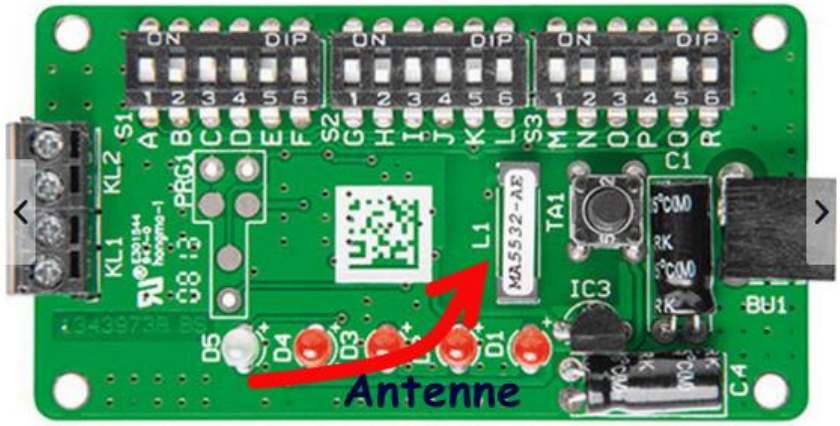
Im Serial Monitor sieht man, dass ohne ein Blitzereignis (Strikecount=0) der Warnlevel kontinuierlich geringer wird, jedem erkannten Blitz-Ereignis (Strikecount=255) wird der Warnlevel höher gesetzt wird.

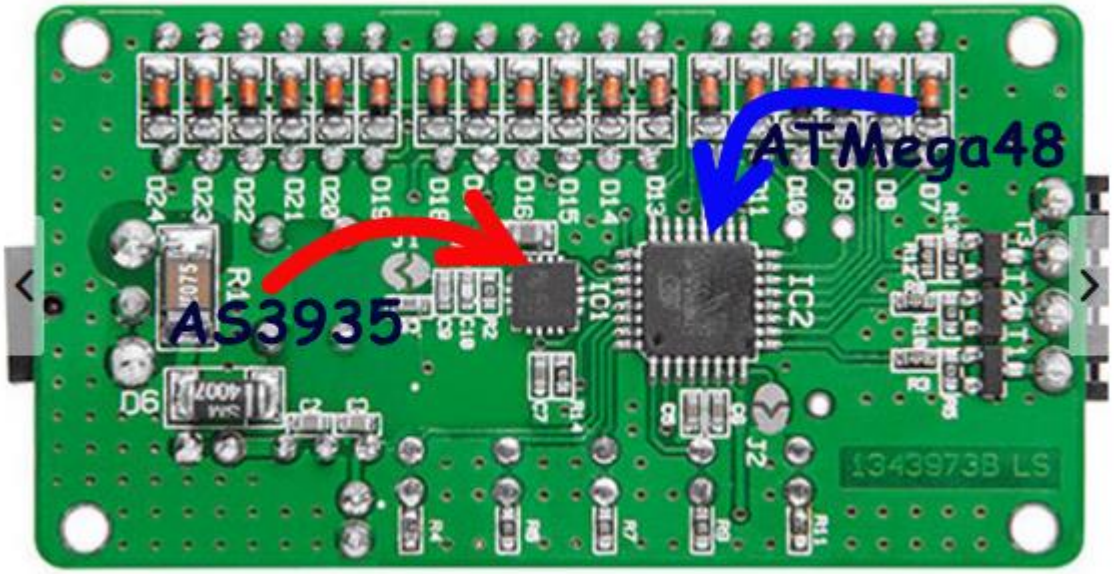
In der Software sind die Warnstufen-Werte wie folgt festgelegt:
< 2000 – keine Anzeige
>= 2000 → grüne LED blinkt bis <4000
>= 4000 → gelbe LED blinkt bis < 6000
Ab 6000 → rote LED blinkt (7000 möglicher maximaler Wert).

```
PWM:26  Actual:742  Averange:741.16  Difference:0  Warnlevel:2059  Decay:8  Strikecount:0
PWM:26  Actual:741  Averange:740.84  Difference:0  Warnlevel:2051  Decay:8  Strikecount:0
PWM:26  Actual:741  Averange:740.93  Difference:0  Warnlevel:2043  Decay:8  Strikecount:0
PWM:26  Actual:742  Averange:740.89  Difference:-1  Warnlevel:2290  Decay:8  Strikecount:255
PWM:26  Actual:741  Averange:740.94  Difference:0  Warnlevel:2536  Decay:9  Strikecount:255
PWM:26  Actual:1023  Averange:991.16  Difference:-31  Warnlevel:2781  Decay:10  Strikecount:255
PWM:26  Actual:1023  Averange:1023.00  Difference:0  Warnlevel:2771  Decay:10  Strikecount:0
PWM:26  Actual:1023  Averange:507.13  Difference:-515  Warnlevel:3015  Decay:11  Strikecount:255
PWM:26  Actual:1023  Averange:539.57  Difference:-483  Warnlevel:3258  Decay:12  Strikecount:255
PWM:26  Actual:0  Averange:507.02  Difference:507  Warnlevel:3500  Decay:13  Strikecount:255
PWM:26  Actual:1023  Averange:550.49  Difference:-472  Warnlevel:3741  Decay:14  Strikecount:255
```

```
if (WarnLevel < 2000){                    // No
  digitalWrite(LED_green, LOW);
  digitalWrite(LED_yellow, LOW);
  digitalWrite(LED_red, LOW);
}
if ((WarnLevel >= 2000) and (WarnLevel < 4000)){ /
  digitalWrite(LED_green, Blink);
  digitalWrite(LED_yellow, LOW);
  digitalWrite(LED_red, LOW);
}
if ((WarnLevel >= 4000) and (WarnLevel < 6000)){ /
  digitalWrite(LED_green, LOW);
  digitalWrite(LED_yellow, Blink);
  digitalWrite(LED_red, LOW);
}
if (WarnLevel >= 6000){                   // Red
  digitalWrite(LED_green, LOW);
  digitalWrite(LED_yellow, LOW);
  digitalWrite(LED_red, Blink);
}
if (WarnLevel >= 7000){                   // Max
  WarnLevel = 7000;
```

ELV Gewitterwarner verwendet den Sensor AS3935 Franklin Blitzdetektor (ca. 20-40.-€)

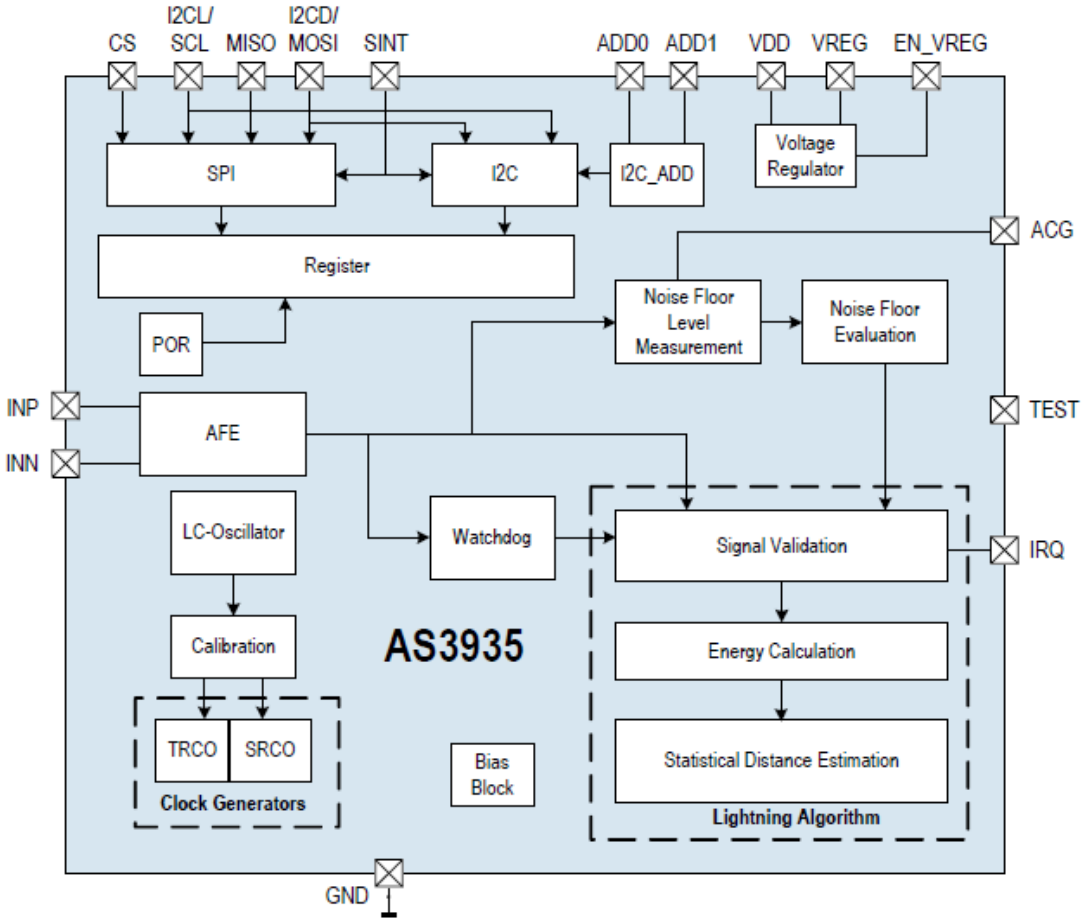




Das Modul ist mit dem Frankin AMS AS3935-System und einer speziellen Coilcraft MA5532-AE-Antenne ausgestattet,
um die Entfernung von Blitzentladungen,
Intensität und Frequenz innerhalb eines
Radius von 40 km im Innen- oder
Außenbereich zu bestimmen.

https://botland.de/schwerkraft-stromsensoren/13752-dfrobot-gravity-lightning-sensor-sensor-blitzdetektor-5903351243773.html

AS3935-Schaltkreis (aus AS3935-Datenblatt):
Die Abbildung zeigt ein Blockdiagramm des AS3935. Die äußere Antenne ist direkt mit dem **Analog Front-End (AFE)** verbunden, der das empfangene Signal verstärkt und demoduliert. Der **Watchdog** überwacht kontinuierlich den Ausgang des AFE und alarmiert den integrierten **Lightning-Algorithmus-Block** im Falle von einem eingehenden Signal. Der Lightning-Algorithmusblock **validiert** das Signal durch Überprüfung des **Signalmusters**. Es ist dazu in der Lage, zwischen Signalen zu unterscheiden, die durch Blitzeinschläge verursacht werden und Signale, die durch künstliche Lärmquellen, sogenannte Störer, verursacht werden.
Falls das Signal als vom Menschen verursachter Störer eingestuft wird, wird das **Ereignis abgewiesen** und der Sensor geht automatisch wieder zum Abhör-Modus über.
Wird das Ereignis als **Blitzeinschlag** eingestuft, führt der **statistische**



Entfernungsschätzungsblock eine Schätzung der Entfernung bis zur Gewitterfront durch.
Der LC-Oszillator kann zusammen mit dem Kalibrierungsblock sowohl den TRCO- als auch den SRCO-Taktgenerator kalibrieren und kompensiert so Prozessschwankungen.

Zu vermeidende Lärmquellen:

Die folgenden Geräuschquellen können beim AS3935 leicht zu Störungen führen, ein Ereignis auslösen und sind zu vermeiden:

- Induktor basierte DC-DC-Wandler. Die AS3935-Antenne ist von Magnetfeldern schwierig abzuschirmen die von solchen Konvertern erzeugt werden, es sei denn, man schließt den Konverter vollkommen in ein Metallgehäuse ein.
- Smartphone- und Smartwatch-Displays
- Betreiben des SPI bei 500 kHz, d. h. der Resonanzfrequenz des AS3935

Hauptmerkmale

- Blitzdetektor warnt vor Gewitteraktivität in einem Radius von 40km
- Abschätzung der Entfernung zur Gewitterfront bis zu 1 km in 14 Stufen
- Erkennt sowohl Wolke-zu-Boden als auch Intra-Wolke (Wolke-zu-Wolke) Blitze
- Eingebetteter Algorithmus zur Unterdrückung von künstlichen Störern
- Programmierbare Erkennungsstufen ermöglichen die Einstellung von Schwellenwerten für optimale Kontrollen
- SPI- und I^2C-Schnittstelle für die Steuerung und das Auslesen von Registern
- Antennenabstimmung zur Kompensation von Schwankungen der externen Komponenten
- Versorgungsspannungsbereich 2,4V bis 5,5V
- Abschalt-, Mithör- und Aktiv-Modus

Spezifikation des DFRobot Gravity - Lightning Sensor-Moduls

- https://wiki.dfrobot.com/Gravity%3A%20Lightning%20Sensor%20SKU%3A%20SEN0290

Der Lightning Sensor verwendet den innovativen **Blitzsensor-IC AS3935 Franklin von AMS** und die spezielle Antenne **Coilcraft MA5532-AE**, um die Entfernung, Intensität und Frequenz von Blitzen in einem Umkreis von 40 km sowohl im Innen- als auch im Außenbereich zu erkennen.

Der eingebettete Algorithmus zur **Unterdrückung künstlicher Störungen** kann die von verschiedenen Haushaltsgeräten erzeugten elektrischen Störungen effektiv vermeiden. Dank seiner kompakten Größe und dem großen Erfassungsbereich ermöglicht es Allgemeinwetterbegeisterten nicht nur die einfache und effiziente Messung lokaler Gewitterdaten, sondern kann auch in verschiedene intelligente tragbare Geräte für **Outdoor-Kletterer** oder Personen, die in der Höhe arbeiten, eingebettet werden. Dies ermöglicht eine frühzeitige und für den Menschen wahrnehmbare Warnung vor Gewittern, so dass Menschen frühzeitig vorbeugen können

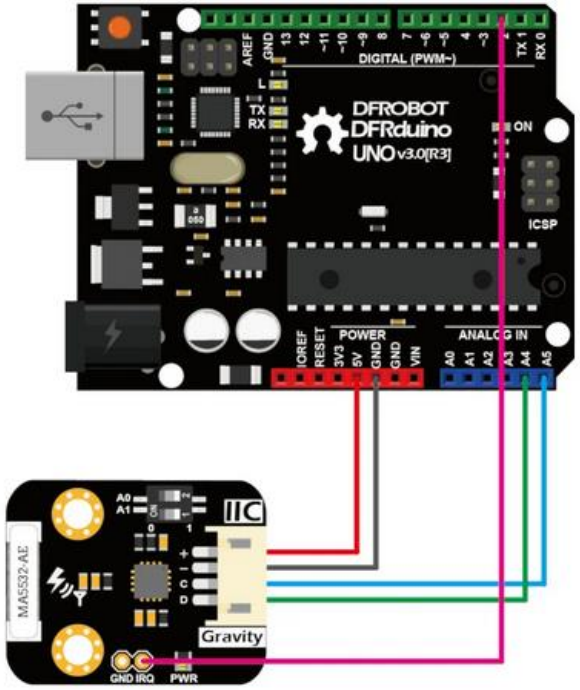


Diagramm zum Verbinden des Moduls mit Arduino .

Anhang:

Arduino-Sketch für das Blitz-o-Shield (Ramser-Electro)

```
// Firmware for Blitz-o-shield Rev.1
// Firmwareversion: Rev.0
// Copyright F.R. 2019
// www.ramser-elektro.at
//
//

const bool SendFullDebugText = true;        // Send debug message over RS232?
const word RemoteImpulseTime = 000;         // Set how long the remote impulse will be

int LED_red = 12;
int LED_yellow = 10;
int LED_green = 9;
int LED_strike = 13;

int PWM_pin = 11;
int REMOTE_pin = 7;

int AnalogIn = A0;

word ActValue;                      // Actual value of analog signal
float AverageValue;                 // Averange value
int Difference;                     // Difference value (AverageValue - ActValue)

word StrikeCount;                   // Actual strikecounter. Every strike 32 is added.
word WarnLevel = 255;               // Actual warning level
```

```
word Decay;                                     // Value, how much the warnlevel is decreased, if no activity

word InactivityTimer;                   // Timer for new PWM tune, if no activity

bool Blink;                             // Helper for LED blinking

unsigned long PreviousRemoteMillis = 0;       // Helper for remote impulse

word PWM_DutyCycle;                     // Actual duty cycle for PWM
word PWM_Duty_Watchdog;                     // Helper for PWM Adjustment

unsigned long PreviousMillis = 0;           // Helper for 1 second cycle

void setup(){                           // Assign the hardware setup
  pinMode(LED_red, OUTPUT);
  pinMode(LED_yellow, OUTPUT);
  pinMode(LED_green, OUTPUT);
  pinMode(LED_strike, OUTPUT);
  pinMode(PWM_pin, OUTPUT);
  pinMode(REMOTE_pin, OUTPUT);
  Serial.begin(9600);
  analogReference(INTERNAL);                // Set the reference voltage to 1.1 volt
  TunePWM();
}

void TunePWM(){                         // Setup the PWM and tune it
  if (SendFullDebugText){
    Serial.print("PWM duty cycle tuning started");
    Serial.print("\r\n");
  }
  PWM_DutyCycle = 0;                    // First discharge the capaciators
  analogWrite(PWM_pin,PWM_DutyCycle);
```

```
digitalWrite(LED_green, HIGH);              // Set green LED = capaciator discharged
delay(250);

ActValue = analogRead(AnalogIn);            // Read dummy
PWM_Duty_Watchdog = 0;                      // Reset watchdog

while (ActValue < 1023){                    // Has analogue input reached the maximum?
  ActValue = analogRead(AnalogIn);          // Read analog input
  AverageValue = ActValue;                  // Preset avarange value
  PWM_DutyCycle = PWM_DutyCycle +1;         // Increase PWM level
  PWM_Duty_Watchdog = PWM_Duty_Watchdog +1; // Increase the PWM watchdog
  analogWrite(PWM_pin,PWM_DutyCycle);
  delay(250);                               // Always wait a little time, to load the capacitors
  if (ActValue > 341){                      // Set yellow LED = ActValue  > 1/3 of maximum.
    digitalWrite(LED_yellow, HIGH);
  }
  if (ActValue > 682){                      // Set red LED = ActValue  > 2/3 of maximum.
    digitalWrite(LED_red, HIGH);
  }
  if (PWM_Duty_Watchdog >= 255){            // Maximum PWM duty cycle reached. Something is wrong!!
    Blink = !Blink;                         // LED Blink Helper
    if (SendFullDebugText){
    Serial.print("Problem while adjust PWM Duty cycle. Maximum reached!!!");
    Serial.print("\r\n");
    }
    digitalWrite(LED_green, Blink);         // Blink all LED in endless loop
    digitalWrite(LED_yellow, Blink);
    digitalWrite(LED_red, Blink);
    digitalWrite(LED_strike, Blink);
    delay(500);
  }
  ActValue = analogRead(AnalogIn);          // Read analog input
```

```
    AverageValue = ActValue;                // Preset avarange value
   }

   PWM_DutyCycle = (PWM_DutyCycle /3 ) * 2;
   analogWrite(PWM_pin,PWM_DutyCycle);
   digitalWrite(LED_strike, HIGH);          // Set white LED = Tune PWM successful
   delay(1000);
   digitalWrite(LED_green, LOW);
   digitalWrite(LED_yellow, LOW);
   digitalWrite(LED_red, LOW);
   digitalWrite(LED_strike, LOW);
   if (SendFullDebugText){
     Serial.print("PWM duty cycle tune successfull");
     Serial.print("\r\n");
   }
}

void loop(){

   ActValue = analogRead(AnalogIn);          // Read in actual value
   AverageValue = AverageValue * 15;         // Calculate the averange value
   AverageValue = AverageValue + ActValue;
   AverageValue = AverageValue / 16;
   Difference = AverageValue - ActValue;

if (Difference >= 15){                      // STRIKE !!!
   digitalWrite(LED_strike, HIGH);
   digitalWrite(REMOTE_pin, HIGH);          // Set remote action
   PreviousRemoteMillis = millis();
   StrikeCount = StrikeCount + 32;
   if (StrikeCount >= 255){                 //  More then 8 Strikes in 1 Second ?
     StrikeCount = 255;                     //  More then 8 strikes per second don't happen naturaly
```

```
  }
}
else{
  digitalWrite(LED_strike, LOW);
}

if ((millis() - PreviousRemoteMillis >= RemoteImpulseTime) and (digitalRead(REMOTE_pin))){  // Remote impulse time elapsed
  digitalWrite(REMOTE_pin, LOW);
}

if (millis() - PreviousMillis >= 1000){       // 1 Second elapsed
  Blink = !Blink;                             // LED Blink Helper

  if (StrikeCount > 32){                      // At leased 2 strikes in one second
    WarnLevel = WarnLevel + StrikeCount;      // Increase Warnlevel
  }

  Decay = highByte(WarnLevel);
  WarnLevel = WarnLevel - Decay;                                  //  Level decreasing

  if (WarnLevel < 2000){                      // No LED on
    digitalWrite(LED_green, LOW);
    digitalWrite(LED_yellow, LOW);
    digitalWrite(LED_red, LOW);
  }
  if ((WarnLevel >= 2000) and (WarnLevel < 4000)){ // Green LED on
    digitalWrite(LED_green, Blink);
    digitalWrite(LED_yellow, LOW);
    digitalWrite(LED_red, LOW);
  }
  if ((WarnLevel >= 4000) and (WarnLevel < 6000)){ // Yellow LED on
    digitalWrite(LED_green, LOW);
```

```
  digitalWrite(LED_yellow, Blink);
  digitalWrite(LED_red, LOW);
}
if (WarnLevel >= 6000){                  // Red LED on
  digitalWrite(LED_green, LOW);
  digitalWrite(LED_yellow, LOW);
  digitalWrite(LED_red, Blink);
}
if (WarnLevel >= 7000){                  // Maximum reached
  WarnLevel = 7000;
}

if (SendFullDebugText == true){            // Send debugmessage over RS232
  // PWM;ActValue;AverangeValue;Difference;Warnlevel;Decay;Strikecount;
  Serial.print("PWM:");
  Serial.print(PWM_DutyCycle);
  Serial.print("  Actual:");
  Serial.print(ActValue);
  Serial.print("  Averange:");
  Serial.print(AverageValue);
  Serial.print("  Difference:");
  Serial.print(Difference);
  Serial.print("  Warnlevel:");
  Serial.print(WarnLevel);
  Serial.print("  Decay:");
  Serial.print(Decay);
  Serial.print("  Strikecount:");
  Serial.print(StrikeCount);
  Serial.print("\r\n");
}

if (Decay == 0){                         // No activity. Increase InactivityTimer
```

```
    InactivityTimer = InactivityTimer +1;
    if (InactivityTimer >= 3600){            // No activity for one hour. Tune PWM.
      TunePWM();
      InactivityTimer = 0;
    }
  }

  StrikeCount = 0;                      // Reset strikecounter
  PreviousMillis = millis();              // Save actual millis, for the next 1 second cycle
}

}
```

Arduino Sketch für DFRobot_AS3935

Sample Code

- ***Connect the module to the Arduino according to the connection diagram. The I2C address defaults to 0x03, which corresponds to "AS3935_ADD3" in the code. If you need to modify the I2C address, you can change it to 0x01 or 0x02 through the DIP switch on the module, and modify the macro definition of the I2C address in the software "#define AS3935_I2C_ADDR AS3935_ADDx", where x can be 1, 2, 3.***
- ***Install DFRobot_AS3935 library.***
- ***Open Arduino IDE, upload the following sample code to the Arduino UNO.***
- ***Open the serial monitor of the Arduino IDE and set the baud rate to 115200.***

https://wiki.dfrobot.com/Gravity%3A%20Lightning%20Sensor%20SKU%3A%20SEN0290

```
/*!
 * @file DFRobot_AS3935_lightning_sensor_ordinary.ino
 * @brief SEN0290 Lightning Sensor
 * @n This sensor can detect lightning and display the distance and intensity of the lightning within 40 km
 * @n It can be set as indoor or outdoor mode.
 * @n The module has three I2C, these addresses are:
 * @n  AS3935_ADD1  0x01   A0 = 1  A1 = 0
 * @n  AS3935_ADD2  0x02   A0 = 0  A1 = 1
 * @n  AS3935_ADD3  0x03   A0 = 1  A1 = 1
 * @copyright   Copyright (c) 2010 DFRobot Co.Ltd (http://www.dfrobot.com)
 * @license     The MIT License (MIT)
 * @author [TangJie](jie.tang@dfrobot.com)
 * @version  V1.0.2
 * @date  2019-09-28
 * @url https://github.com/DFRobor/DFRobot_AS3935
 */

#include "DFRobot_AS3935_I2C.h"

volatile int8_t AS3935IsrTrig = 0;

#if defined(ESP32) || defined(ESP8266)
#define IRQ_PIN       0
#else
#define IRQ_PIN       2
#endif

// Antenna tuning capcitance (must be integer multiple of 8, 8 - 120 pf)
#define AS3935_CAPACITANCE   96

// Indoor/outdoor mode selection
#define AS3935_INDOORS       0
```

```
#define AS3935_OUTDOORS      1
#define AS3935_MODE          AS3935_INDOORS

// Enable/disable disturber detection
#define AS3935_DIST_DIS      0
#define AS3935_DIST_EN       1
#define AS3935_DIST          AS3935_DIST_EN

// I2C address
#define AS3935_I2C_ADDR      AS3935_ADD3

void AS3935_ISR();

DFRobot_AS3935_I2C  lightning0((uint8_t)IRQ_PIN, (uint8_t)AS3935_I2C_ADDR);

void setup()
{

  Serial.begin(115200);
  Serial.println("DFRobot AS3935 lightning sensor begin!");

  while (lightning0.begin() != 0){
    Serial.print(".");
  }
  lightning0.defInit();

  #if defined(ESP32) || defined(ESP8266)
    attachInterrupt(digitalPinToInterrupt(IRQ_PIN),AS3935_ISR,RISING);
  #else
    attachInterrupt(/*Interrupt No*/0,AS3935_ISR,RISING);
  #endif
```

```
  // Configure sensor
  lightning0.manualCal(AS3935_CAPACITANCE, AS3935_MODE, AS3935_DIST);
  // Enable interrupt (connect IRQ pin IRQ_PIN: 2, default)

//  Connect the IRQ and GND pin to the oscilloscope.
//  uncomment the following sentences to fine tune the antenna for better performance.
//  This will dispaly the antenna's resonance frequency/16 on IRQ pin (The resonance frequency will be divided by 16 on this pin)
//  Tuning AS3935_CAPACITANCE to make the frequency within 500/16 kHz ± 3.5%
//  lightning0.setLcoFdiv(0);
//  lightning0.setIRQOutputSource(3);

}

void loop()
{
  // It does nothing until an interrupt is detected on the IRQ pin.
  while (AS3935IsrTrig == 0) {delay(1);}
  delay(5);

  // Reset interrupt flag
  AS3935IsrTrig = 0;

  // Get interrupt source
  uint8_t intSrc = lightning0.getInterruptSrc();
  if (intSrc == 1){
    // Get rid of non-distance data
    uint8_t lightningDistKm = lightning0.getLightningDistKm();
    Serial.println("Lightning occurs!");
    Serial.print("Distance: ");
    Serial.print(lightningDistKm);
    Serial.println(" km");
```

```
    // Get lightning energy intensity
    uint32_t lightningEnergyVal = lightning0.getStrikeEnergyRaw();
    Serial.print("Intensity: ");
    Serial.print(lightningEnergyVal);
    Serial.println("");
  }else if (intSrc == 2){
    Serial.println("Disturber discovered!");
  }else if (intSrc == 3){
    Serial.println("Noise level too high!");
  }
}
//IRQ handler for AS3935 interrupts
void AS3935_ISR()
{
  AS3935IsrTrig = 1;
}
```

Gewitterwarner Fabian Kainka (Blitzlogger)

```
#include <EEPROM.h>

byte  A, Trigger, Sta;
word U, D;
word Daten[130];

byte B,J, I;
word Eeadr, Level, N;
```

```
word Ticks;

int Out1 = 3;
int Out2 = 2;

void setup(){
 pinMode(Out1, OUTPUT);
 pinMode(Out2, OUTPUT);

 Serial.begin(9600);
 //ADCSRA |= PS_16; // 8 prescaler
 Serial.println ("Logger");
 U = analogRead(A0);
 pinMode(13, OUTPUT);
 //digitalWrite(13, HIGH);

 Eeadr = 0;
 A = 1;
 Sta = 0;
 Trigger= 0 ;
 Level = 0;
}

void loop(){
  D =  analogRead(A0);
   Daten[A] = D;
   U = U *15;
   U = U + D;

   U = U /16;

  if (Trigger == 1){
```

```
  if (Sta == A){
    Serial.println("Blitz!");
      for (N = 1; N <= 32; N++){
       A = A + 1;
       if (A > 128){ A = 1;}
       D = Daten[A];
       D = D /2;
       D = D & 255;
       B = D;
       Serial.println (B);

       if (Eeadr < 512) { EEPROM.write(Eeadr, B);}
       Eeadr++;
       digitalWrite(13, LOW);
     }
     I = I +1;
     Sta = 0;
     Trigger = 0;
     Level = Level + 600;
     if (Level > 6000) {Level = 6000;}
  }

  A = A + 1;
  if(A > 128){A = 1;}
}
else{
  D = D + 15;
  if (D < U){
    Trigger = 1;
    Sta = A + 120;
    if(Sta > 128){Sta = Sta - 128;}
    digitalWrite(13, HIGH);
```

```
  }
  A = A +1;
  if (A > 128){A = 1;}
  if (Serial.available() > 0) {
      J = Serial.read();
      if (J == 100){
          B = 0;
        for(N = 0; N<512;N++){
          EEPROM.write (N,B);
        }
          Eeadr = 0 ;
          Serial.write("Daten Gelöscht");
      }
      else{
        Eeadr = J *128;
        for (N = 1; N<= 128; N++){
          B =  EEPROM.read (Eeadr);
          Serial.write (B);
          Eeadr ++;
        }
        Eeadr = 0;
      }
    }

}

Ticks++;
if (Ticks >= 22000) {
    Ticks = 0;
    if (Level > 0) {
        Level--;
        if (Level >2000){ Level = Level -9;}
```

```
    }
    if (Level > 2000){digitalWrite(Out2, HIGH);}
      else {digitalWrite(Out2, LOW); }
    if (Level > 1000){  digitalWrite(Out1, HIGH);}
      else {digitalWrite(Out1, LOW);}
  }


}




>EOF>
```