TECHNISCHE
UNIVERSITÄT
DARMSTADT

ANSIAN - ANDROID SIGNAL ANALYZER

DENNIS MANTZ AND MAX ENGELHARDT

SEEMOO Secure Networking Lab

September 1, 2016

Secure Mobile Networking Lab
Department of Computer Science

SECURE MOBILE NETWORKING

AnSiAn - Android Signal Analyzer
SEEMOO Secure Networking Lab

Submitted by Dennis Mantz and Max Engelhardt
Date of submission: September 1, 2016

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

Technische Universität Darmstadt
Department of Computer Science
Secure Mobile Networking Lab

ABSTRACT

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetuer id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

CONTENTS

1 INTRODUCTION 1

2 PROJECT DEFINITION 3
- 2.1 Features 3
  - 2.1.1 Mandatory Features 3
  - 2.1.2 Optional Features 4
- 2.2 Time Schedule 4

3 PROJECT PROGRESS 7
- 3.1 Sprint 1: Alpha Version 7
- 3.2 Sprint 2: Beta Version 7
- 3.3 Sprint 3: Final Version 7

4 DESIGN AND IMPLEMENTATION 9
- 4.1 Optimizations and Bugfixes 9
  - 4.1.1 Memory Optimizations 9
  - 4.1.2 Bugfixes 10
- 4.2 Demodulators 10
  - 4.2.1 Design and Structural Changes 10
  - 4.2.2 Re-Implementation of Morse Demodulator 12
  - 4.2.3 Radio Data System 12
  - 4.2.4 PSK31 18
- 4.3 GUI 24
  - 4.3.1 Reorganization of Preferences 24
  - 4.3.2 Transmit Tab 24
- 4.4 Support for new SDR Platforms 24
  - 4.4.1 rad1o 24
  - 4.4.2 SDRPlay 24
- 4.5 Transmission 24
  - 4.5.1 Transmission of Raw I/Q Files 24
  - 4.5.2 Morse Modulator 24
  - 4.5.3 PSK31 Modulator 24

5 CONCLUSION 25

A APPENDIX 27

BIBLIOGRAPHY 33

LIST OF FIGURES

Figure 1 Signal processing architecture with blocking queues 10
Figure 2 Architecture of the extended demodulation logic and communication with the GUI 11
Figure 3 RDS envelope waveform after Frequency demodulation 13
Figure 4 Coding scheme of RDS: group 0A 13
Figure 5 FM Modulated Signal 15
Figure 6 Extracting the RDS signal from the FM signal 16
Figure 7 RDS waveform after take the absolute values . 17
Figure 8 Screenshot of the RDS demodulator on a Nexus 7 19
Figure 9 PSK31 modulation scheme: Each symbol is 32ms long. A 180ř phase shift indicates a '0', no phase shift indicates a '1' 20
Figure 10 PSK31 signal in the frequency domain (spectrum) 21
Figure 11 Envelope of the downsampled, USB demodulated signal. 21
Figure 12 Screenshot of the PSK31 demodulator on a Nexus 7 23

LIST OF TABLES

Table 2 Varicode Table 30
Table 1 RDS Group Types 31

LISTINGS

Listing 1 Logcat output before memory optimizations . 9
Listing 2 Logcat output after memory optimizations . . 9
Listing 3 Octave implementation of the RDS demodulator 14
Listing 4 Octave implementation of the BPSK demodulation 17
Listing 5 Octave implementation of the PSK31 demodulator 20
Listing 6 Octave implementation of the BPSK demodulation 22

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
SSB	Single Side Band
USB	Upper Side Band

1 INTRODUCTION

Android Signal Analyzer (AnSiAn) is an Android application developed by the Secure Mobile Network Lab (SEEMOO) at Technische Universität Darmstadt. It allows for common Software-Defined Radio (SDR) platforms, such as the HackRF and the RTL-SDR, to be used with Android devices. This enables a user to inconspicuously sniff and analyze wireless signals on the go.

The project is based on the open-source app RFAnalyzer by Dennis Mantz [2], which features visual browsing through the frequency domain and demodulation of e.g. Amplitude Modulation (AM) and Frequency Modulation (FM) signals. It has been further developed by Markus Grau and Steffen Kreis into what is now AnSiAn in 2015.

To date, AnSiAn adds to following features on top of RFAnalyzer:

- Waveform graph of received signalsu
- Morse Demodulation and Decoding
- RF Scanner that allows for visually scanning a broad spectrum for signals
- Restructured Graphical User Interface (GUI) with better usability
- Restructured codebase that follows the Model–View–Controller (MVC) pattern

While this makes AnSiAn a powerful tool for mobile signal analysis and processing, further features such as support for additional modulation techniques are desirable. This lab aims to further develop AnSiAn in order to extend its feature set, improve app stability and refine existing features.

A detailed descrpition and explanation of the project goals and their planned schedule is given in Chapter 2. Chapter 3 covers the actual project progress over time as well as encountered problems, delays and unplanned features in each phase of the project. Details on the design and implementation of features and bugfixes are given in Chapter 4. Chapter 5 concludes this documentation.

2 PROJECT DEFINITION

This chapter defines the features that were scheduled to be implemented throughout the project and splits them into three sprints.

2.1 FEATURES

The new features, that were to be implemented, can be divided into two groups: mandatory features of high priority within this project and optional features, that were to be implemented if time permitted. As can be seen in Section 2.2, the third sprint was reserved either for the implementation of optional features or for completing mandatory features and this documentation.

2.1.1 *Mandatory Features*

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

- Radio Data System (RDS) demodulation
  If the user selects the existing wide-band FM demodulation option, the app shall try to detect and demodulate any existing RDS signal along with the FM 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 demodulated RDS-, Morse- and PSK31-data to log-files
  If the user selects to demodulate any digital modulation, the demodulated text shall be written to a log file specified by the user.

- Support for the rad10 badge
  The rad10 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

2.1.2 *Optional Features*

Optional features were scheduled for the third and last sprint. However, they were only to be implemented if the last sprint was not needed in order to compensate for delays on the mandatory features. The optional features are listed in 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.

2.2 TIME SCHEDULE

The project had two developers, Dennis Mantz and Max Engelhardt, working in three sprints. There were 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)

3 PROJECT PROGRESS

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.

3.1 SPRINT 1: ALPHA VERSION

3.2 SPRINT 2: BETA VERSION

3.3 SPRINT 3: FINAL VERSION

4 DESIGN AND IMPLEMENTATION

4.1 OPTIMIZATIONS AND BUGFIXES

4.1.1 *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 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 reusage 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.

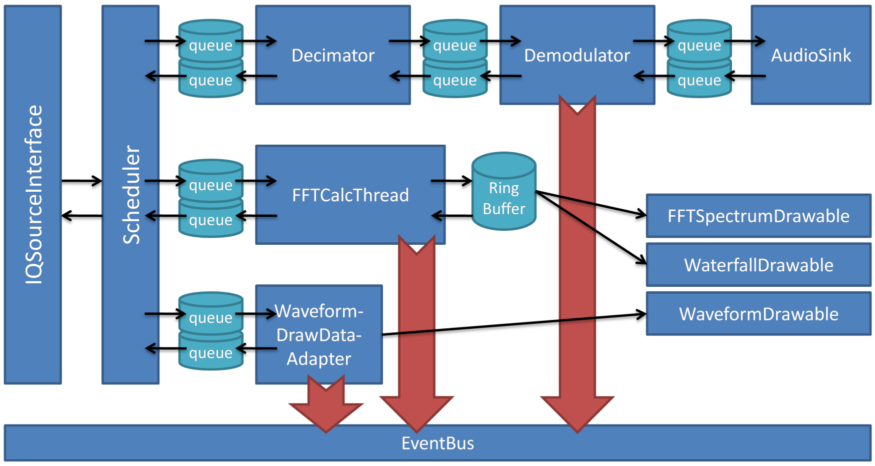


Figure 1: Signal processing architecture with blocking queues

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
```

4.1.2 *Bugfixes*

4.2 DEMODULATORS

4.2.1 *Design and Structural Changes*

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 GUI. Demodulated audio data is passed to the

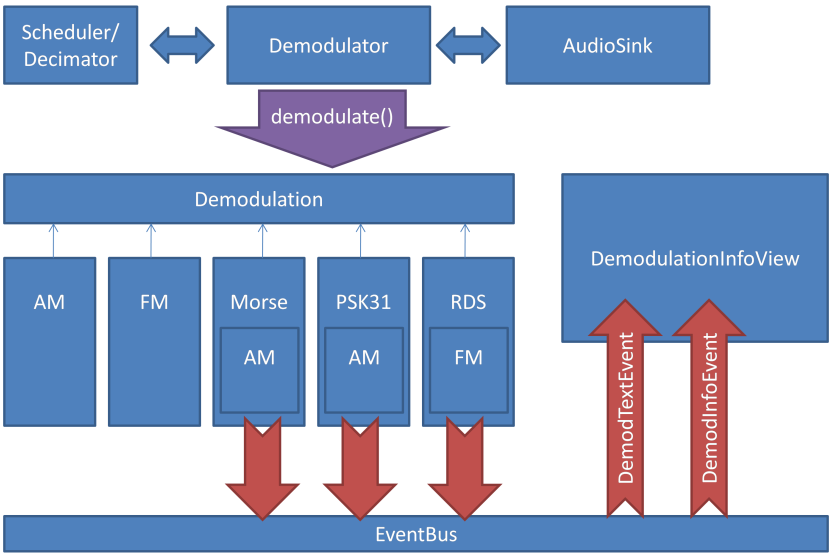


Figure 2: Architecture of the extended demodulation logic and communication with the GUI

`AudioSink` thread by enqueuing it into its input queue. This mechanism is explained in more detail in Section 4.1.1.

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 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.

4.2.2 *Re-Implementation of Morse Demodulator*

4.2.3 *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.

4.2.3.1 *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.

4.2.3.2 *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 1 lists all group types and their descriptions. The RDS demodulation in AnSiAn only decodes types 0 and 2 because they contain the basic information which is also often displayed on the radio receiver.

4.2.3.3 *Evaluation in Octave*

Developing a signal processing application on Android has many drawbacks. One issue is that it is very hard to debug the actual signal

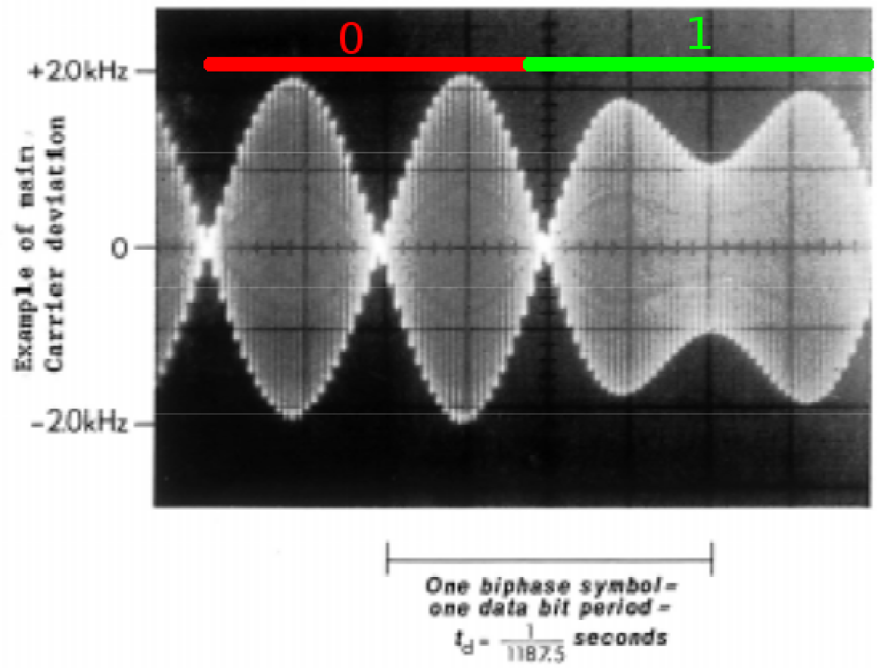


Figure 3: RDS envelope waveform after Frequency demodulation [1]

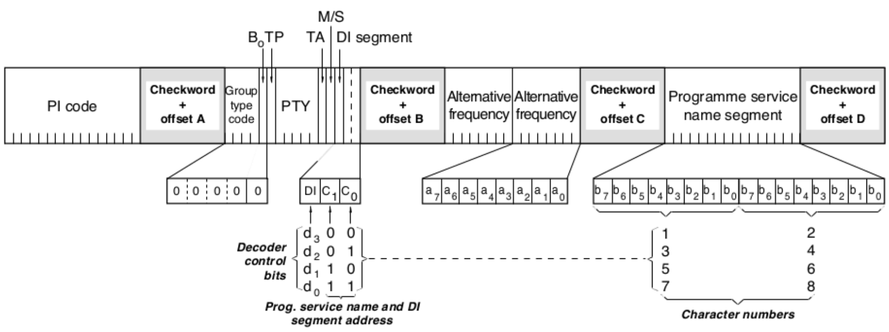


Figure 4: Coding scheme of RDS: group 0A [1]

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 libraries 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 reproducible 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* file 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.

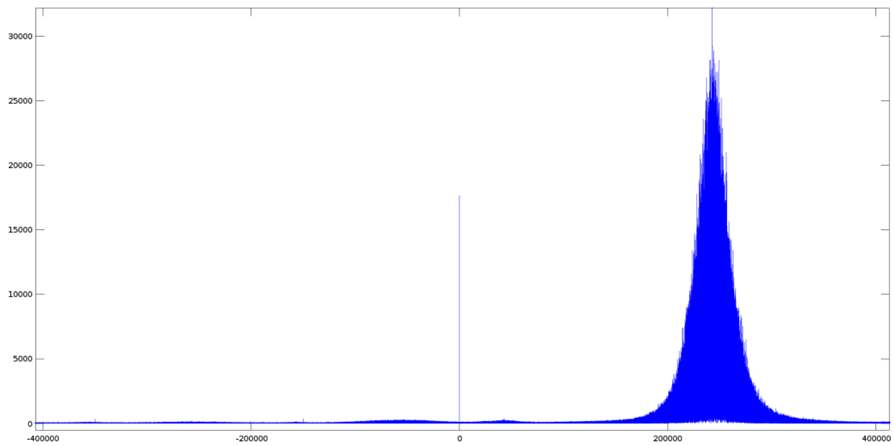
The demodulation is done in the following steps:

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.

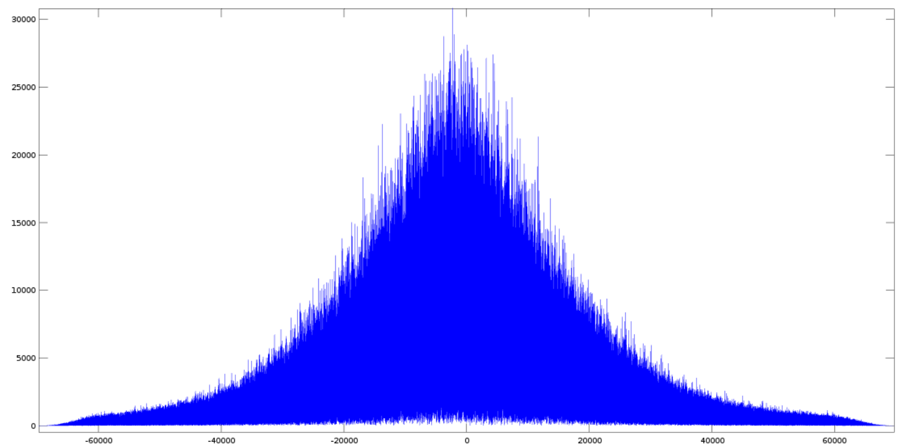
The octave code used to execute the steps mentioned above is shown in the listing below:

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;
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);
```

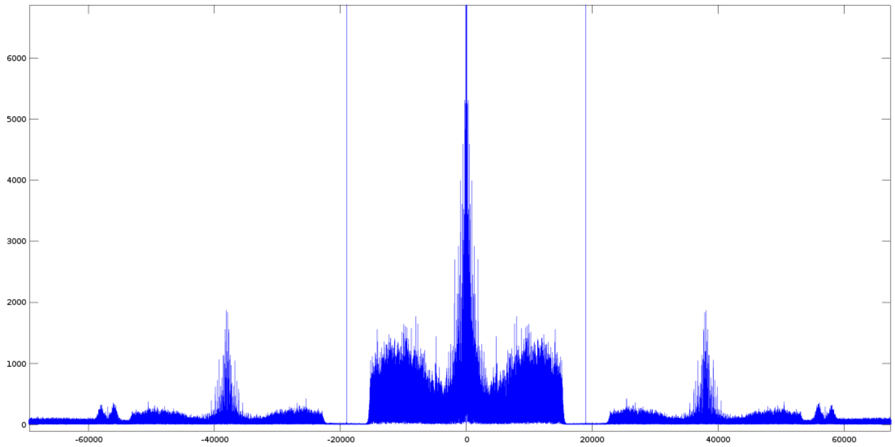


(a) Spectrum of the captured signal

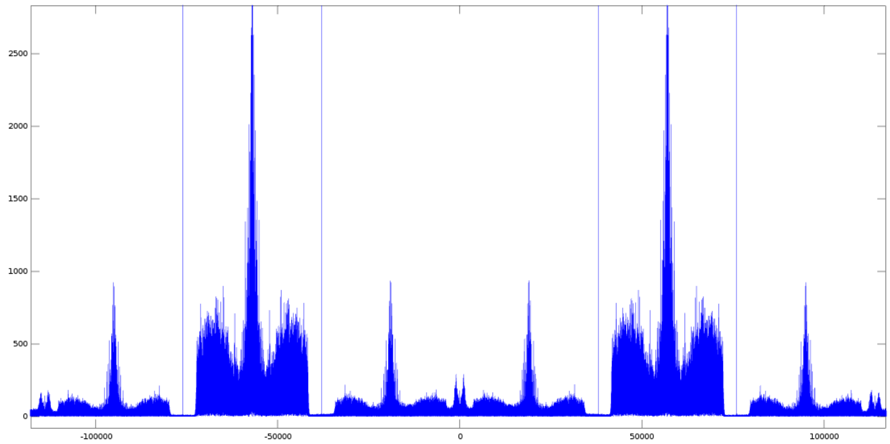


(b) Spectrum after downmixing and filtering

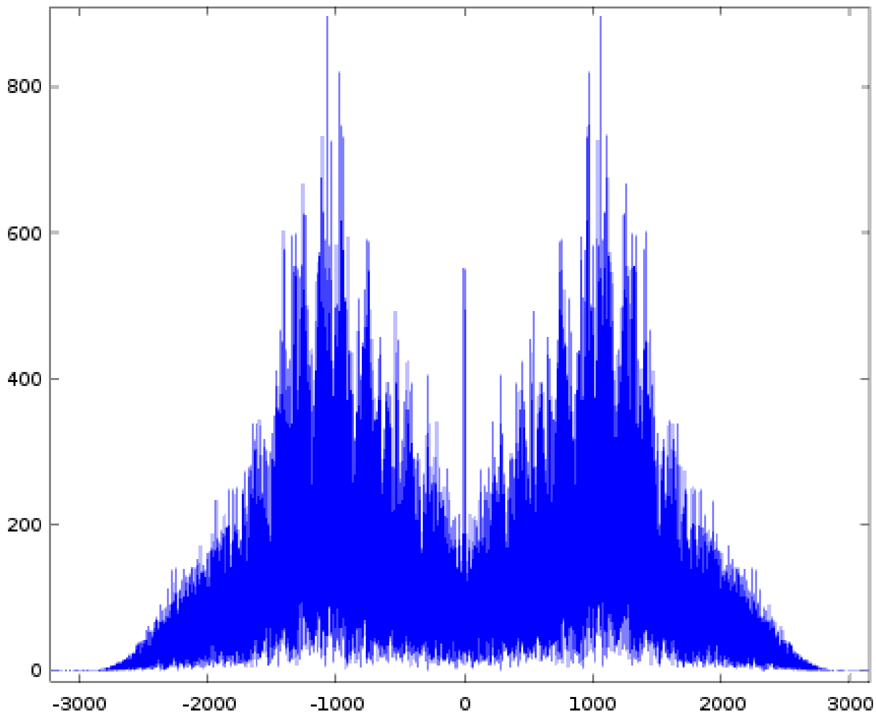
Figure 5: FM Modulated Signal



(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

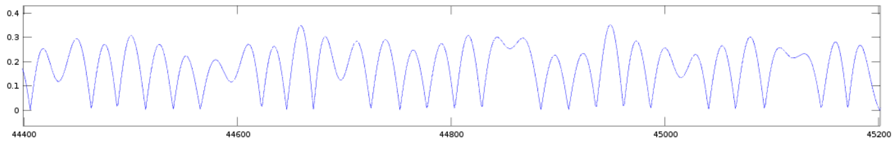


Figure 7: RDS waveform after take the absolute values

```
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;
```

4.2.3.4 *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.

4.2.4 *PSK31*

PSK31 refers to phase shift keying modulation using a baud rate of 31.25 Hz. It was developed by Peter Martinez in 1998 to introduce a narrow bandwidth digital mode for live chatting. Because of the simple and efficent modulation and coding scheme (e.g. it does not have an error correction mechanism) it is very widespread amongst amateur radio operators.

A typical setup for PSK31 operation is a Single Side Band (SSB) transceiver connected to the sound card of a computer. The audio channel that is fed into the computer can contain multiple parallel

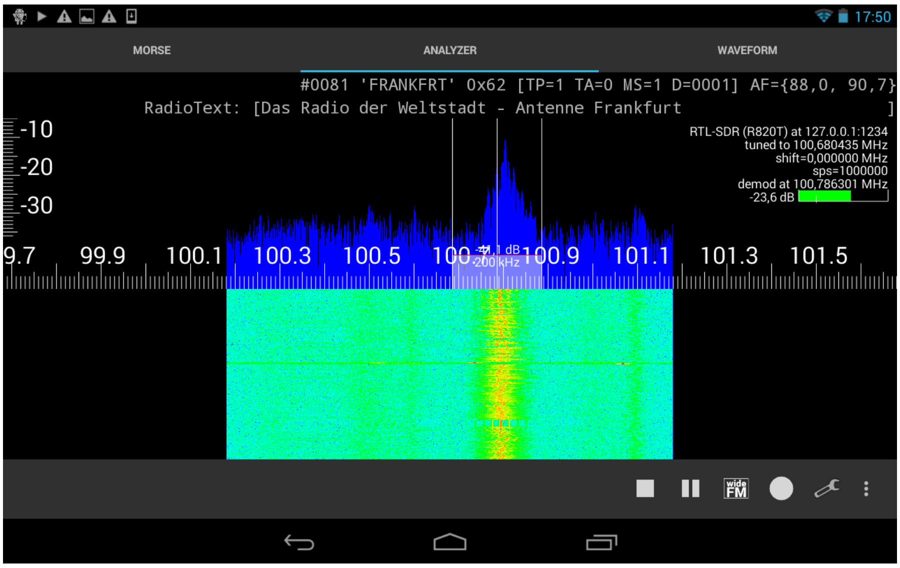


Figure 8: Screenshot of the RDS demodulator on a Nexus 7

PSK31 transmissions, which are demodulated using a special software.

PSK31 is usually operated in USB mode and shall therefore be available in AnSiAn when the USB demodulator is active. As with the RDS demodulator the algorithm to demodulate and decode PSK31 is first evaluated in Octave and later ported to the AnSiAn application.

4.2.4.1 *PSK31 modulation scheme*

PSK31 in its basic form uses BPSK to transmit binary information by sending a single side band signal with either a 180 degree phase shift (digital '1') or without a phase shift (digital '0'). Additionally a root raised cosine filter is used in order to smooth the phase shift and therefore keep the bandwidth narrow. Each symbol contains information about one bit and is always 32 ms long.

Figure 9 shows the modulated PSK31 signal. As already explained in Section 4.2.3.1, the signal may be demodulated by using a second order loop (e.g. a costas loop) in order to recover the phase information and correct for small frequency variations. However the simple approach using envelope detection is also possible. The BPSK code used for RDS demodulation can be partly reused and only needs some modifications as PSK31 does not use Manchester Encoding.

4.2.4.2 *PSK31 coding scheme*

PSK31 uses a variable length encoding called *Varicode* which assigns frequently used characters a shorter code similar to morse. Characters are separated by two consequtive zeros. Table 2 in the appendix lists all characters and their varicode encodings.

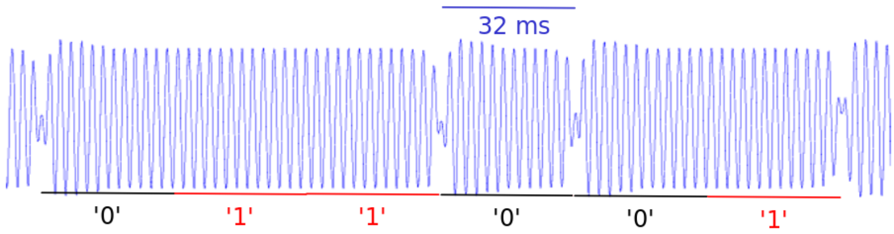


Figure 9: PSK31 modulation scheme: Each symbol is 32ms long. A 180ř phase shift indicates a '0', no phase shift indicates a '1'

4.2.4.3 *Evaluation in Octave*

The demodulation algorithm in octave comprises the following steps:

1. Downmixing the USB signal to baseband.
2. USB demodulation (complex bandpass filter to get only the upper side band). See Figure 10 b.
3. Downsampling in order to reduce the workload for the PSK31 demodulation.
4. Envelope detection. See Figure 11.
5. Find the beginning of a '0' symbol by searching for a minimum in the envelope of the signal. Determine whether the next symbol is a '0' (another minimum is found around 32ms from the beginning of the current symbol) or a '1' (no minimum at the beginning of the next symbol). If another '0' was detected, the timing can be corrected by centering the minimum at the beginning of the signal and search on from there.

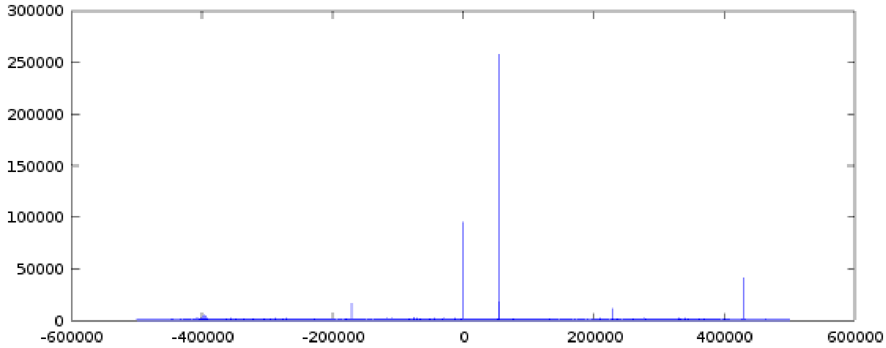
The octave code used to execute the steps mentioned above is shown in the listing below:

Listing 5: Octave implementation of the PSK31 demodulator

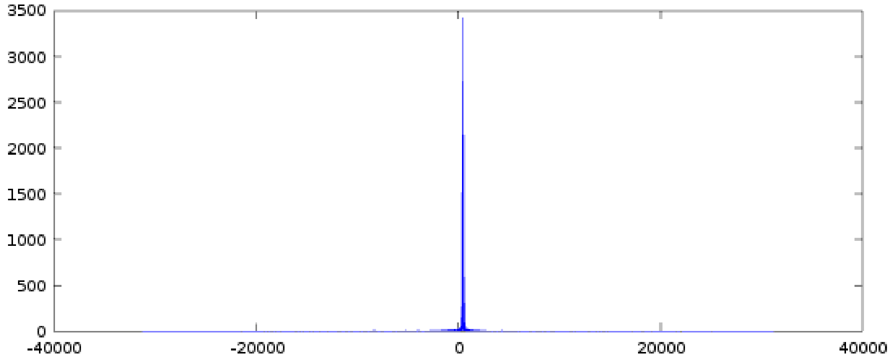
```
% load signal
signal = read_cuchar_binary ("rtlsdr_434570917Hz_1000000Sps.iq");
fs = 1000000;

% downmixing (signal is at ~ 55 Khz)
fc = 55000;
t = linspace(0, length(signal)/fs, length(signal))';
carrier = e.^(2*pi*-fc*t*i);
down = carrier .* signal;

% USB demod (bandpass filter [200Hz-1000Hz])
bpfilter = firls(300, [0 200 200 1000 1000 fs/2]./(fs/2), [0 0 0
    1 0 0]);
filtered = filter(bpfilter, 1, down);
```



(a) Raw spectrum. The PSK31 signal is at around 55 KHz.



(b) Spectrum after downmixing and filtering (USB demodulation)

Figure 10: PSK31 signal in the frequency domain (spectrum)

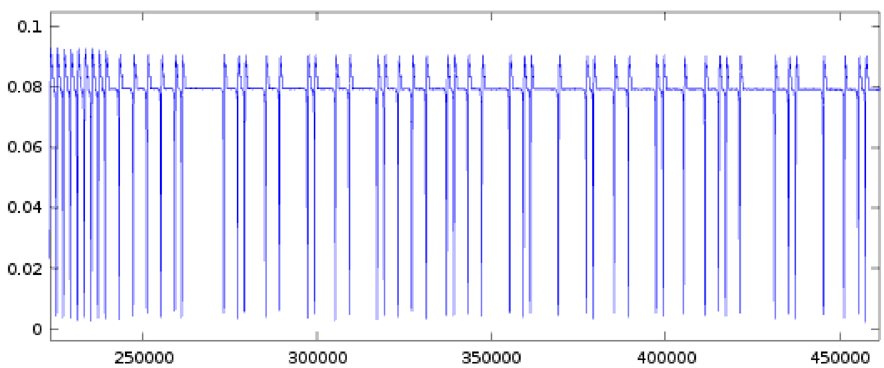


Figure 11: Envelope of the downsampled, USB demodulated signal.

```
% downsampling
downsampled = decimate(filtered, 16);

% BPSK demodulation
bits = psk31_bpsk_demodulate(downsampled, fs/16);

% Varicode decoding
psk31_decode(bits)
```

The *psk31_bpsk_demodulate()* function is shown in the following listing:

Listing 6: Octave implementation of the BPSK demodulation

```
function demod = psk31_bpsk_demodulate(signal, fs)
  % PSK31 baud rate is 31.25 Hz
  samples_per_symbol = fs/31.25
  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))
    % search for a minimum at the position of the next sample
    % (minimum near idx1 + samples_per_symbol)
    from = round(idx1+samples_per_symbol*0.5);
    to   = round(idx1+samples_per_symbol*1.5);
    [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;

    % Check whether we have a minimum (->0) or not (->1).
    if (envelope(idx2) > threshold)
      % In this case we have a bit 1.
      bits = [bits 1];
      idx1 = idx1 + samples_per_symbol;
    else
      % In this case we have a bit 0.
      bits = [bits 0];
      idx1 = idx2;
    endif
  endwhile
  demod = bits;
```

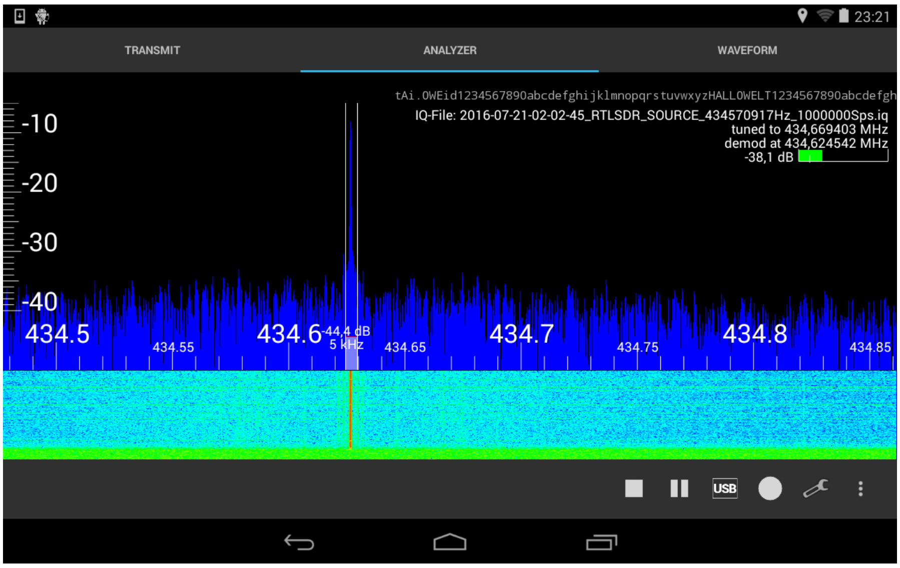


Figure 12: Screenshot of the PSK31 demodulator on a Nexus 7

4.2.4.4 *Android Implementation*

For the Android implementation the *BPSK* class is modified to handle both BPSK variations needed in AnSiAn:

- RDS: BPSK with Manchester Encoding
- PSK31: BPSK without Manchester Encoding

Additionally the class *PSK31* is added which is a subclass of *Demodulation*. It integrates with the existing *USB* class to provide PSK31 demodulation when USB demodulation is active.

A screenshot of the application demodulating a PSK31 signal is shown in Figure 12.

4.3 GUI

4.3.1 *Reorganization of Preferences*

4.3.2 *Transmit Tab*

4.4 SUPPORT FOR NEW SDR PLATFORMS

4.4.1 *rad1o*

4.4.2 *SDRPlay*

4.5 TRANSMISSION

4.5.1 *Transmission of Raw I/Q Files*

4.5.2 *Morse Modulator*

4.5.3 *PSK31 Modulator*

5 CONCLUSION

APPENDIX A

Character	Encoding
NUL	1010101011
SOH	1011011011
STX	1011101101
ETX	1101110111
EOT	1011101011
ENQ	1101011111
ACK	1011101111
BEL	1011111101
BS	1011111111
HT	11101111
LF	11101
VT	1101101111
FF	1011011101
CR	11111
SO	1101110101
SI	1110101011
DLE	1011110111
DC1	1011110101
DC2	1110101101
DC3	1110101111
DC4	1101011011
NAK	1101101011
SYN	1101101101
ETB	1101010111
CAN	1101111011
EM	1101111101
SUB	1110110111
ESC	1101010101
FS	1101011101
GS	1110111011
RS	1011111011
US	1101111111

SP	1
!	111111111
"	101011111
#	111110101
$	111011011
%	1011010101
&	1010111011
'	101111111
(	11111011
)	11110111
*	101101111
+	111011111
,	1110101
-	110101
.	1010111
/	110101111
0	10110111
1	10111101
2	11101101
3	11111111
4	101110111
5	101011011
6	101101011
7	110101101
8	110101011
9	110110111
:	11110101
;	110111101
<	111101101
=	1010101
>	111010111
?	1010101111
@	1010111101
A	1111101
B	11101011
C	10101101
D	10110101
E	1110111

F	11011011
G	11111101
H	101010101
I	1111111
J	111111101
K	101111101
L	11010111
M	10111011
N	11011101
O	10101011
P	11010101
Q	111011101
R	10101111
S	1101111
T	1101101
U	101010111
V	110110101
W	101011101
X	101110101
Y	101111011
Z	1010101101
[	111110111
\	111101111
]	111111011
^	1010111111
_	101101101
‘	1011011111
a	1011
b	1011111
c	101111
d	101101
e	11
f	111101
g	1011011
h	101011
i	1101
j	111101011
k	10111111

l	11011
m	111011
n	1111
o	111
p	111111
q	110111111
r	10101
s	10111
t	101
u	110111
v	1111011
w	1101011
x	11011111
y	1011101
z	111010101
{	1010110111
|	110111011
}	1010110101
~	1011010111
DEL	1110110101

Table 2: Varicode Table

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 1: RDS Group Types

BIBLIOGRAPHY

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

[2] *demantz/RFAnalyzer: Spectrum Analyzer for Android using the HackRF*. `https://github.com/demantz/RFAnalyzer`.

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, 1. September 2016

Dennis Mantz and Max Engelhardt