Testing Signal Processing Techniques for Digital VHF/UHF Transceiver in High-level SDR Programming Environment

Jiří Skořepa, Pavel Kovář, Pavel Puričer,
Faculty of Electrical Engineering, Czech Technical University in Prague
Czech Republic, Prague
skoreji3@fel.cvut.cz

***Abstract*—Nowadays, due to cost savings, flexibility or improvement of service quality, modern communication devices solely rely on the digital domain when it comes to signal processing. The paper presents a comfortable, fast and straightforward approach of testing and verifying developed digital signal processing techniques before their implementation into a digital mobile transceiver. The approach benefits from the concept of an Software-defined radio (SDR) by employing a high-level programming environment capable of direct analog radio frequency (RF) front end control in real-time via a user-defined software.**

***Index Terms*—digital signal processing; digital mobile radio; software-defined radio**

I. Introduction

This paper presents several tests performed during the development of a Very high frequency/Ultra high frequency (VHF/UHF) transceiver operating in analog narrowband frequency modulation (NFM) mode [1] and digital Digital mobile radio (DMR) mode [2]. The transceiver is developed together with an industrial partner T-CZ, a.s. under the project of innovation of a T-CZ's professional product — VHF and UHF transceiver for railway communication. The transceiver's design is in accordance with European Telecommunications Standards Institute (ETSI) standards for analog NFM [1] and for Digital Mobile Radio [2] communication. In the previous paper [3], we focused on Digital Up/Down Convertor (DUC, DDC). The next step is developing digital signal processing algorithms which are implemented in the microprocessor inside the developed transceiver.

Thanks to mathematical correspondence between the analog and digital domain of treatment the signal, over the past years, there has been an increasing trend of reducing the number of analog parts inside all communication devices in behalf of digital logic circuits to minimize construction and manufacturing costs. Another benefit of employing digital circuits results in the possibility of deploying advanced signal processing techniques and algorithms which would be burdensome or even impossible to implement in the analog domain. Hence, not only in many commercial off-the-shelf (COTS) transceivers, operations like DDC/DUC, signal detection, automatic gain control, modulation, equalization or voice compression are performed entirely inside microcontroller units (MCUs), Application-specific integrated circuits (ASICs) or Field-programmable gate arrays (FPGAs) of the digital transceiver.

The problem arises when the user-developed digital signal processing methods have to be tested and validated. In the analog domain, the RF front end can be readily tested using conventional RF measurement equipment. Testers for frequency modulation (FM) or DMR standards is mostly expensive equipment and often, due to the diverse nature of newly-applied digital signal processing (DSP) algorithms, the market does not offer a device capable of testing each specific algorithm. Therefore, the process of testing a device under development deploying the applied processing algorithms becomes burdensome.

Fortunately, we can profit from software-defined radio concept allowing us to use a device with an RF front end parameters corresponding to parameters of the developed device. Then, combining the SDR device with an appropriate high-level programming environment, we can test and verify our developed/applied DSP algorithms. In this paper, we present a useful and simple way to verify several of those DSP algorithms to be implemented in the developed VHF/UHF transceiver.

II. Tested Algorithms

A successful development of a DMR transceiver requires appropriate testing and implementation of several signal processing techniques. From a practical point of view, testing the algorithms only at the level of simulation in Matlab, Octave or Scilab may not reveal all the weaknesses of the algorithms and testing the algorithms after direct implementation inside the MCU or FPGA of the developed transceiver may be vastly time-consuming. Hence, it is convenient to perform more realistic examinations of the tested algorithm. The developed transceiver, which can work in DMR and legacy NFM mode, is composed of several digital processing blocks; nevertheless, in this paper, we focus on testing the following three blocks.

ISBN 978-80-261-0892-4 © University of West Bohemia, 2020

A. FM Demodulator

The signal in the legacy mode [1] is FM modulated; in the DMR mode, the standard [2] defines 4-state frequency shift keying (FSK) modulation. Samples of a signal complex envelope s_I, s_Q coming from the DDC block is forwarded to an FM demodulator. The FM demodulator, is realized as a frequency discriminator [4] described as

$$s_\text{NF}[n] = \frac{\text{atan2}(s_\text{cross}[n], s_\text{dot}[n])}{2\pi T_s} \tag{1}$$

where $n = 0, 1, ..., N-1$ is the time-domain index of the sample, atan2 denotes four-quadrant inverse tangent function and T_s is the sampling period of the incoming IQ samples — in our case, the $T_s = (96 \cdot 10^3)^{-1}$ s. Cross and dot terms of IQ samples are

$$s_\text{cross}[n] = s_\text{I}[n-1]s_\text{Q}[n] - s_\text{I}[n]s_\text{Q}[n-1] \tag{2}$$

$$s_\text{dot}[n] = s_\text{I}[n-1]s_\text{I}[n] + s_\text{Q}[n-1]s_\text{Q}[n] \tag{3}$$

The initial conditions are $s_\text{cross}[-1] = s_\text{dot}[-1] = 0$.

B. CTCSS Detector

The analog legacy operation mode of the developed transceiver requires reception and detection of the Continuous Tone-Coded Squelch System tones (CTCSS) [5], [6]. The frequencies of the CTCSS tones are in the subaudible range of 60 – 260 Hz, thus, the demodulated s_NF signal needs to be bandpass-filtered and decimated first by a factor of 12 resulting in a new $f_\text{s} = 8$ kHz. As we see in table I taken from [5], the CTCSS tones are closely spaced meaning that high frequency resolution is required. Together with a requirement for fast detection of the tones, simple methods based on computing fast Fourier transform (FFT) over an interval of N samples are not applicable. Our solution adopted the following algorithm.

TABLE I
A List of CTCSS Tones

f_tone [Hz]	f_tone [Hz]	f_tone [Hz]	f_tone [Hz]
67.0	94.8	131.8	186.2
69.3	97.4	136.5	192.8
71.9	100.0	141.3	203.5
$\vdots$	$\vdots$	$\vdots$	$\vdots$
91.5	127.3	179.9	250.3

A phase of a tone signal replica with frequency f_tone is being accumulated:

$$\theta[n] = \theta[n-1] + 2\pi\frac{f_\text{tone}}{f_\text{s}}\,, \tag{4}$$

where $n = 0, 1, ...N-1$ and N spans the integration interval. Then the mutual energy between the demodulated and decimated signal s'_NF and the f_tone replica is computed and integrated over the interval of N samples:

$$\begin{aligned} y_\text{I}[n] &= y_\text{I}[n-1] + s'_\text{nf}[n]\cos(\theta[n])\,, \\ y_\text{Q}[n] &= y_\text{Q}[n-1] + s'_\text{nf}[n]\sin(\theta[n])\,. \end{aligned} \tag{5}$$

At the end of the integration interval, energy of the integrated signal and its phase is computed:

$$E[N-1] = y_\text{I}^2[N-1] + y_\text{Q}^2[N-1]\,, \tag{6}$$

$$\varphi[N] = \text{atan2}(y_\text{I}[N], y_\text{Q}[N])\,. \tag{7}$$

To comply with range of principal values of the atan2 function, which is $(-\pi, \pi]$, the phase difference $\Delta\varphi[N]$ of the two consecutive integration intervals needs to be checked and wrapped

$$\Delta\varphi[N] = \varphi_\text{current}[N] - \varphi_\text{old}[N]\,. \tag{8}$$

The phase difference $\Delta\varphi[N]$ is also required for a detection condition preventing from the false detection due to phase error φ_err. The CTCSS tone is detected when following condition is met:

$$E[N] > \gamma_E \wedge |\Delta\varphi[N]| < \varphi_\text{err}\,. \tag{9}$$

The initial conditions at the beginning of each integration interval are set to be $\theta[-1] = y_\text{I}[-1] = y_\text{Q}[-1] = \varphi[N]_\text{old} = 0$

C. Voice Dynamic Range Compressor

When the transceiver is operating in the NFM mode, the dynamic range of the sampled ($f_\text{s} = 8$ kHz) modulating voice signal must be compressed to prevent overmodulation of the FM carrier leading to corruption of the allowed spectral mask [1]. However, at the same time, the compressed voice must remain comprehensible. The voice compressor we designed for our VHF/UHF transceiver can be perceived as a nonlinear system with an adjustable gain $g[n]$ and hard limiter. In the first step, the signal s_NF is multiplied by the $g[n]$ set at programmable initial value

$$x[n] = s_\text{NF}[n]g[n-1]\,. \tag{10}$$

Then the first fixed threshold γ_1 determines either the gain coefficient must be decreased, or increased by a factor γ_1^-, or γ_1^+:

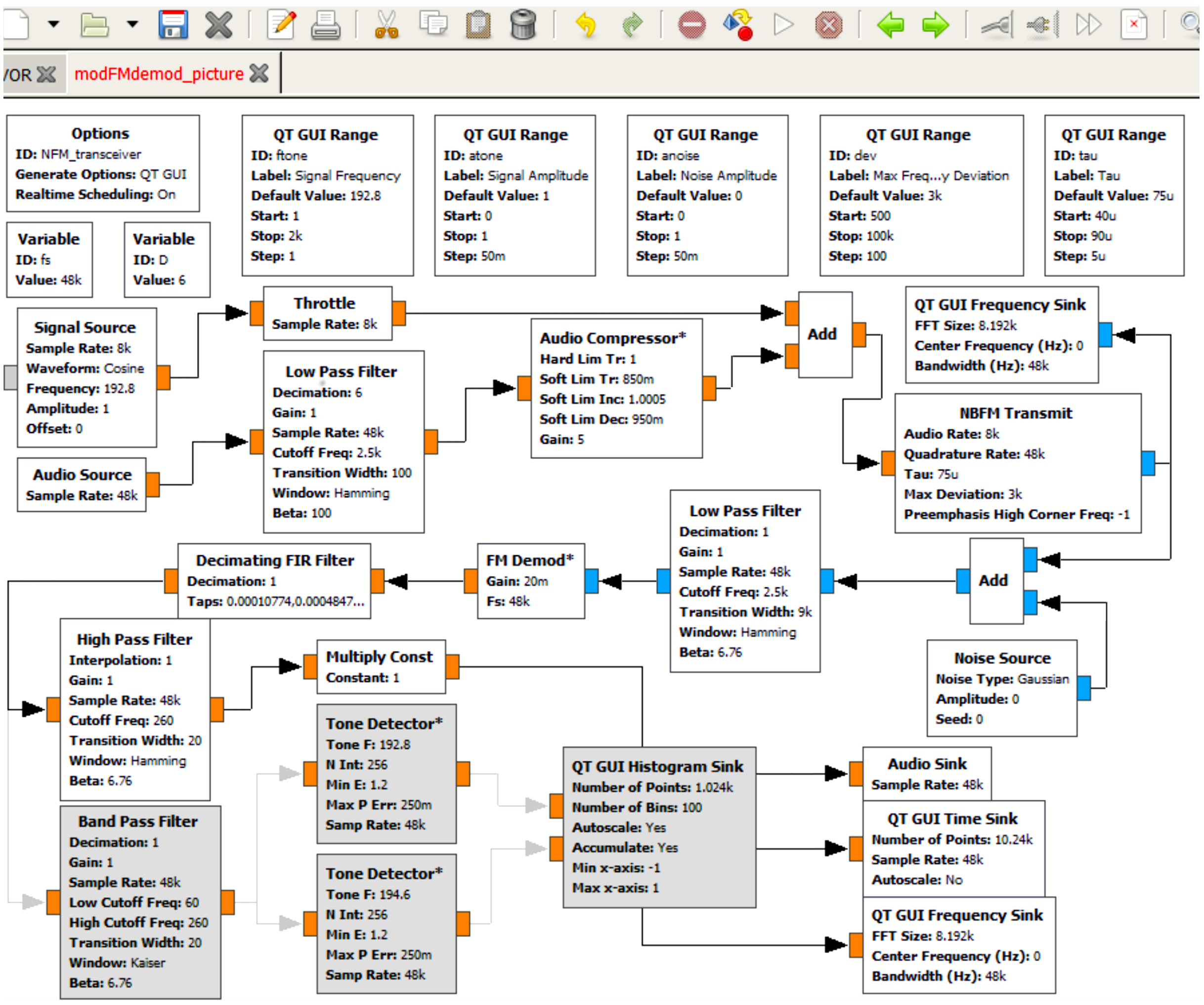
$$\Delta[n] = |x[n]| - \gamma_1\,, \tag{11}$$

$$g[n] = \begin{cases} \gamma_1^- g[n-1], & \Delta > 0 \quad (12) \\ \gamma_1^+ g[n-1] & \Delta \leq 0\,, \quad (13) \end{cases}$$

Passing the compressor part, the signal is clipped in a hard limiter following condition determined by a second threshold γ_2:

$$y[n] = \begin{cases} \gamma_2, & x[n] \geq \gamma_2 \quad (14) \\ -\gamma_2, & x[n] \leq \gamma_2\ . \quad (15) \end{cases}$$

Initial and fixed parameters of the described dynamic range compressor — $g[-1]$, γ_1, γ_2, γ_1^-



Options
ID: NFM_transceiver
Generate Options: QT GUI
Realtime Scheduling: On

Variable
ID: fs
Value: 48k

Variable
ID: D
Value: 6

QT GUI Range
ID: ftone
Label: Signal Frequency
Default Value: 192.8
Start: 1
Stop: 2k
Step: 1

QT GUI Range
ID: atone
Label: Signal Amplitude
Default Value: 1
Start: 0
Stop: 1
Step: 50m

QT GUI Range
ID: anoise
Label: Noise Amplitude
Default Value: 0
Start: 0
Stop: 1
Step: 50m

QT GUI Range
ID: dev
Label: Max Freq...y Deviation
Default Value: 3k
Start: 500
Stop: 100k
Step: 100

QT GUI Range
ID: tau
Label: Tau
Default Value: 75u
Start: 40u
Stop: 90u
Step: 5u

Signal Source
Sample Rate: 8k
Waveform: Cosine
Frequency: 192.8
Amplitude: 1
Offset: 0

Audio Source
Sample Rate: 48k

Throttle
Sample Rate: 8k

Low Pass Filter
Decimation: 6
Gain: 1
Sample Rate: 48k
Cutoff Freq: 2.5k
Transition Width: 100
Window: Hamming
Beta: 100

Audio Compressor*
Hard Lim Tr: 1
Soft Lim Tr: 850m
Soft Lim Inc: 1.0005
Soft Lim Dec: 950m
Gain: 5

Add

QT GUI Frequency Sink
FFT Size: 8.192k
Center Frequency (Hz): 0
Bandwidth (Hz): 48k

NBFM Transmit
Audio Rate: 8k
Quadrature Rate: 48k
Tau: 75u
Max Deviation: 3k
Preemphasis High Corner Freq: -1

Low Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 48k
Cutoff Freq: 2.5k
Transition Width: 9k
Window: Hamming
Beta: 6.76

Add

Noise Source
Noise Type: Gaussian
Amplitude: 0
Seed: 0

FM Demod*
Gain: 20m
Fs: 48k

Decimating FIR Filter
Decimation: 1
Taps: 0.00010774,0.0004847...

High Pass Filter
Interpolation: 1
Gain: 1
Sample Rate: 48k
Cutoff Freq: 260
Transition Width: 20
Window: Hamming
Beta: 6.76

Band Pass Filter
Decimation: 1
Gain: 1
Sample Rate: 48k
Low Cutoff Freq: 60
High Cutoff Freq: 260
Transition Width: 20
Window: Kaiser
Beta: 6.76

Multiply Const
Constant: 1

Tone Detector*
Tone F: 192.8
N Int: 256
Min E: 1.2
Max P Err: 250m
Samp Rate: 48k

Tone Detector*
Tone F: 194.6
N Int: 256
Min E: 1.2
Max P Err: 250m
Samp Rate: 48k

QT GUI Histogram Sink
Number of Points: 1.024k
Number of Bins: 100
Autoscale: Yes
Accumulate: Yes
Min x-axis: -1
Max x-axis: 1

Audio Sink
Sample Rate: 48k

QT GUI Time Sink
Number of Points: 10.24k
Sample Rate: 48k
Autoscale: No

QT GUI Frequency Sink
FFT Size: 8.192k
Center Frequency (Hz): 0
Bandwidth (Hz): 48k

Fig. 1. Block scheme – the flow graph – of the GNU Radio high-level programming environment for testing the developed algorithms *(marked with *)*.

adjusting a step of a gain decrease, γ_1^+ adjusting a step of gain increase — are programmable.

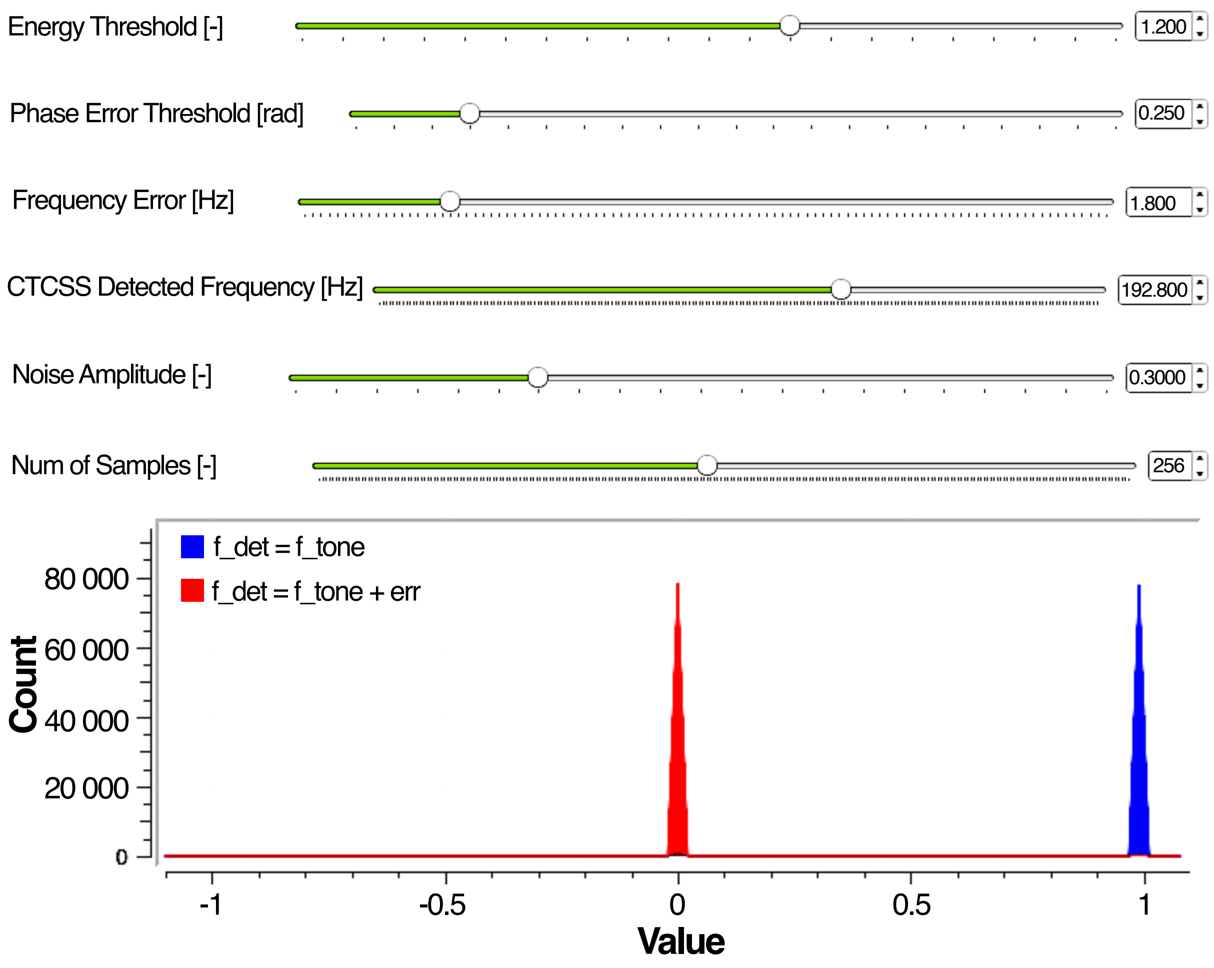
III. Testing Environment

These days, we have several options on how to examine the previously described algorithms. Besides well-established MATLAB and its graphical programming environment Simulink, LabVIEW and LabVIEW Communication System Design Suite graphical programming platforms allow us practical testing of developed DSP algorithms before their implementation. The common problems of listed software are unflattering licence prices and limited support of SDR front end hardware.

The other high-level programming environment we use in this paper to validate the developed algorithms with the help of SDR is GNU's Not Unix (GNU) Radio. Since the GNU Radio works under free GNU General Public License, it is becoming popular among the community of RF and DSP engineers who are free to contribute to its development. GNU Radio combines graphical and text-based programming where a programmer can choose from a library of predefined blocks implementing routine processing algorithms, then set their input parameters and interconnect them into a radio system as shown in the Fig. 1.

Those blocks are mainly written in Python and placed on the top of C++ ”core”. Thanks to that, the programmer can create custom blocks either in Python or C++, including support for a vast number of RF front end hardware — from the state-of-the-art Universal Software Radio Peripheral (USRP) SDRs to the handy crowd-founded ones. Those advantages allow real-time implementation helping rapid prototyping with inexpensive SDR. Moreover, similarly to previously mentioned frameworks, GNU Radio offers blocks for simulations, in which hardware is not deployed. However, comparing to other environments, the GNU Radio often miss proper documentation making it less attractive to use.

Together with GNU Radio, we use SDR receiver SDRplay RSP1A with parameters found in [7]. As a reference source of signals, we use signal generator R&S SMC100A.


Energy Threshold [-] 1.200

Phase Error Threshold [rad] 0.250

Frequency Error [Hz] 1.800

CTCSS Detected Frequency [Hz] 192.800

Noise Amplitude [-] 0.3000

Num of Samples [-] 256

Fig. 2. Graphical interface for setting and observing the parameters of the tested CTCSS detector.

IV. Algorithms Testing

In the GNU Radio environment, we developed a system depicted in the flow-graph in the Fig. 1 testing the overall functionality of the transceiver in which the designed blocks were included. This environment takes as a source of the signal either the input of the RSP1A SDR (not in the scheme) – the signal generator – with the sampling frequency identical as in designed for VHF/UHF transceiver, or an input of the audio card — the microphone — of the computer running the GNU Radio with maximum available sample frequency (48 kHz in our case) allowing us real-time switching between all developed signal processing blocks (the ones in a grey shade are disabled in the real-time computation) and to verify their proper behaviour from a system point of view. Nevertheless, first, the developed blocks had to be tested individually.

A. Testing FM Demodulator

To test only the the FM demodulator processing block proposed in the previous section, the flowgraph was reduced to the necessary minimum. The sampling frequency of the complex envelope of the signal received by the developed transceiver is 96 kHz which is the frequency at which the FM demodulator runs. Then, the demodulated signal is decimated by a factor of 12 to comply with the 8 kHz sample frequency of the audio signal. The flowgraph was hence simplified to include only RSP1A source, complex envelope low-pass filter, developed FM demodulator, decimating block, and root mean square (RMS) block measuring the average power of the received signal. Then, we input the FM modulated signal from the signal generator via coaxial cable to the RSP1A. The carrier is modulated with a 1 kHz tone with the frequency deviation of 1.5 kHz according to the conditions of measurement defined in standard [1]. Receiving the modulated signal at the carrier frequency in the GNU Radio program, we were able to estimate Signal-to-noise and distortion ratio (SINAD) parameter by turning on and off the FM modulation at the signal generator. The estimated values were 10 dB for -120 dBm signal power at the generator and 20 dB when -110 dBm.

By deploying the FM demodulator in the system from the Fig. 1, we also verified its functionality by listening a demodulated real voice signal coming from the audio card of the PC and comparing the result with the inbuilt narrowband frequency modulation (NBFM) receiver.

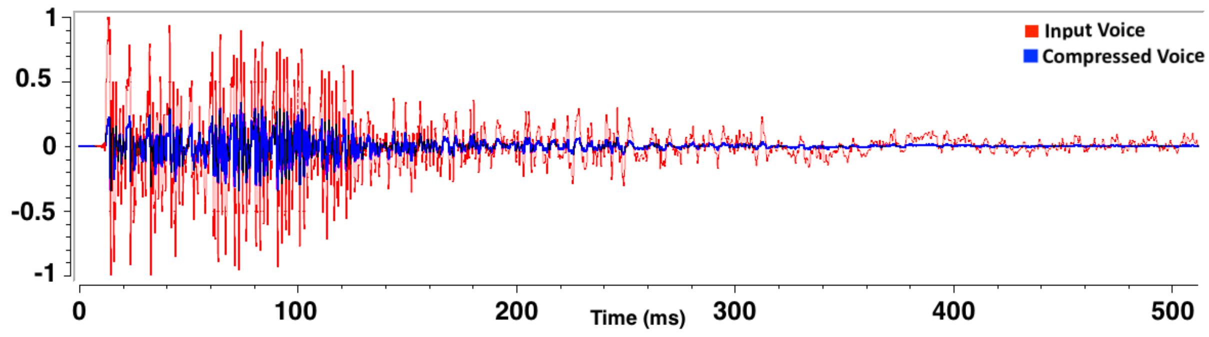


Fig. 3. Time-domain voice signal record of word ”karel” (red) and its compressed duplicate (blue).

B. *Testing CTCSS Detector*

The main issue with the CTCSS tones is their closely spaced frequencies and the requirement on fast detection. The proposed algorithm in the second section needs to be verified, i.e. we are interested in the frequency interval inside which the detector returns positive detection. This interval denotes the eventual CTCSS tone frequency drift of a transmitted signal which is received by the developed transceiver. Clearly, two neighbouring CTCSS tones' ”guarding intervals” must not overlap.

To verify such a possibility, we enabled two developed tone detectors in our GNU flow graph envisioned in Fig. 1. The first one is tuned exactly to the CTCSS tone frequency; the second one is slightly detuned by a frequency error added to the CTCSS tone frequency. Their binary outputs are then plotted in a histogram in

As envisioned in th Fig. 2, by manually increasing or decreasing the added frequency error to the signal with the tone frequency, which is received by the CTCSS detector, we determine at which frequency the tone detector starts reacting. With this flow graph, we can also investigate the effect of the length of the integration interval N since its length corresponds to the response of the proposed detector. Then we can examine the effect of added noise or refine the parameters of the phase and energy thresholds $\varphi_{\text{err}}, \gamma_{\text{E}}$.

C. *Testing Voice Dynamic Range Compressor*

The standard [1] strictly defines a spectral mask which needs to be abided by the developed transceiver. The highest restriction is for 12.5 kHz channel spacing where the frequency deviation of the FM modulated signal must be 2.5 kHz at most. Hence, the compression of the voice signal amplitude is needed to prevent unwanted emissions at frequencies where the transmission is not allowed and to avoid the unpredictable distortion caused highly dynamic voice at the transmitter side or by the input filters at the receiver side.

To test the algorithm proposed in the second section, we used the GNU Radio flowgraph in the Fig. 1 allowing us to speak directly to a microphone, plot the uncompressed time-domain curve, compress the voice, plot the compressed voice and listen to it via speakers in real-time with negligible latency. That helps us to verify the compressed voice comprehensibility and the amount of compression, eventually refine parameters of the thresholds γ_1, γ_2 and the slope of the gain coefficient γ_1^+, γ_1^-. The Fig. 3 shows an example test of the voice record of the word ”karel” sampled at $fs = 8$ kHz and highly compressed, however, remaining understandable.

V. Conclusion

In this article, we showed a complementary way to test, verify and refine proposed algorithms in GNU Radio environment. The profit coming with the implementations and simulations of radio systems in GNU Radio is the free license and ease of use despite missing documentation. Only fundamental knowledge of Python scripting is needed. Reusing the predefined blocks, the user is capable of comparison with user's custom blocks. When developed algorithms are not computationally extensive as, e.g. computation of a singular value decomposition and similar, the developer can obtain processed results with GNU radio flow graph in real-time. During the development of the VHF/UHF transceiver, this high-level programming environment proved to be a viable and valuable tool which shortened the whole process of development.

Acknowledgment

This work has been supported by grant no. FV30365 “Mobile radio with digital signal processing” funded by the Ministry of Industry and Trade of the Czech Republic.

References

[1] REN/ERM-TGDMR351, “Land mobile service; radio equipment with an internal or external rf connector intended primarily for analogue speech; harmonised standard covering the essential requirements of article 3.2 of the directive 2014/53/EU,” ETSI, Sophia Antipolis, Standard EN 300 086 V2.1.2, 2016.

[2] REN/ERM-TGDMR-361, “Land mobile service; radio equipment intended for the transmission of data (and/or speech) using constant or non-constant envelope modulation and having an antenna connector; harmonised standard covering the essential requirements of article 3.2 of the directive 2014/53/EU,” ETSI, Sophia Antipolis, Standard EN 300 113 V2.2.1, 2016.

[3] P. Kovář, P. Puričer, T. Morong, and F. Šturc, “Digital up and down converter for high performance vhf and uhf transceiver,” in *2019 International Conference on Applied Electronics (AE)*, 2019, pp. 1–4.

[4] J. T. Curran, G. Lachapelle, and C. C. Murphy, "Improving the design of frequency lock loops for gnss receivers," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 48, no. 1, pp. 850–868, 2012.

[5] X. Liu, Y. Zhao, and F. Song, "Comparative analysis of CTCSS decoding algorithms," in *2010 3rd International Congress on Image and Signal Processing*, vol. 7, 2010, pp. 3268–3271.

[6] "Continuous Tone Controlled Squelch System," http://radiohistory.uk/manuals/pye/CTCSS.pdf, Philips, Tech. Rep., accessed: 03/30/2020.

[7] "SDRPlay radio spectrum processor 1," www.sdrplay.com/wp-content/uploads/2017/01/161129RSP1DatasheetV3.pdf, accessed: 03/30/2020.