Enabling Ambient Backscatter Using Low-cost Software-defined-radio

Maximilian Stiefel
Master Programme Embdedded Systems
Uppsala University
maximilian.stiefel.8233@student.uu.se

Elmar van Rijnswou
Master Programme Embdedded Systems
Uppsala University
elmar.vanrijnswou.9818@student.uu.se

Carlos Prez-Penichet
Communication Research Group
Uppsala University
carlos.penichet@it.uu.se

Ambuj Varshney
Communication Research Group
Uppsala University
ambuj.varshney@it.uu.se

Christian Rohner
Communication Research Group
Uppsala University
christian.rohner@it.uu.se

Thiemo Voigt
Communication Research Group
Uppsala University
thiemo.voigt@it.uu.se

***Abstract*—Backscatter communication is attractive for energy-constrained devices due to its very low power requirements. Ambient backscatter takes this aspect to the limit by leveraging existing radio frequency signals for the purpose of communication without the need for generating energy expensive carrier signal. In this paper we investigate the use of ambient television broadcast signals for communication. As opposed to state-of-the-art restricted to operations under conditions of strong signal strength, we demonstrate a low cost software defined radio as receiver enables operations even in conditions when ambient signals are weak in strength. We build the system using low-cost off-the-shelf microcontroller, and RTLSDR software-defined radio receiver. We also conduct survey of signal strength of TV broadcast in a mid-sized swedish city, and observe that our system can operate in most parts of the city.**

I. Introduction

April 7, 2017

II. Background

A. RTL2832U

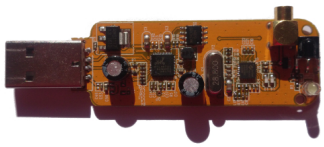

Fig. 1. This figure is a photo of the RTLSDR used in this project.

The Raeltek RTL2832U is a terrestrial digital video broadcast (DVB-T) chip, which is the basis of a very low-cost software defined radio: The RTL-SDR. One feature of this chip is, that it supports USB 2.0. Ham radio enthusiasts combined their efforts to enable accessing the raw I/Q data. With their software driver (basically the *librtlsdr*, cf. [1]) DVB-T sticks, which are based on the RTL2832, can be converted into a wideband SDR. This sort of hardware has been in the range of a few thousands USD a few years ago. An RTL2832U based DVB-T receiver can be bought with antenna for less than 10 USD. What is a software defined radio? Traditionally basic radio components like mixers or filters have been realized in hardware. Nowadays as soon as one works in the baseband frequency the coputational resources of a personal computer are sufficient to realize these components with signal processing done in software. An RTL-SDR does not only consist of the RTL2832U. The second essential component is the tuner, which mixes down the radio frequency (RF) to a intermediate frequency (IF). In the case of the used hardware this tuner component was represented by a *Rafael Micro R820T/2*. It subsumes a low-noise amplifier (LNA) and filters. The image frequency is rejected automatically with 65 dBc and the noise figure is with 3.5 dB quite low. It alows a frequency range from 42 to 1002 MHz (cf. [2]). The I/Q demodulator, also shown in figure 2, is the most important part of the *RTL2832U* for our work. The received signal can be intepreted as

$$s_{IF}(t) = I(t) \cdot cos(\omega_0 t) + Q(t) \cdot sin(\omega_0 t) \tag{1}$$

This is multiplied with a cosine (and a sine as well) from a local oscillator.

$$s_{IF}(t) \cdot s_L(t) = I(t) \cdot cos(\omega_0 t) \cdot cos(\omega_0 t) + Q(t) \cdot sin(\omega_0 t) \cdot cos(\omega_0 t) \tag{2}$$

With $2\,cos(a)cos(b) = cos(a-b) + cos(a+b)$ and $2\,sin(a)cos(b) = sin(a+b) + sin(a-b)$ follows

$$2 \cdot s_{IF}(t) \cdot s_L(t) = I(t) \cdot \big[1 + cos(2\omega_0 t)\big] + Q(t) \cdot \big[sin(2\omega_0 t) + sin(0)\big]. \tag{3}$$

One can see, that the interesting inphase part in this case is represented by a DC value after mixing. With a low-pass filter this DC value can be seperated from the undesired rest. Similar things are hapening with the quadrature part when mixing with a sine, which is indeed done to get the quadrature part.

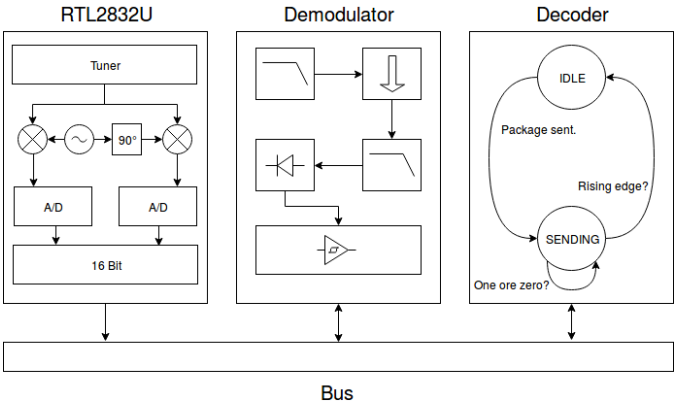

Fig. 2. This figure shows the receiver architecture from an abstract point of view.

III. Design

A. Sender

B. Receiver

The receiver was the most work intensive part of the design. C++ has been used to implement the receiver. We provide the code publicly available under https://github.com/s3xm3x/backscatterBASKReceiver. In figure 2 one can see the architecture of the receiver from an abstarct point of view. A highly sophisticated bus system has been developed to exchange data between the different components of the system. The signal flow is from the left to the right. *Librtlsdr* (cf. [1]), which is based on *libusb*, is used to transfer the data from the TV stick into our program. As described under II-A, the RTL2832 mixes down the high frequency to a intermediate frequency (both values are in SW adjustable). Subsequently the data is mixed with sine and cosine, low-pass filtered and finally transfered in the digital world. This data is available as two 8 Bit values. It is a well-known fact, that this technique, which is commonly known as I/Q demodulation, results in a real and an imagenery part. With this two values phase and absolute value of the data can be determined for every sample. So every sample consists out of a real and and imaginary part. And can be seen as:

$$I + j\,Q = abs(I, Q) \cdot e^{ang(I,Q)} \tag{4}$$

So the first block, which is entitled with RTL2832U, provides the interface to the TV dongle. This block is represented as one object. It can set the frequency f_{tuned}, where the receiver is listening. Also it can set the sampling frequency f_{samp} and the analog gain of the tuner. The raw samples are pushed to the bus. Demodulator is the block called, which is responsible to convert the sampled signals into something, which looks like ones and zeroes. Thereforre the real and imgainary values first have to be deinterleaved out of the sample message. Now they are represented as integer values between 0 and 255 (8 bit). So one converts them to floats, whereas 127.5 equals 0. After that the values have to be downsampled. Usually we were sampling with 250 kHz. This is still much more than we need assuming an additive white gausian noise (AWGN) channel with a $S/N = 10\,dB$ (cf. equation 5). So we decided to reduce the sampling frequency to 25 kHz.

$$C = B \cdot log_2(1 + \frac{S}{N}) = 25\,kHz \cdot log_2(11) \approx 86.5\,kbit/s \tag{5}$$

To do this it is important to use an anti-aliasing filter, as the Shannon-Nyquist theorem has to be satisfied:

$$f_{samp} \geq 2 \cdot f_a \tag{6}$$

f_a is the highest frequency in the signal. A FIR filter naivly implemented with floats has been used to achieve this. 10 kHz is the cut-off frequency of the first filter. All filters have been constructed with the internal filter design tool of GNU radio. After downsampling another filter is used to suppress noise. This filter has a cut-off frequency of 3 kHz. The last demodulation step is to take the signal, rectify it (cf. equation 7) and decide with a software-defined Schmitt trigger whether a sample is a 1 or a 0.

$$abs(I, Q) = \sqrt{I^2 + Q^2} \tag{7}$$

These samples are broadcasted on the bus. A registered listener receives the processed samples. This registered listener is the decoder, which is represented as an object as well. After the decoder received the samples it decides when a frame is sent or when the channel is idle. On the basis of a defined threshold the decoder determines wether a certain sample pattern of ones and zeros is a 1 bit or a 0 bit. The baudrate and the sampling rate lead to a certain length of one bit. For example a sample rate $f_{sampling}$ of 25 kHz combinded with a baudrate of 1 kbit/s leads to 25 samples/bit. The decoder also includes a function to correlate the received bitstream with an expected pattern. Hence it is able to determine the bit ro error ratio (BER). Furthermore the code we have written so far subsumes a simulator to simulate ideal data coming out of the demodulator to test the algorithms behind the decoder. Also a simulator for the RTL2832U has been written to replay recorded samples, which facialiates debugging as well.

IV. Evaluation

A. Signal Strength Variation within the City

B. Signal Strength Variation over Time

C. Communication Performance

V. Discussion

Acknowledgment

The authors would like to thank...

References

[1] Steve Markgraf et. al., *librtlsdr*, https://github.com/steve-m/librtlsdr. Github: 2012-2017.

[2] *High Performance Low Power Advanced Digital TV Silicon Tuner*. R820T. Rev 1.2. Rafael Micro. November 2011.