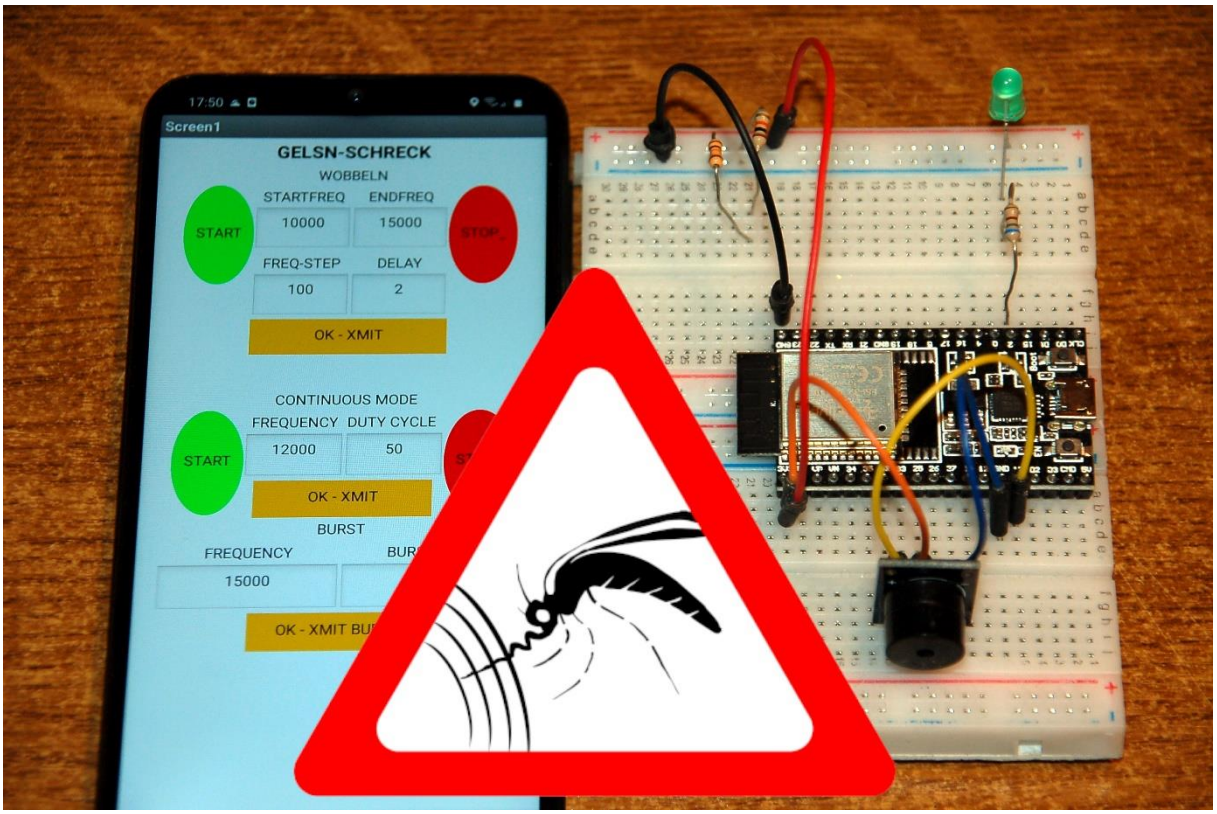

Abbildung 1: Titel

Die Insekten-Fauna nimmt immer mehr ab, hören wir immer wieder. Gut- wenn ich die Bestückung des Sichtschilds meines Motorradhelms nach einer Nachtfahrt heute mit der vor 10 Jahren vergleiche, mag das vielleicht stimmen. Schön wäre es aber dennoch, wenn man nach einem Abend im Freien nicht am nächsten Tag 20 oder mehr Mückenstiche bekratzen müsste. Von den Plagegeistern gibt es offenbar immer noch viel zu viele. Manche Leute schwören auf die Bekämpfung der Biester um die Sitzgruppe herum mit Ultraschall. Gemeint sind Frequenzen oberhalb des menschlichen Hörbereichs, also so ab 20kHz. Die Schaltung, die ich heute vorstelle, zusammen mit einer App für ein Android-Handy, das ich als Steuerung einsetzen werde (nächste Folge), könnte also für Abhilfe gegen die Mücken-Mafia sorgen. Die Töne aus dieser Schaltung können aber auch Nagetiere mit langen Schwänzen, Hunde, Katzen und anderes Getier vergraulen. Der Aufbau ist dank der wenigen Bauteile sehr übersichtlich und daher für Neueinsteiger bestens geeignet. Damit herzlich willkommen zum Blog mit dem Titel

Mücken- und Marderschrecker mit dem ESP32 und MicroPython

Gelsen (oder Gelsn sprich: "Göisn") bezeichnet im niederbayrischen und österreichischen Sprachgebrauch die Mücke. Selbige Mistviecher können einem den Abend im Freien ganz schön vermiesen. Man kann sich mit Mückenspray einduften, bis zum Hals in Schutzkleidung verpacken oder einfach die Ultraschallkanone gegen den Angriff klar machen. Genau Letzteres ist der Inhalt dieser Beschreibung. Das erfordert natürlich einen gewissen Hard- und Softwareeinsatz. Für die vier möglichen Ausbaustufen folgt hier die Liste der Teile.

Hardwarekomponenten

	Grundaufbau
1	ESP32 Lolin LOLIN32 oder ESP32 D1 Mini NodeMCU WiFi Modul
1	KY-006 Passiver Piezo Buzzer Alarm Modul
1	LED (grün)
1	Widerstand 680 Ω
2	Widerstand 10 kΩ
2	Mini Breadboard 400 Pin mit 4 Stromschienen für Jumper Kabel
diverse	Jumperkabel
	Ausbaustufe 1
1	CD40106 sechsfach Inverter, integrierter CMOS-Schaltkreis
1	Transistor BC550 o. ä.
1	Widerstand 2,2 kΩ
1	Widerstand 10 kΩ
1	12 V-Spannungsquelle
	Ausbaustufe 2
1	Hochtonhorn 105 dB-110 dB, 4000..40000 Hz
	Ausbaustufe 3
1	MT3608 DC-DC Netzteil Adapter Step up Modul
	Verstärker mit H-Brücke aus folgenden Teilen
2	BD135
2	BD136
3	BC550
1	Widerstand 1 kΩ
1	Widerstand 1,5 kΩ
1	Widerstand 10 kΩ
2	Widerstand 2,7 kΩ
2	Widerstand 10 Ω
2	Widerstand 820 Ω
1	Elektrolytkondensator 100 µF / 16 V
1	Elektrolytkondensator 470 µF / 35 V
1	Kühlkörper ca. 30 x 70 mm
4	Silikon- oder Glimmerscheiben
4	Schrauben M3x10 + Muttern
1	Platine 60 x 100mm, einseitig kaschiert oder Lochrasterplatine

Die Software

Fürs Flashen und die Programmierung des ESP32:
Thonny oder
µPyCraft
packetsender zum Testen des ESP32 als UDP-Server

Verwendete Firmware:

MicropythonFirmware
Bitte eine Stable-Version aussuchen

Die MicroPython-Programme:
http://www.grzesina.de/az/gelsenschreck/gelsnschreck_test.py zum Testen via REPL-Eingabe-Aufforderung

MicroPython - Sprache - Module und Programme

Zur Installation von Thonny finden Sie hier eine ausführliche Anleitung. Darin gibt es auch eine Beschreibung wie die MicropythonFirmware auf den ESP-Chip gebrannt wird.

MicroPython ist eine Interpretersprache. Der Hauptunterschied zur Arduino-IDE, wo Sie stets und ausschließlich ganze Programme flashen, ist der, dass Sie die MicroPython-Firmware nur einmal zu Beginn auf den ESP32 flashen müssen, bevor der Controller MicroPython-Anweisungen versteht. Sie können dazu Thonny, µPyCraft oder esptool.py benutzen. Für Thonny habe ich den Vorgang hier beschrieben.

Sobald die Firmware geflasht ist, können Sie sich zwanglos mit Ihrem Controller im Zwiegespräch unterhalten, einzelne Befehle testen und sofort die Antwort sehen, ohne vorher ein ganzes Programm compilieren und übertragen zu müssen. Genau das stört mich nämlich an der Arduino-IDE. Man spart einfach enorm Zeit, wenn man einfache Tests der Syntax und der Hardware bis hin zum Ausprobieren und Verfeinern von Funktionen und ganzen Programmteilen, über die Kommandozeile vorab prüfen kann, bevor man ein Programm daraus strickt. Zu diesem Zweck erstelle ich auch gerne immer wieder kleine Testprogramme. Als eine Art Macro fassen sie wiederkehrende Befehle zusammen. Aus solchen Programmfragmenten entwickeln sich dann mitunter ganze Anwendungen.

Autostart

Soll das Programm autonom mit dem Einschalten des Controllers starten, kopieren Sie den Programmtext in eine neu angelegte Blankodatei. Speichern Sie diese Datei unter boot.py im Workspace ab und laden Sie sie zum ESP-Chip hoch. Beim nächsten Reset oder Einschalten startet das Programm automatisch.

Programme testen

Manuell werden Programme aus dem aktuellen Editorfenster in der Thonny-IDE über die Taste F5 gestartet. Das geht schneller als der Mausklick auf den Startbutton oder über das Menü **Run**. Lediglich die im Programm verwendeten Module müssen sich im Flash des ESP32 befinden.

Zwischendurch doch mal wieder Arduino-IDE?

Sollten Sie den Controller später wieder zusammen mit der Arduino-IDE verwenden wollen, flashen Sie das Programm einfach in gewohnter Weise. Allerdings hat der ESP32/ESP8266 dann vergessen, dass er jemals MicroPython gesprochen hat. Umgekehrt kann jeder Espressif-Chip, der ein compiliertes Programm aus der Arduino-IDE oder die AT-Firmware oder LUA oder … enthält, problemlos mit der MicroPython-Firmware versehen werden. Der Vorgang ist immer so, wie hier beschrieben.

Der Aufbau des Mückenschreckers und die erste Inbetriebnahme

Die Basisversion

Die Grundstufe ist sehr übersichtlich mit den wenigen Teilen bestückt. Der einzige aktive Baustein ist der ESP32. Die Möglichkeit, dem Controller PWM-Signale bis zu (theoretischen) 40 MHz zu entlocken, brachte mich auf die Idee zu diesem Projekt. Die anvisierte Steuerung des Outputs der Schaltung kann über ein Androidgerät, oder neuerdings auch iPhone, mit Hilfe einer App erfolgen, die wir mit dem App-Inventor2 in der nächsten Blogfolge bauen werden. Auch dafür stellt der ESP32 alles bereit, was wir für den Funkverkehr brauchen. Ein ESP8266 schafft übrigens die hohen Frequenzen beim PWM nicht, deshalb muss es der große Bruder sein.

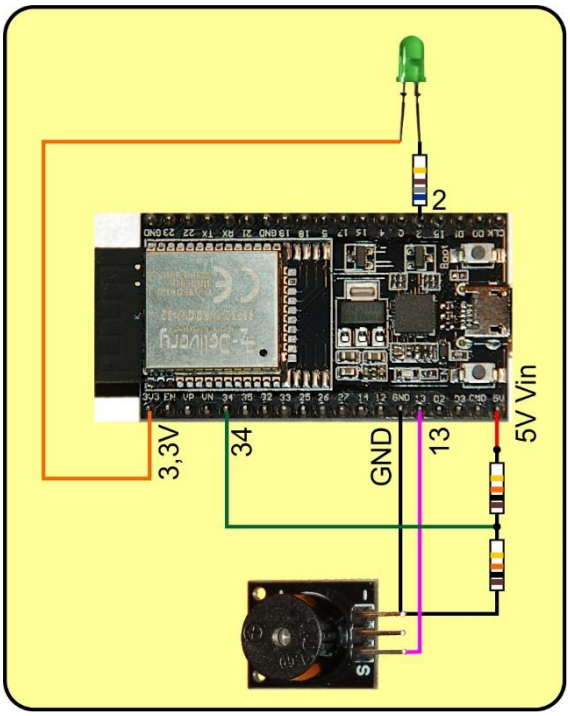

Abbildung 2: Gelsenschreck_Basisschaltung

Für erste Versuche ist die Basisversion der Schaltung sehr schnell zusammengesteckt. Um ihr auch gleich Töne entlocken zu können, habe ich im Programm gelsnschreck.py die Befehle versteckt, die für die Funkverbindung

zuständig sind. Somit müssen wir uns zunächst nicht darum kümmern. Statt dem gezwickten Text wurden zwei Zeilen eingefügt, welche die Bedienung über das USB-Kabel ermöglichen. Der Inputbefehl ersetzt fürs Erste die Funkübertragung. Nach dem Start des modifizierten Programms gelsnschreck_test.py blinkt die LED lang – kurz – kurz, das ist das Signal für die Einsatzbereitschaft.

Hier kommt schon einmal die Liste der möglichen Befehle für die drei Betriebsmodi Wobbeln, Dauerton- und Burstmodus.

Wobbeln ist die Erzeugung von Schwingungen, wobei die Frequenz zyklisch zwischen zwei einstellbaren Endwerten variiert.
Burst (=platzen) bedeutet eine begrenzte Anzahl von Schwingungen fester Frequenz.

Modus	**Befehl**	**Bemerkung**
Wobbeln	w:f:<Startfrequenz in Hz>	untere Frequenzgrenze ab der die Tonhöhe gesteigert wird.
	w:t:<Endfrequenz in Hz>	obere Grenze des Frequenzdurchlaufs
	w:s:<Frequenzzuwachs in Hz>	Steigerungsrate der Durchlauffrequenz
	w:d:<Verweildauer in ms>	Zeit während der die Frequenz beibehalten wird
	w:start	Ton einschalten
	w:stop	Ton ausschalten
Dauerton	c:f:<Frequenz in Hz>	Dauertonfrequenz
	c:d:<Dutycycle in %>	Verhältnis Puls zu Periodendauer – des Ausgangssignals
	c:start	Ton einschalten
	c:stop	Ton ausschalten
Burst	b:f:<Frequenz>	Frequenz des Frequenzpakets und Ton ein
	b:p:<Pulsdauer in s>	Dauer des Frequenzpakets

Diese Befehle werden in der Testversion des Programms über die PC-Tastatur eingegeben. Das ermöglicht es uns, Erweiterungen oder Änderungen des Programms schnell zu überprüfen, ohne nach jedem Neustart auf die Funkverbindung warten zu müssen.

Mit dem Programmstart werden folgende Parameter auf ihre Startwerte gesetzt.

```
startFreq=10000  # Startfrequenz Wobbeln
endFreq=20000    # Endfrequenz Wobbeln
step=10          # Zeitstufe Wobbeln in ms
delta=10         # Frequenzstufe beim Wobbeln
repeat=True      # Wobbeln wiederholen

freq=startFreq   # Continuous Freq.
duty=512         # Dutycycle in 1/1024; 512 entspricht 50%
pulse=5          # Burstdauer in s
```

Im Terminal erscheint folgende Startmeldung. Danach können die **Befehle** eingegeben werden.

ADC Initialized: 0
Socket established, waiting...
Kommando:**b:p:1**
from 10.0.1.230
Content = b:p:1
B-P:1
Kommando:**b:f:4000**
from 10.0.1.230
Content = b:f:4000
B-F:4000
Kommando:

Diese Kommandos erzeugen einen Frequenzpuls von 4000Hz und einer Sekunde Dauer.

Ausbaustufe 1

Das ist alles? Das hört man ja kaum. Stimmt schon, sehr laut war das nicht. Deshalb gibt es ja auch die Ausbaustufe 1. Mit 3,3V Steuersignal wird der Buzzer nur ein wenig am Bauch gekitzelt. Und deshalb spendieren wir der Schaltung jetzt einen kleinen Verstärkerzusatz in Form der sechs Inverter in dem CMOS-Baustein CD40106. Der wird mit 12V betrieben und formt außerdem die eher verwaschene Kurve durch die Schmitt-Triggerfunktionalität der Inverter zu einem sauberen Rechtecksignal.

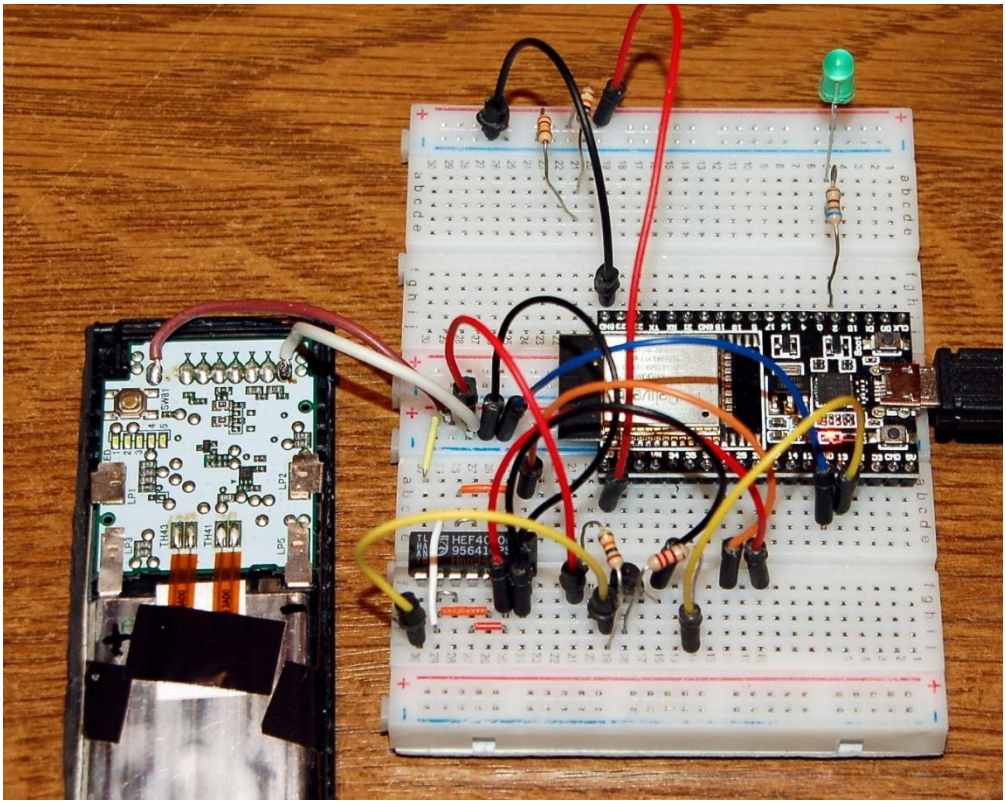

Abbildung 3: Ausbaustufe2 mit 12V-Li-Akku

So sieht das jetzt aus, und das steckt dahinter.

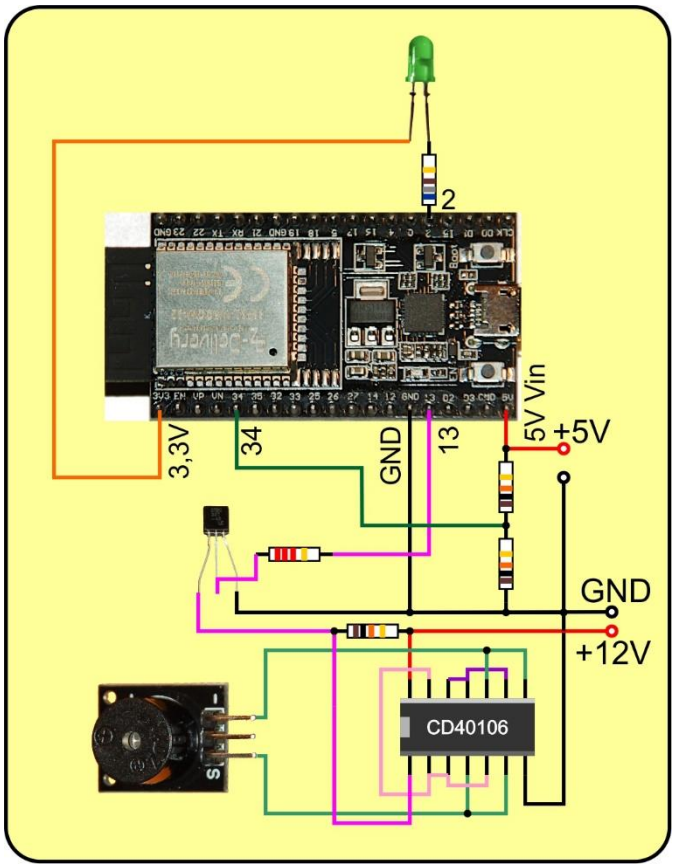

Abbildung 4: Ausbauversion 1

Sie wollen noch mehr Input – bitte, sehr gerne. Jedes der Dreiecke symbolisiert eine der sechs Schmitt-Triggerstufen. Der Punkt am Ausgang bedeutet, dass das Eingangssignal invertiert wird. Aus einer 1 am Eingang wird also eine 0 am Ausgang und umgekehrt. Diese Funktion nutzen wir genau einmal ganz gezielt. Die Einheit 13 / 12 kehrt das Signal, das von 1 / 2 kommt um, damit die nachfolgenden "Endstufen" im Gegentakt angesteuert werden.

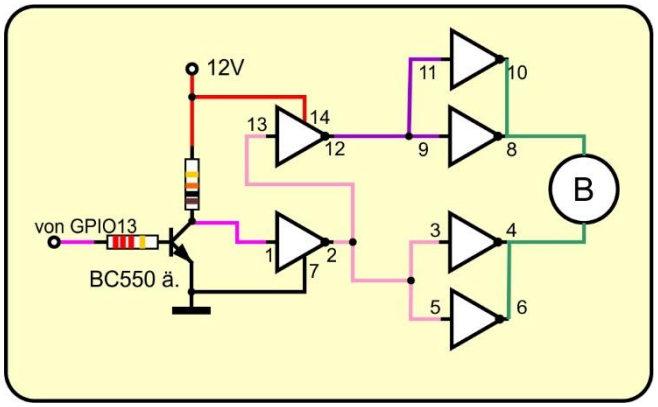

Abbildung 5: Gelsnschreck_CMOS-Verstärkerschaltung

Und damit Sie die Nummern an den Anschlüssen auch zuordnen können, folgt jetzt auch noch ein Blick ins Innenleben des CD40106.

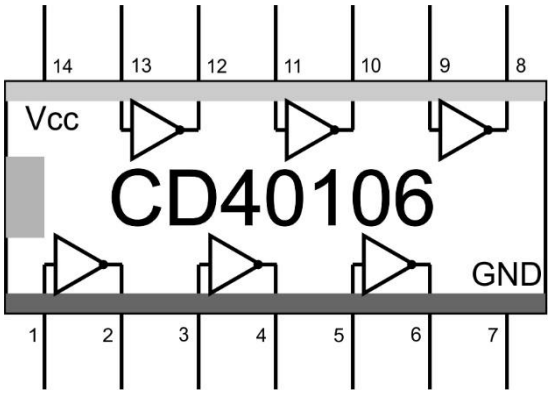

Abbildung 6: CMOS40106_Innenleben

Die schematische Darstellung eines Inverters offenbart uns, dass dessen Endstufe im Gegentakt arbeitet. Schaltet man zwei dieser Einheiten, gegenpolig angesteuert, gegeneinander, dann wird daraus eine Vollbrücke.

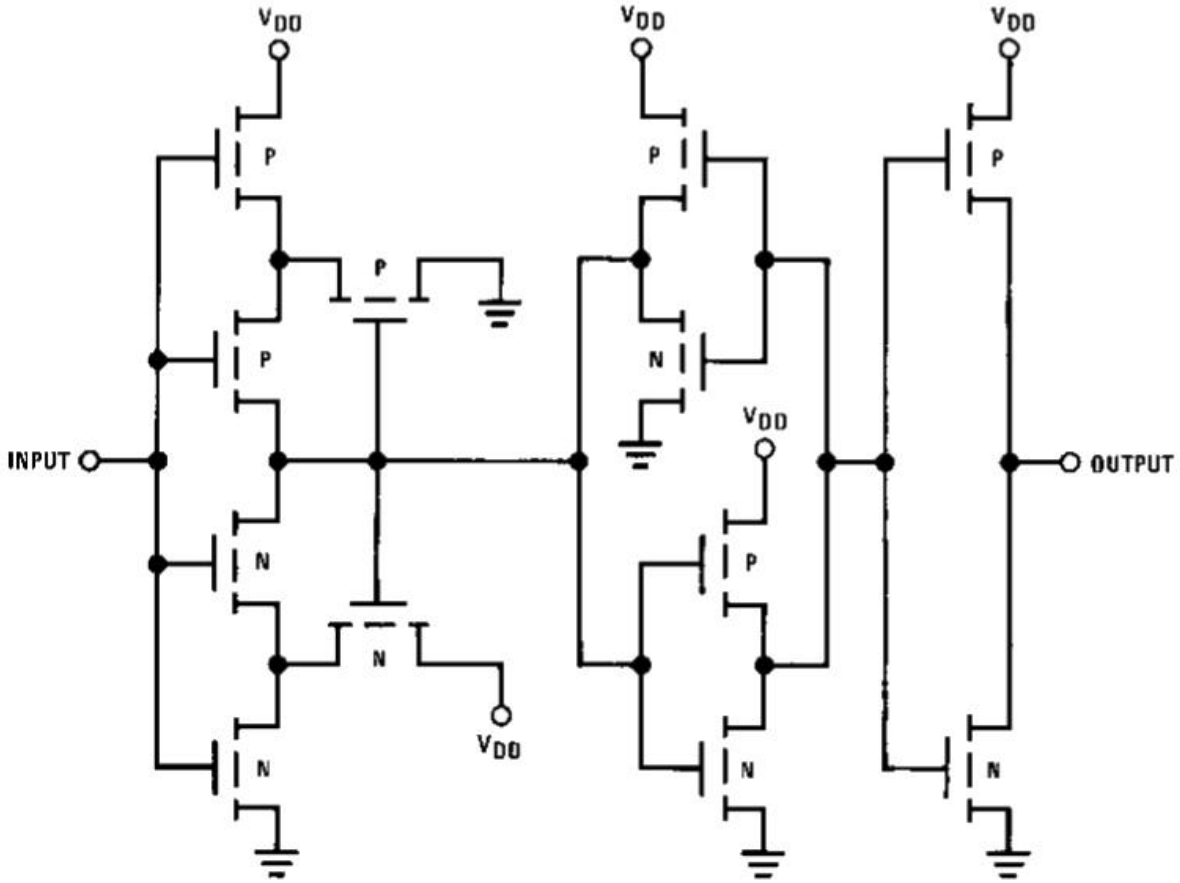

Abbildung 7: Inverterstufe

Betrachten wir die neuen Teile der Schaltung und klären wir deren Funktion. Der ESP32 liefert maximal 3,3V-Signalspitze. Das wäre als Logikpegel für eine 1 ausreichend (U_{LH} = 2,9V), wenn wir den 40106 an Vcc mit 5V-betreiben würden. Aber es reicht nicht für 12V-Betriebsspannung (U_{LH}>5,9V). Deshalb brauchen wir den Transistor für die Pegelanpassung. Der BC550 begnügt sich mit den 3,3V. Es fließt

dann genügend Basisstrom durch den 2,2 kΩ-Widerstand, damit der Transistor durchschaltet und den 10 kΩ-Widerstand auf GND-Potential zieht. Liegt an der Basis dagegen 0V vom ESP32 her an, dann sperrt der BC550 und am Kollektor liegen die 12V der Betriebsspannung.

Die Inverter arbeiten als Verstärker und Former, wobei mit jedem Durchgang das Signal invertiert wird. Diese Begleiterscheinung stört uns aber nicht, ist jedoch in einem Fall notwendig, das habe ich weiter oben schon angemerkt. Die erste Stufe putzt das Signal. Die beiden folgenden Einheiten in derselben Reihe sind parallelgeschaltet, das bringt uns mehr Leistung.

Das Signal aus der ersten Stufe wird in der oberen Etage erst einmal beabsichtigt invertiert und dann ebenfalls durch die beiden parallelgelegten Stufen verstärkt. An den Pins 6 und 8 erhalten wir eine Spannung, die nicht nur wie beim ESP32 ein- und ausgeschaltet wird, sondern hier wird umgepolt. Der Buzzer liegt also an den Mittelabgriffen der Halb-Brücken in den Endstufen der Inverter.

Ausbaustufe 2

Leider ist das Signal des Buzzers auch dadurch nicht überwältigend lauter geworden. Für die unmittelbare Umgebung reicht das vielleicht auch aus. Trotzdem können wir bedeutend mehr Sound herausholen, wenn wir statt dem Buzzer einen ausgewachsenen Lärmlümmel engagieren. Gemeint ist ein Hochtonhorn, das wie der Buzzer auf Piezobasis arbeitet, aber bedeutend mehr Phon abgibt.

Abbildung 8: Hochtonhorn

An der bisherigen Schaltung selbst braucht nichts verändert zu werden. Wir schließen einfach nur statt dem Buzzer das Horn an.

Habe ich zu viel versprochen? Testen Sie das doch einfach mal an Nachbars Wauzi, wenn der mit seinen Tretminen mal wieder Ihren Vorgarten verziert. Spielen Sie einfach mal ein bisschen mit der Frequenz. Das Horn macht locker bis 40 kHz mit, der ESP32 natürlich auch. 40kHz ist übrigens auch das Kommunikationsband der Fledermäuse. Zur Ortung von Fluginsekten und Hindernissen senden sie durch Rufe kurze Bursts von einigen Hundertstelsekunden in einer Folge von 5 bis 10 Paketen aus. Um diese Rufe für unsere Ohren hörbar zu machen, setzt man Fledermausdetektoren ein. Auf diese Geräte komme ich am Schluss dieser Folge noch kurz zu sprechen. Die Anwesenheit dieser schnellen Flugkünstler hilf uns übrigens auch, die ungeliebten Mücken und Motten zu dezimieren.

Nachdem nun die Funktion der Hardware klar ist, wollen wir doch auch das Programm näher untersuchen.

Wie die Töne entstehen

Das Programm gliedert sich im Wesentlichen in zwei Teile. Da ist zum einen die Funkverbindung via UDP, die Befehle von einem Client empfängt und zum anderen der Parser, der sie auf Syntax und Inhalt abklopft sowie die Ausführung veranlasst. Optional sendet der UDP-Server auch Nachrichten an den Client zurück.

Die Funkverbindung kann sowohl über einen vorhandenen WLAN-Accesspoint als auch über den hauseigenen Accesspoint des ESP32 erfolgen. Letzteres ist dort von Vorteil, wo kein lokales Netzwerk existiert.

Für den Test der UDP-Verbindung kann das Tool packetsender.exe hilfreich sein. Das Programm wird auf dem Windows-PC installiert und dient dort als UDP-Client. Damit können wir Befehle an den Server auf dem ESP32 senden, wie wir es ganz am Anfang über die Terminaleingabe getan haben. Auch die Rückmeldungen vom ESP32 werden hier angezeigt. Der Umgang mit dem Programm ist nicht schwierig und erfolgt intuitiv. In der folgenden Abbildung muss nur die IP-Adresse des ESP32 an Ihr Homenet angepasst werden, den Port können Sie beibehalten. Wichtig ist auch der Eintrag UDP:9181 in der Statuszeile (ganz unten). Anstatt der Portnummer 9181 können Sie natürlich eine beliebige andere im Bereich zwischen 1024 und 65535 wählen.

Befehle, die im oberen Drittel des Fensters eingegeben wurden, können gespeichert werden. Dadurch werden sie im Mittelteil aufgelistet und sind durch send erneut abrufbar. Im unteren Drittel sehen Sie, was Sie gesendet haben und was der ESP32 darauf antwortet.

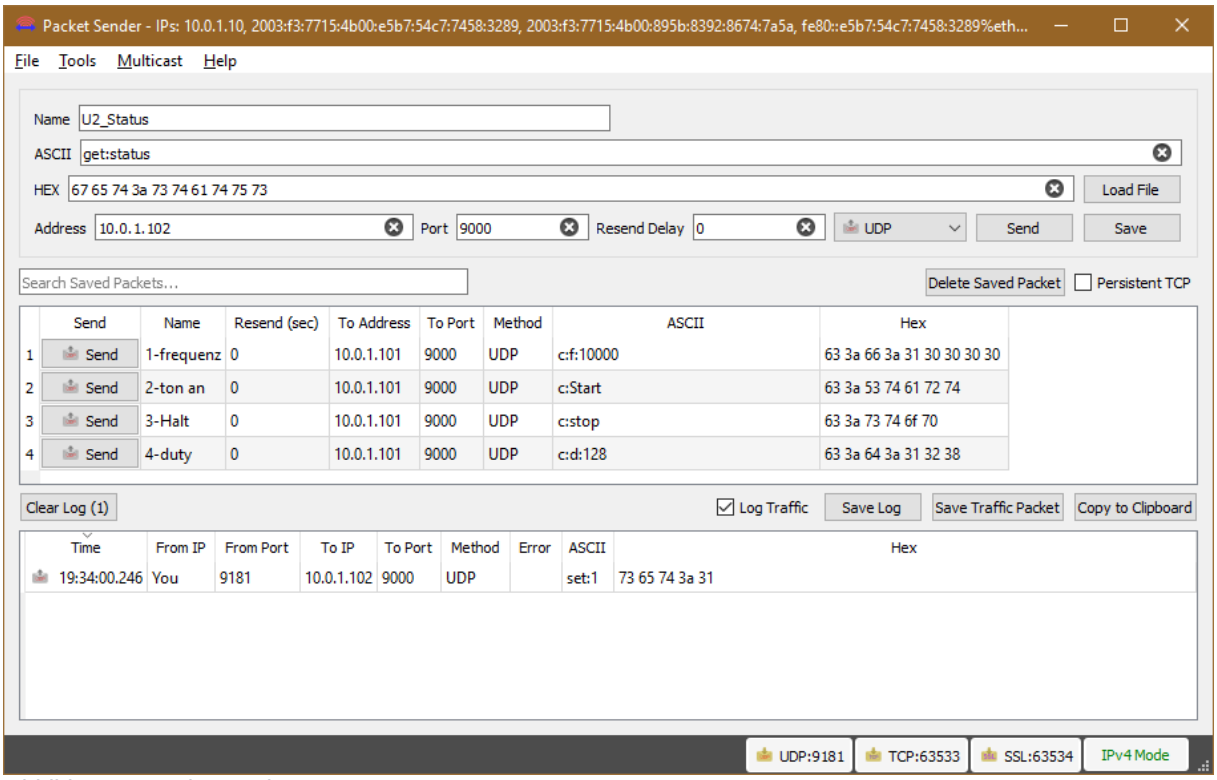

Abbildung 9: Packetsender

Jetzt aber zum Programm gelnschreck_test.py selbst.

```
#gelsnschreck_test.py
# ********************** Importgeschaeft ********************
import esp
esp.osdebug(None)
import os
import gc                  # Platz fuer Variablen schaffen
gc.collect()
try:
  import usocket as socket
except:
  import socket
import ubinascii
import network
from machine import ADC,Pin,I2C,PWM,Timer
from time import sleep,time,sleep_ms
import sys
#from bmp280 import BMP280
#from i2cbus import I2CBus
UbatPin=34
Ubat=ADC(Pin(UbatPin))
print("ADC Initialized: ",Ubat.read())
taste=Pin(0,Pin.IN)
blinkLed=Pin(2,Pin.OUT)
request = bytearray(160)
act=bytearray(30)
```

```
response=""
# Pintranslator fÃ¼r ESP8266-Boards
# LUA-Pins     D0 D1 D2 D3 D4 D5 D6 D7 D8
# ESP8266 Pins 16  5  4  0  2 14 12 13 15
#                 SC SD  FL L
i2c=I2C(-1,scl=Pin(21),sda=Pin(22))
#b=BMP280(i2c)

p = PWM(Pin(13))      # create PWM object from a pin
p.deinit()

startFreq=10000  # Startfrequenz Wobbeln
endFreq=20000    # Endfrequenz Wobbeln
step=10          # Zeitstufe Wobbeln in ms
delta=10         # Frequenzstufe beim Wobbeln
repeat=True      # Wobbeln wiederholen

freq=startFreq   # Continuous Freq.
duty=512         # Dutycycle in x/1023
pulse=5           # Burstdauer in s

T=Timer(0)

#********************Variablen deklarieren *******************
# Die Dictionarystruktur (dict) erlaubt  die Klartextausgabe
# des Verbindungsstatus anstelle der Zahlencodes
connectStatus = {
    1000: "STAT_IDLE",
    1001: "STAT_CONNECTING",
    1010: "STAT_GOT_IP",
    202:  "STAT_WRONG_PASSWORD",
    201:  "NO AP FOUND",
    5:    "GOT_IP"
    }

flag = 0
#******************Funktionen deklarieren ******************
def hexMac(byteMac):
  """
  Die Funktion hexMAC nimmt die MAC-Adresse im Bytecode
  entgegen und bildet daraus einen String fuer die Rueckgabe
  """
  macString =""
  for i in range(0,len(byteMac)):     # Fuer alle Bytewerte
    macString += hex(byteMac[i])[2:]  # ab Position 2 bis Ende
    if i <len(byteMac)-1 :            # Trennzeichen
      macString +="-"
  return macString

def blink(pulse,wait,inverted=False):
    if inverted:
        blinkLed.off()
```

```
        sleep(pulse)
        blinkLed.on()
        sleep(wait)
    else:
        blinkLed.on()
        sleep(pulse)
        blinkLed.off()
        sleep(wait)

def increment(tim):
    global freq,p
    freq=(freq+step if freq <= endFreq else startFreq)
    p.freq(freq)
    print(freq)

def shutOff(tim):
    p.deinit()

def parse(action):
    global p, startFreq,endFreq,step,duty,freq,repeat
    global delta,pulse
    art="E"
    act="_"
    value=-1
    action=action.upper()
    if action.find(":",1) !=-1 :
        art,rest=action.split(":",1)
        if art in ["W","C","B",]:
            if rest.find(":") != -1: # Befehl mit Parameter
                act,value=rest.split(":")
                value=int(value)
                if art=="W": # wobbeln
                    if act=="F": # Startfrequenz setzen
                        startFreq=value
                    elif act=="T": # Endfrequenz setzen
                        endFreq=value
                    elif act=="S": # Stufenzeit setzen
                        step=value
                    elif act=="D": # Stufenzeit setzen
                        delta=value
                elif art=="C":  # Continuous mode
                    if act=="F": # Frequenz setzen
                        freq=value
                        p.freq(freq)
                    elif act=="D": # Dutycycle setzen
                        duty=value
                        p.duty(duty)
                elif art=="B":
                    if act=="F": # Frequenz setzen und Start
                        freq=value
                        p.init()
                        p.duty(duty)
```

```
                        p.freq(freq)
                        T.init(mode=Timer.ONE_SHOT,period=\
                               pulse*1000,callback=shutOff)
                    elif act=="P": # Bustdauer setzen + Start
                        p.init()
                        p.duty(duty)
                        p.freq(freq)
                        pulse=value
                        T.init(mode=Timer.ONE_SHOT,period=\
                               pulse*1000,callback=shutOff)
            else: # Befehle ohne Parameter
                act=rest
                value=0
                if art=="W":
                    if act=="START": # wobbeln starten
                        p.init()
                        p.duty(duty)
                        p.freq(startFreq)
                        T.init(mode=Timer.PERIODIC,period=\
                               delta,callback=increment)
                    elif act=="STOP":
                        T.deinit()
                        p.deinit()
                elif art=="C":
                    if act=="START": # Dauerlauf starten
                        p.init()
                        p.duty(duty)
                        p.freq(freq)
                    elif act=="STOP":
                        p.deinit()
                else:
                    act="INVALID ACTION"
                    value=-3
        else:
            art=art+": ERROR"
            act="UNKNOWN COMMAND"
            value=-4
    else:
        if action=="E":
            print("ABGEBROCHEN DURCH USER")
            p.deinit()
            T.deinit()
            sys.exit()
        art=art+": ERROR"
        act="ARGUMENT MISSING"
        value=-5
    return(art,act,value)

# # *****************  Setup accesspoint ***************
#
# nic = network.WLAN(network.AP_IF)
# nic.active(True)
```

```
# ssid="unit1"
# passwd="uranium238"
#
# # Start als Accesspoint
#
nic.ifconfig(("10.0.2.101","255.255.255.0","10.0.2.101","10.0.
2.101"))
#
# print(nic.ifconfig())
#
# # Authentifizierungsmodi ausser 0 werden nicht unterstuetzt
# nic.config(authmode=0)
#
# MAC=nic.config("mac") # liefert ein Bytes-Objekt
# # umwandeln in zweistellige Hexzahlen ohne Prefix und in
String decodieren
# MAC=ubinascii.hexlify(MAC,"-").decode("utf-8")
# print(MAC)
# nic.config(essid=ssid, password=passwd)
#
# while not nic.active():
#   print(".",end="")
#   sleep(0.5)
#
# print("Unit1 listening")
# # *****************  Setup accesspoint end ***************

# # ***************  Setup Router connection **************
# mySid = mySid = 'YOUR_SSID'; myPass = "YOUR_PASSWORD"
# nic = network.WLAN(network.STA_IF)  #  erzeuge WiFi-Objekt
nic
# nic.active(True)  # Objekt nic einschalten
# #
# MAC = nic.config('mac')    # # binaere MAC-Adresse abrufen
und
# myMac=hexMac(MAC)          # in eine Hexziffernfolge
umgewandelt
# print("STATION MAC: \t"+myMac+"\n") # ausgeben
# # Verbindung mit AP im lokalen Netzwerk aufnehmen,
# # falls noch nicht verbunden
# # connect to LAN-AP
# if not nic.isconnected():
#   # Geben Sie hier Ihre eigenen Zugangsdaten an
#   # Zum AP im lokalen Netz verbinden und Status anzeigen
#   nic.connect(mySid, myPass)
#   # warten bis die Verbindung zum Accesspoint steht
#   print("connection status: ", nic.isconnected())
#   while not nic.isconnected():
#     blink(0.8,0.2,True)
#     print("{}.".format(nic.status()),end='')
#     sleep(1)
```

```
# # Wenn bereits verbunden, zeige Verbindungsstatus & Config-
Daten
# print("\nconnected: ",nic.isconnected())
# print("\nVerbindungsstatus: ",connectStatus[nic.status()])
# print("Weise neue IP zu:","10.0.1.101")
# nic.ifconfig(("10.0.1.101","255.255.255.0","10.0.1.20", \
#               "10.0.1.100"))
# STAconf = nic.ifconfig()
# print("STA-IP:\t\t",STAconf[0],"\nSTA-NETMASK:\t",\
#       STAconf[1],"\nSTA-GATEWAY:\t",STAconf[2] ,sep='')
#
# # *************  Setup Router connection end ***************


# ----------------- Server starten -------------------------
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.bind(('', 9000))
print("Socket established, waiting...")
s.settimeout(2.0) # timeout, damit 'while True' durchläuft
blink(2,0.5,True)
blink(0.3,0.8,True)
blink(0.3,0.8,True)
blink(0.3,1.5,True)
# ------------------- Serverschleife ------------------------
while True:
    try:
        r=""
        # recvfrom wird nach 2 Sec. mit einer Exception
abgebrochen
        # Zwischenzeitlich eintreffende Zeichen bleiben bis
zum
        # nächsten Durchlauf im Empfangsbuffer und werden dann
        # abgeholt.
        #
#         request, addr = s.recvfrom(256)
#         r=request.decode("utf8")
        r=input("Kommando:")
        addr="10.0.1.230"
        print('from {}\nContent = {}'.format(addr,r))
        if r!= "":
            job=parse(r)
            response=job[0]+"-"+job[1]+":"+str(job[2])
            print(response)
#             s.sendto(response,addr)
    except OSError as e:
        #print(e.args[0])
        pass
    if taste.value()==0:
        print("Mit Flashtaste abgebrochen")
        sys.exit()
    blink(0.1,0.9, inverted=True)
    #sleep(1)
```

Nach einigen Modul-Importen und der Definition einiger globaler Variablen werden ein paar Funktionen deklariert. Die Funktion blink() ermöglicht das einfache Aufrufen von Blinkzeichen über die LED.

Die Zeitsteuerung erledigen beim Wobbeln (aka automatischer Durchgang durch ein Frequenzband) sowie beim Burst (aka kurzer Puls mit bestimmter Frequenz) die zwei Timer-Serviceroutinen, increment() und shutoff().

Der umfassendste Teil des Programms ist die Funktion parse(). Wir prüfen hier auf die drei Bereichskenner W, C oder B für Wobbeln, Continuousmode und Burstmode. Dann gibt es Befehle, die einen Parameterwert senden und solche, die einen Prozess starten oder beenden sollen. Wird ein gültiger Befehl gefunden, sorgt der Parser auch für dessen Ausführung.

Es folgen zwei Abteilungen, von denen eine die Verbindung über das Hausnetz aufbaut. Für diesen Teil müssen wir die Zugangsdaten zum Accesspoint kennen und eintragen. Die andere Abteilung stellt einen eigenen Accesspoint über den ESP32 zur Verfügung. Als Netzwerkadresse können Sie eine aus den freien Bereichen wählen **10**.X.X.X oder **192.168**.X.X. Es kann aber stets nur eine der beiden Möglichkeiten aktiv sein. Die Auswahl geschieht durch Markieren des Blocks und Kommentieren (Alt + 3) oder Entkommentieren (Alt+4). Im Moment sind beide Blocks auskommentiert, weil wir ja über den input-Befehl mit unserem ESP32 sprechen.

Die Empfangsschleife des UDP-Servers wird mit einem Timeout auf 2 Sekunden festgelegt. Das bedeutet, dass nach dieser Zeit die Serverschleife weiter durchlaufen wird, wodurch auch beliebige andere Befehle ausgeführt werden können, während der Server aktiv bleibt. Das wird genutzt, um gegebenenfalls den Parser aufzurufen, eine Nachricht an den Client zurückzusenden und durch Druck der Flashtaste am ESP32 das Programm zu beenden. Letztere Möglichkeit ist wichtig, wenn das Programm autonom als boot.py läuft. Im Moment sind die Empfangsbefehle und das Rücksenden einer Nachricht auskommentiert. Dafür wurden die beiden folgenden Zeilen eingefügt.

```
r=input("Kommando:")
addr="10.0.1.230"
```

Zum Thema Fledermäuse gibt es noch einen interessanten Ansatz mit dem ESP32. Weil der Controller Töne von genau bekannter, stabiler Frequenz erzeugen kann, eignet sich der Aufbau auch als Generator für einen Fledermausdetektor. Diese Geräte nehmen über ein geeignetes Mikrophon die Rufe der Tiere auf. Das Signal wird verstärkt und mit einer zweiten Schwingung gemischt. Die Rufe werden für uns als eine Art Geckern vernehmbar, wenn die Mischfrequenz, als Differenz der beiden Frequenzen, in unserem Hörbereich liegt.

Der weckt Tote auf

Gemeint ist der ultimative HF-Endverstärker, denn Sie haben sicher auf die Ausbaustufe 3 gewartet. Hier ist der Bauplan ohne weiteren Kommentar. Die

Versorgungsspannung für die Endstufe kann bis zu 30 V betragen. Der Aufbau schließt die Ausbaustufe1 komplett mit ein. Sie können den ESP32 also direkt am Eingang anschließen. Hier steht die PDF-Datei mit Layout und Bestückungsplan zum Download bereit. Den Leistungstransistoren sollten Sie unbedingt einen Kühlkörper gönnen. Wichtig sind dann die Silikonmatten, die Sie im Bild erkennen, wenn Sie einen durchgehenden Kühlkörper verwenden, sonst gibt es Kurzschlüsse und das mag die Spannungsversorgung gar nicht. Das liegt daran, dass die Kollektoranschlüsse beider Transistortypen auch an der Gehäuserückseite liegen. Der Kollektor des BD135 (NPN) liegt auf 30V, der des BD136 (PNP) auf GND. Kennen Sie den Spruch " Plus auf Masse, das knallt Klasse"? Deshalb die Silikonmatten oder Glimmerscheiben als Isolierung.

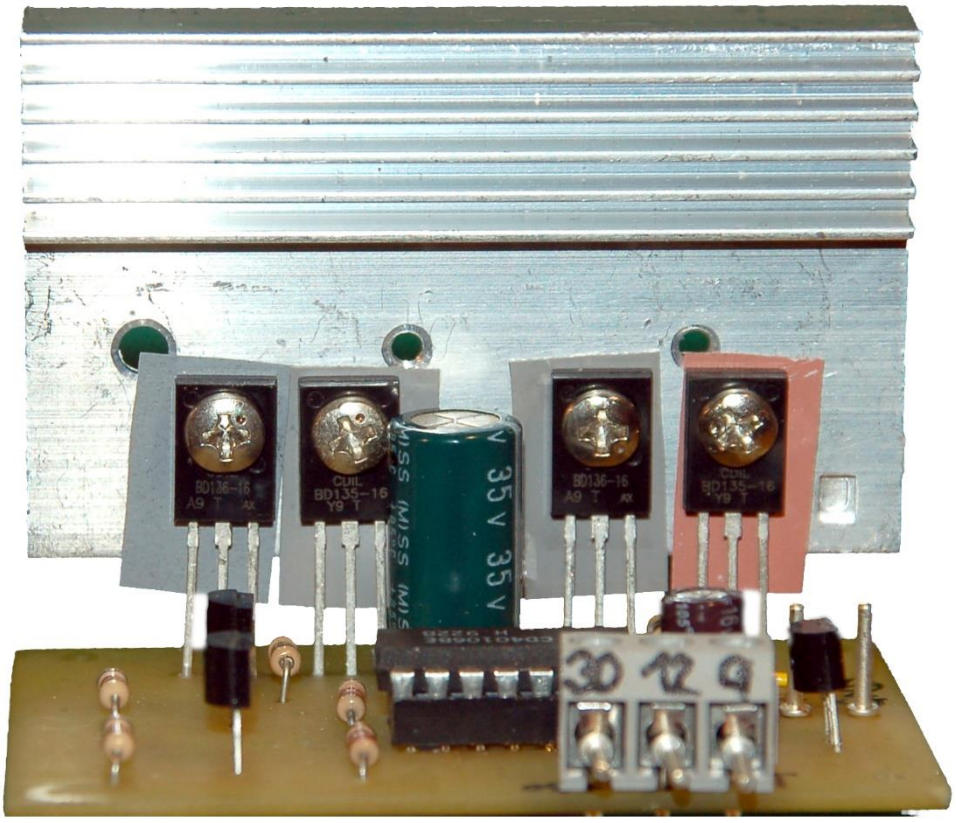

Abbildung 10: Verstärker

Ich wünsche schon mal viel Vergnügen beim Basteln und Programmieren mit den ungewöhnlichen Tönen und Tonlagen.