TECHNISCHE
UNIVERSITÄT
DARMSTADT

ANSIAN - ANDROID SIGNAL ANALYZER

DENNIS MANTZ AND MAX ENGELHARDT

SEEMOO Secure Networking Lab

April 28, 2016

Secure Mobile Networking Lab
Department of Computer Science

AnSiAn - Android Signal Analyzer
SEEMOO Secure Networking Lab

Submitted by Dennis Mantz and Max Engelhardt
Date of submission: April 28, 2016

Advisor: Prof. Dr.-Ing. Matthias Hollick
Supervisor: Jiska Classen

Technische Universität Darmstadt
Department of Computer Science
Secure Mobile Networking Lab

CONTENTS

ACRONYMS

AM	Amplitude Modulation
AnSiAn	Android Signal Analyzer
BPSK	Binary Phase Shift Keying
FM	Frequency Modulation
GC	Garbage Collector
GUI	Graphical User Interface
LSB	Lower Side Band
MVC	Model–View–Controller
PI	Program Identifier
PLL	Phase Locked Loop
PSK31	Phase Shift Keying, 31 Baud
RDS	Radio Data System
SDR	Software-Defined Radio
USB	Upper Side Band

1 INTRODUCTION

Android Signal Analyzer (AnSiAn) is an Android application by the Secure Mobile Network Lab (SEEMOO) at Technische Universität Darmstadt. It features a graphical signal analyzer that can be used with common Software-Defined Radios (SDRs) like the HackRF and the RTL-SDR. The project is based on RF Analyzer, an application by Dennis Mantz. AnSiAn currently extends RF Analyzer by the following features:

- Time Domain Signal Graph (Waveform)
- Morse Decoder
- Scanner
- Codebase structured according to the Model–View–Controller (MVC) pattern

This lab aims to further extend the feature set of AnSiAn while also making the app more stable and refining existing features. The description of the project goals are listed in Section 1.1.

1.1 PROJECT DEFINITION

This section defines the features that will be implemented throughout the project and schedules them into three sprints.

1.1.1 *Features*

The new features can be divided into mandatory features, that will have high priority within this project, and optional features, that will be implemented if time permits. As can be seen in Section 1.1.2, the third sprint is reserved for either optional features or completing mandatory features and the documentation.

1.1.1.1 *Mandatory Features*

The following features are scheduled for implementation during the first and second sprint:

- Radio Data System (RDS) demodulation
  If the user selects the existing wide-band Frequency Modulation (FM) demodulation option, the app shall try to detect and

demodulate any existing RDS signal along with the audio demodulation. The extracted information shall be displayed on the screen.

- Phase Shift Keying, 31 Baud (PSK31) demodulation
  If the user selects either of the single side band demodulation modes (Upper Side Band (USB) and Lower Side Band (LSB)), he or she shall have the option to enable PSK31 demodulation along with or instead of the audio demodulation. The demodulated text string shall appear and scroll through the analyzer window.

- Extraction of RDS-, Morse- and PSK31-data to logfiles
  If the user selects to demodulate any digital mode, the demodulated text shall be written to a log file specified by the user.

- Support for the rad1o badge
  The rad1o badge, which is a modified low-cost replica of the HackRF, shall be supported as a signal source by AnSiAn.

- Transmission support for HackRF and rad1o
  If AnSiAn is used with an SDR capable of transmitting signals, it shall offer options to send signals in the following ways:
  - Replay I/O samples from a file
  - Generate and send Morse code from text
  - FM-modulate and send audio from a file

1.1.1.2 *Optional Features*

The optional features are scheduled in the third and last sprint. However, they will only be added to the feature set if the last sprint is not needed in order to compensate for delays on the mandatory features. The optional features are listed in the order of priority:

- Walkie-Talkie Mode
  The user shall have the possibility to put AnSiAn into a Walkie-Talkie mode. In this mode, the application will demodulate an FM channel and the user can quickly switch between demodulation and transmission of audio recorded from the internal microphone.

- Packet Radio demodulation
  A new mode *Packet Radio* shall be added to AnSiAn. Once selected, it shall allow the user to tune to a Packet Radio channel and display information about demodulated packets on the screen. If time permits, it might even be possible to implement a transmission feature for Packet Radio.

1.1.2 *Time Schedule*

The project will have two developers, Dennis Mantz and Max Engelhardt, working in three sprints. There are three milestones corresponding to the sprints, labeled Alpha, Beta and Final Version. They each add an independent and self-contained set of features to the application:

- Software Design (due 12.05.)
- Sprint 1: Alpha Version (due 09.06.)
  - RDS demodulation
  - PSK31 demodulation
  - Extraction of RDS-, Morse- and PSK31-data to logfiles
- Sprint 2: Beta Version (due 21.07.)
  - Support for the rad1o badge
  - Transmission support for HackRF and rad1o
    - Replay I/O samples from a file
    - Generate and send Morse code from text
    - FM-modulate and send audio from a file
- Sprint 3: Final Version (due 25.08.)
  - Complete leftovers from previous sprints
  - Walkie-Talkie Mode (optional)
  - Packet Radio demodulation (optional)

1.2 CLEANUP TASKS

Outside the scope of the planned sprints, some cleanup tasks were necessary to prepare the implementation of the features listed in Section 1.1.

1.2.1 *Memory Optimizations*

The original RF Analyzer application's architecture is based on blocking queues that synchronize the various signal processing threads and efficiently manage memory buffers. Unfortunately, this architecture was partly dropped by the developers of AnSiAn when changing to a new architecture based on the EventBus library. As a result, memory allocation management does not work as efficiently with the current version of AnSiAn.

Instead of using cycling buffers for inter-thread-communication, AnSiAn uses EventBus to deliver data. Buffers are always allocated

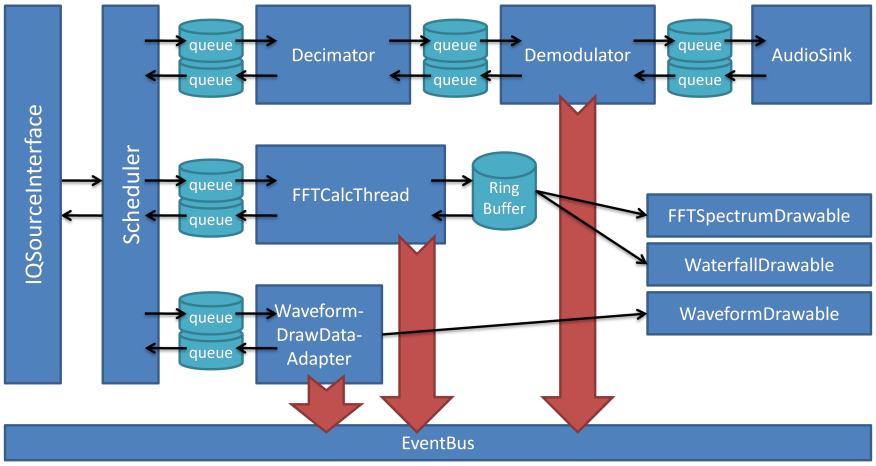


Figure 1: Signal processing architecture with blocking queues

freshly and discarded after use. This results in a high activity of the Garbage Collector (GC) and therefore in a bad overall performance of the app.

Listing 1 shows a logcat output of the app before any optimizations were applied. The GC runs approximately 8 times per second and the slow performance results in stuttering audio demodulation on older hardware.

Listing 1: Logcat output before memory optimizations

```
05-12 17:55:04.060 D/dalvikvm: GC_FOR_ALLOC freed 4347K, 14% free
     54695K/62984K, paused 28ms, total 28ms
05-12 17:55:04.180 D/dalvikvm: GC_FOR_ALLOC freed 4321K, 14% free
     54737K/62984K, paused 26ms, total 26ms
05-12 17:55:04.300 D/dalvikvm: GC_FOR_ALLOC freed 4507K, 14% free
     54705K/62984K, paused 32ms, total 32ms
05-12 17:55:04.420 D/dalvikvm: GC_FOR_ALLOC freed 4454K, 14% free
     54759K/62984K, paused 30ms, total 30ms
```

In order to fix this performance issue, the architecture is reverted to using blocking queues and cycling buffers in places where large memory buffers are passed between threads. EventBus is still used for delivering information which is not tied to large buffers. A schema of the new architecture is depicted in Figure 1.

In this architecture, the buffers cycle between the threads. The re-usage of buffers helps to reduce the memory allocation and garbage collection overhead to a minimum. Listing 2 shows the logcat output

after the architecture changes have been applied. The GC only needs to run every 10 to 20 seconds.

Listing 2: Logcat output after memory optimizations

```
05-12 17:27:29.230 D/dalvikvm: GC_FOR_ALLOC freed 3233K, 15% free
     19706K/23000K, paused 32ms, total 33ms
05-12 17:27:40.780 D/dalvikvm: GC_FOR_ALLOC freed 3528K, 16% free
     20235K/23824K, paused 30ms, total 31ms
05-12 17:28:00.110 D/dalvikvm: GC_FOR_ALLOC freed 4130K, 18% free
     20338K/24528K, paused 36ms, total 37ms
05-12 17:28:24.520 D/dalvikvm: GC_FOR_ALLOC freed 4263K, 18% free
     20341K/24664K, paused 49ms, total 49ms
```

2 DESIGN

The software design for each sprint is done ahead of the respective sprint. This procedure goes along well with the Agile Manifesto which encourages the design of a complex system in small incremental parts.

2.0.1 *Sprint 1: PSK31 and RDS demodulation*

The existing architecture of AnSiAn features individual threads for scheduling, downsampling, demodulation and audio output. The `Demodulator` thread demodulates quadrature samples by calling the `demodulate()` method on an instance of `Demodulation`. `Demodulation` is an abstract class that is implemented by concrete demodulation methods such as `AM`, `FM` and `Morse`.

AnSiAn utilizes the EventBus library in order to pass demodulated Morse text to the Graphical User Interface (GUI). Demodulated audio data is passed to the `AudioSink` thread by enqueuing it into its input queue. This mechanism is explained in more detail in Section 1.2.1.

In order to extend AnSiAn with demodulation functionality for PSK31 and RDS, the existing architecture needs to be extended. The extended architecture is depicted in Figure 2 and explained in the following.

Two new classes `PSK31` and `RDS`, that inherit from `Demodulation`, need to be implemented to represent the new demodulation mechanisms.

As PSK31 demodulation works on the envelope of the received signal and Amplitude Modulation (AM) demodulation essentially performs envelope detection, `PSK31` uses an instance of `AM` for envelope detection.

RDS transmits metadata for FM radio channels. It is therefore desirable for the RDS demodulation mode to not only display this metadata, but to also play the FM-modulated audio at the same time. The `RDS` class uses an instance of `FM` for this purpose.

Like the existing architecture, the new architecture will use the EventBus library to pass the demodulated text to the GUI. The existing View `MorseReceiveView` is refactored into a universal `DemodulationInfoView` that displays the text output of any selected demodulator. Demodulators pass `DemodTextEvents` and `DemodInfoEvents` via the EventBus to the `DemodulationInfoView`, which contain demodulated text and further information (e.g. baud rates or raw dits and dahs) and are displayed in separate lines.

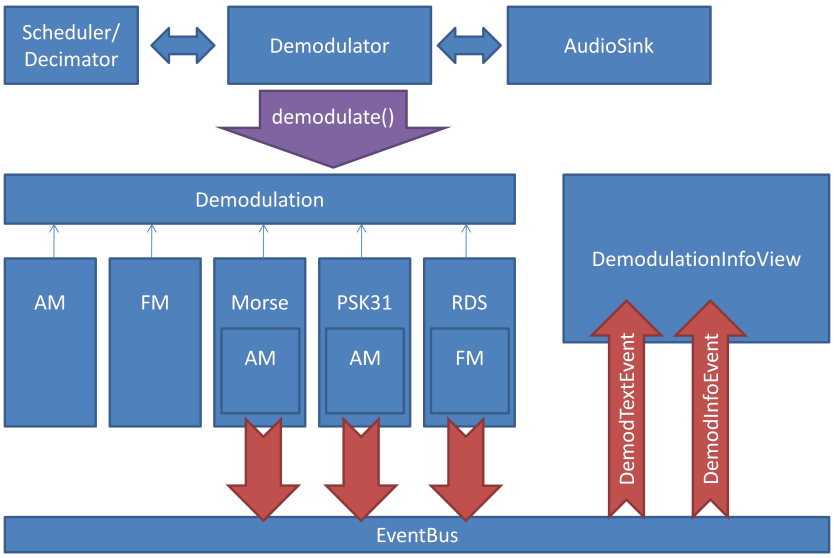


Figure 2: Architecture of the extended demodulation logic and communication with the GUI

3 IMPLEMENTATION

3.1 RADIO DATA SYSTEM

The RDS signal is transmitted along with wide band FM radio signals to provide additional information about the radio station and program.

Demodulation of the RDS signal is first done in Octave in order to evaluate the demodulation algorithm. The octave implementation also helps by providing reference data of the different stages of demodulation.

3.1.1 *RDS modulation scheme*

RDS uses Binary Phase Shift Keying (BPSK) with Manchester encoding. The signal is transmitted with an offset of 57 kHz relative to the center frequency of the mono audio signal (baseband). The 19 kHz pilot tone of wideband FM can therefore be used to retrieve the RDS carrier by multiplying it with itself 3 times. The complete FM spectrum can be seen in Figure 6a.

After the RDS baseband signal has been retrieved from the FM signal there are multiple ways of demodulating the BPSK modulation. A sophisticated approach tries to recover the phase synchronised RDS carrier from the signal by using e.g. a form of Phase Locked Loop (PLL) or Costas Loop. The symbols can then be extracted by multiplying the carrier with the modulated signal and apply a threshold operation to get bits.

A much simpler approach is to analyze the envelope of the signal and dectect bits based on known shapes of ones and zeros in the waveform. Figure 3 shows the envelope and the pattern of symbols which can be detected.

3.1.2 *RDS coding scheme*

RDS frames are called groups and each group consists of 4 blocks called A, B, C and D. One block has a length of 16 bit plus a 10 bit checkword. Block A always contains the Program Identifier (PI) which identifies the radio station. The content of the other blocks depends on the group type which is located in block B (see Figure 4).

Table 2 lists all group types and their descriptions. The RDS demodulation in AnSiAn only decodes types 0 and 2 because they contain

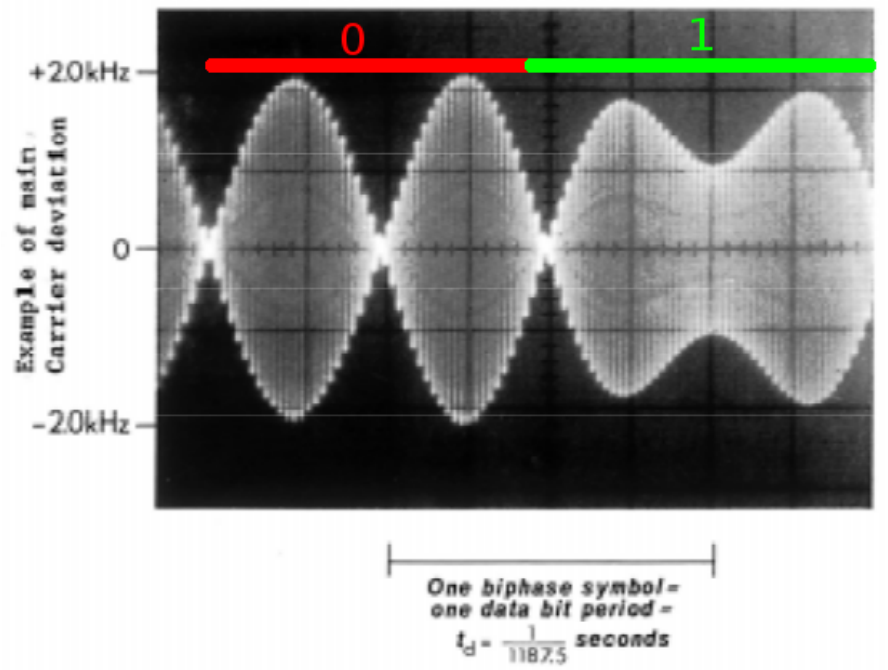


Figure 3: RDS envelope waveform after Frequency demodulation. [1]

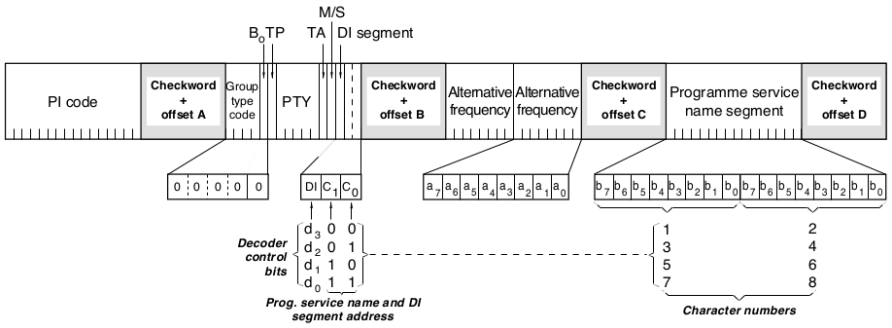


Figure 4: Coding scheme of RDS: group 0A. [1]

Group Type	Group Version	Description
0	A	Basic tuning and switching information only
0	B	Basic tuning and switching information only
1	A	Programme Item Number and slow labelling codes
1	B	Programme Item Number
2	A	RadioText only
2	B	RadioText only
3	A	Applications Identification for ODA only
3	B	Open Data Applications
4	A	Clock-time and date only
4	B	Open Data Applications
5	A	Transparent Data Channels
5	B	Transparent Data Channels
6	A	In House applications or ODA
6	B	In House applications or ODA
7	A	Radio Paging or ODA
7	B	Open Data Applications
8	A	Traffic Message Channel or ODA
8	B	Open Data Applications
9	A	Emergency Warning System or ODA
9	B	Open Data Applications
10	A	Programme Type Name
10	B	Open Data Applications
11	A	Open Data Applications
11	B	Open Data Applications
12	A	Open Data Applications
12	B	Open Data Applications
13	A	Enhanced Radio Paging or ODA
13	B	Open Data Applications
14	A	Enhanced Other Networks information only
14	B	Enhanced Other Networks information only
15	A	Defined in RBDS [15] only
15	B	Fast switching information only

Table 2: RDS Group Types

the basic information which is also often displayed on the radio receiver.

3.1.3 *Evaluation in Octave*

Developing a signal processing application on Android has many drawbacks. One issue is that it is very hard to debug the actuall signal processing components because of the lack of proper tools to visualize and analyze the data that is being processed. It is also not possible to do rapid prototyping without sufficient signal processing libaries available. Therefore the RDS demodulator was first developed in Octave and afterwards ported to Android.

For development and testing it is better to work on recorded samples instead of live captures. This makes tests reproducable and simplify the development environment. The file was recorded using the record feature of RF Analyzer. It can be imported to Octave by using the *read_cuchar_binary()* script provided by the GNU Radio project. After each step the produced output data can be written back to an *IQ* in order to use it in the Android application. This way it is possible to develop each component of the demodulation process separately and the output can be visualized on the developing machine.

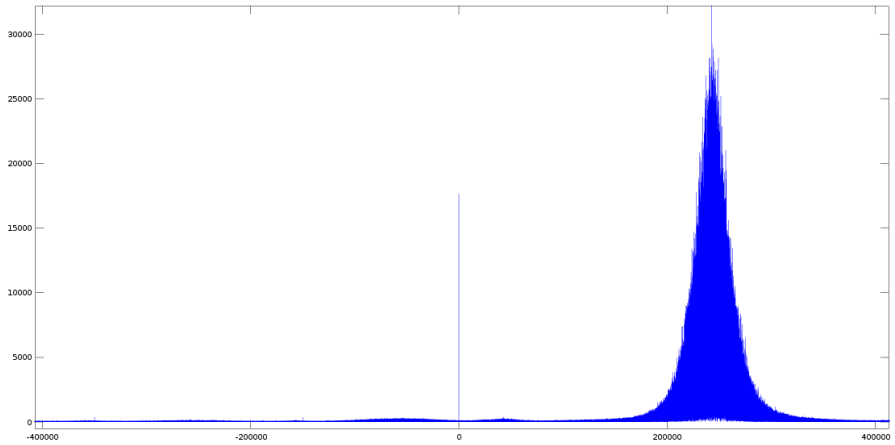
The demodulation is done in the following steps:

1. Downmixing the radio signal to baseband and filter it (see Figure 5).
2. FM demodulation (see Figure 6a).
3. Downmixing the RDS signal to baseband and filter it (see Figure 6 b and c).
4. Take the absolute value of the signal to get the envelope that was shown above (see Figure 7).
5. Find the beginning of a symbol by searching for a minimum in the waveform. From there find the end of the symbol with the same strategy. Now determine whether the symbol is a one or a zero according to the value of the minimum found in the middle of the sample compared to its peaks.

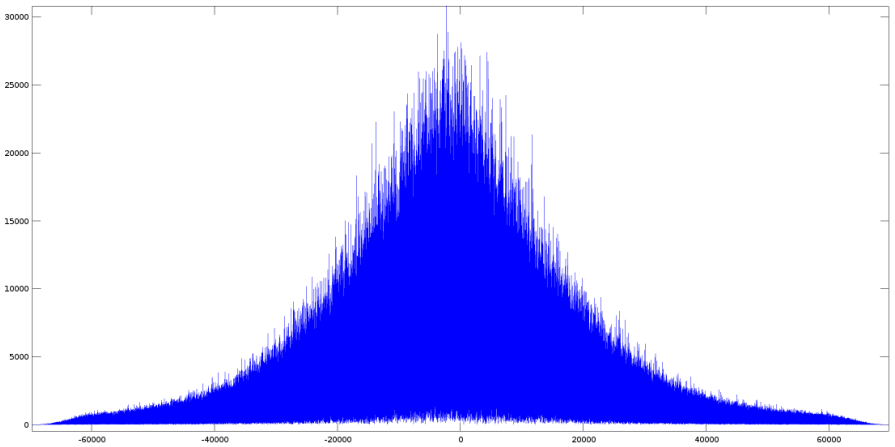
The octave code used to execute the steps mentioned above is shown in the listing below:

Listing 3: Octave implementation of the RDS demodulator

```
signal = read_cuchar_binary ("~/Downloads/2016-06-01-20-17-18
    _rtlsdr_100550000Hz_1000000Sps.iq" );
t = linspace(0, length(signal)/1000000, length(signal))';
carrier = e.^(2*pi*-245000*t*i);
down = carrier .* signal;
```

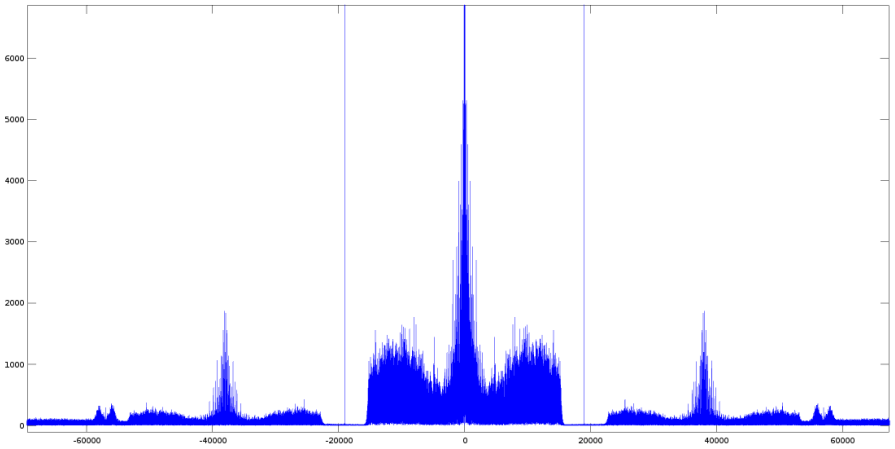


(a) Spectrum of the captured signal

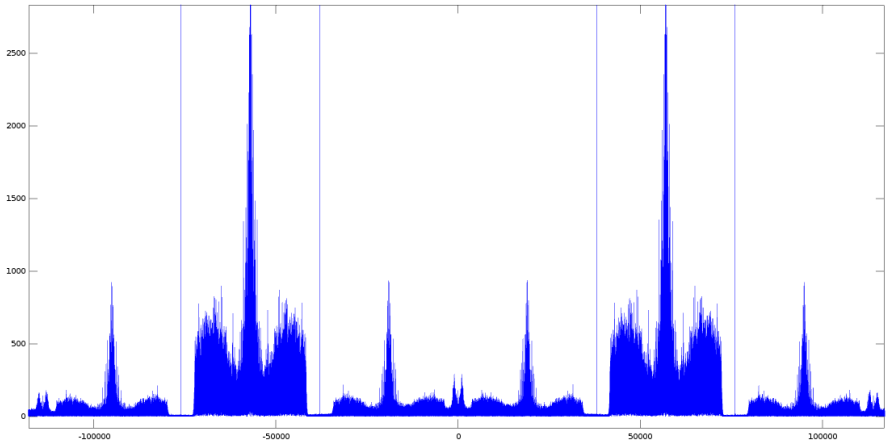


(b) Spectrum after downmixing and filtering

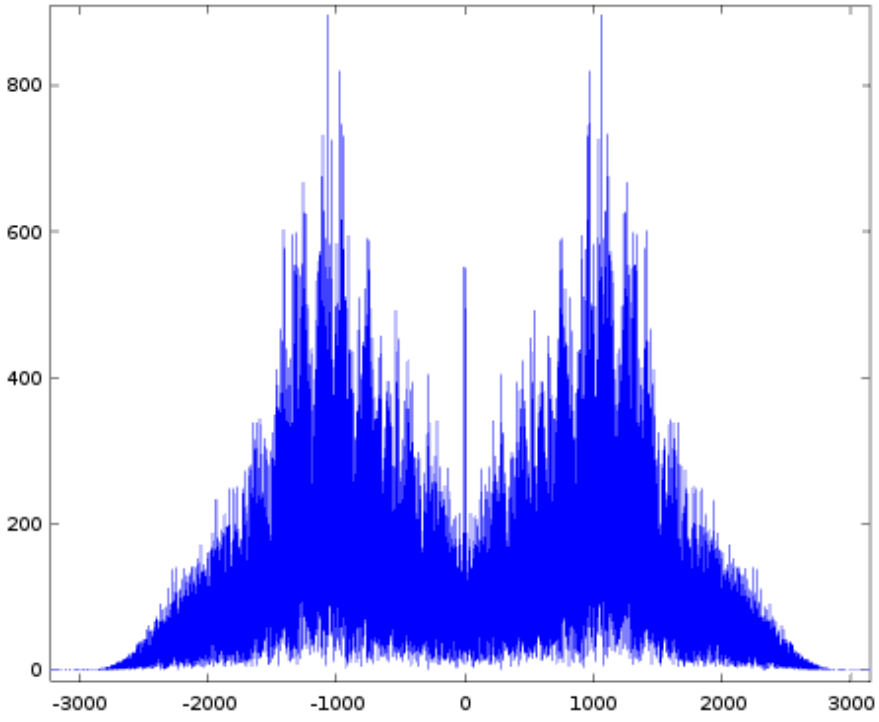
Figure 5: FM Modulated Signal



(a) Signal spectrum after FM demodulation



(b) RDS baseband spectrum after downmixing



(c) RDS baseband spectrum after filtering

Figure 6: Extracting the RDS signal from the FM signal

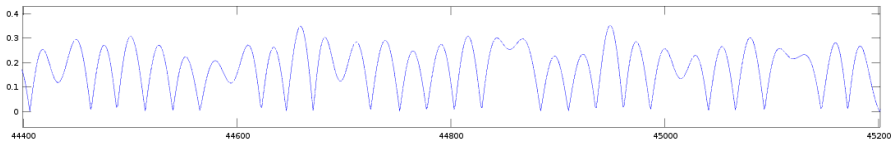


Figure 7: RDS waveform after take the absolute values

```
fl = fir1(300, 100000/1000000*2);
filtered = filter(fl, 1, down);
demod = quad_demod(filtered, 1);
t2 = linspace(0, length(demod)/1000000, length(demod));
rdscarrier = cos(2*pi*-57000*t2)';
rdsbase = demod(1:length(rdscarrier)) .* rdscarrier ;
frds = fir1(300, 2400/1000000*2);
rdsbase_filtered = filter(frds,1,rdsbase);
downsampled = decimate(rdsbase_filtered, 16).*80;
write_cuchar_binary (downsampled, "~/Downloads/
    rds_baseband_62500sps.iq");
bits = rds_bpsk_demodulate(downsampled, 62500);
rds_decode(bits)
```

The *quad_demod()* function does the quadrature demodulation (FM demodulation). The *rds_bpsk_demodulate()* function is shown in the following listing:

Listing 4: Octave implementation of the BPSK demodulation

```
function demod = rds_bpsk_demodulate(signal, fs)
  samples_per_symbol = fs/1187.5
  samples_per_symbol = ceil(samples_per_symbol)
  envelope = abs(signal);

  % Find the first minimum
  [minimum, idx1] = min(envelope(1:samples_per_symbol))

  bits = [];
  while (idx1 + samples_per_symbol*2 < length(envelope))
    % find end of symbol idx2 (minimum near idx1 +
        samples_per_symbol)
    from = round(idx1+samples_per_symbol*0.75);
    to   = round(idx1+samples_per_symbol*1.25);
    [minimum, idx2] = min(envelope(from:to));
    idx2 = idx2 + from;

    % calc mean of all samples between idx1 and idx2 and calc
        threshold = mean/2
    m = mean(envelope(idx1:idx2));
    threshold = m/2;

    % get minimum sample in the middle between idx1 and idx2 ...
    span = idx2 - idx1;
    from = round((idx1+idx2)*0.5 - 0.25*span);
    to   = round((idx1+idx2)*0.5 + 0.25*span);
```

```
    [minimum, idxmiddle] = min(envelope(from:to));
    idxmiddle = idxmiddle + from;

    % Check whether we have the correct timing. It might be, that
         idx2 is
    % actually in the middle of a symbol than at its end.
    if (envelope(idx2) > threshold)
      % In this case we find the minimum between idx1 and idx2
          and set it
      % as idx1 for the next round:
      %printf("WARNING: Wrong timing. thres=%f < envelope(idx2=%d
          )=%f\n",threshold,idx2,envelope(idx2));
      idx1 = idxmiddle;
      continue;
    endif

    % ... and check it against the threshold
    s = envelope(idxmiddle);
    if (s > threshold)
      bits = [bits 1];
    else
      bits = [bits 0];
    endif

    % idx1 = idx2 and continue with the next symbol..
    idx1 = idx2;

  endwhile
  demod = bits;
```

3.1.4 *Android Implementation*

For the Android implementation two classes are added to the AnSiAn codebase:

- BPSK: This class handles the BPSK demodulation and can be reused by other demodulators using the BPSK modulation scheme (e.g. PSK31).
- RDS: This class integrates in the existing FM class for frequency demodulation. It handles the decoding and processing of RDS groups.

A screenshot of the application demodulating the RDS signal of the *Antenne Frankfurt* station is shown in Figure 8.

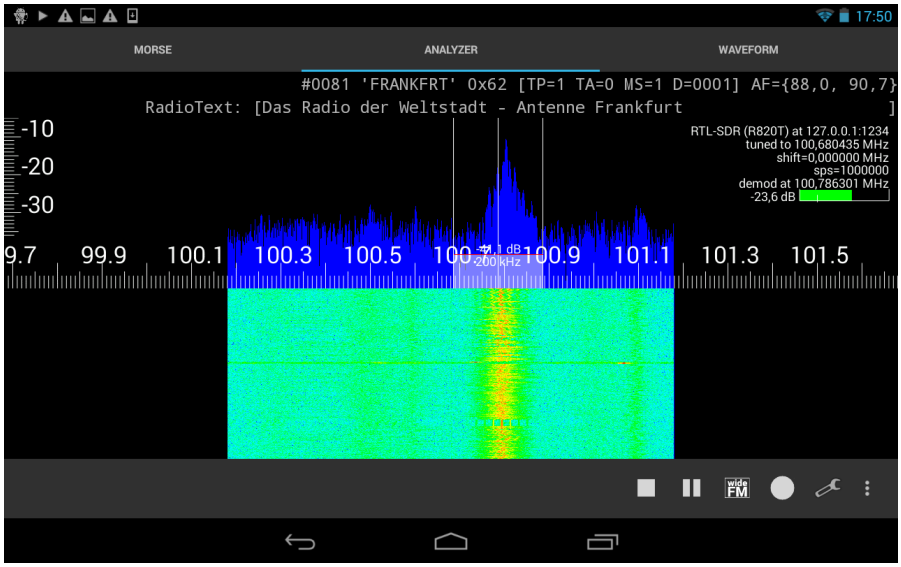


Figure 8: Screenshot of the RDS demodulator on a Nexus 7

BIBLIOGRAPHY

[1] RDS Forum Office. “The new RDS IEC 62106:1999 standard.” In: (1999).

ERKLÄRUNG

Hiermit versichere ich gemäß der Allgemeinen Prüfungsbestimmungen der Technischen Universität Darmstadt (APB) § 23 (7), die vorliegende Masterarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, 28. April 2016

Dennis Mantz and Max Engelhardt