ET4394 Wireless Networking - Decoding RFD900 packets

Bart Rijnders - 4103505
Bernard Bekker - 4221656

***Abstract*—This report describes a method to decode packets from the RFD900 900MHz Ultra Long Range Radio Modem using MATLAB. A script was written that is able to demodulate and decode a signal from the modem that was captured using an SDR. The script outputs an array of bytes that represent a packet sent by the modem.**

I. Introduction

The goal of this project is to create a real time decoder of telemetry packets sent by a FS Team Delft formula student car using an SDR. The car uses an RFD900 transmitter to send real time telemetry data to a base station. The RFD900 is an Ultra Long Range Radio Modem that operates on a frequency of 900MHz with a datarate up to 250kbit/s. It uses GFSK modulation, data whitening before transmitting and FHSS to switch carrier frequency after a packet has been transmitted. These options make decoding the signal a challenging task. It also offers options such as AES encryption, Golay encoding and Manchester encoding which have been turned off for this project. The firmware of the RFD900 is open source and can be found on GitHub [3]. All radio functions are handled in hardware, the used transceiver is the Silabs Si1020 [5].

II. Methodology

A. Data Capture

Data was captured by using a airspy SDR near the RFD900 modem transmitting random data. The capture was made at 3Mhz to capture the signal at all frequencies in the device's frequency hopping range. To make testing the decoder easier, the signal was post processed to include only two data packets. A graph of the resulting signal can be seen in figure 1. It can be seen that during the preamble the amplitude slowly increases to it's peak values.

B. Exploring the signal

To explore the signal, we used a gnuradio script to try some demodulation techniques and get an overview of what the signal looks like. For this, we used the quadrature demodulation block, and multiplied it with the signal amplitude to get a signal when the device is transmitting, and a low signal when only noise is received. To remove the effects of signal hopping, the preamable signal was used. The minimum and maximum frequency components in the signal in a window are calculated, and the signal shifted such that it is centered around the 0 frequency. This stabilizes before the preamble ends.

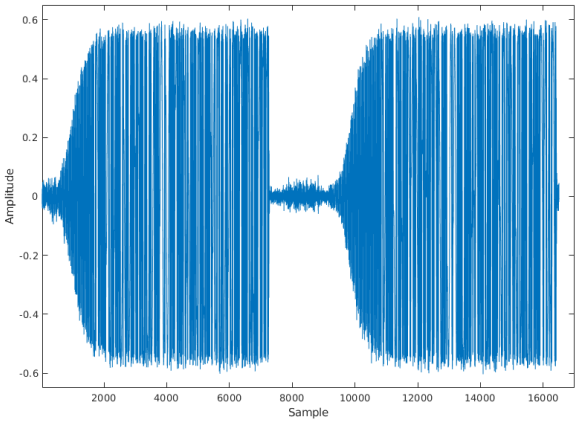

Figure 1. Signal of two subsequently received packets

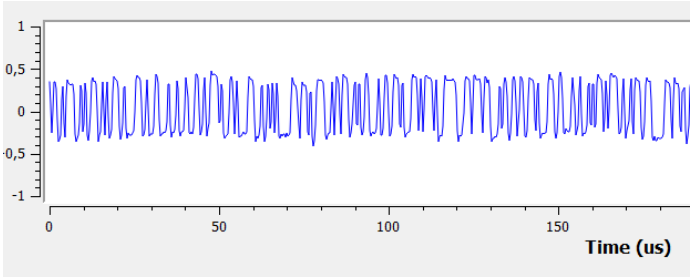

Figure 2. GNU radio FSK

This resulted in a clear Frequency modulated signal. However, clock recovery and converting this to a digital format proved very hard. While GNU radio has blocks that are made to decode this type of signal (figure 2), the documentation is non existing and examples are hard to come by. Also, we expected it to become quite hard to perform the final decoding steps after the bits are recovered in GNUradio. After spending a week on this problem, we decided to move to the more familiar area of MATLAB.

C. Matlab

In MATLAB the script consists of multiple separate functions, see the full script for details. First the data file is read and the IQ values are represented as a vector with imaginary numbers. Then a threshold is used to determine where a packet starts and ends, to filter out noise. This is done by creating a mask of logical values indicating whether a data signal is present at a given index of the vector. It can be used to isolate

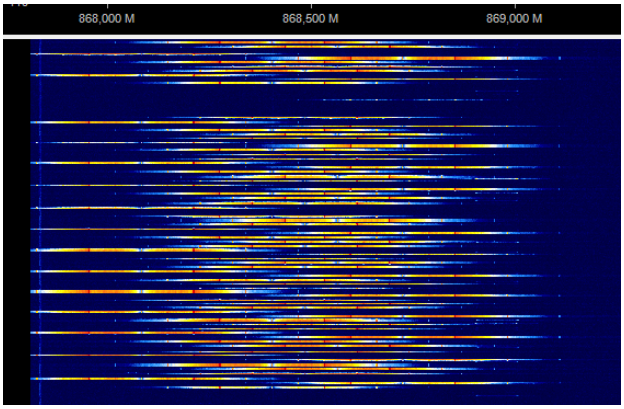

Figure 3. The frequency hopping signal in SDR#

the two different packets that have been transmitted. This mask is then applied to the original signal to obtain the two different packets as vectors.

These two isolated signals are then decoded by a separate function. Firstly the signal is tuned have the signal centered around the zero frequency.

D. Frequency hopping

Based on the SiK firmware, we were able to reverse engineer the frequency hopping algorithm used. The transmitter changes its frequency on every transmit window, typically after one or two packets. There are 10 equally spaced frequencies used from 868 to 869 Mhz (fig 3). The sequence of frequencies is shuffled on boot based on a pseudorandom generator, seeded with a known value derived from the channel ID and encryption keys. When connecting both sides transmit any data they may have in their buffers, and try to find the other transmitter by cycling trough all frequencies when listening. Because both transceivers have an identically ordered set of frequencies, once they match their frequencies, they can step trough the list in lockstep.

In a real time decoder, we can implement the same algorithm to efficiently predict the hopping sequence. However, we first made use of the wide frequency range and large amount of memory available in our SDR setup to ”bruteforce” the frequency-hopping. Because of the alternating preamble, and whitened data field, the middle frequency is close to the average frequency of a packet. Our matlab decoder determines the average frequency offset Δf using the matlab function `meanfreq()`. The packet is then shifted to this center frequency by mixing it with a $e^{-2i\pi\Delta ft}$

E. Demodulating

To demodulate the data, multiple options were considered. The first method used is the quadrature demodulation where the phase between the I and Q signals is compared. A second method we found in a ”tech-note” from 1980 was to have two narrow bandpass filters, and make a decision based on the two signal strength levels [1]. This method is more robust against in band interference. Finally, we went with method which was easy to implement in Matlab using the capabilities of matlab based on the LoRa lecture. We create a spectrogram with only two bins from $-\pi$ to π, and select the largest bin for each sample.

F. Clock recovery

Clock recovery is required to sample the signal at the correct frequency and phase to recover the binary data. Traditional methods of clock recovery use a feedback loop to lock on to the signals frequency and phase in real time. However, these methods don’t work that well on our signal, which does not enforce switching at a minimum frequency such as when using Manchester encoding. After spending a lot of time reading and trying out these methods in GNUradio, we opted to use a simpler method that made use of the possibility of having a whole packet in memory.

Two methods were considered, the first is to sample the data at a fixed frequency, and update the phase at each zero crossing. This limits the drift due to a frequency mismatch, while being simple and fast without any parameters to tune. The total amount of periods between two crossings is counted, and the bits are decoded based on the signal frequency between the crossings. A second method was presented at a GNU radio conference which is available on youtube. The author used a FFT to find the frequency and phase of the signal, both of which are encoded in the FFT’s data. Afterwards the bits can be sampled at the described frequency and phase [2]. We implemented the first option.

G. Interpretation

This results in a stream of bits which can be parsed by first finding the preamble. The preamble mask used is an alternating 24 bit array of ones and zeros starting with a zero. It will occur only once per packet, so it is used to determine the start of the data. After the preamble a sync word is used to delimit the start of the packet. The Si1020 can be configured between 1 and 4 bytes of sync word, but we determined this to be two bytes, using the chips standard word. After these bytes, two header bytes should follow with the network ID value. However this data seemed to be scrambled.

H. Dewhitening

Based on the SiK firmware, we expected the payload data to be visible in cleartext. However, it turned out that the data whitening configuration register on the Si1020 is set to true on reset, and was not configured in the firmware. The datasheet or firmware did not provide any information on the data whitening method. Based on internet search results, the most common datawhitening method seemed to be the IBM method, where the output from a linear feedback shift register is xor’ed with the payload data [4]. We determined the initial state and polynomial by sending a data packet with only zeros, as the first bits will show the initial state of the LFSR. This matched with the IBM polynomial and the initial state ’0 1 1 1 1 0 0 0 0’.

I. Data

After dewhitening, we can read the two header bytes, a packet length byte, and convert all data bytes. The resulting data conforms with the expected result, and can be interpreted by an existing application. Packets start with a 129, delining the start of a data packet in the transmission control protocol. They end with a 130 followed by a 1 byte integrity check of the transmission protocol, and 2 bytes CRC of the transmitter's protocol.

J. CRC

Finally a CRC check of the packet can be done. The used polynomial is the CRC-16-IBM variant which is also known as just CRC-16. It has a generator polynomial of `0x8005` and an uses an initial value of `0xFFFF`. However our current implementation does not include a CRC check yet. A CRC script was ported from the firmware but has so far not been able to correctly check a CRC frame of a packet. See the script for more details.

III. Future improvements

The final goal would be to do real time decoding of packets, and sending them to an application that can interpret the data. For this, a few additional steps are necessary that we were not able to perform in the available time-frame. A preprocessing step that collects samples with a magnitude above a threshold and sends them as a packet to the matlab code would be sufficient as the code already runs faster than realtime. We attempted to implement this using simulink blocks and in GNURadio using tags and PDU's, but got stuck on annoying problems.

References

[1] *fsk: signals and demodulation*. URL: http://edge.rit.edu/edge/P09141/public/FSK.pdf.

[2] *GNU radio clock recovery (Youtube)*. URL: https://www.youtube.com/watch?v=rQkBDMeODHc.

[3] Luke Hovo. 2014. URL: https://github.com/RFDesign/SiK/releases.

[4] *IBM data whitening method*. URL: https://www.silabs.com/documents/public/application-notes/AN592.pdf.

[5] *Si10x0 datasheet*. URL: https://www.silabs.com/documents/public/data-sheets/Si102x-3x.pdf.