SvxLink-UDP

SvxLink-UDP

svxlink erlaubt die Anbindung von Audio (sowohl als "Mic", wie als "Lautsprecher") über UDP.

Dazu ist etwa folgender Abschnitt in der Konfiguration von svxlink notwendig:

```
[TxUDP]
TX_ID=TU
TYPE=Local
AUDIO_DEV=udp:127.0.0.1:4300
AUDIO_CHANNEL=0
PTT_TYPE=NONE
CARDS_CHANNELS =1
```

Die Daten werden mit zwei Kanälen mit jeweils 16 Bits kodiert. Pro Sekunde werden 48 UDP-Pakete mit einer Länge von 4032 Bytes übermittelt, dh. je Paket 1008 Samples mit einer Sample-Rate von 48.000 Samples/s, der wohl internen Sample-Rate von svxlink.

```
[2025-03-10 21:12:45.603] Received 4032 bytes from ('127.0.0.1', 53786):
8eff00006eff000037ff0000fafe0000befe000099fe000094fe0000a7fe0000cefe0000fbfe00001eff000(
```

Der digitale Kanal kann auch parallel zu einem bestehenden Kanal erstellt werden, etwa durch eine Multi-TRX-Konfiguration:

```
[MultiTX]
TYPE=Multi
TRANSMITTERS=Tx1,TxUDP
```

Einfacher Python-Code zur Ausgabe der UDP-Daten:

```
import socket
from datetime import datetime

def start_udp_server(host='0.0.0.0', port=4300):
    # Create a UDP socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

    # Bind the socket to the address and port
    server_address = (host, port)
    sock.bind(server_address)

    print(f"Starting UDP server on {host}:{port}. Waiting for data...")

    try:
        while True:
            # Receive data from the client
            data, address = sock.recvfrom(65535)

            # Get the current timestamp with milliseconds
            timestamp = datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f')[:-3]
```

```
            # Convert the data to hexadecimal format
            hex_data = data.hex()

            # Print the timestamp and the received hex data
            print(f"[{timestamp}] Received {len(data)} bytes from {address}: {hex_data}'

    except KeyboardInterrupt:
        print("\nServer is shutting down.")
    finally:
        sock.close()

if __name__ == '__main__':
    start_udp_server()
```

Einfacher Pyhton-Code um die Daten auf 8000 Samples/s zu konvertieren:

```
import socket
import numpy as np
from scipy import signal
import wave

def low_pass_filter(data, cutoff, fs, order=5):
    nyquist = 0.5 * fs
    normal_cutoff = cutoff / nyquist
    b, a = signal.butter(order, normal_cutoff, btype='low', analog=False)
    return signal.lfilter(b, a, data)

def start_udp_server_with_processing(host='0.0.0.0', port=4300, output_file='output.wav
    sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    sock.bind((host, port))

    print(f"Listening on {host}:{port}")

    out_wave = wave.open(output_file, 'wb')
    out_wave.setnchannels(1)
    out_wave.setsampwidth(2)
    out_wave.setframerate(8000)

    try:
        while True:
            data, _ = sock.recvfrom(4032)
            samples = np.frombuffer(data, dtype=np.int16)[::2]

            filtered_samples = low_pass_filter(samples, cutoff=3400, fs=48000)

            downsampled_samples = signal.resample_poly(filtered_samples, up=1, down=6)

            out_wave.writeframes(downsampled_samples.astype(np.int16).tobytes())

    except KeyboardInterrupt:
        print("\nServer shutting down.")
    finally:
        sock.close()
        out_wave.close()

if __name__ == '__main__':
    start_udp_server_with_processing()
```

Das Downsampling und die Ratenadaption basiert auf Samples von lediglich 20ms, damit kommt es zu Artifakten.