Master Thesis

Development, Implementation and Assessment of DVB-S2 Frame Synchronization and SNR Estimation on the GNU Radio Platform

Eral Türkyilmaz

Institute of Communication Networks and Satellite Communications
Graz University of Technology, Austria
Head: Univ.-Prof. Dipl.-Ing. Dr. Otto Koudelka

Assessor:
Univ.-Prof. Dipl.-Ing. Dr. Otto Koudelka

Supervisor:
Dipl.-Ing. Dr. Wilfried Gappmair

Graz, January 2012

Deutsche Fassung:
Beschluss der Curricula-Kommission für Bachelor-, Master- und Diplomstudien vom 10.11.2008
Genehmigung des Senates am 1.12.2008

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Graz, am 31.1.2012 .....

.....
(Unterschrift)

Englische Fassung:

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

31.1.2012
.....
date

.....
(signature)

Abstract

Future applications in satellite communication services will need more and more bandwidth. This makes the use of higher frequency bands like Q/V necessary. Hence studies of satellite communications in the Q/V-band will be performed by the European Space Agency (ESA) in the context of the ALPHASAT project using a DVB-S2 transmission format.

Due to the high frequencies in the Q/V-band, the propagation effects are significant and deep fades can occur. Dynamic link capacity assignment with Adaptive Coding and Modulation (ACM) is an effective way for fade mitigation. But ACM needs SNR estimation and MODCOD (Modulation Scheme and Coding) determination out of the sent DVB-S2 signal to work properly.

The following thesis will provide a famework for these necessary tasks on a GNU Radio platform. In this respect, possible frame synchronization algorithms for DVB-S2 are examined and a suitable algorithm will be implemented on the platform. Additionally, performance-critical operations should be realized in FPGA hardware.

Furthermore, SNR estimation algorithms are needed to monitor the current channel conditions. Therefore an assessment of different SNR estimations available from the open literature has been done. Data-aided and non-data-aided variants of the SNR estimators are in the sequel applied to the DVB-S2 format and evaluated.

Additionally, a receiver architecture has been developed and implemented on the chosen GNU Radio platform. The whole system was then tested in a laboratory setup.

Acknowledgements

I would like to thank Univ.-Prof. Dipl.-Ing. Dr. Otto Koudelka for giving me the opportunity to work on this thesis in a professional field at Joanneum Research.

Furthermore, many thanks must be given to my advisors at Joanneum Research DIGITAL SPA (Space Technology and Acoustics), namely Dipl.-Ing. Harald Schlemmer, Dipl.-Ing. Michael Schmidt and Dipl.-Ing. Dr. Johannes Ebert. I am very grateful for their countless hours of support during this work.

Special thanks go to my supervisor, Dipl.-Ing. Dr. Wilfried Gappmair for proofreading, giving me support in mathematical concerns and for providing me with literature as background for my work.

Moreover, I would like to express my gratitude to all the people who have supported me in any way during my work on this thesis.

Contents

List of Figures

List of Tables

List of Acronyms

8-PSK	8-ary Phase Shift Keying
16-APSK	16-ary Amplitude and Phase Shift Keying
32-APSK	32-ary Amplitude and Phase Shift Keying
ACM	Adaptive Coding and Modulation
AGC	Automatic Gain Control
AWGN	Additive White Gaussian Noise
BB	Baseband
BER	Bit Error Rate
BPSK	Binary Phase Shift Keying
CDF	Cumulative Distribution Function
CFO	Carrier Frequency Offset
CIC	Cascaded Integrator Comb
CNR	Carrier-to-Noise Ratio
CR	Code Rate
CRC	Cyclic Redundancy Check
CRLB	Cramer-Rao Lower Bound
DA	Data-Aided
DC	Direct Current
DD	Decision-Directed
DFT	Discrete Fourier Transform
DRM	Digital Radio Mondiale
DSP	Digital Signal Processor
DTH	Direct To Home
DVB	Digital Video Broadcasting
EEP	Equal Error Protection
EIRP	Equivalent Isotropically Radiated Power
ESA	European Space Agency
ETSI	European Telecommunications Standards Institute
FEC	Forward Error Correction
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FM	Frequency Modulation
FPGA	Field Programmable Gate Array
GRC	GNU Radio Companion
HBF	Half Band Filter
HPA	High Power Amplifier
ICI	Inter-Carrier Interference
IDFT	Inverse Discrete Fourier Transform
IFFT	Inverse Fast Fourier Transform
I/Q	In-Phase/Quadrature-Phase
ISI	Inter-Symbol Interference
LNA	Low Noise Amplifier
MCRLB	Modified Cramer-Rao Lower Bound

ML	Maximum Likelihood
MODCOD	Modulation and Coding
MSB	Most Significant Bit
MSE	Mean Squared Error
NCRLB	Normalized Cramer-Rao Lower Bound
NDA	Non-Data-Aided
NMSE	Normalized Mean Squared Error
PDF	Probability Density Function
PL	Physical Layer
PLSC	Physical Layer Signalling Code
PLH	Physical Layer Header
PSK	Phase Shift Keying
QAM	Quadrature Amplitude Modulation
QPSK	Quarternary Phase Shift Keying
RF	Radio Frequency
RRC	Root-Raised Cosine
SA	Spectrum Analyzer
SDR	Software Defined Radio
SER	Symbol Error Rate
SNR	Signal to Noise Ratio
SOC	System-On-Chip
SOF	Start of Frame
STO	Symbol Timing Offset
UHD	Universal Hardware Driver
VCM	Variable Coding and Modulation

1. Introduction and Motivation

1.1. Q/V-Band Communication Experiment

Nowadays most of the lower satellite frequency bands are already heavily congested. This makes an investigation of higher frequency bands necessary. The Q/V-band ($35\,GHz$ to $75\,GHz$) is a potential candidate which could provide additional bandwidth for new applications. Since at such high frequencies the wavelengths are very small, wave propagation effects have a significant impact. The traditional approach of simply providing a large link margin for fades by using high EIRP and G/T figures are impractical. Adaptive Coding and Modulation (ACM), offered by the DVB-S2 standard, provides a more effective approach by performing a dynamic link adaptation to the current propagation conditions.
The ALPHASAT satellite, launched in late 2012, will carry an experimental Q/V-band payload. Communication experiments will use DVB-S2 as transmission format and will therefore leaving focus on ACM for fade mitigation.[Kou11][RCL+09]

1.2. Scenario

The principal scenario of the Q/V-band communication experiment is shown in figure 1.1. The real experiment will use a bidirectional communication link, but for simplicity reasons it is only shown in one direction (unidirectional).

The payload data are generated by a packet generator, modulated into the DVB-S2 format according to the current MODCOD (modulation scheme and coding), and converted to the L-Band. The L-Band signal is then upconverted into the Q/V-band. The resulting signal is amplified by a High Power Amplifier (HPA), fed to the antenna and sent to the satellite. The ALPHASAT transponder prepares the signal for the downlink. The receive station amplifies the signal by a Low Noise Amplifier (LNA) and performs the downconversion from Q/V-band to the L-Band. The demodulator now performs all the processing to extract the payload data out of the DVB-S2 frames. A packet analyzer shows how many packets were sent correctly over the link.

For the ACM experiment it is important to provide the sending base station with a feedback on the channel conditions at the receive base station. Therefore we need an accurate SNR estimate. The monitoring parameters (MODCOD and SNR) can be returned to the sender by using an appropriate return channel (e.g.: return satellite channel, terrestrial link). In the sending base station the experiment is controlled by an ACM algorithm that decides when to switch the MODCOD based on the SNR estimates of the receiver station (each available transmission format has a certain minimal SNR to offer quasi error free transmission [ETS09, p. 34]). Additionally, the experiment will be monitored which means that all received return data (MODCODs and SNR) are logged at the sending base station.

The experiment will cover an SNR dynamic range of about 12.2 dB where the lowest modulation scheme is QPSK 1/2 (min. SNRs 1.6 dB) and the highest modulation scheme is 16-APSK 8/9 (min. SNRs 13.8 dB) .

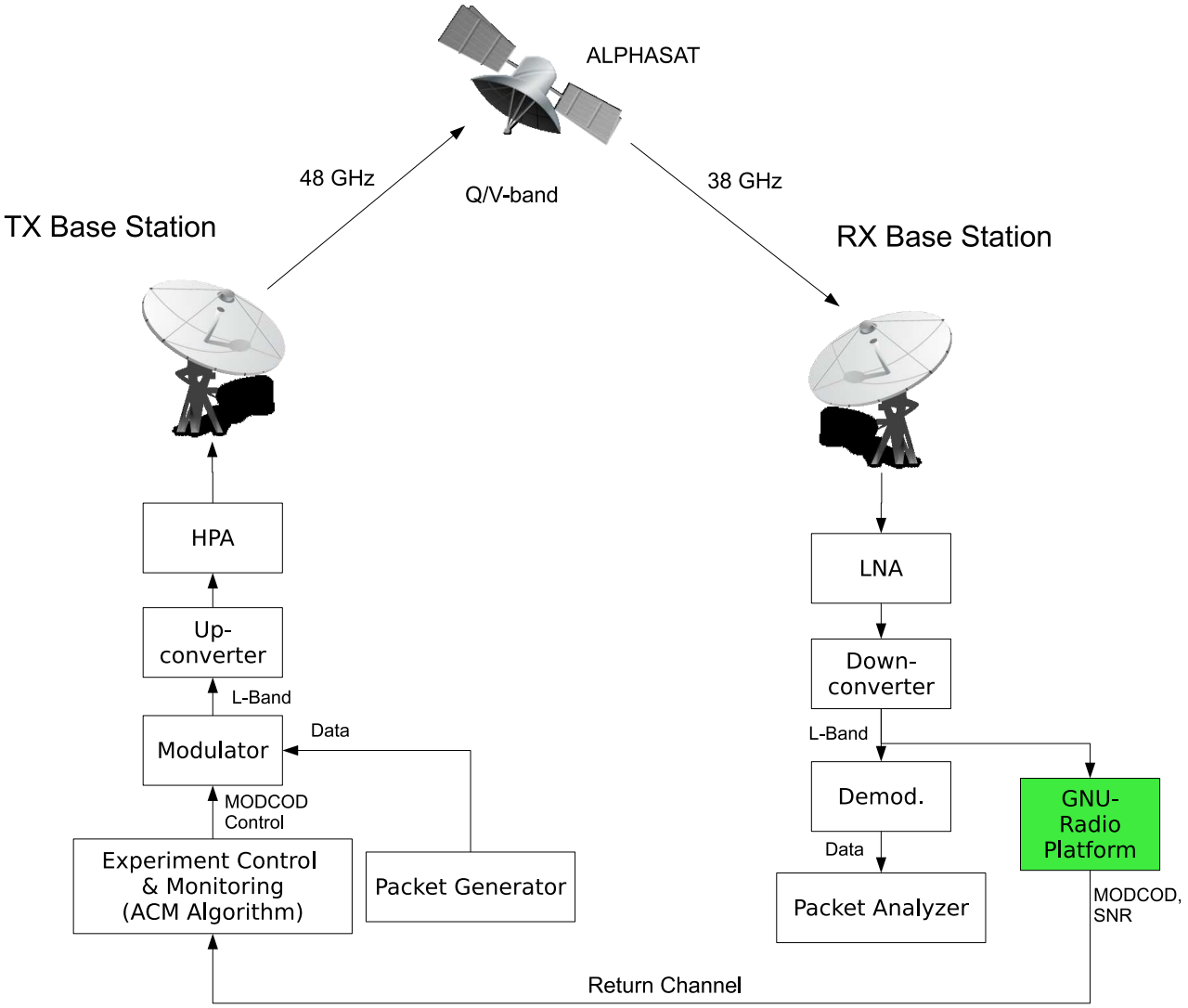

Figure 1.1.: Communication experiment (unidirectional)

1.3. Using the GNU Radio SDR Platform

The existing modems offer an SNR estimation but there is no information available about the used algorithm and the time granularity of these SNR estimates.Therefore it is necessary to provide a customized solution. The advantage is that we can realize our own methods and, based on this, select the required MODCODs.
A flexible Software Defined Radio (SDR) platform (USRP + GNU Radio running on the PC) has been chosen to implement this tasks. A short overview of the SDR platform is shown in figure 1.2. The incoming RF signal is downconverted, sampled and decimated by the USRP. In the sequel the samples are sent to the PC where the GNU Radio platform processes the incoming samples.

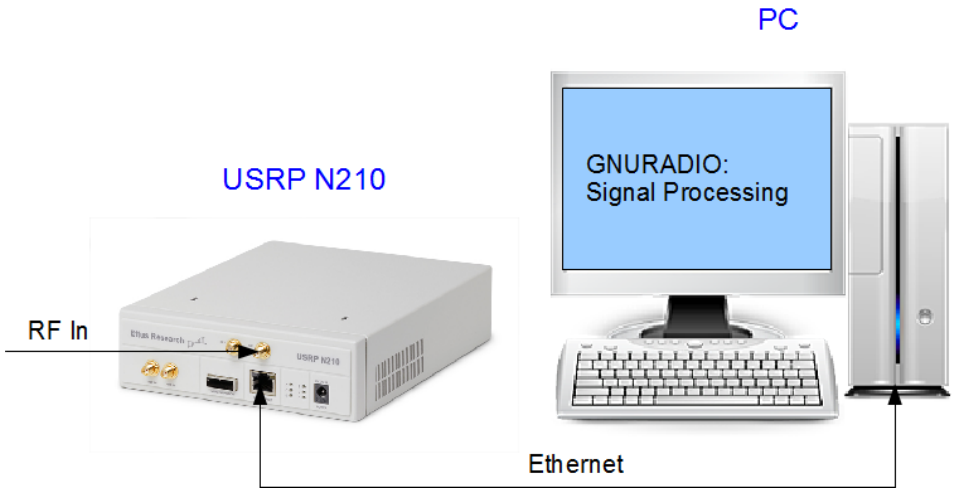

Figure 1.2.: The GNU Radio platform (USRP + PC)

1.4. Tasks

Two important tasks have to be performed on the SDR platform. On the one hand, a frame synchronization has to be achieved to get the current MODCOD and, on the other hand, an SNR estimation has to be performed to evaluate the current channel conditions.
The goal of this Master Thesis is to develop a prototype for DVB-S2 frame synchronization and SNR estimation for the Q/V-band communication experiment. The algorithms will be implemented on the GNU Radio platform in combination with the USRP N210. Performance-critical operations should be implemented on the FPGA of the USRP. Additionally, an assessment of the different algorithms has to be performed. The following list summarizes the tasks to be performed in the context of the current Master Thesis:

- Frame synchronization algorithms for DVB-S2 have to be analyzed; an appropriate algorithm will be chosen which has to be:
  - implemented in software as floating-point and bit-accurate model (C++ GNU Radio modules)
  - simulated in software to evaluate the performance and to get a reference for the hardware implementation
  - implemented in FPGA if necessary (parts of the algorithm)
- Different SNR estimators have to be selected and evaluated; they have to be:
  - implemented in software (C++ GNU Radio modules)
  - simulated in software to measure their performance
- Developing a prototype on the GNU Radio platform:
  - propose and implement an architecture
  - practical measurements to evaluate the performance of the implemented system
  - comparison to simulation

2. The DVB-S2 Standard

2.1. History

The DVB (Digital Video Broadcasting) Project introduced the DVB-S standard in 1994. It is intended for DTH (Direct-to-Home) video broadcasting applications and at the time of writing used by most satellite operators worldwide. It offers QPSK modulation and uses a convolutional code with Viterbi decoding in combination with a Reed-Solomon code.
In 1997 DVB-DSNG was introduced. This standard also specifies 8-PSK and 16QAM modulations. [ETS09]
In 2003 DVB-S2 was defined by the DVB Project. It is the second-generation specification for broadband satellite applications. It introduced additional higher order modulation schemes like 16-APSK and 32-APSK and also used recent developments in channel coding to approach the channel capacity. So DVB-S2 can reach a performance gain of about 30% in comparison to DVB-S at a given transponder bandwidth and transmit EIRP [MM06]. DVB-S2 was later standardized by the ETSI (European Telecommunication Standards Institute) and characterize by three key concepts:

- total flexibility
- best transmission performance
- reasonable receiver structure complexity

2.2. Transmitter Block Diagram

The principle transmission flow block diagram is shown in figure 2.1. Each block is designed to operate independently and provides only its interfaces to its connected blocks. In this Thesis only the lower layers (namely: MAPPING, PL FRAMING and MODULATION) of the DVB-S2 standard are covered.

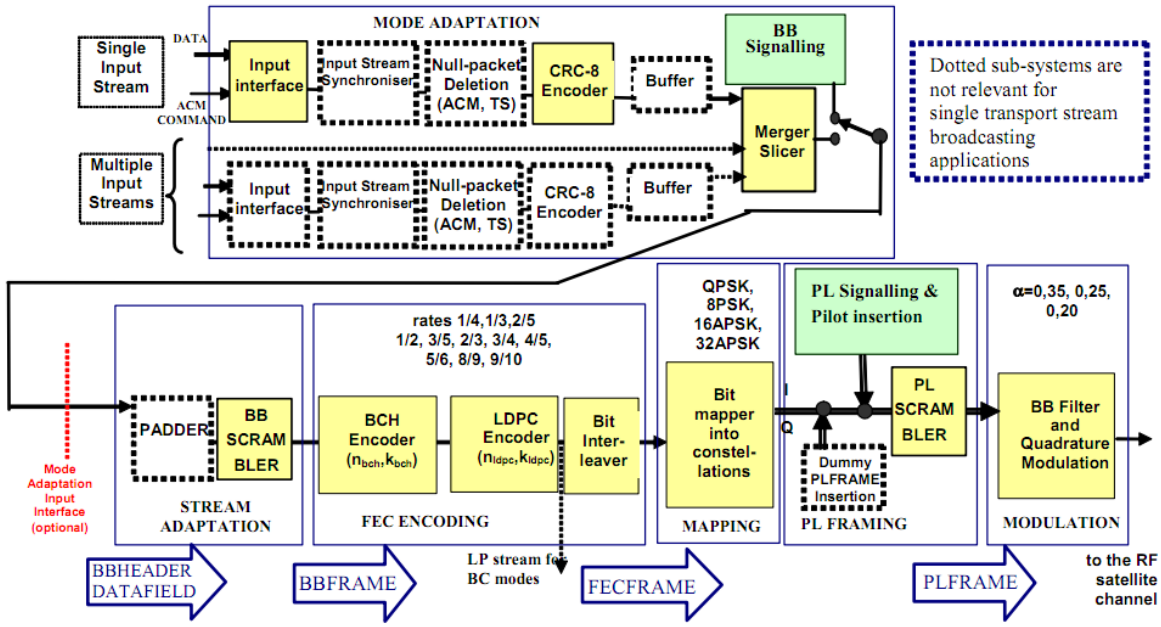

Figure 2.1.: Functional block diagram (from [ETS09])

2.3. Forward Error Correction Encoding

Forward Error Correction (FEC) encoding is provided by an inner LDPC (Low-Density Parity Check) code and an outer BCH (Bose-Chaudhuri-Hocquenghem) code. The length of a FECFRAME may either be 64800 coded bits (normal frame) or 16200 bits (short frame).

2.4. Constellation Mapping

2.4.1. Modulation

DVB-S2 offers the following modulations for the payload data:

- QPSK
- 8-PSK
- 16-APSK
- 32-APSK

According to [MM06] QPSK and 8-PSK are typically used for broadcast applications since they have a constant envelope and therefore they are not so strongly impaired by a nonlinear

channel. 16-APSK and 32-APSK are targeted towards professional applications. The 16-APSK and 32-APSK offer better performance on the nonlinear channel in comparison to 16-QAM and 32-QAM, while offering similar performance on the AWGN channel.

2.4.2. Symbol Mappings

Figures 2.2 to 2.5 show the symbol mappings for the according modulation schemes specified for DVB-S2. In 16-APSK and 32-APSK the circle ratios γ_i are dependent of the used code rate. Further details can be found in sections 5.4.3 and 5.4.4 of [ETS09].

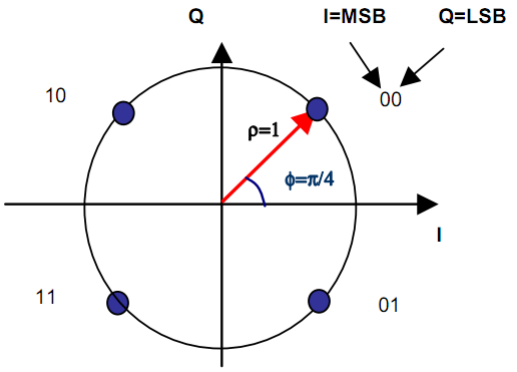

Figure 2.2.: QPSK symbol mapping

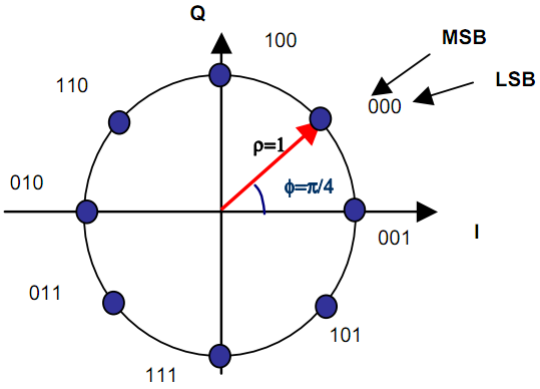

Figure 2.3.: 8-PSK symbol mapping

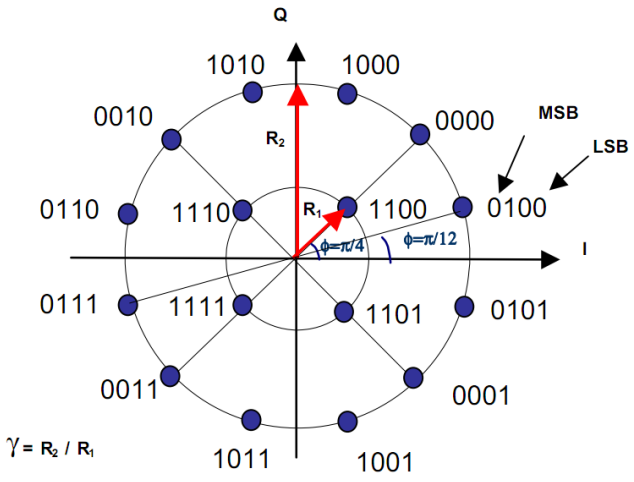

Figure 2.4.: 16-APSK symbol mapping

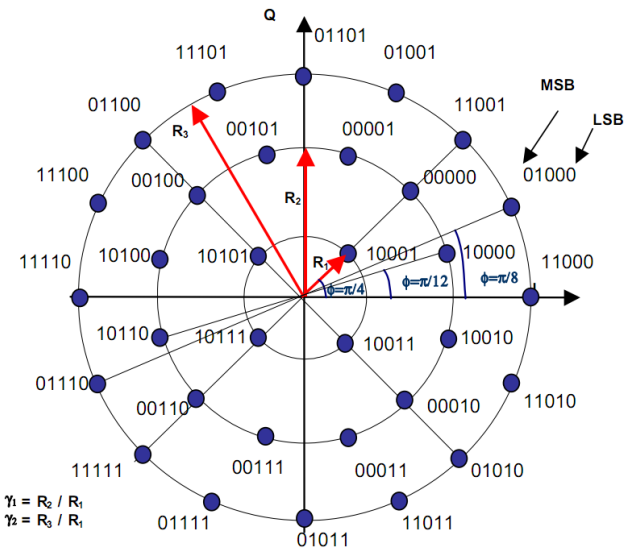

Figure 2.5.: 32-APSK symbol mapping

2.5. PL Framing

2.5.1. Physical Layer Frame Format

DVB-S2 uses the PLFRAME structure of figure 2.6. First there is a PLHEADER which consists of a Start Of Frame (SOF) and a Physical Layer Signaling Code (PLSC). The SOF is a unique word (0x18D2E82) with 26 symbols. The PLSC is a non-systematic binary code of length 64 and dimension 7 with minimum distance 32 (Reed-Muller code). It contains the coded information for the physical layer signaling which consists of:

- MODCOD (5 bits): identifies the used modulation scheme and the code rate.
- TYPE (2 bits): identifies the length of the FECFRAME (normal or short frame) and whether pilots are present or not.

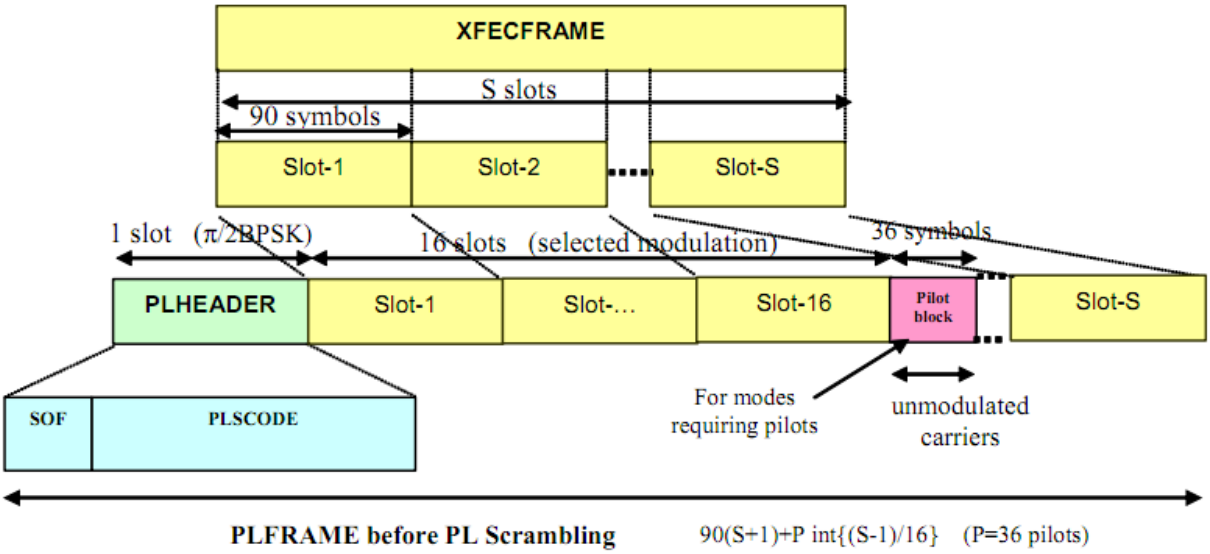

Figure 2.6.: PLFRAME format [ETS09]

2.5.2. Generating PLS Code

The MODCOD and TYPE field are bi-orthogonally coded with a $(64, 7)$ Reed-Muller code. This code is generated by first calculating a bi-orthogonal $(32, 6)$ code and then using an XOR bit operation to get the final code. In more detail, we have that the MODCOD and TYPE field are given as:

$$\left(\underbrace{b_1, b_2, b_3, b_4, b_5}_{MODCOD}, \underbrace{b_6, b_7}_{TYPE}\right)$$

Then we can get the (32,6) code by multiplying the first 6 data bits (MODCOD + MSB of TYPE) with the generator matrix $\mathbf{G}$ of the (32,6) Reed-Muller code:

$$(y_1, \ldots, y_{32}) = (b_1, \ldots, b_6) \cdot \mathbf{G}$$

where

$$\mathbf{G} = \begin{pmatrix} 01010101010101010101010101010101 \\ 00110011001100110011001100110011 \\ 00001111000011110000111100001111 \\ 00000000111111110000000011111111 \\ 00000000000000001111111111111111 \\ 11111111111111111111111111111111 \end{pmatrix}$$

Afterwards we generate the (64,7) code: First we XOR each bit of the (32,6) code with b_7 yielding another codeword with 32 bit; Then we construct the final codeword by interleaving both subcodes as:

$$(z_1, \ldots, z_{64}) = (y_1, y_1 \otimes b_7, y_2, y_2 \otimes b_7 \ldots, y_{32})$$

Depending on b_7 two consecutive bits y_{2i-1} and y_{2i} (for $i = 1...32$) are now either equal or

inverted. A block diagram of the construction of the whole PLSC is shown in figure 2.7.

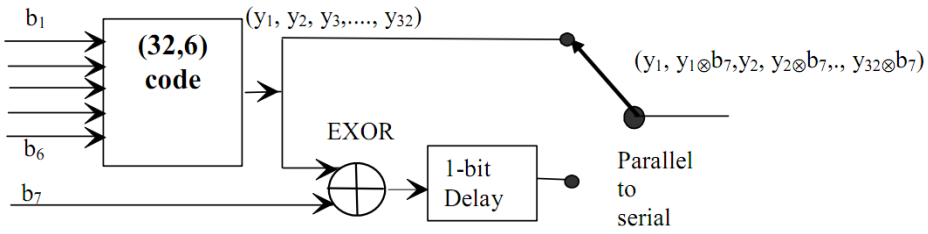

Figure 2.7.: PLFRAME format [ETS09]

The PLSC output bits are then further scrambled (XOR) by a binary scrambling sequence to improve the autocorrelation properties. The scrambling sequence is give by:

$$s = (01110001100111011000001111001001010100110100001000101101111111010)$$

2.5.3. $\frac{\pi}{2}$-BPSK

The PLHEADER data are always modulated by a $\frac{\pi}{2}$-BPSK which is a variant of the BPSK modulation. It has two binary symbol constellations that are 90° degrees rotated against each other. One symbol constellation is chosen when odd data bits are modulated, whereas the other is used when even data bits are modulated. The advantage of this modulation scheme is whereas a reduction in the transmitted signal envelope fluctuation, because there are no symbol transitions through the origin of the I/Q-plane [MM06].

Consider a PLHEADER data sequence represented by the binary sequence $(y_1, y_2, ..., y_{2N})$. The symbol mapping follows the following translation:

$$I_{2i-1} = Q_{2i-1} = \frac{1}{\sqrt{2}}(1 - 2 \cdot y_{2i-1}) \; and \; I_{2i} = -Q_{2i} = \frac{1}{\sqrt{2}}(1 - 2 \cdot y_{2i-1}) \; for \, i = 1, 2, ..., N$$

So for data bits with an odd index the modulation scheme from figure 2.8 is used and for data bits with an even index the modulation scheme from figure 2.9 is used.

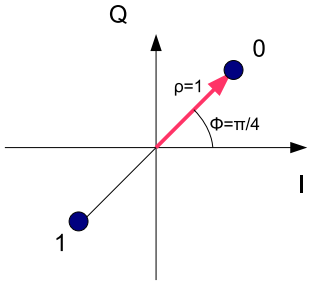

Figure 2.8.: Symbol mapping for odd bits

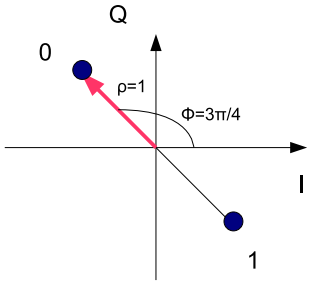

Figure 2.9.: Symbol mapping for even bits

2.5.4. Payload

The payload is transmitted using the modulation scheme of the chosen MODCOD. The FECFRAME is divided into S slots where 1 slot consists of 90 symbols. The number S depends on the spectral efficiency of the chosen modulation and of the TYPE field.

2.5.5. Pilots

When pilots are turned on in the TYPE field, a pilot block of 36 symbols will be inserted after each 16 slots of payload data. The pilot symbols are represented by the following constellation:

$$I_i = Q_i = \frac{1}{\sqrt{2}}$$

If a pilot block coincides with the beginning of the next frame then no pilot block will be inserted.

2.5.6. DUMMY PLFRAME

Dummy frames can be inserted if there is no useful data ready to be sent on the channel. The Dummy PLFRAME consists of a PLHEADER and 36 slots of un-modulated symbols:

$$I_i = Q_i = \frac{1}{\sqrt{2}}$$

2.5.7. PL Scrambling

The payload data (including the pilots) is additionally scrambled by multiplying the symbols by a complex randomization sequence. The complex scrambling sequence is always reset at the end of each PLHEADER. Further details about the structure of the scrambling sequence are given in the standard.

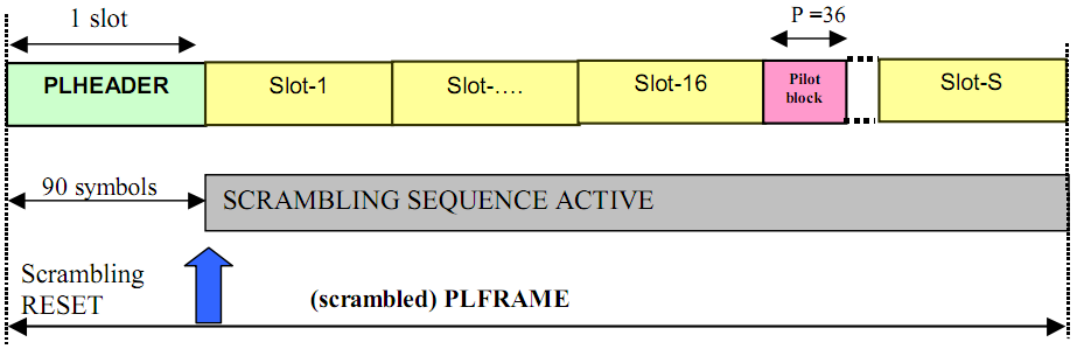

Figure 2.10.: PL scrambling [ETS09]

2.6. Baseband Shaping and Quadrature Modulation

The baseband shaping should be done by a root-raised cosine pulse. The rolloff factor α of the pulse can be chosen out of 3 values: $\{0.35, 0.30, 0.2\}$. After baseband shaping the quadrature modulation into the passband is performed.

2.7. Support of Adaptive Coded Modulation (ACM)

ACM is a special feature of DVB-S2. It allows an optimization of the transmission parameters with respect to the current channel conditions. So, if for example the SNR decreases on the link, DVB-S2 allows to change the MODCOD of each frame. Since the available transponder bandwidth is constant, the DVB-S2 modulator will operate at a fixed symbol rate. To maintain the service continuity during signal fades (e.g.: rain fades as shown in figure 2.11) the information rate (user bits) needs to be reduced by reducing the coding rate and/or the modulation order [MR04].

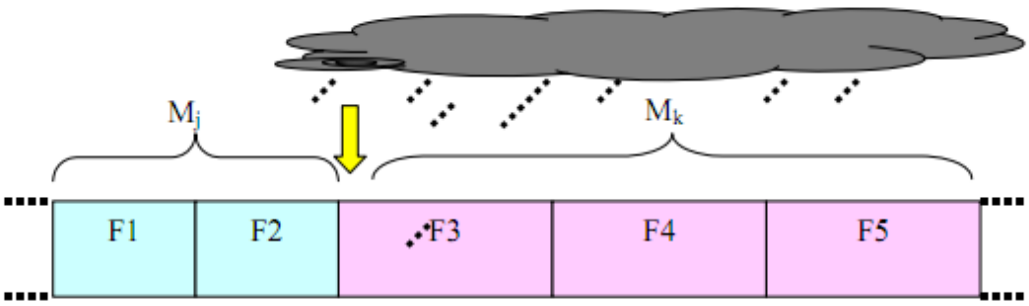

Figure 2.11.: PL frames changing protection during a rain fading [ETS09]

2.8. Receiver Block Diagram (ETSI)

An overview over a proposed receiver block diagram for the physical layer (figure 2.12) can be found in the DVB-S2 user guidelines [ETS05]. In this Thesis only certain aspects of

this receiver have been considered. The implemented parts include the frame synchronization, PLSC decoding, phase synchronization and additionally SNR estimation. Symbol clock recovery and digital interpolation are investigated in a companion thesis [Bis12].

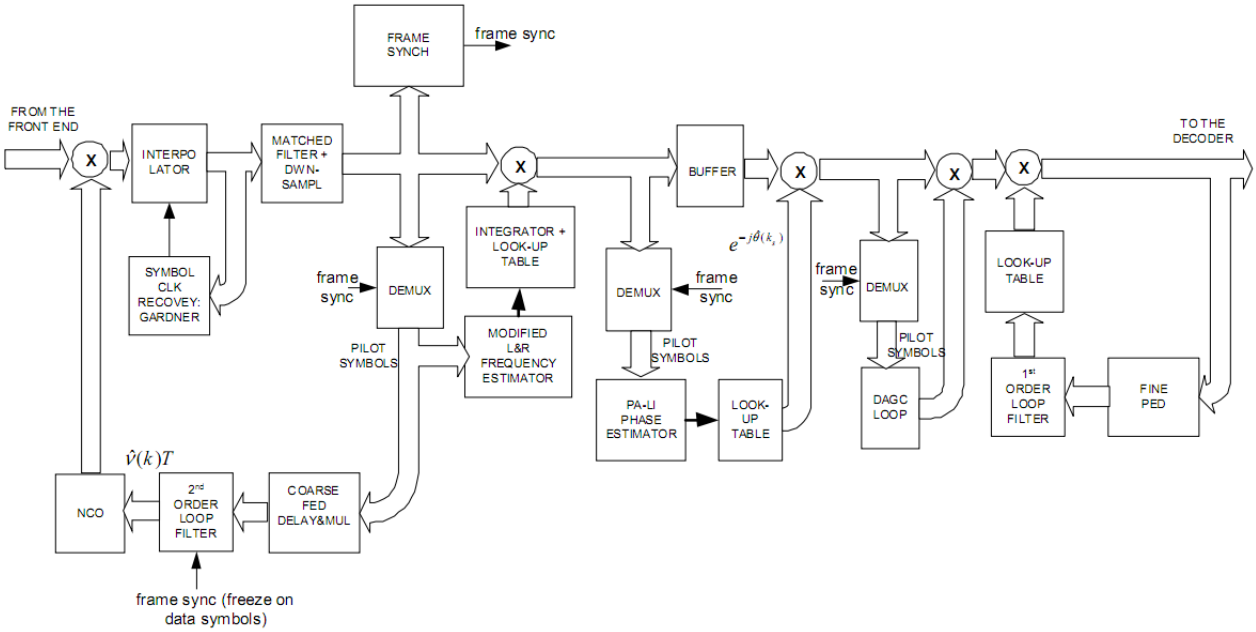

Figure 2.12.: ETSI proposed receiver structure [ETS05]

3. Frame Synchronization Theory

3.1. Introduction

Since DVB-S2 uses a frame-based transmission format, a synchronization is necessary to find the beginning of a frame. This is on the one hand required for both PLSC Decoding and on the other hand for DA SNR estimation, to apply our algorithms sucessfully. The algorithm should be resistant against channel impairments and acquire a synchronized state as fast as possible. Also the algorithm has to be smart enough to cope efficiently with all possible frame formats (e.g. after a MODCOD switch).
DVB-S2 sends its data in a constant stream format (no bursts). Hence it is possible to find the beginning of the next frame out of the decoded PLSC of the current frame. This holds as long as the symbol timing loop is locked and no PLSC decoding error occurs (simulations showed that the decoding works quasi-error-free). This means that the frame sync can be maintained once it was acquired. Therefore it is just required to find an initial frame synchronization. [SJL04]

3.1.1. Signal Model

Under the assumption that the symbol timing is correct, the received baseband signal r_k consists of the sent symbols s_k rotated by a carrier frequency offset Δf_c and a carrier phase offset θ and additive noise samples n_k:

$$r_k = s_k e^{j(2\pi k \Delta f_c T_s + \theta)} + n_k \tag{3.1}$$

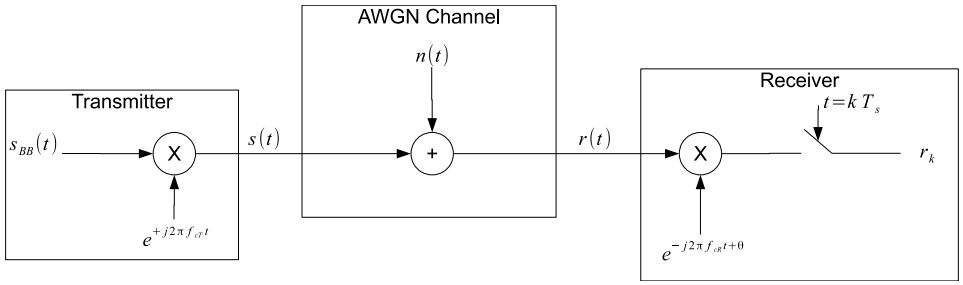

Figure 3.1.: Signal model

3.1.2. Carrier frequency and phase offsets

The carrier frequency offset distorts the received signal. Reasons for a carrier frequency offset $\Delta f_c = f_{c_T} - f_{c_R}$ (f_{c_T}: transmitter frequency; f_{c_R}: receiver frequency) are as follows:

- Mismatch between the transmitter carrier frequency oscillator and the receiver frequency oscillator.

- Doppler frequency shifts due to relative motions between receiver and transmitter

This means that the received symbols rotate away from their ideal constellation points over time.
Carrier phase offsets are also present and as a result the symbol constellations are rotated by a fixed angle.

3.1.3. Noise

Another reason for signal distortions in the receiver is thermal noise which is always present in communication systems. The noise samples n_k are assumed to be zero mean Gaussian distributed with a variance of $\frac{N_0}{2}$ in the I/Q components.

$$\begin{aligned} \mathrm{Re}\{n_k\} &\sim N\left(\mu{=}0, \sigma^2 = \frac{N_0}{2}\right) \\ \mathrm{Im}\{n_k\} &\sim N\left(\mu{=}0, \sigma^2 = \frac{N_0}{2}\right) \end{aligned}$$

3.2. Frame Timing Recovery

A conventional approach to determine the frame timing is to first calculate correlation results followed by a peak detection to analyze the correlation results (see figure 3.2).
The correlation algorithm should deliver high peak results if a certain pattern (unique word) inside the input stream occurs. Unique words typically represent the beginning of a frame. In DVB-S2 the SOF sequence (which represents a unique word) is sent at the beginning of each frame.
The second stage (peak detector) determines the beginning of a frame. This is done by using the correlation results and, optionally, the input symbols. If the peak detector algorithm finds enough evidence that there could be a framestart, we can declare a successful synchronization and tell the following modules the related positions in the symbol stream.

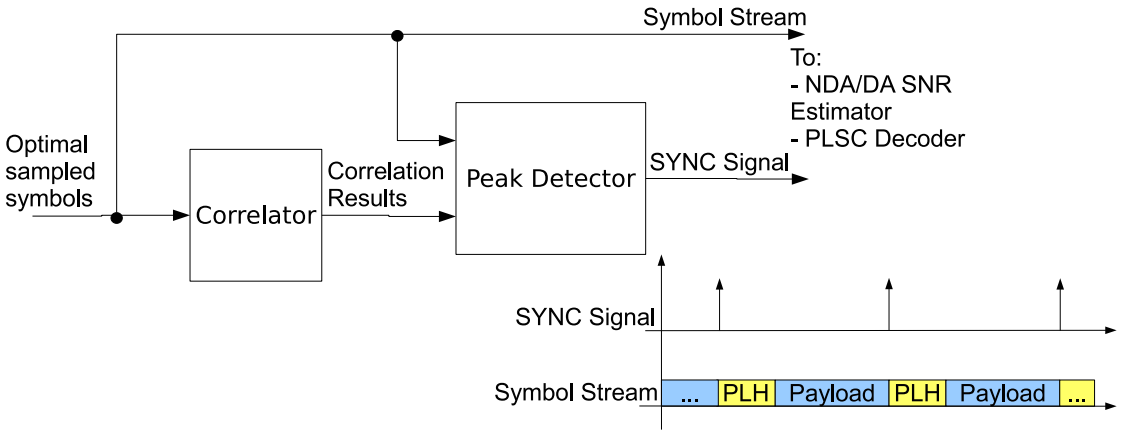

Figure 3.2.: Combination of correlator and peak detector to determine the frame timing (SOF=Start-of -Frame)

3.2.1. Correlation

A short overview of different possibilities to get correlation results is presented in this section. The goal is to find an algorithm that copes best with the channel impairments but is also simple enough in terms of complexity to be used in a hardware FPGA implementation.
The discussion about the correlation algorithms is followed by the application of the chosen correlation algorithm in DVB-S2.

3.2.1.1. Conventional Correlation

This algorithm simply takes the received symbols and multiplies it with the expected complex conjugated symbol of the unique word and sums this over the known sequence length:

$$C_{normal}(\mu) = \left| \sum_{k=0}^{N_H-1} r_{k+\mu} \cdot s_k^* \right| = \left| \sum_{k=0}^{N_H-1} c_{k+\mu} \right|$$

Now we look what happens if we correlate exactly at the SOF position ($\mu = 0$). We insert the signal model for the received signal from (3.1) and neglect the noise term n_k:

$$c_k = r_k \cdot s_k^* = s_k \cdot e^{j(2\pi k \Delta f_c T_s + \theta)} \cdot s_k^* = |s_k|^2 e^{j(2\pi k \Delta f_c T_s + \theta)}$$

Inserting back into the correlation sum we get:

$$C_{normal}(0) = \left| \sum_{k=0}^{N_H-1} r_k \cdot s_k^* \right| = \left| \sum_{k=0}^{N_H-1} |s_k|^2 e^{jk2\pi \Delta f_c T_s + \theta} \right| \tag{3.2}$$

From (3.2) it can be seen that the correlation sum is quite sensitive to carrier frequency offsets Δf_c even when correlating exactly at the SOF and no noise is present. This is because the exponential term remains inside the sum. This may lead to unreliable correlation results.

3.2.1.2. Choi-Lee Detector

Choi and Lee proposed frame synchronization techniques in the presence of frequency offsets [CL02]. This algorithms are based on an approximate Maximum Likelihood (ML) criterion. These algorithms can improve the performance of the ML rule significantly in the presence of a large frequency offset. For a short overview the equations to calculate the L_0 to L_3 estimator are given here. Further details about performance can be found in the paper cited above.

$$L_0(\mu) = \sum_{i=1}^{L-1} \left\{ \left| \sum_{k=i}^{L-1} r_{\mu+k}^* s_k r_{\mu+k-i} s_{k-i}^* \right|^2 - \sum_{k=\mu+i}^{\mu+L-1} |r_k|^2 \, |r_{k-i}|^2 \right\}$$

$$L_1(\mu) = \sum_{i=1}^{L-1} \left\{ \left| \sum_{k=i}^{L-1} r_{\mu+k}^* s_k r_{\mu+k-i} s_{k-i}^* \right| - \sum_{k=\mu+i}^{\mu+L-1} |r_k| \, |r_{k-i}| \right\}$$

$$L_2(\mu) = \left| \sum_{k=1}^{L-1} r_{\mu+k}^* s_k r_{\mu+k-1} s_{k-1}^* \right| - \sum_{k=\mu+1}^{\mu+L-1} |r_k| \, |r_{k-1}|$$

$$L_3(\mu) = \left| \sum_{k=1}^{L-1} r_{\mu+k} s_k^* r_{\mu+k-1}^* s_{k-1} \right|$$

3.2.1.3. Differential Correlation

The differential correlation is in fact the proposed Choi-Lee detector $L_3(\mu)$ [CL02]. The idea behind differential correlation is not to take a sample-wise correlation but to take a correlation on a pairwise differential.

$$C_{diff}(\mu) = \left| \sum_{k=0}^{N_H-2} r_{k+\mu} \cdot r_{k+\mu+1}^* \cdot \left(s_k \cdot s_{k+1}^* \right)^* \right| = \left| \sum_{k=0}^{N_H-2} c_{k+\mu} \right|$$

Again we look what happens when the correlation is performed at the SOF position ($\mu = 0$). If we insert the signal model for the received signal from (3.1) and neglect the noise sample n_k the pairwise differential term gets:

$$r_k = s_k e^{j(2\pi k \Delta f_c T_s + \theta)}$$

$$r_{k+1} = s_{k+1} e^{j(2\pi (k+1) \Delta f_c T_s + \theta)}$$

$$r_k \cdot r_{k+1}^* = s_k e^{j(2\pi k \Delta f_c T_s + \theta)} \cdot s_{k+1}^* e^{-j(2\pi (k+1) \Delta f_c T_s + \theta)} = s_k \cdot s_{k+1}^* \cdot e^{-j2\pi \Delta f_c T_s}$$

By correlating the differentials with the unique word differentials we get:

$$\begin{aligned} C_{diff}(0) &= \left| \sum_{k=0}^{N_H-2} r_k \cdot r_{k+1}^* \cdot \left(s_k \cdot s_{k+1}^* \right)^* \right| \\ &= \left| \sum_{k=0}^{N_H-2} s_k \cdot s_{k+1}^* \cdot e^{-j2\pi \Delta f_c T_s} \cdot s_k^* \cdot s_{k+1} \right| \\ &= \left| \sum_{k=0}^{N_H-2} |s_k|^2 \cdot |s_{k+1}|^2 e^{-j2\pi \Delta f_c T_s} \right| = \underbrace{\left| e^{-j2\pi \Delta f_c T_s} \right|}_{1} \left| \sum_{k=0}^{N_H-2} |s_k|^2 \cdot |s_{k+1}|^2 \right| \end{aligned} \tag{3.3}$$

A great advantage of this technique is that, on the one hand, the phase offset term completely vanishes from the correlation sum and, on the other hand, the exponential term with the carrier frequency offset does not affect the sum since the index k drops out because of the pairwise differential.

3.2.1.4. Application of Differential Correlation in DVB-S2

The phase and frequency offsets are not corrected before frame synchronization. Therefore we need a correlation mechanism that copes with both impairments. As already stated in section 3.2.1.3 we can use the differential correlation to overcome these problems with an acceptable complexity especially with regard towards hardware implementations.
As stated in [MM06] correlating only on the 26 SOF symbols has been investigated but leads to a bad performance in the presence of low SNRs. [SJL04] describes an algorithm that uses the whole PLHEADER (90 symbols) for differential correlation on a symbol-by-symbol basis. The following points are important to understand the algorithm:

- Since $\frac{\pi}{2}$-BPSK is used in the PLHEADER, each differential $s_k \cdot s_{k+1}^*$ either takes the value of $\pm j$. Therefore one tap can either take the $\pm j$.
- The SOF symbols are always the same so we can use every differential $s_k \cdot s_{k+1}^*$ for the differential correlation.
- Different modulations and code rates can be set in the PLSC. Obviously this may lead to very different taps. Extensive researches in papers [ZCF+10, QXC+08] have shown that the PLSC part of the PLHEADER can also be used for the differential correlation despite the fact that it does not have a fixed value. After the scrambling of all different PLSC an evaluation of the possible differential taps with MATLAB has shown that:
  - Not all possible differential taps can be used for correlation since only 32 of the 64 differentials are known.
  - There are only two possible PLSC correlation word symbol sequences where one is the negative of the other. This means that it is possible that we receive a big negative peak if we receive the opposite sequence of the taps settings.
  - This circumstance can be exploited by the circuit if we additionally determine the difference of the SOF correlation result and the PLSC correlation result. Since this will yield to yet another high correlation value if there is a big negative value in the PLSC correlation term. This is a massive improvement in terms of speed since we do not have to change the PLSC taps until we find the right ones. Later we can choose between the maximum of the sum or the difference.

The circuit has two parts. The right part correlates on the SOF symbols and the left part is associated with the PLSC. The following equations are relevant in this context:

$$\begin{aligned} C_{SOF} &= \sum_{k=0}^{N_{SOF}-1} r_k \cdot r_{k+1}^* \cdot tap_{SOF}[k] \\ C_{PLSC} &= \sum_{k=0}^{N_{PLSC}/2-1} r_{2k} \cdot r_{2k+1}^* \cdot tap_{PLSC}[2k] \\ C &= \max\left(|C_{SOF} + C_{PLSC}|, |C_{SOF} - C_{PLSC}|\right) \end{aligned} \tag{3.4}$$

The taps can be determined by using the circuit in figure 3.3 in the following way: First all registers in the delay line are set to zero. Then shift the $\frac{\pi}{2}$-BPSK modulated SOF and any

scrambled and modulated PLSC codeword into the circuit. This whole PL-Header symbol sequence can be described:

$$s = \left(\underbrace{s_1, s_2, \ldots, s_{N_{SOF}}}_{SOF\ Symbols}, \underbrace{s_{N_{SOF}+1} \ldots, s_{N_{SOF}+N_{PLSC}}}_{PLSC\ Symbols} \right)$$

When the register associated to $tap_{SOF}[0]$ becomes nonzero, all taps can be determined by simply taking the complex conjugate of the associated registers. This leads to the following mathematical description to determine the taps:

$$\begin{aligned} tap_{SOF}[k] &= \left(s_k \cdot s_{k+1}^*\right)^* \\ tap_{PLSC}[k] &= \left(s_{2k+N_{SOF}} \cdot s_{2k+N_{SOF}+1}^*\right)^* \end{aligned}$$

While the SOF taps sequence is fixed, the PLSC taps can only take two different sequences where one sequence is the negative of the other.

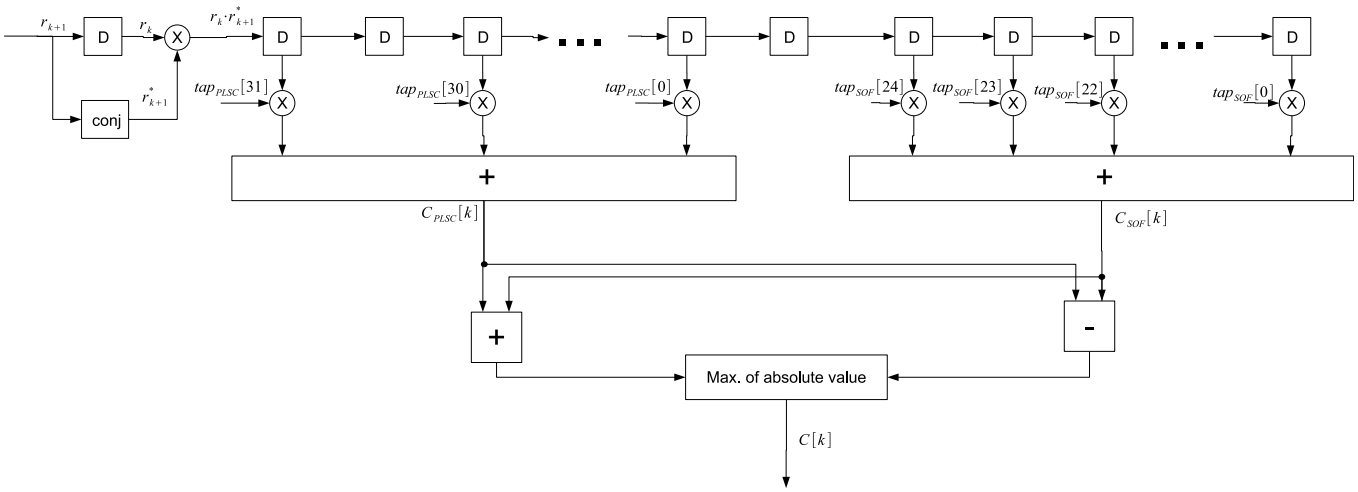

Figure 3.3.: Circuit to perform the differential correlation

3.2.2. Peak Detection

After performing the correlation algorithm, a detection of the maximum peak position is necessary to find the correct beginning of the frame. Especially when there is a lot of noise added to the received signal (e.g.: $\frac{E_s}{N_0} = -2dB$) these peaks may occur at the wrong position as it can be seen in section 6.1. The algorithm has to be smart enough to handle with these low SNRs and also acquire the SYNC status as fast as possible.

3.2.2.1. Threshold Detector

Obviously the most intuitive method is to define an absolute threshold value and look for correlation results that exceed this threshold. While being relatively simple to implement this method has its major drawback in a great sensitivity towards the threshold. So a received signal fading effect may lead to missed or false detections because the threshold is never exceeded or exceeded too often.

3.2.2.2. Exponential Averaging + Threshold Detector

A better method is to calculate an average over the correlation results to get a kind of average signal power. If the current correlation result rises above this average signal power by a relative amount (e.g: 150% over the average signal power) then there is the chance to have a SOF at this position.
While this method may work well at high SNRs it becomes problematic at lower SNRs. The reason for this is that the overall power in comparison to the peak values rises. So the peaks may not rise significantly over the average signal power and this will lead to missed detections. Another drawback is that there is some freedom in the practical implementation: on the one hand the optimal threshold β and on the other hand an optimal averaging parameter with the weighting factor α have to be found. The averaging parameter determines how "fast" the detector can react to a fading signal.
The exponential averager is a simple and effective way to compute a signal average [Lyo10]which is given by the following recursive equation:

$$C_{avg}[n] = \alpha \cdot C[n] + (1 - \alpha) \cdot C_{avg}[n-1]$$

This leads to the following transfer function in the z- domain:

$$H(z) = \frac{\alpha}{1 - (1-\alpha)z^{-1}}$$

For the detector we need to determine if the current signal rises above the threshold and declare a successful SYNC if this is the case:

$$SYNC := \begin{cases} 1 & C[n] \geq (1+\beta)C_{avg}[n] \\ 0 & else \end{cases}$$

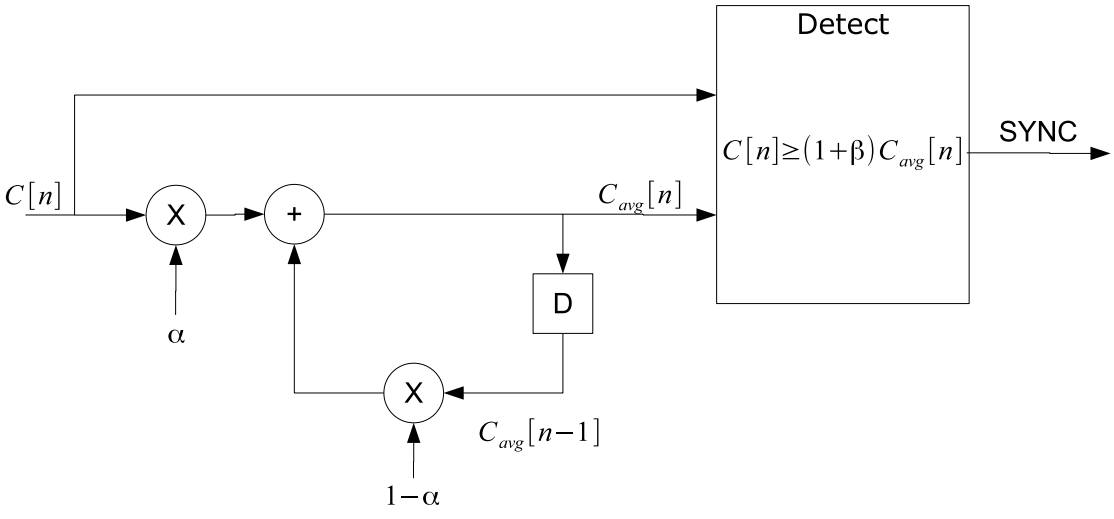

Figure 3.4.: Exponential averaging and threshold detector

The benefit from this method illustrated in figure (3.4) is a very simple implementation which perfectly suits for a digital circuit. Further simplifications are possible if α takes certain values (see [Lyo10, p. 790]) to get a multiplier-free implementation. This method has been implemented but simulations showed it to be too sensitive towards SNR changes. This means

that the optimal values for α and β, where a good relation between missed detections and false detection is given, depends strongly of the current SNR. Therefore it was dropped in favour of the adapted peak search detector.

3.2.2.3. Modified Peak Search Detector

The following algorithm is proposed in [QXC+08]. This algorithm uses more than one peak search window i.e. one for each of the four different frame lengths (if only considering one frame mode (short/long frames , with or without pilots)). In each window the two greatest peaks are recorded. If the peaks represent a possible SOF, then it should repeat in the following search window. By determining the distance (in number of symbols) between the peaks in consecutive windows and comparing them to the known frame lengths. By further comparing the peaks between different search windows, we can determine if there is repeating pattern in consecutive windows. If such a pattern occurs, there is a high likelihood that we found the correct framelength and we can declare initial frame synchronization.

One advantage of this scheme is that it does not need to decode the PLSC to find the framelength, this saves some processing power. Another important thing to keep in mind is that in conventional approaches the decoding of the PLSC is very hard to perform if there is a high carrier frequency offset.

An obvious drawback of this solution is that it only works for transmissions in which MODCODs are not changed on a frame-by-frame basis since the initial synchronization relies on a repetitive pattern of the peaks. Therefore it is only suited for CCM (Constant Code and Modulation) transmission which are often used in Direct-to-Home TV broadcasting applications. Since we use ACM in this project the algorithm can not be used.

3.2.2.4. Peak Search Detector

This peak search algorithm is proposed in [SJL04]. It finds the maximum peak within a search window and declares it as candidate (see figure 3.5).

The window should be as small as possible but long enough to contain at least one SOF. Hence the search window needs to be 33282 symbols (QPSK normal frames with pilots) long. The PLSC at the candidate location is now decoded. Based on the decoded PLSC it gets the location of the next PLHEADER. If there is a a sufficient strong correlation at this position a successful INIT SYNC is declared. If this is not the case then the algorithm will check the next candidate.

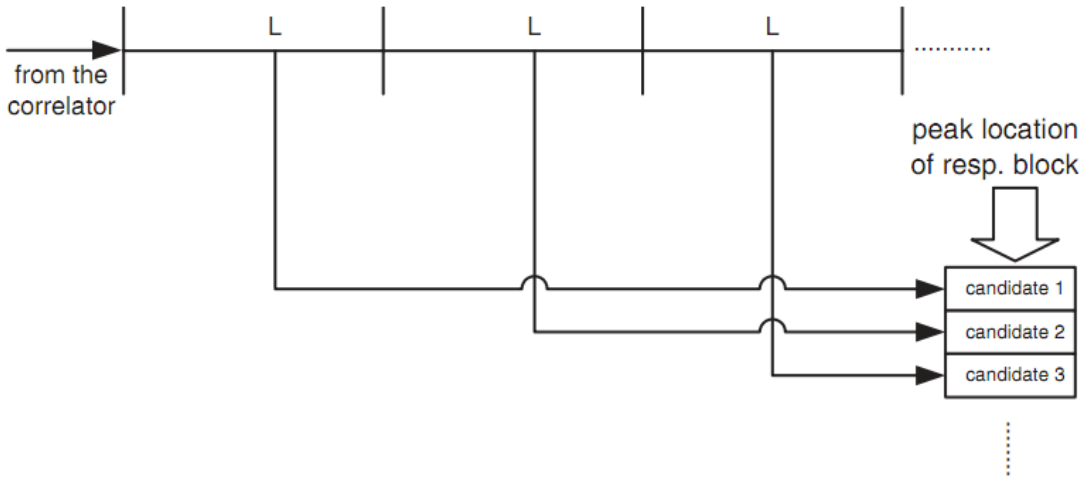

Figure 3.5.: Peak search algorithm (from [SJL04])

3.2.2.5. Adapted Peak Search Detector

The idea behind this algorithm is the same as with the the peak search detector mentioned before but looks for more than one peak within a search window. Also the post verification looks out for more then two high correlation results. The adapted algorithm works as follows:

- The big search window is divided into a certain amount (M) of small windows
- In each of the small windows the N highest peaks are determined (correlation results with the highest values) (see figure (3.6))
  - The position within the big window of each of the peaks is recorded in a table
  - The PLSC at each peak position is decoded and the framelength determined
  - The next peak position (=peak position + decoded framelength) is also recoded in the table
- When the big search window passed, a post processing algorithm checks if there is a path through the table (see figure 3.7):
  - Therefore it checks if there are matching peaks in the table
  - If there is a certain amount of L consecutive peaks matching we declare a successful frame synchronization at the first peak of the path

The following properties have to be considered for choosing the window lengths:

- A small window can only be as long as the minimal framelength (=3330 Symbols if DUMMY Frames are use), otherwise it may be that the right peak of the next frame is not recorded
- The big search window must be at least as long as L (the number of consecutive peaks) times the maximum framelength (=33282 Symbols), since it has to be guaranteed that at least L full framelengths pass so that each SOF of one frame can yield to a peak

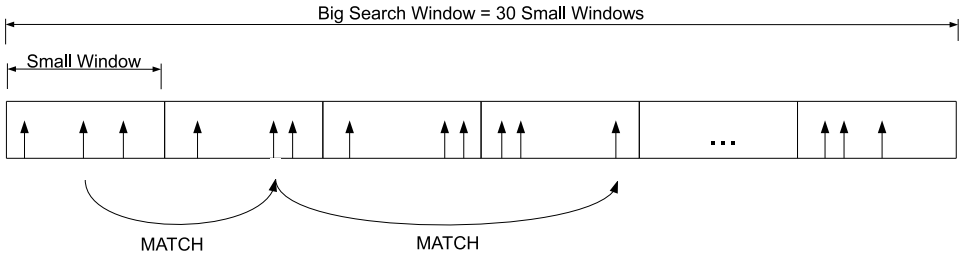

Figure 3.6.: Adapted peak search algorithm

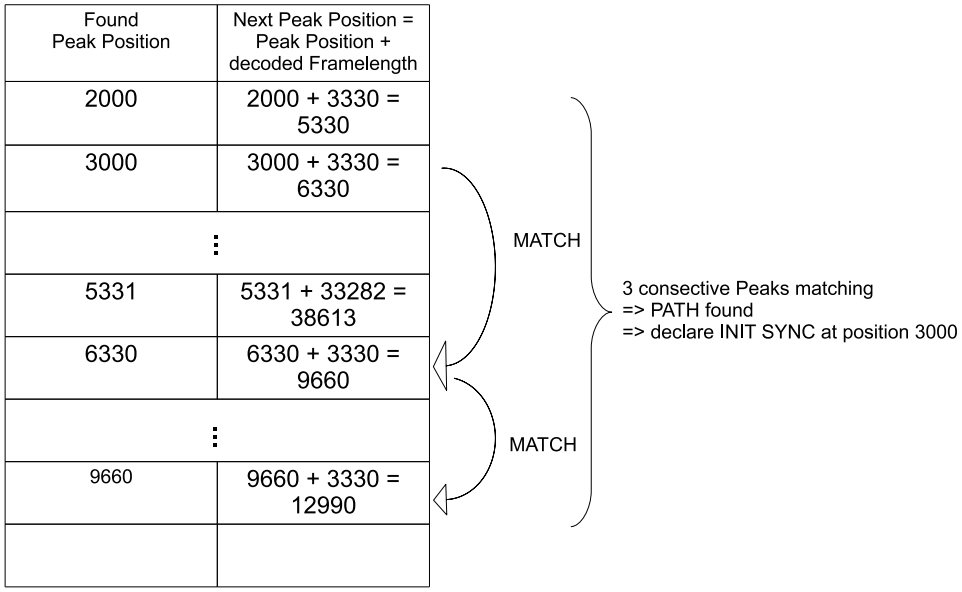

Figure 3.7.: Example for the path search in the peak table

3.3. PLSC Decoding

To get the MODCOD out of the received PLSC symbols we first need the sent binary stream out of the sent symbol sequence. Then we need to decode the PLSC out of this binary stream.

3.3.1. Correction of Carrier Phase Offsets

Since the carrier phase offset of the signal model in (3.1) rotates the symbol constellations by a fixed angle we first need to estimate and correct the carrier phase offset in the received symbols before decoding. This can be done by a DA correlation between the received symbols $r_{k_{rec}}$ with the known SOF symbols p_k. This leads to the following ML estimate (see [MD97]):

$$\hat{\theta}_{DA} = arg\left\{\sum_{k=1}^{N_{SOF}} r_{k_{rec}} \cdot p_k^*\right\}$$

After the estimate has been calculated all PLSC symbols can be corrected im phase by using the phase estimate gained from the SOF symbols. Under the assumption that the phase offset for the PLSC symbols is the same as for the SOF symbols (no frequency offsets), we can correct the PLSC symbols by multiplying them with the complex exponential:

$$r_k = r_{k_{rec}} \cdot e^{-j\hat{\theta}_{DA}}$$

Generally, the modified Cramer-Rao Lower Bound (MCRLB) for this kind of phase estimation is given by:

$$MCRLB = \frac{1}{2 \cdot L_{N_{SOF}} \cdot \frac{E_s}{N_0}}$$

3.3.2. Correction of Carrier Frequency Offsets

For tiny frequency offsets ($\Delta f_c T_s < 0.001$) the phase offset estimation and correction on small blocks like the 90 symbols of the PLHEADER is sufficient but leads to a biased estimation of the phase estimate (see [MD97, p. 198]).
For large carrier frequency offsets better algorithms have to be used (e.g. application of the Luise & Reggiannini algorithm in DVB-S2 [SJL04]).

Another option that has been tried out was to determine a frequency estimate out of the differential correlation sum (see [MMF98, p. 487]). When the sum is calculated exactly at the beginning of the frame μ_{opt} (see equation 3.3) we can determine the estimate :

$$\Delta f_c T_s = \frac{1}{2\pi} arg \left\{ \left| \sum_{k=0}^{N_H-2} r_{k+\mu} \cdot r^*_{k+\mu+1} \cdot \left(s_k \cdot s^*_{k+1}\right)^* \right| \right\}_{\mu=\mu_{opt}}$$

This kind of estimator has been implemented but turned out to be not good enough. The jitter variance is quite high at low SNRs (< 5dB) and therefore it is not suited for application in DVB-S2.

3.3.3. Symbol Decoding using Hard Decisions

This method uses the absolute symbol positions from the $\frac{\pi}{2}$-BPSK symbol constellations in section 2.5.3 to decode the information bits from the received symbols. When a potential start of a PLSC has been found we can decode by first rotating the received odd symbols by $-\frac{\pi}{4}$ and even symbols by $-\frac{3\pi}{4}$. Than we can make a hard decision between 0 and 1 for bit b_k by checking the real part of the rotated signal leading to the following decision rules for odd data symbols:

$$\mathrm{Re}\left\{r_k \cdot e^{-j\frac{\pi}{4}}\right\} \geq 0 \Longrightarrow decide\ for\ 0 \tag{3.5}$$

$$\mathrm{Re}\left\{r_k \cdot e^{-j\frac{\pi}{4}}\right\} < 0 \Longrightarrow decide\ for\ 1 \tag{3.6}$$

The calculation can be further simplified by analyzing the structure of the left part of the inequation:

$$\begin{aligned} \mathrm{Re}\left\{r_k \cdot e^{-j\frac{\pi}{4}}\right\} &= \mathrm{Re}\left\{\left(\mathrm{Re}\{r_k\} + j \cdot \mathrm{Im}\{r_k\}\right) e^{-j\frac{\pi}{4}}\right\} \\ &= \left(\frac{\sqrt{2}}{2}\right)\left(\mathrm{Re}\{r_k\} + \mathrm{Im}\{r_k\}\right) \end{aligned}$$

Since the factor $\left(\frac{\sqrt{2}}{2}\right)$ can be left out of (3.5)-(3.6) we can define the following simplified decision rules for odd data symbols:

$$\begin{aligned} \mathrm{Re}\{r_k\} + \mathrm{Im}\{r_k\} &\geq 0 \Longrightarrow decide\, for\, 0\, (b_k = 0) \\ \mathrm{Re}\{r_k\} + \mathrm{Im}\{r_k\} &< 0 \Longrightarrow decide\, for\, 1\, (b_k = 1) \end{aligned}$$

In a similar way we can determine the decision rules for even data symbols leading to:

$$\begin{aligned} \mathrm{Im}\{r_k\} - \mathrm{Re}\{r_k\} &\geq 0 \Longrightarrow decide\, for\, 0\, (b_k = 0) \\ \mathrm{Im}\{r_k\} - \mathrm{Re}\{r_k\} &< 0 \Longrightarrow decide\, for\, 1\, (b_k = 1) \end{aligned}$$

The new decision rules reduce the implementation overhead since rotations are not necessary. The received PLSC sequence consists now of the hard decisions for each of the bits:

$$PLSC_{rec} = (b_{N_{SOF}+1}, ..., b_{N_{SOF}+N_{PLSC}})$$

3.3.4. PLSC Decoding of Hard Decisions

3.3.4.1. Maximum Likelihood Method

The idea behind behind this decoding algorithm is really simple. We take our received codeword $PLSC_{rec}$ and form the Hamming distance to each of the 2^7 possible codewords $PLSC_i^{(k)}$. The Hamming distance is calculated by counting the bits that are different in both codewords (essentially a XOR operation in digital hardware). Then we choose the code k that yields to the minimal Hamming distance:

$$PLS = \arg\min_k \left\{ \sum_{i=0}^{NPLSC-1} PLSC_{rec\,i} \oplus PLSC_i^{(k)} \right\}$$

Since there are only $2^7 = 128$ possible codewords for the PLSC this method works quite efficiently.

3.3.4.2. Fast Hadamard Transformation

Another option is to decode the PLSC by a transformation. Since the PLSC is an interleaved first-order Reed-Muller code with a length of 64, deinterleaving leads to a first-order Reed-Muller codeword. This can be decoded by using the Fast Hadamard Transformation (FHT) [SJL04]. Further information about the FHT can be found in [KLH+11].

4. SNR Estimation Theory

4.1. Introduction

SNR estimation is an important task in digital communication systems. When using ACM we should have reliable estimates to detect when the current SNR is decreasing and therefore an adaptation of the link is necessary (e.g. MODCOD switch). DVB-S2 can change the modulation scheme on a frame-by-frame basis. Therefore our aim is to estimate one SNR per frame.
The DVB-S2 signal includes both known sequences and also unknown data (payload). Therefore we can perform two estimations: a Data-Aided (DA) estimation on the known data and a blind Non-Data-Aided (NDA) SNR estimation on payload data.
Figure4.1 shows the principal operation of the SNR estimation. The DA estimator operates on the PLHEADER (including SOF and PLSC after decoding) and on the pilot blocks if available. The NDA estimator may operate on the whole frame because we do not need to know any sent symbol sequences for using this estimator.

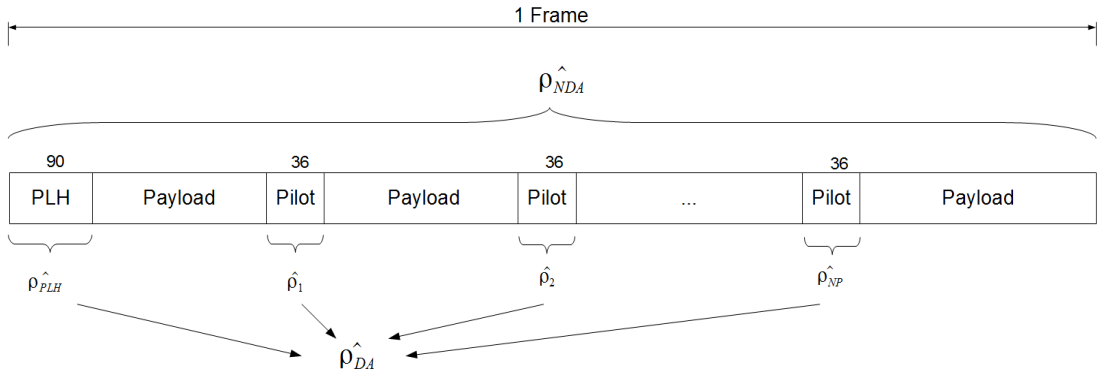

Figure 4.1.: Block diagram for SNR estimation

4.1.1. Signal Model

Perfect symbol timing recovery and perfect carrier recovery are assumed to be established by former stages of the receiver chain. The following signal model is used for all further derivations. The symbol c_k is sent with an average symbol energy of $\sqrt{S}$ and impaired by a carrier phase offset θ so that we receive:

$$r_k = \sqrt{S}s_k + \sqrt{N}w_k, \quad s_k = c_k \cdot e^{j\theta}$$

Additionally there is thermal noise on the channel ($\sqrt{N}w_k$) where both real and imaginary part of the complex noise are assumed to be independent zero mean Gaussian with variance $\frac{1}{2}$:

$$w_{kI} = \mathrm{Re}\{w_k\} \sim \mathrm{N}\left(\mu{=}0, \sigma^2 = \frac{1}{2}\right)$$

$$w_{kQ} = \mathrm{Im}\{w_k\} \sim \mathrm{N}\left(\mu{=}0, \sigma^2 = \frac{1}{2}\right)$$

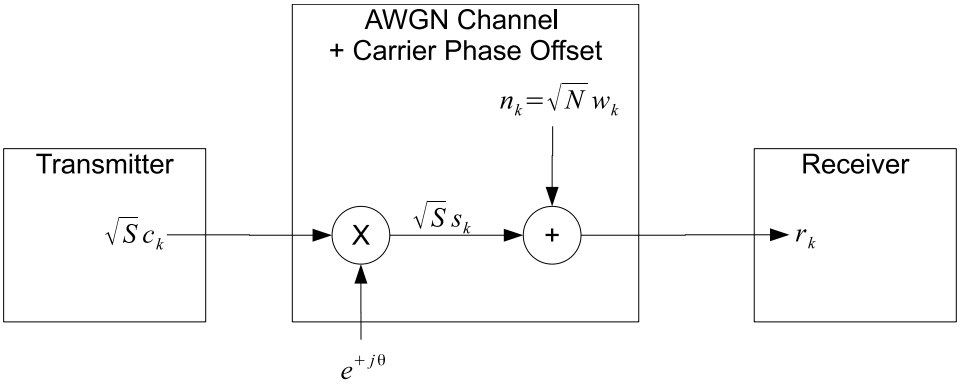

Figure 4.2.: Signal model for SNR estimation

4.2. Classification and Assessment of SNR Estimators

Paper [PB00] was used as background for this chapter. The following estimators are described and compared within this paper:

- Split-Symbol Moments (S&M) estimator
  - In-service estimator that can only be applied to BPSK-Signals in the real AWGN channel
- Maximum-Likelihood (ML) estimator
  - Based on ML estimation theory this estimator is suited for M-PSK signals in complex AWGN channel
  - Uses an oversampled signal (N_{ss} samples per symbol)
- Squared Signal-to-Noise Variance (SNV) estimator
  - Special case of the ML estimator where the optimally sampled output of the matched filter is taken.
  - No oversampling
- Second- and Fourth-order Moments (M_2M_4) estimator
  - NDA estimator that is based on the calculation of signal moments for the complex AWGN channel
- Signal-to-Variation Ratio (SVR) estimator
  - NDA moment-based method intended for multipath fading but it can be applied also for an M-ary PSK signals in complex AWGN channels

For the DA estimation we have chosen the *SNV estimator* since it fits perfectly to our application where we have one sample per symbol (perfect symbol timing assumed) to perform ML Estimation.
For the NDA estimation we have chosen the M_2M_4 *estimator* (with special adaptations for 16-APSK and 32-APSK). The SNR performance comparisons at the end of [PB00] have shown that the M_2M_4 estimator in general outperforms the SVR estimator.

4.2.1. DA Squared Signal-to-Noise Variance Estimation

The SNV Estimation is basically a ML estimation of the SNR under the assumption of optimally sampled output of the matched filter with one sample per symbol [PB00].

4.2.1.1. Derivation of the Algorithm

The derivation is based on [PB00] and uses the signal model for the received samples given in section 4.1.1.
Since both noise components are independent we can write the joint probability density function for the noise samples as:

$$p(n_{kI}, n_{kQ}) = \frac{1}{\pi N} e^{-(n_{kI}^2 + n_{kQ}^2)/N}$$

Therefore the received signal sample has the following pdf:

$$p(r_{kI}, r_{kQ}|S, N) = \frac{1}{\pi N} \exp\left(\frac{\left(r_{kI} - \sqrt{S}s_{kI}\right)^2 + \left(r_{kQ} - \sqrt{S}s_{kQ}\right)^2}{N} \right)$$

For K received samples we get the joint probability density function (samples are assumed to be independent) of the sequence as:

$$p(\mathbf{r_I}, \mathbf{r_Q}|S, N) = \left(\frac{1}{\pi N}\right)^K \exp\left[-\frac{1}{N} \left(\sum_{k=0}^{K-1} \left(r_{kI} - \sqrt{S}s_{kI}\right)^2 + \sum_{k=0}^{K-1} \left(r_{kQ} - \sqrt{S}s_{kQ}\right)^2 \right) \right]$$

This yields to the following log-likelihood function:

$$\begin{aligned} \Gamma(S, N) &= \ln p(\mathbf{r_I}, \mathbf{r_Q}|S, N) \\ &= -K \cdot \ln(\pi N) - \frac{1}{N} \left(\sum_{k=0}^{K-1} \left(r_{kI} - \sqrt{S}s_{kI}\right)^2 + \sum_{k=0}^{K-1} \left(r_{kQ} - \sqrt{S}s_{kQ}\right)^2 \right) \end{aligned}$$

To express the ML estimate of the SNR $\hat{\rho}_{ML}$ we differentiate the log-likelihood function with respect to both S and N and divide both individual ML estimates:

$$\left. \frac{\partial \Gamma(S, N)}{\partial S} \right|_{S=\hat{S}_{ML};\, N=\hat{N}_{ML}} = 0$$

$$\left. \frac{\partial \Gamma(S, N)}{\partial N} \right|_{S=\hat{S}_{ML};\, N=\hat{N}_{ML}} = 0$$

$$\hat{\rho}_{ML} = \frac{\hat{S}_{ML}}{\hat{N}_{ML}} = \frac{\left[\frac{1}{K}\sum_{k=0}^{K-1} \mathrm{Re}\left\{r_k^* s_k\right\}\right]^2}{\frac{1}{K}\sum_{k=0}^{K-1} |r_k|^2 - \left[\frac{1}{K}\sum_{k=0}^{K-1} \mathrm{Re}\left\{r_k^* s_k\right\}\right]^2} \tag{4.1}$$

Studies have shown that the direct use of this estimator leads to a bias [Tho67]. The bias can be reduced by multiplying the denominator with a correction factor. This yields to the following expression for the reduced bias estimator:

$$\hat{\rho}_{ML} = \frac{\left[\frac{1}{K}\sum_{k=0}^{K-1} \mathrm{Re}\left\{r_k^* s_k\right\}\right]^2}{\frac{1}{K-\frac{3}{2}}\sum_{k=0}^{K-1} |r_k|^2 - \frac{1}{K\left(K-\frac{3}{2}\right)}\left[\sum_{k=0}^{K-1} \mathrm{Re}\left\{r_k^* s_k\right\}\right]^2} \tag{4.2}$$

4.2.1.2. RxDA / TxDA Estimator

The SNV may be used as an in-service RxDA estimator which means that the sent sequence s_k does not have to be knownat the receiver. Instead we use the symbol decisions of the receiver for the sent sequence s_k.
Another possibility is to use the estimator as an TxDA estimator (non in-service type). Here we have perfect knowledge of the transmitted sequence and do not rely on receiver decisions and can use the known sent symbols s_k directly.

4.2.2. Moment-Based Estimation

The following section presents a blind SNR estimation using second- and fourth-order moments. One of the main advantages is that no knowledge about sent data and carrier phase is needed for processing. The following section including the derivations are a summary of the ideas presented in the two papers [GK06] and [PB00]

4.2.2.1. Derivation of the Algorithm

Assuming a symbol sequence of length L the estimator M_2 for the 2nd moment $E\{|r_k|^2\}$ is given by:

$$\hat{M}_2 = \frac{1}{L}\sum_{k=1}^{L} |r_k|^2$$

If L is large enough then $\hat{M}_2$ is a reliable estimate for the real expected value which yields to $S+N$:

$$\hat{M}_2 \approx \mathrm{E}\{|r_k|^2\} = S\mathrm{E}\left\{|s_k|^2\right\} + \sqrt{SN}\mathrm{E}\left\{s_k w_k^*\right\} + \sqrt{SN}\mathrm{E}\left\{s_k^* w_k\right\} + N\mathrm{E}\left\{|w_k|^2\right\} = S+N$$

The estimator $\hat{M}_4$ of the fourth moment $\mathrm{E}\{|r_k|^4\}$ is given by:

$$\hat{M}_4 = \frac{1}{L}\sum_{k=1}^{L} |r_k|^4$$

If again L is large enough then $\hat{M}_4$ is an estimate for $\mathrm{E}\{|r_k|^4\}$ which yields to the following expression, where $K_c = \mathrm{E}\{|c_k|^4\}$ is defined as the symbol curtosis.:

$$\hat{M}_4 \approx \mathrm{E}\{|r_k|^4\} = K_c S^2 + 4 \cdot S \cdot N + 2 \cdot N^2$$

Both equations can be solved with respect to S and N establishing the moment-based estimator:

$$\hat{S} = \sqrt{\frac{2\hat{M}_2^2 - \hat{M}_4}{2 - K_c}} \tag{4.3}$$

$$\hat{N} = \hat{M}_2 - \hat{S} = \hat{M}_2 - \sqrt{\frac{2\hat{M}_2^2 - \hat{M}_4}{2 - K_c}}$$

4.2.2.2. Estimation for M-ary PSK

The SNR estimate for an M-ary PSK is now given by simply dividing both estimators and by knowing that the symbol curtosis $K_c = 1$:

$$\hat{\rho}_{M-PSK\,NDA} = \frac{\hat{S}}{\hat{N}} = \frac{\sqrt{2\hat{M}_2^2 - \hat{M}_4}}{\hat{M}_2 - \sqrt{2\hat{M}_2^2 - \hat{M}_4}}$$

To evaluate the estimator performance we will compare the jitter variance of this estimator with the normalized CRLB of for DA SNR Estimation:

$$NCRLB = \frac{MCRLB(\rho)}{\rho^2} = \frac{1}{L}\left(1 + \frac{2}{\rho}\right)$$

4.2.2.3. Estimation for Non-Constant Envelope Modulations

According to [GK06] the above estimators are well suited for constant envelope modulations (e.g.: M-ary PSK modulation) but the the performance degrades rapidly when using modulations with a non-constant envelope. This problem arises because of the jitter which is introduced by patterns with varying amplitudes. Two modulation formats of DVB-S2 namely 16-APSK and 32-APSK have such a non-constant envelope. To estimate the SNR for this higher modulation schemes an adaption of the conventional algorithm is necessary.

The idea used from [GK06] is to generally estimate the SNR using the M_2M_4 algorithm just on a portion of the available symbols (e.g. only the symbols on the outer ring). Therefore the signal space has to be reliably partitioned into symbols that have the same amplitude. Out of the received symbols the partition radii (borders between the partitions) have to be estimated.

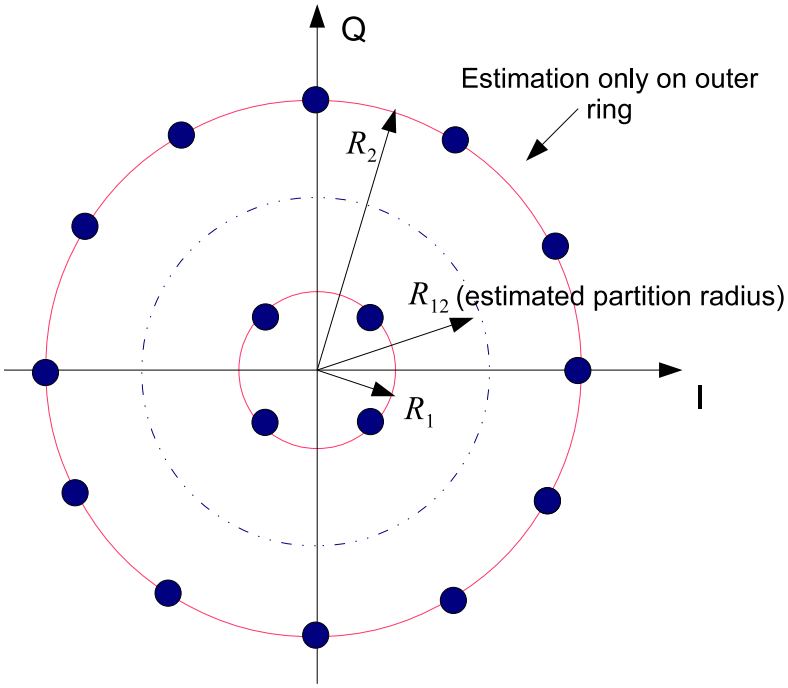

Figure 4.3.: Partitioning of the signal space in 16-APSK

For this purpose a general signal power average has to be estimated by using (4.3). This estimate exhibits a jitter floor for increasing SNR but can be reduced by choosing the observation length L as long as possible (e.g.: all symbols within the frame). Since the modulation scheme is known in advance we can define the partition radii $R_{i\to i+1}$ out of the signal power. Now we could choose between the different rings. Because in 16-APSK and 32-APSK there are always more symbols placed on the outer ring (therefore a longer estimator length L_z is possible) we will always take this choice. The following two sections will present a short overview of how applying the algorithm to the 16-APSK and 32-APSK modulation schemes.

Estimation for 16-APSK For the 16-APSK modulation type the symbol curtosis is given by:

$$K_c = E\left\{|c_i|^4\right\} = \frac{1}{4}R_1^4 + \frac{3}{4}R_2^4$$

where R_1 and R_2 represent the modulation radii which are known and depend of the chosen MODCOD. The factors before R_1 and R_2 represent the relative occurrence of a symbol on the according ring. This can be seen in the modulation scheme since 4 of 16 ($=\frac{1}{4}$) symbols lie on the inner ring R_1 and 12 of 16 ($=\frac{3}{4}$) symbols on the outer ring R_2.
The estimate for the signal power is now found by inserting the given values into (4.3), yielding to:

$$\hat{S} = \sqrt{\frac{2\hat{M}_2^2 - \hat{M}_4}{2 - \left(\frac{1}{4}R_1^4 + \frac{3}{4}R_2^4\right)}}$$

The estimated partition radius between the outer and inner ring is now given by:

$$\hat{R}_{12} = \frac{1}{2}\sqrt{\hat{S}}(R_1 + R_2)$$

After estimating the partition radius R_{12} we can realize the idea of an SNR estimation only for the outer ring by accounting only the symbols z_k that have a magnitude greater than R_{12}.

$$|r_k| > \hat{R}_{12} : \; z_k = r_K$$

$$\hat{M}_2' = \frac{1}{L_z} \sum_{k=0}^{L_z-1} |z_k|^2 \, , \; \hat{M}_4' = \frac{1}{L_z} \sum_{k=0}^{L_z-1} |z_k|^4$$

The final SNR estimate is now given by:

$$\hat{\rho}_{16-APSK} = \frac{1}{R_2^2} \cdot \frac{\sqrt{2\hat{M}_2'^2 - \hat{M}_4'}}{\hat{M}_2' - \sqrt{2\hat{M}_2'^2 - \hat{M}_4'}}$$

The division factor of $\frac{1}{R_2^2}$ is because the following term would relate the SNR only to the outer circle. This term has of course a higher average signal power R_2^2 and therefore would lead to a higher SNR.

Estimation for 32-APSK For the 32-APSK modulation type the symbol curtosis is given by (where the factors are again defined by the probability that a symbol lies on the according ring):

$$K_c = E\left\{|c_i|^4\right\} = \frac{1}{8}R_1^4 + \frac{3}{8}R_2^4 + \frac{4}{8}R_3^4$$

The signal power estimate follows from (4.3) as:

$$\hat{S} = \sqrt{\frac{2\hat{M}_2^2 - \hat{M}_4}{2 - \left(\frac{1}{8}R_1^4 + \frac{3}{8}R_2^4 + \frac{4}{8}R_3^4\right)}}$$

The estimated borderline between outer and middle ring (R_2 and R_3 are known and depend of the chosen MODCOD) can now be determined as:

$$\hat{R}_{23} = \frac{1}{2}\sqrt{\hat{S}}(R_2 + R_3)$$

Again only symbols from the outer ring are taken for the M_2M_4 estimator :

$$|r_k| > \hat{R}_{23} : \; z_k = r_K$$

$$\hat{M}_2' = \frac{1}{L_z} \sum_{k=0}^{L_z-1} |z_k|^2 \, , \; \hat{M}_4' = \frac{1}{L_z} \sum_{k=0}^{L_z-1} |z_k|^4$$

The final SNR estimate is given by:

$$\hat{\rho}_{32-APSK} = \frac{1}{R_3^2} \cdot \frac{\sqrt{2\hat{M}_2'^2 - \hat{M}_4'}}{\hat{M}_2' - \sqrt{2\hat{M}_2'^2 - \hat{M}_4'}}$$

4.3. Application of the SNR Estimators in DVB-S2

4.3.1. DA SNV Estimation on PLHEADER and Pilots

The PLHEADER consists of the unique word SOF which is always the same and therefore known to the receiver. Additionally, if we assume that we have decoded the PLS part of the PLHEADER correctly, we can also use this data for the DA SNR estimation.
Another option is to use the sent pilots in the frame format. After descrambling the pilot blocks at the receiver, they can also be used for DA SNR estimation. Since we assume that we have achieved a proper frame synchronization, we will know the transmitted symbol sequence perfectly. Hence we will use the SNV estimation as TxDA estimator.

4.3.1.1. Phase Offset Correction

Since the phase offset between the receiver and transmitter would lead to completely wrong results, a phase estimation and correction for the received symbols is necessary before determining the SNR estimate. The phase correction is performed by using the same ML estimation as in section 3.3.1. Since the phase estimation adds an extra jitter, the SNR result has a higher variance which can be seen in section 6.3.2.2.
The application of the SNV estimator now works by simply using the phase-corrected r_k symbols and inserting the known pilot symbols s_k (the pilot symbols depend of the currently decoded block) into the bias-corrected SNV estimator from (4.2).

4.3.1.2. Calculation on Individual Blocks And Combining

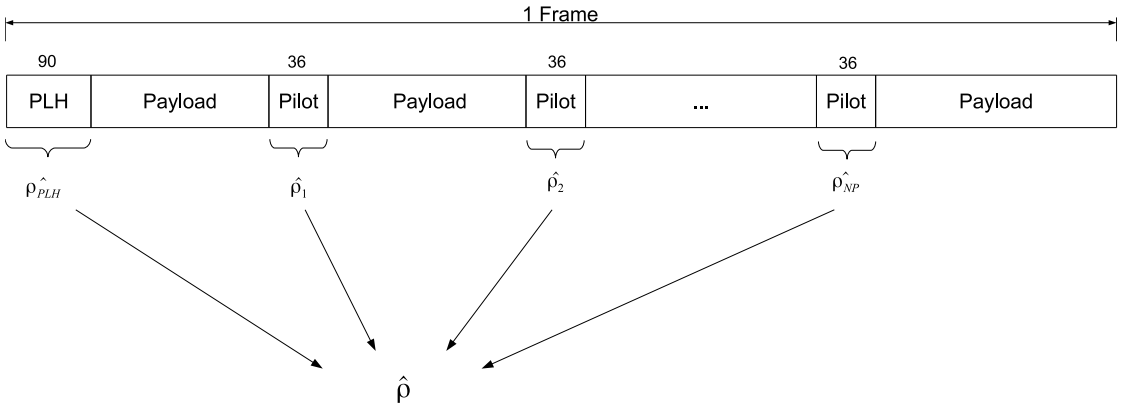

Figure 4.4.: DA SNR estimation for a single frame

We perform an individual SNR estimation on each of the known data blocks within a frame (PLHEADER, pilot blocks). In each block also a new phase estimation & correction is performed because already small frequency offsets may lead to big errors when using the phase estimates of the previous block . Since we want to get one result per frame we need to average the final result of the individual block results (see figure 4.4). Here we calculate a weighted average since the PLHEADER carries more symbols (90 symbols) than the pilot blocks (36 symbols). The number of pilot blocks, which depends on the chosen frame format, is denoted as N_p.

$$\hat{\rho} = \frac{90 \cdot \hat{\rho}_{PLH} + N_P \cdot 36 \cdot \sum_{k=1}^{N_p} \hat{\rho}_k}{90 + N_p \cdot 36}$$

4.3.2. M_2M_4 Estimator

After the PLSC has been decoded, we know the modulation scheme that has been used within this frame. Now the appropriate NDA SNR estimation algorithm can be chosen (M-PSK, 16-APSK or 32-APSK). Since only the payload symbols within a frame are modulated by the same modulation scheme we choose to exclude the PLHEADER symbols and pilots from our estimation. To allow a *fair comparison* of the SNR estimator performance in the simulation we limit the effective amount of symbols used by all estimators to the same number L_z .

4.3.2.1. Constant Envelope Modulation (QPSK, 8-PSK)

Figure 4.5 illustrates which symbols are used for the estimation. For simplicity reasons the pilot blocks are not shown. We only use the first L_z symbols of the payload and perform the estimation that has been described in section 4.2.2.2.

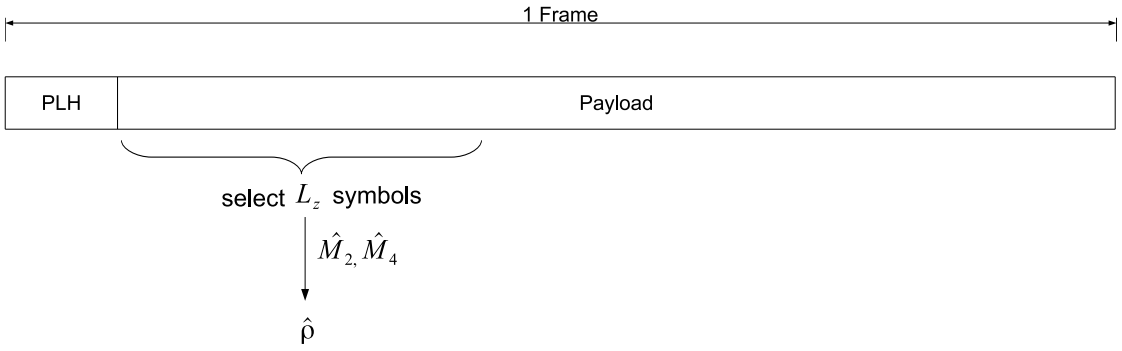

Figure 4.5.: Moment-based SNR estimation for M-PSK

4.3.2.2. Non-Constant Envelope Modulation (16-APSK, 32-APSK)

In case of non-constant envelope modulations (16-APSK, 32-APSK) we use all symbols of the frame to calculate the estimate of the signal power $\hat{S}$ as illustrated in figure 4.6. Then the saved buffer of frame symbols will be checked for symbols that have a greater magnitude than the calculated partition radius and $\hat{M}_2{}'$ and $\hat{M}_4'$ are calculated. If there are not enough payload symbols within a frame whose magnitudes reach over this threshold then we will not use this frame for the estimation process.

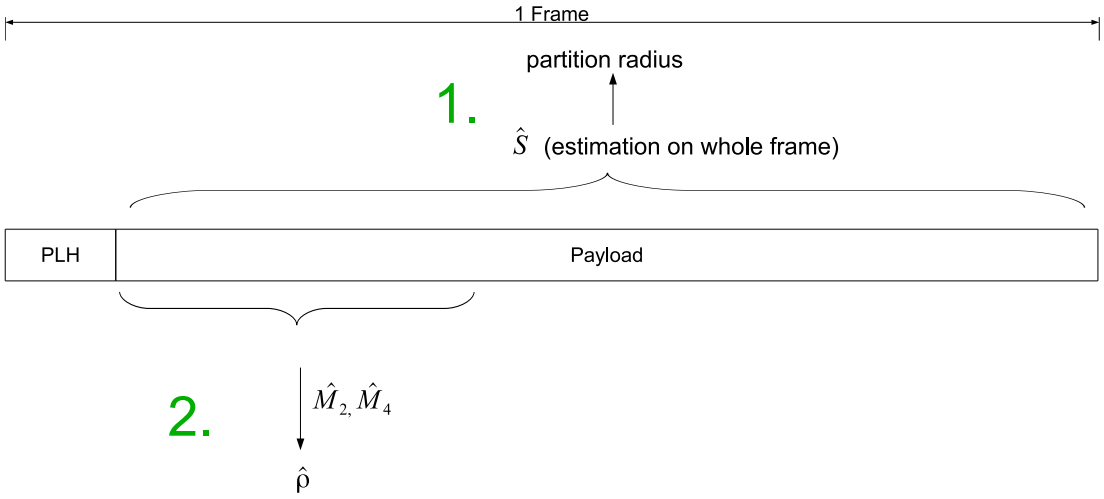

Figure 4.6.: Moment-based SNR estimation for 16-APSK and 32-APSK

4.3.2.3. Remark

All the square root expressions for moment-based SNR estimation may rarely have negative arguments. If this is the case then we will throw away the estimated result of this frame and will have to wait for the next frame.

5. Implementation

This chapter presents the practical implementation of the system. First we show the general architecture of the GNU Radio platform. Then we will have a look at the possibilities of how to map our developed block diagram on to the GNU Radio platform. Furthermore it will be shown how to perform software-only simulations on this platform. This will be followed by an overview of the implemented software-blocks. A protoptype architecture is proposed and it is shown which parts have been implemented in the FPGA and which have been implemented in software. In the last section the detailed implementation of the differential correlator in the FPGA will be explained.

5.1. Receiver Architecture Using the GNU Radio Platform

5.1.1. The GNU Radio Platform

The GNU Radio platform (figure 5.1) has been chosen as the favoured Software Defined Radio (SDR) architecture. GNU Radio offers a flexible SDR architecture that is all open source. It is essentially a software framework that is running on a PC for processing digital samples.
GNU Radio can use a hardware sampling device called USRP to get the samples into the processing PC. The USRP needs a daughterboard (WBX) that performs the downconversion of the chosen band. Afterwards the signal is AD-converted and processed digitally on a FPGA. Finally the samples are sent over Ethernet to the PC were they are received through a hardware driver (UHD).
The received samples are then further processed on the GNU Radio software platform. Further details about the USRP, the WBX and the UHD drivers can be found in the appendix section. Details about GNU Radio can be found in [Bis12].

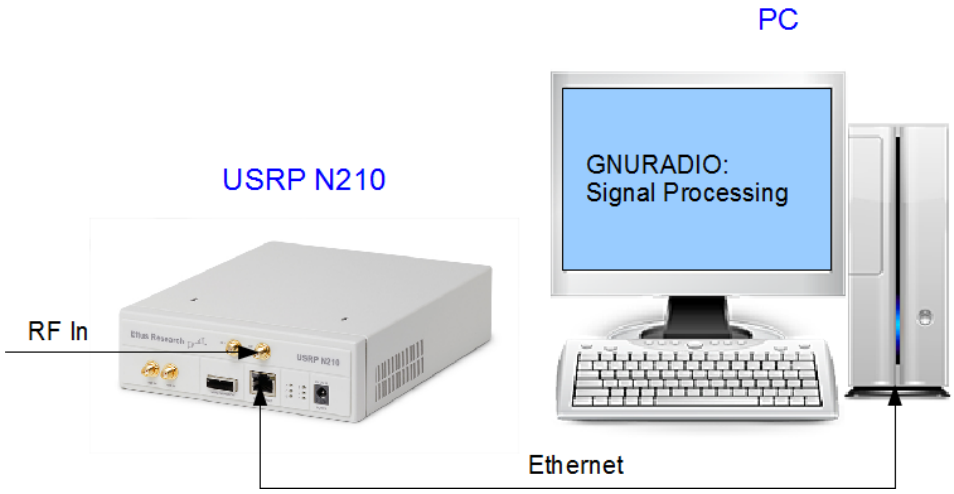

Figure 5.1.: The GNU Radio platform (USRP + PC)

5.1.2. Block Diagram of the Receiver

This section presents an overview of the principal receiver architecture and with blocks necessary to perform SNR estimation and MODCOD decoding. The detailed architecture of the implemented prototype can be found in section 5.4.
The WBX daughterboard provides the downconversion from L-Band to baseband. After A/D conversion of the baseband signal the mainboard of the USRP performs: decimation through halfband filtering and CIC filtering. These blocks are already provided by the USRP FPGA in standard configuration. The target sample frequency after decimation is four times the symbol rate for the chosen architecture. The custom made blocks now follow. The Root-Raised Cosine (RRC) filter, symbol timing recovery and the interpolator have been implemented in another thesis [Bis12]. The following blocks: correlator, peak detection, PLSC decoding and DA/NDA SNR estimation were all implemented in this Master Thesis.

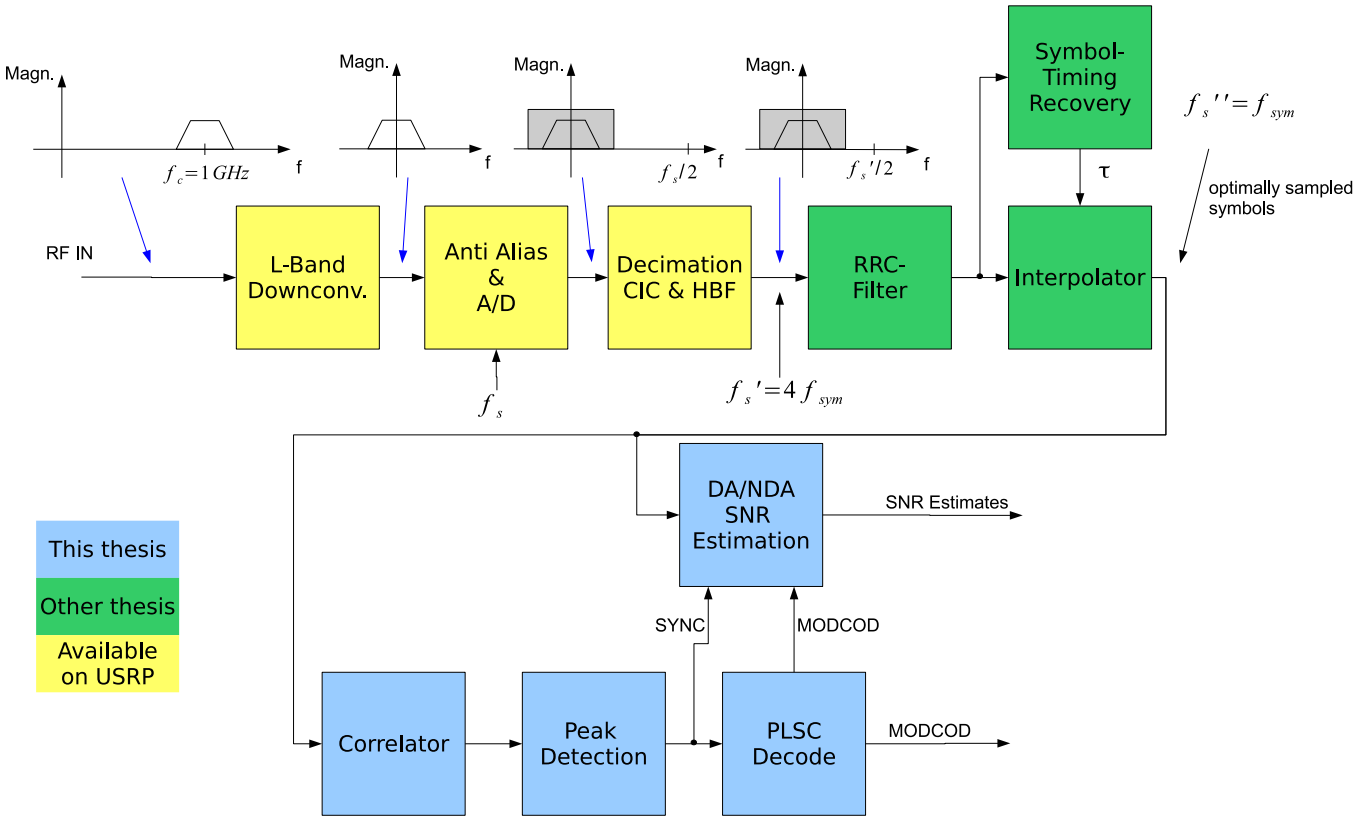

Figure 5.2.: Principal receiver architecture

The custom made blocks may now be implemeneted in software or hardware. The following section will provide an overview of some possible configurations.

5.1.3. Possible Configurations

The great advantage of the SDR concept [Mit00] is that the blocks may float between software and hardware. This allows the following implemented reasonable configurations:

- Only downsampling on the FPGA
  - RRC filter, digital interpolation, correlation, peak detection, PLSC decoding and SNR estimation all implemented in software.

  - Extremely high CPU workload

- RRC filter and interpolation on the FPGA
  - Correlation, peak detection, PLSC decoding and SNR estimation in software.
  - moderate CPU workload

- RRC filter, interpolation and correlation on the FPGA
  - Only peak detection, PLSC decoding and SNR estimation in software.
  - Low workload for CPU

Implementing everything in software offers the great advantage of flexibility. Changes can be made very easy by changing some lines of software code. Additionally the PC has the great advantage that every signal processing operation can be performed in floating point and therefore the implementation loss compared to a fixed point FPGA implementation can be much lower. The drawback is that there is a huge amount of data to calculate on the PC. Depending of the CPU speed the processing power may not be enough to fullfill the realtime calculation especially when high symbol rates are chosen. The preferred architecture of the prototype will be presented in section 5.4.

5.2. Architecture of the Simulation Platform

Before implementing algorithms in hardware and generally to evaluate the performance of the algorithms simulations are necessary. The GNU Radio platform can also be used for simulation only. Samples may be generated by software blocks instead of coming from a real hardware (USRP). The following section will provide a short overview of how to use GNU Radio as a simulation environment.

5.2.1. GNU Radio Companion (GRC)

GNU Radio includes a comfortable graphical user interface called GNU Radio Companion to draw block diagrams. It provides a nice overview over the connections between the blocks. An exemplary simulation graph is shown in figure 5.3. Individiual block connections and the parameters can easily be altered in this GUI. In the background GRC automatically generates a Python script that generates the connections between the blocks. One drawback of this method is that automatic parameter changes, e.g. SNR, are not possible.

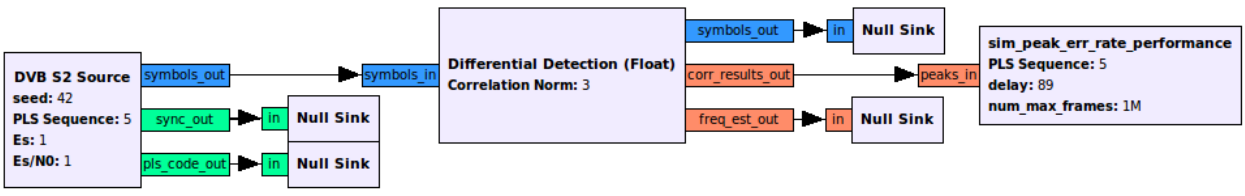

Figure 5.3.: Simulation graph in GRC

5.2.2. Python

To automate parameter changes Python scripts can be used. The automatically generated Python script (`top_block.py`) of the GRC environment can be used as template to simplify the procedure. The graph is started and a block inside the graph terminates the simulation (i.e. after N frames). Afterwards the Python script gets the calculated results of a block by calling an appropriate function on this GNU Radio block. Then another simulation can be started with different initial parameters. At the end all collected results are written into a .mat file.

5.2.3. Graphical Analysis

The produced .mat files can now be post-processed by MATLAB. Here we can generate some plots to get graphical results (e.g.: error rate curves).

5.3. Implemented Blocks in GNU Radio

The great benefit of using the GNU Radio platform is that it can be used for either realtime applications (when using samples from the USRP) or for simulations (when using self generated samples) by using the same C++ code. The following section provides an overview of the implemented software modules.

5.3.1. Tools Library

To implement some common functions used in more than one module, a tools library has been implemented. Functions include:

- PLSC decoding by minimum Hamming distance
- Determine frame lengths (with or without header and pilots) for a given PLS
- Phase correction by using correlation
- Scrambling and descrambling of PLSC
- Determination of complex scrambling sequence
- $\frac{\pi}{2}$ - BPSK symbol mapping and demapping
- QPSK, 8-PSK, 16-APSK, 32-APSK symbol mapping
- Radii determination for 16-APSK and 32-APSK
- Various debugging functions (Matlab export, vector printing, ...)

5.3.2. DVB-S2 Source Block Implementation

A DVB-S2 source has been developed on the GNU Radio platform using a C++ module. Its purpose is to provide complex samples according to the DVB-S2 frame format (see section 2.5) at its output. These samples can be used for simulations and to test modules. The source block provides functions to set the signal-to-noise ratio $\frac{E_S}{N_0}$ and the average signal energy E_s. Aditionally an arbitrary PLS sequence can be defined so the module produces valid frames according to the MODCOD, long/short Frames, and pilots/no pilots configuration. The PLHEADER and pilot parts of the frame confirm to the DVB-S2 standard while the other parts just consist of randomly chosen data bits modulated according to the current MODCOD.
To test the source block the output samples were written out to a file (file sink). Later the output samples were read with Matlab and a histogram of the samples amplitudes was plotted. Then the relative occurence was checked.

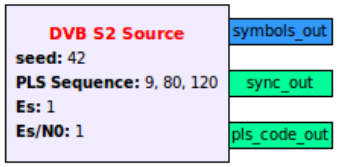

Figure 5.4.: DVB-S2 source block in GRC

5.3.3. Differential Correlation Implementation

The differential correlation has been implemented in a C++ module as floating point model and also as a bit-accurate model. The model offers to choose between the three correlation norm calculation methods (see section 5.5.5 for a detailed discussion):

- Euclidean norm (Norm 2)
- approximated Euclidean Norm (Alpha Max Plus Beta Min): using $\alpha = \frac{15}{16};\ \beta = \frac{15}{32}$
- Manhatten norm (Norm 1)

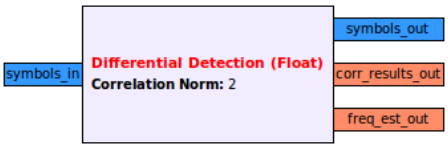

Figure 5.5.: Differential correlator block in GRC

5.3.4. Peak Detector

Two different peak detectors have been evaluated. The implementations of the peak detectors already include the PLSC decoding of figure 5.2. After a frame synchronization has been

found a FRAME SYNC signal (sync_out) and the according PLS (which is the MODCOD + TYPE fields) will be signaled at the output of the block. The symbols (symbols_out) will be delivered in synchronization to frame synchronization. So at each first symbol of a frame the FRAME SYNC will be signalled.

5.3.4.1. Exponential Averaging Implementation

This block represents the C++ implementation of the algorithm that has been discussed in section 3.2.2.2. The two parameters α (weigthing parameter) and β (threshold) need to be set before the simulation starts.

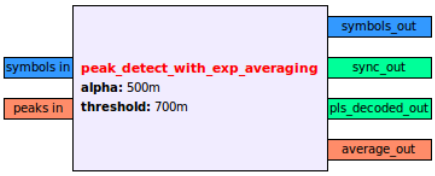

Figure 5.6.: Exponential averaging block in GRC

First simulations of this algorithm have shown that it is heavily threshold-sensitive even at SNRs of above $5\,dB$. Therefore this algorithm has not been further anlayzed since we should support much lower SNR values in this work.

5.3.4.2. Adapted Peak Search Implementation

The used algorithm (see figure 5.8) works by simply finding the initial SYNC according to the peak search algorithm of section 3.2.2.5. After acquisition a frame by frame decoding of the incoming symbols (no correlation results needed anymore for this) can be performed to maintain the synchronization. The following parameters were chosen for the final algorithm:

Parameter	Chosen value
# of peaks in each small window	$N = 3$
# of small windows	$M = 30$
# Symbols in big search window	$M \cdot 3330 = 99900$
# matching peaks	$L = 3$

Table 5.1.: Chosen parameters for the adapted peak search algorithm

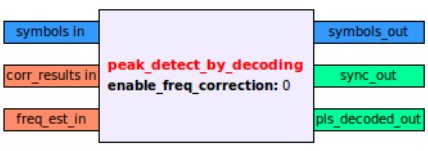

Figure 5.7.: Adapted peak search block in GRC

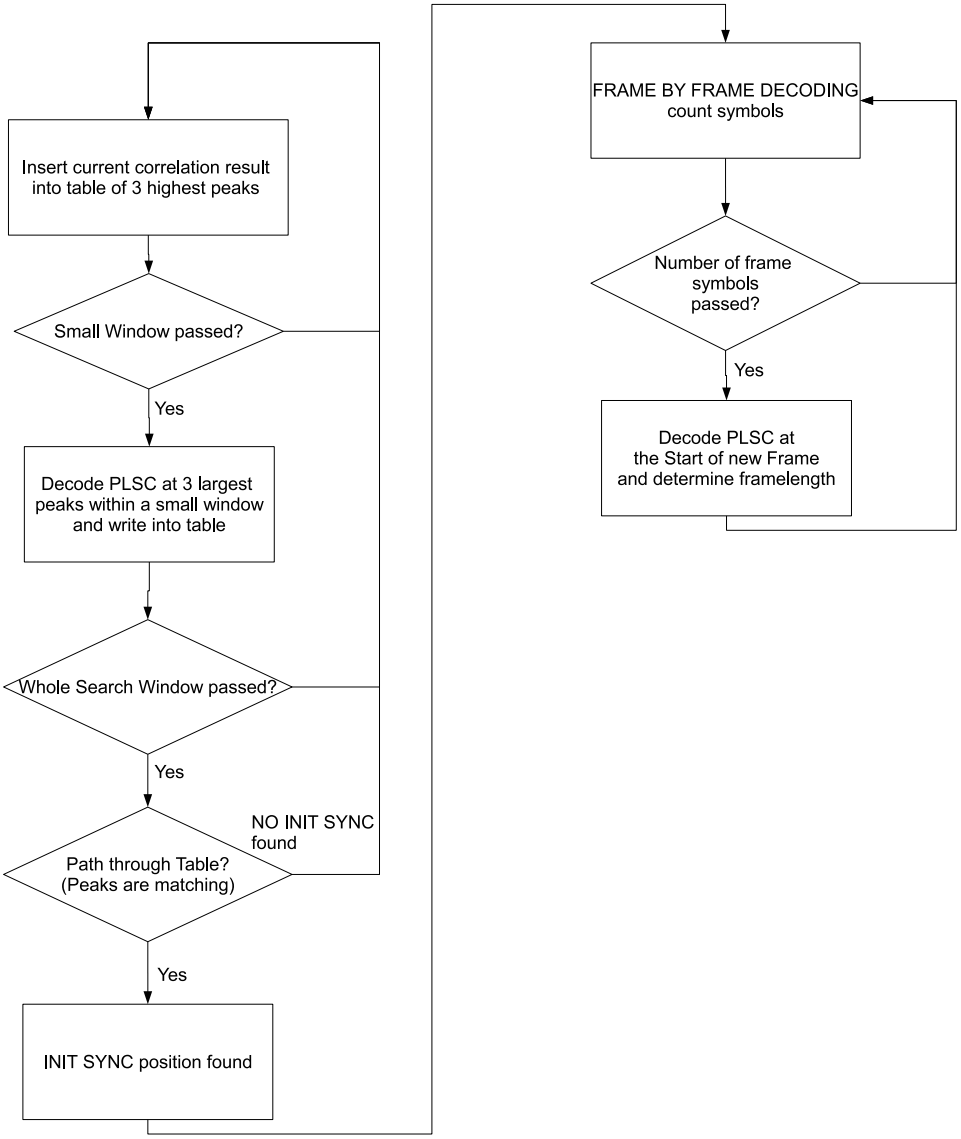

Figure 5.8.: Algorithm for INIT frame SYNC

5.3.4.3. Adapted Peaks Search and Resyncing

One problem that was observed during tests was that the symbol timing sometimes got out of lock. This was due to timing slips occuring in the recovery algorithm. This violates our assumption that the symbol timing is locked all the time and the frame-by-frame decoding process will not work as expected. The problem was solved by performing a repeated INIT SYNC calculation (using the same algorithm as mentioned above) in parallel to the frame-by-frame decdoding. After a repeated INIT SYNC has been found it is checked if the frame-by-frame decoded SOF position still matches with the found INIT SYNC position. If there is a mismatch then somehow the symbol timing got lost. Therefore a RESYNC to the correct position is necessary. If both positions match then there is no further action needed.

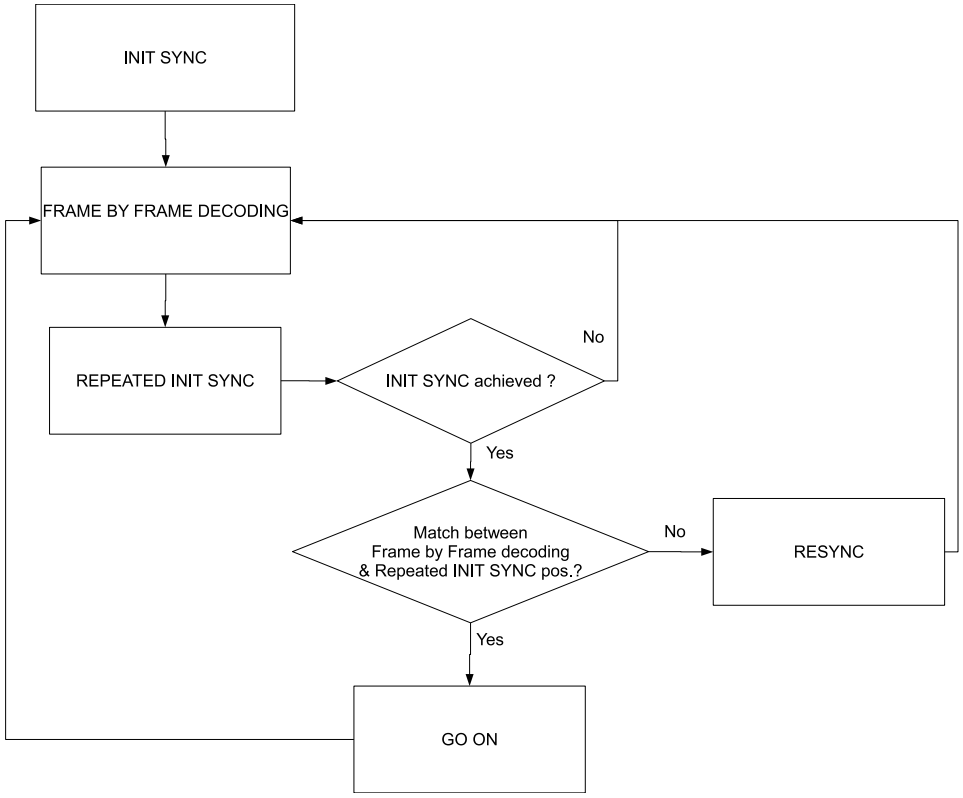

Figure 5.9.: Algorithm for frame SYNC with resyncing

5.3.5. SNR Estimation Blocks Implementation

The SNR estimation blocks (figure 5.10) are implemented according to the algorithms presented in section 4.3. Each time a SYNC signal is asserted at the `sync_in`-input the SNR estimation will start and produce one result per DVB-S2 frame. The DA estimator will even produce two results: one is the estimate on the whole frame (PLHEADER and pilots), the other the estimate only on the PLHEADER part of the frame). The NDA estimation algorithms (constant envelope, non-constant envelope) are chosen depending on the decoded PLSC which is transmitted on a seperate pin (`pls_decoded_in`).

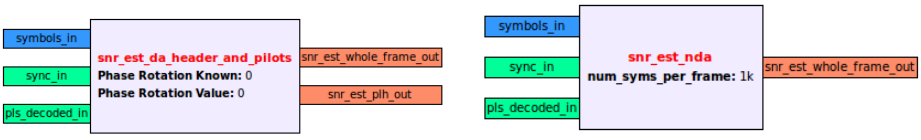

Figure 5.10.: SNR estimator blocks in GRC

5.4. Architecture of the Prototype

Filter operations like the RRC-filter, digital interpolation and differential correlation require a lot of processing power. Since these filter operations are optimally suited to be processed on the FPGA and the symbol rates are quite high (up to $8MSyms/s$) we decided to implement them inside the FPGA. So the proposed architecture is shown in figure 5.11.

The WBX daughterboard is responsible for the amplification and complex downconversion of the received signal. The downconverted I/Q signals then pass through the anti-aliasing filters. The ADC now samples both paths with 100 Mhz to get the digital samples.
Then the digital samples are further decimated by a combination of halfband filters and CIC filters. The CIC filter and halfband combination can decimate in a variable range from 4 to 128. The actual decimation factor depends on the required symbol rate and can be set through the UHD driver. The decimation factor is chosen such that there are 4 samples per symbol. Therefore allowing a maximum and minimal symbolrates:

$$f_{sym_{max}} = \frac{f_s}{M_{min}} = \frac{100\,Mhz}{4 \cdot 4} = 6.25\,MSym/s$$

$$f_{sym_{min}} = \frac{f_s}{M_{max}} = \frac{100\,Mhz}{4 \cdot 128} = 0.1953\,MSym/s$$

The samples are then filtered by a RRC filter and passed to a digital interpolator that restores the optimal symbol timing. The digital interpolator gets the symbol timing estimate input calculated by a selected recovery algorithm. The former components (RRC, interpolator and timing estimator) were implemented in another master thesis [Bis12]. The now optimally sampled symbols (1 sample per symbol) are then fed into the differential correlator.
The optimally sampled I/Q symbols are now transmitted on channel 0 and the correlation results are transmitted synchronized with the samples on channel 1 (each channel offers 32 bit). The remaining tasks (peak detection and SNR estimation) are now performed on the PC.

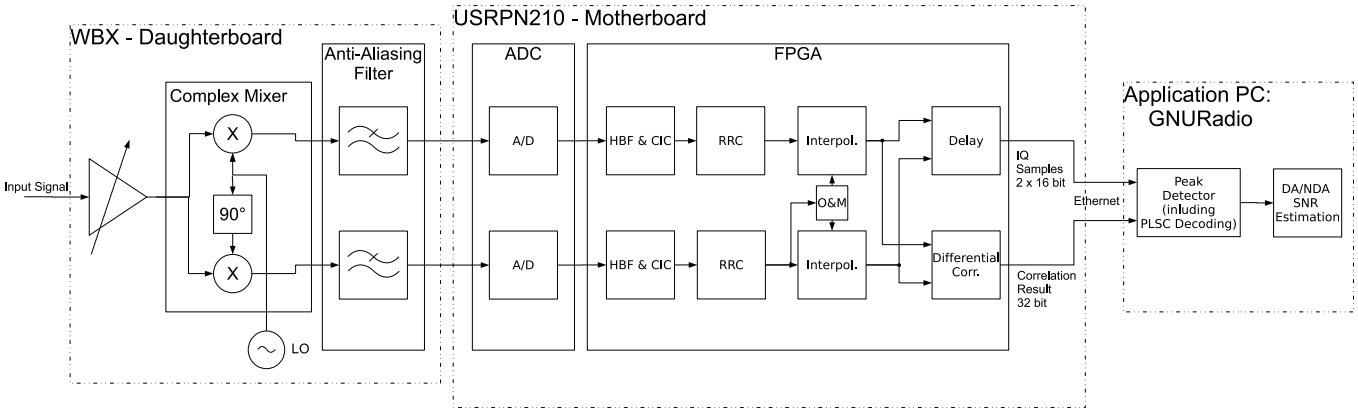

Figure 5.11.: Proposed architecture

5.5. Differential Correlator (FPGA Implementation)

The I/Q samples after the digital interpolator are used as input for the differential correlator circuit. The correlation results are calculated and sent to the PC via Ethernet on RX channel 1. Further we wanted to synchronize the correlation results with the incoming samples. Therefore we need to delay the I/Q samples for the same amount of time the correlation results appear at the output (delay line).

5.5.1. High Level Model in Matlab and C++

Before implementing the circuit in hardware a high level model has been implemented in C++ and Matlab. This allows the evaluation of the needed bit widths and also provides the opportunity to generate data to test the hardware module. In fact this is simply a bit-accurate model of the floating point implementation in section 5.3.3.

5.5.2. Simulation Environment in VHDL

An essential part of testing VHDL modules is to make a functional verification by simulation. Figure 5.12 shows an automatic test bench for functional verification. The test data that were generated by Matlab were exported as .dat-file and used as an input stimulus to the VHDL module inside a test bench environment. Additionally the calculated results out of the Matlab model were also exported into a .dat-file. The resulting data in this "golden" file was used to compare them against the outputs of the VHDL module.

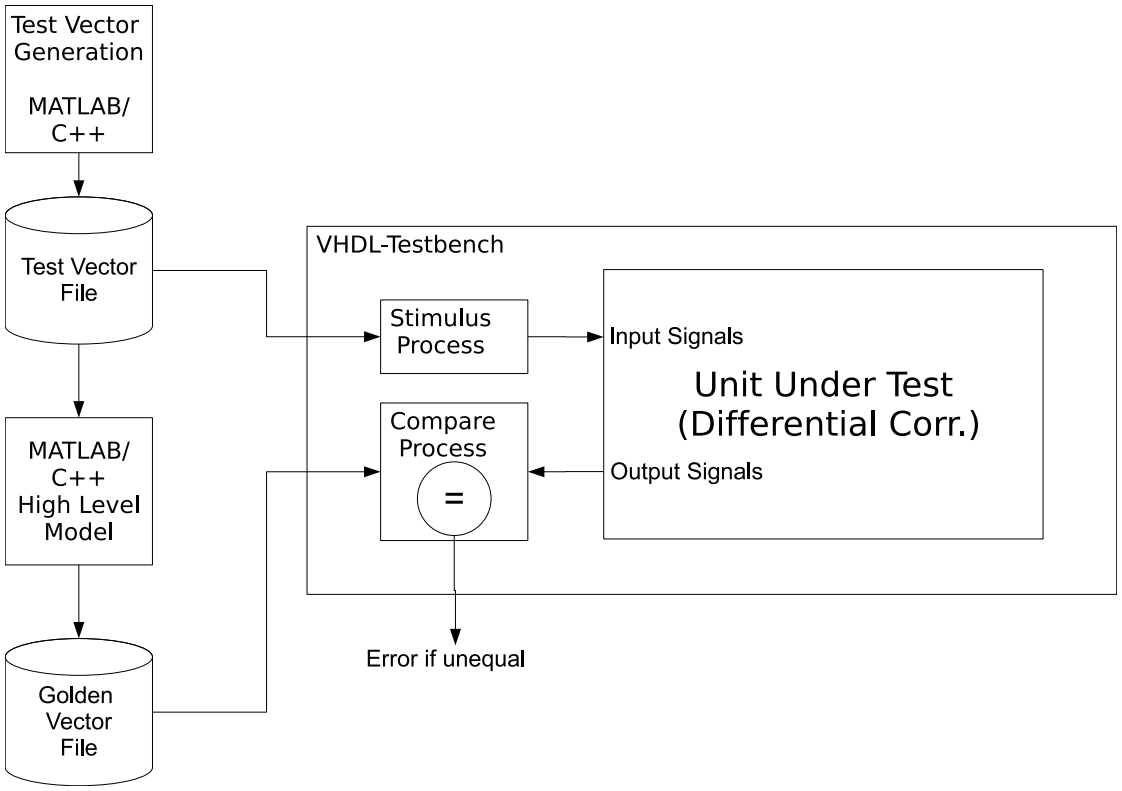

Figure 5.12.: Test bench for the differential correlator module

5.5.3. Determing the Differential

To determine the differential $r_k \cdot r_{k+1}^*$ on the FPGA the following operation has to be implemented:

$$r_k \cdot r_{k+1}^* = (\mathrm{Re}\{r_k\} + j \cdot \mathrm{Im}\{r_k\}) \cdot (\mathrm{Re}\{r_{k+1}\} - j \cdot \mathrm{Im}\{r_{k+1}\})$$

To simplify the equation the following substitutions are inserted:

$$\begin{aligned} a &= \mathrm{Re}\{r_k\} \\ b &= \mathrm{Im}\{r_k\} \\ c &= \mathrm{Re}\{r_{k+1}\} \\ d &= -\mathrm{Im}\{r_{k+1}\} \end{aligned}$$

$$r_k \cdot r_{k+1}^* = (a + j \cdot b) \cdot (c + j \cdot d) = ac + j \cdot bc + j \cdot ad + j^2 \cdot bd = (ac - bd) + j \cdot (bc + ad)$$

The implementation of this equation can be seen in figure 5.13:

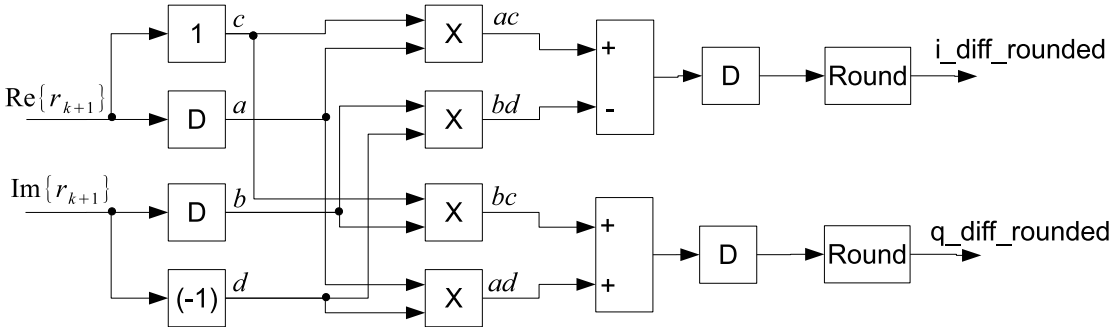

Figure 5.13.: Determine the differential

Since the input of the module is 16 bit the ouputs after the adders may grow up to 33 bit where 32 bit are from the multiplication of two 16 bit numbers plus one bit for the addition of two 32 bit numbers. We do not want to use that result by full extend and only use 16 bit registers for the correlator cicuit. Therefore a scaling by an appropriate rounding is necessary.

5.5.4. Use of Swapping Units instead of Multipliers

To reduce the hardware amount in the FPGA implementation a simple swapping unit has been used because the complex multiplication would be too costly [QXC+08]. According to section 3.2.1.4 the complex taps only take values of $\pm j$ hence the complex multiplication of the taps with the incoming data can be replaced by a simple swapping unit:

$$(a + j \cdot b) * (\pm j) = \mp b \pm j \cdot a$$

So we know that we can simply swap the real and imaginary part of the complex number and set the sign depending according to the current tap. For the practial implementation of this operation that consists of two multiplexers (see figure 5.14).

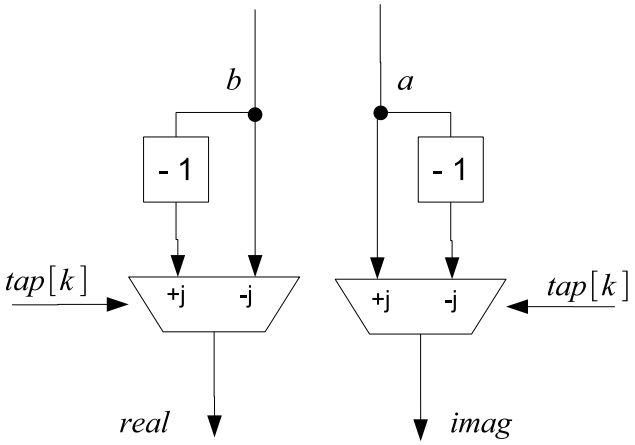

Figure 5.14.: Swap unit

5.5.5. Correlation Result Calculation

The final step of the correlation result calculation requires to choose the maximum of the absolute value of two complex numbers:

$$C = \max\left(\left|C_{SOF} + C_{PLSC}\right|, \left|C_{SOF} - C_{PLSC}\right|\right)$$

The calculation of the absolute value may be performed in different ways depending of the needed accuracy and complexity. The following section shows different routes to implement the calculation of the absolute value with a special focus on digital implementation. In the C++ software module the correct calculation and the approximations of the norm have been implemented to compare their performances (see section 6.1.2).

5.5.5.1. Using the Euclidean Norm (Norm 2)

Determing the correct absolute value (magnitude) of a complex number requires to square the real and imaginary part, add them and take the square root (Euclidiean norm). Since in (3.4) we only need to compare the absolute values of the two complex numbers we could omit the square roots but this would lead to a massive growth in bitlength. Therefore a square root calculation on the FPGA would also be necessary.

$$|c| = \|c\|_2 = \sqrt{a^2 + b^2}$$

Since the square root is a really unpleasant function to calculate on an FPGA the following approximations may be used to simplify the magnitude calculation:

Alpha Max Plus Beta Min (Approximation of Norm 2) This algorithm uses the following approximation for the magnitude [Lyo10, p. 679]:

$$|c| \approx \alpha \cdot \max\left(|a|, |b|\right) + \beta \cdot \min\left(|a|, |b|\right)$$

This avoids the calculation of squares and square roots and only introduces additional comparators, one multiplier and one adder. According to [Lyo10, p. 683] a reduced estimation error (maximum error: 6.3%) is provided by choosing the following parameters:

$$\alpha = \frac{15}{16};\ \beta = \frac{15}{32}$$

Another advantage of using this parameter pair is a simplified implementation by using only one multiplier and two binary right shifts:

$$|c| \approx 15\left(\max\left(|a|,|b|\right) + \min\left(|a|,|b|\right)/2\right)/16$$

The structure of the FPGA implementation is shown in figure 5.15.

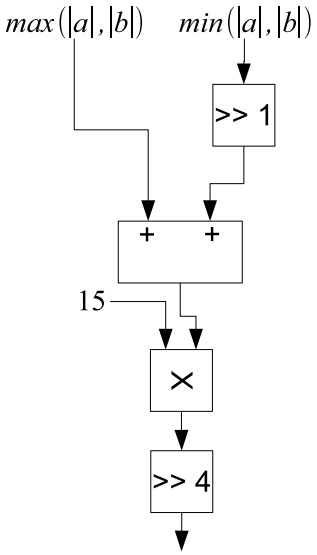

Figure 5.15.: Aproximate computation of the magnitude

To save area (since the FPGA was nearly 80% full and routing took a long time) and for simplicity reasons this approximated norm was used inside the FPGA implementation.

CORDIC Another option to calculate the magnitude is to use the CORDIC algorithm [MB07]. The magnitude of a complex number is determined by rotating a complex vector until it has a phase of zero or nearly zero. So the magnitude after rotating the vector is then simply given by the real part of the rotated vector.
The drawback of this method is that the rotation requires an additional amount of clock cylces. Therefore the number of registers to synchronize the correlation results and the I/Q samples increases. Since the FPGA was already filled to a decent extend it was decided not to use this method.

5.5.5.2. Using the Manhattan Norm (Norm 1)

Since we are only interested in finding maximum peaks we may use an approximation and replace the Euclidean norm by the Manhattan norm (see [Kae08]) :

$$|c| \approx \|c\|_1 = |a| + |b|$$

This results in an implementation loss regarding the Max Peak Error Rate Simulation in section 6.1.2. This implementation loss leads to a higher peak error rate and thus the acquisition process will take longer. This norm has also been implemented on the FPGA but dropped later in favour of the approximated Euclidean norm.

5.5.6. Correlator Structure

5.5.6.1. Direct Form

The circuit from figure 3.3 has been implemented in VHDL. One problem is that the critical path (maximum combinatoric delay) of the adders gets very long so it may be that we can not satisfy the operating speed. [QXC+08] solves this problem by introducing several pipeline registers between the adders. This requires of course an additional amount of registers for the pipelines and an additional amount for the delay line (since we required that the correlation signal is synchronized with the I/Q samples). Therefore a better way is to look for a different structure that can save the additional use of registers and reduce the maximum combinatoric delay.

5.5.6.2. Transposed Form

According to [Lyo10] and [RS09] the tranposition theorem can be used to change the structure of a linear time-invariant digital network without changing the transfer function. Since the circuit from figure 3.3 represents a linear time-invariant tapped delay line structure we can use this property here to convert from a "direct-form" filter into a "transposed-form" filter.

The transposition has many advantages:

- It shortens the critical path of the above circuit since values get registered between the adders.
- In principle it is enough to only use 2 swapping units. We only need to choose one of the outputs depending of the tap at this current position. This saves some amount of space.

One prerequisite to use this technique is that we need the delayed original I/Q samples at the input of the SOF correlator. Since we are going to send the correlation results synchronized to the I/Q samples anyway we already have a delay line. Now we are taking the signal after 64 registers (delay of the PLSC correlator) of this delay line as the input for the SOF correlator.

By integrating the first flip-flops of the two diff_corr modules into the delay line we can save two flip-flops in the delay line. This is recognized by the synthesis tool and automatically optimized.

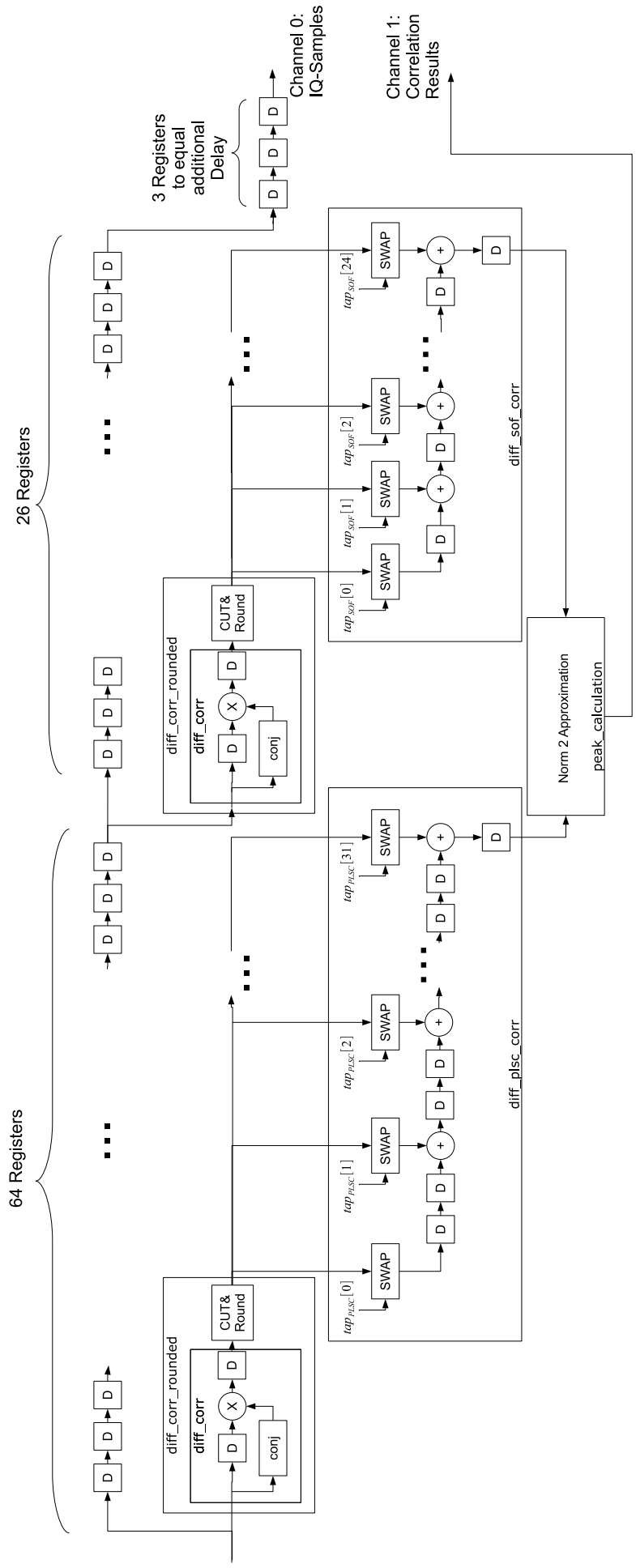

Figure 5.16.: Improved architecture (transposed form) of the differential correlator

5.5.6.3. Comparison between Direct Form Implemenation and Transposed Form Implementation

To analyze the benefit of using the transposed form implementation instead of the direct form, area resource comparisons and the maximal possible clock frequency have been analyzed. The results are presented in table 5.2.

	% of occupied Slices	max. Clock Frequency
Direct Form w. approx. Norm 2	16 %	22.11 MHz
Transposed Form w. approx Norm 2	10%	33.116 Mhz
Transposed Form w. Norm 1	9 %	100 Mhz

Table 5.2.: Comparisoin between different FPGA implementations

We can clearly see that we have reduced the number of occupied slices in the FPGA by using the transpased structure. Additionally we even improved the max. clock frequency speed of the implementation.

6. Simulation Results and Discussion

This chapter will present the simulations that have been performed on the GNU Radio platform and their results.

6.1. Performance of the Differential Correlation

6.1.1. Correlation Results at Different SNRs

Simulations of the differential correlator have been performed using GRC. A QPSK long frame format including pilots has been used. The correlation results have been written into a file sink, read and plotted with MATLAB.

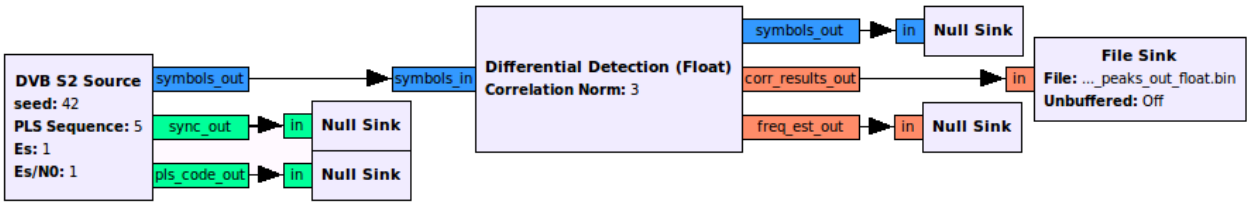

Figure 6.1.: Schematic to plot correlation results in GRC

Figures ?? and 6.3 show the correlation results for two different SNRs ($10dB$ and $-2dB$). At $10dB$ we can clearly see that peaks are rising high above the other correlation results. Each peak is exactly $N = 33282$ symbols (which represents the frame size for the chosen modulation) away from the next peak. So the SOF can be identified by the maximum peaks. An SNR of $\frac{E_s}{N_0} = -2dB$ leads to the circumstance that peaks cannot be identified anymore out of the correlation results. The highest peaks may not correspond to the points where the frames begin. Further studies of this phenomenon have been performed in the following section.

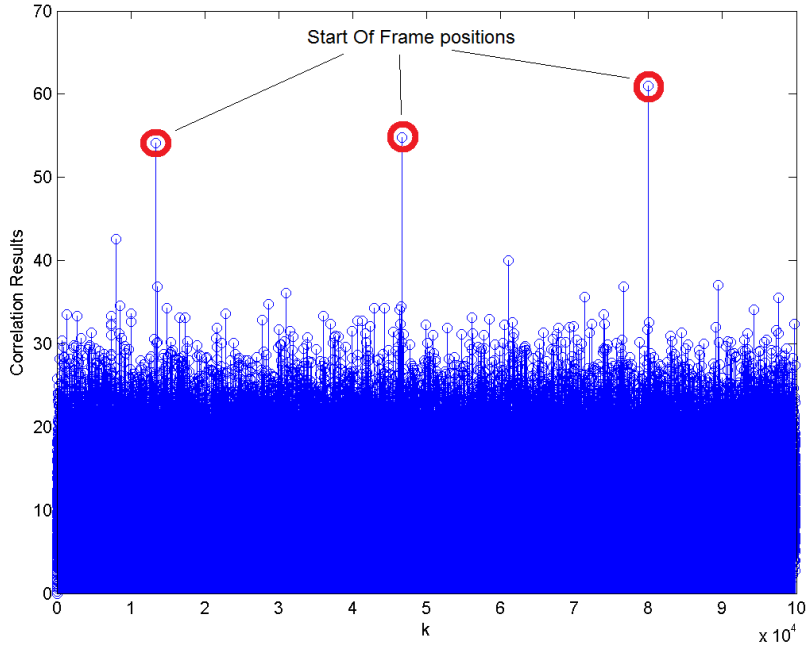

Figure 6.2.: Correlation results at $\frac{E_S}{N_0} = 10\,dB$

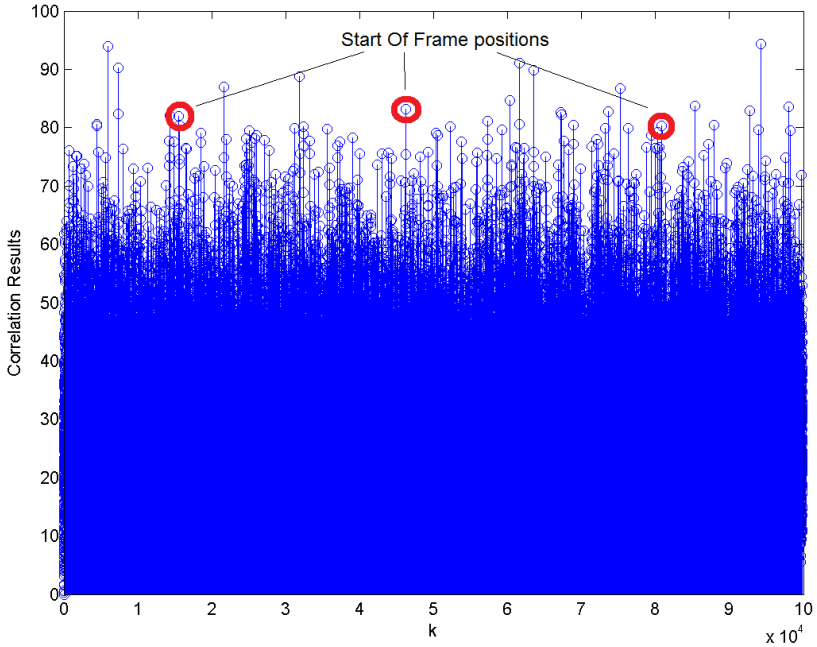

Figure 6.3.: Correlation results at $\frac{E_S}{N_0} = -2\,dB$

6.1.2. Max-Peak Error Rate Simulation

The performance of the differential correlation has been evaluated at different SNR points using GRC. The performance was measured in terms of the maximum peak error rate. This

means that we measure how often the highest peak within a frame occurs at the wrong position (not at the real SOF).

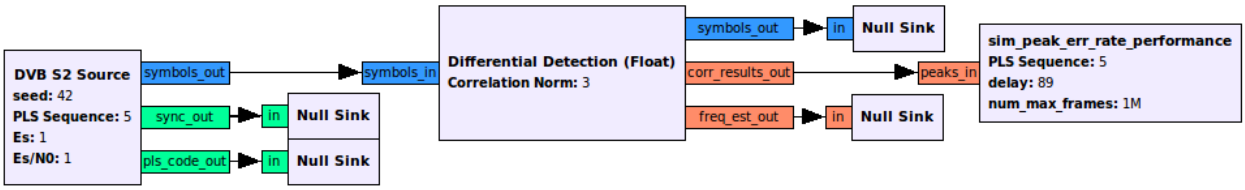

Figure 6.4.: Max-peak error rate in GRC

Figure 6.5 shows a simulation of the peak error rate over $N = 10^6$ QPSK Short Frames (to decrease the simulation time a short frame format has been used) using the floating point model. The curves were plotted for a frequency error free simulation $\Delta f \cdot T_S = 0$ using the different correlation result norms mentioned in section 5.5.5.

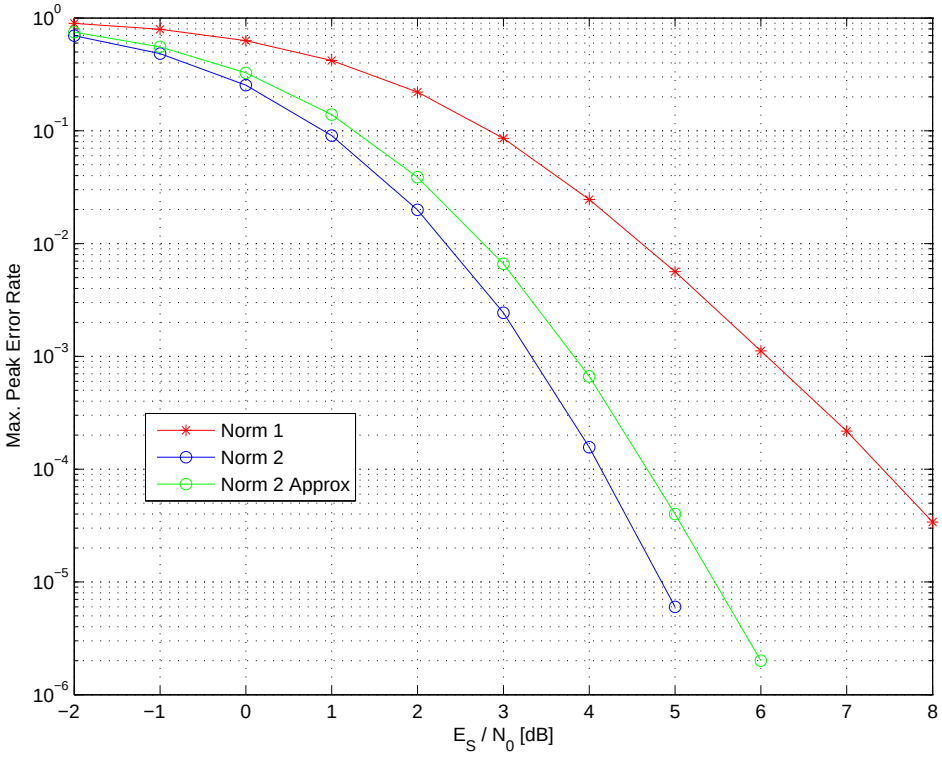

Figure 6.5.: Max-peak error rate ($\Delta f \cdot T_s = 0$)

From this analysis we can see that using the Norm 1 as an approximation leads to a higher peak error rate and results in an implementation loss of more than $2\,dB$. A high max peak error rate will lead to a *longer* INIT SYNC acquisition time. The peak detector algorithm (see implementation 3.2.2.5) will store only the N highest peaks within a small window. Therefore the probability that the correct peak (where the frame starts) is among those N peaks reduces drastically at low SNRs due to this high implementation loss. A higher peak error rate may also increase the probability of false INIT SYNCs. Wrong peaks may lead to

a false synchronization if the peak positions match and there are M consecutive matching peaks.
By using the approximation for the Norm 2 we get a much better performance and the implementation loss is below $0.5\,dB$. Therefore the Norm 2 approximation has also been used in the FPGA implementation.

6.1.3. Max-Peak Error Rate with Frequency Error

In this section we focus on the frequency sensitivity of the correlation algorithm. No frequency error ($\Delta f \cdot T_s = 0$) and a frequency error of 10% ($\Delta f \cdot T_s = 0.1$) have been tested for the ideal (Norm 2) calculation and the approximated Norm 2 calculation that has been used inside the FPGA. The resulting max-peak error rate can be seen in figure 6.6.

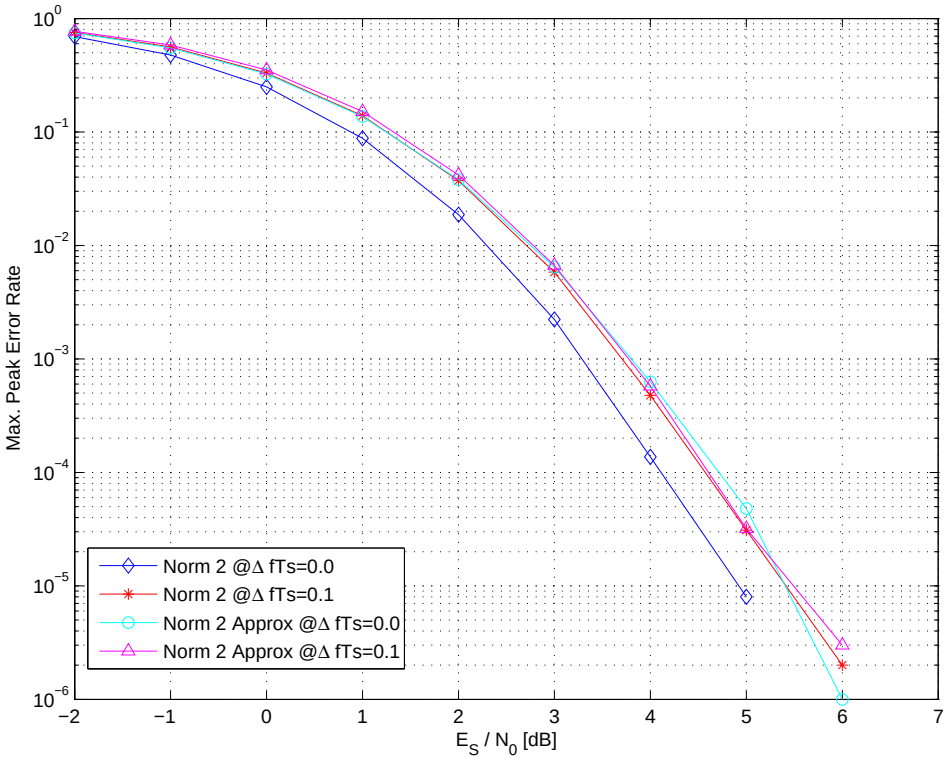

Figure 6.6.: Max-peak error rate for different frequency errors

The shown curves indicate that in the case of the Norm 2 ideal calculation a small frequency error leads to a small decrease in the performance when compared to the frequency error free error. When using the approximated Norm 2 calculation there is nearly no difference between both cases.
This also proves the shown frequency error immunity of the differential correlation that has been explained at the end of section 3.2.1.3.

6.2. INIT SYNC Acquisition Performance

The following section analyzes the performance of the peak search algorithm (adapted peak detector) in combination with the differential correlation. For this simulation the ideal norm (Norm 2) has been used to show the possible performance of the system. The chosen frame

format for this simulation is QPSK (normal frames) using pilots. This combination represents the biggest possible frame format (=33282 symbols within a frame). Therefore the simulated acquisition times represent the worst-case scenario due to the design of the algorithm. For shorter framelengths the acquisition time will reduce. Figure 6.7 shows the simulation graph for the acquisition time. Each time a INIT SYNC is found the module (`peak_detect_by_decoding_w_reset`) is reset to the initial conditions and will count how many search windows are needed until the next INIT SYNC is found. Within this module also the number of false INIT SYNCs will be counted. An additional module (`sim_plsc_decode`) will evaluate how many PLSC fields have been decoded wrong (decoding failure)

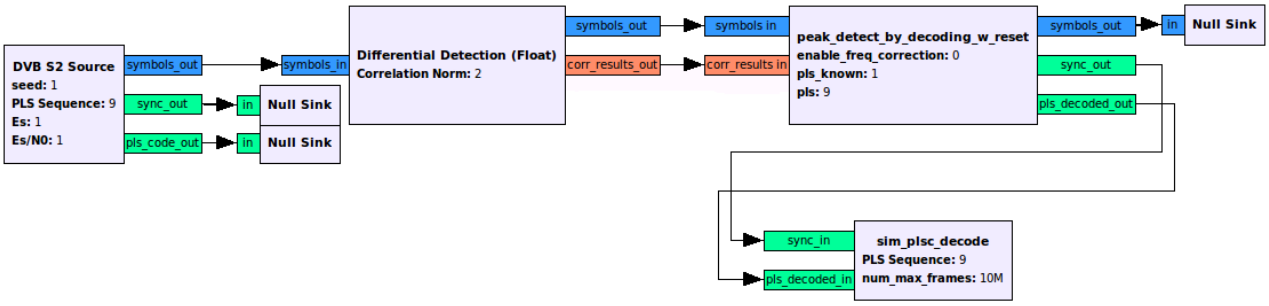

Figure 6.7.: Simulation of the INIT SYNC acquisition time in GRC

6.2.1. Acquisition Time

We can model the acquisition process as a random experiment were the random variable X represents the number of search windows needed until a successful INIT SYNC is achieved. This leads to the empirical determined Probability Density Function (PDF) $p(X = k)$ (shown in figure 6.8 for example).
Further we can define the cumulative distribution function (CDF) as the probability of finding the INIT SYNC in less or equal then k search windows:

$$p(X \leq k) = \sum_{i=0}^{k} p(X = i)$$

This allows us to determine also a mean acquisition time defined as:

$$E(X) = \sum_{k=0}^{\infty} p(X = k) \cdot k$$

To get the final mean acquisition time T_{acq} in seconds we can calculate the average number of search windows times the symbols per search window($= 30 * 3330\, Symbols$ chosen) times the inverse baudrate:

$$T_{acq} = E(X) \cdot 30 \cdot 3330 \cdot \frac{1}{Baudrate}$$

Additionally we define a time with $\alpha = 99.5\%$ confidence and $\alpha = 99.9\%$ confidence to acquire frame synchronization. This is essentially the number k of search windows where the CDF gets above this confidence value (inverse CDF).

$$k : P(X \leq k) \geq \alpha$$

6.2.2. Simulation Results

Table 6.1 shows the simulation results for different SNR values. Since the simulations require a lot of time we reduced the possible values of $\frac{E_s}{N_0}$ to a few points.

A histogram of the acquisition time has been generated by counting the number of search windows (SW) between successive INIT SYNCs.

$\left(\frac{E_s}{N_0}\right)_{dB}$	Mean Acquisition Time	99.5% confidence	99.9% confidence	P(False INIT Sync)	P(False decoded PLSC)
-2 dB	4.39 SW	21 SW	27 SW	6.4261e-005	2.25066e-006
0 dB	1.20 SW	3 SW	4 SW	1.3064e-005	QEF*
1 dB	1.03 SW	2SW	3 SW	3.3389e-006	QEF*
3 dB	1 SW	1 SW	1 SW	QEF*	QEF*

*QEF (Quasi Error Free) means that within the simulation time (10 Mio Frames) no errors occured

Table 6.1.: Peak detector performance

A false INIT SYNC means that a frame synchronization has been achieved by the module but the position is not the correct one. For the chosen algorithm this might happen if coincidentally 3 consecutive peaks match. In the ideal case (symbol timing locked, no PLSC decoding errors) the INIT SYNC just has to be found once and it can be maintained through the frame by frame decoding process explained in 5.3.4.2.

The PLSC decoding performance shows that there is only one entry in the table. In the required experiment range of $1dB$ to $13dB$ the decoding error is QEF. Simulations have shown that by using soft bits instead of hard decision when decoding the PLSC (section 3.3.4) an additional gain of $2\,dB$ is possible.

Figure 6.8 shows the empirical determined PDF and CDF of the search windows needed until acquisition at an SNR of $-2dB$. We can see that the amount of the search windows needed has an exponential decay.

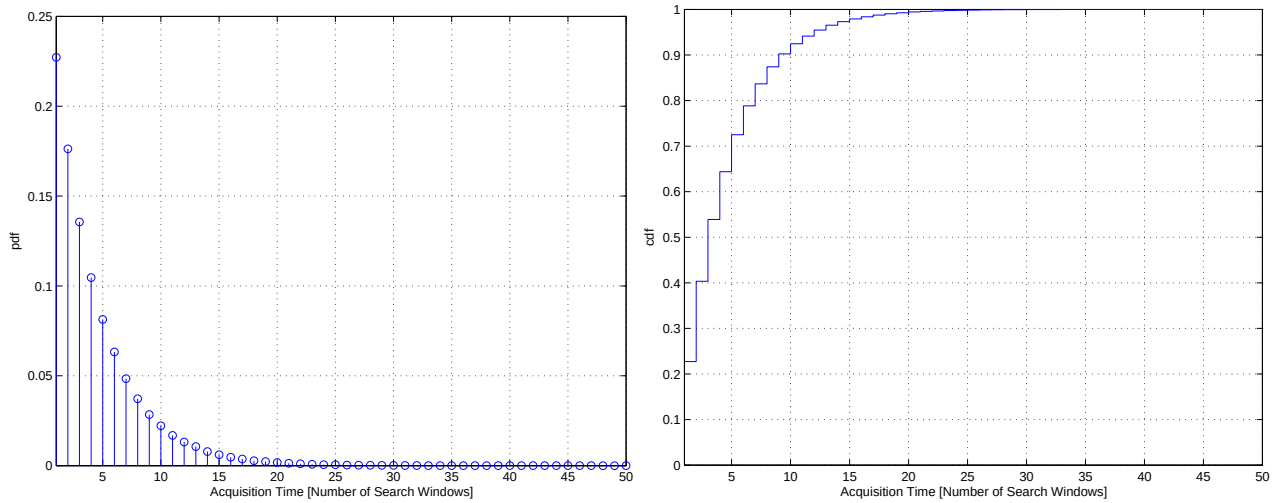

Figure 6.8.: Empirical PDF and CDF of the acquisition Time at $\frac{E_s}{N_0} = -2dB$

6.3. SNR Estimation

6.3.1. Performance Measurements

One important figure is the mean estimator output. It is determined by simply averaging over all SNR estimates $\hat{\rho}_{Frame_k}$ (which are calculated over one frame).

$$Mean(\hat{\rho}) = \frac{1}{N_{Frames}} \sum_{k=0}^{N_{Frames}-1} \hat{\rho}_{Frame_k}$$

The bias may now be calculated by simply determining the deviation of the mean estimator output of the real SNR value:

$$bias(\hat{\rho}) = Mean(\hat{\rho}) - \rho$$

Another important figure of merit is the mean squared error (MSE). The MSE has been chosen because it accounts both the variance and the squared bias of the estimator. It is determined by calculating the following sum:

$$MSE(\hat{\rho}) = \frac{1}{N_{Frames}} \sum_{k=0}^{N_{Frames}-1} \left(\hat{\rho}_{Frame_k} - \rho\right)^2$$

The normalized MSE is then determined by dividing the MSE by the squared value of the true SNR value ρ^2:

$$NMSE(\hat{\rho}) = \frac{MSE(\hat{\rho})}{\rho^2}$$

The NMSE can now be compared to the NCRLB to evaluate how far away the current estimator is from the optimum performance.

6.3.2. DA Estimation

The estimator block is directly attached to the DVB-S2 source block. Each time the sync input of the SNR estimator block is triggered it will start to perform its operation. When a whole frame has passed the SNR estimator will output two estimates (see section 4.3.1.2):

- One estimate is the SNR estimate only calculated on the PLHEADER ($\hat{\rho}_{PLH}$)
- The other estimate will provide a combined estimate (PLHEADER + pilot blocks) ($\hat{\rho}$)

A frame format with PLS = 5 (QPSK normal frames with pilots) has been chosen for the simulation because it contains the most pilot blocks. There are two options to calculate the SNR estimate which have been simulated and compared:

- Practical case: carrier phase offset is not known and has to be corrected
- Simulation case: carrier phase ideally known and corrected

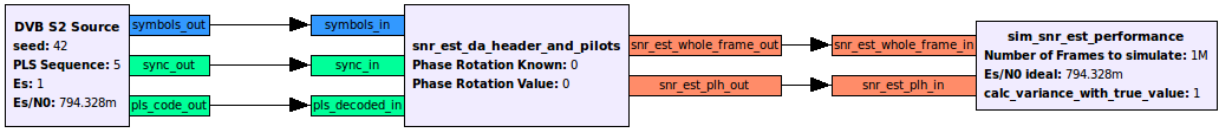

Figure 6.9.: SNR estimation in GRC

The last block in the simulation chain determines the normalized MSE of the SNR output.

6.3.2.1. Mean Estimator Output Simulation

The mean estimator output has been plotted in figures 6.10 and 6.11. In the first figure we can see that all estimates have a quite good bias performance and never induce a bias of more than $0.1\,dB$ over the whole simulated region.

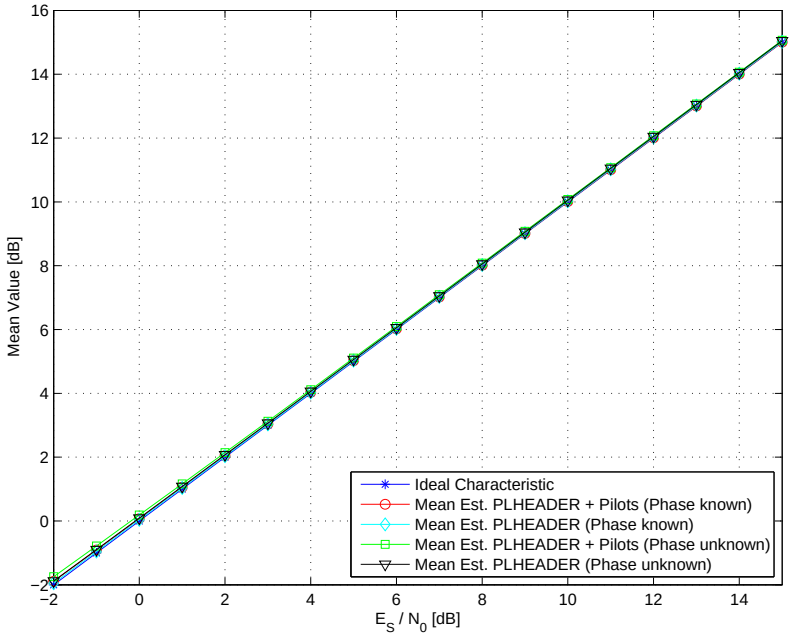

Figure 6.10.: Mean estimator output

In the following plot the mean estimator output has been zoomed in an range of about 6 dB to show details abut the behaviour. We can see that the "Phase known" cases in general have a lower bias and perform similarly well. When the phase is not known and estimated and corrected a small bias is added to the estimation.

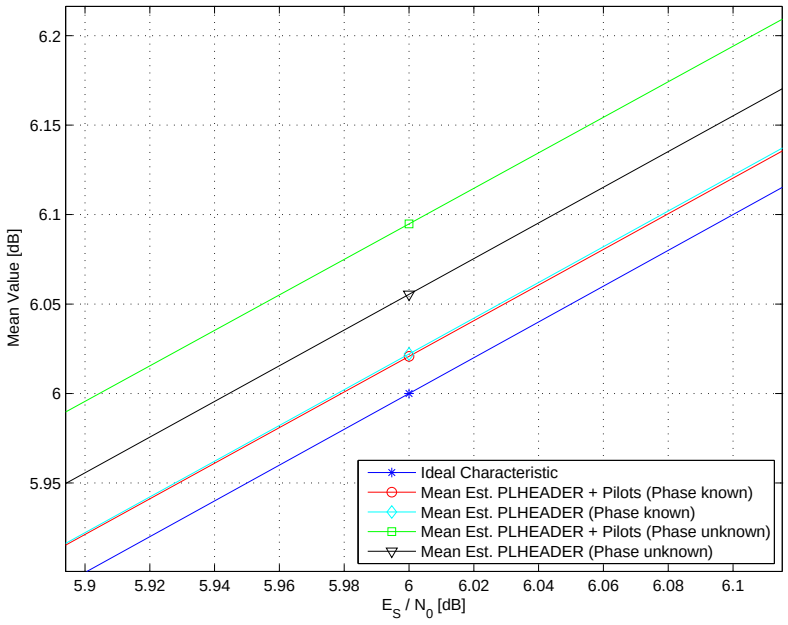

Figure 6.11.: Mean estimator output (zoomed at $6dB$)

6.3.2.2. NMSE Performance Simulation

The normalized Cramer-Rao Lower Bound has been plotted and the simulated NMSE results are compared to it (see figure 6.12). We can clearly see that for both cases (PLHEADER only, PLHEADER + pilots) the DA SNV estimators perform close to the CRLB. The curves for the "phase unknown" case has a slight offset in the PLHEADER only estimation. In the case of PLHEADER + pilots estimation, phase estimation and correction has a stronger effect. Additional uncertainty is introduced from the phase estimation which also influences the SNR estimation. That has already be seen in the mean estimator output where a small bias was induced. Since the NMSE also accounts the bias we can see it resulting in an offset here.

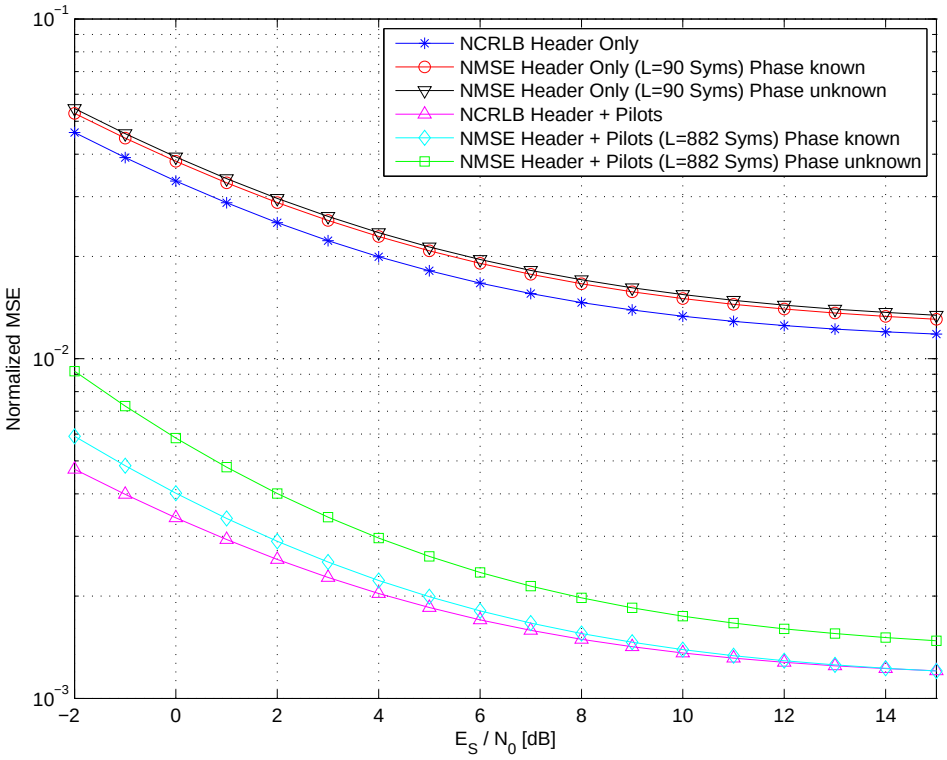

Figure 6.12.: Normalized MSE

6.3.3. Moment-Based Estimation

The setup of the simulation graph is essentially the same as with the DA estimator. This time we can only get one SNR estimate on a whole frame. The following cases have been simulated:

- NDA estimation for constant envelope modulation (QPSK, 8-PSK):
  - estimation limited to $L_z = 1000$ symbols
- NDA estimation for non-constant envelope modulation (16-APSK, 32-APSK)
  - average signal energy calculated on whole frame
  - estimation limited to $L_z = 1000$ symbols that reach over the partition margin

Because we want to have a kind of fair comparison between all the different NDA estimators we limited the amount of symbols used to an equal amount of $L_z = 1000$ Symbols.

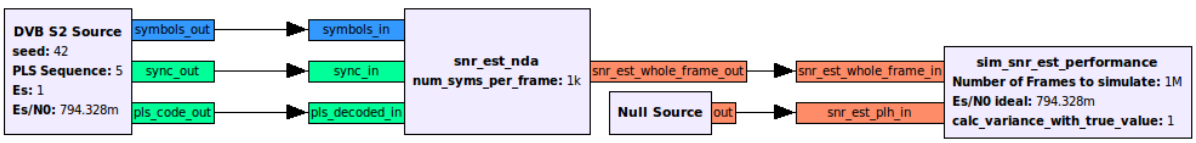

Figure 6.13.: Simulation of the NDA estimation in GRC

6.3.3.1. Mean Estimator Output Simulation

We can see the mean estimator output in the following plot. It can be clearly seen that the constant envelope (QPSK, 8-PSK) estimators perform quite well over the whole tested range and nearly induce no bias. We can see that the non-constant envelope modulations induce a high bias in the lower SNR range but that the estimation gets better with increasing SNR.

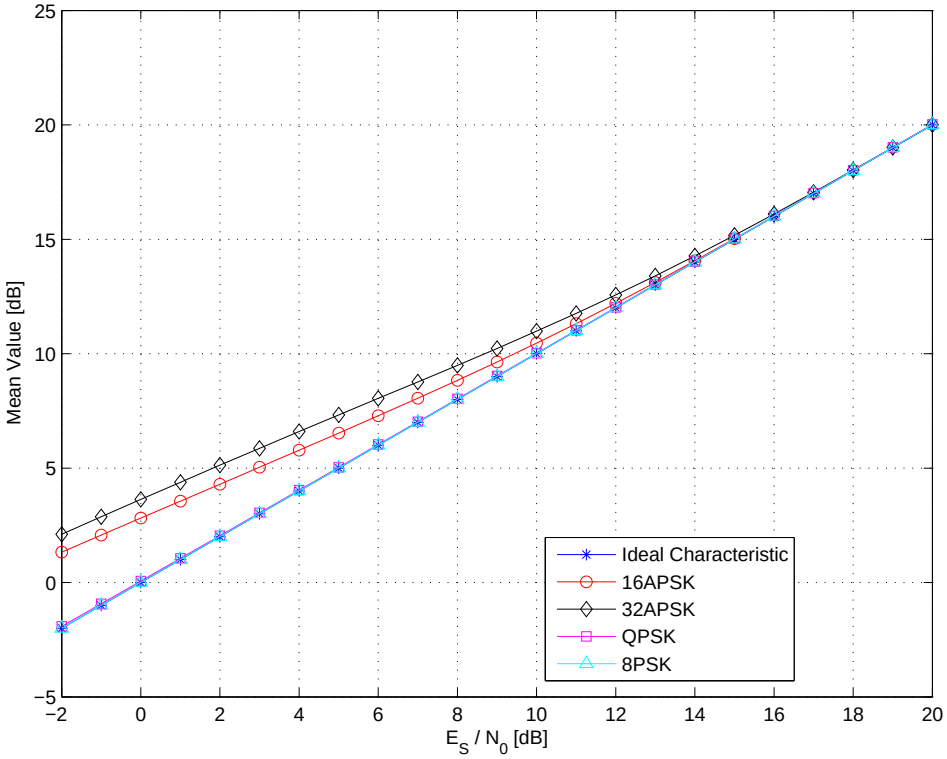

Figure 6.14.: Mean estimator output for different modulation schemes ($L = 1000$)

6.3.3.2. NMSE Performance Simulation

The bias problem can also be seen in the following plot which displays the simulated NMSE for all estimators (figure 6.15). Due to the high bias the non-constant envelope modulations have a quite high NMSE in the low SNR region. The 16-APSK case performs slightly better than the 32-APSK. This is due to the fact that 16-APSK has more symbols within a frame than 32-APSK. Therefore more symbols can be used for the estimation of the average signal power (which is used for determining the partition radius). For low SNR values generally the variance of all estimators is high. For higher SNR values we can see that they all reach an asymptotic NMSE floor value which is above the NCRLB. According to [BS67, GT05] this is a typical behaviour for envelope-based estimators.

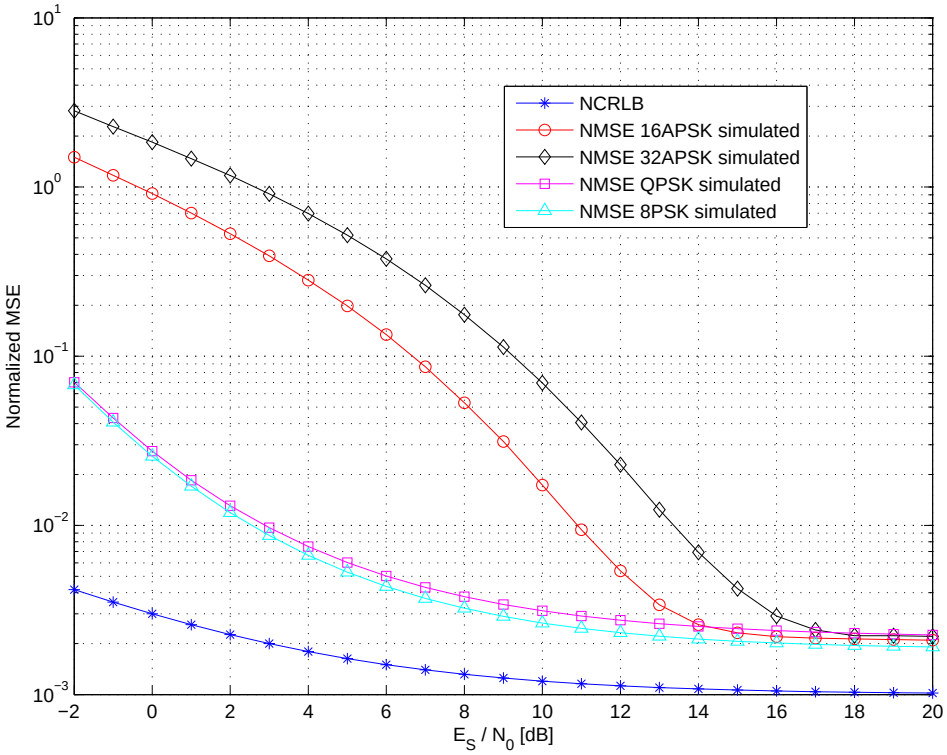

Figure 6.15.: NMSE for different modulation schemes ($L = 1000$)

7. Practical Measurements on the Prototype

In this chapter the measurement setup with the implemented prototype will be explained. We'll perform the same measurements as in the simulation and make a comparison.

7.1. Test Setup and Components

The following components have been used for the test setup (figure 7.1):

- Modulator / Demodulator: Newtec Elevation EL470
  - One modem is acting as modulator, the other as demodulator
- Noise Generator: Noisecom UFX7112A
- Spectrum Analyzer: Rhode & Schwarz FSV Signal Analyzer (9kHz - 40GHz)
- Logic Analyzer: Agilent 16702B Logic Analysis System
- Power Splitter 1: Mini Circuits ZFSC 2-11
- Power Splitter 2: Mini Circuits ZA4PD-2

APP PC1 generates the payload for the Newtec modem, modulating the data according to the DVB-S2 frame format and converting the spectrum up to 1 GHz. The programmable noise generator now attenuates the signal and adds the needed amount of noise.

Through the power splitters the signal is now distributed to the demodulator, the spectrum analyzer and the USRP N210 with the proposed prototype FPGA image. A PC with the GNU Radio blocks running on it now performs the needed signal processing (MODCOD determination, SNR estimation).

On the demodulator the DVB-S2 frames are decoded and send to the APP PC2 were the payload is further analyzed.

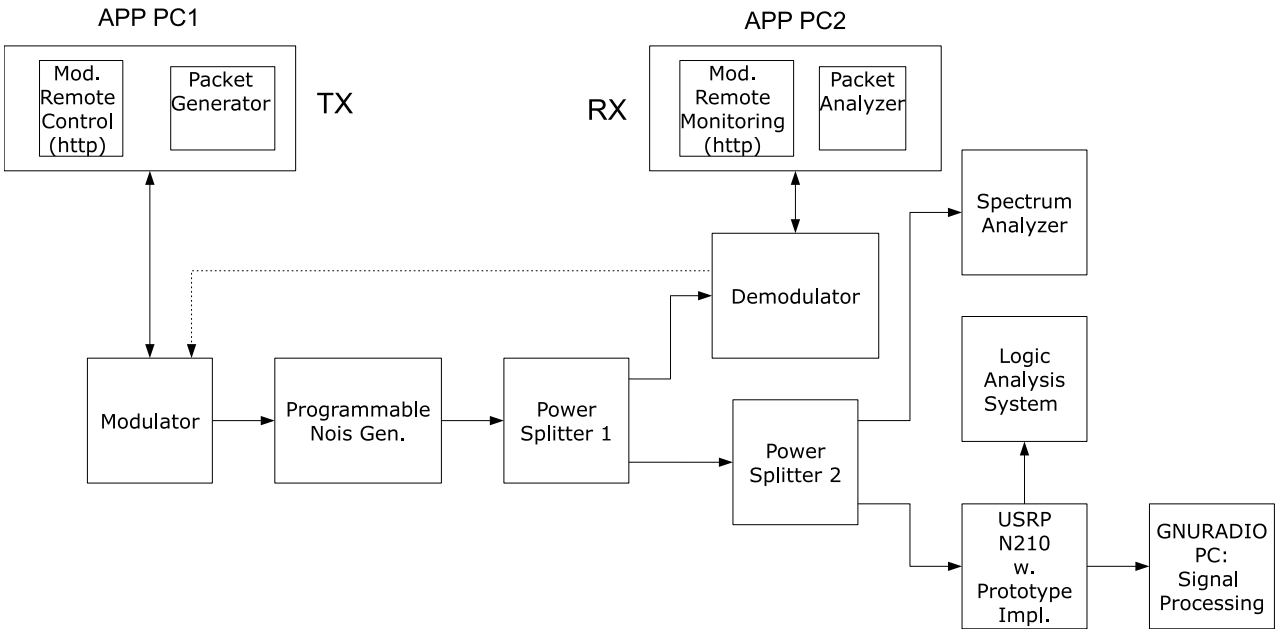

Figure 7.1.: Test setup in the laboratory

7.1.1. Modulator / Demodulator Remote Control

Both modems (figure7.2) offer a http-WebInterface that can be used to control and monitor the modems.

Figure 7.2.: Both Newtec modems on top of the noise generator

On the modulator Web-Interface all transmission relevant parameters (like the MODCOD, Symbol Rate, Output Power, etc) can be set (figure 7.3).

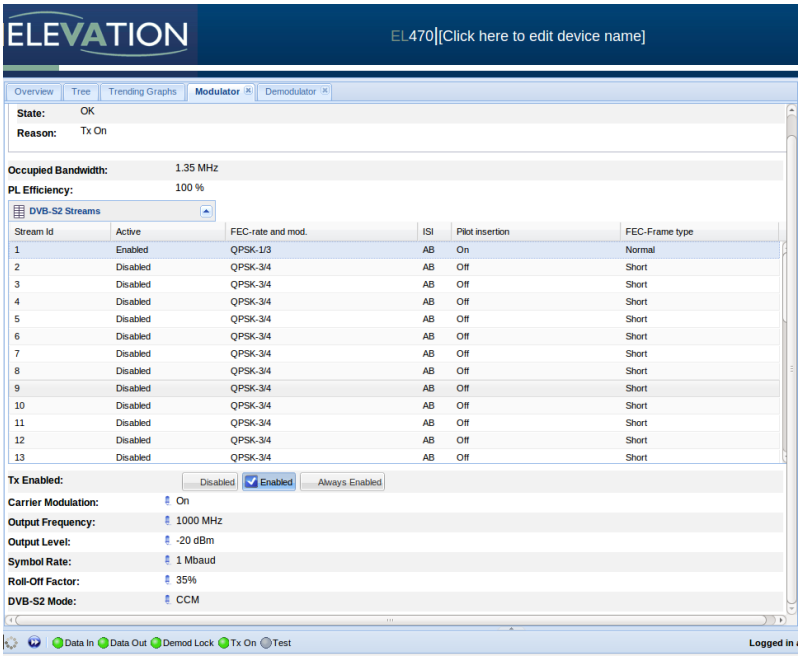

Figure 7.3.: Modulator setup

On the demodulator interface some important receiving figures can be seen (receive SNR, link margin, etc).

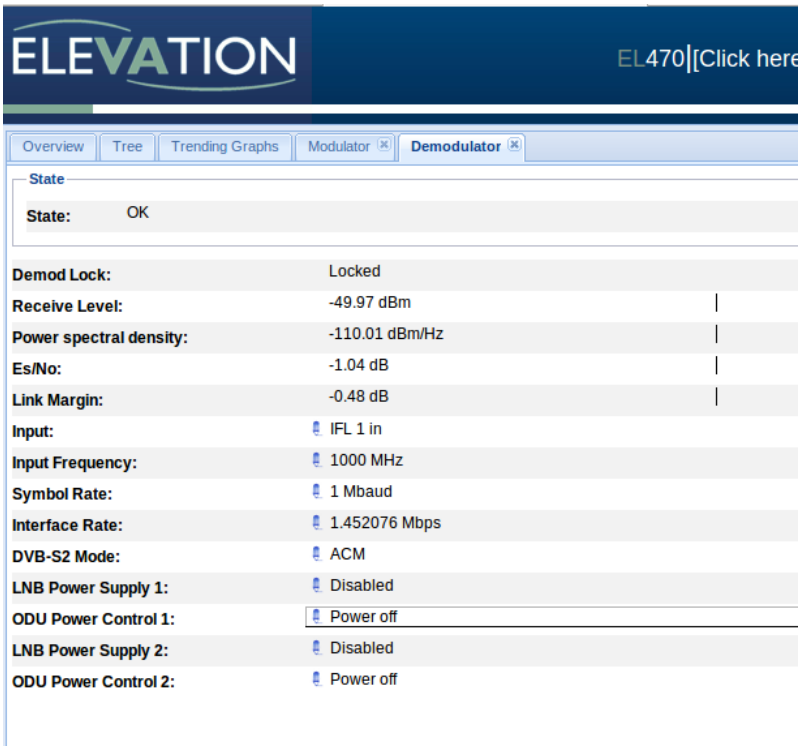

Figure 7.4.: Demodulator setup

7.1.2. Packet Generator and Analyzer

A packet generator running on the APP PC1 side generates the payload data (UDP packet data) and sends it to the receiver over the DVB-S2 channel. The received data is now analyzed by the APP PC2 where a packet analyzer is running.

The generated UDP packets are first encapsulated into IP datagrams. If the modulator is configured in bridge mode then the datagrams are encapsulated into Ethernet frames. The modulator encapsulator now embedds this Ethernet frame into a DVB-S2 frame where the payload section represents the Ethernet frame. The overview of this setup can be seen in figure 7.5).

Additionally there is a backlink from the demodulator to the modulator. Since the modulator is configured in the bridge mode we need to know the MAC address of the receiver (APP PC2) and this is done by ARP (Address Resolution Protocol). Therefore the backlink provides a way to send back an ARP reply from APP PC2 to APP PC1.

One important setting on APP PC1 is to generate enough data with the packet generator within a certain time since the modem will otherwise start to insert so called DUMMY frames. We want to have a constant MODCOD for our sent data and no DUMMY frames. The capacity utilization can be determined in the modulator overview when looking at the efficiency field. It needs to be 100% so that no DUMMY frames get inserted.

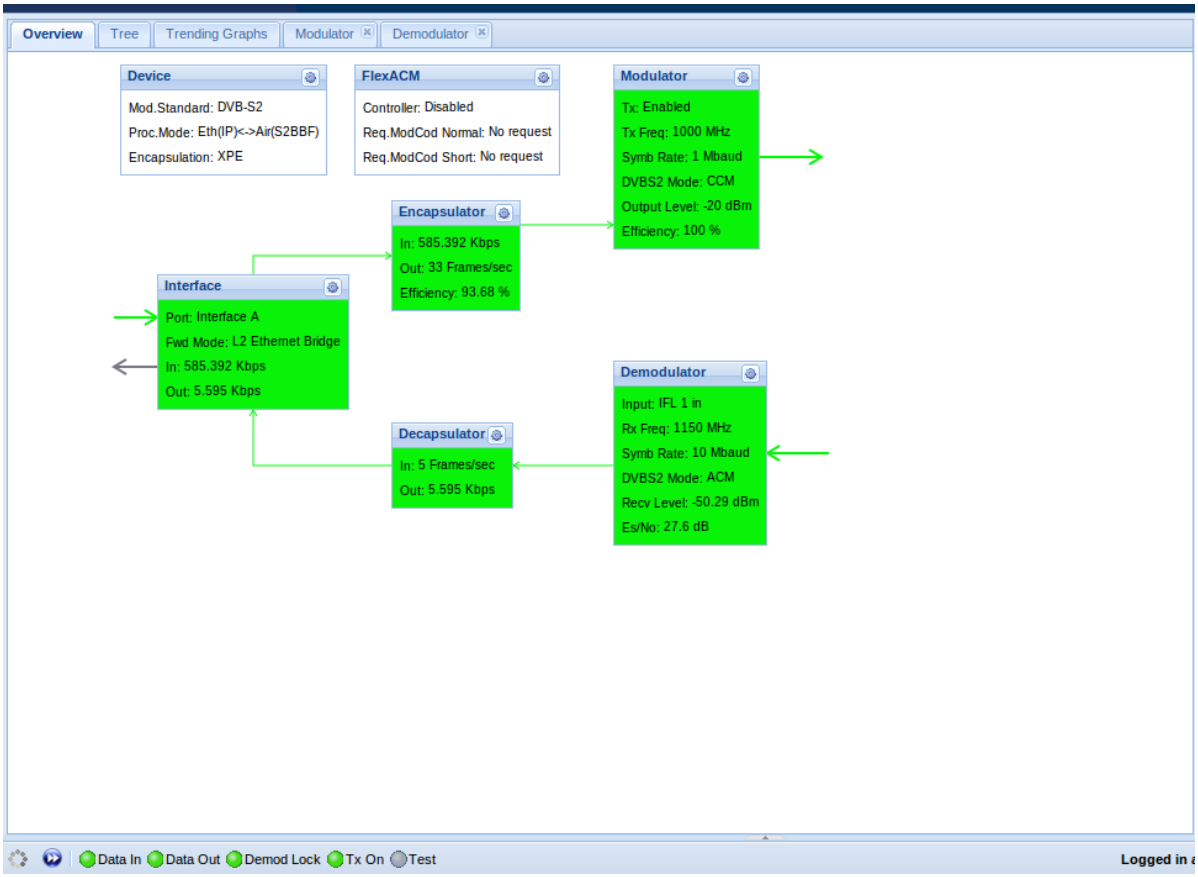

Figure 7.5.: Modulator overview setup

7.1.3. Noise Generator

As shown in figure 7.6 the noise generator allows to set a certain signal attenuation on the input signal and allows to turn off or on a noise signal with a certain attenuation in respect to the maximal noise output power.

Figure 7.6.: Settings on the noise generator

One problem is that the noise generator produces a none flat spectrum with a maximum flatness of $\pm 2dB$ [Noi] which has also been verified by measurements (see figure 7.7). Therefore the measurement of the carrier-to-noise ratio is influenced by this circumstance. The higher the symbol rate gets, the worse the produced error of the noise generator will be.

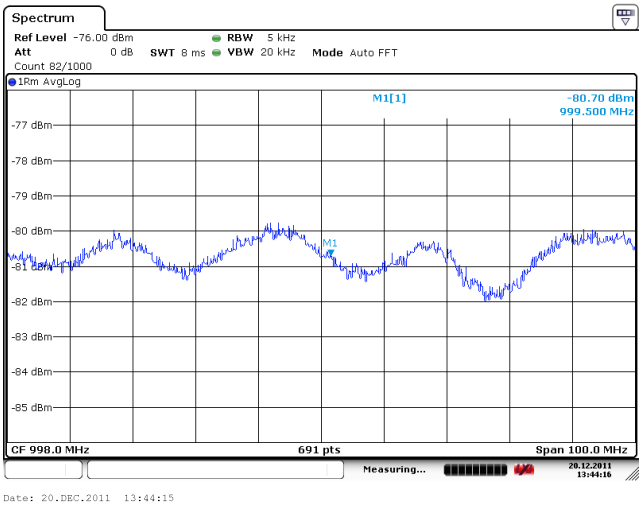

Figure 7.7.: None-flatness of the noise generator in a 100 MHz band

7.1.4. SNR Measurement on the Spectrum Analyzer

A good way to measure the carrier-to-noise ratio is to measure both powers in the same frequency band. This is done by first turning on the carrier in the modem and determining

the carrier-plus-noise power within the bandwidth area where the carrier is flat (figure 7.8). Afterwards the carrier is turned off at the modem and the noise power is measured in the same area (figure 7.9). The difference between both power levels determines the Carrier -plus-Noise to Noise ratio:

$$\left(\frac{C+N}{N}\right)_{dB}$$

To determine the real CNR we need to perform the following operation:

$$\left(\frac{C}{N}\right)_{dB} = 10 \cdot log\left(10^{\frac{\left(\frac{C+N}{N}\right)_{dB}}{10}} - 1\right)$$

Especially at low CNRs ($< 5dB$) this calculation is very sensitive towards jittering decimal places of the power measurements. Therefore the CNR measurements at the spectrum analyzer can only give a rough estimation ($\pm 0.3\,dB$) for the real CNR.

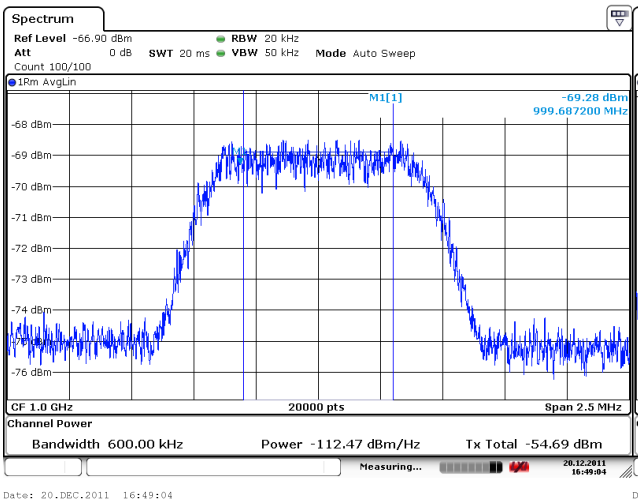

Figure 7.8.: Carrier-plus-noise power measurement

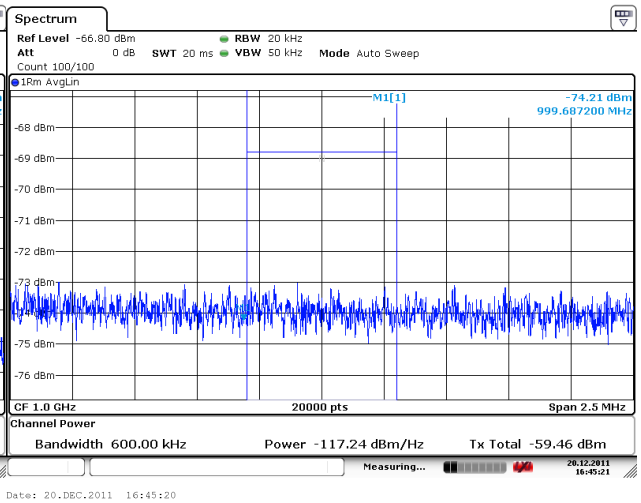

Figure 7.9.: Noise power measurement

7.1.5. Optimal Signal Level for ADC

The WBX board does not include an Automatic Gain Control (AGC) therefore a manual setting of the noise generator settings and the amplifier gain is necessary to prevent the ADC from clipping. Clipping would mean an unwanted signal distortion. When adding noise to the signal it should also be considered that the added noise should not drive the ADC too much into saturation. Otherwise the clipping would take away some signal-plus-noise power and therefore the SNR estimation would become inaccurate. Figures 7.10 and 7.11 show how the ADC range was determined on the logic analysis system (for further details have a look in appendix B.2).
Additionally we should take care that the input of the correlator after the decimation (CIC, HBF, Interpolator) is resolved with enough bits. Since we are going to cut and round bits after determining the differential we need to have at least approx. 8 bits (shown in a simulation) .

To determine the resolution we may write the received samples from the USRP to a file and determine with MATLAB the average power of the signal. And how many bits are effectively used out of the RMS value.

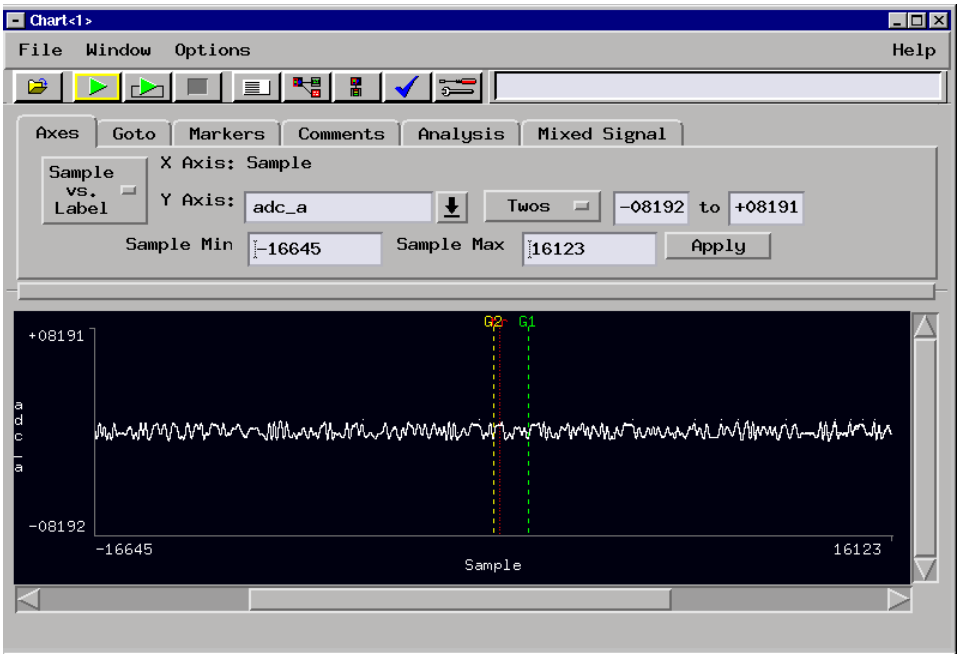

Figure 7.10.: Input samples without Noise

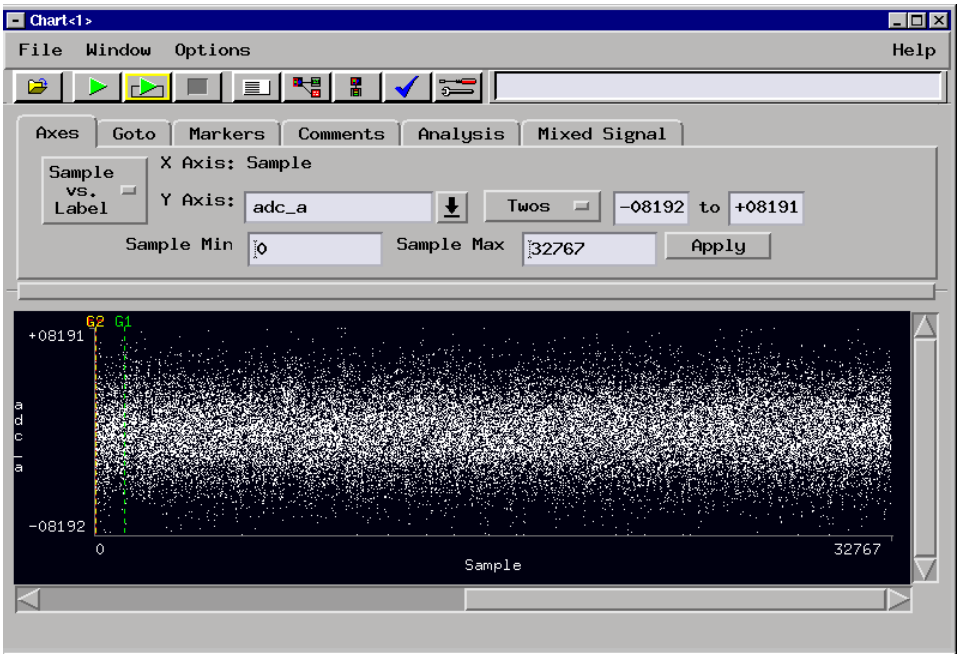

Figure 7.11.: Input samples contaminated with noise ($\frac{C}{N} = 1dB$)

7.1.6. GNU Radio Setup for the Measurements

The following GRC graph shows the setup during measurements on the GNU Radio PC. The UHD USRP source delivers the samples that are sent from the USRP through Ethernet to

the GNU Radio environment.

On the upper channel the optimally sampled symbols (1 Sample / Symbol) are available. The channel transmits two 16 bit signed integers as a combined vector. Since the peak detection and the other modules are performing with floating points we need to convert the samples to a complex float format. This is shown in the figure as the chain `Vector to Streams` => `Short To Float` => `Float To Complex`. All those conversion blocks are already provided by the standard GNU Radio toolkit.

The other channel transmits the correlation results that have been determined in the FPGA. Since the correlation result has a maximum bit width of 22 we needed to send them on two separate 16 bit integers. The following custom block (`short_vec_to_float`) performs the shifting and correct conversion of the incoming samples into a floating point format.

Both converted channels are now fed into the peak detector which produces SYNC signals at the beginning of each frame. Additionally within this module an analysis of the INIT SYNC acquisition performance is performed by calculating repeated INIT SYNCs in the background and writing the number of needed search windows into a file. Additionally the number of false INIT syncs is recorded by comparing the decoded PLS field with the known (since the modulator is kept fixed to a certain modulation scheme) PLS field.

The DA and NDA SNR estimation are now triggered by the generated SYNC signals and estimate the $\frac{E_s}{N_0}$ according to the implemented algorithms. The estimated values are logged in a text-file. After the measurements are finished all log-files can be post-processed by MATLAB.

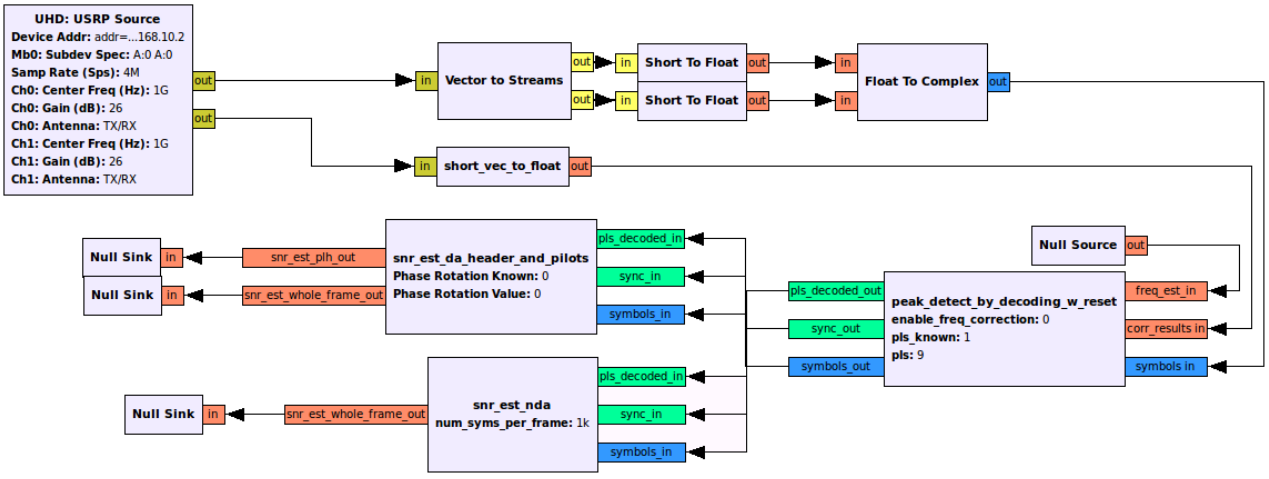

Figure 7.12.: GRC graph setup for measurements

7.2. Test Cases

Now different test cases have been defined in table 7.1. They cover nearly the needed dynamic range of $12dB$ that is used in the ACM experiment. Since the SNR measurements on the spectrum analyzer are not very reliable (especially in case of low SNRs) the test cases are denoted with the targeted SNR. All tests have been performed using a symbol rate of $f_s = 1\,MSym/s$ since in the according bandwidth the noise generator produces a nearly flat spectrum.

Settings	QPSK@ $\frac{C}{N} \approx -1\,dB$	QPSK@ $\frac{C}{N} \approx 1\,dB$	QPSK@ $\frac{C}{N} \approx 3\,dB$	QPSK @ $\frac{C}{N} \approx 5\,dB$	8 PSK @ $\frac{C}{N} \approx 7\,dB$	16-APSK@ $\frac{C}{N} \approx 10\,dB$
Modem Output Power	$-20\,dBm$	$-20\,dBm$	$-20\,dBm$	$-20dBm$	$-20\,dBm$	$-20\,dBm$
NG*: Noise Attenuation	$1\,dB$	$3\,dB$	$3\,dB$	$3\,dB$	$3\,dB$	$3\,dB$
NG*: Signal Attenuation	$11.2\,dB$	$11.2\,dB$	$8.9\,dB$	$7.3\,dB$	$5.3\,dB$	$3\,dB$
USRP: Gain	$26\,dB$	$26\,dB$	$26\,dB$	$26\,dB$	$26\,dB$	$26\,dB$
SA+ measured $\frac{C+N}{N}$	$2.44\,dB$	$3.55\,dB$	$4.8\,dB$	$6.25\,dB$	$7.75\,dB$	$10.45\,dB$
SA+ measured $\frac{C}{N}$	$-1.227\,dB$	$1.02\,dB$	$3.05\,dB$	$5.07\,dB$	$6.95\,dB$	$10.04\,dB$

* NG = Noise Generator
+ SA = Spectrum Analyzer

Table 7.1.: Test cases

7.3. INIT Sync Acquisition Performance

Measures of the acquisition time have been performed in the -1 dB to 5dB range for QPSK normal frames using pilots (PLS = 0x09). This frame format was chosen because it has the longest frame length (=33282 Symbols) and displays the worst case in terms of acquisition time. The same performance measurements (mean acquisition time, 99.5% and 99.9% confidence acquisition time) as in the simulation (see section 6.2) have been performed. The two other test cases (8-PSK, 16-APSK) have a shorter framelength (=>shorter acquisition time) and therefore would not allow a fair comparison. All acquisition times are expressed in Search Windows (SW) where one search window is $1\,SW = 30 \cdot 3330 = 99900$ Symbols.

Test case	Mean Acq. Time	99.5% Acq. Time	99.9% Acq. Time	P(False INIT Sync)	#Search Windows
QPSK@ $\frac{C}{N} \approx -1\,dB$	6.92 SW	35 SW	46 SW	1.480e-003	962728
QPSK@ $\frac{C}{N} \approx 1\,dB$	1.70 SW	6 SW	8 SW	4.1708e-004	167228
QPSK@ $\frac{C}{N}{}_{dB} \approx 3\,dB$	1 SW	1 SW	2 SW	2.0052e-004	628171
QPSK @ $\frac{C}{N} \approx 5\,dB$	1 SW	1 SW	1 SW	2.1872e-004	169558

Table 7.2.: Frame acquisition measurement results

The two figures (7.13 and 7.14) show the empirically determined PDF and CDF of the INIT SYNC acquisition time at two test cases $\frac{C}{N} \approx -1dB$ and $\frac{C}{N} \approx +1dB$. We can see that in both cases the empirical PDF and CDF have the same characteristics as shown in the simulation of section 6.2. The PDFs show an exponential decay of the acquistion time.

As we compare the measurements with the simulations we can see that the acquistion process in the real system is taking slightly longer. This is on the one hand due to the SNR degradation caused by the implementation loss of the CIC filters and digital interpolator (through jittering recovery algorithm which has a limited resolution and limited estimation length on the FPGA). On the other hand the use of the approximated Norm 2 inside the FPGA introduces an additional degradation in the Max-peak error rate curve as explained in section 6.1.2.

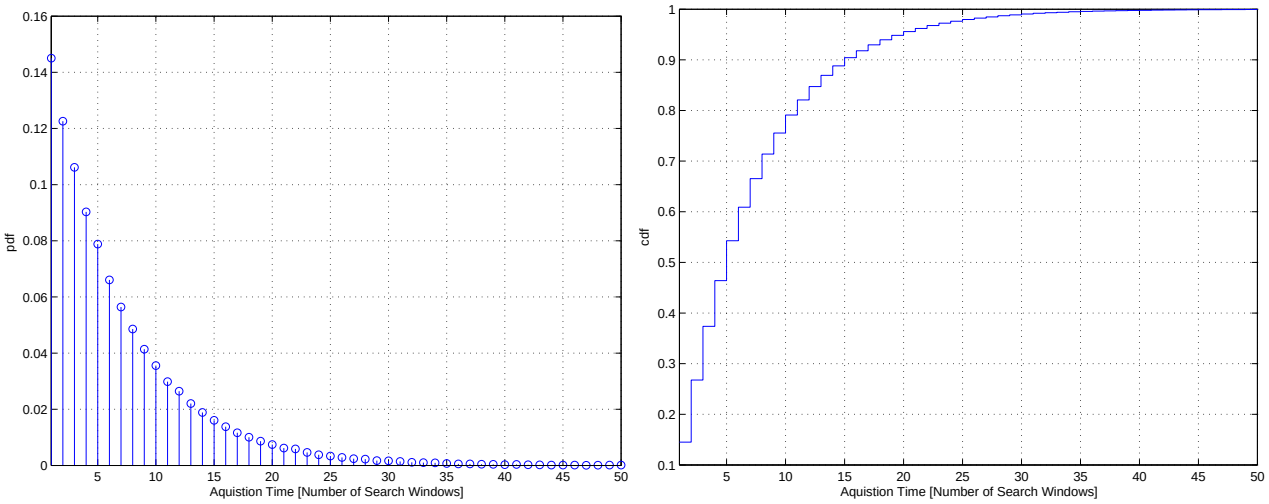

Figure 7.13.: empirical PDF (left) and CDF (right) of the acquisition time at $\frac{C}{N} \approx -1dB$

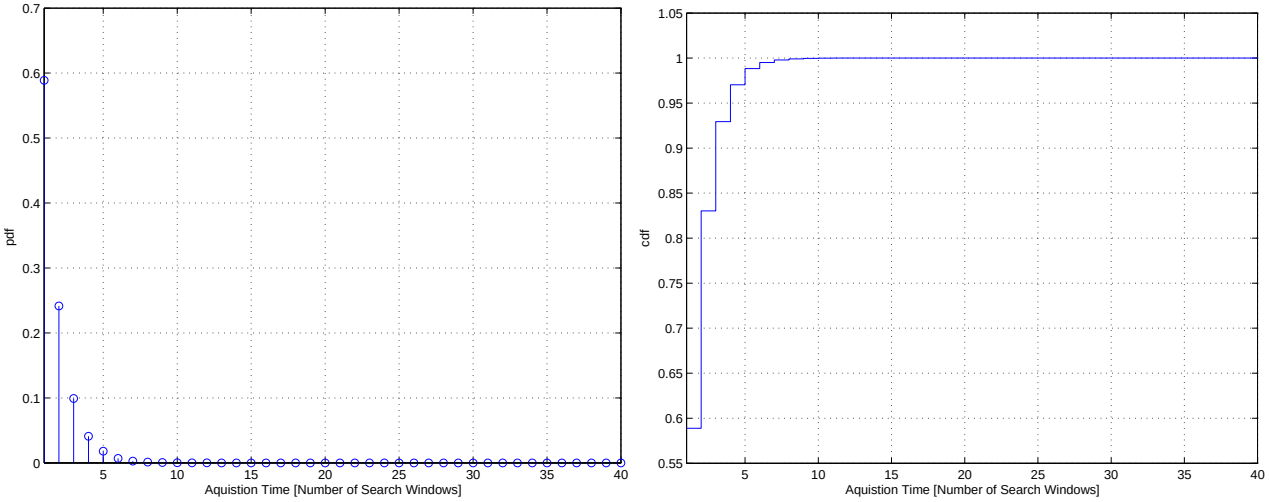

Figure 7.14.: empirical PDF (left) and CDF (right) of the acquisition time at $\frac{C}{N} \approx +1dB$

7.4. SNR Measurements

The following three estimations were performed in the GNU Radio environment for all test cases and the results written to a file:

- DA estimation only on PLHEADER
- DA estimation on PLHEADER + PILOT blocks
- NDA estimation on whole frame

7.4.1. Empirical PDF

The measured SNR estimates are read in from the log-files with MATLAB and post-processed. Out of this data we can generate a histogram of the estimator values to evaluate the estimator

distribution. An exemplary empirical determined PDF is shown in figure 7.15. We can see that the estimates are Gaussian distributed. The wider the Gaussian estimator PDF is the bigger the variance will be.
The DA estimator (PLHEADER only) has the biggest variance because it uses the lowest amount of symbols (90 symbols). The DA SNR estimator on PLHEADER and pilots (882 symbols) has a much lower variance even when compared to the NDA estimator variance. Although the NDA estimator uses more symbols (1000 symbols) for the estimation it has a higher jitter variance (compare to the simulation).

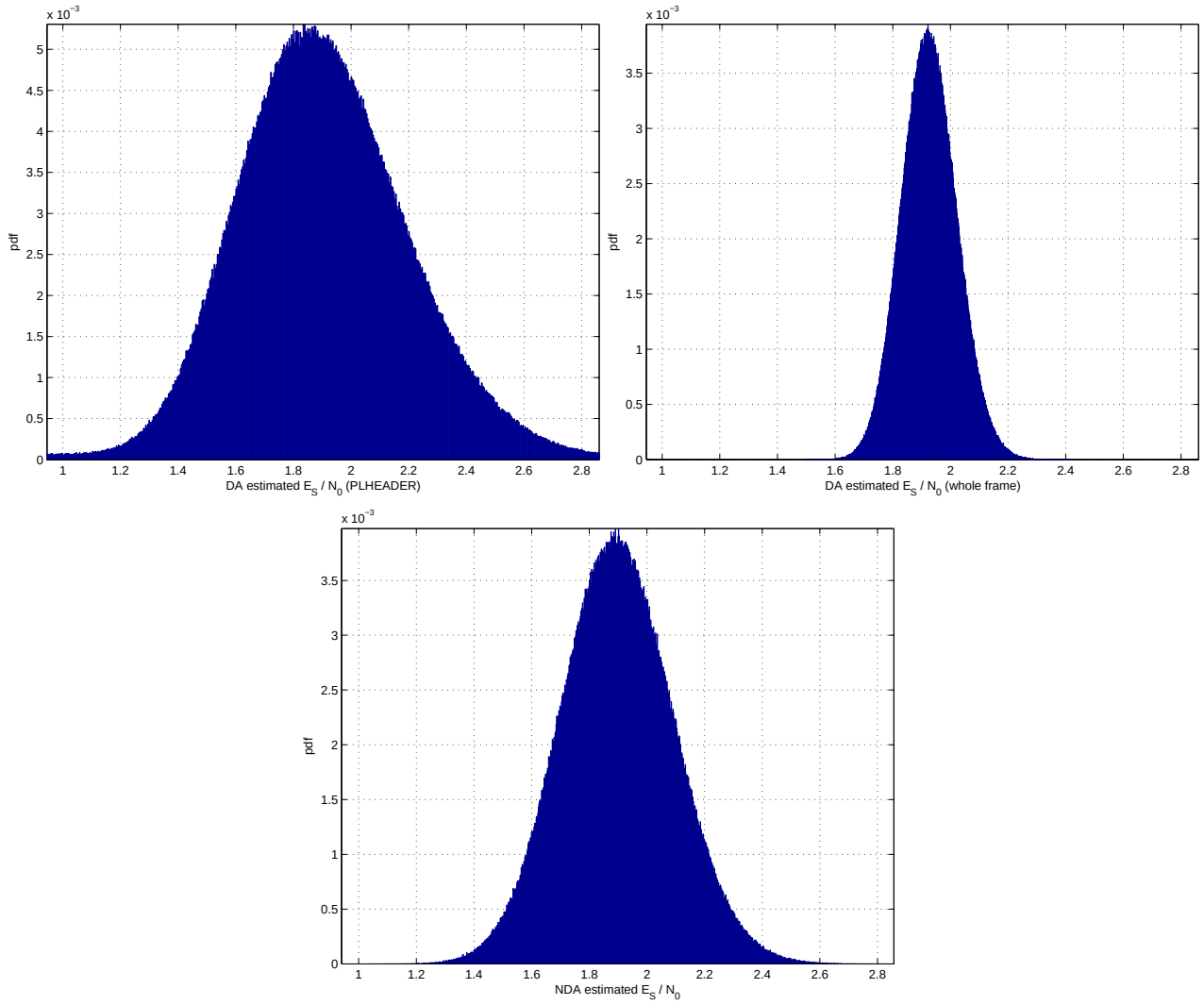

Figure 7.15.: empirical determined PDFs at $\frac{C}{N} \approx 3dB$

7.4.2. Measurement Results

In table 7.3 we can see the results of our measurements.that One problem is that we do not know the exact SNR. The measured $\frac{C}{N}$ on the Spectrum Analyzer are only approximate values ($\pm 0.3\,dB$).
Overall there is an implementation loss of our SNR estimates when compared to the measured SNR on the spectrum analyzer. The implementation loss is again caused by the digital signal processing chain inside the FPGA (that is the CIC filter and non ideal sampling (caused by limited FPGA resources) in the timing recover algorithm).

SNR Est	QPSK@ $\frac{C}{N} \approx -1dB$	QPSK@ $\frac{C}{N} \approx 1\,dB$	QPSK@ $\frac{C}{N} \approx 3\,dB$	QPSK @ $\frac{C}{N} \approx 5\,dB$	8-PSK @ $\frac{C}{N} \approx 7\,dB$	16-APSK@ $\frac{C}{N} \approx 10\,dB$
Spec. Analyzer [dB]	-1.2	1.02	3.05	5.07	6.95	10.04
Mean PLH [dB]	-1.36	0.33	2.79	3.90	6.29	8.91
MSE PLH	0.024	0.041571	0.091755	0.343095	0.417481	0.918290
Mean Whole Frame [dB]*	-1.24	0.4140	2.8486	4.0465	6.2719	8.9175
MSE Whole Frame*	0.002559	0.0043	0.0092	0.0341	0.0547	0.1694
Number Frames	2886608	501963	1885518	508858	2791090	1258604
Mean NDA [dB]+	-1.38	0.3334	2.7844	4.5191	6.2312	9.5004
MSE NDA	0.027805	0.0281	0.0374	0.0541	0.1020	0.2514
Number Frames	2885315	501964	1885519	508860	2791161	1258604

*not equal amount of PILOT Blocks in QPSK, 8-PSK and 16-APSK
+ different algorithms in QPSK, 8-PSK, 16-APSK

Table 7.3.: Summary of the SNR measurement results

7.4.3. Comparison to CRLB and Simulation

To measure the performance of the estimator first the mean value $\bar{\hat{\rho}}$ is determined by:

$$\bar{\hat{\rho}} = Mean(\hat{\rho}) = \frac{1}{N_{Frames}} \sum_{k=0}^{N_{Frames}-1} \hat{\rho}_{Frame_k}$$

Like in the simulation we determine the normalized MSE of the estimator but this time due to lack of knowledge we can not use the the real value for ρ. The assumption that the measured SNR is bias free can be made because the used algorithm is nearly bias-free as seen in the simulation. Therefore it is assumed that the mean value $\bar{\hat{\rho}}$ is the SNR that is really present in the digital system (after all implementation losses due to the digital signal processing) $\rho \approx \bar{\hat{\rho}}$. The MSE is now calculated by (remark: if the bias = 0 then the MSE = variance):

$$MSE(\hat{\rho}) = \frac{1}{N_{Frames}} \sum_{k=0}^{N_{Frames}-1} \left(\hat{\rho}_{Frame_k} - \rho\right)^2 \overset{\rho \approx \bar{\hat{\rho}}}{=} \frac{1}{N_{Frames}} \sum_{k=0}^{N_{Frames}-1} \left(\hat{\rho}_{Frame_k} - \bar{\hat{\rho}}\right)^2$$

And finally the NMSE is given by:

$$NMSE(\hat{\rho}) = \frac{MSE(\hat{\rho})}{\bar{\hat{\rho}}^2}$$

Here only the DA SNR Estimator on the PLHEADER has been considered. This is due to the fact that this is the only estimator that uses the same amount of symbols in every test case (regardless of which modulation is actually used). Figure 7.16 shows the measured NMSE and in comparison the simulated NMSE and the normalized CRLB. We can see that they match pretty good.

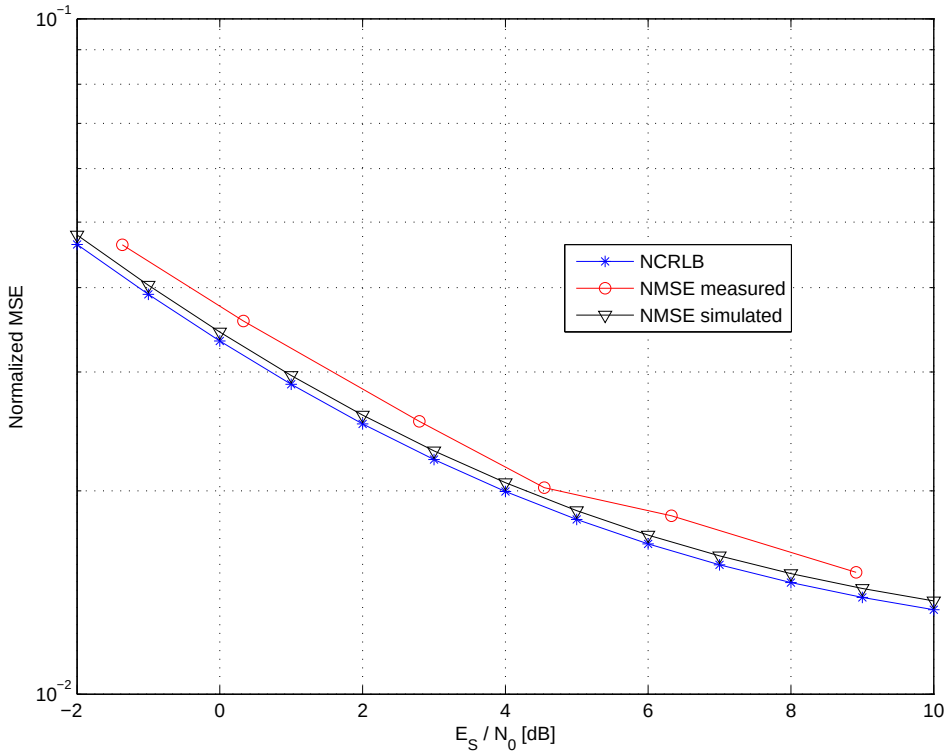

Figure 7.16.: DA estimator (on PLHEADER only $L = 90$) performance

7.5. Real-time Performance Tests

Additional tests have been performed to test the real-time performance at higher symbol rates. The maximum achievable symbol rate at the current configuration is $6.25\,MHz$. This comes from the minimal possible CIC & HBF combination decimation factor being 4. The downsampling rate of the digital interpolator is fixed to 4. Therefore the maximal symbol rate is given by dividing the ADC sampling rate through the minimal possible decimation rate:

$$f_{sym_{max}} = \frac{f_s}{M_{min}} = \frac{100\,Mhz}{4 \cdot 4} = 6.25\,MSym/s$$

Future implementations could replace the standard CIC & HBF filters offered by the USRP by filters that allow lower decimation rates to achieve even higher symbol rates.
The real-time performance of the proposed prototype (see figure 5.11) has been tested on a PC running Ubuntu Linux using an Intel Core2 vPro (Quadcore CPU, each core has 2.83 GHz). At the maximal symbol rate of $6.25\,MSym/s$ the maximum utilization of each core was approximately 80%. No overflows occurred at the input network card buffer because the CPU processed the data fast enough.

8. Conclusions and Future Work

We have seen that the implemented algorithm for frame synchronization is really efficient. The differential correlation is quite resistant against frequency and phase errors. The combination with the proposed peak detector algorithm provides a robust solution with both, a fast acquisition time and a low probability of false detection even for $\frac{E_s}{N_0}$ values as low as $-2dB$.

It was shown that the theoretical SNR estimators from the literature can be successfully applied to DVB-S2 transmissions. The implemented DA / NDA methods have proved to have a decent performance in the practical range of $1 - 13\,dB$.

The SDR concept turned out to be quite flexible and ideally suited for the required tasks. Depending on the necessary speed performance we can place blocks either in software or place them inside the FPGA. After implementation in software, the required max. symbol rate of $8\,MSym/s$ could not be reached in realtime. Therefore it was decided to place the correlator and the former stages in FPGA. In future versions the CIC and halfband filters need to be replaced because they only allow a maximal symbol rate of $6.25\,MSym/s$.

The GNU Radio platform and the USRP proved to be quite mature and stable but both lack a bit of documentation. Especially many of the USRP internals are not well documented so that we had to resort to the source code. This is especially tedious when custom parts of the FPGA have to be changed and no signal descriptions are available. For this reason the studies of the internals of the USRP and WBX are presented in the appendix section. Another problem that arose was that the FPGA was already quite full (80%) and therefore some time optimizing the timing constraints had to be spent.

A. USRP N210 Internals

A.1. Description

The USRP (Universal Software Radio Peripheral) N210 is a hardware sampling device that is suited for application in GNU Radio. Figure A.1 shows the USRP N210 with open case. The WBX Daughterboard (further details in appendix B) is attached on top of the USRP Motherboard. All used external interface connections (Ethernet, MICTOR, platform cable USB, serial connection) are also shown in that figure.

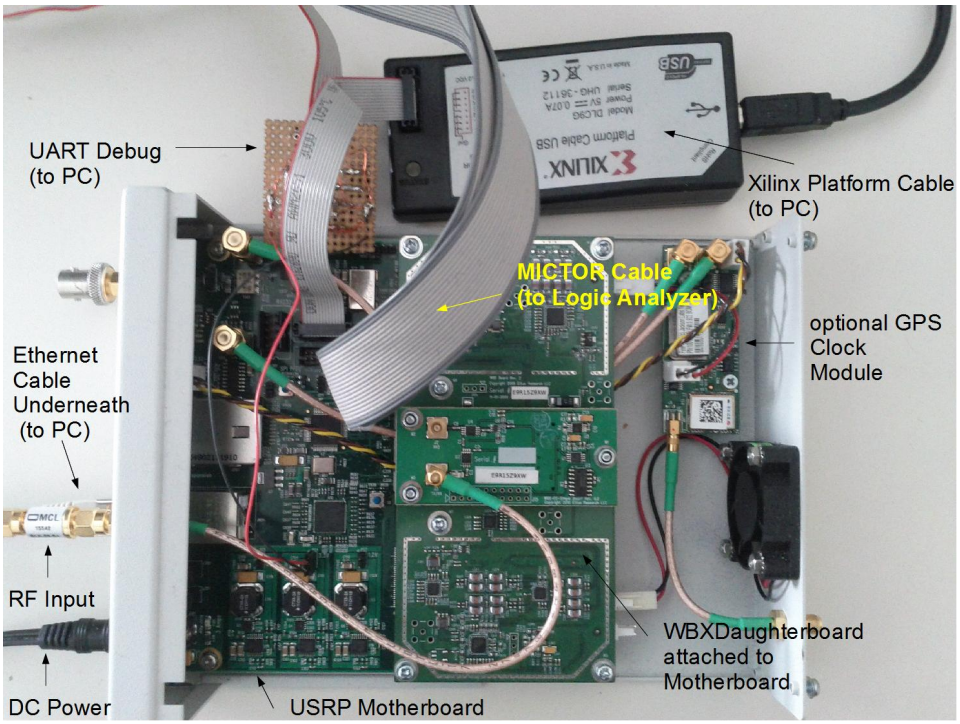

Figure A.1.: Opened USRPN210

Figure A.2.: Front view of the USRP N210

A.2. Important Features and Components

- FPGA: Xilinx Spartan XC3SD3400A
- ADCs: 14-bits 100 MS/s
  - ADS62P44: dual channel, 14-bits, 125/105/80/65 MSPS ADC with DDR LVDS/CMOS outputs
  - ADC's full range is 2V peak-to-peak with the input 50 Ohm impedance
- DACs: 16-bits 400 MS/s
  - AD9777: 16-Bit 160 MSPS 2x/4x/8x Interpolating Dual TxDAC+® D/A Converter
- Gigabit Ethernet connectivity:
  - LSI TruePHY ™ ET1011C Gigabit Ethernet Transceiver
- Clock:
  - Distribution PLL: AD95101.2 GHz Clock Distribution IC, PLL Core, Dividers, Delay Adjust, Eight Outputs
  - Reference Clock MUX: SY89545L 3.3V, 3.2Gbps differential 4:1 LVDS multiplexer with internal input termination
  - TCXO Frequency Reference (~2.5ppm)
  - Optional internal GPS locked reference oscillator
- MIMO capable - Requires two or more USRP N210 devices

A.3. Clocking

A.3.1. Clocking on the Mainboard

The AD9510 PLL is the main is responsible for clock distribution on the motherboard. It provides two programmable dividers R and N that can be used to change the intermediate clock (`clk_2`) that is derived of the incoming reference clock (`ref_clk`). The PLL provides eight clock outputs where each can further divide this intermediate clock to get individual clock rates. This is again done with individual programmable dividers (shown in figure A.3). All programmable dividers are accessible through an SPI interface. The relation between the intermediate clock and the reference clock is given by:

$$clk\_2 = ref\_clk \cdot \frac{N}{R} = ref\_clk \cdot \frac{P \cdot B + A}{R}$$

where R is set directly and N is set indirectly through 3 register settings:

$$N = (P * B + A)$$

The USRP N210 in standard configuration provides a 100 MHz clock on all outputs of the PLL. In normal configuration the reference clock for the PLL comes from the 10 MHz onboard oscillator. The device has also an input for an external reference clock that allows to synchronize the clocks to a common source [Ettc]. The selection of the source reference clock is done by the *clk_sel* wires whose settings are stored inside the FPGA in a status register. Normally the PC-Side UHD driver sets this register by sending control-messages to the device during initialization (see UHD/clock_ctrl.cpp) for further details. The UHD driver also sets the dividers correctly by sending control-messages to provide a 100 MHz clock signal out of the internal reference clock (10Mhz). It sets the clock ratio to $\frac{N}{R} = 10$ and all other programmable dividers to 1.

A.3.2. External REF CLOCK Specifications

According to [Etta] the USRP N210 allows the use of an external 10MHz reference clock. Square wave input is the preferred input waveform (best phase noise performance) but sinusoidal waveform is also acceptable. The N210 allows an reference clock input power level of *0 to 15dBm*.

A.3.3. Clock Distribution inside the FPGA

The Xilinx Spartan XC3SD3400A includes eight Digital Clock Managers (DCM) which can be used for clock skew elimination (delay locked loop) and for frequency synthesis from a reference clock. Only one DCM is used in the standard configuration of the USRP N210 for the clock distribution. It provides a 100 MHz clock used for the DSP core. Another clock (50 MHz) is provided for the ZPU and the Wishbone Interfaces. Additionally there is a third clock that provides a 270° shifted clock signal to interface the SRAM blocks.
Besides these clocks there are also 3 clocks that pass the DCM and are generated from external peripherals. GMII_RX_CLK and clk_to_mac are generated by the Ethernet PHY. The ser_rx_clk is used for the SERDES component.

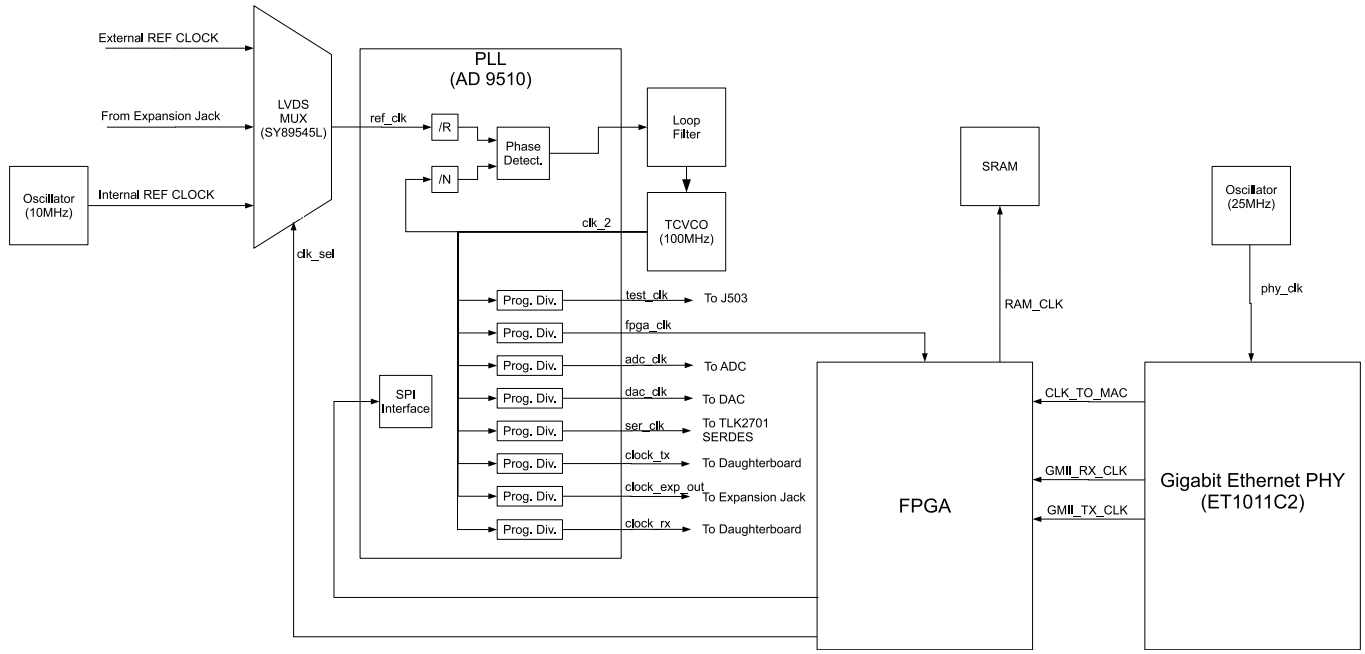

Figure A.3.: Clock distribution on the mainboard

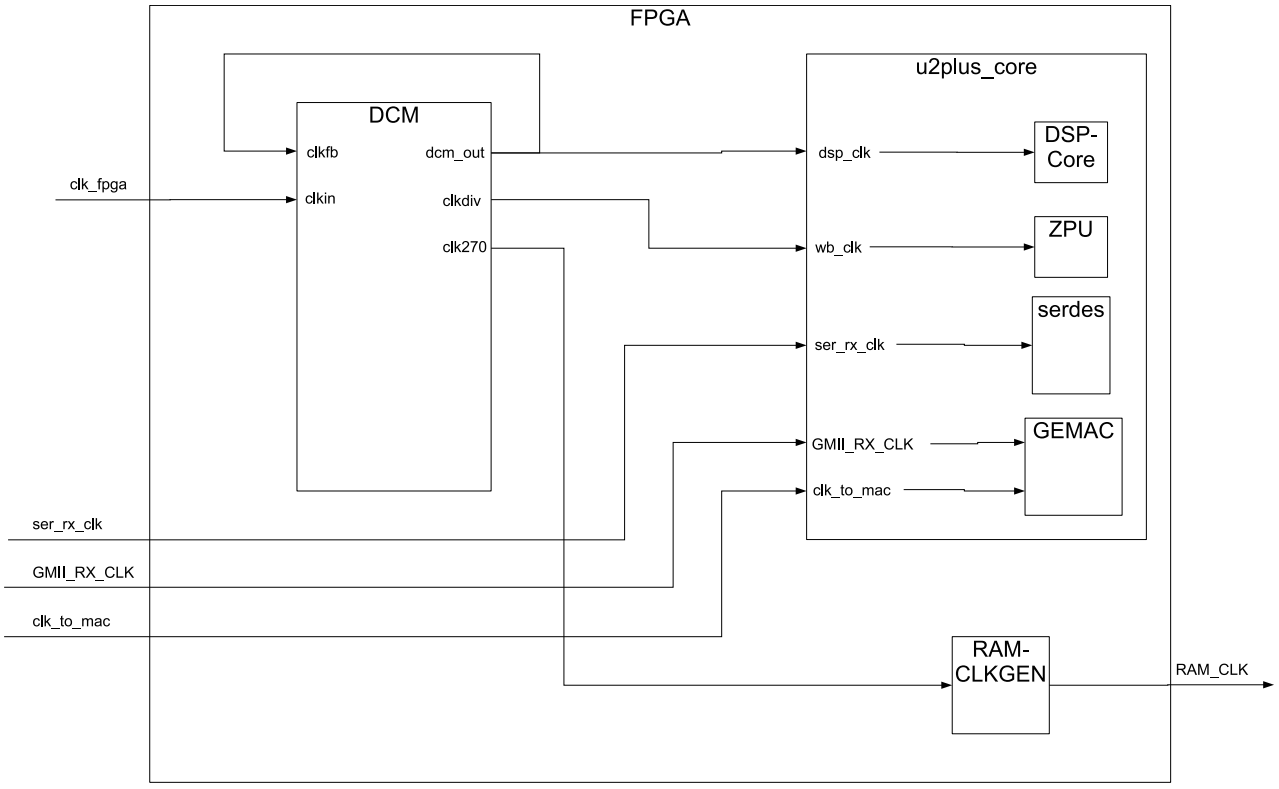

Figure A.4.: Clock distribution inside the FPGA

A.4. Internal FPGA Design

The USRP N210 FPGA design consists of a modern System-on-Chip (SOC) architecture (see figure A.5). Since GNU Radio is designed to be an open source project it makes heavy use of free IP cores from opencores.org. To connect various components, a Wishbone bus (the recommended bus system by opencores.org) has been used. The design includes the following main components:

- ZPU (a lightweight Soft Core CPU available from [Ope])
  - stack-based
  - address bus: 32bit wide
  - small op-code: 8bit
- Memory (internal block RAM memory)
- Bootloader memory
- Wishbone bus to interconnect all components
- 2 RX-DSP Cores / 1 TX-DSP Core
  - these is the are the main data paths for RX and TX
  - performs mainly the decimation / interpolation of the samples
- UART
- Programmable interrupt controller
- Ethernet controller
  - Interface to Ethernet PHY, Packet Router and ZPU

A.4.1. Memory Layout

The following memory layout was chosen by the USRP N210 designers for the memory mapped IO in the system (further details in `<FIRMWARE_DIR>/lib/memory_map.h`):

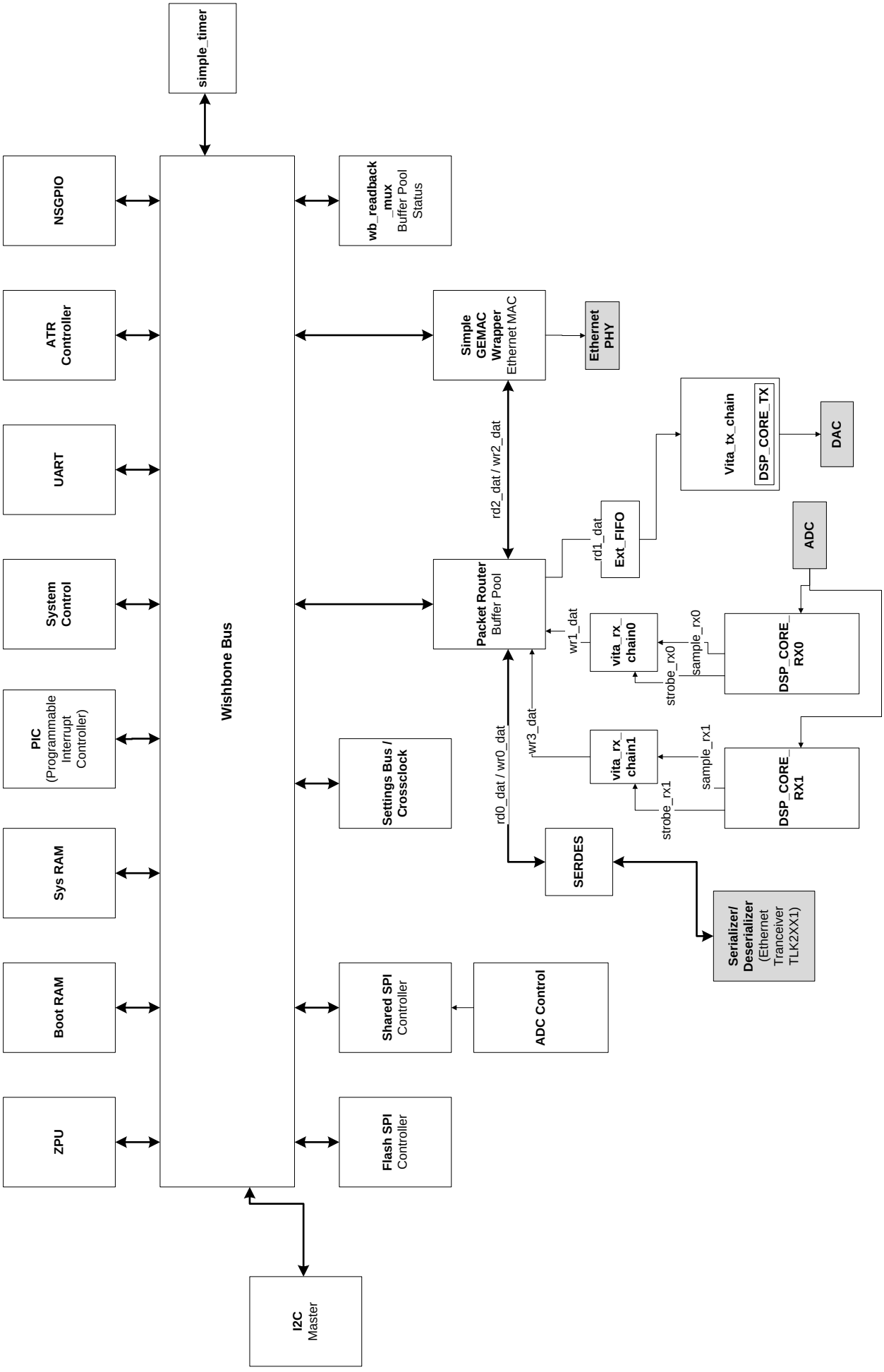

Figure A.5.: Internal SOC architecture inside the FPGA

Name	Address
ROUTER_RAM_BASE	0x4000
SPI_BASE	0x5000
I2C_BASE	0x5400
GPIO_BASE	0x5800
READBACK_BASE	0x5C00
ETH_BASE	0x6000
SETTING_REGS_BASE	0x7000
PIC_BASE	0x8000
UART_BASE	0x8800
ATR_BASE	0x8C00
ICAP_BASE	0xA000
SPIF_BASE	0xB000
RAM_BASE	0xC000

Table A.1.: Memory layout

The settings registers (addresses start from SETTING_REGS_BASE) are used to set the common properties of the components used in the SOC system. One subset of the settings registers is used to control the properties of the DSP RX (SR_RX_DSP0) path for example. So it can be used to set the decimation rate of the integrated CIC filters and enable the appropriate halfband filter if needed. Further details have to be looked up in the VHDL source code.

A.4.2. Samples via Ethernet

The samples are transported to the PC via the Ethernet interface using the UDP protocol. The host side gets the sent data by using the UHD driver which allows a transparent access to the I/Q samples.

A.4.2.1. SPI Flash and Memory Layout

Since FPGAs don not have non-volatile memory it is required to load the FPGA configuration ("FPGA image") and the firmware from an external source. This external source is an 8MB SPI flash located on the backside of the motherboard. Its memory layout is shown in table A.2:

Area	Start-Address	End-Address
SAFE_FPGA	0x000000	0x17FFFF
PROD_FPGA	0x180000	0x2FFFFF
PROD_FIRMWARE	0x300000	0x3EFFFF
SAFE_FIRMWARE	0x3F0000	0x3F3FFF

Table A.2.: SPI flash memory layout

A.4.2.2. Bootloading Sequence

According to [Fos11] and by studying the bootloader source code the boot sequence is performed the following way:

- The FPGA loads its configuration automatically starting from SPI flash address 0x000000 (which corresponds to the safe FPGA image)
- This image contains a bootloader (a prefilled boot RAM) which tries to find a valid production FPGA image
  - a valid FPGA image: leading 0xFF padding, followed by the sync bytes 0xAA 0x99
- The bootloader then tries to load this production image using ICAP (a technique to dynamically reconfigure FPGA images)
  - a flag is written into the EEPROM that we should now be in production mode
- The bootloader of the production FPGA image then tries to locate a valid production firmware and tries to load it
  - the production mode is detected by reading out the EEPROM flag
- If no valid production firmware is found it will try to load the safe firmware
- If that also fails the bootloader will fall back to a basic prompt where a valid firmware image can be loaded by the serial port

A.4.3. Generating FPGA Images

To compile FPGA image for the USRP N210 the paid version of the Xilinx ISE Design Suite is necessary. We recommend using ISE v12.4. in favour of v13 since the whole compilation was faster.

To generate an FPGA image go into FPGA image source path and create a XILINX ISE -project file and synthesize the design by using the following commands :

```
cd <SOURCE_DIRECTORY>/uhd/fpga/usrp2p/top/N2x0
make N210R3
```

This will generate a Xilinx ISE project `u2plus`.xise for the chosen board type (N210 Revision 3) in the directory:

```
<SOURCE_DIRECTORY>/uhd/fpga/usrp2p/top/N2x0/build-N210R3/
```

To generate the FPGA binary images (.bin and .bit files) we can open the project in the Xilinx Project Navigator. Inside the project environment we may change according VHDL files to our needs. Afterwards we can start the compilation process. The process steps of this compilation are shown in figure A.6. After generating FPGA images they can be used to reconfigure the FPGA.

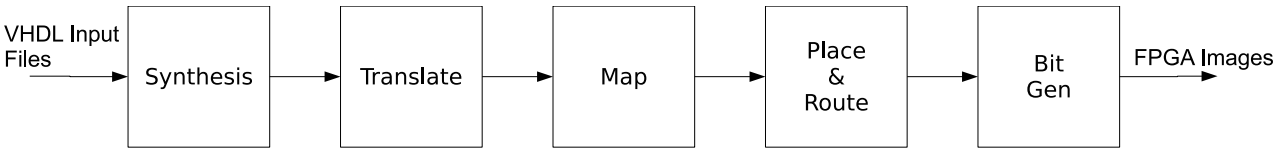

Figure A.6.: FPGA image generation process steps

A.5. Firmware

A.5.1. Bootloader

The main task of the bootlaoder code is to load the production FPGA image (through ICAP) and a production firmware into the block RAM memory. For further details on the boot sequence may be found in section A.4.2.2.

A.5.2. Main Firware

The main task of the firmware is to initialize the components of the SOC system. It additionally provides a TCP/IP Stack (LWIP) to perform the communication (e.g. to set the sampling rate or start streaming) between the host PC and the USRP. Also the SPI flash programming over Ethernet is provided in the firmware.

A.5.3. Generate Firmware Images

There is a small CPU (called ZPU [Ope]) running on the FPGA for which a firmware is necessary. To generate this firmware a compiler (zpu-elf-gcc) for the ZPU is available from [Zyl].
To compile the firmware you will have to change into the source directory, create a build-directory and call the Build System (cmake):

```
cd <SOURCE_DIRECTORY>/uhd/firmware/zpu
mkdir build
cd build
cmake ../
make
```

This will create a `usrp2p_txrx_uhd.bin` file in the `<SOURCE_DIRECTORY>/uhd/firmware/zpu/build/usrp2p` directory.
This file can be later used for flashing.

A.6. Flashing

A.6.1. Burn FPGA Images into the SPI Flash

The programming of the SPI flash is done via Ethernet. The UHD environment offers an application called "Net Burner" for this purpose. The program comes in a GUI and a command line application.

To start burning using the Net burner *command line* use:

```
python <UHD_INSTALL_PATH>/utils/usrp_n2xx_net_burner.py --addr=<ip address>
--fw=<path for firmware image (.bin)> --fpga=<path to FPGA image (.bin)>
```

You may use the option `--overwrite_safe` if you want to overwrite bad safe images.

To start the *GUI application*:

```
python <UHD_INSTALL_PATH>/utils/usrp_n2xx_net_burner_gui.py
```

Remark: The Net Burner Tool checks for a valid image which should match the FPGA Revision by its Filename. If for a example a N210 Rev 3 board is connected then the FPGA binary image (.bin File) must contain the substring "`n210_r3`"

A.6.2. Repair Bricked Boards

It may rarely occur that the flashing program gets stuck or crashes. In such a case it may be possible that the production FPGA image and or the firmware image are invalid.
The first step in trying to recover these images is to boot into "safe mode" by pressing the button S2 (located on the mainboard) during the boot process until the LEDs in the front remain solid. In this process the bootloader tries to load the safe firmware from the SPI flash instead of the production image. In this safe mode the device is always configured with the IP address 192.168.10.2. Now it is possible to use the "Net Burner" tool to recover the FPGA and firmware image.
If this does not help and nothing happens after booting (LEDs do not light) the only way to recover USRP board is to directly configure the FPGA by JTAG using the Xilinx Platform Cable in combination with the iMPACT software that comes with the Xilinx ISE Toolchain. An appropriate image (.bit File) is needed to load the FPGA. If there is a valid firmware on the device then it will be loaded and the FPGA image and the firmware image in the SPI-Flash can be recovered by using "Net Burner". [Neg]
If there is no valid firmware image available the device will switch to a Intel HEX prompt. So you can transmit a valid image using the serial interface by using a supplied tool.

A.7. Logic Analysis

There are two possibilities to monitor and analyze the internal signals produced by the USRP N210.

A.7.1. Using the MICTOR Connector

The USRP N210 provides a MICTOR (Matched Impedance Connector) (J301) that can be connected to a logic analyzer. The Main Verilog File provides a 32 bit-signal set called "debug" which is routed to this MICTOR interface. Any signal that has to be observed can be simply connected to this debug signal set.

A.7.2. Using ChipScope

Another opportunity to monitor signals is to use the ChipScope Technology provided by Xilinx. This integrates a small Logic Analysis System into the FPGA design and can be later monitored by JTAG through the platform cable. Since this technique requires an additional amount of block RAM space and slices inside the FPGA and the FPGA was already full to a decent extent we did not use it.

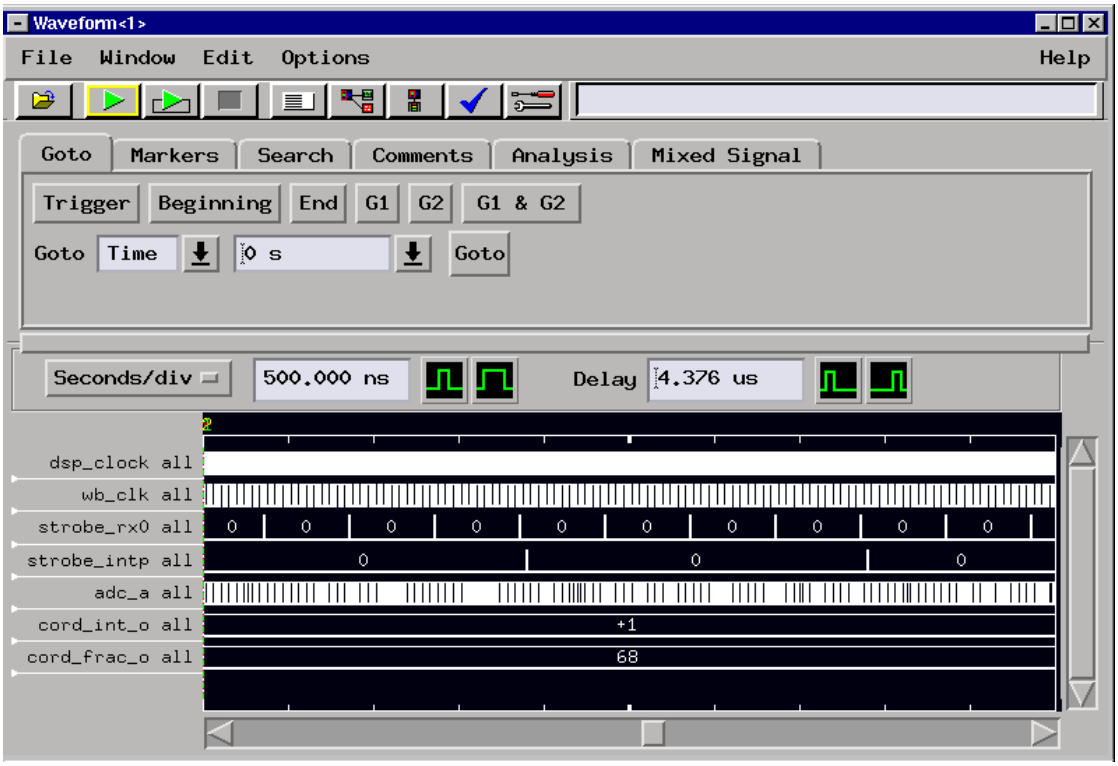

Figure A.7.: Example of logic analysis system attached over MICTOR cable

A.8. Monitoring UART Output

To verify correct startup and to log the debug outputs of the ZPU UART via the serial interface a small circuit has been soldered to provide the necessary signal level conversion from 3.3V to RS232. The circuit basically uses a MAX2323 IC.

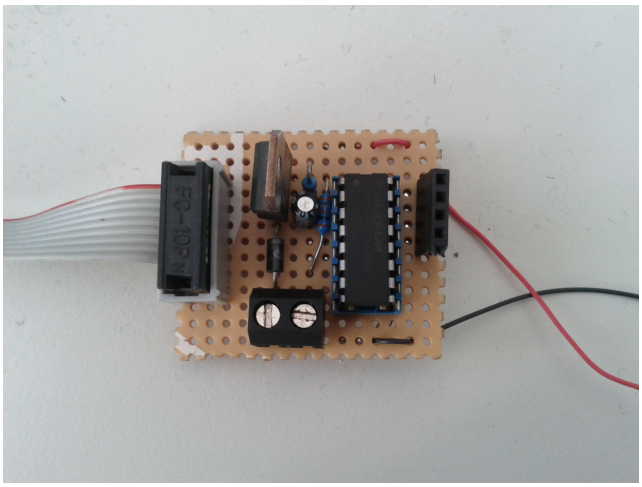

Figure A.8.: Signal level converter for UART

A.9. Modifying the Original FPGA Source

A.9.1. Removing Unnecessary FPGA Parts

For this project a TX path was not needed so the TX path can be removed to save some space inside the FPGA. The usage level of the FPGA can be further reduced by removing the second DSP core (see below). Space comparisons have been performed and can be seen in table A.3. Obviously there is a great reduction of area utilization by removing the mentioned parts. This available space can now be used for custom logic parts.

FPGA Resource	Standard Configuration	Modified Configuration (TX and RX1 removed)
Number of occupied Slices	79%	55%
Number of DSP48As	21%	8%
Number of RAMB16BWERs	24%	23%

Table A.3.: FPGA utilization in 2 different configurations

One thing we found out is that the SERDES-module which would not be necessary can not be removed. After burning the image without the SERDES module the communication between the USRP and the host PC did not work anymore.

A.9.2. Providing Custom Calculations on the FPGA / UHD Settings

Sometimes there is a need to perform computational instensive calculations on the FPGA instead of using the PC (e.g.: the differential correlator in this Master Thesis). For this purpose the FPGA design has to be modified to transmit the additional samples on a second channel. The standard FPGA design includes two complete DDC chains. This includes a second RX data path (called `dsp_core_rx1`) which consists of the same elements as the first RX path. This second channel is originally intendend to use a second carrier which will be downconverted by the CORDIC unit. This DSP core can be replaced by a custom logic. So an additional 32 bits are available to transmit the results of custom calculations.
On the PC host side a proper configuration of the USRP UHD source is necessary. The UHD source block has to be configured to:

- use two channels
- use following WBX subdevice specification: "`A:0 A:0`".

Practical measurements have also shown that it has to be guaranteed that the digital strobe signals of both channels (`strobe_rx0` and `strobe_rx1`) occur at the same time. Otherwise the UHD driver on the host PC will notify the user that the timestamps of the received samples mismatch and will not continue to work.

A.9.3. Timing Constraints Settings

When implementing custom logic into the data path of the FPGA it is important to fulfill the timing (setup and hold time)[Kil07]. If the FPGA is filled to some decent extent (rule of thumb $> 80\%$) and no timing constraints are provided, errors concerning the timing on certain paths may occcur. The place and route algorithm tries the best to avoid violations of

the timing but cannot avoid them because the FPGA is already quite full. Timing constraints provide a solution to this problem if certain parts of the design work at lower clock speeds (through clock-enable signals). We may add them to a timing group where we can loosen the timing constraints. Therefore the place and route algorithm can prioritize other paths that have more stringent timing constraints. In this project for example the correlator works at a much lower speed than the $100\,MHz$ clock speed that is provided by the FPGA. Therefore we added some instructions to the User Constraints File (UCF) that relaxing the timing requirements of each flip-flop of the correlator. The UCF file can be found in the directory: `<SOURCE_DIRECTORY>/uhd/fpga/usrp2p/top/N2x0/build-N210R3/u2plus.ucf`

B. WBX Internals

B.1. Specification

The WBX board is a transceiver daughterboard for the USRP N210 produced by Ettus Research. It has the following specifications [Ettd]:

- Full Duplex Transmitter/Receiver
- 50 MHz to 2.2 GHz coverage
- Front-end boards (Grand-Daughterboards) may be changed to provide higher power amplifier, custom filters, antenna switches, etc.
- Receiver noise figure of 5-7 dB

B.1.1. RX Path Components

The following components are used in the WBX daughterboard:

- HMC174MS8 (TX/RX switch)
- MGA 62563 (GaAs MMIC Amplifier)
  - 22 dB Gain
  - 0.9 dB Noise Figure
  - max. 21 dBm Input
- HMC 472LP4 (GaAS MMIC Attenuator)
  - variable Attenuator: -31.5 to -0.5 dB
  - 0.5 dB Steps
  - max. 27dBm Input
- MGA 82563 (GaAs MMIC Amplifier)
  - 13 dB Gain
  - max. 13dBm Input
- ADF4350 (Synthesizer)
- ADL5387 (I/Q Demodulator)
- ADA 4937 (Ultralow Distortion Differential ADC Driver)
- Anti Aliasing Filters for the ADC

– 20 MHz Cheb. Lowpass

– 50 Mhz Butterw. Lowpass

The operation of the WBX receive path is shown as block diagram in figure B.1.
The WBX performs the complex downconversion into the baseband. This is done by first amplifying the received signal with an amplifier chain. The amplifier chain is able to produce a variable gain through the attenuator whose attenuation factor can be serially programmed in $0.5\,dB$ steps. The amplified signal is then mixed into the complex baseband. The LO frequency for the I/Q demodulator is provided by a frequency synthesizer. It is programmable through a serial interface and allows to set the required LO frequency.

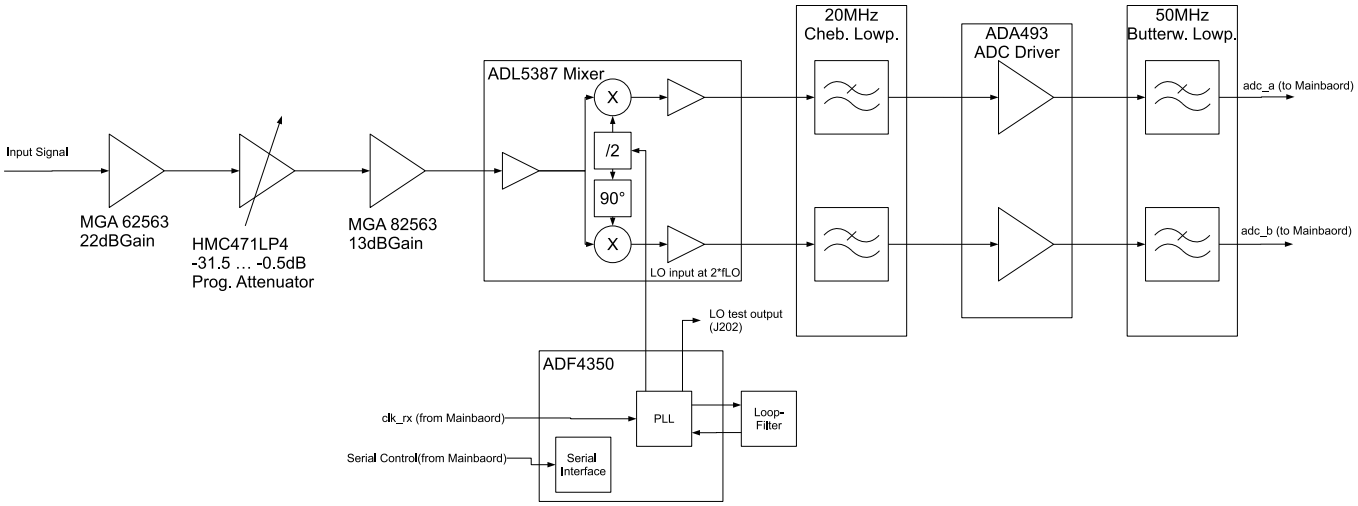

Figure B.1.: WBX-Board receive path block diagram

B.1.2. Maximum Input Power

According to discussion of [Lee10] and [Abe10] the maximum WBX input power should not exceed *-10dBm* ($\sim 0.2\,Vpp$).

B.2. Measuring the ADC Input Power

First a FPGA image was generated where one of the two 14-bit AD converter I/Q sample pair was routed to the MICTOR. The Logic Analysis System (LAS) was configured to display the received digital data in a two-complements format. By manually testing different gains of the UHD source we can change how a signal drives the ADC. To measure the power of the digital samples and to find out how many bits are effectively used we can export the digital sample values out of the logic analyzer and read them with MATLAB. The average power can now be determined by squaring all samples

$$P_{in} = \frac{1}{L}\sum_{k=0}^{L} |x[i]|^2$$

This can now be used to calculate the RMS (Root Mean Square):

$$x_{RMS} = \sqrt{\frac{1}{L}\sum_{k=0}^{L}|x[i]|^2}$$

To get to the number of driven bits we simply determine:

$$N_{bits} = ld(x_{RMS})$$

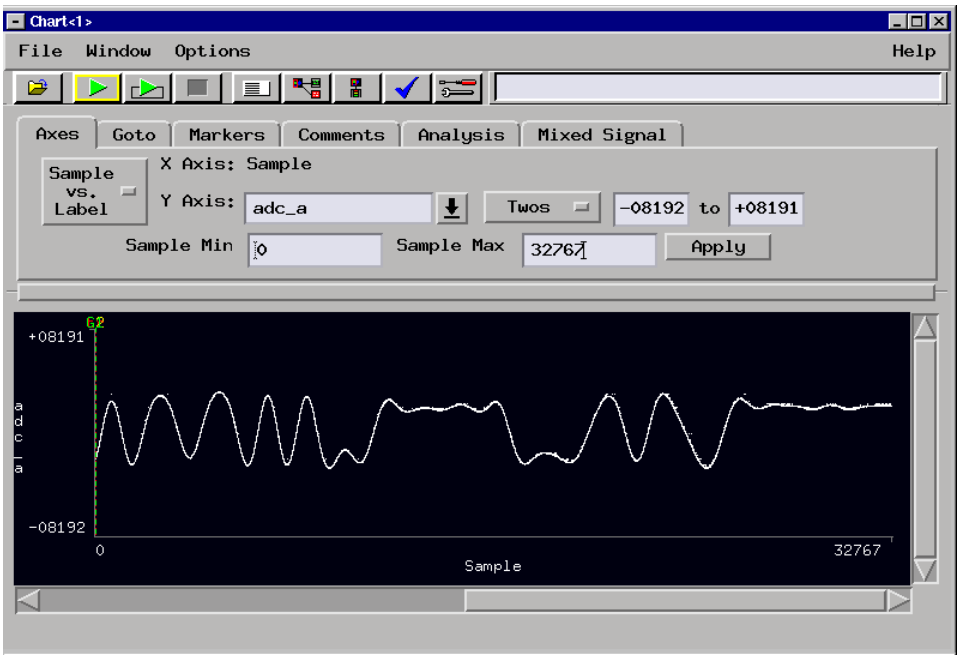

Figure B.2.: Measuring the power of the inphase ADC-Samples

B.3. Performing AGC

The WBX Board itself does not offer Automatic Gain Control (AGC) to drive the ADC optimally. We can only set a static value of the receiver gain. An idea for future implementations may be to measure the power in software and determine the average power of the input samples. According to the measured power we can now send a command (`wbx_base::set_rx_gain()`) to the WBX to either reduce or increase the gain.

C. UHD

UHD is short for Universal Software Radio Peripheral Hardware Driver. UHD provides a unified access to all devices developed by Ettus Research and works on common platforms (Linux, Windows and MAC). UHD provides access to the device through the driver standalone libraries or with 3rd party application like Simulink or GNU Radio (gr-uhd) [Ettb].
The UHD driver provides many functions like:

- get samples out of the device
- set the sample rate on daughterboards (i.e.: WBX)
- set the clock source for the FPGA
- set the decimation rate
- handling of multiple USRPs attached to a PC

C.1. Installing

The install process is usually done by compiling the UHD source code. The current UHD driver source can be downloaded from the Ettus Homepage by cloning the GIT repository:

```
git clone git://code.ettus.com/ettus/uhd.git
```

The checked out source code does not only include the host PC code but also the device firmware and FPGA image.

Alternative to compiling the source there are prebuilt binary images for Windows (.exe) and Linux (RPM and DEB packages) which can be installed on the system. They are available from:

```
http://files.ettus.com/uhd_releases/
```

C.1.1. Building and Installing on Linux

On Linux the cmake script will generate an "automake"-project which can be compiled and installed using the following commands:

```
mkdir build
cd build
cmake ../
make
sudo make install
sudo ldconfig
```

The UHD driver libuhd.so should now reside in the Library Path. All other UHD related files go into `usr/local/bin/` and `/usr/local/share/uhd/`.

C.1.2. Building on Windows

On a Windows operating system the UHD PC host source code can be build by using the Microsoft Visual C++ Compilers. The source code includes a cmake script which can be configured to generate a MSVC project. This project can now be compiled by the MSVC. Instructions can be found at:

`http://files.ettus.com/uhd_docs/manual/html/build.html#build-instructions-windows`

C.2. Tools

There exist a bunch of tools to test, verify and program the USRP devices. The most important ones are listed here.

C.2.1. UHD find Devices

This tool will find all USRP devices attached to the host PC and will plot their according IPs. It can be called using:
`/usr/local/bin/uhd_find_devices`

C.2.2. UHD USRP Probe

This tool can probe all settings of the mainboard and also all attached daughterboards and their settings and displays it on the screen. The tool can be used by calling:
`/usr/local/bin/uhd_usrp_probe`

C.2.3. UHD Net Burner:

The Net Burner (a Python script) is an essential tool for burning custom firmware and FPGA images into the SPI Flash. It sends commands to reprogran the SPI flash via UDP to the device firmware which will rewrite the according sections of the the SPI flash memory. The tool offers an option to overwrite the fail-safe firmware or FPGA image. This option can be enabled by using the command line option `--overwrite_safe`
Its location is in:
`/usr/local/share/uhd/utils/usrp_n2xx_net_burner.py`

For further information on how to flash the board have a look in section A.6.1.

C.3. gr-uhd

To use the UHD driver in GNU Radio there is a block called “gr-uhd”. It provides a kind of wrapper to integrate the UHD driver into the GNU Radio environment. All necessary functions like setting the center frequency or the sampling rate can be called through this GNU Radio block.

D. Deliverables

The following sections show a short overview of the most important implemented modules and their purpose.

D.1. VHDL Source Files

The VHDL Source Files are located in `<MAIN_SOURCE>/fpga/usrp2/alphasat/diff_corr/` path

- `constants.vhd`: Constants File including taps and fixed bitwidth constants
- `correlator.vhd`: Top module of the differential correlator
- `delay_line_corr.vhd`: Implements a parameterizeable Delay Line
- `diff_corr.vhd`: Implements the calculation of the differential
- `diff_corr_rounded.vhd`: Top module for diff_corr and round_signed to zero
- `diff_plsc_corr.vhd`: Differntial correlation on PLSC part
- `diff_sof_corr.vhd`: Differential correlation on SOF part
- `peak_calculation.vhd`: Implements the result calculation, offers 2 different architectures (Norm1 and Alpha Max Beta Min Approximation of Norm 2)
- `peak_detector_exp_averaging.vhd`: Implements the exponental averaging + threshold method for peak detection
- `round_signed_to_zero.vhd`: Used for proper offset free rounding of the results
- `signed_adder.vhd`: Parameterizeable Signed Adder
- `swapping_unit.vhd`: Implementation of the Swapping Unit.
- `tb_correlator.vhd`: Test bench of the Top Module
- `tb_diff_corr.vhd`: Test bench for the diff_corr module
- `tb_peak_detector_exp_averaging.vhd`: Test bench for the peak_detector_exp_averaging module

D.2. GNU Radio Source Files

The GNU Radio modules are all in the `<MAIN_SOURCE_PATH>/gnuradio/alphasat_blocks/` path:

- `apps/`: Source directory for GNU Radio Companion files and Python simulations
- `lib/alphasat_CONSTANS.h`: Some common constants for the GNU Radio blocks
- `lib/alphasat_DEBUG.h`: Some common debug functions for the GNU Radio blocks
- `lib/alphasat_tools.cpp/.h`: Important tools library that implements some common functions
- `lib/alphasat_differential_detection_cf.cc/.h`: Floating point model of the differential correlator
- `lib/alphasat_differential_detection_ss.cc/.h`: Bit-Accurate Model of the differential correlator
- `lib/alphasat_dvb_s2_source.cc/.h`: DVB-S2 source block, generate symbols according to the DVB-S2 standard
- `lib/alphasat_peak_detect_by_decoding.cc/.h`: Implements the adapted peak search algorithm without resyncing
- `lib/alphasat_peak_detect_by_decoding_w_reset.cc/.h`: Implements the adapted peak search algorithm with resyncing
- `lib/alphasat_peak_detect_with_exp_averaging.cc/.h`: Implements the exponental averaging + threshold method for peak detection
- `lib/alphasat_short_vec_to_float.cc/.h`: Used for conversion of the correlation results delivered by the USRP
- `lib/alphasat_sim_*.cc/.h`: Some modules used during simulation
- `lib/alphasat_snr_est_da_header_and_pilots.cc/.h`: Implementation of DA SNR Estimation
- `lib/alphasat_snr_est_nda.cc/.h`: Implementation of NDA SNR Estimation

D.3. MATLAB Source File

As already mentioned `MATLAB` is used for analysis. The files are located in the directory `<MAIN_SOURCE_PATH>/matlab/test_signal_generation/`. All the logs and binary files are dumped into subdirectories. The following MATLAB scripts were implemented (only important ones shown):

- `agilent_read_samples_from_files.m`: Reads in the saved ADC samples from the Logic Analysis System
- `constants.m`: Some common constants

- `diff_corr.m`: High Level Model of differential correlation. Implements all different correlation result calculations
- `generate_dvbs2_test_signal.m`: High Level Model of DVB-S2 source
- `generate_scrambling_sequence.m`: High Level Model to genreate the scrambling sequence for the DVB-S2 payload
- `gnuradio_read_*.m`: Various functions to read from binary files generated by GNU Radio File Sinks
- `gnuradio_write_*.m`: Various functions to write to binary files to be used by GNU Radio File Sources
- `meas_adc_power.m`: Analyze the power on the ADC samples from the Logic Analysis System
- `meas_acqu_time_snr_est_performance.m`: Analyzes the log-files from the peak detector, DA/NDA estimator generated during practical measurements.
- `meas_plheader_var.m`: Used to genreate a plot to analyze the performance of the DA estimator (PLHEADER only)
- `meas_resyncs.m`: Analyze how often the adapted peak search algorithm needed to resync
- `test_codes_and_taps_corr.m`: Tests to study the PLSC codes and their structure. Generating taps for the differential correlator
- `meas_usrp_calc_power.m`: Reads the data generated by the USRP channel and calculates the power of the signal
- `sim_*.m`: Analyze various log-files from simulations generated by GNU Radio
- `vhdl_*.m`: Various functions to generate testdata for the VHDL testbench

Bibliography

[Abe10] J. Abele. GNU Radio Mailing List - Max Voltage swing for WBX boards, 2010. Retrieved December 24, 2011. Available from: http://lists.gnu.org/archive/html/discuss-gnuradio/2010-09/msg00227.html.

[Bis12] C. Bischof. Development and Implementation of a High Performance Interpolator on a Software Defined Radio Platform. Master's thesis, Graz University of Technology, 2012.

[BS67] T. Benedict and T. Soong. The joint estimation of signal and noise from the sum envelope. *IEEE Transactions on Information Theory*, 13(3):447 – 454, jul 1967.

[CGG04] E. Casini, R. De Gaudenzi, and A. Ginesi. DVB-S2 modem algorithms design and performance over typical satellite channels. *International Journal of Satellite Communications and Networking*, 22(3):281–318, 2004.

[CL02] Z. Y. Choi and Y.H. Lee. Frame synchronization in the presence of frequency offset. *IEEE Transactions on Communications*, 50(7):1062 – 1065, jul 2002.

[ETS05] ETSI. Digital Video Broadcasting (DVB) User guidelines for the second generation system for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2) (ETSI TR 102 376 V1.1.1 (2005-02)). Technical report, feb 2005.

[ETS09] ETSI. Digital Video Broadcasting (DVB); Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications (DVB-S2) (ETSI EN 302 307 V1.2.1 (2009-08)), aug 2009.

[Etta] EttusResearch. UHD - USRP2 and N Series Application Notes. Retrieved December 24, 2011. Available from: http://files.ettus.com/uhd_docs/manual/html/usrp2.html#ref-clock-10mhz.

[Ettb] EttusResearch. UHD Start. Retrieved December 24, 2011. Available from: http://code.ettus.com/redmine/ettus/projects/uhd/wiki.

[Ettc] EttusResearch. USRP Clocking Notes. Retrieved December 24, 2011. Available from: http://gnuradio.org/redmine/projects/gnuradio/wiki/USRPClockingNotes.

[Ettd] EttusResearch. WBX. Retrieved December 24, 2011. Available from: http://www.ettus.com/WBX.

[Fos11] N. Foster. USRP Mailing List - A few questions about USRP N210, 2011. Retrieved December 24, 2011. Available from: http://lists.ettus.com/pipermail/usrp-users_lists.ettus.com/2011-April/001030.html.

[GK06] W. Gappmair and O. Koudelka. Moment-Based SNR Estimation of Signals with Non-Constant Envelope. In *3rd Conference on Advanced Satellite Mobile Systems (ASMS)*, pages 301–303, Herrsching, Germany, 2006.

[GT05] P. Gao and C. Tepedelenlioglu. SNR estimation for nonconstant modulus constellations. *IEEE Transactions on Signal Processing*, 53(3):865 – 870, mar 2005.

[Kae08] H. Kaeslin. *Digital integrated circuit design: from VLSI architectures to CMOS fabrication*. Cambridge University Press, 2008.

[KCP+07] P. Kim, G.E. Corazza, R. Pedone, M. Villanti, D.-I. Chang, and D.-G. Oh. Enhanced Frame Synchronization for DVB-S2 System Under a Large of Frequency Offset. In *IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1183 –1187, mar 2007.

[Kil07] S. Kilts. *Advanced FPGA design: architecture, implementation, and optimization*. Wiley, 2007.

[KLH+11] I.S. Kang, H. Lee, S.J. Han, C.S. Park, J.H. Soh, and Y.J. Song. Reconstruction method for Reed-Muller codes using Fast Hadamard Transform. In *13th International Conference on Advanced Communication Technology (ICACT)*, pages 793 –796, feb. 2011.

[Kou11] O. Koudelka. Q/V-band communications and propagation experiments using ALPHASAT. *Acta Astronautica*, 69(11-12):1029 – 1037, 2011.

[Lee10] M. D. Leech. GNU Radio Mailing List - maximum input signal power for WBX, 2010. Retrieved December 24, 2011. Available from: http://lists.gnu.org/archive/html/discuss-gnuradio/2010-11/msg00542.html.

[Lyo10] R.G. Lyons. *Understanding Digital Signal Processing*. Pearson Education Canada, 2010.

[MB07] U. Meyer-Baese. *Digital signal processing with field programmable gate arrays*. Springer, 2007.

[MBS10] G. Maral, M. Bousquet, and Z. Sun. *Satellite communications systems: systems, techniques and technology*. John Wiley, 2010.

[MD97] U. Mengali and A.N. D'Andrea. *Synchronization techniques for digital receivers*. Plenum Press, 1997.

[Mit00] J. Mitola. *Software radio architecture: object-oriented approaches to wireless systems engineering*. J. Wiley & Sons, 2000.

[MM06] A. Morello and V. Mignone. DVB-S2: The Second Generation Standard for Satellite Broad-Band Services. *Proceedings of the IEEE*, 94(1):210 –227, jan. 2006.

[MMF98] H. Meyr, M. Moeneclaey, and S. Fechtel. *Digital communication receivers: synchronization, channel estimation, and signal processing*. Wiley, 1998.

[MR04] A. Morello and U. Reimers. DVB-S2, the second generation standard for satellite broadcasting and unicasting. *International Journal of Satellite Communications and Networking*, 22(3):249–268, 2004.

[Neg] V. Negnevitsky. GNU Radio Mailing List - Bricking and recovery of N210. Retrieved December 24, 2011. Available from: http://lists.gnu.org/archive/html/discuss-gnuradio/2011-04/msg00371.html.

[Noi] NoiseCom. UFX7000 Noise Generator Datasheet. Retrieved December 25, 2011. Available from: http://noisecom.com/products/instruments/ufx7000-noise-generator?go=datasheet.

[Ope] OpenCores. ZPU - the worlds smallest 32 bit CPU with GCC toolchain. Retrieved December 24, 2011. Available from: http://opencores.org/project,zpu.

[PB00] D. R. Pauluzzi and N. C. Beaulieu. A comparison of SNR estimation techniques for the AWGN channel. *IEEE Transactions on Communications*, 48(10):1681–1691, 2000.

[QXC+08] L. Qing, Z. Xiaoyang, W. Chuan, Z. Yulong, Yunsong D., and J. Han. Optimal frame synchronization for DVB-S2. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 956 –959, may 2008.

[RCL+09] T. Rossi, E. Cianca, M. Lucente, M. De Sanctis, C. Stallo, M. Ruggieri, A. Paraboni, A. Vernucci, L. Zuliani, L. Bruca, and G. Codispoti. Experimental Italian Q/V band satellite network. In *IEEE Aerospace conference*, pages 1 –9, mar 2009.

[RS09] J. Reichardt and B. Schwarz. *VHDL-Synthese: Entwurf digitaler Schaltungen und Systeme*. Oldenbourg Wissensch.Vlg, 2009.

[SJL04] F.-W. Sun, Y. Jiang, and L.-N. Lee. Frame synchronization and pilot structure for second generation DVB via satellites. *International Journal of Satellite Communications and Networking*, 22(3):319–339, 2004.

[Tho67] C.M. Thomas. *Maximum likelihood estimation of signal-to-noise ratio*. PhD thesis, 1967.

[XWL10] R. Xue, C. Wang, and X. Li. A multiple correlation peak value detecting method for frame synchronization of DVB-S2. In *IEEE International 12th Conference on Communication Technology (ICCT)*, pages 837 –840, nov. 2010.

[YYW+10] D. Yang, C. Yan, H. Wang, J. Kuang, H. Zhang, and N. Wu. Performance evaluation of different detectors for frame synchronization in DVB-S2 system. In *International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1 –5, oct. 2010.

[ZCF+10] Y. Zhang, X. Chen, W. Fan, J. Han, and X. Zeng. Robust and reliable frame synchronization method for DVB-S2 system. In *Proceedings of the 9th Conference on Wireless Telecommunications Symposium*, WTS'10, pages 318–322, Piscataway, NJ, USA, 2010. IEEE Press.

[Zyl] Zylin. Zylin-CPU. Retrieved December 24, 2011. Available from: http://opensource.zylin.com/zpudownload.html.