Freisprecheinrichtung für Yaesu FTM-400

Auto-Freisprecheinrichtung für das Yaesu FTM-400

(und sicher viele andere Funkgeräte)

letzte Änderung des Dokuments : 04.02.2019

Ab dem 1.Juli 2020 endet die „§52 Übergangs- und Anwendungsbestimmung der Straßenverkehrsordnung“, ab diesem Zeitpunkt ist es nicht mehr zulässig ein Mikrofon während der Fahrt in der Hand zu halten. Also musste eine Freisprecheinrichtung her - aufs Funken will ich ja nicht verzichten während der Fahrt. Alles auf dem Markt verfügbare war nicht so das was ich mir vorgestellt hatte….

Das ganze war damit ein neues großes Projekt für den Winter 2018/2019. Zuerst sammelte ich meine Anforderungen was die Schaltung können sollte :

- Komfort wie bei einer Telefon-Freisprecheinrichtung im Auto

- Keine Kabel zur einer Tasten-Fernsteuerung

- Keine Mikrofone die man erst aufsetzen/anstecken muss

- Hoch und Runterfahren der Schaltung gemeinsam mit dem Funkgerät

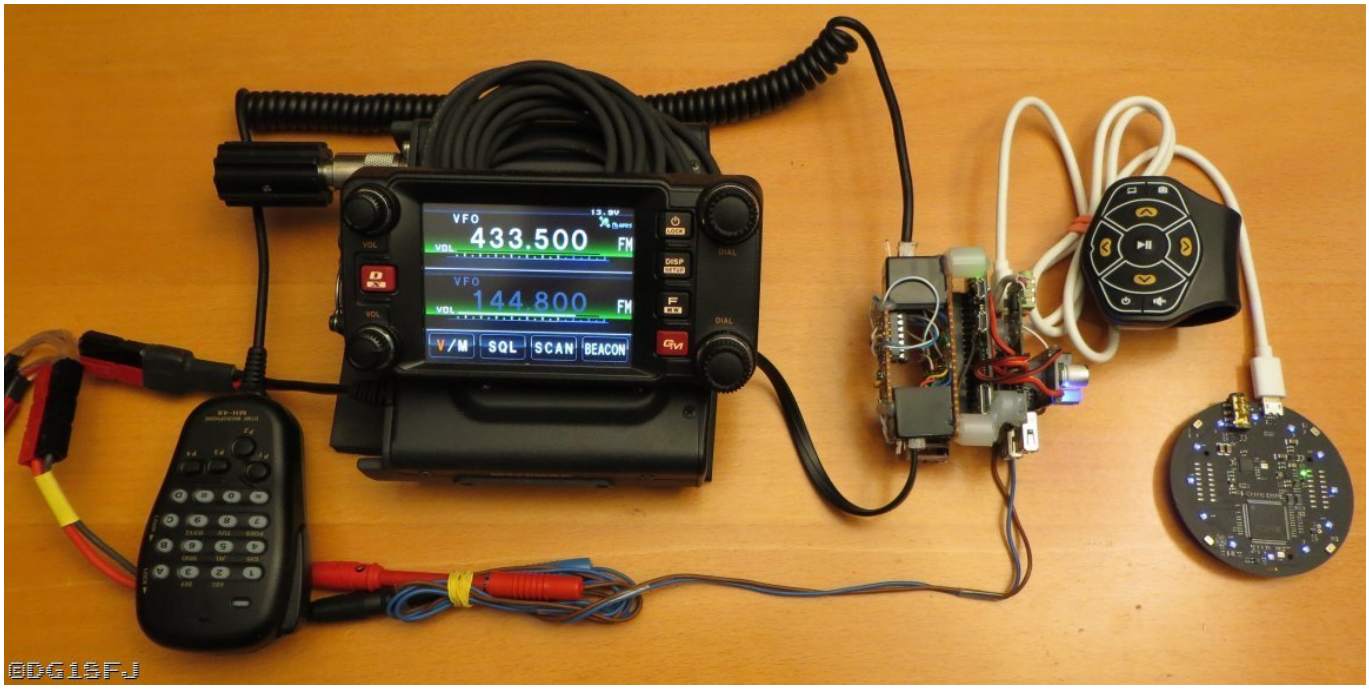
- Bequeme Steuerung vom Lenkrad aus (Kanal Up/Down, Tonruf, PTT, Analog/Digital, …)

- Nutzung von Handmikrofon und Freisprecheinrichtung parallel möglich

- automatisches einfrieren der AGC-Parameter bei loslassen der PTT und wiederherstellen bei PTT drücken

Wie immer wurde das Projekt doch aufwändiger wie man denkt, ist aber nun endlich im Beta-Stadium angekommen. Das ganze ist keine Schritt-für-Schritt Nachbauanleitung sondern soll nur Ideen für eigene Lösungen liefern. Eine fertige Lösung muss natürlich alle geltenden Richtlinien für z.B. Elektronik im Kfz einhalten aber auch viele andere . Hier gibt es also nur Bilder vom Labortisch. Ich übernehme keine Haftung, keine Garantie, keine Beratung u.s.w. Ich betone es gleich vorweg : Kein Vertrieb, kein Verkauf, keine Anfragen mit „kannst du mir das zusammenbauen“, „kannst du mir das auch bauen für mein Funkgerät xyz“u.s.w. , …Es handelt sich um ein rein privates Projekt ohne jegliche wirtschaftliche Interessen.

So sieht der gesamte Versuchsaufbau mit dem FTM400 aus, die Schaltung wird zwischen Funkgerät und normalem Handmikrofon eingesteckt. Man sieht noch die Bluetooth Fernsteuerung fürs Lenkrad sowie die kreisförmige Leiterplatte des Mikrofon-Arrays mit dem weissen USB-Kabel. Dieses Projekt ist das erste was ich jemals mit einem Raspberry Pi Zero umgesetzt habe, dazu noch das erste große mit Python. Dank der vielen Bibliotheken für die einzelnen Module läßt sich also auch von einem Programmier-Einsteiger zügig eine Idee umsetzen.



Die Elektronik besteht aus 5 Leiterplatten-Ebenen :

- Die Audio-Umschaltung (nur ein IC auf Lochraster) zwischen Mikro und Freisprech

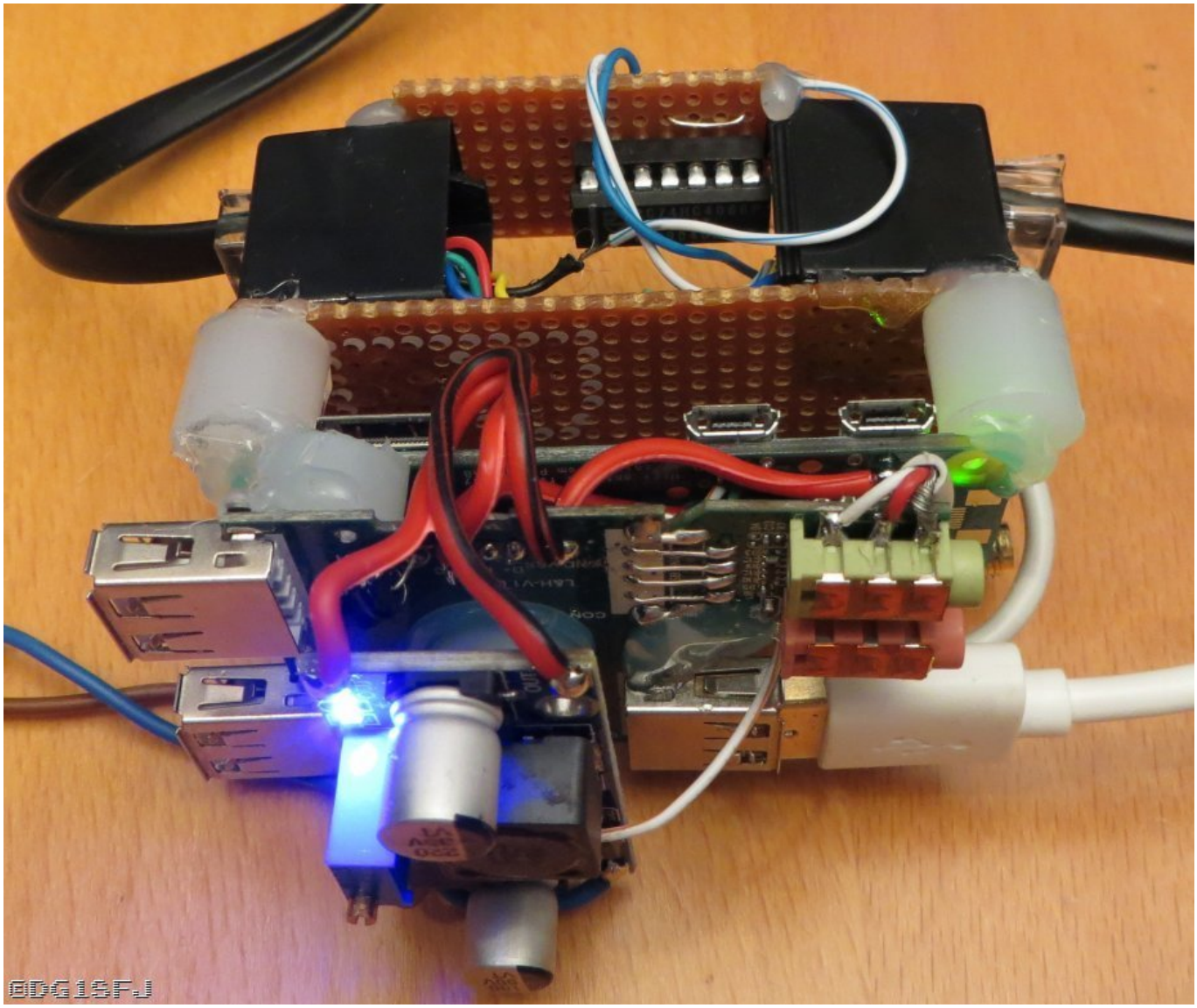
- Die Signalaufbereitung und Generierung für die Mikrofon-Emulation auf Lochraster

- Der Raspberry-Pi-Zero (mit WLAN und Bluetooth)

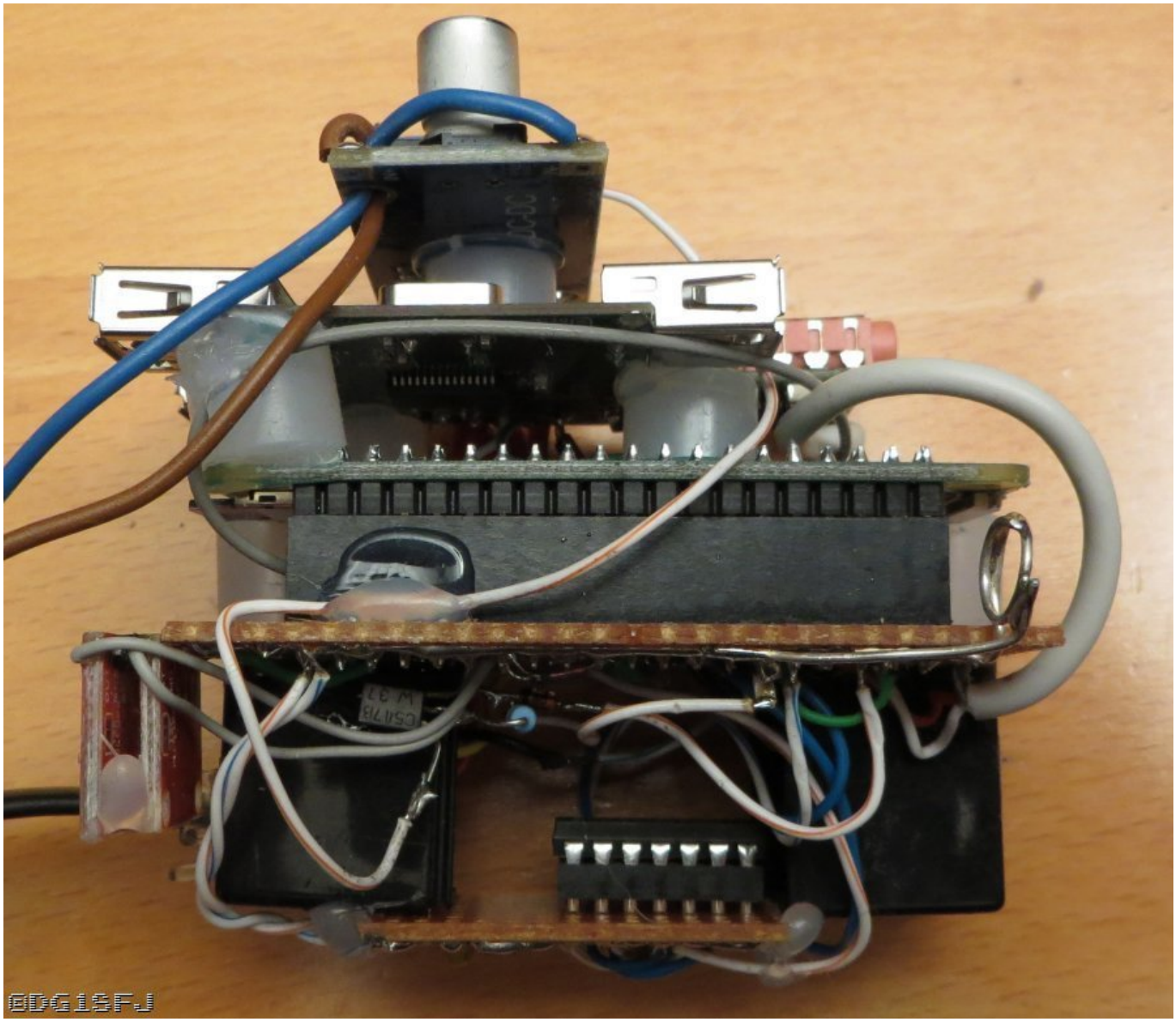
- Die Leiterplatte eines USB-Hubs mit angelöteter USB Soundkarte

- Die Stromversorgung per DCDC Wandler, gesteuert vom Raspi

Seitenansicht 1 :



Seitenansicht 2 :



Inhaltsverzeichnis :

```
Pinout USB und Supply Testpins
Skripte
Python-Programm
Blockschaltbild mit den einzelnen Komponenten
```

Funktion :

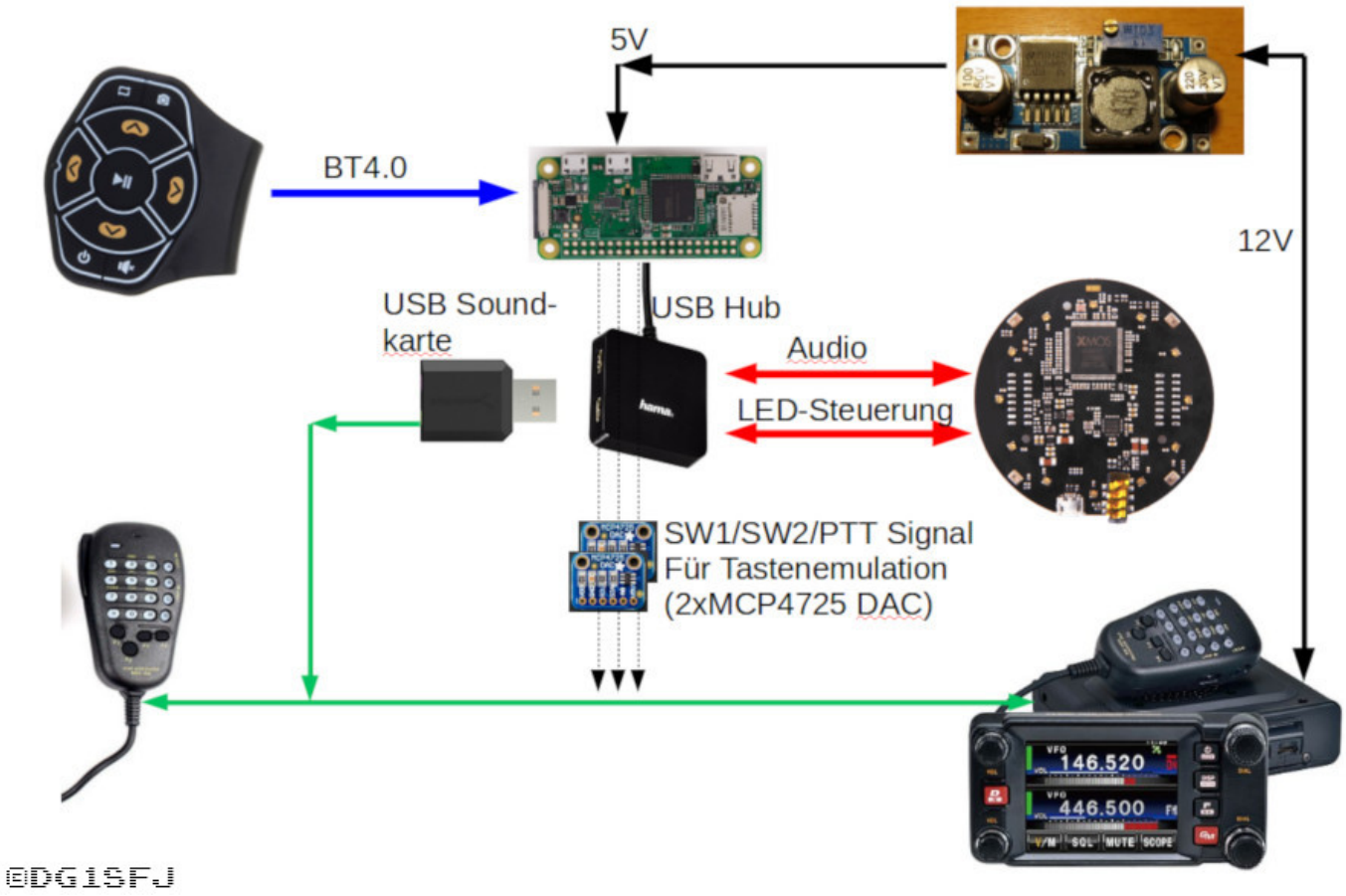
Die Schaltung prüft ob auf der Leitung zwischen Mikrofon und Funkgerät 5V anliegen. Wenn ja, wird der Schaltregler aktiviert und erzeugt aus den 12V die 5V für den Raspi. Dieser fährt dann sein Betriebsystem hoch. In dieser Zeit werden über den USB Hub zwei Komponenten aktiviert und eingestellt : Einerseites der ReSpeaker der das gesprochene Wort auffängt, die AGC enthält und dann per USB die Audio Digital Daten an den Raspi gibt. Der Raspi gibt dann die Audiodaten per USB weiter an die USB Soundkarte, die dann das Audio wieder in die Mikrofon-Leitung einspeist. Damit hier nicht 2 Signale zeitgleich vorkommen gibt es noch einen Audio-Umschalter der zwischen Handmikrofon und Soundkarte wählt.

Drückt der User die Bluetooth Fernbedienung wird automatisch das Pairing herstellt und die Kommandos der Fernbedienung kommen als Multimedia Tastaturkommandos an der Software an. Diese steuert dann abhängig vom Betriebsmodus die 12 RGB LEDs die auf der Respeaker Platine sind als eine Art User Interface. Die Tastenkommandos (Rauf/Runter/PTT/Tonruf/…) werden als analoge Spannungen durch 2 DACs erzeugt und emulieren ein Drücken der Tasten auf dem Mikrofon.

Beim Drücken der PTT Taste wird der Respeaker mit seinen DSP Parametern (AGC!) neu programmiert mit dem Wert den er beim letzten loslassen der PTT hatte. Notwendig ist dies da die AGC der Respeaker Platine sich nicht stoppen läßt, also permanent mitläuft - was bedeutet das wenn nicht gesprochen wird, der DSP die AGC raufzieht.

Nebenbei beobachtet der Raspi immer die Versorgungsleitung des Handmikrofons. Ist dieser wieder 0V, da der User das Funkgerät ausgeschaltet hat, wird das Betriebssystem kontrolliert heruntergefahren und nimmt sich dann selbst die Versorgungsspannung durch abschalten des Schaltreglers.

Blockschaltbild :



Mikrofon

Zum FTM400 gibt es das MH-48A6J Fernbedienungsmikrofon. Ganz viele Knöpfe, alles steuerbar. Wenn man sich mal daran gewöhnt hat, will man den Komfort einiger Knöpfe nicht mehr wissen. Damit war klar das ich die Funktionalität dieses Mikrofons mit einer Freisprecheinrichtung nachbauen muss. Startfrage also : Wie überträgt das Mikro die ganze Knopf-Kommandos ?

Im Mikrofon sind die Tasten in einer Art Matrix angeordnet. Je nach Zeile und Spalte stellt sich durch Widerstände nach Masse eine andere Spannung ein. Multimeter raus und los gehts. Aufgrund der hochohmigkeit der Spannung ist die spätere Simulation des Tastendruckes einfach - wir nehmen 2 D/A Wandler und hauen das auf die Leitung. Somit ist die Bedienung vom Original-Mikrofon sowie vom Raspi aus möglich da das Mic nur Widerstände nach Masse legt und der Raspi per DA nur eine Spannung anlegt.

Taste	SW2 [V]	SW1 [V]
nix	3,198	3,198
1	0,703	0,083
2	1,303	0,083
3	1,948	0,082
4	0,703	0,669
5	1,303	0,669
6	1,948	0,669
7	0,703	1,292
8	1,303	1,293
9	1,948	1,293
0	1,304	1,949
*	0,703	1,949
#	1,948	1,951
A	2,623	0,082
B	2,623	0,668
C	2,623	1,292
D	2,624	1,951
P1	0,704	2,559
P2	1,306	2,559
P3	1,952	2,559
P4	2,624	2,563
DWN	0,21	1,408
[illegible]	0,206	0,827

```
Die Kabelbelegung beim Original-Mikrofon :

SW2 - Grün
SW1 - Grau
SW5V - Rot
GND - Schwarz
MIC - Weiss
PTT - Blau
```

Die SW5V liegt auf 4.94V solange das Funkgerät eingeschaltet ist. An dieser Leitung läßt sich also gut erkennen welchen Power-Zustand die Funke hat.

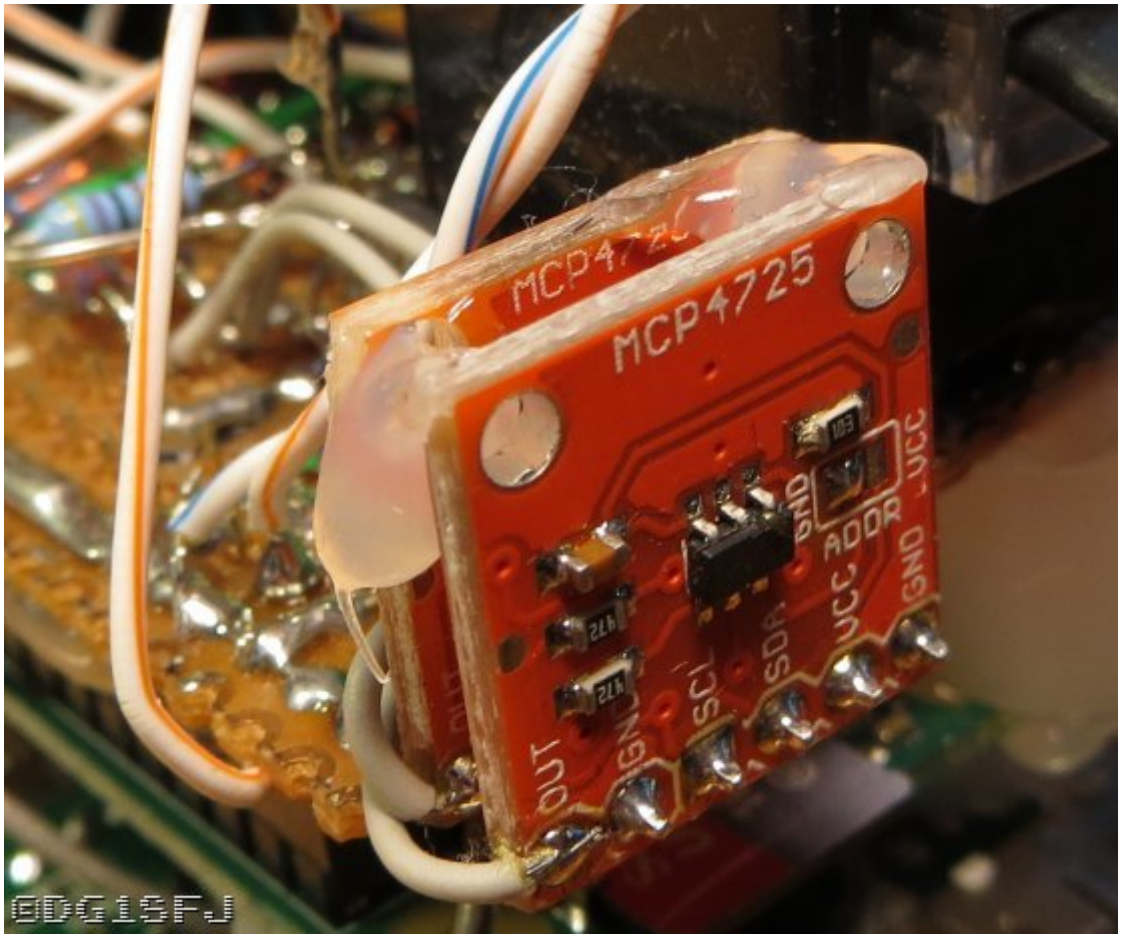
Die PTT wird per Widerstand nach Masse gezogen, es liegen 3.12V im Grundzustand an, bei PTT on 0.58V.

Die Leitungen SW1 und SW2 haben jeweils 3.2V im Grundzustand.

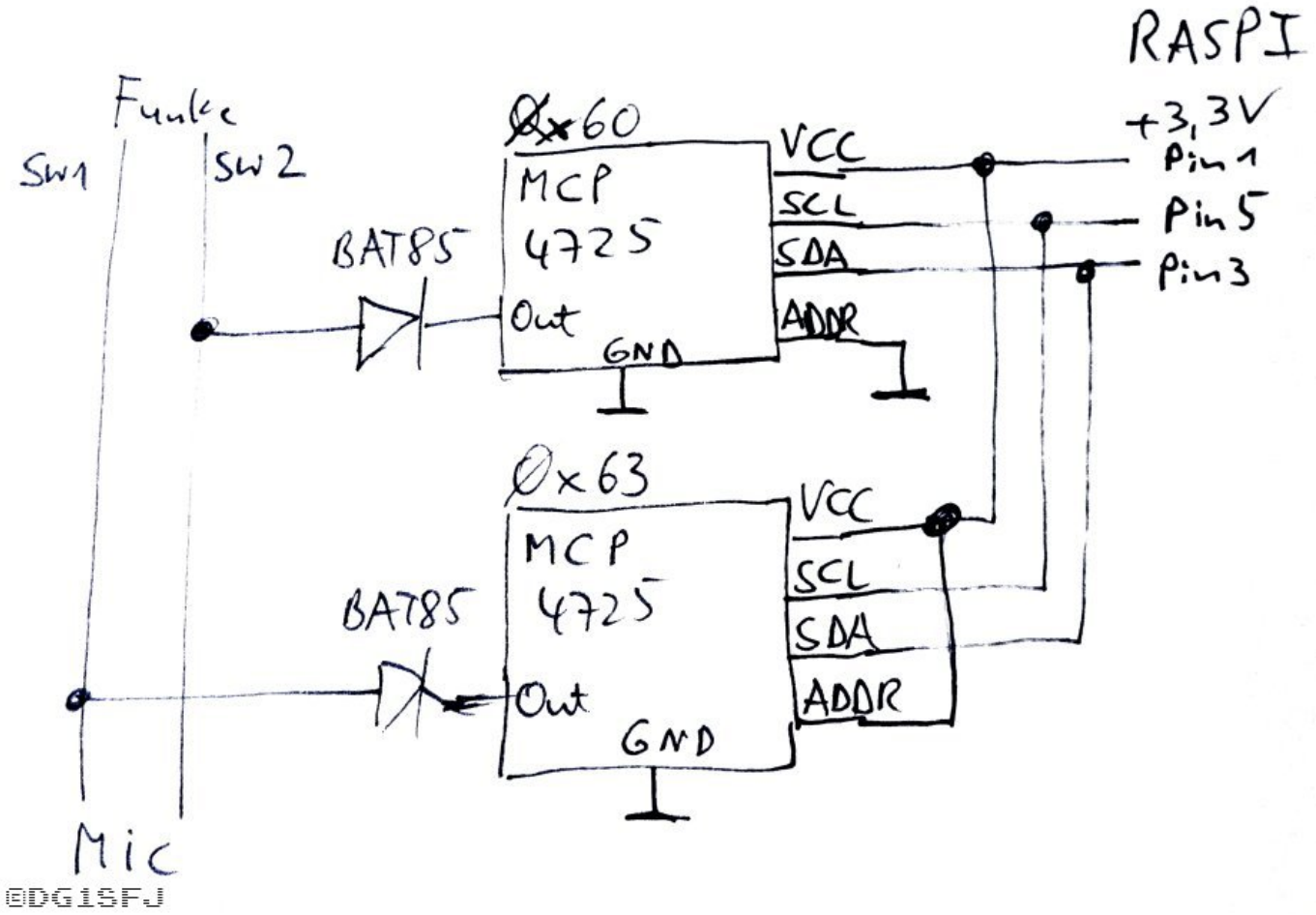
Tasten-Simulation

Wie beschrieben müssen wir nur mit 2 DAC Platinen mit dem MCP4725 die Spannungen für die SW1 und SW2 Leitungen emulieren. Die Platinen gibt es fertig bestückt für wenig Geld und man kann per Lötbrücke eine der beiden Adressen am I2C Bus auswählen.

Für SW1 habe ich die Adresse 0x63 und für SW2 die Adresse 0x60 gewählt. Die beiden Platinen lassen sich also Parallelschalten, am Raspi ist Pin 3 SDA und Pin 5 SCL. Versorgung mit 3.3V



Die Leitungen von SW1/2 liegen vom Funkgerät aus auf 3.2V. Wenn wir also per DAC 3.3V ausgeben sind die DACs dank der Diode abgehängt und unsichtbar. Erst wenn die Spannung kleiner wird, werden die SW1/2 Leitungen manipuliert. Somit erspart man sich aufwendige Umschaltungen.



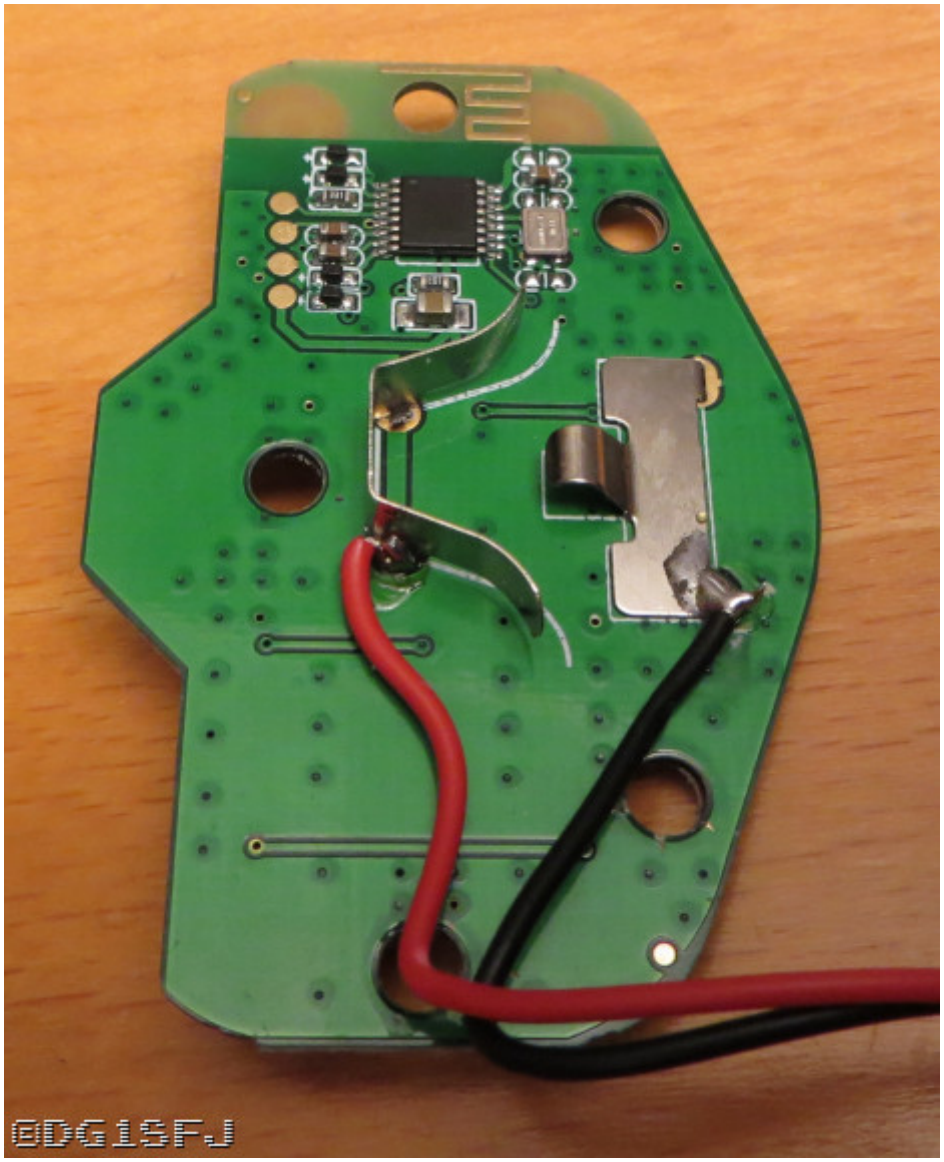
Mit dem Oszi kann man auch schön sehen das die Zeitdauer für die Spannungsänderung je nach gedrückter Taste am Originalmikrofon so zwischen 200 und 600us dauert. Der DAC ist schnell genug das sich so ein Tastendruck emulieren läßt.

Fernbedienung

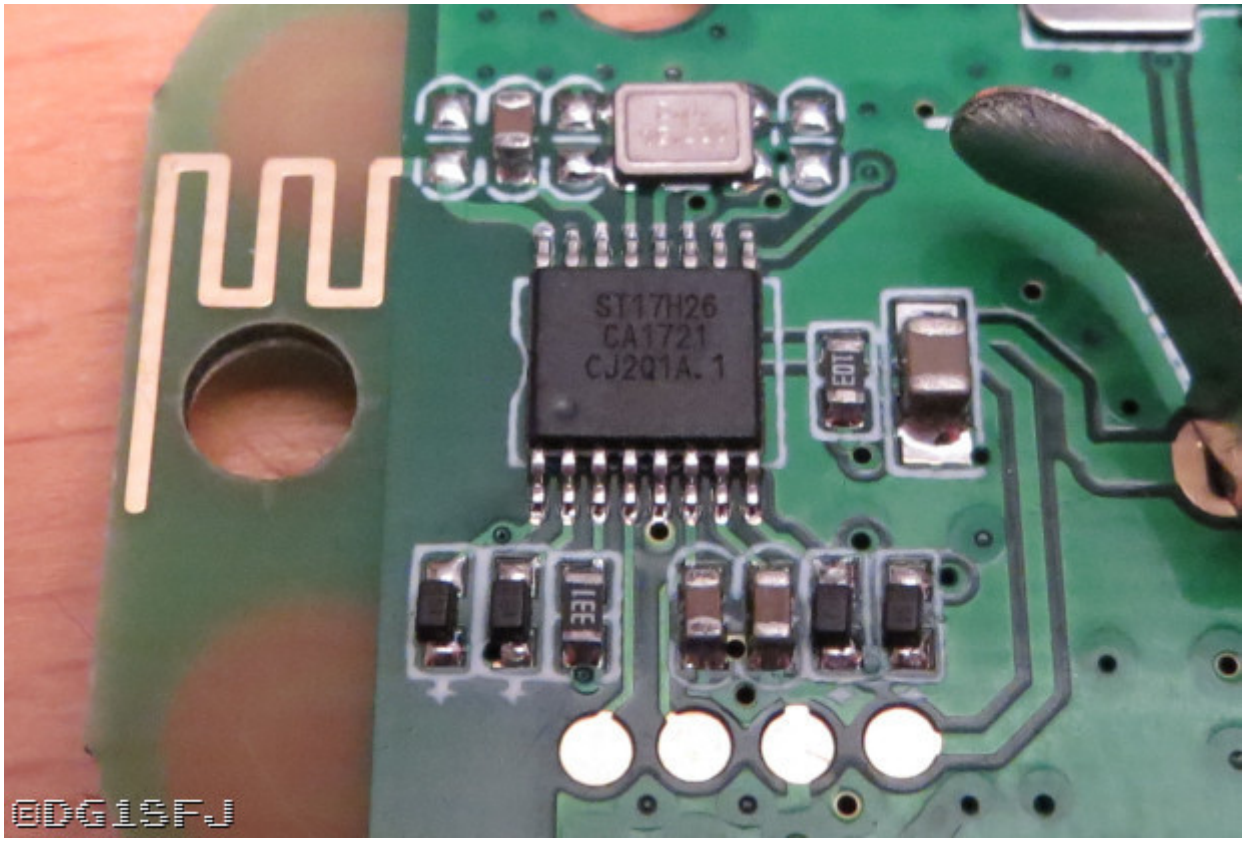
Zur Steuerung des FTM400 hat man ja das MH-48A6J Fernbedienungsmikrofon. Das muss dann ersetzt werden. Als Idee entstand die Nutzung einer Multimedia Lenkradfernbedienung für Autos. Die gibt es in verschiedenen Ausführungen. Ich habe mich für die X09 entschieden, da diese genug Tasten bietet und Bluetooth 4.0 Low Energy hat. Dies hat den Vorteil das nur beim Drücken ein Sendesignal losgeht sowie beim loslassen der jeweilige Taste. Bei älteren Bluetooth 2/3 Fernbedienungen konnte ich beobachten das beim Drücken einer Taste die Fernbedienung auf Dauer-Sendung geht und damit waren die Batterien schnell leer.

So sieht die X09 - im Prinzip wie eine Uhr wird es um das Lenkrad geschnallt. X09 wegen der 9 Tasten. Eine ist allerdings nur für Ein/Aus und die Photo sowie Up-Taste erzeugen dasselbe Kommando. Scheint wohl ein Bug zu sein. Ich habe 2 davon gekauft und beide haben dasselbe Verhalten. Sind es halt nur noch 7 Funktionen. Das reicht auch.

So sieht die X09 von Innen aus (auf der anderen Seite sind nur diese Knackfrosch-Taster) :



So sieht der Chip der X09 aus :



Diese Bluetooth Fernbedienung emuliert einfach eine Tastatur mit ihren Multimedia-Tasten :

```
Event code 114 (KEY_VOLUMEDOWN)
Event code 115 (KEY_VOLUMEUP)
Event code 163 (KEY_NEXTSONG)
Event code 164 (KEY_PLAYPAUSE)
Event code 165 (KEY_PREVIOUSSONG)
Event code 113 (KEY_MUTE)
Event code 305 (BTN_EAST)

Snap (falsch belegt!) : Event code 115 (KEY_VOLUMEUP)
```

Die Fernbedienung wird wie üblich auf dem Raspi gesucht, gepaired und getrustet. Danach ist sie bei jedem Neustart verfügbar.

Belegt habe ich die Tasten wie folgt :

```
KEY_VOLUMEDOWN   : Kanal runter
KEY_VOLUMEUP     : Kanal rauf
KEY_NEXTSONG     : PTT
KEY_PLAYPAUSE    : 1750Hz Rufton
KEY_PREVIOUSSONG : FM / C4FM
KEY_MUTE         : Knopf V/M
BTN_EAST         : Knopf Display
```

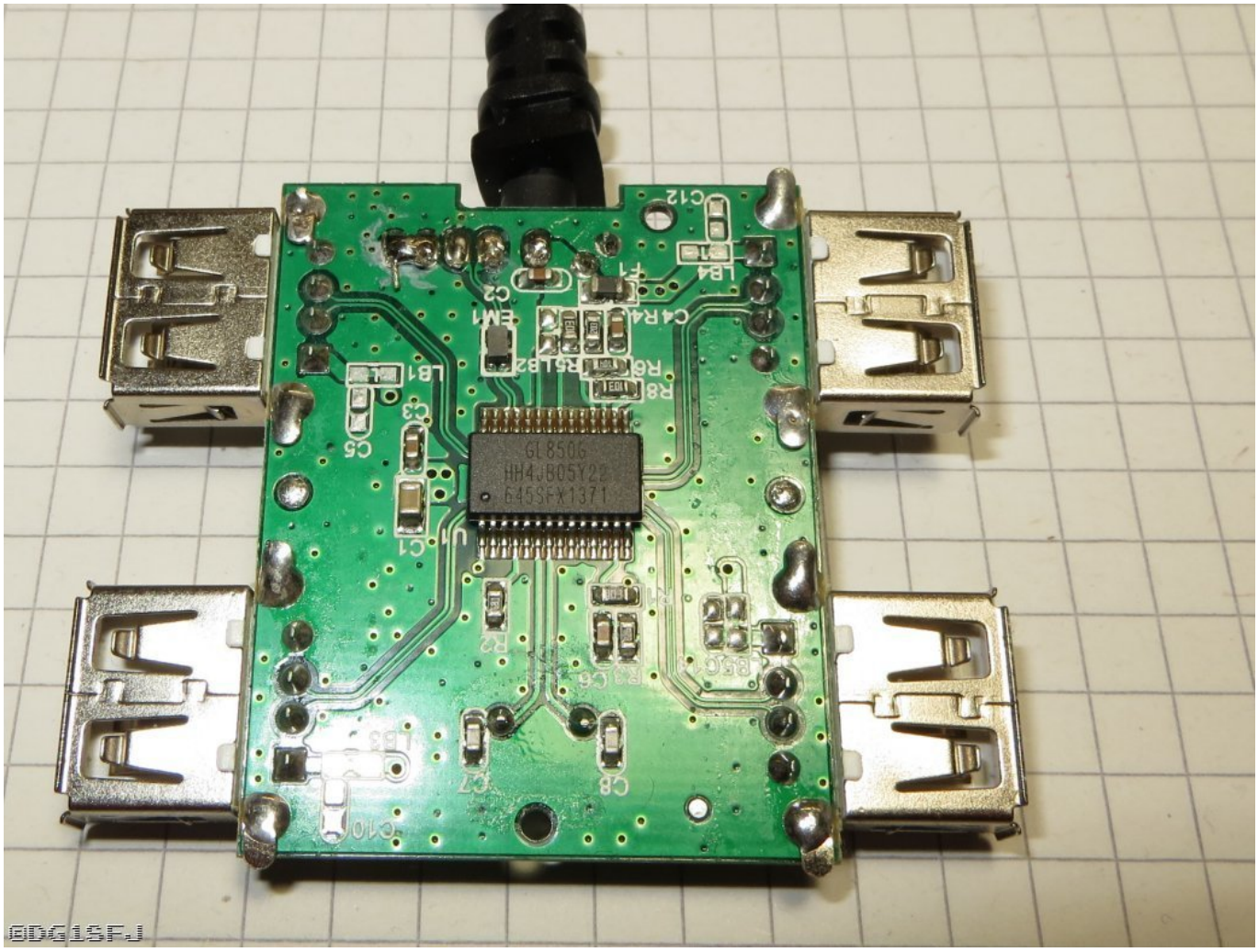
Das ganze läßt sich im Python Code natürlich frei ändern.

USB-Hub

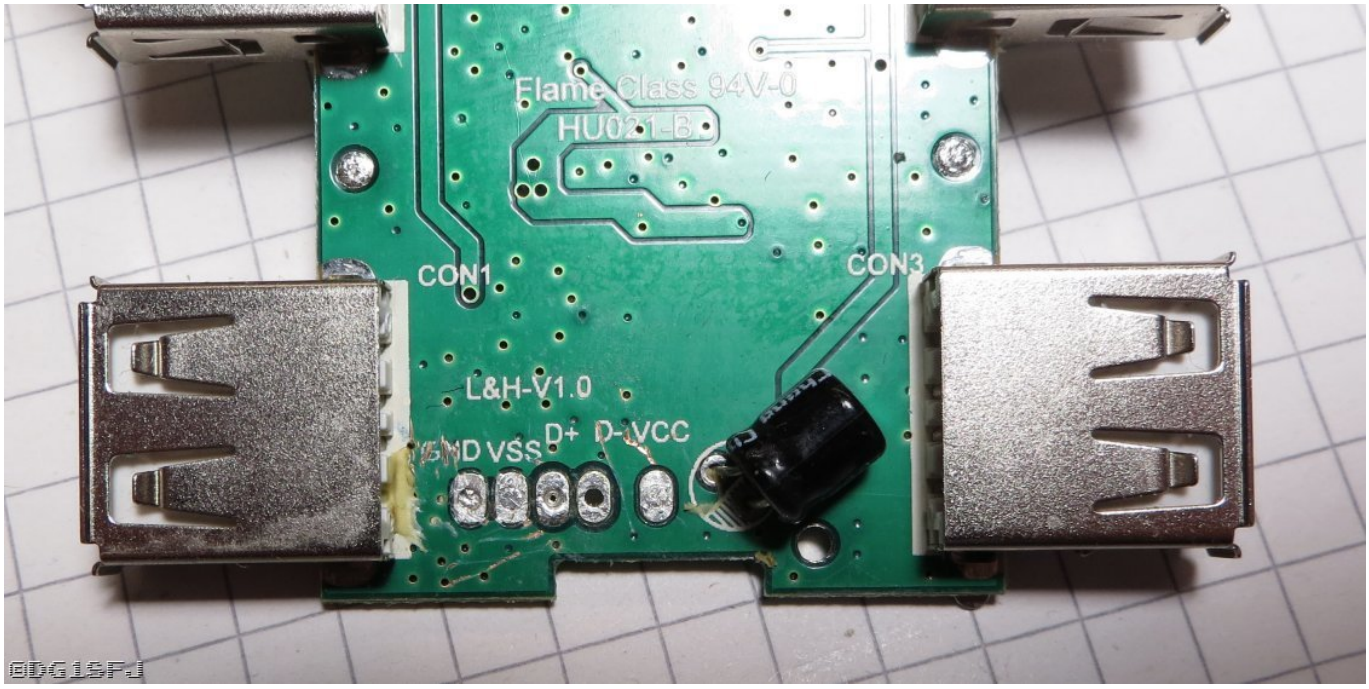
Der Raspberry Pi Zero W hat leider nur einen USB-Port. Das reicht nicht für die Schaltung. Kurzerhand den nächst besten 3er USB Hub zerlegt und direkt mit dem Raspberry Pi verbunden (dazu gibt es auf der Rückseite der Leiterplatte Testpins die man gut benutzen kann). Damit alles kompakt wird habe ich noch eine der 4 Buchsen entlötet und die USB Soundkarte direkt mit Silberdrähten angelötet.

Der USB ist ein alter USB2 Hub von Hama :

Gehäuse brauchen wir nicht, ebenso das Anschlußkabel :



Nett von Hama die Anschlußleitungen noch zu beschriften 🙂. Von Hier gehts zum DCDC Wandler mit den 5V und GND sowie mit einer verdrillten Zweidrahtleitung zum Raspi.

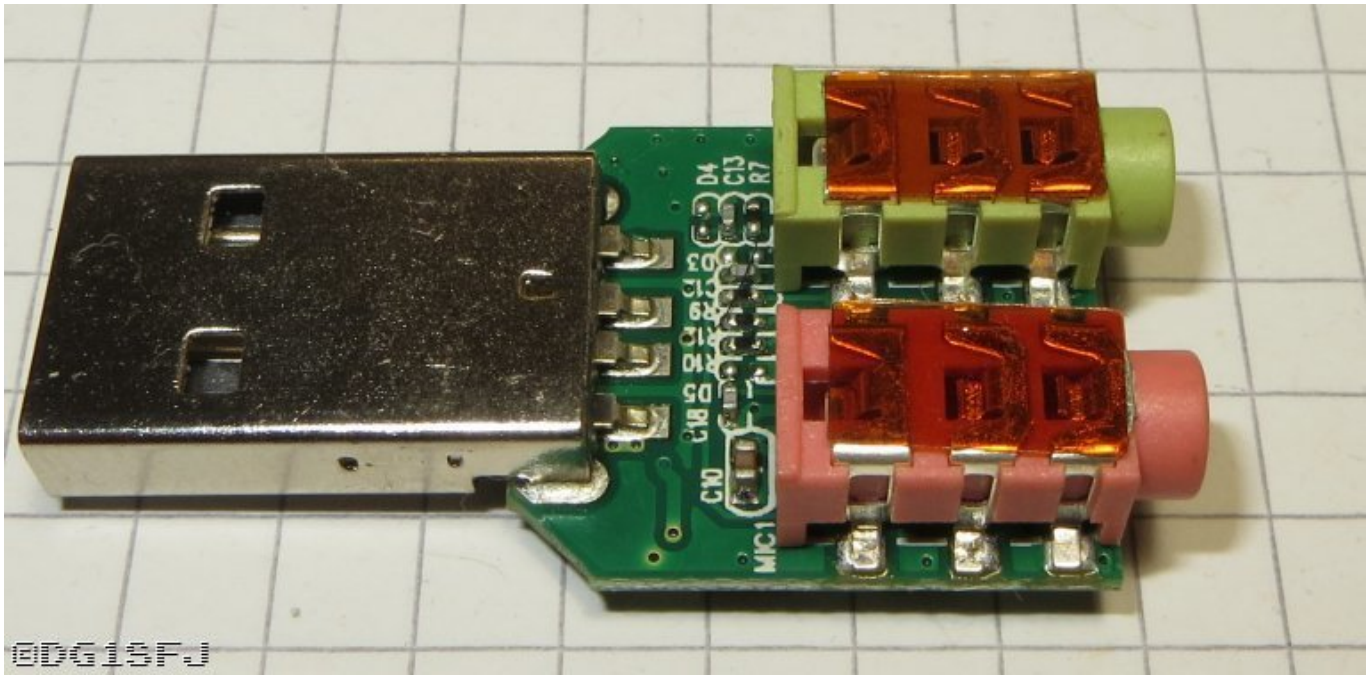


An eine Buchse wird also die Soundcard gelötet. An eine zweite Buchse kommt der USB2 Kabel der Respeaker dran. Sind also noch 2 Buchsen übrig für weitere Spielereien.

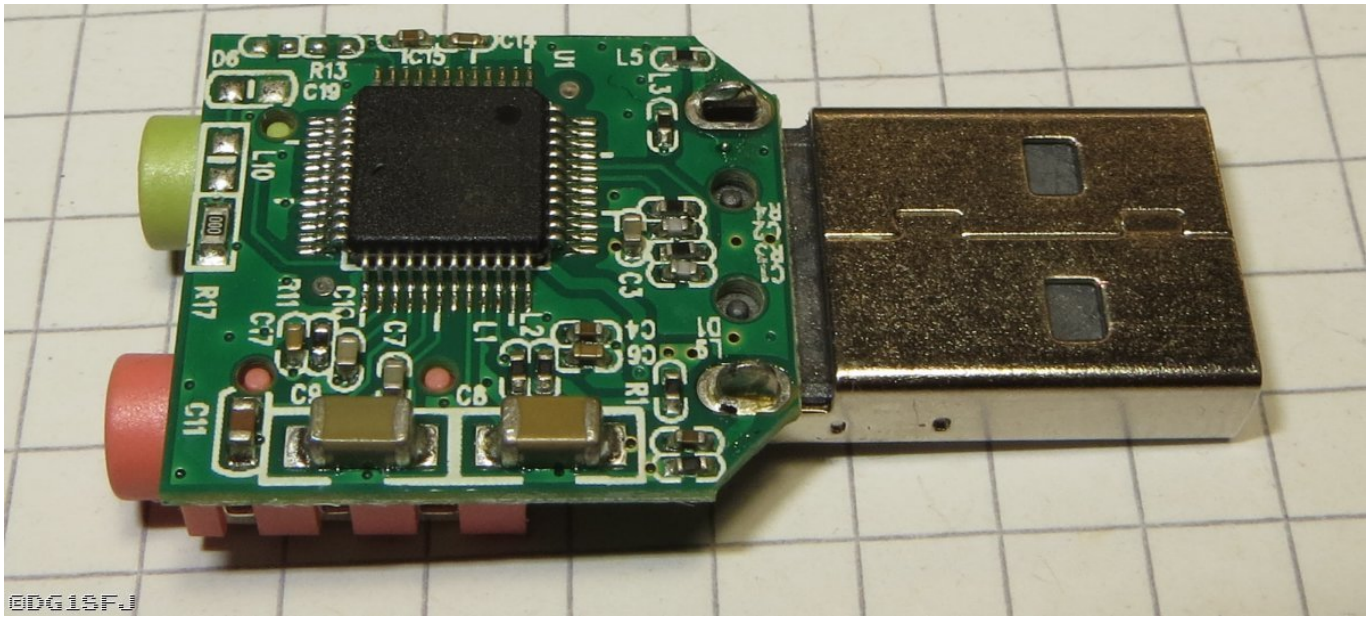
USB-Soundcard

Wenn das Audio per USB vom Respeaker kommt muss es wieder in „analoges“ Audio gewandelt und in die Mikrofonleitung eingespeist werden. Dies übernimmt eine kleine USB Soundkarte. Die Auswahl war eher zufällig - was es eben gab und was am Raspi funktioniert ohne Frickelei. Es war dann eine Sabrent USB External Soundcard

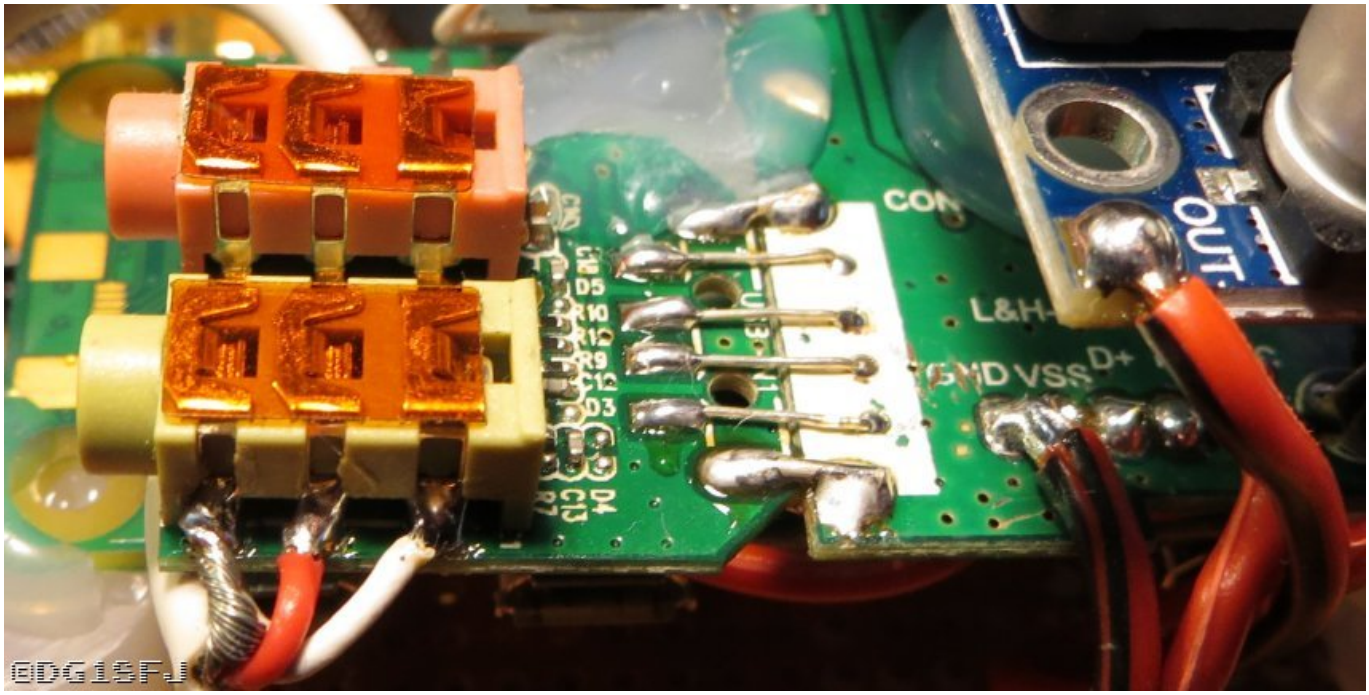
Gehäuse weg, stört nur ...



Rückseite ...



Der USB Stecker wird abgelötet und die Soundkarte direkt an den USB Hub gelötet. Praktischerweise kann man gleich ein Stereokabel an die Headphone Buchse anlöten und zur weiteren Verarbeitung nutzen. Siehe Schaltplan Audioumschaltung.



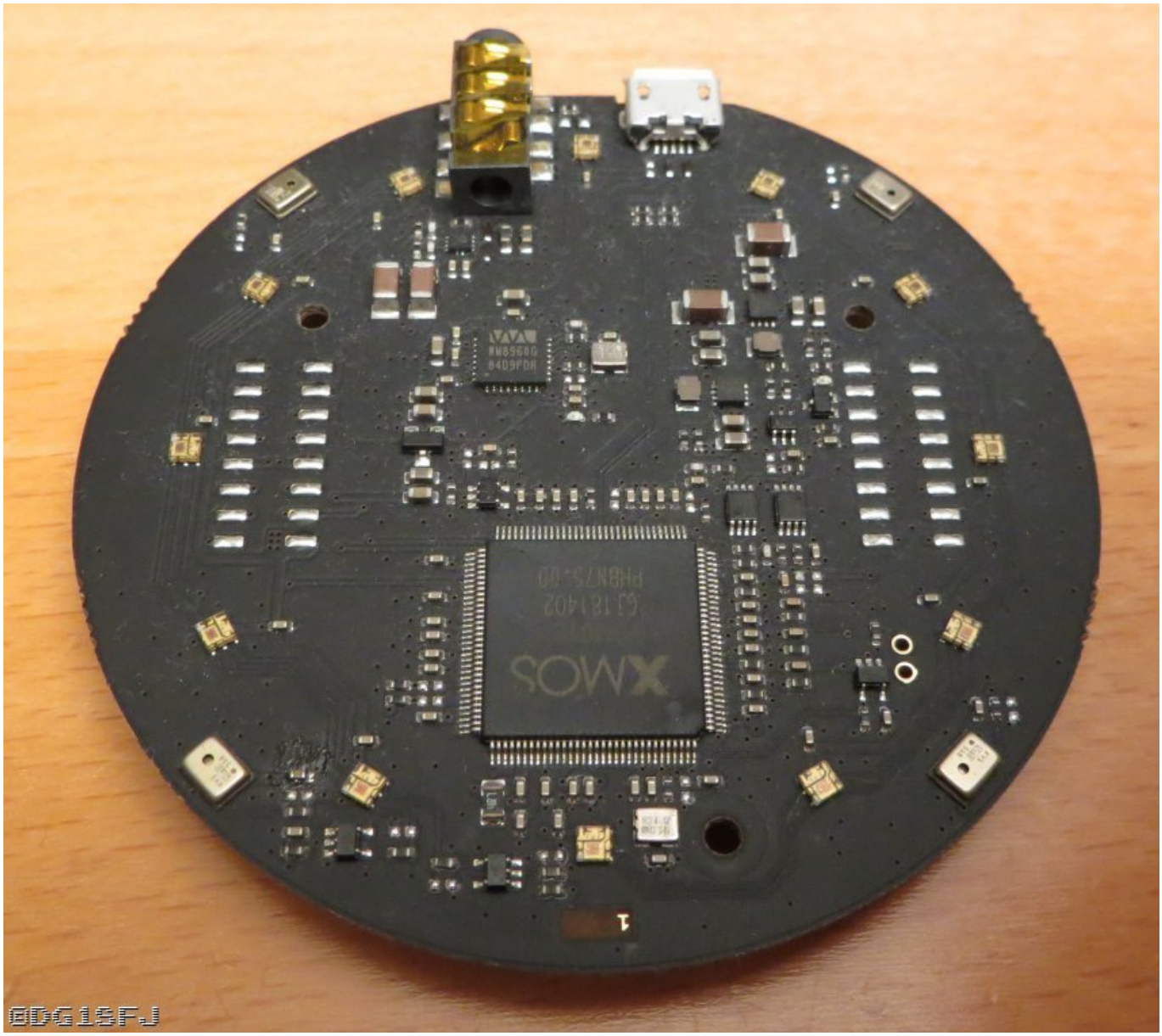
USB-Respeaker

Eines der wichtigsten Komponenten ist der ReSpeaker Mic Array V2.0 von Seeedstudio. Dieses geniale Teil erledigt alles war ich gesucht habe. 4 Mikrofone die gezielt in eine Richtung hören können, Umgebungsgeräusche ausfilter, eine AGC besitzen, per USB mit Parametern frei einstellbar sind u.s.w. Als Add-On gibts noch 12 RGB LEDs die sich ebenso per USB steuern lassen und mit der ich zum Spass eine Art User-Interface mit unterschiedlichen Farben progammiert habe. Von Seeedstudio sind auch alle dafür benötigten Python Komponenten.

Der DSP des ReSpeakers läuft die ganze Zeit und passt damit die AGC permanent an. In den Sprechpausen zieht er also die Verstärkung immer weiter hoch. Das reicht im Haus um Gespräche aus über 10m mitverfolgen zu können. Das ist natürlich im Fahrzeugeinsatz unnötig. Deswegen die Idee der Kopplung mit der PTT : Wenn man die PTT drückt schreibt der Raspi die AGC Parameter vom letzten Sendedurchgang wieder zurück. Während PTT gedrückt ist, ist die AGC aktiv und regelt mit Zeitkonstanten von ca. 1 Sekunde die Lautstärke und passt sich an den Sprecher an. Beim loslassen der PTT wird per USB der letzte AGC Wert ausgelesen und abgespeichert für den nächsten Sendedurchgang. Dieses Feature war ausschlaggebend das ich den ReSpeaker Mic Array v2.0 gekauft hatte. Und es funktioniert wirklich gut !

Ausfiltern von Klappern, Fahrgeräuschen, dem Sprecher im Radio, Musik aus dem Radio geht mal mehr mal weniger gut. Da muss man zum Teil schon Mitleid mit dem DSP haben - der hats nicht einfach.

Auf der Platine ist zusätzlich noch ein kleiner 1W Audioverstärker, das reicht aber für den Kfz-Einsatz eher nicht aus. Ermöglicht aber sicher noch weitere Spielereien in Zukunft. Die Mikrofone geben übrigens kein analoges Signal an den DSP sondern ein Digitales aus. Wer mehr wissen will, Datenblatt zu den MP34DT01 von ST Microelectronics anschauen.



An den Parametern habe ich länger gespielt um den Einfluss zu beobachten. Letzendlich bin ich mit diesen Einstellungen gut gefahren :

```
dev2.write('AGCDESIREDLEVEL',0.02)
dev2.write('AGCGAIN',9.0)
dev2.write('AGCMAXGAIN',31.6)
dev2.write('AGCONOFF',1)
dev2.write('AGCTIME',0.1)

dev2.write('CNIONOFF',0)
dev2.write('ECHOONOFF',1)
dev2.write('HPFONOFF',3)

dev2.write('STATNOISEONOFF',1)
dev2.write('STATNOISEONOFF_SR',1)
dev2.write('TRANSIENTONOFF',1)
```

Es gibt noch viel mehr, aber die sind eher für andere Anwendungen gedacht.

Die LEDs sind alle als „Pixel-Ring“ geschaltet. Da RGB sind alle Farb-Kombinationen möglich. Folgende habe ich umgesetzt :

- Grundzustand : Alle LEDs leuchten ganz leicht blau

- Kommando ans Funkgerät : eine LED grün, zwei LEDs leicht grün

- PTT fürs Funkgerät : eine LED rot, zwei LEDs leicht rot

- 1750Hz Tonruf : 3 LEDs gelb, 6 LEDs leicht gelb




Hier noch die Liste aller Parameter, die man per USB verstellen kann :

```
AECFREEZEONOFF      int 1   0   rw  Adaptive Echo Canceler updates inhibit.
                                                        0 = Adaptation
enabled
                                                        1 = Freeze
adaptation, filter only
AECNORM             float   16  0.25    rw  Limit on norm of AEC filter
coefficients
AECPATHCHANGE       int 1   0   ro  AEC Path Change Detection.
                                                        0 = false (no
path change detected)
                                                        1 = true (path
change detected)
AECSILENCELEVEL     float   1   1e-09   rw  Threshold for signal detection
in AEC [-inf .. 0] dBov (Default: -80dBov = 10log10(1x10-8))
AECSILENCEMODE      int 1   0   ro  AEC far-end silence detection status.
                                                        0 = false
(signal detected)
                                                        1 = true
(silence detected)
AGCDESIREDLEVEL     float   0.99    1e-08   rw  Target power level of the
output signal.
                                                        [-inf .. 0] dBov
(default: -23dBov = 10log10(0.005))
AGCGAIN             float   1000    1   rw  Current AGC gain factor.
                                                        [0 .. 60] dB
(default: 0.0dB = 20log10(1.0))
AGCMAXGAIN          float   1000    1   rw  Maximum AGC gain factor.
                                                        [0 .. 60] dB
```

```
(default 30dB = 20log10(31.6))
AGCONOFF            int 1   0   rw  Automatic Gain Control.
                                                        0 = OFF
                                                        1 = ON
AGCTIME             float   1   0.1 rw  Ramps-up / down time-constant in
seconds.
CNIONOFF            int 1   0   rw  Comfort Noise Insertion.
                                                        0 = OFF
                                                        1 = ON
DOAANGLE            int 359 0   ro  DOA angle. Current value. Orientation
depends on build configuration.
ECHOONOFF           int 1   0   rw  Echo suppression.
                                                        0 = OFF
                                                        1 = ON
FREEZEONOFF         int 1   0   rw  Adaptive beamformer updates.
                                                        0 = Adaptation
enabled
                                                        1 = Freeze
adaptation, filter only
FSBPATHCHANGE       int 1   0   ro  FSB Path Change Detection.
                                                        0 = false (no
path change detected)
                                                        1 = true (path
change detected)
FSBUPDATED          int 1   0   ro  FSB Update Decision.
                                                        0 = false (FSB
was not updated)
                                                        1 = true (FSB
was updated)
GAMMAVAD_SR         float   1000    0   rw  Set the threshold for voice
activity detection.
                                                        [-inf .. 60] dB
(default: 3.5dB 20log10(1.5))
GAMMA_E             float   3   0   rw  Over-subtraction factor of echo
(direct and early components). min .. max attenuation
GAMMA_ENL           float   5   0   rw  Over-subtraction factor of non-
linear echo. min .. max attenuation
GAMMA_ETAIL         float   3   0   rw  Over-subtraction factor of echo
(tail components). min .. max attenuation
GAMMA_NN            float   3   0   rw  Over-subtraction factor of non-
stationary noise. min .. max attenuation
GAMMA_NN_SR         float   3   0   rw  Over-subtraction factor of non-
stationary noise for ASR.
                                                        [0.0 .. 3.0]
(default: 1.1)
GAMMA_NS            float   3   0   rw  Over-subtraction factor of
stationary noise. min .. max attenuation
GAMMA_NS_SR         float   3   0   rw  Over-subtraction factor of
stationary noise for ASR.
                                                        [0.0 .. 3.0]
(default: 1.0)
```

```
HPFONOFF            int 3   0   rw  High-pass Filter on microphone signals.
                                                        0 = OFF
                                                        1 = ON - 70 Hz
cut-off
                                                        2 = ON - 125 Hz
cut-off
                                                        3 = ON - 180 Hz
cut-off
MIN_NN              float   1   0   rw  Gain-floor for non-stationary noise
suppression.
                                                        [−inf .. 0] dB
(default: −10dB = 20log10(0.3))
MIN_NN_SR           float   1   0   rw  Gain-floor for non-stationary noise
suppression for ASR.
                                                        [−inf .. 0] dB
(default: −10dB = 20log10(0.3))
MIN_NS              float   1   0   rw  Gain-floor for stationary noise
suppression.
                                                        [−inf .. 0] dB
(default: −16dB = 20log10(0.15))
MIN_NS_SR           float   1   0   rw  Gain-floor for stationary noise
suppression for ASR.
                                                        [−inf .. 0] dB
(default: −16dB = 20log10(0.15))
NLAEC_MODE          int 2   0   rw  Non-Linear AEC training mode.
                                                        0 = OFF
                                                        1 = ON - phase 1
                                                        2 = ON - phase 2
NLATTENONOFF        int 1   0   rw  Non-Linear echo attenuation.
                                                        0 = OFF
                                                        1 = ON
NONSTATNOISEONOFF   int 1   0   rw  Non-stationary noise suppression.
                                                        0 = OFF
                                                        1 = ON
NONSTATNOISEONOFF_SR    int 1   0   rw  Non-stationary noise suppression for
ASR.
                                                        0 = OFF
                                                        1 = ON
RT60                float   0.9 0.25    ro  Current RT60 estimate in seconds
RT60ONOFF           int 1   0   rw  RT60 Estimation for AES. 0 = OFF 1 = ON
SPEECHDETECTED      int 1   0   ro  Speech detection status.
                                                        0 = false (no
speech detected)
                                                        1 = true (speech
detected)
STATNOISEONOFF      int 1   0   rw  Stationary noise suppression.
                                                        0 = OFF
                                                        1 = ON
STATNOISEONOFF_SR   int 1   0   rw  Stationary noise suppression for ASR.
                                                        0 = OFF
                                                        1 = ON
```

```
TRANSIENTONOFF      int 1   0   rw  Transient echo suppression.
                                                        0 = OFF
                                                        1 = ON
VOICEACTIVITY       int 1   0   ro  VAD voice activity status.
                                                        0 = false (no
voice activity)
                                                        1 = true (voice
activity)
```

Audio-Umschaltung

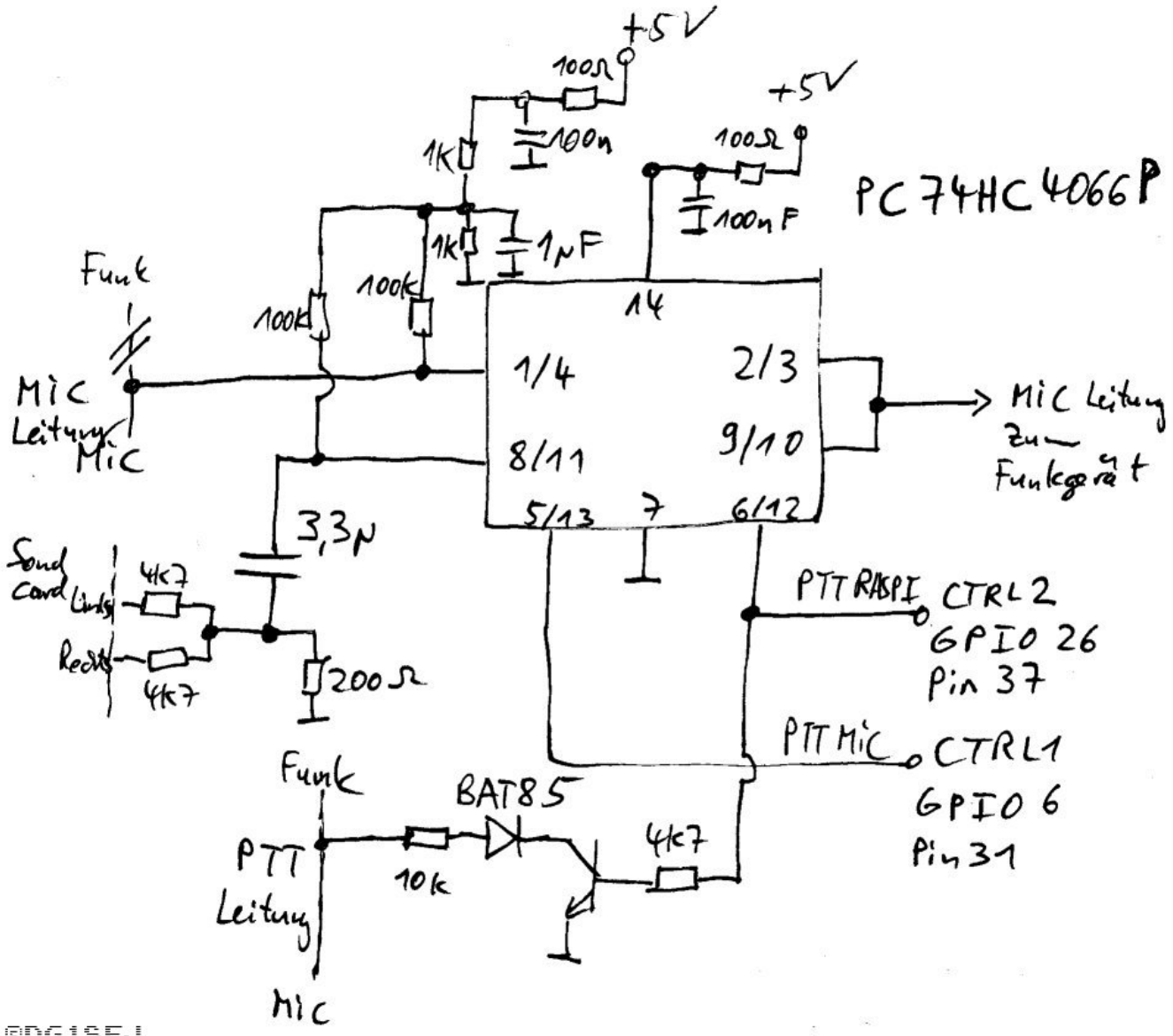
Der Wunsch bestand das nicht nur die Freisprecheinrichtung sondern auch das normale Handmikrofon benutzbar sein sollte ohne umstecken. Ergo benötigt man einen Audioumschalter. Die Lösung ist ziemlich simpel … Nutzung eines 4066 Gatters zur Umschaltung. Ein paar Anmerkungen dazu :

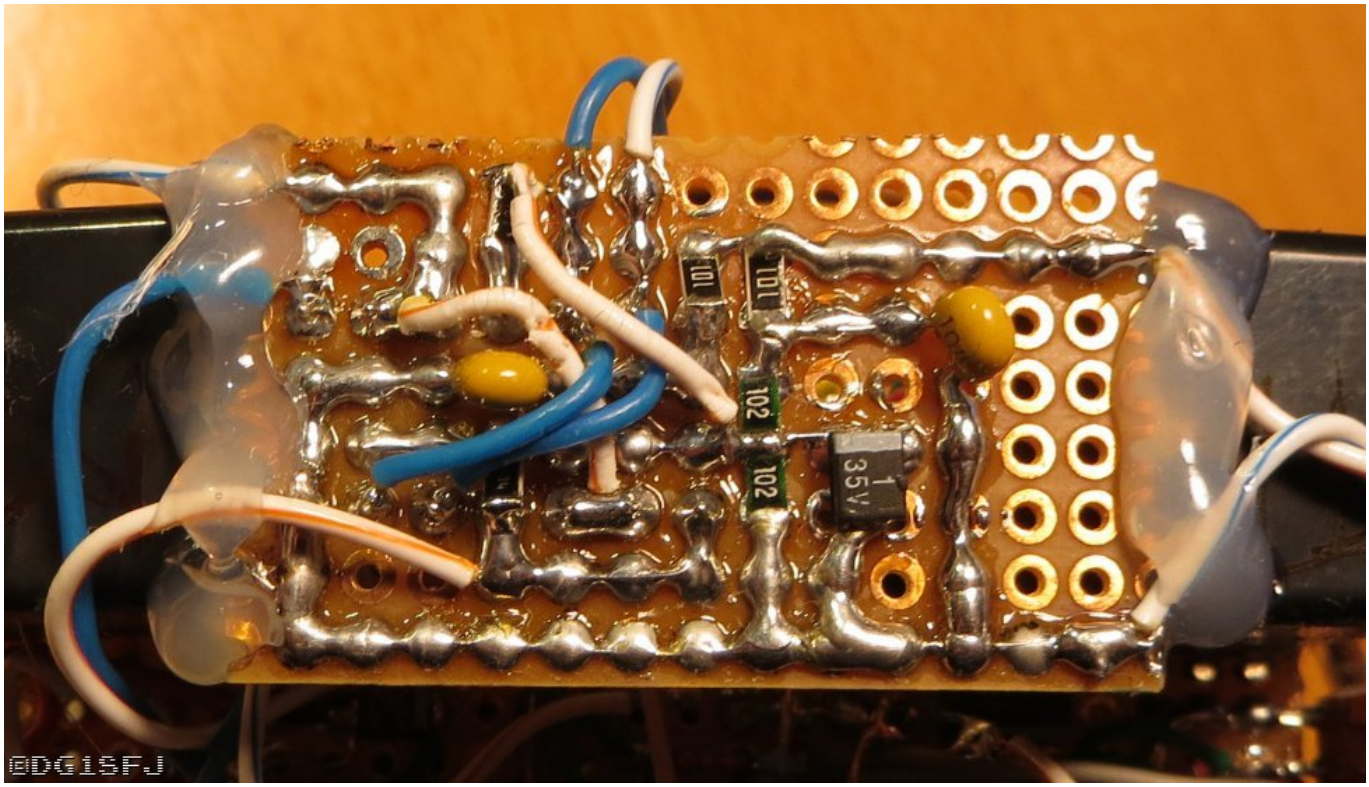
- Versorgungsspannung muss sauber sein (wird hier per Lowpass aus 100Ohm und 100nF gelöst)

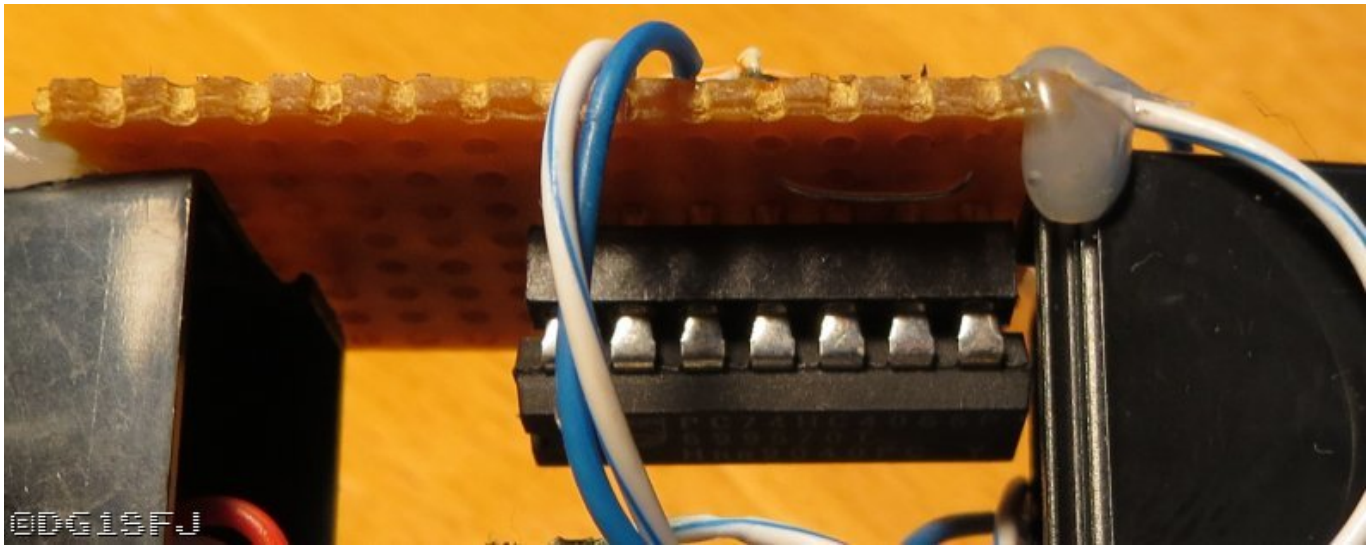
- die NF-Spannung muss in der Betriebsspannungsmitte liegen (aufkoppeln per C auf eine Hilfsspannung)

- Hilfsspannung mit Betriebsspannungsmitte (2x1k Spannungsteiler, 1uF Stabi-C, 100Ohm 100nF Lowpass)

- Parallelschalten von jeweils 2 Transfer-Gates zur reduktion des Rdson


+5V
100Ω
100n
1K
+5V
100Ω
100nF
PC74HC4066P
1K
1nF
Funk
100K
100K
14
1/4
2/3
8/11
9/10
5/13
7
6/12
MIC
Leitung
MIC
MIC Leitung
zum
Funkgerät
3,3n
Sound
Card
Links
4k7
Rechts
4k7
200Ω
PTT RASPI
CTRL2
GPIO 26
Pin 37
PTT MIC
CTRL1
GPIO 6
Pin 31
Funk
PTT
Leitung
MIC
10k
BAT85
4k7
©DG1SFJ





Gesteuert wird die Umschaltung per Software wie folgt :

GPIO 26 (Pin37) : PTT Raspi (High=Senden mit Audio vom Raspi, Low=RX), der GPIO hat ein Pulldown intern

GPIO 6 (Pin 31) : PTT Handmic (High=Senden mit Audio vom Mic, Low=RX), der GPIO hat ein Pullup intern

Per Default ist immer das Handmic aktiv, nur wenn der Raspi selber senden wird werden die Leitungen gewechselt.

Die Mic-Leitung hat im FTM400 ein 100nF Koppel-C damit brauchen wir beim Ausgang keines zusätzlich.

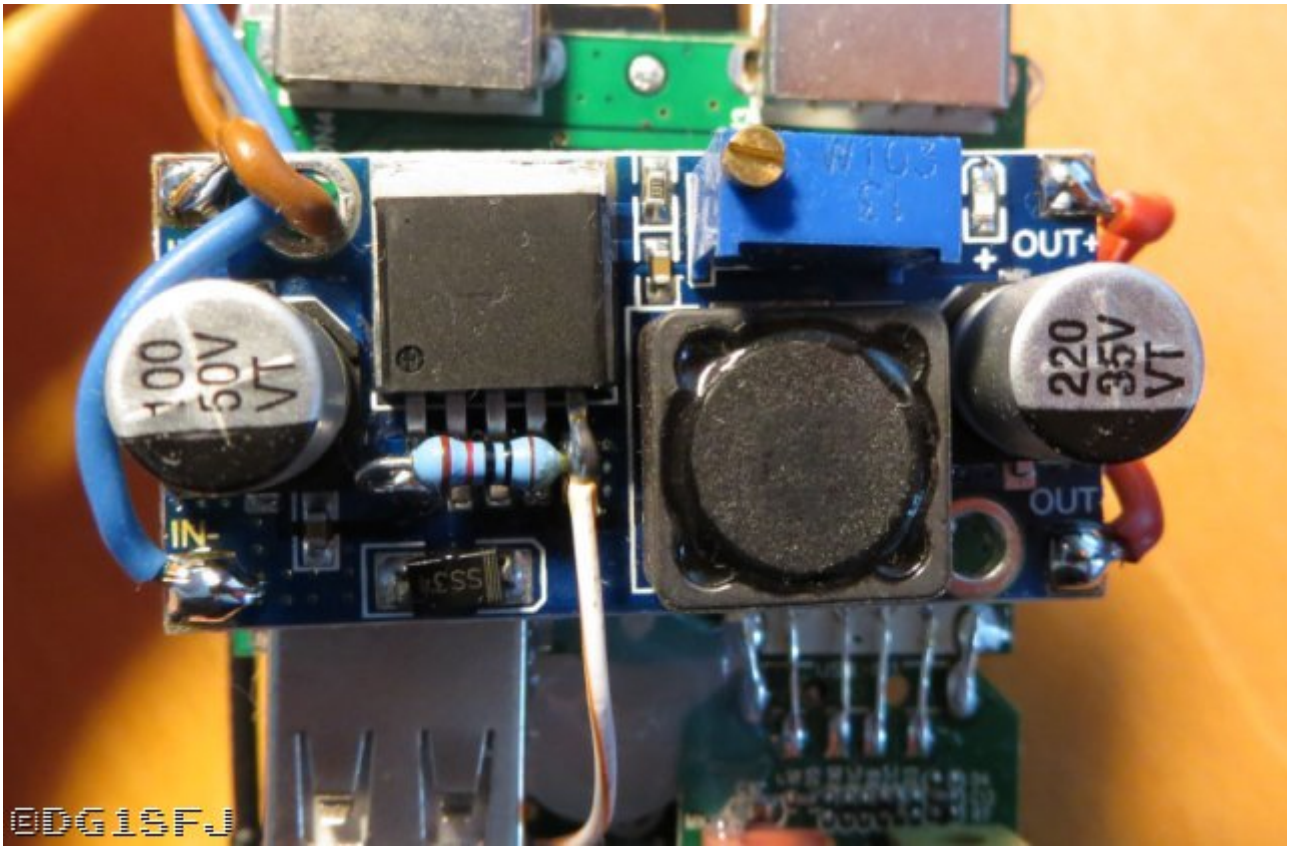
Von der Soundkarte ist noch die Summierung von Rechter/Linker Kanal sowie ein Dämpfung

eingebaut mit Koppel-C.

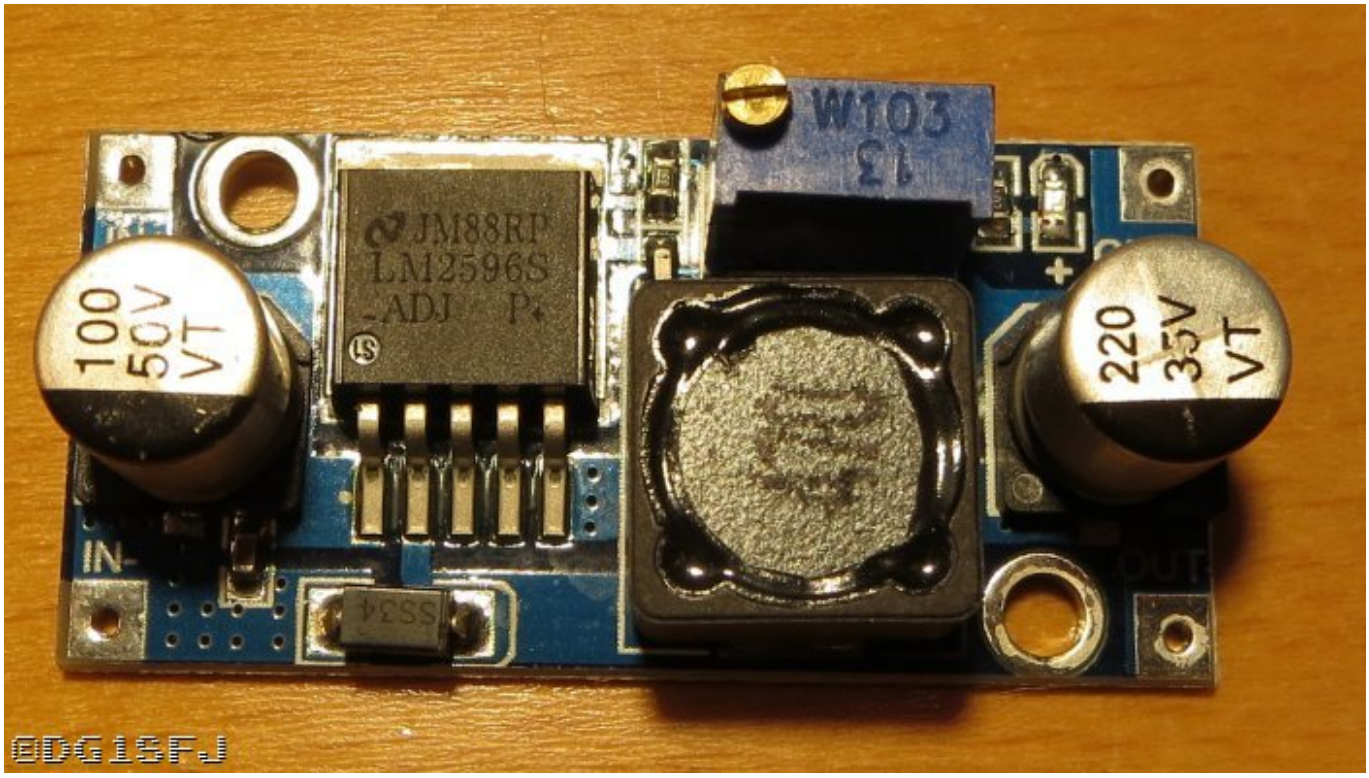
GPIO26 steuert zusätzlich einen Pulldown nach Masse (mit BAT85 Diode) um die PTT Leitung herunterzuziehen wenn gesendet werden soll.

DCDC

Hier greifen wir auf eine fertige Modulplatine aus China zurück. Sie wandelt die 12V auf 5V für den Raspberry Pi Zero. Um später per Raspi den DCDC Wandler steuern zu können muss die Leiterplatte leicht umgebaut werden. Der rechte Pin des LM2596S liegt auf Masse was „enabled“ bedeutet. Dieser Pin wird angehoben und per 10kOhm auf die Eingangsspannung gelegt. Der Pin wird dann mit einer Leitung weiter zum Raspi verdrahtet der die Selbsthaltung übernimmt.



Hier nochmal die Original-Platine von oben und unten :



Der Pins rechts aussen liegt auf Masse.

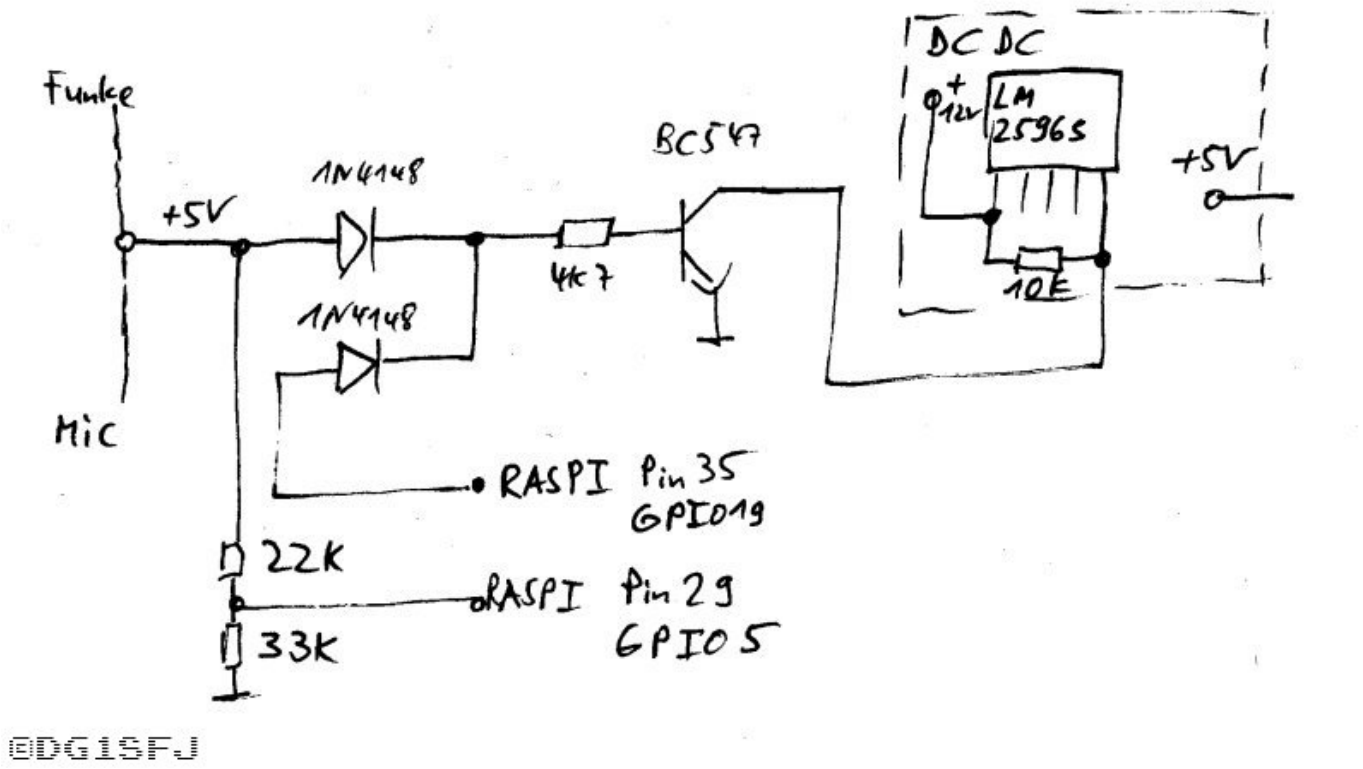
Hier noch die Unterseite :

Sobald am Mikrofon die +5V Leitung auf „High“ geht, wird über die Dioden-Veroderung und den Transistor der DCDC Wandler aktiviert. Wenn der Raspi läuft gibt dieser ein „High“ auf GPIO19 aus um sich selbst am Leben zu halten. Über den GPIO5 wird dann immer zurückgelesen ob die +5V noch anliegen. Wenn dies nicht der Fall ist, das Funkgerät also abgeschaltet hat, fährt der Raspi das Betriebssystem kontrolliert herunter und nimmt sich selber den Saft weg.

Damit das klappt beim rauf und runterfahren muss man wissen das GPIO19 (Pin35) Raspi-Intern einen

Pull-Down Widerstand besitzt und der GPIO5 (Pin29) einen Pull-Up Widerstand. Steht beschrieben im Broadcom BCM2835 ARM Peripherals unter 6.2 Alternative Function Assignments.

Hier noch der Schaltplan dieses Schaltungs-Teils :



Der Prozess des einschaltens dauert ca : 35sec

Der Prozess des abschaltens dauert ca : 10sec

Nicht besonders schnell aber ausreichend.

Raspi Pinout

In der Schaltung werden folgende Pins benutzt :

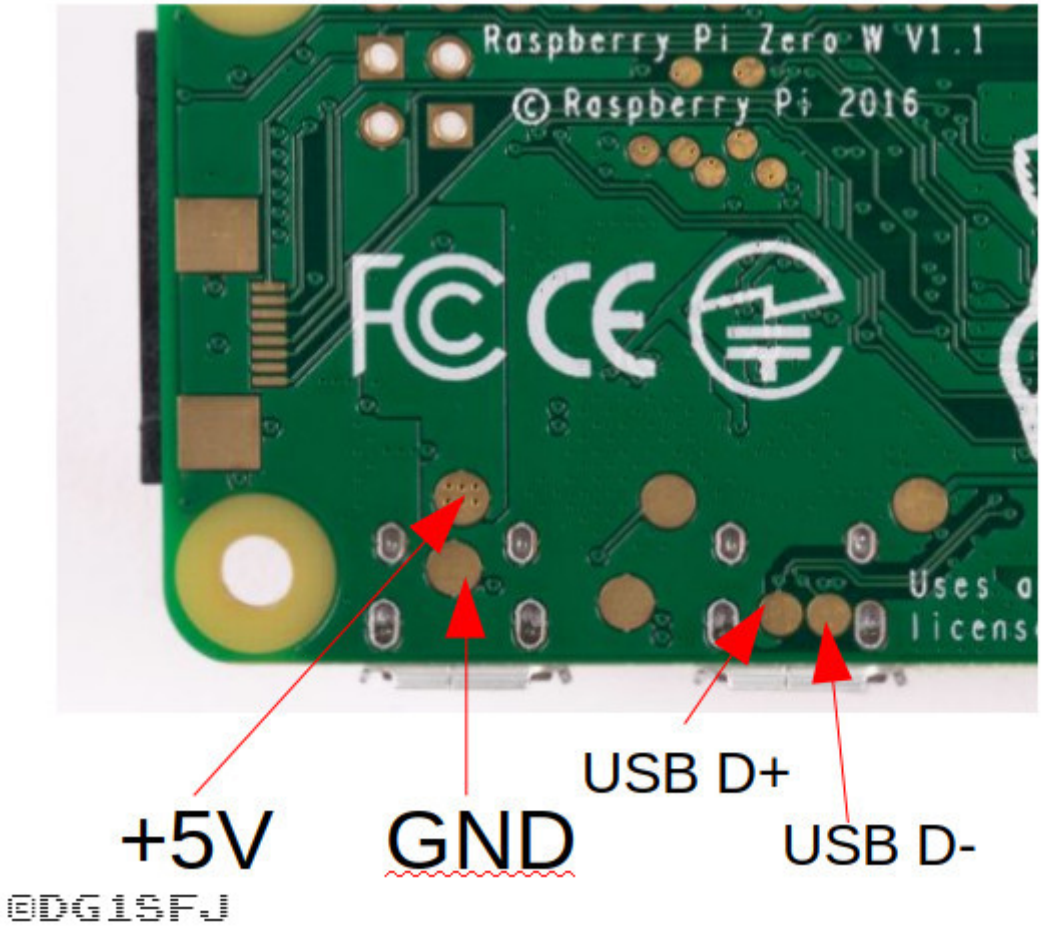
```
1 - 3.3V - Versorgung für die MCP4725 DACs

2 - 5.0V - Versorgung für den Audio-Umschalter

3 - SDA I2C zu den MCP4725

5 - SCL I2C zu den MCP4725

6 - GND

9 - GND

20 - GND

25 - GND
```

```
29 - GPIO5 - Sense-Eingang für die Erkennung ob Funke an oder aus

30 - GND

31 - GPIO6 - Select Audio Pfad Hand-Mic

34 - GND

35 - GPIO19 - Selbsthaltung an die Diodeveroderung der Versorgung

37 - GPIO26 - Select Audio Pfad Soundkarte / Respeaker

39 - GND
```

Raspi Versorgungspads

Auf der Rückseite des Raspi gibt es die Testpunkte über die man die Versorgung der +5V/Masse einspeisen kann sowie der Abgriff der USB+/- für den USB Hub.

Damit brauchen wir dann auch die beiden Micro-USB Buchsen nicht mehr auf dem Raspi und alles ist fest verdrahtet.



Raspi Skripte

Beim Start des Raspi wird über crontab folgende Zeile gestartet :

```
@reboot /home/pi/connectspeaker.sh
```

Die Datei „connectspeaker.sh“ startet PulseAudio und lädt ein Modul mit dem USB Respeaker als Audio Quelle sowie als Audio Senke unsere USB Soundkarte.

Beide sind dann miteinander verbunden (loopback). Mit „pacmd“ wird noch die Lautstärke der Soundkarte eingestellt. 40000 war noch ok, keine Übersteuerung des Funkgerätes.

Auf der Leiterplatte ist aber wie vorher beschrieben noch ein Spannungsteiler.

Dann wird in einer Schleife unser Python Programm gestartet. Wenn dieses wegen eines Fehler beendet wird (z.B. kein Kopplung mit dem Bluetooth Fernsteuergerät) dann wird einfach wieder neu versucht. Solange bis alles verbunden ist (ReSpeaker an USB und Bluetooth Fernbedienung gekopplt)

Das eigentliche Steuerprogramm ist das das ftmctrl10.py gestartet.

```
#!/bin/bash

pactl info
pactl load-module module-loopback source="alsa_input.usb-
SEEED_ReSpeaker_4_Mic_Array__UAC1.0_-00.analog-mono" sink="alsa_output.usb-
C-Media_Electronics_Inc._USB_Audio_Device-00.analog-stereo" latency_msec=1
pacmd set-sink-volume "alsa_output.usb-C-
Media_Electronics_Inc._USB_Audio_Device-00.analog-stereo" 40000
while true
do
sudo python ftmctrl10.py
echo "Restart Programm"
done
```

Python Programm

Da sich das Python Programm in den letzten Zügen der Entwicklung befindet, werde ich es hier nicht zum Download anbieten 🙂

Bei den Libraries habe ich natürlich auf die Vorarbeiten anderer Entwickler aufgesetzt.

MCP4725 : Adafruit MCP4725 Library : https://github.com/adafruit/Adafruit_Python_MCP4725

usb_4_mic_array : https://github.com/respeaker/usb_4_mic_array

pixel_ring : https://github.com/respeaker/pixel_ring