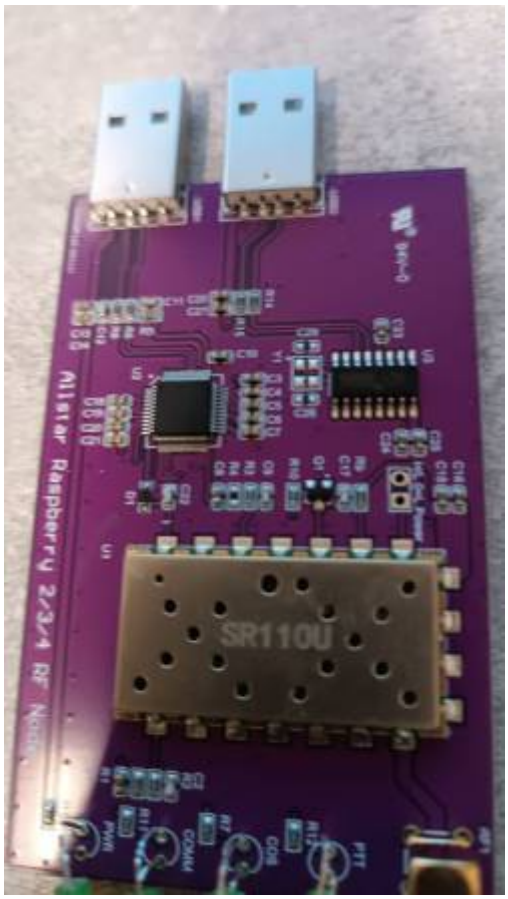
Bitte beachten: Ich bin und werde kein Supporter für diesen Hotspot ! Ich gebe nur Hinweise für dessen Inbetriebnahme, da ich ihn selber auch einsetze, allerdings besitze ich die ursprüngliche Version mit dem SA818.

FM/analog Hotspot SHARI für SVXLINK & Raspberry Pi - geänderte/andere Version mit SR110U anstatt SA818

letzte Aktualisierung: 27.1.2023

Es scheint eine weitere Version zu geben, die nicht mit einem SA818 ausgeliefert wird, sondern mit einem SR110U. Hier ist das entsprechende Datenblatt zu diesem FM-TRX-Modul.
Der Hersteller dieses Moduls ist Shen Zhen Sunrise Electronics Co. Ltd.



Ansonsten gilt alles wie hier bereits in der Version mit dem SA818 beschrieben, das bitte auch durcharbeiten, da es sich hier nur um eine Ergänzung bzw. Erweiterung handelt wegen der Ersetzung des SA818 mit dem SR110U:

1. andere/unterschiedliche Programmierung des FM-Moduls SR110U gegenüber dem SA818
2. Anpassung zweier Parameter in der svxlink.conf
3. Audiopegel neu auf das geänderte FM-Modul anpassen

⚠ Programmiertools für das SA818 funktionieren mit diesem Modul SR110U NICHT, da ein anderer bzw. geänderter Befehlssatz verwendet wird. Ich habe weiter unten ein Python3-Script erstellt, womit man dieses Modul programmieren kann. Alles Wichtige zur Programmierung ist im Datenblatt zu finden. Lest also dort nach, ich habe es verlinkt.

Anmerkungen zu dieser Version

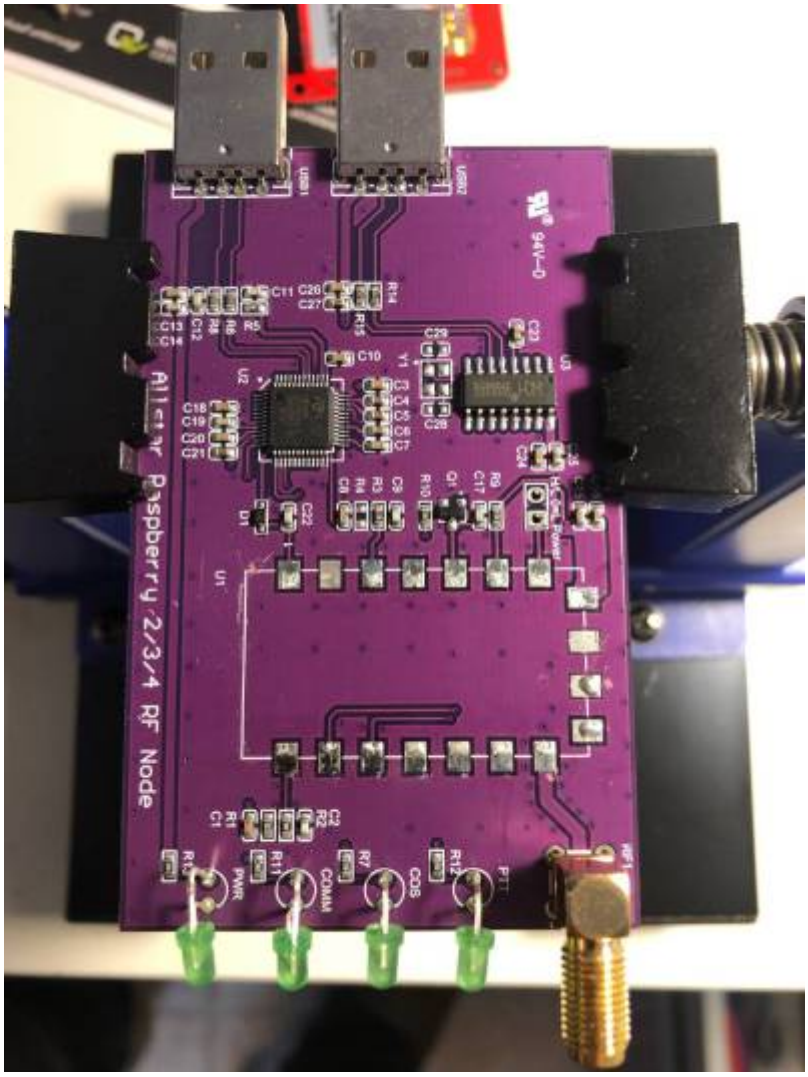
Obwohl nach wie vor in der Produktbeschreibung dieses Gerätes von einem SA818 die Rede ist,

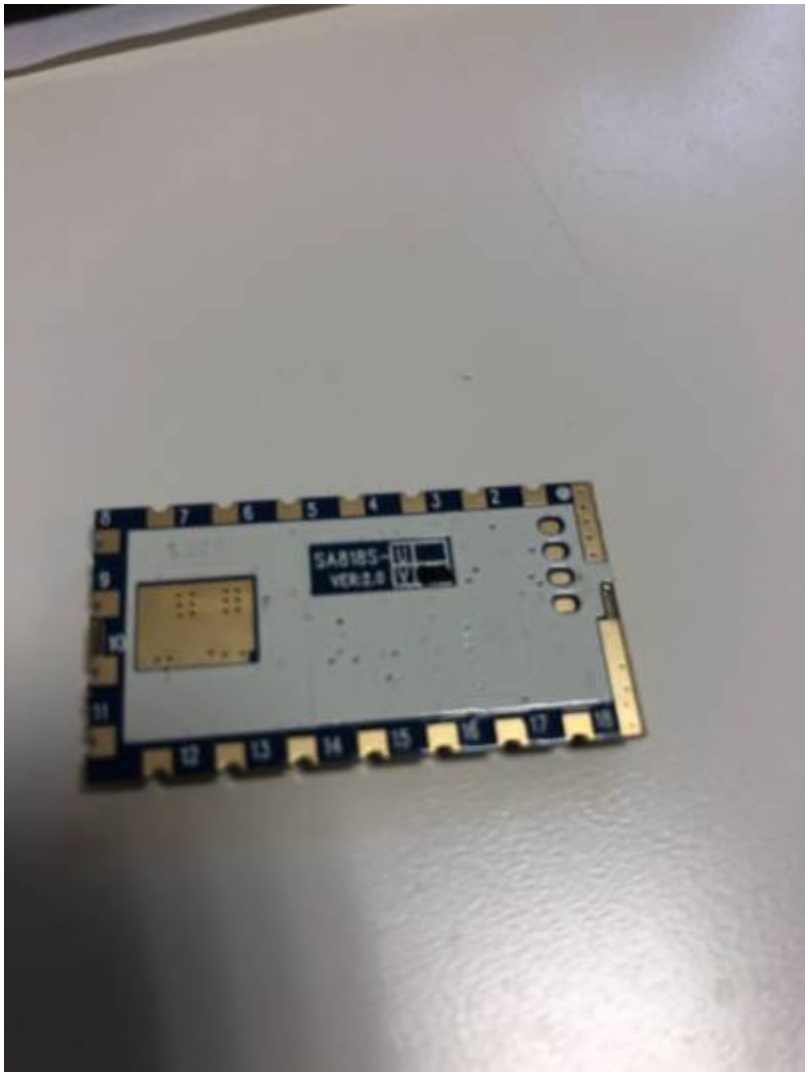
haben die Chinesen mal kurzerhand die Ausstattung geändert. Ärgerlich und eigentlich ein Grund zur Rücksendung, denn das Produkt ist nicht mehr das, wie es technisch beschrieben und angeboten wurde. Leider mit dem SR110U anstatt SA818 auch ein qualitativer Rückschritt.

Was mich an diesem Modul SR110U stört - man kann es nicht mit flataudio programmieren, auch finde ich (subjektiv) die Audioqualität *deutlich schlechter* als bei den SA818, die dort sehr gut war (die SR110U klingen für meine Ohren ingesamt etwas "dumpfer", also mit geringerem Anteil des höheren NF-Frequenzspektrums verglichen mit einem SA818U/V). ~~Weiterhin erscheint mir die CTCSS-Erkennung auch nicht ganz so zuverlässig zu funktionieren wie bei den SA818, womit ich nie Probleme bei der CTCSS-Detection hatte~~. Der einzige Vorteil, die kleine Sendeleistung 0,5W lässt sich per Software einstellen, was wiederrum beim SA818 nicht möglich war. Grundsätzlich spräche aber nichts dagegen, dieses Modul SR110U zu entlöten und stattdessen wieder ein SA818 zu installieren. Für den versierten Funkamateur stellt das sicher kein Problem dar, die Module sind Pin-kompatibel und damit austauschbar.
~~Ich habe aber selbst (noch) keine Variante mit dem SR110U hier vorliegen, deswegen sind das nur Erkenntnisse, die ich aufgrund meiner Hilfestellungen herausfinden konnte, das ist also alles "unter Vorbehalt" meinerseits zu interpretieren~~.
~~Meine zwei noch nachbestellten SHARI sind aktuell noch im Zulauf, wenn sie eingetreffen, teste und konkretisiere ich das alles erneut, wenn ich mir dann selbst ein genaues Bild davon gemacht habe~~.

Update 25.1.2023:
Inzwischen waren meine beiden nachbestellten SHARI auch eingetroffen, wie vermutet jetzt auch mit SR110U bestückt. So konnte ich einen direkten Vergleich zwischen dem mit dem SA818S und dem mit SR110U machen. Hier meine Erkenntnisse:

- die vermuteten Probleme mit CTCSS konnte ich nicht reproduzieren, die Erkennung war zuverlässig, das war aber auch schon das einzig Positive
- was ich bereits zu der FM-Audio-Qualität schreib, hat sich leider bestätigt. Die SA818S haben eine wesentlich bessere FM-Audioqualität gegenüber den SR110U, sowohl was den RX aber auch den TX angeht. Das SR110U ist bis jetzt das am schlechtesten "klingende" FM-TRX-Modul, was ich kenne. Auch in dem sog. DJSpot wird dessen kleinerer Bruder, das SR105U verbaut, was auch nicht viel besser klingt. Für meine(!) Ohren leider ein No-Go, sri.
- inzwischen habe ich das SR110U entlötet und wieder gegen ein SA818S (Kosten ca. 9€) ersetzt, für mich(!) war das Audio des SR110U einfach nicht akzeptabel - da klingt ja der Digitalfunk besser 🤣


Allstar Raspberry 2/3/4 RF Node
PWR
COMM
COS
PTT



Programmierung SR110U

Dazu brauchen wir erstmal Python3 (falls noch nicht installiert):

```
$ sudo apt-get install python3-dev python3-pip
$ sudo pip3 install pyserial
```

Hier ein Beispiel-Script `sr110u-running.py` zur Programmierung des SR110U (bitte mit einem Editor Eurer Wahl erstellen, z.B. nano, vi oder mcedit):

```
#!/usr/bin/env python3

import serial
import time

serport = '/dev/ttyUSB0'

baud = '9600'
```

```
channelspace = '1'      # 0=12.5kHz, 1=25kHz

# Frequenz bitte nach Belieben anpassen, kHz bitte immer 4stellig angeben -
Bandplan beachten !
rxfreq = '430.0250'     # TX frequency
txfreq = rxfreq         # Same as rx freq - we work simplex

# sinnvolle Werte liegen zwischen 2 und 4, je kleiner desto empfindlichere
SQL, 0 bedeutet SQL immer offen
squelch = '2'           # 0-8 (0 = open)

# CTCSS Konfiguration
txcxcss = '4'        # CTCSS TX 77Hz nach Tabelle aus dem Datenblatt
entnommen
rxcxcss = '4'        # CTCSS RX 77Hz
# txcxcss = rxcxcss

# DCS anstatt CTCSS - nicht empfohlen
# txcxcss = '023N'        # CTCSS / CDCSS TX
# rxcxcss = '023N'        # CTCSS / CDCSS RX

# ein Wert von 7 ist weitgehend optimal als Sound-Input
volume = '7'              # betweeen 0..8

# ab hier am besten nichts editieren - es sei denn, man weiss was man tut
und warum :)

ser = serial.Serial(serport, baud, timeout=2)
print('Opening port: ' + ser.name)

print ('\r\nConnecting...')
ser.write(b'AT+DMOCONNECT\r\n')
output = ser.readline()
print ('reply: ' + output.decode("utf-8"))

time.sleep(3)

print ('\r\nRESET...')
ser.write(b'AT+DMOREST\r\n')
output = ser.readline()
print ('reply: ' + output.decode("utf-8"))

time.sleep(5)

print ('\r\nConnecting...')
ser.write(b'AT+DMOCONNECT\r\n')
output = ser.readline()
print ('reply: ' + output.decode("utf-8"))

time.sleep(3)
```

```
print ('\r\nConfiguring radio settings...')
config = 'AT+DMOSETGROUP={},{},{},{},{},{},4\r\n'.format(channelspace,
rxfreq, txfreq, rxcxcss, squelch, txcxcss)
print (config)
ser.write(config.encode())
output = ser.readline()
print ('reply: ' + output.decode("utf-8"))

time.sleep(3)

# Programmierung und Parametrisierung ist dem Datenblatt des SR110U zu
entnehmen

print ('\r\nPower SAVE OFF...')
ser.write(b'AT+DMOAUTOPOWCONTR=1\r\n')
output = ser.readline()
print ('reply: ' + output.decode("utf-8"))

time.sleep(3)

print ('\r\nVOX OFF...')
ser.write(b'AT+DMOSETVOX=0\r\n')
output = ser.readline()
print ('reply: ' + output.decode("utf-8"))

time.sleep(3)

print ('\r\nMic setting level 6...')
config = 'AT+DMOFUN={},6,0,0,0\r\n'.format(squelch)
print(config)
ser.write(config.encode())
output = ser.readline()
print ('reply: ' + output.decode("utf-8"))

time.sleep(3)

print ('\r\nSetting volume...')
config = 'AT+DMOSETVOLUME={}\r\n'.format(volume)
print(config)
ser.write(config.encode())
output = ser.readline()
print ('reply: ' + output.decode("utf-8"))

time.sleep(3)

print ('\r\nGetting Module Version...')
ser.write(b'AT+DMOVERQ\r\n')
output = ser.readline()
print ('reply: ' + output.decode("utf-8"))

print ('\r\nProgrammierung abgeschlossen...')
```

Nach dem Erstellen dieses Scripts muss bzw. sollte dieses ausführbar gemacht werden:

```
$ sudo chmod +x sr110u-running.py
```

CTCSS-Tabelle des SR110U-Moduls für die Parameter `txcxcss` (Encoding für TX) und `rxcxcss` (Decoding für RX) im obigen Python-Script zur Programmierung des SR110U-Moduls:

CH	FREQ.	CH	FREQ.	CH	FREQ.
1	67.0	13	103.5	26	162.2
		14	107.2	27	167.9
2	71.9	15	110.9	28	173.8
3	74.4	16	114.8	29	179.9
4	77.0	17	118.8	30	186.2
5	79.7	18	123.0	31	192.8
6	82.5	19	127.3	32	203.5
7	85.4	20	131.8	33	210.7
8	88.5	21	136.5	34	218.1
9	91.5	22	141.3	35	225.7
10	94.8	23	146.2	36	233.6
11	97.4	24	151.4	37	241.8
12	100.0	25	156.7	38	250.3

Zu empfehlen ist eher CTCSS als DCS. Bei DCS wird ein digitaler Datenstrom mit einer fixen Bitrate von 134,4 bps im unteren Audiobereich wiederholend übertragen. Das führt zwangsweise zu einer Verzögerung, bis erkannt wird, das der Squelch öffnen soll. Bei CTCSS liegt diese Verzögerung etwa bei 200..250ms oder auch weniger, bei DCS jedoch bereits um die 350..400ms. Man muss also eine kleine Pause zwischen PTT am Funkgerät drücken und Sprechbeginn einkalkulieren.

Audiopegel einstellen

Nach der Programmierung des Moduls muss man noch die Audiopegel mittels alsamixer korrekt einstellen, das bitte nicht vergessen:

```
$ alsamixer -c 1
```

Anpassungen der svxlink.conf

Da diese Module im Gegensatz zu den SA818 kein flataudio können, sollte in der `/etc/svxlink/svxlink.conf` für den LocalRX bzw. LocalTX folgendes eingestellt werden:

```
[Rx1]
DEEMPHASIS=0
```

```
[Tx1]
PREEMPHASIS=0
```

Verbesserung des Verhaltens der Rauschsperre des FM-TRX-Moduls

Bei den SA818 wie auch bei den SR110U schliesst die Rauschsperre (SQL oder CTCSS) relativ langsam, was zur Folge hat, das die Gegenstelle nach dem Loslassen der eigenen PTT noch für einen kurzen Moment ein Rauschen übertragen bekommt. Um diesen Effekt zu verrringern bzw. zu vermeiden, sind in der `svxlink.conf` folgende Anpassungen zu machen:

```
[Rx1]
SQL_START_DELAY=50
# HS: stoppt gelegentliches unkontrolliertes Auftasten des Senders
wahrscheinlich durch sporadische Spikes am GPIO im Bereich 0ms..100ms
# ein DELAY von 200ms unterbindet das effektiv
SQL_DELAY=200
# sofort SVXLINK das Schliessen der SQL melden, Wert bedeutet 0ms
SQL_HANGTIME=0
# die letzten 100ms Audio abschneiden evtl. mit Werten zwischen 100 bis 200
experimentieren
SQL_TAIL_ELIM=100
```

---

73 Heiko, DL1BZ
Januar 2023