Software

Signalverarbeitung

Durch einfache, kleine Module wird eine Plug&Play-Pipeline-Architektur geschaffen. Alle Module unterstützen als Ein/Ausgabe von Audio-Dateien entweder Dateien oder stdin/stdout. So ist das in Linux übliche Piping möglich. Das Datenformat für Audio wird festgelegt auf Little Endian, 16 Bit. Die Samplerate wird standardmäßig von jeder Software zu 48kHz angenommen, kann aber durch Setzen der Umgebungsvariable XPLORER_FS überschrieben werden. Ausnahme sind hier die Erzeugung von APRS und SSTV. Diese werden aus Performancegründen mit der kleinstnötigen Samplerate erzeugt und danach auf die genutzte Samplerate skaliert.

Bestandteile

Die Software besteht aus kleinen Modulen, die zentral von einer Ablaufsteuerung koordiniert werden.

- Ablaufsteuerung (Aufnehmen und Abspeichern von Bildern, Erzeugung SSTV und APRS, Aussendung) - Python
- LO-Steuerung (Trägerfrequenzerzeugung durch Einstellung GP0CLK) - C
- APRS-Erzeugung (Mono-NF) - C
- Robot36-Erzeugung (Mono-NF) - C
- FM-IQ-Modulation (Mono-NF zu Stereo-IQ) - C
- Sprachsynthese (Aneinanderreihen von Zahlen-WAVs mit Pause) - Python / C
- Audioplayer für Stereo-IQ-Daten (aplay) - builtin
- Resampling von Audiodateien (resample) - builtin
- Konvertieren der Webcambilder zu 320×240 für Robot36 (convert) - builtin

IQ-Modulation

- mono-Audiodatei frequenzmodulieren (Frequenzhub und Mittenfrequenz einstellbar)
- stereo-Audiodatei ausgeben

PIQ

- GP0CLK auf Trägerfrequenz einstellen
- Stereo-WAV (beinhaltet IQ-Daten) über Soundkarte abspielen
- GP0CLK abschalten

APRS

- übergebene Position (Lat, Lon, Höhe, Temperatur) in WAV schreiben
- TODO Rewrite in C

SSTV

- übergebene Bilddatei in Robot-36 kodierte WAV wandeln
- OK, funktioniert
- TODO Performancetests

Ablaufsteuerung

Im Vorbereitungsbetrieb:

- Start der Software
- Warten auf GPS Fix
- LED an
- Testaussendungen (APRS, Pause, SSTV, Pause, APRS)
- LED blinken
- 1 Minute warten
- Missionsstart (LED aus)

Im Missionsbetrieb:

- Speichern aktuelle Positionsinformation
- Aussendung APRS auf 144.800 MHz (2 Sek)
- 1 Bild speichern
- Ansage 3xLAT, 3xLON auf 145.200 MHz (10 Sekunden)
- Aussendung SSTV auf 145.200 MHz (36 Sekunden)
- Pause (10 Sekunden)
- Aussendung APRS (3 Sek)
- Pause 20 Sekunden
- nach X Minuten (je nach Version) wird die Nutzlast abgesprengt
- nach X + TBD Minuten wird der Raspberry Pi heruntergefahren und/oder abgeschaltet

Absprengung:

- Mikrotaster abfragen
- wenn geschlossen: Heizung an bis er offen ist
- wenn offen: Heizung für ?? Sekunden an
- Aussenden Signalton!

Stromverbrauch nach Herunterfahren

- Muss man den Raspberry Pi von Spannung trennen, oder braucht er nach Herunterfahren nur noch wenig Strom? –> scheinbar ist es so, dass er nach dem runterfahren noch mehr strom braucht - also besser anlassen.
- Raspberry + IQ-Mixer + Soundkarte
    - Pon = 10V*0,23A = 2,3W
    - Poff = 10V*0,09A = 0,9W
    - Verringerung der aufgenommenen Leistung auf unter die Hälfte

Vorbereitung Temperatursensor

- apt-get install i2c-tools lmsensors
- modprobe i2c-dev echo i2c-dev » /etc/modules
- i2cdetect -y 1 –> Anzeige des LM75
- echo lm75 0x48 > /sys/class/i2c-adapter/i2c-1/new_device