NTNU

Norwegian University of
Science and Technology

Design of ground station receiver for Kongsberg Satellite Services based on Software Defined Radio

André Løfaldli

Master of Science in Electronics
Submission date: June 2016
Supervisor: Torbjørn Ekman, IET
Co-supervisor: Kristian Jenssen, Kongsberg Satellite Service
Roger Birkeland, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Preface

The work described in this report is a Master's Thesis, concluding the 5 year long Master's programme in Electronics at the Norwegian University of Science and Technology (NTNU). It was written under supervision from Professor Torbjörn Ekman at the Department of Electronics and Telecommunications and in collaboration with Kongsberg Satellite Services (KSAT).

I would like to thank my supervisor, and Roger Birkeland, for fruitful discussions and guidance, and also Kristian Jenssen and the whole KSAT team for the opportunity to work with them.

All source code is freely available in the Github repository `lofaldli/gr-ccsds`, under the GNU GPLv3 license [1].

André Løfaldli

Trondheim,
16. June 2016

Problem Statement

KSAT would like to initiate a relationship with the next generation engineers through a project where we study the use of Software Defined Radio (SDR) to communicate with operating satellites.

The project is foreseen based on Ettus based SDR platform (or similar) and open-source GNU radio, where the focus will be on developing a SDR for two-way communication with in-orbit satellites. The student will have access to KSATs operating antennas and real satellite data.

Examples of topics to cover:

- Developing a system (based on Ettus and GNU radio) for two-way communication towards satellites. The system should consider various modulation schemes, frequencies, codings, recording data, satellite tracking, etc.
  - Study of GNU radio, its capabilities and limitations
  - Study of the Ettus SDR platform, including reception of 70 MHz IF signal from antennas; capabilities, limitations, necessary extensions/add-dons
- Configuring and testing the system towards a specific satellite
- Presentation of results and recommendation for future work at KSAT.

KSAT will allocate a budget to provide the necessary hardware, and it can also be possible to visit/work from KSATs site in Tromsø/Svalbard.

Abstract

As the space industry keeps growing, the need for low cost solutions increases. This applies not only to the launch vehicles and space segments, but also to the ground station systems. In this report, a software defined radio (SDR) ground station receiver implemented. It features an Ettus Research USRP SDR which converts an analog signal to a digital baseband representation. The baseband signal, is sent to a host computer, which runs an application that demodulates and decodes the incoming signal to extract the transmitted data.

The software component is built with GNU Radio, an open source toolkit with a rich set of signal processing features. It is designed to use a CCSDS frame format, with forward error correction, and demodulate the signal using binary phase-shift keying (BPSK). To synchronize to the symbol timing and carrier frequency of the incoming signal, a clock recovery algorithm is used in addition to a Costas Loop. This accounts for frequency offsets between transmitter and receiver, including those from Doppler shifts.

The proposed system has been tested in two ways. First at short range, between two SDR units, communicating in the VHF band using omnidirectional antennas. Here, another application for uplink communication is implemented, also to be used with an SDR, representing the satellite. The second test was done using the SDR to receive downlink data from a satellite in low earth orbit. The satellite communicates on the S-band with a data rate of 2 Mbit/s. Before the signal reaches the SDR it is processed by RF equipment and mixed down to an intermediate frequency of 70 MHz.

The results show that downlink reception from the space segment is possible. Only a few of the received data frames had too many errors for the Reed Solomon decoder to be able to correct them. Simulations show that the SDR would still be able to perform satisfactory at a lower SNR. This could potentially allow for a cost reduction in the RF equipment.

Sammendrag

Med en romindustri i vekst, øker etterspørselen etter lavkotnadsløsninger. Dette gjelder både for oppskytningsfartøy og satellitter, men også for kommunikasjonsutstyret på bakken. I denne rapporten er det implementert en mottaker for bakkestasjon med Software-definert radio (SDR). Det brukes en USRP SDR, produsert av Ettus Research, som konverterer analoge radiosignaler til digitale basisbandrepresentasjoner. Basisbandsignalet sendes videre til en PC, som kjører et program som demodulerer og dekoder det mottatte signalet, og henter ut data.

Software-delen er implementert i GNU Radio, et åpen kildekode-prosjekt med et stort antall signalbehandlingsverktøy. Den er designet for et CCSDS rammeformat med innebygd feilhåndtering, og demodulerer signalet som er binær faseskift-nøklet (BPSK). For å synkronisere symbol-tidspunkt og frekvensfeil brukes en klokkegjennvinningsalgoritme og en Costas Loop. Med dette blir frekvensvariasjoner mellom sender og mottaker tatt høyde for, inkludert de som kommer fra Doppler-effekten.

Systemet er testet på to forskjellige måter. Først en ende-til-ende test, der to SDR enheter brukes, med kommunikasjon på VHF båndet med rundstrålende antenner. Her brukes også et program for sending, med den ene SDRen som representerer satellitten. Den andre testen ble utført ved å motta data fra en satellitt i LEO-bane. Satellitten sender et signal på S-båndet med en datarate på 2 Mbit/s. Før signalet når inngangen på SDRen, blir det behandlet av RF-utstyr og mikset ned til en mellomfrekvens på 70 MHz.

Resultatene viser at nedlastning av data fra en satellitt i bane er mulig med dette systemet. Kun et fåtall av de mottatte rammene hadde for mange feil til at Reed Solomon algoritmen kunne rette de opp. Simuleringer viser at SDRen ville kunne yte på samme nivå med enda lavere SNR. Dette åpner for muligheten til å redusere kostnaden til RF-utstyret.

Contents

Chapter 1

Introduction

With the growing space industry, the barriers of entry are decreasing, both in cost and in effort. The CubeSat standard allows students at universities to be part of projects where they build satellites from scratch. These projects give students a taste of what its like to work in the space industry, as well as giving experience in teamwork. I addition, the increasing number of private companies make launching of satellites into orbit more affordable. This all means that universities and smaller companies with a tight budget can afford to launch their own satellites.

To fit with the low cost of these projects, new solutions for the ground segments are also needed. Traditional hardware for space communication is very expensive and new solutions are being developed. By using a software defined radio (SDR), the hardware complexity is reduced significantly. Typically, an SDR performs basic passband processing before sampling the signal and digitally converting to baseband signals. For baseband processing a host computer is required. It uses software implementations of traditional signal processing components, such as filters, PLLs and demodulators.

In this report, a baseband processing application has been implemented. It is based on GNU Radio, an open source software toolkit with a rich set of signal processing features. The implemented design use the CCSDS frame format with forward error correction and binary phase-shift keying (BPSK). The downlink application features a Costas Loop, which accounts for any frequency offset between the transmitter and receiver. An uplink application was also implemented for end-to-end testing.

In Chapter 2 the signal specifications will be described along with the properties of the frame format. Then the hardware requirements will be discussed in Chapter 3, both for the SDR and the host computer, and briefly for the RF chain.

The implementation details will be covered in Chapter 4. First the custom C++ GNU Radio blocks for encoding and decoding data frames, and then the flowgraphs making up the applications. Chapter 5 describes how the behavior of the implemented modules can be verified to work in the intended way. This includes using the Reed Solomon decoder to count the number of errors in the received data.

Chapter 6 describes the methods used for testing the performance of the system. Both simulating the communication channel with various signal to noise ratios, and using the system with real hardware receiving from a satellite in orbit.

In Chapter 7 the results of these tests will be discussed, and in Chapter 8 a final judgement is made. Finally, Chapter 9 covers some more topics for future work.

Chapter 2

Specification

Because this system will work with an already operational space segment, the properties of transmitted signal is already set. In this chapter the specifics of the physical and data link layer of the communication system will be described.

2.1 Signal specification

The communication between the space and ground segments take place in the S-band, with a carrier frequency of 2215 MHz. This signal is mixed down to an intermediate frequency IF at 70 MHz before further processing by the ground station.

The data is modulated using differential binary phase-shift keying (DBPSK). This is achieved by using a regular BPSK modulation where the data bits are differentially encoded. Differential encoding is equivalent to the pulse code modulation (PCM) called non-return zero mark (NRZ-M).

Two different data rates are used by the communication system. The high rate $R_H = 2\,\text{Mbit/s}$ and the low rate $R_L = 32\,\text{kbit/s}$. However, only R_H is used by the space segment.

2.2 Frame format

The frame format used by the communication system is based on a standard from the Consultative Committee for Space Data Systems (CCSDS). Its structure and

parameters are defined in the *CCSDS Recommended Standard for TM Synchronization and Channel Coding* [2].

A single frame is of length 1279 B, which is equivalent to 10 232 bit. In addition to the data to be transferred, it consists of an attached sync marker (ASM) and forward error correction (FEC) data. Each frame contain 1115 B of data, which is divided into $I = 5$ subframes of length $K = 223$.

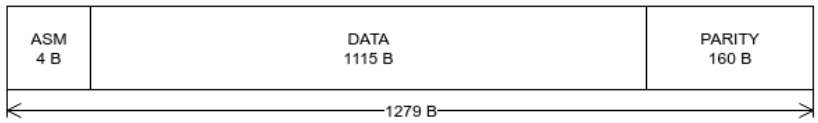


Figure 2.1: CCSDS frame structure

The frames are transmitted in a tailbiting manner. This means that the first symbol of a frame is preceded by the last symbol of the prevoius one.

The FEC method used is Reed Solomon (RS), a systematic non-binary code with symbol size $M = 8$ bit. By encoding each subframe separately they get a total length of $N = 255$, where the number of check symbols are $N - K = 32$. According to Lin and Costello [3], t errors may be corrected in a codeword with $2t$ check symbols. In this case the maximum number of errors in a subframe is $t_{max} = 32/2 = 16$. With the 4 B long ASM, the total frame length becomes $4 + I \times N = 1279$. This gives a code rate of $1115/1279 \approx 0.87$. Figure 2.1 shows the structure of a single frame.

To make the data more robust to burst errors, the subframes are interleaved symbol wise. This can be seen as as forming a matrix (2.1) from column vectors containing the subframes, and reading row-by-row. When deinterleaving the codewords, the matrix is instead read column-by-column.

$$\begin{pmatrix} x_{0,0} & x_{0,1} & x_{0,2} & x_{0,3} & x_{0,4} \\ x_{1,0} & x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,0} & x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{254,0} & x_{254,1} & x_{254,2} & x_{254,3} & x_{254,4} \end{pmatrix} \tag{2.1}$$

By using this method, a total of $I \times t_{max} = 80$ consecutive errors is correctable, assuming the frame is otherwise error free. This is because the interleaving makes sure the errors are evenly distributed into each subframe.

Another procedure used is randomization, or scrambling, of the data. This is a technique which aims to remove long sequences of `1`'s and `0`'s by using a pseudo-

random number (PRN). The PRN is generated by using a shift register, for which the generator polynome is given in (2.2) [2].

$$h(x) = x^8 + x^7 + x^5 + x^3 + 1 \tag{2.2}$$

Chapter 3

Hardware Requirements

In this chapter, the properties and requirements for the hardware in the ground station system will be discussed. First, a description of an SDR in general, with the details of the specific unit used in the experiments. Then the requirements of the host computer will be discussed, followed by a brief description of the RF chain.

3.1 Software Defined Radio

The purpose of an SDR is to sample an analog passband signal, and then convert it to a digital baseband representation. To achieve this, there are several important requirements to meet.

The RF frontend must be able to tune to the center frequency of the incoming signal. For the purpose of this system, it must be able to tune to 70 MHz which is the IF This is within the range of many popular SDRs and daughterboards [4, 5, 6].

It must also have a filter with a wide enough bandwidth to have a sample rate which satisfies the Nyquist theorem [7]. Because the system will operate in real time the communication interface between the SDR and host computer must be able to handle the data rate. In this system, the signal bandwidth is not known exactly, but it can be approximated from the data rate or estimated from an FFT plot. As will be discussed in Chapter 7, a sample rate of 10 MSample/s giving and OSF of 5 is sufficient. Using either Gigabit ethernet or USB 3 for communication with the host, the system is able to operate in real time [8].

The SDR used in this system is the *Universal Software Radio Peripheral* (USRP). It is a family of software defined radios designed and sold by Ettus Research.

The various configurations typically consists of a motherboard and a modular RF-frontend, referred to as a daughterboard.

The USRP N210 has a motherboard featuring a Xilinx Spartan FPGA and communicates with the host computer through Gigabit Ethernet. It has an analog-to-digital converter (ADC) with a 14 bit resolution and a sample rate of 100 MSample/s, and a digital-to-analog converter (DAC) with 16 bit resolution and sample rate 400 MSample/s [9].

The WBX daughterboard has a 40 MHz bandwidth and is capable of operating between 50 MHz and 2.2 GHz [4].

After converting the signal to digital representations they are transferred to the host computer. The Universal Hardware Driver (UHD), timestamps all samples to make sure the host computer receives them in the correct order and without dropping samples.

3.2 Host Computer

The host computer connected to the SDR performs all remaining signal processing in a GNU Radio application. Most of the necessary features are already built in, but the CCSDS frame functionality has been added as an out-of-tree module (OOT). Interaction with the SDR is done using the UHD module, which uses a 32 bit complex floating point number data type.

The rate at which the UHD module is producing item is determined by the *Sample Rate* parameter. This rate also determines the rate at which the blocks downstream from the *UHD: USRP Source* [10] receives items. It is important that the blocks are able to perform their operations at least as fast as their input buffers receive items. If the CPU of the host computer has insufficient calculating power, overflows might occur and negatively affect the performance of the application. For the purpose of the data rates in this system, it appears that a quad-core CPU is sufficient.

If the UHD produce more items than the flowgraph is able to handle, it will drop samples when input buffers are full. This makes communication impossible, and must be sorted out by increasing power or reducing data rates.

It is possible to write the sampled data to a file for later processing, using the *File Sink* block [10]. This block can be connected to the output port of any other block, for inspecting the signal as it goes through the flowgraph. At data rates as high as in this system, the required harddisk space quickly becomes very large. The file size may be calculated from the duration t, sample rate R_s and the size of the complex data type which is $2 \times 32\,\text{bit} = 64\,\text{bit} = 8\,\text{B}$. Equation 3.1 shows the expression for the resulting file size.

$$t \times R_s \times 8\,\mathrm{B} \tag{3.1}$$

For $R_s = 10\,\mathrm{MHz}$ this gives a file "growth rate" of $80\,\mathrm{MB/s}$.

3.3 RF chain

Before the signal is fed to this input port of the SDR it is mixed down from the S-band to an intermediate frequency (IF) signal. This is done by a chain of hardware illustrated in Figure 3.1.

The S-band signal is received by a parabolic antenna with 3 m diameter, that sits inside a radome. To fully utilize the high directivity, the antenna tracks the satellite across the sky during a pass. Before mixing, the signal is filtered to remove out-of-band noise and amplified in an LNA. A local oscillator is used to convert the signal to be centered at 70 MHz After mixing, the IF signal is filtered again to remove out-of-band noise and harmonics.

The resulting signal is the one connected to the input port of the SDR.

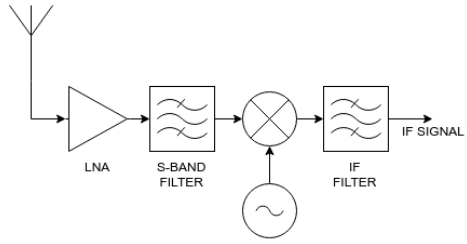


Figure 3.1: Simplified block diagram of the downlink chain

Chapter 4

Implementation

This chapter will cover the implementation details of the application running on the host computer. It is implemented as an out-of-tree module for GNU Radio, which means it will be possible to use along with its default modules [11].

GNU Radio is an open source project with a large community of active users [12]. It is implemented in C++, with a glue layer of Python for flexible application development and rapid prototyping [13]. A typical application consists of several modules, or blocks, each performing a specific task. These blocks communicate run in independent threads and communicate mainly using circular buffers. A built in scheduler makes sure the application runs smoothly and avoids buffer overflows [13].

To implement an application meeting the requirements discussed in Chapters 2 and 3, custom blocks were implemented. Together with the native blocks they form a flowgraph capable of demodulating and decoding the received signal.

4.1 Custom GNU Radio blocks

In order to support the CCSDS frame format within a GNU Radio application, as described in Section 2.2, a new module has been implemented. It contains two blocks, one for encoding and one for decoding, both are written in C++ [1]. The decoder detects the ASM at the beginning of each frame and then a starts to decode the RS codeword. If the decoding is successful, the data contained in the frame will be sent to the output port of the block. The encoder block receive data and produce frames that are compatible with the decoder, and is used for the uplink application.

To generate Reed Solomon codewords and correct errors, KA9Q's `fec-3.0.1` open source library [14] is used. It is a collection of C functions for various FEC schemes, like Viterbi and Polar codes, but only the RS functions are used in this system.

The parameters required for the RS functions are specified in [2]. The codeword length is $N = 255$, with $K = 223$ data symbols. Making the number of parity symbols the remaining $N - K = 32$

To decouple the design of the frame encoder and decoder blocks, the RS functionality is contained to its own class. This class has to has two public functions and acts as a wrapper the respective RS functions in `fec-3.0.1`. `encode()` makes a call to `encode_rs_ccsds()` which generates the check symbols. `decode()` makes a call to `decode_rs_ccsds()` which checks for any errors and corrects them if possible.

The *CCSDS Encoder* [1] block converts data into CCSDS frames encoded with RS data and the ASM. Its input port accepts asynchronous messages containing 1115 B of data.

The data is first divided into $I = 5$ arrays representing the subframes, before generating the RS parity data. These are then interleaved and inserted into the frame, as shown by the following code snippets.

```
for (i=0; i<I; i++) {
    for (n=0; n<N; n++) {
        frame[i + (I*n)] = subframe_i[n];
    }
}
```

After the subframes are interleaved to form a complete frame, the content is scrambled by `xor`'ing with the PRN generated from the polynome (2.2).

```
for (i=0; i<I*N; i++) {
    frame[i] ^= PRN[i%255];
}
```

Finally the ASM is inserted in front of the frame and it is sent from the output port as a stream of packed bytes.

The *CCSDS Decoder* [1] reverses the process from the encoder. Its input port takes a stream of unpacked bytes, which it searches for frames to decode.

It is implemented as a finite state machine (FSM) with two states; `SYNC` and `DECODE`. In the `SYNC` state it will shift one bit at a time into a 32 bit long register `data_reg`. By `xor`'ing with the ASM stored in `sync_word` the result will have `1`'s where they are different. The hamming distance is then calculated by counting the number of `1`'s using a function from the VOLK library [15].

```
data_reg = (data_reg << 1) | (in[i++] & 0x01);
wrong_bits = data_reg ^ sync_word;
volk_32u_popcnt(&nwrong, wrong_bits);
```

If the number in `nwrong` is less than a given threshold, the FSM enters the `DECODE` state. It starts by shifting in enough bits to match the length of a full frame (minus the ASM), given in (4.1), into an array of bytes.

$$1275\,\mathrm{B} \times 8\,\mathrm{bit/B} = 10\,200\,\mathrm{bit} \tag{4.1}$$

To descramble the frame, the symbols are `xor`'ed with the same PRN as in the encoder, giving back their original value. The subframes are then deinterleaved by changing the order of the `for`-loops before the RS data is checked. This is the same as changing from reading a matrix row-by-row to column-by-column.

If the number of errors t in a codeword is $0 < t < 16$ they will be corrected, and the data in the subframe extracted. In cases where $t > 16$ the RS algorithm is unable to correct the frame, and the data is lost.

4.2 GNU Radio flowgraphs

A set of GNU Radio flowgraphs have been implemented to act as the baseband processing application. Figure 4.1 shows a block diagram representation of the GNU Radio and SDR system combined. In the blocks in the GNU Radio box represent the application running on the host computer. While the blocks in the USRP box represents the FPGA and RF hardware. They are connected using the UHD interfaces found in both the host computer and the SDR.

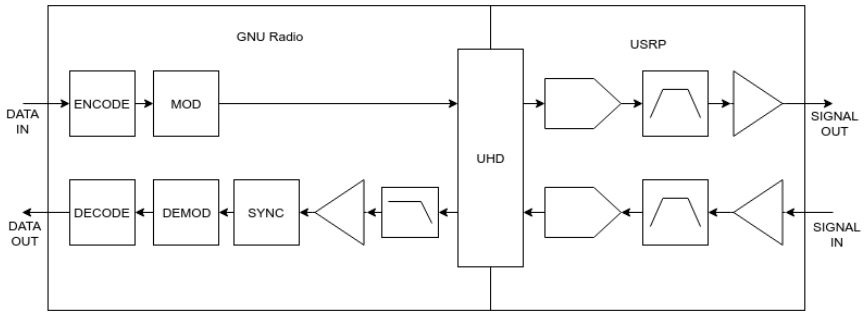


Figure 4.1: Block diagram of the SDR transceiver

4.2.1 Downlink

Figure 4.2 shows the downlink flowgraph as it is represented in the graphical design tool; GNU Radio Companion [12]. The colors on the input and output ports indicate the data types and an overview is included in Appendix C

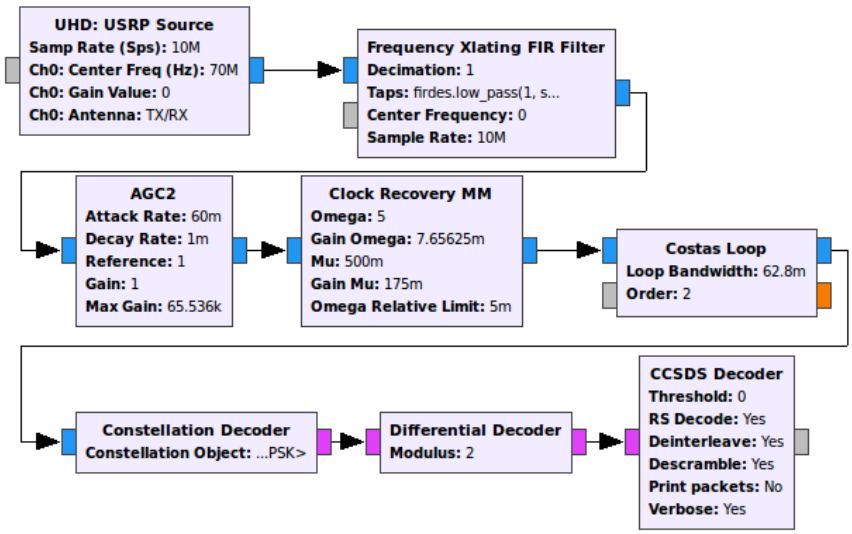


Figure 4.2: Downlink

The *UHD: USRP Source* [10] is used to generate a stream of complex samples in the downlink flowgraph. This is the digital representation of the signals received by the SDR hardware. By setting the parameters *Center Freq*, *Gain* and *Antenna* the SDR hardware may be controlled by the application.

To reduce the out-of-band noise of the received signal, the block *Frequeny Xlating FIR Filter* [10] is used. It is a finite impulse response (FIR) filter with a transfer function defined by assigning a list of numbers to the *Taps* parameter.

To generate the taps for a low pass filter the GNU Radio utility function `firdes.low_pass()` [10] is used. It takes the bandwidth and roll-off factor as arguments and calculates coefficients for the taps. Optionally, a window function may be passed as a parameter to make a matched filter. The default window function is a Hamming Window, which is optimized to minimize the power of the first side lobe [16].

In case of frequency mismatch in the received signal, the parameter *Center Frequency* may be used to translate the signal in the frequency domain, acting as a manual tuner.

The next block in the chain is the *AGC2* [10], which normalizes the signal power to the value of the *Refecence* parameter. Its operating behaviour is controlled by the other parameters, and may be appropriately adjusted during runtime. By normalizing the signal power the performance of the next blocks in the chain are improved.

To synchronize the symbol timing of the receiver, the *Clock Recovery MM* [10]

selects the optimal sample for every symbol. It is based on an optimized version [17] of the Mueller and Müller algorithm for *Timing Recovery in Synchronous Digital Receivers* [18]. By selecting the optimal sample it effectively decimates the signal by a factor set by parameter *Omega*. This value must be equal to the over sampling factor (OSF) of the received signal such that the condition (4.2) is true.

$$\text{sample rate} \times \text{OSF} = \text{symbol rate} \tag{4.2}$$

After the signal is decimated the *Costas Loop* [10] keeps track of the difference in carrier frequency Δf between the local oscillator and the received signal. This phenomenon arises partially from the fact that the transmitter and receiver have an uncertainty in their exact clock frequency. The dominating factor is however the Doppler frequency shift which appears when there is a relative velocity Δv between transmitter and receiver. A simplified expression of Δf is therefore given as in Equation 4.3.

$$\Delta f = \frac{f_{carrier}}{c} \times \Delta v \tag{4.3}$$

Phase coherence between consecutive samples is required for correct demodulation. This is because the *Constellation Decoder* [10] makes a hard decision based on the sign of the real part of each sample. Its behaviour is defined by the *Constellation* parameter.

The *Differential Decoder* [10] converts the data into its original values, before the frames are decoded in the *CCSDS Decoder* [1]. After decoding the data is passed on as asynchronous messages.

4.2.2 Uplink

Figure 4.3 shows an example uplink flowgraph, where a *Socket PDU* [10] block delivers the data to the flowgraph. This allows other applications to interact with the GNU Radio process using UDP or TCP.

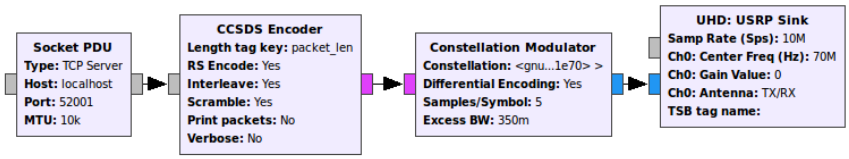


Figure 4.3: Uplink

The uplink chain takes in data as a series of asynchronous messages, which are encoded and modulated before they are sent to the SDR. As described in Section 4.1, the *CCSDS Encoder* produce a stream of packed bytes containing the encoded data.

This stream is sent to the *Constellation Modulator* [10] block which performs generic passband modulation. To determine the modulation type, the *Constellation* parameter must be assigned a Python object. By using a BPSK object, and setting the parameter *Differential* to `True`, it will perform a DBPSK modulation. It uses a root-raised cosine filter for pulse shaping, with a length defined by the *Samples per Symbol* parameter. The rolloff factor of the filter is set by the parameter *Excess BW*.

The *UHD: USRP Sink* [10] acts as the interface to the USRP hardware, and received the complex stream containing the modulated signal. It is controlled by the same parameters as the source in the downlink flowgraph.

Chapter 5

Verification

The purpose of this chapter is to verify the correctness of the system behavior. As the data transmitted from the space segment is not known, other approaches has to be used. Because the RS decoder algorithm reports if the frames are decodable, this can be used as a way of counting frame loss.

5.1 Reed Solomon

Since the data is encoded with Reed Solomon parity data, it is possible to detect any errors in transmission. Each codeword has a total of $N = 255$ symbols, where $K = 223$ symbols are data. The remaining $N - K = 32$ symbols are the check symbols [2].

Lin and Costello [3] states that t errors can be corrected in a codeword containing $2t$ check symbols. If there are more than t errors the codeword can not be decoded.

KA9Q's FEC library [14] is used to encode and decode Reed Solomon codewords. It is written in C and use a number of look-up tables (LUTs) for efficient calculation.

The data is encoded using the function `encode_rs_ccsds()`. It takes two arrays as argument, one containing the data symbols and one to write the check symbols in. The third argument indicates the padding size, for use with shorter codewords, and is set to `0` in this case. The following code shows the process of encoding a string containing known data in the form of ASCII characters.

```
uint8_t data[223] = "Lorem ipsum dolor sit amet...";
uint8_t parity[32];
encode_rs_ccsds(data, parity, 0);
```

The codeword is made up of the data, followed by the check symbols.

```
uint8_t codeword[255];
memcpy(codeword, data, 223);
memcpy(&codeword[223], parity, 32);
```

To insert an error in the codeword means changing the value of one or more of the bits in a symbol. A symbol is treated as an error if any number of bits are different, this number does not effect the correction ability.

```
codeword[42] = 0;
codeword[69] ^= 0xff;
```

After inserting a number of errors, the function `decode_rs_ccsds()` is used. It takes the codeword as the first argument which will be corrected if possible. The three remaining arguments are not used in this case and are set to `0`. The return value indicates the number of errors corrected. If correction is not possible the return value will be `-1`.

```
int8_t nerrors = decode_rs_ccsds(codeword, 0, 0, 0);
```

By ensuring the number of errors is not equal to -1 as well as comparing the corrected codeword to the original, it can be confirmed that the Reed Solomon codes work as expected. A complete C-implementation is found in Appendix A

5.2 Loopback simulation

Figure 5.1 shows the *CCSDS Encoder* and *Decoder* blocks in a GRC flowgraph. The encoder periodically receives a message containing random data, which it wraps in a CCSDS frame and outputs as a packed byte stream. Because the decoder expects an unpacked byte stream, the *Unpack K Bits* block is used with $K = 8$.

By setting the parameter *Verbose* to `True` for the decoder, the result of the Reed Solomon check may be inspected. To compare the original data with the decoded, the parameter *Print Packets* may be set to `True` on both blocks. This setup confirms that the encoder and decoder blocks are compatible with each other.

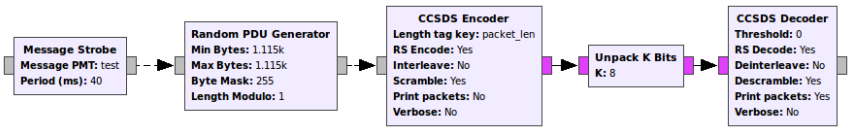


Figure 5.1: Loopback

Chapter 6

Testing

In this chapter the methods of testing the ground station receiver is described. This to get an idea of the capabilities and limitations of the system.

First, a description of a simulation which measures the performance of the receiver under various levels of SNR. This is all done within GNU Radio and assumes a channel with a complex Gaussian noise distribution.

Then, a simple end-to-end test which demonstrates communication between two SDR units with different clock phases. For this to be possible, the receiver must synchronize to the timing of the transmitter using a clock recovery algorithm.

Finally, the setup for the downlink from the space segment, which is the most important test. This because it shows how the system performs in its intended environment.

6.1 Channel Simulation

A GNU Radio flowgraph was used to test the system's performance under various levels of signal-to-noise ratio (SNR). It consists of a transmitter (TX) and a receiver (RX) with an additive white Gaussian noise (AWGN) channel connecting them. Figure 6.1 shows a block diagram representation of the communication system.

The encoder produces a set number of CCSDS frames of length 1279 B which are modulated and sent through the noisy channel. After the receiver has filtered and synchronized the incoming signal, the decoder tries to decode it It counts the number of ASM's it detects as well as the number of successfully decoded frames and subframes.

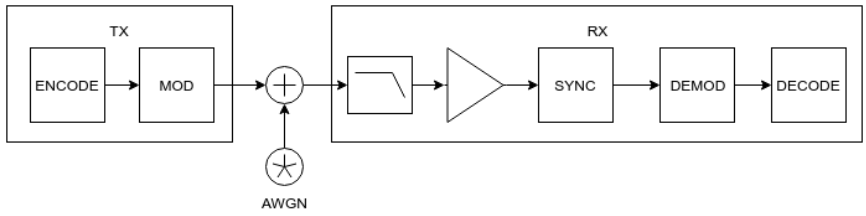


Figure 6.1: Block diagram of communication system

The channel is simulated by adding a zero-mean complex Gaussian noise signal $\mathcal{CN}(0, \sigma^2)$ to the data signal. To generate the noise signal, the *Noise Source* [10] block is used. Its parameter *Amplitude* is used to set the noise voltage amplitude N_0. The noise is added to the signal using the *Adder* [10] block.

Table 6.1 shows the how the SNR effects the frame loss of the communication system. The first column shows the value of the amplitude parameter set in the noise source. Keeping the amplitude of the data signal constant, the resulting SNR is shown in the second column. Column three shows the percentege of ASM's not detected by the decoder, if a frame is not detected it may not be decoded either.

The two rightmost columns shows the percentage of subframes and total frames, respectively, that the decoder where not able to decode. For a frame to be labeled as decodable, all five subframes must be decodable. In this experiment $N = 10000$ frames are sent through the channel, and repeated for all levels of N_0. This should give a rough number on the frame success rate, although higher numbers is preferrable be used to get a more accurate result.

N_0	SNR [dB]	SYNC [%]	SUB [%]	DEC [%]
0.1	26	0.0	0.0	0.0
0.2	20	0.0	0.0	0.0
0.3	16	0.0	0.0	0.0
0.4	14	0.0	0.0	0.0
0.5	12	0.0	0.0	0.0
0.6	10	0.0	0.0	0.0
0.7	9	0.2	0.2	0.2
0.8	8	1.0	1.0	1.0
0.9	7	3.5	4.2	4.2
1.0	6	7.8	10.8	10.9
1.1	5	16.4	39.4	30.4
1.2	4	26.6	83.2	100

Table 6.1: Packet Loss for various SNR

It appears that for SNR $> 10\,\text{dB}$, there is a very low frame loss rate, and as we will see in Section 6.3, this is well below the measured SNR.

6.2 End-to-end between two software-defined radios

A basic end-to-end test were performed using two SDR devices. One of which acts as the satellite and the other as the ground station. The SDRs in question is the USRP E310 [19] and USRP2[1] equipped with omnidirectional antennas.

The carrier frequency was set to 144.5 MHz, which is in the 2 m amateur radio band. This was chosen because it is within the tuning range of both SDRs. A data rate of 32 kbit/s was chosen because to fit the maximum sampling rate of the USRP2.

6.3 Downlink from space segment

Several tests have been made where the SDR was used to receive a signal coming from one of the satellites in the constellation. The test system consisted of the RF chain, which was discussed in Section 3.3, and a USRP N210 with a host computer connected.

The RF chain converts the S-band signal to IF, which is sampled by the SDR at a rate $R_S = 10\,\text{MHz}$. With the data rate $R_D = 2\,\text{Mbit/s}$, this equates to an over sample factor $OSF = \frac{R_S}{R_D} = 5\,\text{Sample/bit}$ These parameters are set by running the downlink flowgraph application as described in Section 4.2.1.

Other parameters that are set is the bandwidth BW at 1.4 MHz and transition width (TW) at 1.1 MHz. This removes most of the out-of-band noise still present in the sampled signal.

By using the graphical sinks native to GNU Radio, one can inspect the signal at various points in the flowgraph. This is achieved by connecting them to the output ports of the blocks in the chain.

It is also possible to store the received signal in a file by using the *File Sink* [10] block. Then this file may be used as the signal source, using the *File Source* [10] block, to analyze the signal after the pass is complete.

[1] The USRP2 is no longer available and has been repaced by the N210 [9]

Chapter 7

Results and Discussion

The experiment described in Section 6.3 was performed several times on different satellite passes. Because the satellite makes different paths across the sky each time, the receiver will see a different channel each time. The channel will also be change during a single pass, as the signal path through the atmosphere also changes.

7.1 Through the flowgraph

Figure 7.1 shows the frequency domain spectrum of the sampled signal. Because it is sampled at an OSF = 5 Sample/bit there will be some out-of-band noise which may be filtered out.

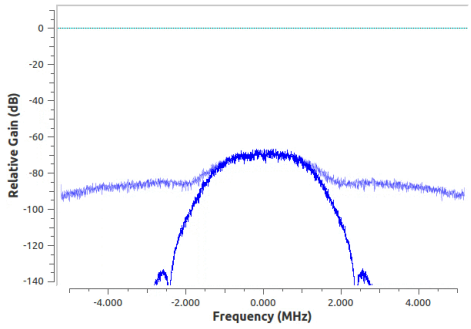


Figure 7.1: Spectrum

The raw spectrum is shown in light blue and the filtered in dark blue. Using a

low pass filter with a Hamming window function and a filter bandwidth[1] of about 4.5 MHz, the out-of-band noise is reduced significantly.

By inspecting the constellation after the signal goes through each of the blocks after the filter, one may see their effects on the signal.

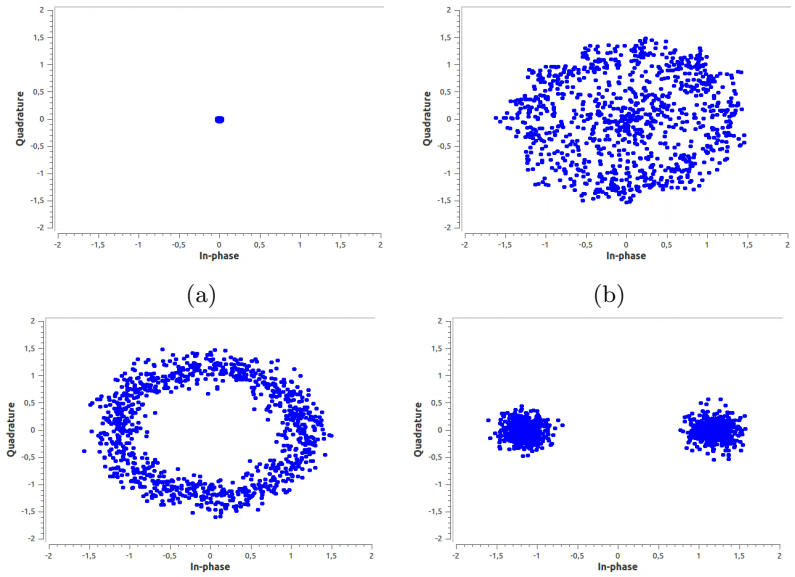


Figure 7.2: Constellation plots from various points on the flowgraph

Figure 7.2a shows the signal directly after the filter, and it is clear that the power is very low. This is confirmed by the spectrum in Figure 7.1, where the maximum power of the signal is less than -60 dB/Hz In Figure 7.2b, the signal is normalized to have values between -1 and 1, by passing it through the *AGC* [10] block. Here, the samples are apparently distributed randomly inside the unit circle. This effect is due the fact that the signal is oversampled and there is an offset in frequency.

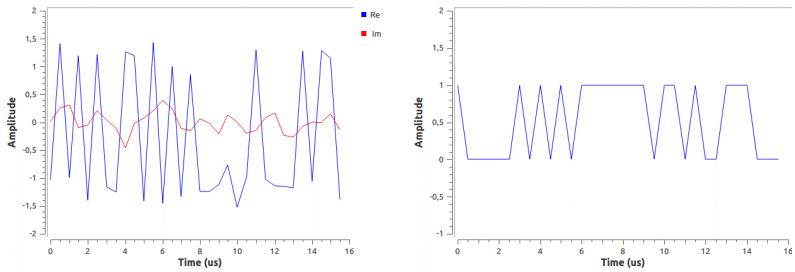
After passing through the *Clock Recovery MM* [10] block, the signal is decimated to have a sample rate equal to the symbol rate. It now only contains the samples with highest power and are therefore distributed on the unit circle, as shown in Figure 7.2c.

Using the *Costas Loop* [10], the error in center frequency is calculated and accounted for. By removing the constant phase offset between consecutive samples, the symbols are now located near the expected constellation points [7]. Figure 7.2d shows samples located near -1 and 1 on the real axis. It is clear that there is still a

[1] Here defined as the width of the main lobe

noise component present in the signal. The properties of this will be investigated in Section 7.2

Figure 7.3 shows the waveforms in the time domain before and after demodulation in the *Constellation Decoder* [10]. In Figure 7.3a the signal is NRZ, while in Figure 7.3b the demodulated bits are shown.



(a) (b)

Figure 7.3: Waveforms of time domain signals

7.2 Statistical analysis

With *File Sink* block connected to the output port of the *UHD: USRP Source* raw IQ samples are written to a file. This is useful for processing the sampled signals at a later time.

A total of 199 MB was recorded, after the synchronisation stage, which contains approximately 25×10^6 samples. They were processed using the Python script in Appendix B, to find the statistical properties of the signal. It uses the statisitics module of the SciPy library [20] for analysis and the Matplotlib library [21] for plotting.

Figure 7.4 shows a heatmap of the samples in the complex plane, with the I component in the x-direction and the Q component in the y-direction. The two non-blue areas indicate that all samples are gathered near the expected constellation points -1 and 1.

By analyzing the I and Q components as independent stochastic variables, the probability distribution of the channel noise may be estimated. Figure 7.5 shows histograms of the received samples with bell-curves drawn over for comparison.

For the I component, the samples were also divided into two categories $x > 0$ and $x < 0$. This allows each of them to be treated as two independent variables, with different μ values. The Q component was also treated as a separate stochastic vari-

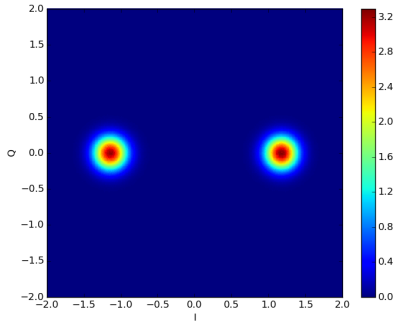
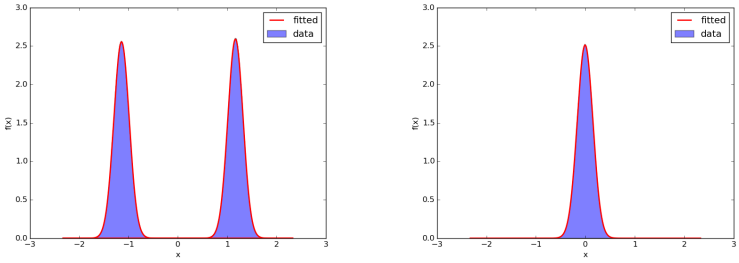


Figure 7.4: Heatmap of I/Q samples



(a) (b)

Figure 7.5: Histograms of I and Q components separately

able, independent from the other two. If the noise is truly Gaussian, the variance σ^2 of the two I signals, as well as the Q signal, should be the same.

signal	μ	σ^2
$I\|x < 0$	-1.1433	0.0227
$I\|x > 0$	1.1674	0.0223
Q	-0.0063	0.0221

Table 7.1: Statistical properties of the received signal

Table 7.1 shows the estimated statistics of the three variables, which are used to draw the fitted curves in Figure 7.5. These results shows a slight difference in σ^2, as well as in the μ values of the two I components. The μ value of Q is also not equal to 0. However, the figures show that the distribution is nearly identical to the true Gaussian fitted line.

The differences may be caused by the fact that the signal is processed by the low-pass filter, clock recovery and Costas loop before it is sampled, possibly having an effect on the distribution.

Chapter 8

Conclusion

As discussed in the previous chapter, the proposed system has been tested in two ways. The short range tests has shown that end-to-end communication is possible using two SDR units. Using omnidirectional antennas it was possible to send and receive signals on the VHF band. It was shown using the same modulation and frame format as the system is intended for, albeit at a lower bitrate of 32 kbit/s.

Utilizing the capabilities of the KSAT ground station site in Tromsø, the system was also proven capable of receiving data from a satellite in low earth orbit. The satellite communicates on the S-band with a data rate of 2 Mbit/s. Before the signal reaches the SDR it is processed by RF equipment and mixed down to an intermediate frequency of 70 MHz.

The results demonstrate that downlink reception from the space segment is possible, with a low number of undecodable frames. Simulations show that the SDR should still be able to perform satisfactory at a lower SNR. This suggests that there are possibilities for reducing the cost of the system as a whole even further, while still having acceptable performance.

Chapter 9

Future Work

To further improve the usefulness of the system there are a few more topics to be investigated.

An application for transmitting data has been implemented, and is meant to be used for sending commands to the space segment. It was has been tested on short range, but a real test with a satellite in orbit is required to verify its performance.

Another thing to investigate is the SDRs ability to handle higher data rates, on the order of 100 Mbit/s area. This would require higher order modulation schemes, such as OQPSK and 8-PSK. These are already featured in GNU Radio, and should be interchangeable with the already present BPSK blocks. Higher order modulation will also require other synchronization techniques.

To get a better picture of the computational requirements of the GNU Radio applications it would be useful to do runtime profiling of the software running on the host computer. This may allow for some optimization and removal of bottlenecks, which is critical when dealing with higher data rates.

As a final task, it would be preferable to have the whole application bundled in a single package. This would allow quick installation and setup for users. It is also desirable to have a graphical frontend for displaying key figures, like E_b/N_o, Doppler shift and frame loss, as well as options for tuning certain parameters, like filter bandwidth.

Appendix A

Reed Solomon verification (C)

```c
#include "string.h"
#include "stdio.h"
#include "stdint.h"
#include "fec.h"
#include "ccsds_tables.h"


void print_data(uint8_t*, size_t);

int main() {
    // encode data
    uint8_t data[223] = "Lorem ipsum dolor sit amet, consectetur
    adipiscing elit. Fusce varius convallis sapien. Cras eu eleifend
    quam. Ut nulla dolor, blandit eu nisl sed, vulputate pellentesque
    leo. Nam dignissim quam tortor, nec mollis sem amet.";
    uint8_t parity[32];
    encode_rs_ccsds(data, parity, 0);

    // make codeword
    uint8_t codeword[255];
    memcpy(codeword, data, sizeof(data));
    memcpy(&codeword[sizeof(data)], parity, sizeof(parity));

    // insert errors
    memset(&codeword[55], 0, 10);

    printf("data with errors:\n");
    print_data(codeword, sizeof(codeword));
    printf("\n");

    // decode data
    int nerrors = decode_rs_ccsds(codeword, 0,0,0);
    if (nerrors != -1) printf("decoded with %i errors\n", nerrors);
```

```
    else printf("could not decode\n");

    printf("data with errors corrected:\n");
    print_data(codeword, sizeof(codeword));
}

void print_data(uint8_t * data, size_t N) {
    for (int i=0; i<N; i+=10) {
        for (int j=0; j<10; j++) {
            if (i+j<N) printf("%x%x ", (data[i+j] >> 4 & 0x0f), data[i
    +j] & 0x0f);
            else printf("   ");
        }
        printf("    ");
        for (int j=0; j<10 && i+j<N; j++) {
            if (data[i+j] > 31 && data[i+j] < 127) printf("%c", data[i
    +j]);
            else printf(".");
        }
        printf("\n");
    }
}
```

Appendix B

Statistical analysis (Python)

```
import numpy as np
from scipy import stats
from gnuradio import gr, blocks
import matplotlib.pyplot as plot

def read_file(filename,N=None):
    src = blocks.file_source(gr.sizeof_gr_complex, filename)
    snk = blocks.vector_sink_c()
    tb = gr.top_block()
    tb.connect(src,snk)
    tb.run()
    data = snk.data()
    if N and N < len(data):
        N = int(N)
        data = data[:N]
    return np.array(data)

def print_stats(data,label=None):
    if label:
        print label
    for k,v in get_stats(data).items():
        print k,v
    print ''


def get_stats(data, exp_mean=0.0):
    stat = {}
    (stat['N'], (stat['min'],stat['max']), stat['mean'], stat['var'],
    stat['skew'], stat['kurt']) = stats.describe(data)
    return stat

def gauss(mean, var, N, label):
    x = np.linspace(stats.norm.ppf(0.01), stats.norm.ppf(0.99),N)
    plot.plot(x, stats.norm.pdf(x,loc=mean,scale=var**0.5), color='r',
     linewidth=2, label=label)
```

```
def histogram(data, label=None):
    plot.hist(data,color='b',
              bins=200,
              histtype='stepfilled',
              linewidth=1,
              normed=True,
              stacked=True,
              alpha=0.5,
              label='data')
    plot.xlabel('x')
    plot.ylabel('f(x)')
    s = get_stats(data)
    gauss(s['mean'],s['var'],s['N'],'fitted')

def scatter(real, imag, label=None):
    plot.hist2d(real, imag,
                bins=300,
                normed=True,
                range=[[-2.0,2.0],[-1.5,1.5]],
                label=label)
    plot.xlabel('I')
    plot.ylabel('Q')

def main():
    data = read_file('./samples.dat',1e7)
    print 'N = ' +str(len(data))
    real = data.real
    imag = data.imag

    print_stats(real[real<0],'re|x<0')
    print_stats(real[real>0],'re|x>0')
    print_stats(imag,'im')

    plot.subplot(2,2,1)
    histogram(real[real<0])
    plot.legend()
    histogram(real[real>0])

    plot.subplot(2,2,2)
    histogram(imag)
    plot.legend()

    plot.subplot(2,2,3)
    scatter(real, imag)
    plot.colorbar()

    plot.show()

if __name__=='__main__':
    main()
```

Appendix C

GNU Radio Types

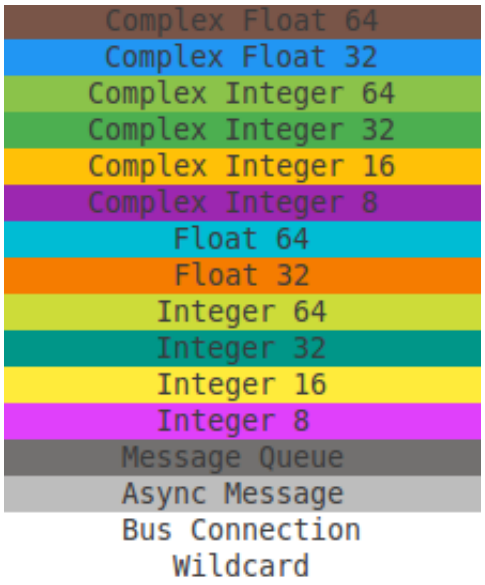


Figure C.1: GNU Radio Types

List of Abbreviations

ADC Analog-to-Digital Converter. 7

ASCII American Standard Code for Information Interchange. 17

ASM Attached Sync Marker. 4

AWGN Additive White Gaussian Noise. 19

BPSK Binary Phase-Shift Keying. 3

BW Bandwidth. 21

CCSDS Consultative Committee for Space Data Systems. 4

CPU Central Processing Unit. 8

DAC Digital-to-Analog Converter. 7

DBPSK Differential Binary Phase-Shift Keying. 3

FEC Forward Error Correction. 4

FIR Finite Impulse Response. 14

FPGA Field Programmable Gate Array. 7

FSM Finite State Machine. 12

GNU GNU's Not Unix. 1

IF Intermediate Frequency. 3

IQ In-phase/Quadrature. 23

LUT Look-Up Table. 17

NRZ-M Non-Return to Zero Mark. 3

OSF Over Sampling Factor. 15

PCM Pulse Code Modulation. 3

PDU Protocol Data Unit. 15

PLL Phase Locked Loop. 1

PRN Pseudorandom Number. 5

RS Reed Solomon. 4

RX Receiver. 19

SDR Software Defined Radio. 1

SNR Signal-to-Noise Radio. 19

TCP Transmission Control Protocol. 14

TW Transition Width. 21

TX Transmitter. 19

UDP User Datagram Protocol. 14

UHD Universal Hardware Driver. 7

USRP Universal Software Radio Peripheral. 7

VOLK Vector Optimized Library of Kernels. 12

List of Figures

List of Tables

References

[1] André Løfaldli. gr-ccsds: GitHub repository. https://github.com/lofaldli/gr-ccsds. (accessed 15.06.16).

[2] TM Synchronisation and Channel Coding. Recommended Standard 131.0-B-2, CCSDS, 2011.

[3] Shu Lin and Daniel J. Costello Jr. *Error Control Coding*. Pearson, 2nd edition, 2004.

[4] Ettus Research. WBX Daughterboard product page. https://www.ettus.com/product/details/WBX. (accessed 12.05.16).

[5] osmocom. RTL-SDR project page. http://sdr.osmocom.org/trac/wiki/rtl-sdr. (accessed 15.06.16).

[6] Great Scott Gadgets. HackRF product page. http://greatscottgadgets.com/hackrf/. (accessed 15.06.16).

[7] Simon Haykin. *Communication Systems*. Wiley, 4th edition, 2000.

[8] SuperSpeed USB. http://www.usb.org/developers/ssusb. (accessed 15.06.16).

[9] Ettus Research. USRP N210 product page. https://www.ettus.com/product/details/UN210-KIT. (accessed 13.06.16).

[10] GNU Radio Manual and C++ Reference. http://gnuradio.org/doc/doxygen/index.html. (accessed 16.06.16).

[11] GNU Radio website: Out-of-tree modules. http://gnuradio.org/doc/doxygen/page_oot_config.html. (accessed 16.06.16).

[12] GNU Radio website. http://gnuradio.org/. (accessed 16.06.16).

[13] GNU Radio website: Flowgraphs. http://gnuradio.org/doc/doxygen/page_operating_fg.html. (accessed 16.06.16).

[14] Phil Karn (KA9Q). fec-3.0.1 C++ library. http://www.ka9q.net/code/fec/. (accessed 30.04.16).

[15] Vector Oprimized Library of Kernels (VOLK library. http://libvolk.org/. (accessed 12.05.16).

[16] Wikipedia: Hamming Window. https://en.wikipedia.org/wiki/Window_function#Hamming_window. (accessed 12.05.16).

[17] G.R. Danesfahani and T.G. Jeans. Optimisation of modified Mueller and Müller algorithm. *IEEE Electronic Letters*, 31(13), 1995.

[18] Kurt H. Mueller and Markus Müller. Timing Recovery in Digital Synchronous Data Receivers. *IEEE Transactions on Communications*, COM-24(5), 1976.

[19] Ettus Research. USRP E310 product page. https://www.ettus.com/product/details/E310-KIT. (accessed 13.06.16).

[20] SciPy: Statistical functions. https://docs.scipy.org/doc/scipy/reference/stats.html. (accessed 24.05.16).

[21] Matplotlib: Python plotting library. http://matplotlib.org/. (accessed 24.05.16).