Master Thesis

Development and Implementation of a High Performance Interpolator on a Software Defined Radio Platform

Christopher Bischof

Institute of Communication Networks and Satellite Communications
Graz University of Technology, Austria

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

Supervisor:
Dipl.-Ing. Dr. Wilfried Gappmair

Graz, March 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 ……. ……..
(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.

……. ……..
date (signature)

Abstract

Interpolation in digital signal processing is essential for procedures like sampling rate conversion or symbol timing recovery. In this context interpolation algorithms are employed to calculate signal values at arbitrary times.

In this thesis an implementation of a high performance interpolator and sampling rate converter is investigated, which is used for emulating fine delay variations on a satellite communications link. In addition, a timing recovery module, which consists of a timing estimator and an interpolator, is developed for a software defined radio platform. This platform is part of an experiment for adaptive coding and modulation, carried out by Joanneum Research in the framework of the ALPHASAT TDP5 - Q/V Band Payload project of the European Space Agency (ESA).

The basic theory of interpolator structures as well as the detailed description of the fundamental algorithm is discussed in this thesis. Furthermore, simulation results are presented which illustrate the performance of the proposed method. These results are in the sequel compared to a reference implementation of a cubic interpolator.

For the purpose of symbol timing recovery, an appropriate estimator is introduced and implemented. The architecture of estimator and interpolator is developed in this thesis. This timing recovery module is additionally implemented on a software defined radio FPGA. The corresponding design and functionality as well as the resulting hardware measurements are presented.

Finally, a basic overview of the GNU Radio software environment and the Universal Software Radio Peripheral device by Ettus Research is attached. This also includes instructions for the design and usage of signal processing blocks on GNU Radio.

Acknowledgements

Many people supported me during the work on this thesis and I will take the opportunity to express my gratitude to them. First of all, I would like to thank my assessor, Univ.-Prof Dipl.-Ing. Dr. Otto Koudelka, who gave me the great chance of working with the team of Joanneum Research as well as for reviewing my thesis.

Additionally, I want to thank my advisor, Dipl.-Ing. Dr. Wilfried Gappmair, who not only supported me with his wide knowledge, but who also initialized the contact to Joanneum Research. Without his work neither the start nor the achievement of this thesis would have been possible and I would have never had the opportunity to accomplish such an interesting topic in such a competent environment.

Special thanks go to my advisors at Joanneum Research DIGITAL SPA (Space Technology and Acoustics), Dipl.-Ing. Michael Schmidt, Dipl.-Ing. Dr. Johannes Ebert, and Dipl.-Ing. Harald Schlemmer. Their professional supervision during this work was invaluable and turned these busy months into an enjoyable and memorable time.

I am also very grateful for the support and the patience of my family and my friends which was the most important incentive throughout these years of study.

Contents

Abstract 5

Acknowledgements 7

List of Figures 12

List of Tables 16

List of Acronyms 17

1. Introduction 19
- 1.1. Motivation . . . 19
- 1.2. Use Case Scenarios for the Interpolator . . . 19
  - 1.2.1. Delay Variations in a Satellite Emulator . . . 19
    - 1.2.1.1. Task of this Thesis . . . 20
  - 1.2.2. Timing Recovery on a Software Defined Radio Platform . . . 20
    - 1.2.2.1. Task of this Thesis . . . 21
- 1.3. GNU Radio . . . 21
  - 1.3.1. Hardware . . . 21
    - 1.3.1.1. Supported Hardware . . . 22
    - 1.3.1.2. USRP . . . 22
    - 1.3.1.3. Universal Hardware Driver . . . 22

2. Interpolation and Digital Sampling Rate Conversion 23
- 2.1. Polynomial Interpolation . . . 23
  - 2.1.1. Implementation of the Polynomial Interpolator . . . 24
    - 2.1.1.1. Farrow Structure . . . 26
- 2.2. FIR Interpolation . . . 27
  - 2.2.1. Interpretation of the Sampling Theorem . . . 28
  - 2.2.2. Interpolation using the Sampling Theorem . . . 29
  - 2.2.3. Windowing of the Sinc Function . . . 32
    - 2.2.3.1. Rectangular Window . . . 32
    - 2.2.3.2. Kaiser Window . . . 33
  - 2.2.4. Implementation of the FIR Interpolator . . . 34
    - 2.2.4.1. Time Register . . . 36
    - 2.2.4.2. Procedure for Upsampling . . . 37
    - 2.2.4.3. Procedure for Downsampling . . . 37
    - 2.2.4.4. Parameter Values for some Example Configurations . . . 38
  - 2.2.5. Filter Analysis . . . 39
    - 2.2.5.1. Decimation Application . . . 39
    - 2.2.5.2. Satellite Emulator Application . . . 42

3. Timing Estimation **45**
3.1. Oerder and Meyr Algorithm . . . 45
3.1.1. Lower Bounds for the Timing Estimator . . . 47
3.2. CORDIC Algorithm for Angle Calculation . . . 48
3.3. Implementation of the Timing Estimator . . . 49
3.4. Combination of Timing Estimation and Interpolation for Timing Recovery . . . 51
3.4.1. Update of μ . . . 53
3.4.2. Problems with the Update of μ . . . 53

4. Nyquist Pulse Shaping **55**
4.1. Raised-Cosine Pulse . . . 56
4.2. Root-Raised-Cosine Pulse . . . 56

5. Implementation on the Software Defined Radio Platform **57**
5.1. Existing Structure of the USRP N210 . . . 58
5.1.1. Analog Receiver Frontend . . . 60
5.1.2. Receiver Frontend . . . 60
5.1.3. DSP Core . . . 60
5.1.3.1. Digital Down-Conversion . . . 61
5.1.3.2. CIC Decimation . . . 61
5.1.3.3. Half-Band Decimation . . . 62
5.2. Root-Raised-Cosine Module . . . 64
5.3. Timing Estimation Module . . . 65
5.3.1. Complex Square Module . . . 66
5.3.2. Fourier Module . . . 67
5.3.3. CORDIC Module . . . 68
5.3.4. Phase Normalization Module . . . 69
5.3.5. Rounding Module . . . 71
5.4. FIR Interpolation Module . . . 72
5.4.1. Delay Line Module . . . 73
5.4.2. Control Logic of the Interpolator . . . 74
5.4.2.1. State Machine . . . 74
5.4.3. FIR Module . . . 75
5.5. Device Utilization . . . 78

6. Simulation Results **81**
6.1. Timing Estimation . . . 81
6.1.1. Variance of the Timing Estimator . . . 81
6.2. FIR Interpolator . . . 83
6.2.1. Influence of Tap Resolution on a Single Carrier Signal . . . 83
6.2.2. Influence of Tap Resolution and the Number of Zero Crossings on a QPSK Signal . . . 87
6.2.3. Influence of the Number of Zero Crossings for the Decimation . . . 89
6.2.4. SFDR Analysis for the Filter Parameters . . . 91
6.2.4.1. SFDR for Cubic Interpolator . . . 92
6.2.4.2. SFDR as a Function of the Amplitude Resolution and the Numbers of Zero Crossings . . . 92
6.2.4.3. SFDR as a Function of the Number of Bits for Filter Resolution and Linear Resolution . . . 93

6.2.4.4. SFDR as a Function of the Filter Resolution and the Number of Zero Crossings . . . 94
6.2.5. Symbol Error Rate . . . 94
6.2.6. Folding of Spectral Images when using Cubic Interpolator for Decimation 96

7. Practical Measurements 99
7.1. Measurement Setup . . . 99
7.1.1. Measurement Points . . . 103
7.2. Output Spectra of the USRP N210 . . . 103
7.3. Timing Estimator . . . 105
7.3.1. Influence of the Clock Mismatch . . . 105
7.4. Output Symbols . . . 107
7.4.1. Influence of the Clock Mismatch . . . 108

8. Conclusion 111

A. GNU Radio and UHD 113
A.1. Installation . . . 113
A.1.1. Installation of UHD on Linux . . . 113
A.1.2. Installation of GNU Radio on Linux . . . 114
A.2. UHD Tools . . . 114
A.2.1. UHD Find Devices . . . 114
A.2.2. UHD USRP Probe . . . 115
A.2.3. UHD USRP Net Burner . . . 115
A.3. GNU Radio Companion . . . 115
A.3.1. USRP Source . . . 116
A.4. Developing in GNU Radio . . . 118
A.4.1. Writing Signal Processing Blocks in GNU Radio . . . 118
A.4.2. SWIG . . . 120
A.4.3. GNU Radio Applications . . . 121
A.4.4. Building and Installation of Blocks . . . 121

Bibliography 122

List of Figures

1.2.1. Architecture of the satellite emulator . . . 19
1.2.2. Structure of the delay variation block of the satellite emulator . . . 20
1.2.3. Basic architecture of the software defined radio platform USRP N210 . . . 21

2.1.1. Impulse response (left plot) and magnitude response (right plot) of the cubic interpolation filter . . . 25
2.1.2. Farrow structure of cubic interpolator . . . 26
2.2.1. Sinc function (left hand side, only a few zero crossings around the origin are shown) and its corresponding magnitude response (right hand side) . . . 27
2.2.2. Alignment of the weighted sinc functions for signal reconstruction . . . 28
2.2.3. The sinc function at three different time instances without interpolation . . . 29
2.2.4. The sinc function at three different time instances with interpolation . . . 29
2.2.5. The sinc function at three different time instances with a sampling rate conversion factor of $\rho = \frac{1}{2}$, without interpolation . . . 30
2.2.6. The sinc function at three different time instances with an upsampling factor of $\rho = 2$, without interpolation . . . 31
2.2.7. The sinc function at three different time instances with decimation of $\rho = \frac{1}{2}$ and interpolation . . . 31
2.2.8. The sinc function at four different time instances with an upsampling factor of $\rho = 2$ and interpolation . . . 32
2.2.9. Rectangular window (left hand side) and its corresponding magnitude response (right hand side) . . . 33
2.2.10. Kaiser window (left hand side) and its corresponding magnitude response (right hand side) for different values of β . . . 34
2.2.11. Illustration of output sample computation for time P . . . 35
2.2.12. Filter table content for $nb_l = 7$ and $N_z = 3$ . . . 36
2.2.13. Composition of the time register t . . . 36
2.2.14. Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 0$ (top plot). The bottom plot shows a zoom into the passband of the magnitude response. . . . 40
2.2.15. Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 4$ (top plot). The bottom plot shows a zoom into the passband of the magnitude response. . . . 41
2.2.16. Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 8$ (top plot). The bottom plot shows a zoom into the passband of the magnitude response. . . . 42
2.2.17. Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 0$ . . . 43
2.2.18. Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 4$ . . . 44

2.2.19.Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 8$ . . . 44

3.3.1.Block diagram of the Oerder and Meyr algorithm . . . 49
3.3.2.Illustration of the timing estimator result . . . 50
3.4.1.Analog timing recovery . . . 51
3.4.2.Hybrid timing recovery . . . 51
3.4.3.Digital timing recovery . . . 51
3.4.4.Block diagram of the implemented timing recovery . . . 52
3.4.5.GNU Radio flow graph of the combination of interpolator and estimator for timing recovery . . . 53
3.4.6.Timing update from $\hat{\varepsilon}_{old} = -1.9$ to $\hat{\varepsilon}_{old} = +1.9$ . . . 54

4.0.1.Receive impulse sequence with Nyquist pulse shaping . . . 55

5.1.1.Simplified block diagram of the USRP N210 and WBX daughterboard including the implemented block for timing recovery . . . 58
5.1.2.Block diagram of existing receiver structure on the USRP N210 . . . 59
5.1.3.DC removal filter using fixed-point quantization[Lyo11] . . . 60
5.1.4.Single stage decimation CIC filter[Lyo11] . . . 61
5.1.5.Magnitude response of the four stage CIC filter with decimation factor of 25 . . . 62
5.1.6.Zoom into the magnitude response of the four stage CIC filter with decimation factor of 25 . . . 62
5.1.7.Magnitude response of the small half-band filter . . . 63
5.1.8.Magnitude response of the second half-band filter . . . 63
5.2.1.Impulse response of the implemented root-raised-cosine filter . . . 64
5.2.2.Magnitude response of the implemented root-raised-cosine filter . . . 65
5.3.1.Block diagram of the timing estimator module . . . 66
5.3.2.Block diagram of the complex_square module . . . 67
5.3.3.Block diagram of the Fourier module . . . 68
5.3.4.Block diagram of the CORDIC module . . . 69
5.3.5.Block diagram of the phase_norm module . . . 70
5.3.6.Block diagram of the rounding module . . . 72
5.4.1.Block diagram of the interpolation module . . . 73
5.4.2.Block diagram of the delay_line module . . . 74
5.4.3.Impulse response of the implemented Kaiser windowed sinc filter with $N_z = 2$ and $\beta = 4$ . . . 76
5.4.4.Magnitude response of the implemented Kaiser windowed sinc filter with $N_z = 2$ and $\beta = 4$ . . . 76
5.4.5.Block diagram of the sinc_fir module . . . 77
5.5.1.Device utilization of the FPGA generated by Xilinx ISE Design Suite . . . 78
5.5.2.Module level utilization of the FPGA generated by Xilinx ISE Design Suite . . . 79

6.1.1.Variance of the timing estimator for different SNR values and window lengths L 81
6.1.2.Variance of the timing estimator compared to the lower bounds for window lengths $L = 64$ (left plot) and $L = 256$ (right plot) . . . 82
6.1.3.Variance of the timing estimator compared to the lower bounds for window lengths $L = 512$ (left plot) and $L = 1024$ (right plot) . . . 82
6.2.1.Spectrum of the complex sinusoidal input signal . . . 83

6.2.2. Spectrum of the interpolator output signal using floating point values for the filter tap resolution . . . 84
6.2.3. Spectrum of the interpolator output signal using an amplitude resolution of 4 bit . . . 85
6.2.4. Spectrum of the interpolator output signal using an amplitude resolution of 8 bit . . . 85
6.2.5. Spectrum of the interpolator output signal using an amplitude resolution of 12 bit . . . 86
6.2.6. Spectrum of the interpolator output signal using an amplitude resolution of 16 bit . . . 86
6.2.7. Spectrum of the QPSK input signal . . . 87
6.2.8. Spectrum of the interpolator output signal using a filter tap resolution of 8 bit and three zero crossings for the left plot and 11 zero crossing for the right plot 88
6.2.9. Spectrum of the interpolator output signal using a filter tap resolution of 16 bit and three zero crossings for the left plot and 11 zero crossing for the right plot . . . 88
6.2.10. Spectrum of the sinusoidal input signal for the decimation process affected by noise . . . 89
6.2.11. Output spectrum of the decimator using one zero crossing (left plot) and three zero crossings (right plot) . . . 90
6.2.12. Output spectrum of the decimator using five zero crossings (left plot) and seven zero crossings (right plot) . . . 90
6.2.13. Output spectrum of the decimator using nine zero crossings (left plot) and 11 zero crossings (right plot) . . . 90
6.2.14. Output spectrum of the cubic interpolator for decimation by 2 . . . 91
6.2.15. Example of an interpolator output spectrum for the measurement of the SFDR 91
6.2.16. Example of a cubic interpolator output spectrum for the measurement of the SFDR . . . 92
6.2.17. SFDR for different amplitude resolutions in dependency of the number of zero crossings N_z . . . 93
6.2.18. SFDR for different linear resolutions nb_η in dependency of the filter resolution nb_l . . . 93
6.2.19. SFDR for different numbers of zero crossings N_z in dependency of the filter resolution nb_l . . . 94
6.2.20. Symbol error rates of a QPSK signal using the implemented timing recovery for different numbers of zero crossings N_z . . . 95
6.2.21. Symbol error rates of a QPSK signal using the implemented timing recovery for different window sizes L of the timing estimator and $N_z = 4$ . . . 95
6.2.22. Input spectrum for the aliasing illustration of the cubic interpolator . . . 96
6.2.23. Output spectrum of the cubic interpolator (left plot) and FIR interpolator (right plot) using a sampling conversion factor of $\rho = \frac{1}{2}$ . . . 97
6.2.24. Output spectrum of the cubic interpolator (left plot) and FIR interpolator (right plot) using a sampling conversion factor of $\rho = 0.4706$ . . . 97

7.1.1. Setup of the components for the measurements on the implemented modules in hardware . . . 100
7.1.2. Configuration overview of the modulator . . . 100
7.1.3. Demodulator state . . . 101

7.1.4.Screenshot of the signal analyzer showing a spectrum of a $f_{sym} = 1\,MBaud$ DVB-S2 signal at a center frequency of $f_{center} = 1\,GHz$ . . . 101
7.1.5.Screenshot of the logic analyzer showing clock and strobe signals of the implemented hardware modules on the USRP N210 . . . 102
7.1.6.Screenshot of the logic analyzer showing the output of the ADC on the USRP N210 . . . 102
7.2.1.Spectrum of the received QPSK signal before (left) and after (right) the root-raised-cosine filter at a signal-to-noise ratio of $SNR = 5\,dB$ . . . 104
7.2.2.Spectrum of a noise-only-signal before (left) and after (right) the root-raised-cosine filter . . . 104
7.3.1.Timing estimator variance of the hardware module and the simulation . . . 105
7.3.2.Averaged drift of μ due to the clock mismatch . . . 106
7.3.3.Averaged drift of μ using the external reference clock reference . . . 106
7.4.1.Constellation diagram of the received QPSK (left) and 16-APSK (right) symbols with external clock reference and $SNR = 29\,dB$ . . . 107
7.4.2.Constellation diagram of the received QPSK (left) and 16-APSK (right) symbols with external clock reference and $SNR = 15\,dB$ . . . 108
7.4.3.Constellation diagram of the received QPSK (left) and 16-APSK (right) symbols with external clock reference and $SNR = 10\,dB$ . . . 108
7.4.4.Constellation diagram of the received QPSK (left) and 16-APSK (right) symbols with no external clock reference and $SNR = 29\,dB$ . . . 109

A.3.1Sample graph of GNU Radio Companion . . . 116
A.3.2Parameters for USRP source block . . . 118

List of Tables

2.1.1.Coefficients $b_l(i)$ for the cubic interpolator 25
2.2.1.Values for the variables n, l and η for $\rho = 1$ and two different initial time registers 38
2.2.2.Values for the variables n, l and η for $\rho = 2$ and two different initial time registers 38
2.2.3.Values for the variables n, l and η for $\rho = \frac{1}{2}$ and two different initial time registers 39

3.3.1.Phasor values for Fourier coefficient calculation using $N = 4$ 50

5.3.1.Input and output signals of the timing estimation module 66
5.3.2.Input and output signals of the complex_square module 67
5.3.3.Hardware estimation of the complex_square module 67
5.3.4.Input and output signals of the Fourier module 68
5.3.5.Hardware estimation of Fourier module 68
5.3.6.Input and output signals of the CORDIC module 69
5.3.7.Input and output signals of the phase_norm module 71
5.3.8.Hardware estimation of the phase_norm module 71
5.4.1.Input and output signals of the interpolator module 73
5.4.2.Input and output signals of the delay_line module 74
5.4.3.Input and output signals of the sinc_fir module 77

7.1.1.Configuration of the programmable noise generator and the corresponding SNR values 103

List of Acronyms

The following acronyms are used in this thesis:

ACM	Adaptive Coding and Modulation
ADC	Analog-to-Digital Converter
APSK	Amplitude and Phase-Shift Keying
AWGN	Additive White Gaussian Noise
CIC	Cascaded Integrator Comb
CORDIC	Coordinate Rotation Digital Computer
DAC	Digital-to-Analog Converter
DC	Direct Current
DDC	Digital Down Converter
DFT	Discrete Fourier Transform
DSP	Digital Signal Processing
DVB	Digital Video Broadcasting
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GRC	GNU Radio Companion
IP	Intellectual Property
IR	Impulse Response
ISI	Intersymbol Interference
LSB	Least Significant Bit
MIMO	Multiple Input Multiple Output
MSB	Most Significant Bit
NCO	Numerically Controlled Oscillator
NDA	Non Data Aided
PSK	Phase-Shift Keying

QPSK	Quadrature Phase-Shift Keying
RF	Radio Frequency
RTT	Round-Trip Time
SFDR	Spurious Free Dynamic Range
SDR	Software Defined Radio
SINC	Sine Cardinal
TDP	Technological Demonstration Payload
UDP	User Datagram Protocol
UHD	Universal Hardware Driver
USRP	Universal Software Radio Peripheral

1. Introduction

1.1. Motivation

Interpolation in digital signal processing is used to calculate signal values at arbitrary times. By using interpolation techniques, sampling rate conversions can be obtained, for example to reduce the computational effort of the following signal processing components.
In this thesis, a specific interpolation technique is analyzed and implemented for two different interpolation applications.
The first application is a satellite emulator. Since the round-trip time (RTT) of a satellite communication link is varying due to the movement of the satellite and the consequential varying distance to the ground station, this delay variation needs to be performed when emulating a satellite communication link. One possibility of this delay variation emulation is by performing an appropriate rate conversion on the input samples.
Another application of the interpolator in this thesis is the recovery of the symbol timing. In this application, an interpolator is used in the context of an asynchronous sampling clock, i.e., the sampling grid does not necessarily include the ideal sampling instant. As a consequence, this introduces an impairment in terms of inter-symbol-interference degrading the performance in digital receivers.

1.2. Use Case Scenarios for the Interpolator

1.2.1. Delay Variations in a Satellite Emulator

Joanneum Research currently works on a project to develop an emulator for a satellite communication link. The movement of a satellite relatively to the ground station leads not only to a Doppler effect, but also to a delay variation due to the varying distance. Therefore the architecture of this emulator consists of a frequency error block simulating the Doppler effect, a noise block and the mentioned delay variation block. The basic architecture of this satellite emulator is illustrated in figure 1.2.1.

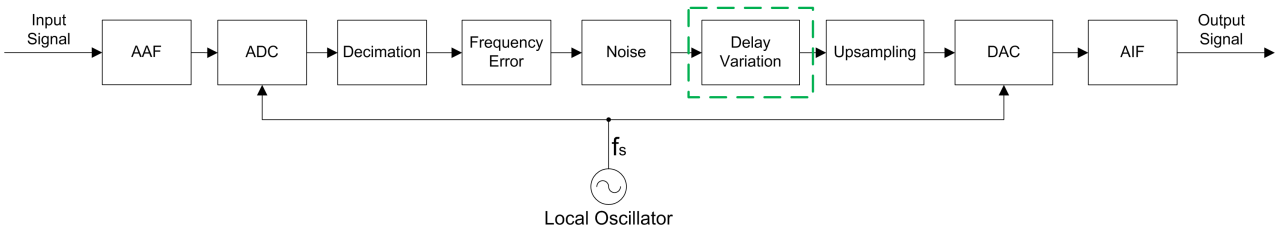


Figure 1.2.1.: Architecture of the satellite emulator

In the satellite emulator project of Joanneum Research, this delay variation block consists of a sampling rate changer and a FIFO with different read and write rates, which is illustrated in figure 1.2.2. The input sampling rate f_s is converted to a new sampling rate f_s' with respect to the movement of the satellite. The resulting samples are stored in a FIFO. The samples in the FIFO are then interpreted as samples of the input sampling frequency f_s by the DAC as illustrated in figure 1.2.1. Since the sampling rates f_s and f_s' are different, the number of samples in the FIFO increases or decreases according to their ratio.

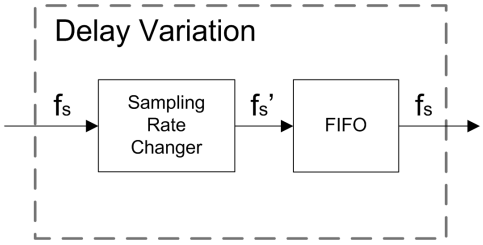


Figure 1.2.2.: Structure of the delay variation block of the satellite emulator

1.2.1.1. Task of this Thesis

The task of this thesis for the satellite emulator application is to implement an interpolator for this sampling rate conversion. Additionally, a performance analysis is applied to identify the capabilities of the implemented interpolator. For the implementation and simulation part, a simulation environment developed by Joanneum Research is used. These results also provide the fundamentals for a potential hardware implementation of this delay variation block in the satellite emulator.

1.2.2. Timing Recovery on a Software Defined Radio Platform

Another project at Joanneum Research is to develop an SNR estimator for the adaptive coding and modulation (ACM) experiment TDP5 on the ALPHASAT satellite, which is launched at the end of 2012 by the European Space Agency (ESA). In this project, a software defined radio (SDR) platform is used to implement the required estimation and synchronization tasks. It consists of a universal software radio peripheral (USRP) device by Ettus Research which is connected to a host PC. The basic architecture is illustrated in figure 1.2.3.
The USRP N210 consists of an analog receiver frontend (WBX daughterboard) which is responsible for signal amplification, complex mixing and anti-alias filtering (AAF). The USRP N210 motherboard converts the analog output of the WBX daughterboard into digital samples (ADC) and decimates these samples by applying a cascaded integrator comb (CIC) filter as well as a half-band filter (HBF). The motherboard also contains the root-raised-cosine (RRC) matched filter and the timing recovery module, which consists of timing estimation and interpolation. The resulting samples are then used by a differential correlator to find the frame start in the transmitted symbols. The symbols and the result of the correlator are sent to a host PC via Ethernet where the final SNR estimation is performed.
The implementation of the SNR estimation part of this project has been detailed in [Tür12]. In figure 1.2.3 this part is indicated by the red frame. The current thesis investigates the implementation of the timing recovery task, which is indicated by the green frame. The other modules illustrated in figure 1.2.3 are already implemented on the USRP N210.

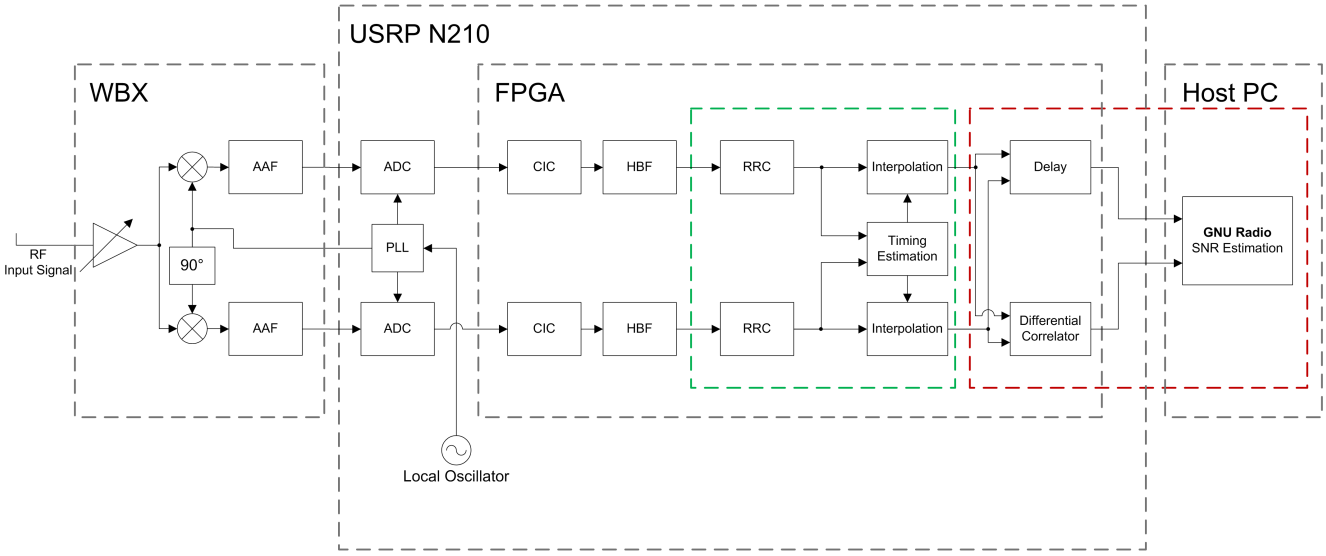


Figure 1.2.3.: Basic architecture of the software defined radio platform USRP N210

1.2.2.1. Task of this Thesis

The task of this thesis is to implement a timing recovery module on the USRP N210 which consists of a root-raised-cosine filter, a timing estimator and an interpolation module. This module is used to extract the symbols out of the input signal prior to the SNR estimation process.

1.3. GNU Radio

GNU Radio is an open source toolkit which allows users to combine signal processing modules in order to generate software defined radios (SDR). It is available on the GNU Radio website http://gnuradio.org[1] which also contains all necessary information on the toolkit.
The software is licensed under the GNU GPL version 3 and is available for the platforms Linux and Windows.
In this thesis GNU Radio was used as a development environment for the high level models of the modules that are later implemented on the FPGA. Additionally, the toolkit was used for verification of these models as well as for generating test vectors for the FPGA modules verification process during the hardware implementation.
This chapter gives a brief introduction to the software environment of GNU Radio and also of the USRP hardware devices. The usage of GNU Radio as well as the procedure of developing software modules is described in appendix A.

1.3.1. Hardware

The GNU Radio software works with several hardware components, although hardware components are not necessarily required for working with the toolkit. There are many blocks already provided by the software which allow reading and writing data of different file formats.

[1] This URL was checked for validity in March 2012

Pre-recorded examples are available which allows the user to develop and simulate applications completely without the need of any additional hardware.
But if working with signals from real applications and not only with simulated or pre-recorded signals, GNU Radio supports several hardware components.

1.3.1.1. Supported Hardware

The most common and cheapest hardware component is the sound card interface which is nowadays available on most computers. This hardware component usually provides stereo input and output which would correspond to two channels for digital signal processing. Many simple DSP applications can be realized using this common hardware device.
Other supported hardware devices are presented on the GNU Radio website like Perseus and Comedi. For more information on these devices please visit the GNU Radio website that is given at the beginning of this chapter.

1.3.1.2. USRP

Some very powerful and capable devices are the USRP series developed by Ettus Research. USRP stands for Universal Software Radio Peripheral and is a family, which consists of many different motherboards using either USB or Gigabit Ethernet interfaces for communication with a host PC. But these boards also can be used as standalone devices or even in a network of devices for MIMO applications.
These boards support reception and transmission of signals up to $5.8\,GHz$ with sampling rates up to $100\,MHz$.
The practical FPGA part of this thesis is implemented on the USRP N210 device by Ettus Research. The detailed description of this device is given in chapter 5.
For the support of these USRP devices in GNU Radio, the universal hardware driver (UHD) is integrated in GNU Radio.

1.3.1.3. Universal Hardware Driver

The universal hardware driver is a project by Ettus Research that enables the usage of every USRP device in the GNU Radio toolkit. Also other applications like LabView and Simulink support this driver and can therefore be used with any USRP device. Also the possibility of using this driver in self-made toolkits is given. The UHD is developed for the platforms Linux, Windows and Mac.
The usage and installation of UHD is described in appendix A.

2. Interpolation and Digital Sampling Rate Conversion

This chapter recaps the theory and implementation of interpolation and digital sampling rate conversion for the FIR-based interpolation as well as for the polynomial based interpolation, which is used for the comparison of implementation complexity and performance.
The major goal of interpolation and digital sampling rate conversion is to compute signal values at arbitrary times in digital signal processing.

2.1. Polynomial Interpolation

First of all, the polynomial-based interpolation technique is discussed. Lots of literature exist for this kind of interpolation due to its popularity. Therefore, the algorithm is detailed for comparison purposes with the algorithm presented later in this thesis. Hence, the basic points of this procedure are presented in this section.
In this class of interpolators the impulse response of the underlying continuous filter is a polynomial or a piecewise polynomial, which is fitted to a set of signals samples such that the value of the polynomial exactly matches the sample values. This polynomial then gets evaluated at the desired time instance.
In [EGH93] the fundamental interpolation equation is given by the following formula:

$$y\left(k \cdot T_i\right) = y\left[\left(m_k + \mu_k\right) \cdot T_s\right] = \sum_{i=I_1}^{I_2} x\left[\left(m_k - i\right) \cdot T_s\right] \cdot h_I\left[\left(i + \mu_k\right) \cdot T_s\right] \tag{2.1.1}$$

where $x\left(m\right)$ are the signal samples at intervals T_s, $h_I\left(t\right)$ is a continuous interpolation filter, m_k is the basepoint index, μ_k is the fractional interval and T_i is the sample interval of the output samples.
When using polynomial filters for the interpolation, the Lagrange coefficient formula (equation 2.1.3) can be used to create the interpolating polynomial. So the interpolation polynomial can be constructed according to [Gar90]

$$p\left[\left(m_k + \mu_k\right) \cdot T_s\right] = y\left[\left(m_k + \mu_k\right) \cdot T_s\right] = \sum_{l=I_1}^{I_2} L_l\left[\left(m_k + \mu_k\right) \cdot T_s\right] \cdot x\left[\left(m_k - i\right) \cdot T_s\right] \tag{2.1.2}$$

where

$$L_l\left(t\right) = \prod_{j=0, j\neq l}^{N} \frac{t - t_j}{t_l - t_j} \tag{2.1.3}$$

Using $t = \left(m_k + \mu_k\right) \cdot T_s$, the fact that $t_l - t_j$ is an integer multiple of the sampling time T_s and by setting the ranges of the product from I_1 to I_2, equation 2.1.3 leads to

$$L_l(\mu_k) = \prod_{j=I_1, j\neq l}^{I_2} \frac{\mu_k + j}{j - l} \tag{2.1.4}$$

Having a look at equation 2.1.2 and comparing it with equation 2.1.1, we observe that the filter impulse response can be calculated using the Lagrange formula given in equation 2.1.4. A very common kind of polynomial interpolator is the cubic interpolator which has a degree of $N = 3$. Hence, the resulting filter has $N + 1 = 4$ taps. When using this order, we can interpret equation 2.1.4 as [Gar90]

$$\begin{array}{rcl} L_1(\mu_k) & = & -\frac{1}{6} \cdot (\mu_k - 2) \cdot (\mu_k - 1) \cdot \mu_k \\ L_0(\mu_k) & = & \frac{1}{2} \cdot (\mu_k - 2) \cdot (\mu_k - 1) \cdot (\mu_k + 1) \\ L_{-1}(\mu_k) & = & -\frac{1}{2} \cdot (\mu_k - 2) \cdot \mu_k \cdot (\mu_k + 1) \\ L_{-2}(\mu_k) & = & \frac{1}{6} \cdot (\mu_k - 1) \cdot \mu_k \cdot (\mu_k + 1) \end{array} \tag{2.1.5}$$

by using the taps from $I_1 = -\frac{N+1}{2} = -2$ to $I_2 = \frac{N+1}{2} - 1 = 1$.
Now we can see that the resulting piecewise polynomial impulse response of the filter is given by [EGH93]:

$$h_I[(i + \mu_k) \cdot T_s] = L_i(\mu_k) = \sum_{l=0}^{N} b_l(i) \cdot \mu_k^l \tag{2.1.6}$$

where $b_l(i)$ are fixed coefficients which are independent from μ_k.
According to [EGH93], equation 2.1.2 can now be rearranged using equation 2.1.6

$$\begin{array}{rcl} y(k) & = & \sum_{i=I_1}^{I_2} x(m_k - i) \cdot \sum_{l=0}^{N} b_l(i) \cdot \mu_k^l \\ & = & \sum_{l=0}^{N} \mu_k^l \cdot \sum_{i=I_1}^{I_2} b_l(i) \cdot x(m_k - i) \\ & = & \sum_{l=0}^{N} \mu_k^l \cdot v(l) \end{array} \tag{2.1.7}$$

where

$$v(l) = \sum_{i=I_1}^{I_2} b_l(i) \cdot x(m_k - i) \tag{2.1.8}$$

This defines the input-to-output relation for the polynomial interpolator using the Lagrange coefficients.

2.1.1. Implementation of the Polynomial Interpolator

For a cubic interpolator, equation 2.1.7 leads to the nested evaluation:

$$y(k) = \{[v(3) \cdot \mu_k + v(2)] \cdot \mu_k + v(1)\} \cdot \mu_k + v(0) \tag{2.1.9}$$

where

$$v(l) = \sum_{i=-2}^{1} b_l(i) \cdot x(m_k - i)$$

The coefficients $b_l(i)$ can be extracted from equation 2.1.5 and are summarized in table 2.1.1 which is also given in [EGH93].

i	$l=0$	$l=1$	$l=2$	$l=3$
-2	0	$-\frac{1}{6}$	0	$\frac{1}{6}$
-1	0	1	$\frac{1}{2}$	$-\frac{1}{2}$
0	1	$-\frac{1}{2}$	-1	$\frac{1}{2}$
1	0	$-\frac{1}{3}$	$\frac{1}{2}$	$-\frac{1}{6}$

Table 2.1.1.: Coefficients $b_l(i)$ for the cubic interpolator

The corresponding impulse response of the cubic interpolator can be seen in figure 2.1.1. The impulse response is piecewise cubic polynomial in T_s which can be seen in this plot. Also the fact that the impulse response has its zeros at all non-zero integer multiples of T_s ensures the exact interpolation of the basepoint set.

When looking at the magnitude response of the cubic interpolator shown in figure 2.1.1, we can see the broad main lobe that results from the narrow main lobe in the impulse response. This wide main lobe makes prefiltering necessary when using this cubic interpolation for decimation procedures, where unwanted signal components in the spectrum are present and need to be filtered to avoid aliasing. On the other hand, these high sidelobes results in a folding of the spectral images onto the desired signal when resampling is applied. This can lead to impairments in the resulting output spectrum if the decimation rate is not an integer value, like for example $\rho = 0.4706$. An illustration of this occuring effect is shown in section 6.2.6.

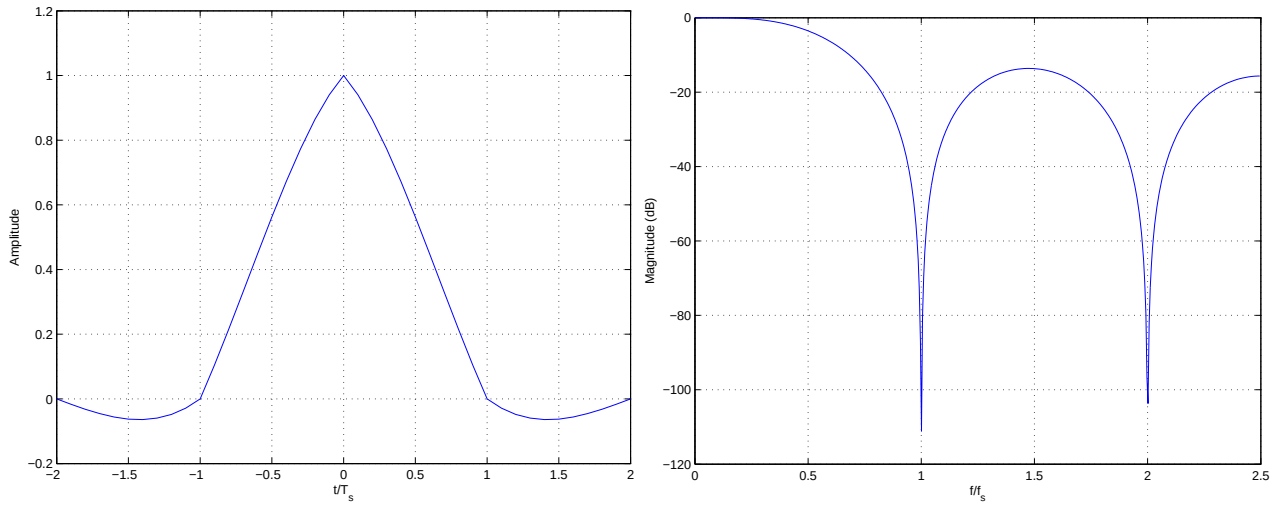


Figure 2.1.1.: Impulse response (left plot) and magnitude response (right plot) of the cubic interpolation filter

2.1.1.1. Farrow Structure

In the Farrow structure proposed by [EGH93] which consists of $N + 1 = 4$ columns of FIR filters. Each filter has a fixed set of filter coefficients. This is a very efficient approach, since all the computations in equation 2.1.9 are performed online and there is no need for additional filter coefficient memory.
Figure 2.1.2 shows the Farrow structure of the cubic interpolator. The corresponding coefficients $b_l(i)$ are listed in table 2.1.1.

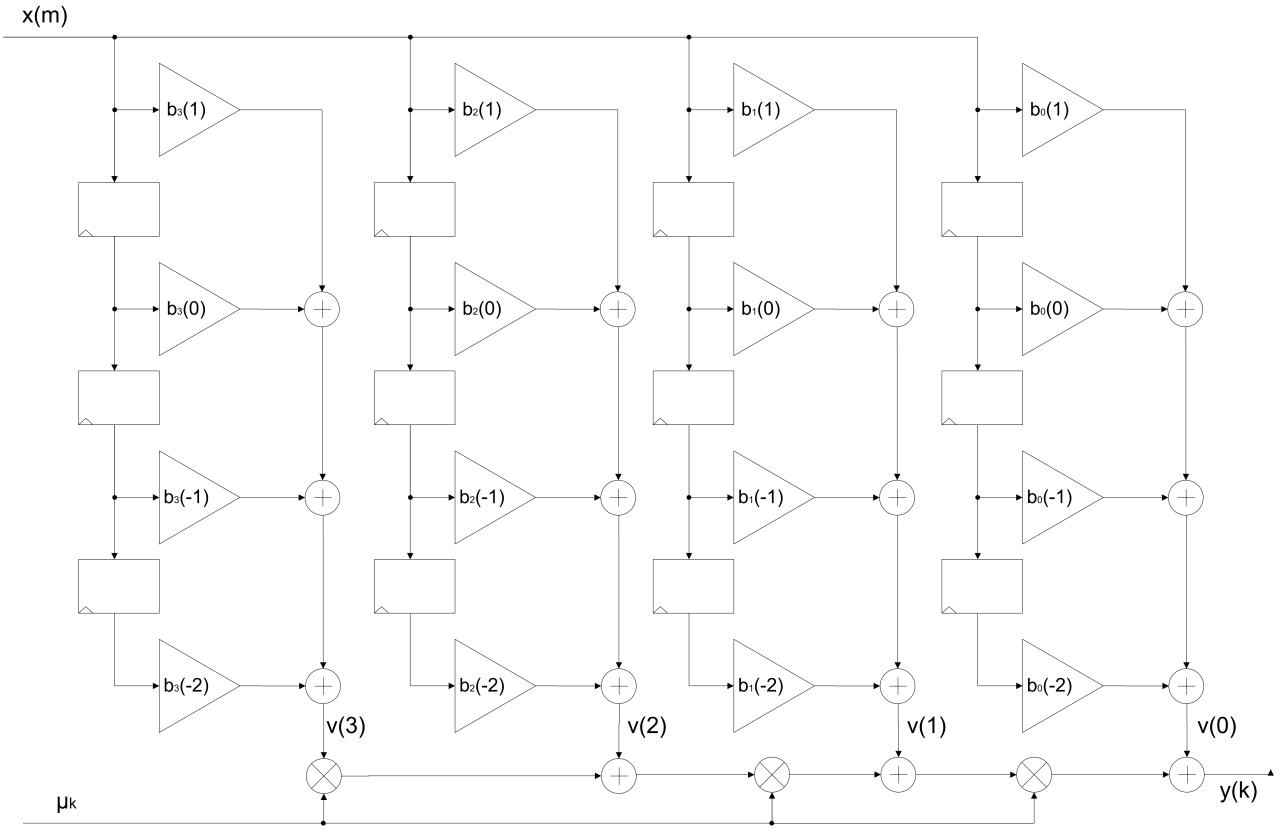


Figure 2.1.2.: Farrow structure of cubic interpolator

Since the cubic interpolator is a very popular interpolator in digital receivers, the hardware implementation of this Farrow structure can be found in literature. For example in [MB07], the VHDL source code for such an interpolator is given and only the rate change needs to be adapted.
If we compare this implementation with the FIR implementation, we can see that no coefficient table is needed since they are independent from μ_k. However, 15 (19 multipliers - 4 zero coefficients) multiplications are needed, which would be equal to a hardware effort for a FIR filter with 15 taps.

2.2. FIR Interpolation

The FIR interpolation is based on the sampling theorem [MMF98] and the basic structure of the FIR algorithm is presented in [Smi11].
In this respect, $x(n \cdot T_s)$ represents the samples of a continuous signal $x(t)$, where t is the time, n an integer number and T_s is the sampling period. We assume that $x(t)$ is bandlimited to $\pm\frac{F_s}{2}$, where $F_s = \frac{1}{T_s}$ is the sampling frequency.
The sampling theorem says that the original signal can be perfectly reconstructed from the samples $x(n \cdot T_s)$ using the formula:

$$x(t) = \sum_{n=-\infty}^{+\infty} x(n \cdot T_s) \cdot \text{sinc}\left(\frac{t - n \cdot T_s}{T_s}\right) \tag{2.2.1}$$

where

$$\text{sinc}\left(\frac{t}{T_s}\right) = \frac{\sin\left(\pi \cdot \frac{t}{T_s}\right)}{\pi \cdot \frac{t}{T_s}}$$

is the sine cardinal or sinc function.
The sinc function and its Fourier transform are illustrated in figure 2.2.1. The maximum amplitude is one and all the zeros are at non-zero integer values of T_s. Shifted to the frequency domain, the sinc function shows a rectangular shape

$$\text{rect}\left(\frac{f}{F_s}\right) = \begin{cases} 1 & |f| \leq \frac{F_s}{2} \\ 0 & |f| > \frac{F_s}{2} \end{cases}$$

of an ideal lowpass filter. The cut-off frequency is at $f_c = \frac{F_s}{2}$, hence it is frequently denoted as a brick-wall filter, which means that this filter perfectly cuts off frequencies higher than f_c and passes all frequencies equal or lower than f_c.

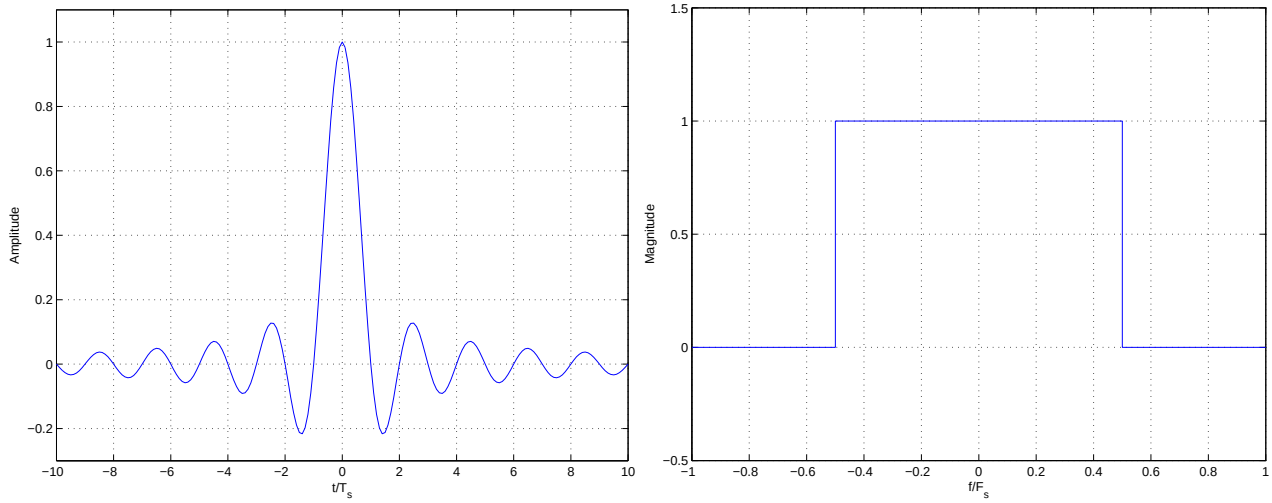


Figure 2.2.1.: Sinc function (left hand side, only a few zero crossings around the origin are shown) and its corresponding magnitude response (right hand side)

2.2.1. Interpretation of the Sampling Theorem

Equation 2.2.1 shows the convolution of signal $x(n \cdot T_s)$ with the sinc function sinc $\left(\frac{t-n \cdot T_s}{T_s}\right)$. Due to the fact that the convolution is commutative, equation 2.2.1 can be interpreted in two different ways.

The first way is to see the equation as the sum of weighted sinc functions. In this case, one sinc function for every input sample is shifted to its corresponding time position. The sinc functions then get weighted by the related input sample and the results are added together. As mentioned before, the sinc function is zero at all non-zero integer values. This means that if we have a look at a specific sinc function in figure 2.2.2, we can see that there is only an impact from a single input sample because the value of the sinc function is zero at all other sample positions. If we finally sum up all the weighted sinc functions, we get a signal passing exactly the input samples of the original signal.

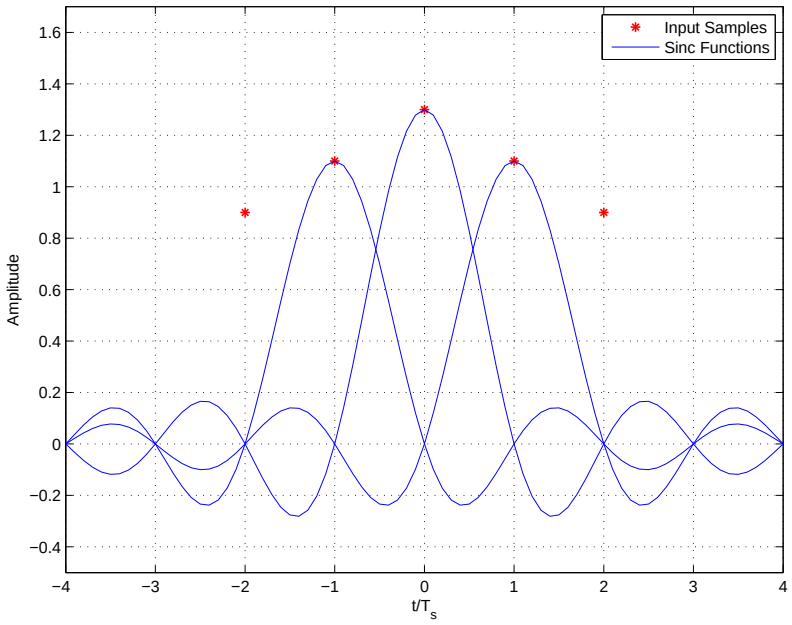


Figure 2.2.2.: Alignment of the weighted sinc functions for signal reconstruction

Another way of looking at equation 2.2.1 is to shift the input samples over one single sinc function. This means that the input samples are shifted such that the peak of the sinc function is at the time position for which the output sample is requested. The output is calculated by multiplying the input samples with the sinc values at the position of the input samples which corresponds to a filter operation. Figure 2.2.3 shows the sinc function at three different time instances where every time instance of the sinc function is indicated with a different colour. At every time instance the sinc function is multiplied with the input samples and these results are added together to compute the desired output value. Then the input samples get shifted by the sampling period before computing the next output.

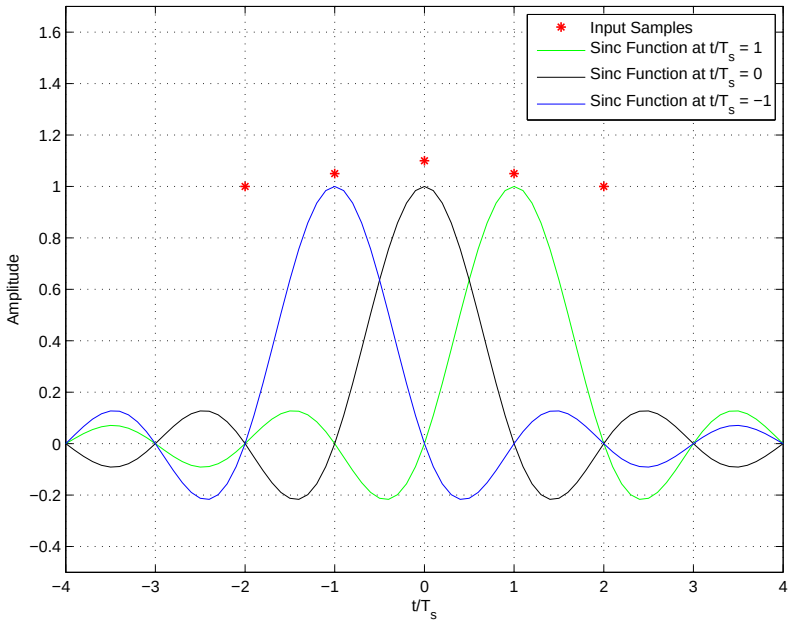


Figure 2.2.3.: The sinc function at three different time instances without interpolation

2.2.2. Interpolation using the Sampling Theorem

To compute the signal value at an arbitrary time between two input samples, which is needed for interpolation and sampling rate conversion, the input samples need to be shifted such, that the peak of the sinc function is at the time position of the desired signal value. Now each input sample again gets multiplied with the sinc value at its time position and all the results are added together to compute the desired output value. If we have a look at figure 2.2.4, where we want to calculate the signal value at the time P, we observe that every input sample influences the output at the time instance P.

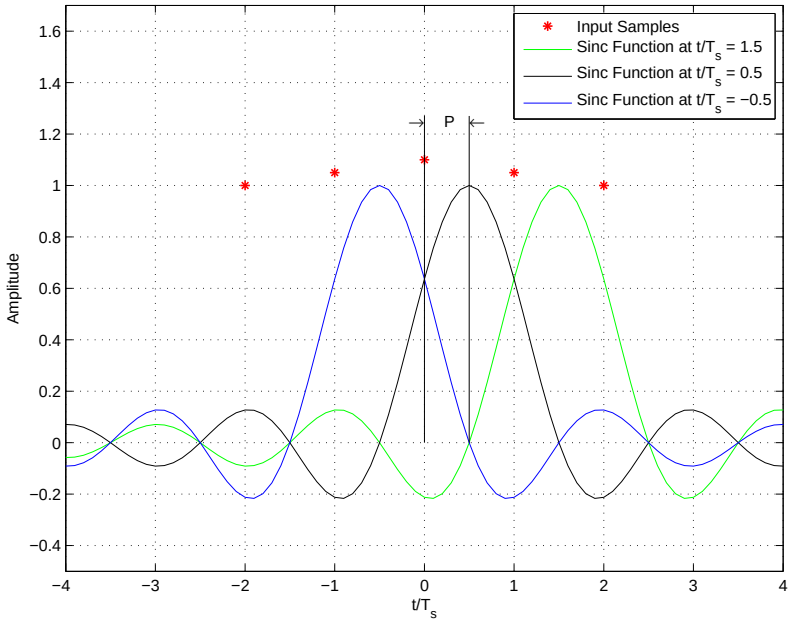


Figure 2.2.4.: The sinc function at three different time instances with interpolation

In order to change the sampling rate to $\tilde{F}_s = \frac{1}{\tilde{T}_s}$, we need to evaluate the sampling formula (equation 2.2.1) at integer multiples of $\tilde{T}_s$. If the new sampling rate is lower than the original one ($F_s > \tilde{F}_s$), the cut-off frequency of the ideal lowpass filter has to be set to the half of the new sampling rate: $\tilde{f}_c = \frac{\tilde{F}_s}{2}$. This means in general that the sinc function in equation 2.2.1 needs to be replaced by the new sinc function $\text{sinc}\left(\min\left(\frac{1}{T_s}, \frac{1}{\tilde{T}_s}\right) \cdot t\right)$. Figure 2.2.5 shows the sinc function for three different time instances in case of an conversion factor of $\rho = \frac{\tilde{F}_s}{F_s} = \frac{1}{2}$, which equals a decimation by 2. Now the input samples are shifted by $\tilde{T}_s = \frac{T_s}{\rho}$ before calculating the output sample which is twice the sampling period in this example. As it can be seen, the time duration of one zero crossing of the sinc function is increased when using $\rho = \frac{1}{2}$. This means that the cut-off frequency of the filter is decreased.

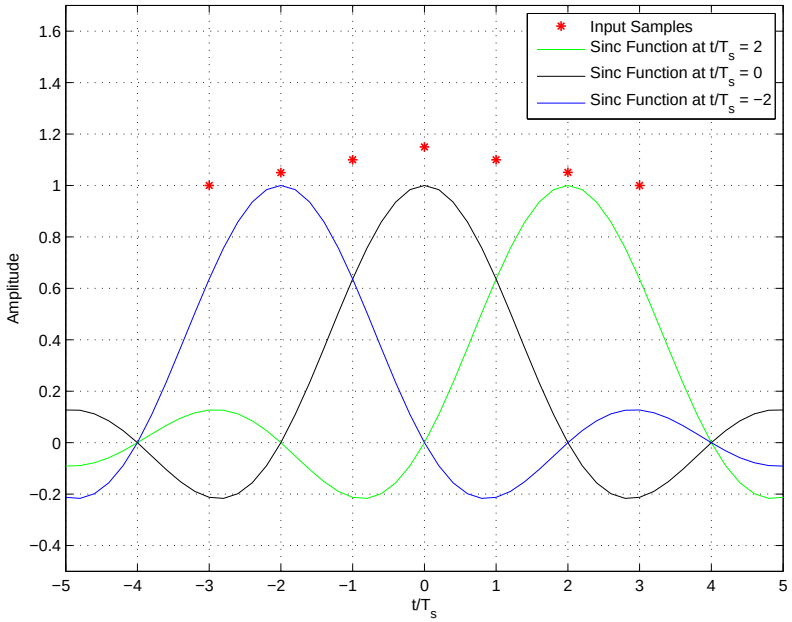


Figure 2.2.5.: The sinc function at three different time instances with a sampling rate conversion factor of $\rho = \frac{1}{2}$, without interpolation

Figure 2.2.6 shows the sinc function for three different time instances in case of an upsampling factor of $\rho = \frac{\tilde{F}_s}{F_s} = 2$. Now the input signal needs to be shifted by $\tilde{T}_s = \frac{T_s}{\rho}$ after producing one output value which is half the sampling period in this example.

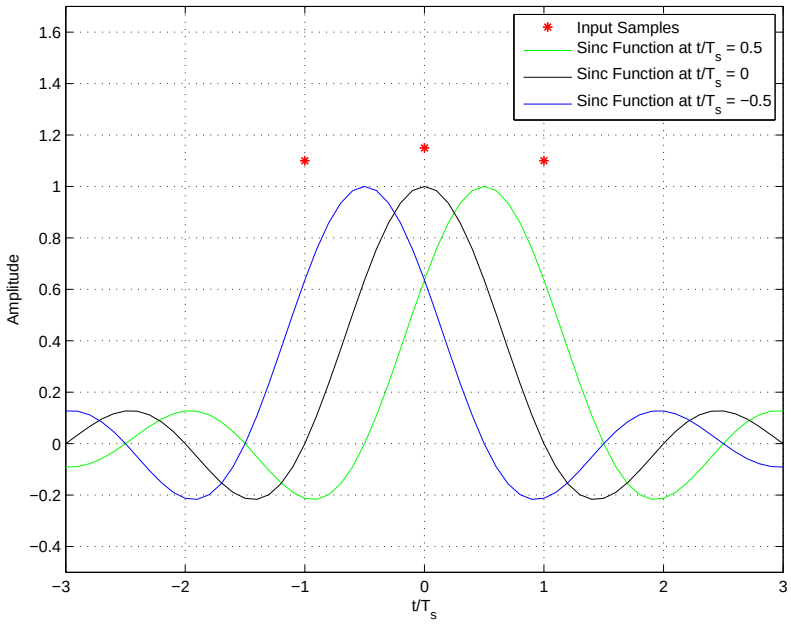


Figure 2.2.6.: The sinc function at three different time instances with an upsampling factor of $\rho = 2$, without interpolation

If decimation plus interpolation is requested, the input samples need to be shifted additionally by P. After computing an output value, the input signal has to be shifted by $\tilde{T}_s = \frac{T_s}{\rho}$ which is twice the sampling period in the example configuration in figure 2.2.7.

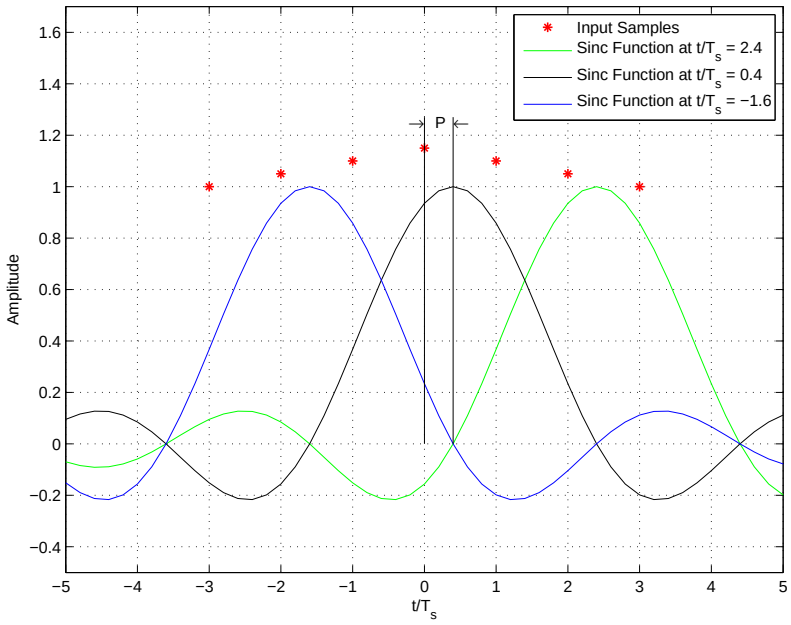


Figure 2.2.7.: The sinc function at three different time instances with decimation of $\rho = \frac{1}{2}$ and interpolation

In case of upsampling plus interpolation the input samples need to be shifted additionally by P compared to the pure upsampling case. After computing an output value, the input

samples need to be shifted by $\tilde{T}_s = \frac{T_s}{\rho}$ which is half the sampling period in the example configuration in figure 2.2.8.

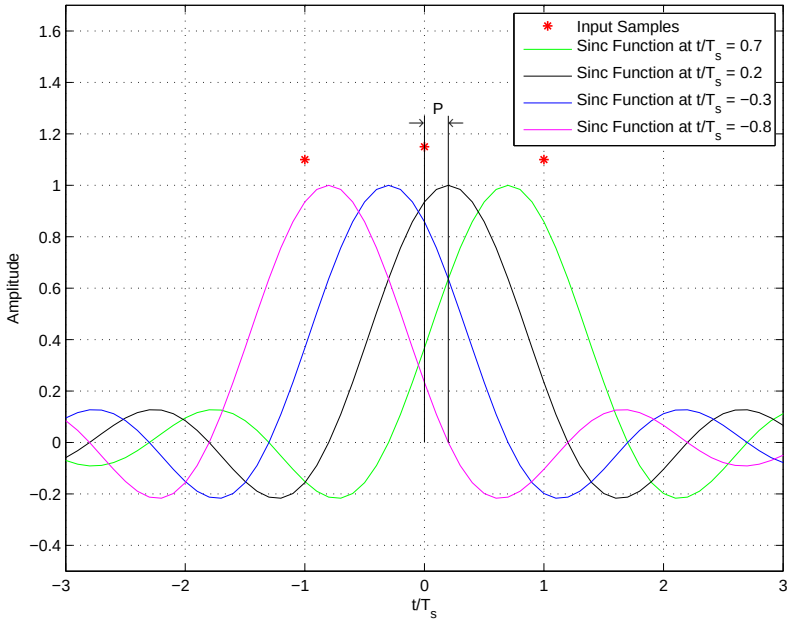


Figure 2.2.8.: The sinc function at four different time instances with an upsampling factor of $\rho = 2$ and interpolation

2.2.3. Windowing of the Sinc Function

When working on real systems the ideal impulse lowpass filter cannot be implemented because of its non-causal impulse response which extends from negative infinity to positive infinity. In order to cope with this problem, the infinite impulse response of the ideal lowpass filter needs be truncated to become finite. This can be obtained using windowing methods.
When windowing the ideal lowpass filter, its impulse response is multiplied with a window function. There are several different kinds of window functions mentioned in the literature of digital signal processing for example in [PM96]. Two window functions are explored in this thesis. First the rectangular window is analyzed, which is a very common window due to its simplicity since the filter IR only needs to be truncated. Secondly, the Kaiser window is presented due to its superior performace proposed in [Smi11].

2.2.3.1. Rectangular Window

The rectangular window is a function which has the value 1 in a specific interval $[-N, N]$ and the value 0 outside of this interval. Its function is given by:

$$w_{rect}[n] = \begin{cases} 1 & |n| \leq N \\ 0 & |n| > N \end{cases}$$

In figure 2.2.9 an example of the rectangular window and its Fourier transform are shown. The Fourier transform of the rectangular window is given in [PM96] by

$$W_{rect}(\omega) = \sum_{n=0}^{M-1} e^{-j\cdot\omega\cdot n} = e^{-j\cdot\omega\cdot\frac{(M-1)}{2}} \cdot \frac{\sin\left(\omega\cdot\frac{M}{2}\right)}{\sin\left(\frac{\omega}{2}\right)}$$

where M is the number of samples of the window. As can be seen, the abrupt change of the amplitude in the window results in a narrow main lobe and quite high sidelobe levels in the spectrum. According to [PM96] and [Lyo11] the main lobe gets narrower the more samples M are used for the window function. But the height of the sidelobes cannot be affected by increasing M.
Other windows reduce this discontinuities in the amplitude in order to reduce the magnitude of the sidelobes in the spectrum.

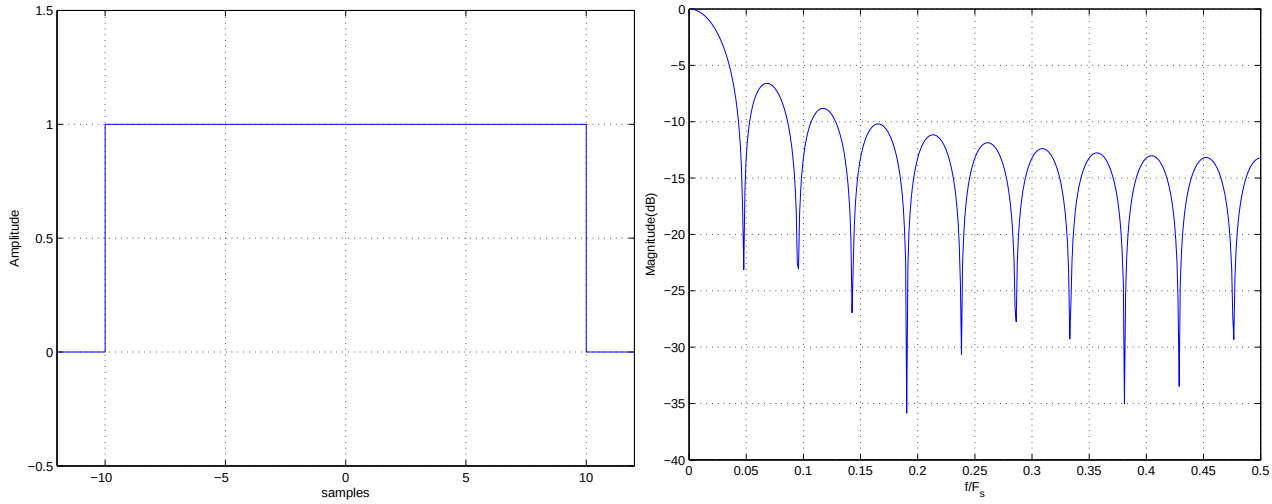


Figure 2.2.9.: Rectangular window (left hand side) and its corresponding magnitude response (right hand side)

2.2.3.2. Kaiser Window

The Kaiser window has the parameter β which is defined in [Kai74] by

$$\beta = \begin{cases} 0.1102\cdot(\alpha - 8.7) & \alpha > 50 \\ 0.5842\cdot(\alpha-21)^{0.4} + 0.07886\cdot(\alpha-21) & 50 \geq \alpha \geq 21 \\ 0 & \alpha < 21 \end{cases} \tag{2.2.2}$$

where α is the sidelobe attenuation in dB.
The parameter β can be used to control the trade off between the main lobe width and the sidelobe levels. The window function is defined in [Lyo11] by

$$w[n] = \frac{I_0\left(\beta\cdot\sqrt{1-\left(\frac{n-p}{p}\right)^2}\right)}{I_0(\beta)} \tag{2.2.3}$$

for $n = 0, 1, \ldots, N-1$and $p = \frac{N-1}{2}$. $I_0(\cdot)$ is the zero-order modified Bessel function of the first kind which can be approximated by the formula given in [Lyo11]

$$I_0(x) = \sum_{i=0}^{24} \frac{x^{2 \cdot i}}{4^i \cdot (i!)^2} \tag{2.2.4}$$

As mentioned before, the parameter β can be used to control the sidelobe attenuation as well as the width of the main lobe. If we have a look at figure 2.2.10, we can see that for low values of β the main lobe is quite narrow but the sidelobe attenuation is low. When increasing β the sidelobe attenuation increases with the drawback of a wider main lobe.

The final choice of β depends on the requested specifications for the filter design. This filter design is used in section 2.2.5.

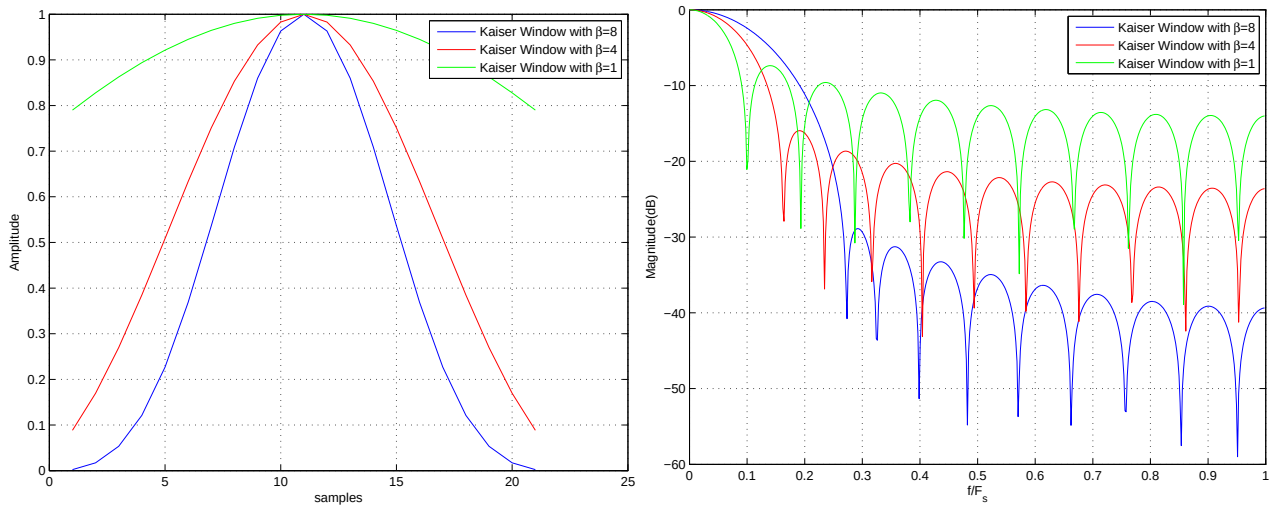


Figure 2.2.10.: Kaiser window (left hand side) and its corresponding magnitude response (right hand side) for different values of β

2.2.4. Implementation of the FIR Interpolator

As already mentioned in section 2.2.1, one interpretation of equation 2.2.1 is to shift the signal samples over one filter IR. Which means that we can shift the input samples such, that the peak of the filter function is at the time position of the desired output sample. The input samples are then multiplied with the filter values at the sample time positions and all these results are then added together in order to compute the desired output sample.

Figure 2.2.11 illustrates this computation where the desired output is shifted by P relative to the input samples. Using this method, any conversion factor of

$$\rho = \frac{F_{out}}{F_{in}} \tag{2.2.5}$$

where F_{in} is the input sampling frequency and F_{out} is the output sampling frequency, can be realized. For $\rho = 1$ simple interpolation, for $\rho > 1$ upsampling and for $\rho < 1$ decimation is performed.

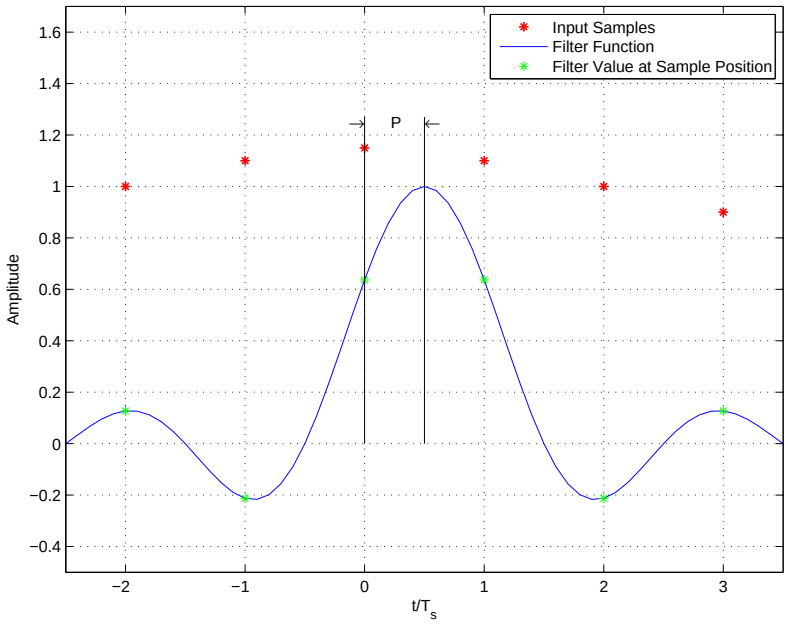


Figure 2.2.11.: Illustration of output sample computation for time P

The shift value $P \in [0, 1)$ defines the position of the desired output sample between two input samples. P needs to be quantized in order set the number of possible positions between two successive input samples. This number of possible values for P defines also the number of filter values that are needed for one zero crossing of the filter function.

If we have nb_P bits to represent P we need to have $N_P = 2^{nb_P}$ samples of the filter function per zero crossing. Due to the fact that for high nb_P values the number of filter taps can get very high, there is the possibility to add linear interpolation of the filter function. Therefore the nb_P bits can be separated into nb_l and nb_η bits so that $nb_P = nb_l + nb_\eta$, where nb_η is the number of bits for the quantization of the linear interpolation.

Now there are $L = 2^{nb_l}$ filter samples per zero crossing pre-calculated. The filter function is a symmetric function, hence only one half of the samples needs to be stored in a filter table.

The number of zero crossings in the filter function is denoted as N_z. These filter values are stored in the filter table $h[l]$. The variable $l \in [0, L \cdot N_z]$ is used to represent the current position in the filter table.

Figure 2.2.12 shows an example content of the filter table $h[l]$ with $nb_l = 7$ and $N_z = 3$. Which means that three zero crossings and $2^7 = 128$ samples per zero crossing are stored in the table.

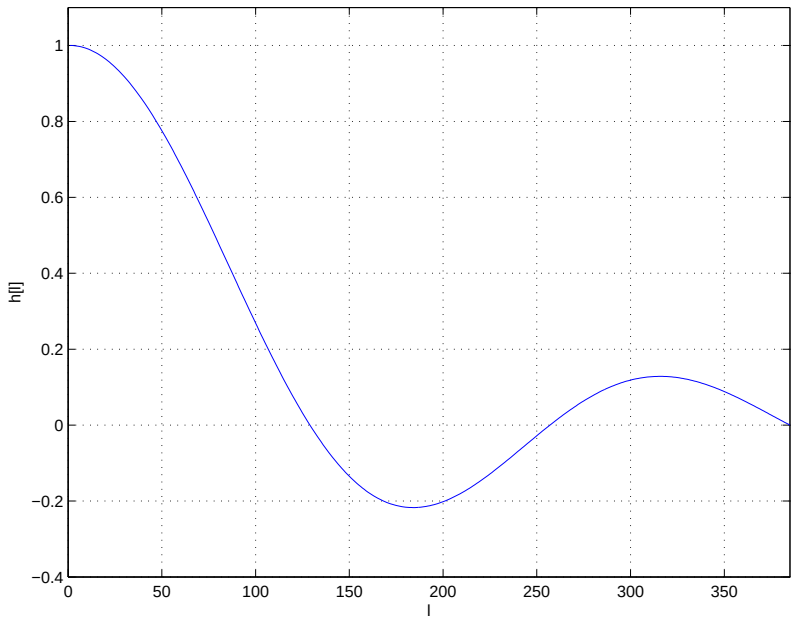


Figure 2.2.12.: Filter table content for $nb_l = 7$ and $N_z = 3$

In addition a second table is generated, which stores the difference values of the filter table. This table $\bar{h}[l] = h[l+1] - h[l]$ with the endpoint $h[N_z \cdot L] = 0$ is used to store the amplitude difference of the filter function between two successive filter samples. Using these values, linear interpolation of the filter function can be implemented. To do so, the interpolation variable $\eta \in \left[0, 2^{nb_\eta} - 1\right]$ is used to represent the position between two successive filter samples. By calculating $h[l] + \eta \cdot \bar{h}[l]$ the linear interpolation of the filter samples can be performed.

2.2.4.1. Time Register

In order to control the input sample shifting, the variable n is introduced. This variable defines the current reference input sample index.

The three variables n, l and η can now be combined into one time register t, which is illustrated in figure 2.2.13. This time register represents the current time position for the actual output sample calculation. It can be interpreted as a binary fixed point number which has n_n integer bits and $n_l + n_\eta$ fractional bits.

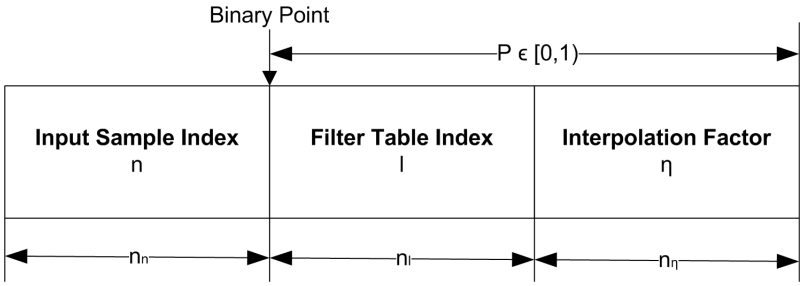


Figure 2.2.13.: Composition of the time register t

2.2.4.2. Procedure for Upsampling

The paper [Smi11] proposes following procedure for a sampling rate conversion $\rho = \frac{F_{out}}{F_{in}}$ for $\rho \geq 1$ (upsampling / interpolation):
The whole calculation is separated into two parts, because only one half of the impulse response is stored in the filter table in order to save memory.
At the beginning the values for n, l and η must be loaded from the time register. This time register is initialized with the time position of the first requested output sample.
The first part of the output is calculated using the left part of the filter impulse response:

$$v = \sum_{i=0}^{h_{end}} x(n-i) \cdot \left[h(l + i \cdot L) + \eta \cdot \bar{h}(l + i \cdot L)\right] \tag{2.2.6}$$

After this calculation, a temporary time register has to be created, so that the values for l and η for the right part of the filter impulse response are updated. This can be done by performing the following operation:

$$P_{tmp} = 1 - P \tag{2.2.7}$$

Before calculating the right part of the filter impulse response, the new values for l and η must be loaded from the temporary time register P_{tmp}. If the temporary time register P_{tmp} is zero, which means the P was also zero before, the variable l has to be incremented by L. Then the right part can now be calculated using the following formula:

$$y(t) = v + \sum_{i=0}^{h_{end}} x(n+1+i) \cdot \left[h(l + i \cdot L) + \eta \cdot \bar{h}(l + i \cdot L)\right] \tag{2.2.8}$$

When the calculation of one output sample is finished, the time register needs to be incremented

$$t = t + \frac{2^{nb_l + nb_\eta}}{\rho} \tag{2.2.9}$$

which performs the shifting of the input signal as well as the calculation of the new time position for the next output sample.

2.2.4.3. Procedure for Downsampling

If a sampling rate conversion for $\rho < 1$ is desired, the procedure changes slightly. These changes are needed since the cut-off frequency of the filter needs to be adapted to avoid aliasing in the output signal.
The initial P has to be replaced by $P' = \rho \cdot P$, which is used to load the values of l and η. The stepsize through the filter table also changes:

$$v = \sum_{i=0}^{h_{end}} x(n-i) \cdot \left[h(l + i \cdot \rho \cdot L) + \eta \cdot \bar{h}(l + i \cdot \rho \cdot L)\right] \tag{2.2.10}$$

In order to calculate the values for l and η for the right part of the filter impulse response, a temporary time register P'_{tmp} is created from which the new values for l and η are loaded:

$$P'_{tmp} = \rho - P' = \rho \cdot (1 - P) \tag{2.2.11}$$

If the temporary time register P'_{tmp} is zero, which means the P' was also zero before, the variable l has to be incremented by $\rho \cdot L$. Now again, the final output value can be calculated, but again with a different step size through the filter table:

$$y(t) = v + \sum_{i=0}^{h_{end}} x(n+1+i) \cdot \left[h(l + i \cdot \rho \cdot L) + \eta \cdot \bar{h}(l + i \cdot \rho \cdot L)\right] \tag{2.2.12}$$

Finally, after calculating one output sample, the time register needs to be incremented by

$$t = t + \frac{2^{nb_l + nb_\eta}}{\rho}$$

which performs the input signal shifting as well as the calculation of the new time positions for the next output sample.

2.2.4.4. Parameter Values for some Example Configurations

For the purpose of illustration, the values for n, l and η are listed in three tables for three example cycles of the algorithm. The examples are shown for a filter impulse response with the following parameters: $N_z = 2$, $nb_l = 7$ and $nb_\eta = 4$.
Table 2.2.1 shows the variables for a simple interpolation using $\rho = 1$ with two different initial values.

step	n	l	η
initial	10	0	0
P_{tmp}	10	128	0
increment	11	0	0

step	n	l	η
initial	10	64	0
P_{tmp}	10	64	0
increment	11	64	0

Table 2.2.1.: Values for the variables n, l and η for $\rho = 1$ and two different initial time registers

Table 2.2.2 shows the variables for an upsampling case using $\rho = 2$ with two different initial values.

step	n	l	η
initial	10	0	0
P_{tmp}	10	128	0
increment	10	64	0
P_{tmp}	10	64	0
increment	11	0	0

step	n	l	η
initial	10	32	0
P_{tmp}	10	96	0
increment	10	96	0
P_{tmp}	10	32	0
increment	11	32	0

Table 2.2.2.: Values for the variables n, l and η for $\rho = 2$ and two different initial time registers

Table 2.2.3 shows the variables for an decimation case using $\rho = \frac{1}{2}$ with two different initial values.

step	n	l	η
initial	10	0	0
P_{tmp}	10	64	0
increment	12	0	0
P_{tmp}	12	64	0
increment	14	0	0

step	n	l	η
initial	10	16	0
P_{tmp}	10	48	0
increment	12	16	0
P_{tmp}	12	48	0
increment	14	16	0

Table 2.2.3.: Values for the variables n, l and η for $\rho = \frac{1}{2}$ and two different initial time registers

2.2.5. Filter Analysis

As already mentioned in section 2.2.3, the ideal lowpass filter cannot be implemented in real systems due to its non-causal impulse response. There are several parameters in the filter design for the FIR interpolation which will be discussed in this section.

First of all, the number of zero crossings is denoted by N_z in the lowpass filter impulse response. The number of taps of the resulting filter is $N_{taps} = 2 \cdot N_z + 1$, which means the larger N_z, the more filter taps are used. The second important parameter is the β value of the Kaiser window that is used to truncate the length of the ideal lowpass filter impulse response.

For illustration of the impact of these two parameters, the magnitude response of the filter is plotted for two interpolator applications.

2.2.5.1. Decimation Application

The first application is the decimation configuration of the interpolator. In this simulation, the sample rate conversion factor is set to $\rho = \frac{1}{2}$, which equals a decimation by 2. Figures 2.2.14 to 2.2.16 show the magnitude response of the filter for different numbers of zero crossings N_z and different β values. Using $\beta = 0$, the Kaiser window equals a rectangular window.

If we compare these three figures (2.2.14 - 2.2.16) of the decimation application, we can see that the number of zero crossings N_z has a high impact on the width of the main lobe and the steepness to the stopband. Also the sidelobe attenuation is increased when more filter samples are used. This means that for higher values of N_z, the attenuation of unwanted spectral components above the new Nyquist frequency $\frac{f_s}{4}$ gets increased, which is highly desireable.

If we look at the influence of the β parameter of the Kaiser window, we can see that a higher value for β increases the width of the main lobe, but also increases the attenuation of frequency components above cut-off frequency. As a result, the passband ripples can be decreased when using a higher β value with the drawback of a higher passband attenuation.

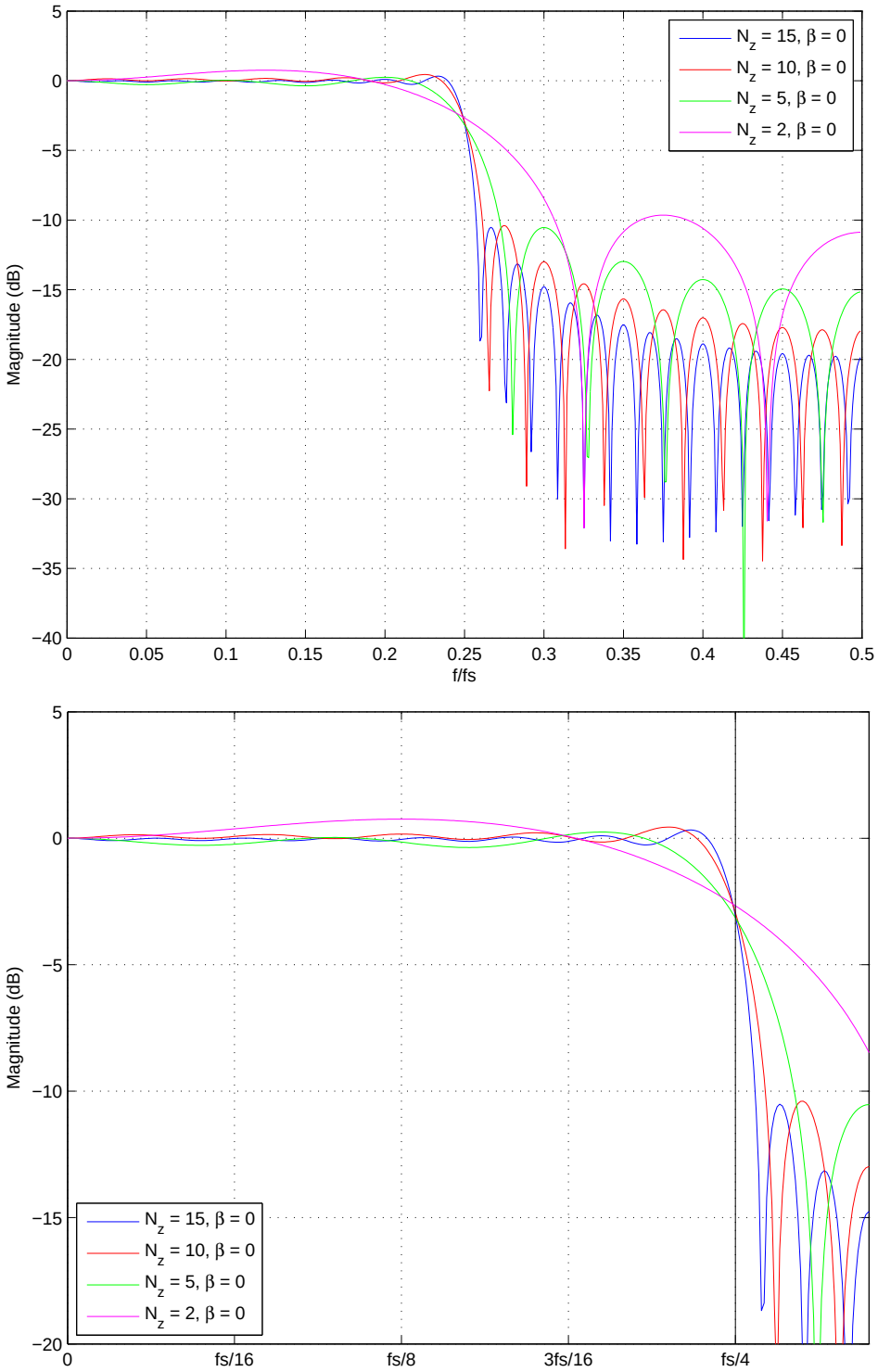


Figure 2.2.14.: Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 0$ (top plot). The bottom plot shows a zoom into the passband of the magnitude response.

For the timing recovery application in this thesis, the filter parameters are set to $N_z = 2$ and $\beta = 4$ which is shown in figure 2.2.15. In this figure the cut-off frequency of the root-raised-cosine-filter at $\frac{f_s}{8}$ is also indicated. Here we can see that the attenuation of the signal bandwidth is lower than $0.1\,dB$. Only the excess bandwidth of the root-raised-cosine filter is attenuated slightly. This trade-off had to be made since the FPGA resources are very limited

and therefore the number of filter taps is needed to be kept as small as possible.

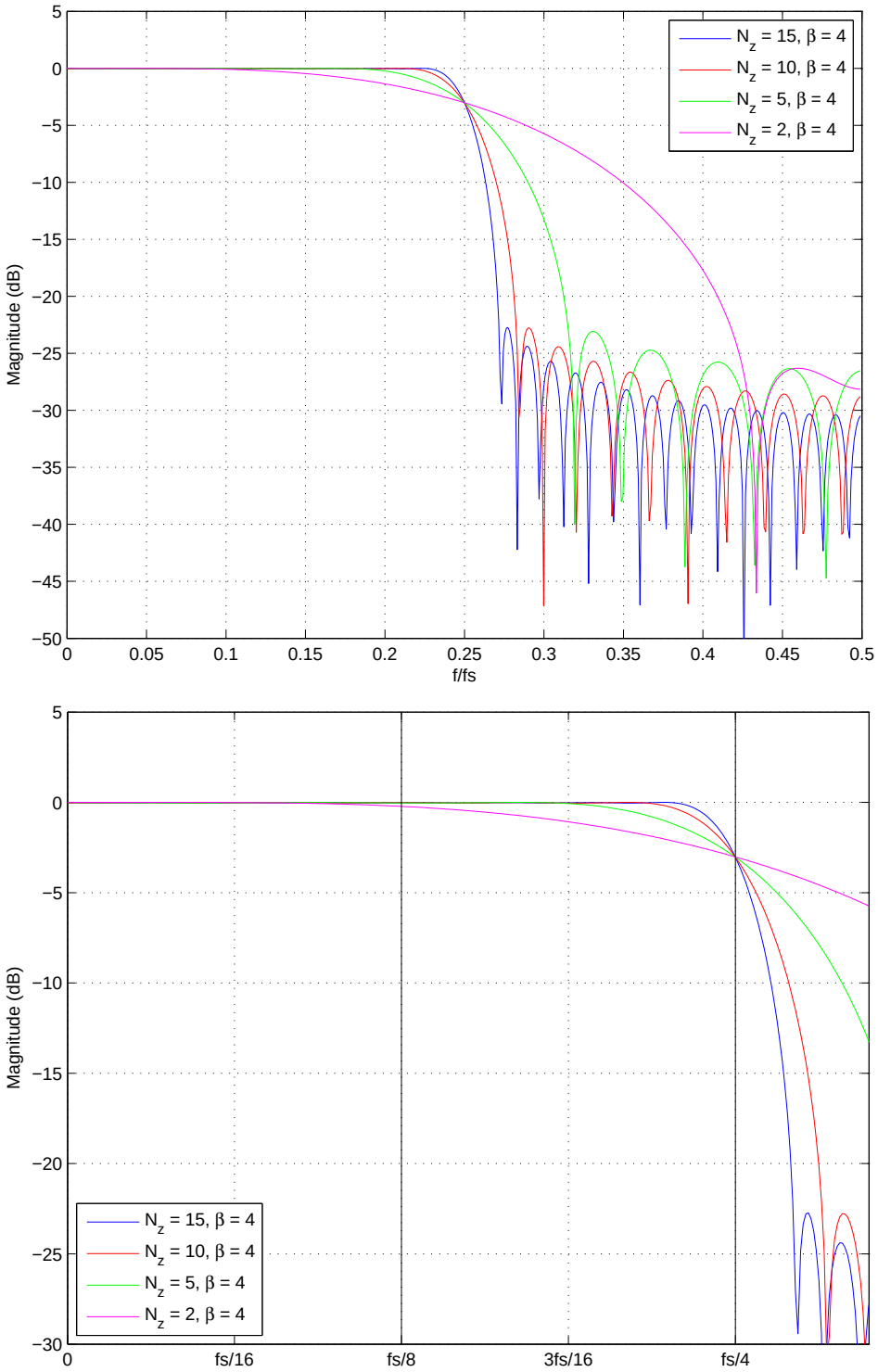


Figure 2.2.15.: Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 4$ (top plot). The bottom plot shows a zoom into the passband of the magnitude response.

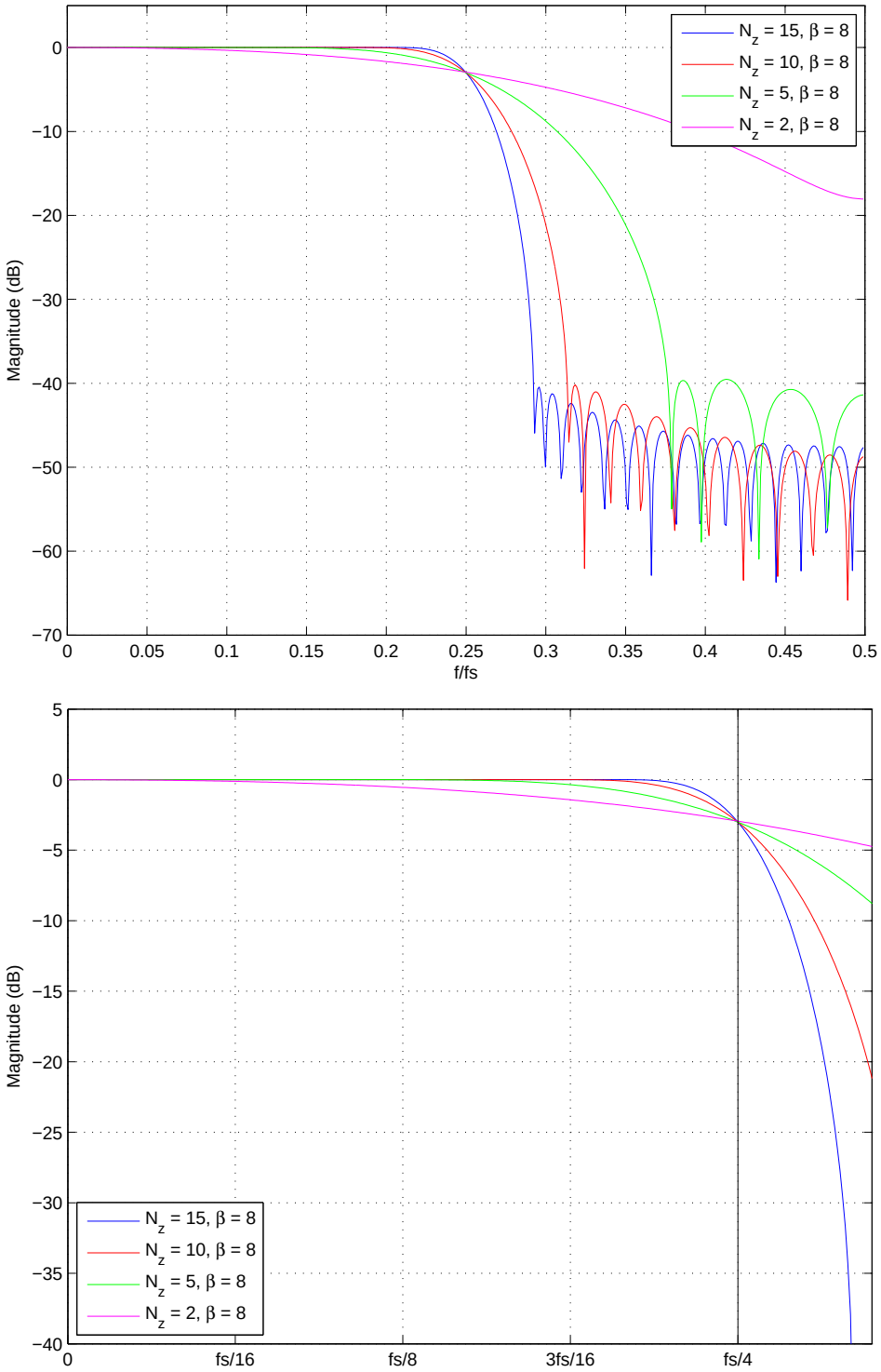


Figure 2.2.16.: Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 8$ (top plot). The bottom plot shows a zoom into the passband of the magnitude response.

2.2.5.2. Satellite Emulator Application

A second application of the interpolator is the satellite emulation. Here we want to introduce short delays by performing a sample rate with conversion factor $\rho \approx 1$. Therefore, the next filter analysis shows the resulting filter of a conversion factor $\rho = 0.99$. Figures 2.2.17 to

2.2.19 show the magnitude response of the resulting interpolation filter for different numbers of zero crossings N_z and different values of the Kaiser window parameter β.

It can be observed that more zero crossings N_z lead to a steeper decay of the magnitude response and therefore also to a smaller passband attenuation. Unfortunately, this higher number of filter taps also leads to passband ripples. These passband ripples can be decreased by a higher β value. But the drawback of this higher β value is the resulting wider main lobe, which also induces a higher passband attenuation.

Since the goal in this application is to maximize the usable bandwidth of the input spectrum and a minor passband ripple in a certain range is acceptable, e.g. up to $\pm 0.05\,dB$, it is recommended to use a higher number of zero crossings N_z and a lower value for β. For example in figure 2.2.18 where $\beta = 4$ and $N_z = 15$, we have only passband ripples smaller than $0.01\,dB$ and a nearly constant passband gain up to $f = 0.45 \cdot f_s$. For example, if the sampling rate is $f_s = 100\,MHz$ like it is in the GNU Radio platform, the usable bandwidth would be up to $f = 0.45 \cdot f_s = 45\,MHz$. At higher frequencies, the attenuation does not exceed $0.9\,dB$.

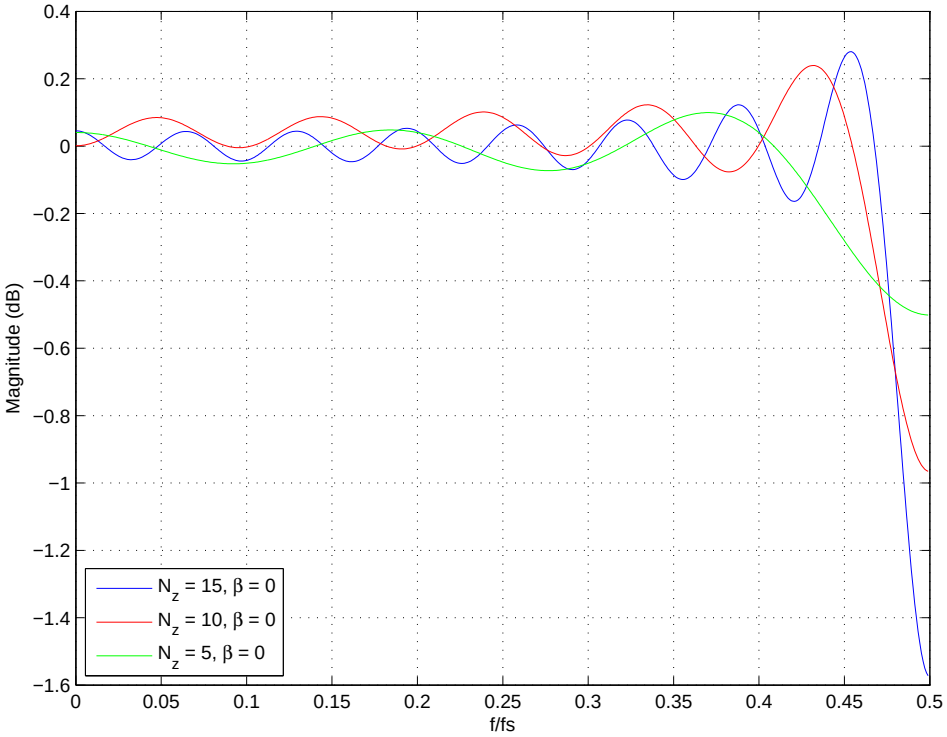


Figure 2.2.17.: Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 0$

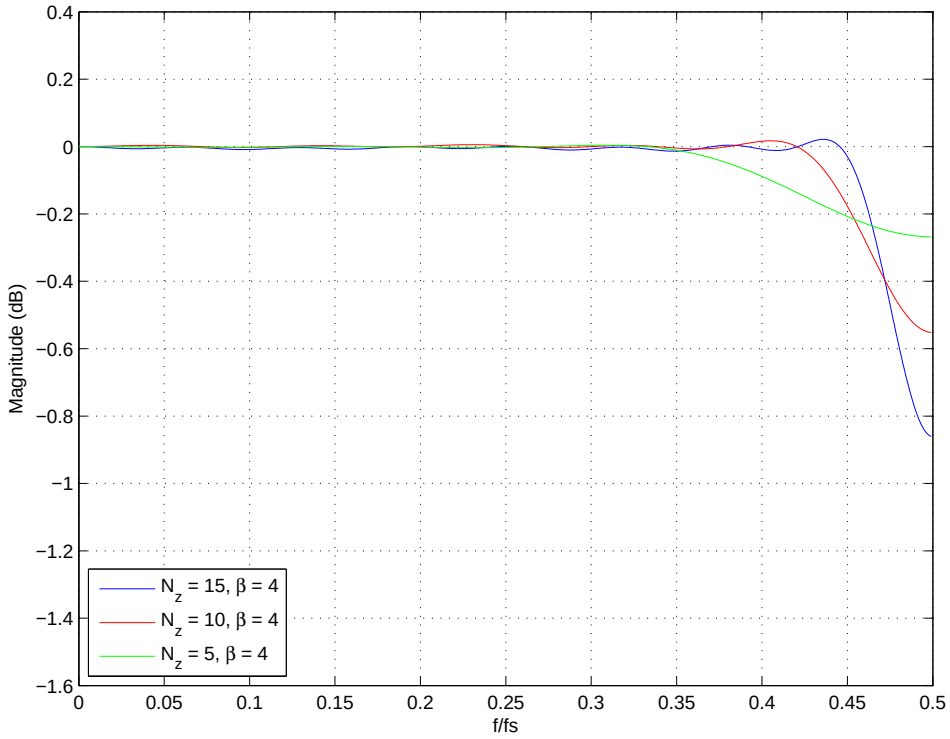


Figure 2.2.18.: Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 4$

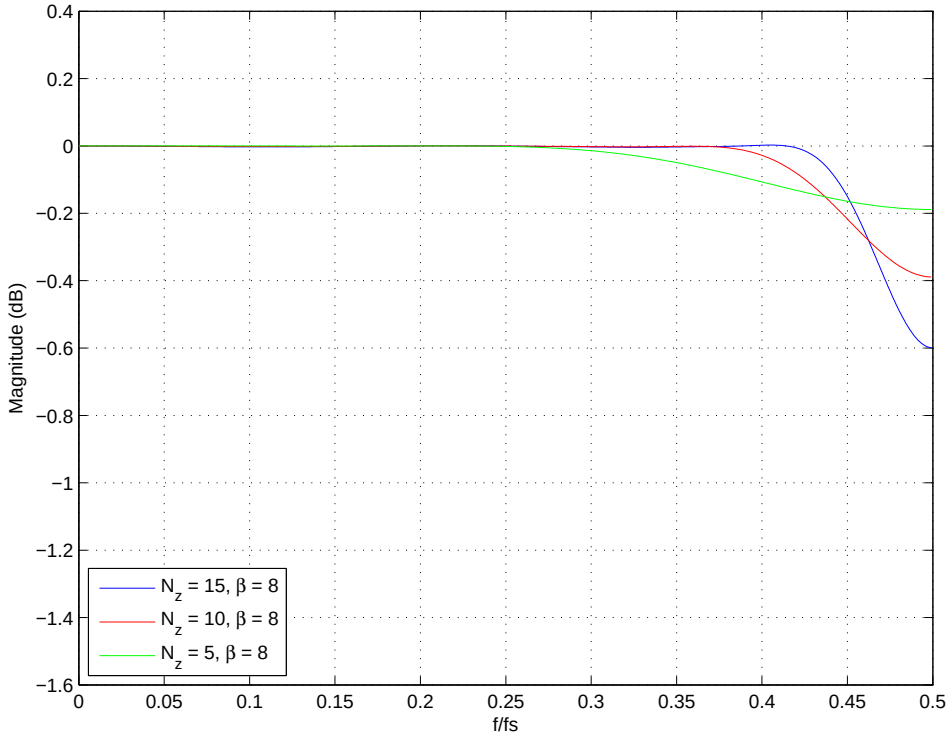


Figure 2.2.19.: Magnitude response of the interpolation filter for different numbers of zero crossings N_z and $\beta = 8$

3. Timing Estimation

This section discusses the timing estimation procedure which is needed for the task of timing recovery.
In most digital receivers the analog input signal is sampled at a fixed rate. These resulting samples are used for all further processing. Unfortunately, the receiver's sampling clock may not be in phase with the clock of the transmitter, which leads to a sample timing error and therefore to ISI. To cope with this error, this timing offset needs to be estimated. Using this estimated timing offset, an interpolator can be used to produce the samples at the desired sampling time position to minimize the ISI.

3.1. Oerder and Meyr Algorithm

Since no data of the received signal is known, a non-data-aided algorithm for the timing estimation must be chosen. Due to its simplicity and therefore common usage, the feed-forward Oerder and Meyr algorithm presented in [OM88] is used for timing estimation in this thesis.
The mentioned paper proposes a method to estimate the timing offset for linear modulation schemes like QAM, PAM or PSK. The model of the received signal of digital data transmission is

$$r(t) = \sum_{n=-\infty}^{\infty} a_n \cdot g_T\left(t - n \cdot T - \varepsilon(t) \cdot T\right) + n(t) = u(t) + n(t) \tag{3.1.1}$$

where a_n are the complex valued symbols, g_T is the transmission signal pulse, T is the symbol duration, $n(t)$ is additive white Gaussian noise and $\varepsilon(t)$ is the unknown time delay which needs to be estimated. $\varepsilon(t)$ is slowly varying and therefore the input signal can be separated into sections in which ε is assumed to be constant.
The received signal is filtered with the matched filter at the receiver, that is

$$\tilde{r}\left(t\right) = r\left(t\right) * g_R\left(t\right) \tag{3.1.2}$$

where $g_R\left(t\right)$ is the impulse response of the receiving filter, and then sampled at a fixed rate $f_s = \frac{1}{T_s} = \frac{N}{T}$ to produce the samples

$$z\left(l \cdot T_s\right) = \tilde{r}\left(l \cdot \frac{T}{N}\right) \tag{3.1.3}$$

When inserting equation 3.1.1 and 3.1.2 into equation 3.1.3, we obtain

$$z\left(l \cdot T_s\right) = \sum_{n=-\infty}^{\infty} a_n \cdot g\left(\frac{l \cdot T}{N} - n \cdot T - \varepsilon \cdot T\right) + \tilde{n}\left(\frac{l \cdot T}{N}\right)$$

where $g\left(t\right) = g_T\left(t\right) * g_R\left(t\right)$ and $\tilde{n}\left(\frac{l \cdot T}{N}\right)$ is the filtered and sampled noise.
The objective function for the timing estimation as now defined as [MMF98]

$$L(\varepsilon) = \sum_{l=0}^{L-1} |z(l \cdot T + \varepsilon \cdot T)|^2 \tag{3.1.4}$$

which is to be maximized. The detailed derivation is given in [MMF98]. This objective function is a time average over the interval $[0, L-1]$ of the squared samples at the output of the matched filter.
The squared samples at the output of the matched filter $|z(l \cdot T + \varepsilon \cdot T)|^2$ have the following Fourier series representation:

$$|z(l \cdot T + \varepsilon \cdot T)|^2 = \sum_{n=-\infty}^{\infty} c_n^{(l)} \cdot e^{j \cdot \frac{2 \cdot \pi}{T} \cdot n \cdot T \cdot \varepsilon} = \sum_{n=-\infty}^{\infty} c_n^{(l)} \cdot e^{j \cdot 2 \cdot \pi \cdot n \cdot \varepsilon} \tag{3.1.5}$$

where the Fourier coefficients $c_n^{(l)}$ are random variables.
When inserting equation 3.1.5 into equation 3.1.4 we get

$$\sum_{l=0}^{L-1} |z(l \cdot T + \varepsilon \cdot T)|^2 = \sum_{l=0}^{L-1} \sum_{n=-\infty}^{\infty} c_n^{(l)} \cdot e^{j \cdot 2 \cdot \pi \cdot n \cdot \varepsilon} = \sum_{n=-\infty}^{\infty} c_n \cdot e^{j \cdot 2 \cdot \pi \cdot n \cdot \varepsilon} \tag{3.1.6}$$

where

$$c_n = \sum_{l=0}^{L-1} c_n^{(l)} \tag{3.1.7}$$

Due to the fact that the received and squared signal $|z(l \cdot T + \varepsilon \cdot T)|^2$ in equation 3.1.5 is a real valued signal such that $c_n = c_{-n}^*$. Hence, equation 3.1.6 can be rewritten as

$$\sum_{l=0}^{L-1} |z(l \cdot T + \varepsilon \cdot T)|^2 = c_0 + \sum_{|n| \geq 1} 2 \cdot \Re\left\{c_n \cdot e^{j \cdot 2 \cdot \pi \cdot n \cdot \varepsilon}\right\}$$

In [MMF98] it is proved that only the three coefficients c_{-1}, c_0 and c_1 have a nonzero mean, so that all other coefficients can be omitted for the estimation of the timing parameter ε.
Now we want to maximize our objective function which can be done by applying

$$\hat{\varepsilon} = \arg\max_{\varepsilon} \left(c_0 + 2 \cdot \Re\left\{c_1 \cdot e^{j \cdot 2 \cdot \pi \cdot \varepsilon}\right\}\right)$$

But since c_0 and the absolute value of c_1 are independent of ε which is shown in [MMF98], the maximum can also be established by calculating

$$\hat{\varepsilon} = -\frac{1}{2 \cdot \pi} \cdot \arg(c_1) \tag{3.1.8}$$

So the aim of the algorithm is to calculate the complex Fourier coefficient c_1 and determine the angle of c_1. Equation 3.1.7 tells us that c_1 is the average of all Fourier coefficient $c_1^{(l)}$ in the interval $[0, L-1]$.
In order to calculate the Fourier coefficient $c_1^{(l)}$the sampling rate must be chosen such that the spectral component can still be represented by the DFT. When looking at the bandwidth of the signal $z(t)$ at the matched filter output, we can see that it is bandlimited to $B_z = \frac{1+\alpha}{2 \cdot T}$. Since the sampling theorem must also be fulfilled for the signal $|z(t)|^2$ which doubles the bandwidth of the signal, the sampling frequency must be chosen by

$$B_{|z|^2} = \frac{1+\alpha}{T} < \frac{1}{2 \cdot T_s}$$

where $T_s = \frac{T}{N}$ is the sampling period.
When setting $N = 4$, we can fulfill the requirement of representing the spectral component c_1 and we obtain a quite simple implementation of the Fourier coefficient calculation for the implementation.
In paper [OM88] the algorithm is summarized using the nomenclature $x_k = z(k \cdot T_s)$ and $X = c_1$.
The spectral component $X^{(l)}$ can be calculated as

$$X^{(l)} = \sum_{k=0}^{N-1} x_k^{(l)} \cdot e^{-j \cdot 2 \cdot \pi \cdot \frac{k}{N}}$$

Now the spectral component X can be determined for every section m of length $L \cdot T_s$, which corresponds to $L \cdot N$ samples, using the following summation:

$$X_m = \sum_{k=m \cdot L \cdot N}^{(m+1) \cdot L \cdot N - 1} x_k \cdot e^{-j \cdot 2 \cdot \pi \cdot \frac{k}{N}} \tag{3.1.9}$$

where for $N = 4$ the exponential function becomes $e^{-j \cdot 2 \cdot \pi \cdot \frac{k}{4}} = (-j)^k$ which leads to a implementation without the need of multiplication.

3.1.1. Lower Bounds for the Timing Estimator

In [MdJ92] the Modified Cramér-Rao Lower Bound (MCRLB) is given by the formula

$$MCRLB = \frac{1}{-2 \cdot L \cdot T^2 \cdot \ddot{g}(0) \cdot \frac{E_s}{N_0}} \tag{3.1.10}$$

with

$$-T^2 \cdot \ddot{g}(0) = \frac{1}{3} \cdot \pi^2 \cdot \left(1 + 3 \cdot \alpha^2\right) - 8 \cdot \alpha^2$$

where L is the window size, E_s is the symbol energy, N_0 is the noise power spectral density and α is the roll-off factor of the root-raised-cosine filter.
Especially for the Oerder and Meyr algorithm there is another lower bound (MOLB) which is given in [MdJ92] for non-data-aided maximum likelihood estimators:

$$MOLB = \frac{1}{\alpha \cdot L \cdot \pi^2 \cdot \frac{E_s}{N_0}}$$

3.2. CORDIC Algorithm for Angle Calculation

The CORDIC (**co**ordinate **r**otation **di**gital **c**omputer) algorithm is a common way to compute efficiently trigonometric functions such as [Xil11]:

- Vector Rotation
- Vector Translation
- Sine and Cosine
- Square Root
- etc.

The following part describes the functionality of the algorithm for vector rotation and angle calculation where the basic formulas can be looked up in [Xil11]. Initially the CORDIC algorithm was designed to perform vector rotations where the initial vector $v = x + j \cdot y$ is rotated through the angle Θ to the desired vector v' . This vector rotation can be written as

$$\begin{aligned} v' &= (x + j \cdot y) \cdot e^{j \cdot \Theta} \\ &= (x + j \cdot y) \cdot (\cos(\Theta) + j \cdot \sin(\Theta)) \\ &= (x \cdot \cos(\Theta) - y \cdot \sin(\Theta)) + j \cdot (y \cdot \cos(\Theta) + x \cdot \sin(\Theta)) \end{aligned} \tag{3.2.1}$$

where $x = \Re(v)$ is the real part and $y = \Im(v)$ is the imaginary part of the initial vector v. The new vector v' also consists of a real part $x' = \Re(v')$ and and imaginary part $y' = \Im(v')$. According to equation 3.2.1, the real and the imaginary part can also be written as

$$\begin{aligned} x' &= x \cdot \cos(\Theta) - y \cdot \sin(\Theta) \\ y' &= y \cdot \cos(\Theta) + x \cdot \sin(\Theta) \end{aligned} \tag{3.2.2}$$

Using the relation

$$\tan(\varphi) = \frac{\sin(\varphi)}{\cos(\varphi)} \rightarrow \sin(\varphi) = \cos(\varphi) \cdot \tan(\varphi)$$

we can rewrite the real and imaginary part of v' in equation 3.2.2 as

$$\begin{aligned} x' &= \cos(\Theta) \cdot (x - y \cdot \tan(\Theta)) \\ y' &= \cos(\Theta) \cdot (y + x \cdot \tan(\Theta)) \end{aligned} \tag{3.2.3}$$

The algorithm now restricts the values of $\tan(\Theta) = \pm 2^{-k}$ and uses the identifier $\tan(\varphi) \equiv -\tan(\varphi)$ and $\cos(-\varphi) \equiv \cos(\varphi)$ to create a micro-rotation which defines one iteration:

$$\begin{aligned} x_{k+1} &= K_k \cdot \left(x_k - y_k \cdot d_k \cdot 2^{-k}\right) \\ y_{k+1} &= K_k \cdot \left(y_k + x_k \cdot d_k \cdot 2^{-k}\right) \\ z_{k+1} &= z_k - d_k \cdot \arctan\left(2^{-k}\right) \end{aligned} \tag{3.2.4}$$

with the loop condition

$$d_k = \begin{cases} -1 & z_k < 0 \\ +1 & z_k \geq 0 \end{cases} \tag{3.2.5}$$

where $K_k = \cos\left(\arctan\left(2^{-k}\right)\right)$ is the scale factor, $d_k = \pm 1$ defines the direction of the rotation, $k = 0 \ldots n-1$ is the iteration index and z_0 is the rotation angle.

When looking at the micro-rotation in equation 3.2.4 combined with the loop condition in equation 3.2.5, we can see that the initialized angle z_0 gets minimized during the iterations since the rotation consists of a sequence of smaller micro-rotations. In every iteration of the CORDIC algorithm, the output gains one bit of resolution, hence the number of iterations n controls the accuracy of the vector rotation.
The values for $\arctan\left(2^{-k}\right)$ can be stored in a small look-up table and as it can be seen in equation 3.2.4, each iteration only consists of simple binary shifts, additions and subtractions.
When using the CORDIC algorithm for angle calculation, the loop condition has to be altered to

$$d_k = \begin{cases} +1 & y_k < 0 \\ -1 & y_k \geq 0 \end{cases} \tag{3.2.6}$$

where $z_0 = 0$. In this configuration, the algorithm minimizes the value of the imaginary part. In every iteration the vector gets rotated by a smaller angle and the number of iterations again defines the accuracy of the calculation. The final value of z is the calculated angle Θ.
Unfortunately, these two methods only work in the range of $\pm\frac{\pi}{2}$. To achieve full operational range of $\pm\pi$, a pre-rotation has to be performed according to the angle's sector. This pre-rotation can be performed by manipulating the initial values of the algorithm:

$$x_0 = \begin{cases} -x & x < 0 \\ +x & x \geq 0 \end{cases}$$

$$y_0 = \begin{cases} -y & x < 0 \\ +y & x \geq 0 \end{cases}$$

$$z_0 = \begin{cases} -180 & x < 0 \\ 0 & x \geq 0 \end{cases}$$

3.3. Implementation of the Timing Estimator

For the implementation of the timing estimation algorithm, the basic block diagram is visualized in figure 3.3.1.

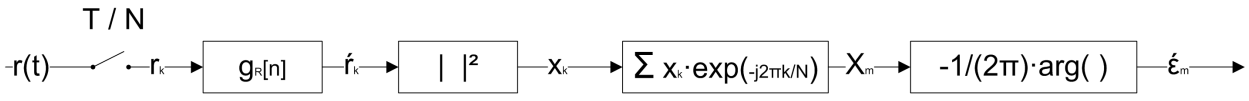


Figure 3.3.1.: Block diagram of the Oerder and Meyr algorithm

First of all the received signal is sampled at the rate $\frac{T}{N}$. Then these complex valued samples are filtered by the receive matched filter $g_R[n]$. In this thesis, this is a root-raised-cosine filter which is used for matched filtering in order to minimize ISI (see chapter 4).
In order to calculate x_k, the filtered complex valued samples get squared according to

$$x_k = \Re\left\{\tilde{r}_k\right\}^2 + \Im\left\{\tilde{r}_k\right\}^2 \tag{3.3.1}$$

To realize the calculation of the Fourier coefficient in equation 3.1.9, a section of $L \cdot N$ samples, where $N = 4$ (see section 3), is extracted. A closer look at the summation shows that for the extracted section k runs from 0 to $L \cdot N - 1$. Observing the exponential term in the summation, we can see that the argument $-j \cdot 2 \cdot \pi \cdot \frac{k}{N}$ only results in four possible values due to the $2 \cdot \pi$ periodicity of the unit circle. These possible values are presented in table 3.3.1.

k	$e^{-j \cdot 2 \cdot \pi \cdot \frac{k}{N}}$
0	1
1	$-j$
2	-1
3	j

Table 3.3.1.: Phasor values for Fourier coefficient calculation using $N = 4$

This means that instead of complex calculation of the Fourier coefficients, the squared samples only need to be multiplied by ± 1 and $\pm j$, according to their position in the section. The angle of the resulting complex valued Fourier coefficient X_m needs to be normalized by applying equation 3.1.8 in order to get the estimate $\hat{\varepsilon}$ for the current section.
This means that $\hat{\varepsilon}$ is an estimate that can take values in the interval $\hat{\varepsilon} \in [-0.5, 0.5]$. As a result and due to the oversampling rate of $N = 4$, the estimated sampling time lies in between ± 2 samples. Figure 3.3.2 illustrates this result.

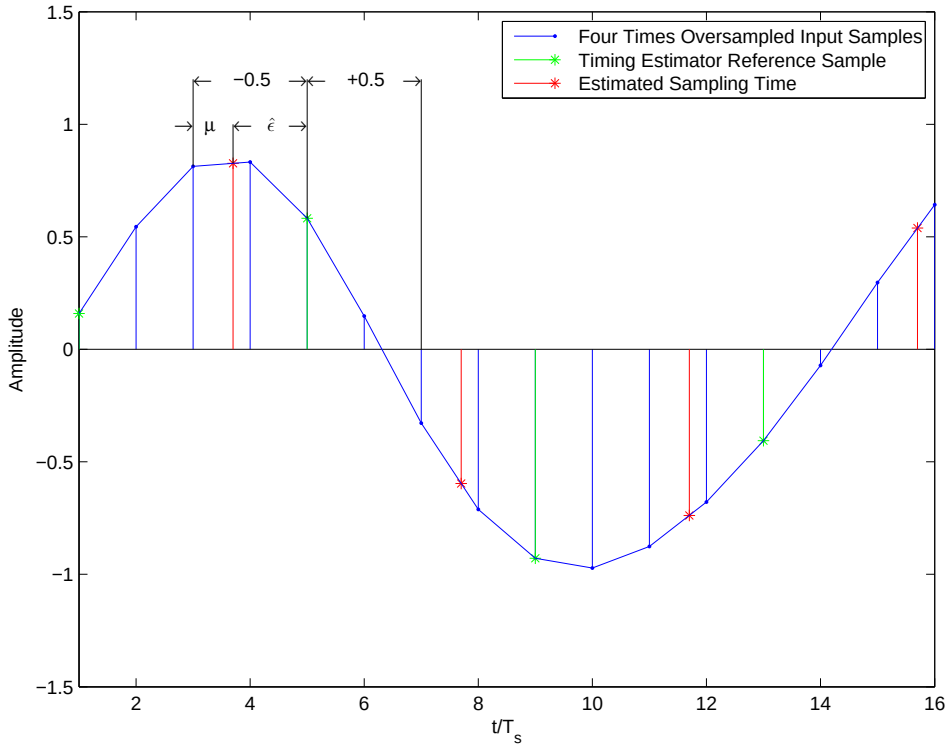


Figure 3.3.2.: Illustration of the timing estimator result

3.4. Combination of Timing Estimation and Interpolation for Timing Recovery

Timing recovery in a demodulator can be achieved in three different modes according to [Gar90]. The first one is the analog recovery, where the symbol timing is adjusted by analog circuits and a controlled sampling clock. This method achieves an output whose samples are synchronized to the symbol timing. The corresponding block diagram can be seen in figure 3.4.1.

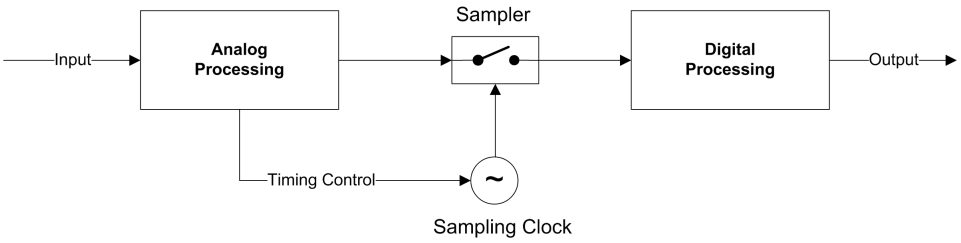


Figure 3.4.1.: Analog timing recovery

A second method is the hybrid recovery where the symbol timing is recovered from the digital samples and a controlled sampling clock. Just like the first method, the hybrid recovery produces samples that are synchronized to the symbol timing. Figure 3.4.2 shows the corresponding block diagram.
This first two methods are based on the adjustment of the sampling clock which is simpler to implement according to [Gar90], but depending on the used architecture not always possible.

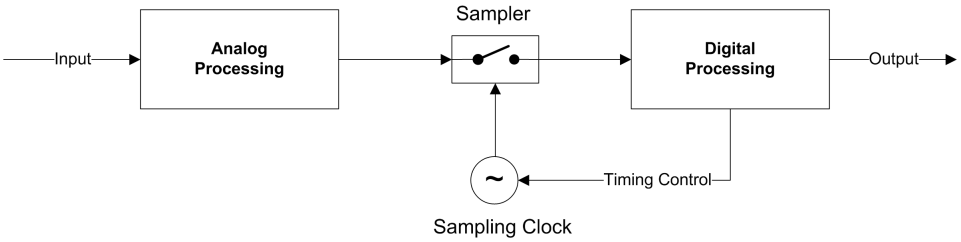


Figure 3.4.2.: Hybrid timing recovery

For digital architectures the third method is suggested. In this method a fixed clock controls the sampling of the input signal where no adjustment can be implemented. So the whole process of timing adjustment is performed in the digital domain after the sampling procedure. A basic block diagram of this digital recovery method can be seen in figure 3.4.3.

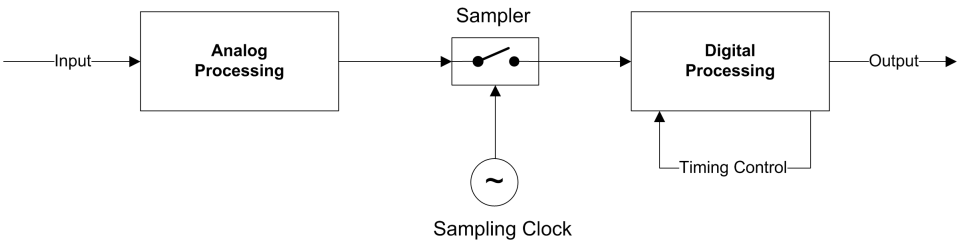


Figure 3.4.3.: Digital timing recovery

Since in the given architecture for this thesis the input samples are provided with a fixed sampling clock, the digital recovery method is applied. In this case the timing recovery is implemented using a timing estimator which produces a timing control signal and an interpolator which adjusts the delay to perform synchronization with the symbol timing. Figure 3.4.4 shows the basic block diagram of the timing recovery described in this section. The timing estimator obtains the symbol timing out of the samples from a sampler with a fixed sampling frequency. Then the timing estimator produces a timing control signal which is the input of the interpolator. The interpolator then performs the delay to synchronize to the symbol timing and also does the proper symbol extraction out of the interpolated samples. The delay block in front of the interpolator is needed since the timing estimator needs some time to produce the output and therefore the interpolator needs to be synchronized to the timing estimator.

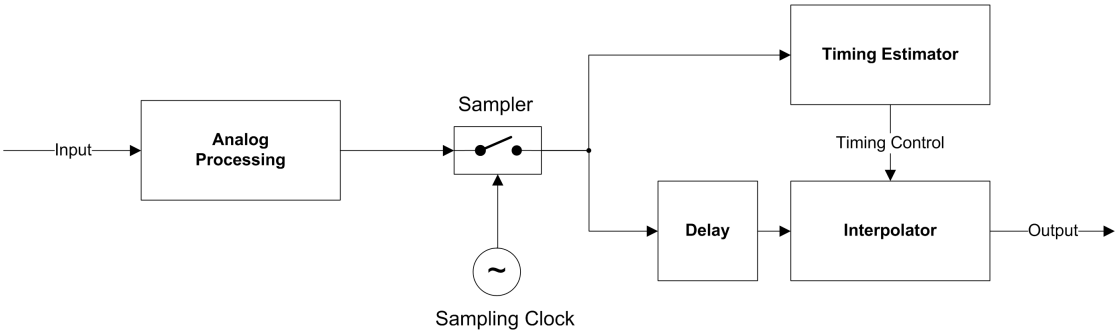


Figure 3.4.4.: Block diagram of the implemented timing recovery

The algorithm presented in the section 3.3 uses a $N = 4$ times oversampled input signal to estimate the sampling time. This means that $f_s = N \cdot f_{Symbol} = 4 \cdot f_{Symbol}$. Hence the result of this timing estimator can then be used in an interpolator which then does the proper interpolation and symbol extraction.
Since the estimator result lies in a range of ± 2 samples, this result has to be separated first into an integer part μ_{int} and a fractional part μ_{frac} before it can be used by the interpolator. The integer part defines the shift of samples from the timing estimator reference point to the nearest sample in the past (on the left side in figure 3.3.2) of the estimated sampling time. The fractional part then defines the residual μ to the estimated sampling time.
For the illustrated example in figure 3.3.2 the values are: $\mu_{int} = -2$ and $\mu_{frac} = \mu$.
This separation can be calculated using the following formulas:

$$\mu_{int} = \begin{cases} -2 & -0.5 \leq \hat{\varepsilon} < -0.25 \\ -1 & -0.25 \leq \hat{\varepsilon} < 0 \\ 0 & 0 \leq \hat{\varepsilon} < 0.25 \\ 1 & 0.25 \leq \hat{\varepsilon} < 0.5 \\ 2 & \hat{\varepsilon} = 0.5 \end{cases} \tag{3.4.1}$$

$$\mu_{frac} = \begin{cases} \hat{\varepsilon} + 0.5 & -0.5 \leq \hat{\varepsilon} < -0.25 \\ \hat{\varepsilon} + 0.25 & -0.25 \leq \hat{\varepsilon} < 0 \\ \hat{\varepsilon} & 0 \leq \hat{\varepsilon} < 0.25 \\ \hat{\varepsilon} - 0.25 & 0.25 \leq \hat{\varepsilon} < 0.5 \\ \hat{\varepsilon} - 0.5 & \hat{\varepsilon} = 0.5 \end{cases} \tag{3.4.2}$$

With these two values, the interpolator is able to calculate the symbol value at this specific time position. This combination can also be seen in the GNU Radio flow graph illustrated in figure 3.4.5. This figure shows that the integer and the fractional output of the timing estimator are connected to the μ input ports of the interpolator. The GNU Radio environment ensures that the input samples of the interpolator are delayed by the time the estimator needs to provide one estimation value.
The interpolator now shifts the input samples such that the center of the filter impulse response is at the exact position of the estimated timing value. Therefore, the input samples are shifted by μ_{int} samples and the impulse response is applied according to the μ_{frac} value.

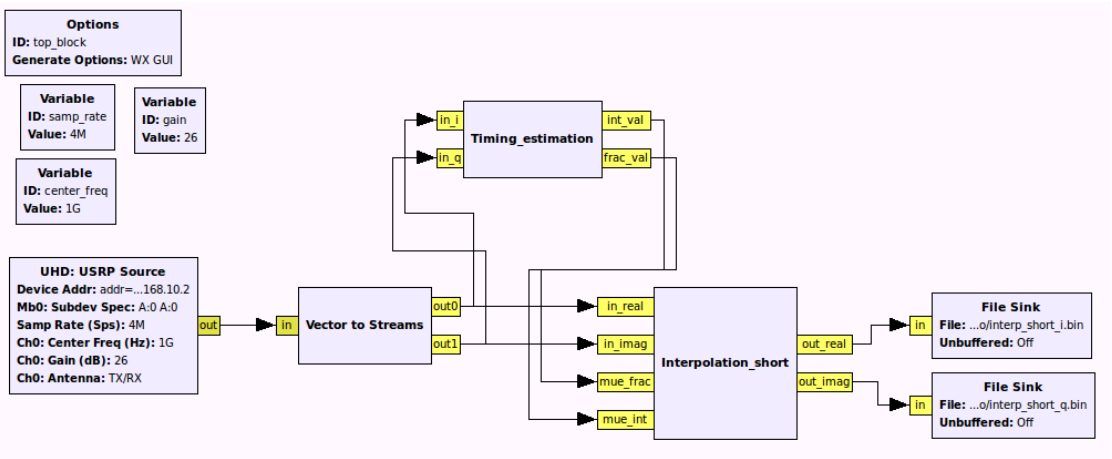


Figure 3.4.5.: GNU Radio flow graph of the combination of interpolator and estimator for timing recovery

3.4.1. Update of μ

The timing estimator provides an estimation of the symbol timing after every $L \cdot N$ samples. Differences in the fractional part can be implemented quite easily by simply changing the μ_{frac} value and therefore changing the initial filter table index l.
When the integer value μ_{int} changes, the time register t has to be incremented by

$$\frac{2^{nb_l+nb_\eta}}{\rho} + 2^{nb_l+nb_\eta} \cdot (\mu_{int_{new}} - \mu_{int_{old}})$$

instead of

$$\frac{2^{nb_l+nb_\eta}}{\rho}$$

3.4.2. Problems with the Update of μ

Due to estimation variance and sampling frequency mismatches, the estimator value can change significantly between two successive estimation results.
If the values of $\mu_{int_{new}}$ and $\mu_{int_{old}}$ differ too much, e.g. if the estimated value changes from $\hat{\varepsilon}_{old} = -1.9$ to $\hat{\varepsilon}_{old} = +1.9$ which is shown in figure 3.4.6, the possibility of missing one symbol is given, which is then called a cycle slip. This results from a moving stable operation point of the estimator. An example is illustrated in figure 3.4.6 where the missed symbol is indicated in a four times oversampled signal. Also the possibility of duplicating one output symbol is given, e.g. from $\hat{\varepsilon}_{old} = +1.9$ to $\hat{\varepsilon}_{old} = -1.9$.

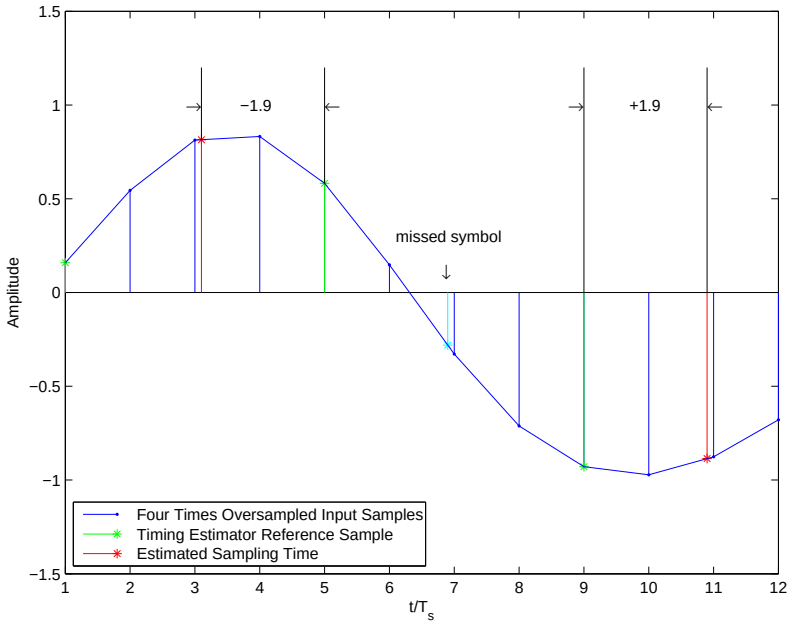


Figure 3.4.6.: Timing update from $\hat{\varepsilon}_{old} = -1.9$ to $\hat{\varepsilon}_{old} = +1.9$

These missed or duplicated symbols need to be detected in order to calculate the right output symbol, since most following processes depend on a continuous symbol stream where no symbol must be duplicated or missing.
To cope with this high estimator variations, the estimator output needs to be corrected accordingly. In this implementation this is done by comparing the old and the new values of μ_{int}. Since in this application the input signal is four times oversampled, the timing estimator reference points are spaced by four samples. So if no update needs to be done, the input signal is simply shifted by four samples.
But if a new μ_{int} arrives, the shift can be calculated by:

$$N_{shift} = 4 + \mu_{int_{new}} - \mu_{int_{old}}$$

If this new shift is too small or too big, the probability of missing or duplicating one symbol (cycle slip) is given. Therefore the shift value N_{shift} needs to be kept in a certain range. In this implementation this range is chosen by:

$$N_{shift_{corrected}} = \begin{cases} N_{shift} - 4 & N_{shift} > 6 \\ N_{shift} + 4 & N_{shift} < 2 \\ N_{shift} & else \end{cases}$$

Using this procedure, some of these cycle slips can be avoided. But if the variance of the timing estimator is too high, there are still some cycle slips occurring due to ambiguity of the estimator result which cannot be prevented by this procedure.
An extended variation of this procedure is given by the unwrapping algorithm presented in [MD97], which should lead to a better performance. More information on the source and effects of cycle slips can be found in [MMF98].
The integration of this extended unwrapping algorithm into this implementation would be a major task for future work.

4. Nyquist Pulse Shaping

For the timing recovery application on the software defined radio, matched filtering is necessary to optimize the signal-to-noise ratio. Due to the fact that for the application a DVB-S2 (see [Tür12] and [ETS05]) signal is used which implies root-raised-cosine filtering, the fundamentals of Nyquist pulse shaping and the root-raised-cosine pulse are summarized in this chapter.
Since bandwidth is very valuable in digital communications and especially in satellite communications, the transmitted signal needs to be filtered in order to limit the occupied bandwidth. Due to filtering, successive symbols can overlap and interfere with other symbols, which is called ISI (intersymbol interference) as mentioned in [Skl01]. This means that at the decision point the signal is not only affected by the current symbol, but also by adjacent symbols which will lead to a degradation of the error performance. In [Skl01] it is shown, that by using an ideal Nyquist filter, which has a rectangular transfer function with the single-sided bandwidth of $W = \frac{1}{2 \cdot T}$ (minimum Nyquist bandwidth), no ISI occurs at the receiver. This is exemplified in figure 4.0.1. At every decision point $n \cdot T$ the received signal is only given by the current pulse and by no other adjacent pulses if the timing is perfect. This is because the rectangular transfer function leads to a sinc pulse in the time domain which is zero at every non-zero integer multiple of T.

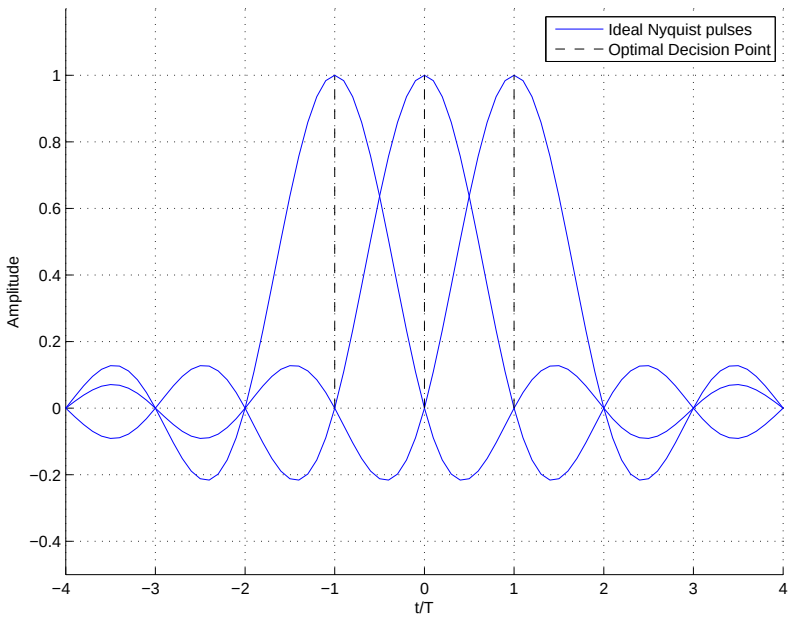


Figure 4.0.1.: Receive impulse sequence with Nyquist pulse shaping

Unfortunately this ideal Nyquist filter cannot be implemented in real systems due to its non-causal impulse response. In [BLM04] it is claimed that in real systems the minimum Nyquist bandwidth, which is established by the ideal Nyquist filter, can be extended to $W = \frac{1+\alpha}{2 \cdot T}$ where $0 \leq \alpha \leq 1$ is called excess-bandwidth or roll-off factor. If $\alpha = 1$, the spectrum of the pulse

occupies 100% more bandwidth than the ideal Nyquist pulse. The increase of the bandwidth leads to a less complex implementation due to simpler filtering. A common pulse shape with a nonzero excess bandwidth which satisfies ISI-free transmission is the raised-cosine pulse.

4.1. Raised-Cosine Pulse

The raised-cosine pulse is defined in [BLM04] by the function

$$g_{RC}(t) = \left(\frac{\sin\left(\pi \cdot \frac{t}{T}\right)}{\pi \cdot \frac{t}{T}}\right) \cdot \left(\frac{\cos\left(\alpha \cdot \pi \cdot \frac{t}{T}\right)}{1 - \left(2 \cdot \alpha \frac{t}{T}\right)^2}\right)$$

and its Fourier transform

$$G_{RC}(f) = \begin{cases} T & |f| \leq \frac{1-\alpha}{2 \cdot T} \\ T \cdot \cos^2\left[\frac{\pi \cdot T}{2 \cdot \alpha} \cdot \left(|f| - \frac{1-\alpha}{2 \cdot T}\right)\right] & \frac{1-\alpha}{2 \cdot T} < |f| \leq \frac{1+\alpha}{2 \cdot T} \\ 0 & \frac{1+\alpha}{2 \cdot T} < |f| \end{cases}$$

where T is the symbol duration.
This pulse satisfies ISI-free transmission because its zero crossings are at nonzero integer multiples of the symbol duration T. Unfortunately the raised-cosine pulse also has an infinite impulse response. Hence the pulse has to be approximated by truncating at a multiple of T for practical implementation.

4.2. Root-Raised-Cosine Pulse

In practice the raised-cosine filter function gets separated into two root-raised-cosine filters $G_{RRC} = \sqrt{G_{RC}}$ which are located at the transmitter for pulse shaping and at the receiver for matched filtering. The root-raised-cosine filter itself does not exhibit zero ISI, but the product of both receiving and transmitting filter is again a raised-cosine filter which will result in zero ISI if the timing is perfect.
In DVB-S2, which is used for the SNR estimation experiment, the transmit filter is a root-raised cosine filter with the choice of three different roll-off factors which is presented in [ETS05]. Hence a root-raised-cosine filter is implemented for matched filtering in the timing recovery application.

5. Implementation on the Software Defined Radio Platform

For implementing the matched filter, the timing estimation and the interpolation module for timing recovery on an FPGA, the USRP N210 Software Defined Radio by Ettus Research is used as the prototype device. An overview on the hardware features of this SDR are presented in [Resb] and [Resa]. Some important features of the hardware component for this thesis are listed here briefly:

- USRP N210
  - Two 100 $\frac{MS}{s}$ 14-bit analog-to-digital converters
  - Digital down converters with programmable decimation rates
  - Gigabit Ethernet interface
  - Xilinx Spartan 3A-DSP3400 FPGA
- Daughterboard WBX
  - Frequency Range: 50 MHz to 2.2 GHz

For the practical implementation this SDR is used for mixing, sampling and decimation of the received input signal as well as for the transmission of the data to the host PC.
This chapter is concerned with the existing hardware structure of the USRP N210 and the FPGA implementation of the root-raised-cosine filter, the timing estimator and the interpolator.
The RF input signal is connected to the daughterboard representing the analog receiver frontend. This component is responsible for filtering, mixing and amplification of the input RF signal. The output of the daughterboard is then AD converted. These samples are in the sequel sent to the FPGA where the digital signal processing is performed. The first two blocks on the FPGA (figure 5.1.1) are responsible for digital down-conversion and decimation. The timing estimation and interpolation module is placed after the decimation stage providing a four times oversampled signal. Using this signal, the symbols are produced by applying the presented procedures for timing estimation and interpolation developed in this thesis. This output is connected to the existing modules that are responsible for generating packets, which are then sent to the host PC using UDP packets over the Gigabit Ethernet interface. On the host PC further signal processing can be performed.

5.1. Existing Structure of the USRP N210

Figure 5.1.1 shows a simplified block diagram of the SDR structure. The timing estimation and interpolation block that is developed in this thesis is indicated in bold style.
The basic function of the first two blocks in the FPGA are briefly explained in this chapter in order to give an overview on the functionality of the existing receiver structure. Figure 5.1.2 shows the block diagram of these two modules implemented in FPGA on the USRP N210, which are located prior to the timing estimation and interpolation module.

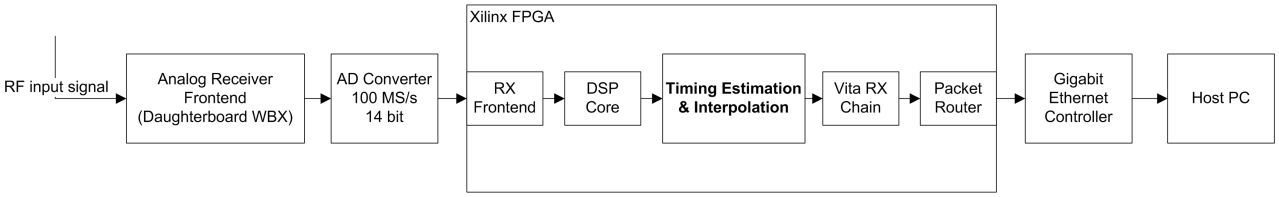


Figure 5.1.1.: Simplified block diagram of the USRP N210 and WBX daughterboard including the implemented block for timing recovery

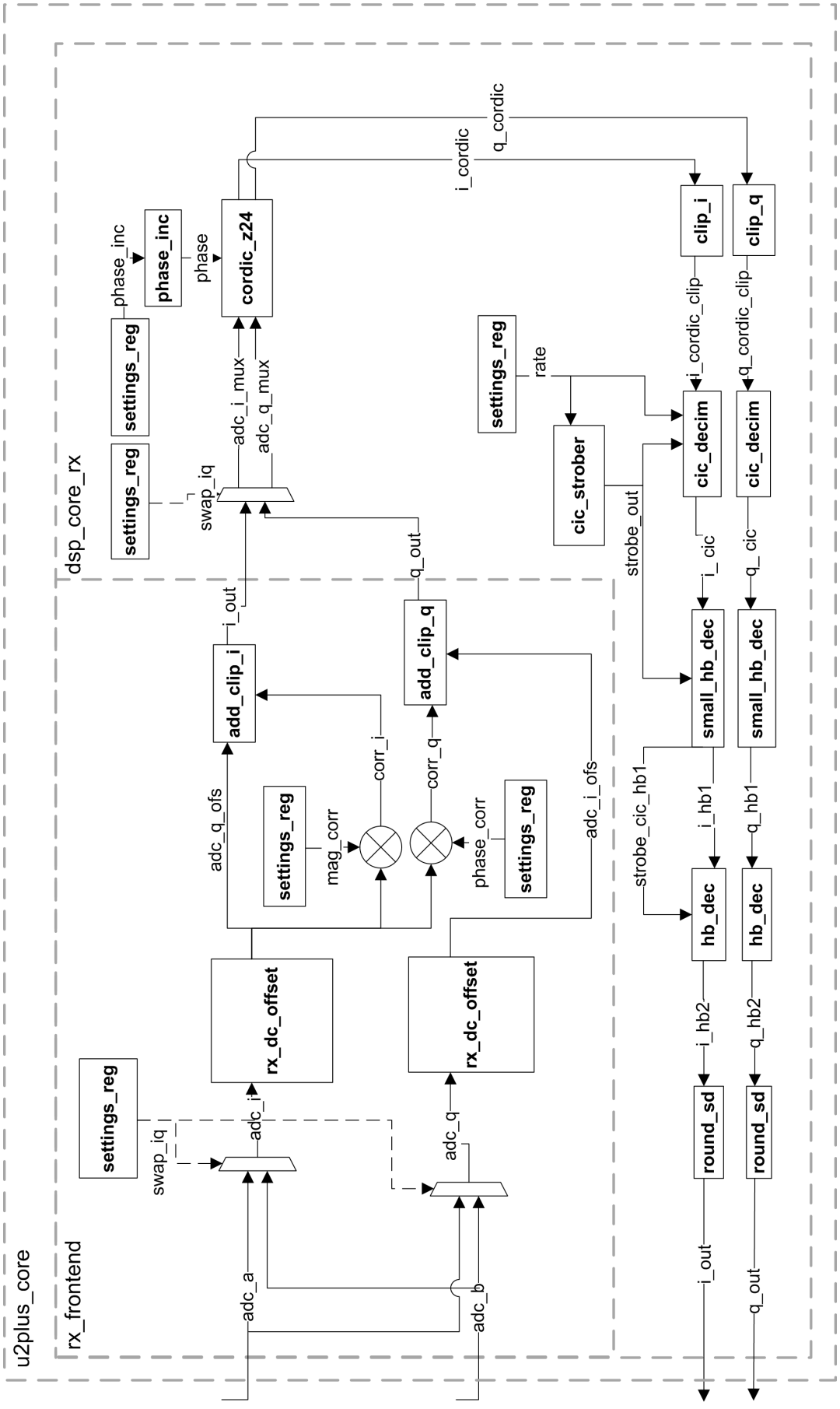


Figure 5.1.2.: Block diagram of existing receiver structure on the USRP N210

5.1.1. Analog Receiver Frontend

As already mentioned, the analog receiver frontend consists of the WBX daughterboard. It it responsible for filtering, amplification and mixing of the RF input signal prior to the AD conversion. This daughterboard has a frequency range from $50\,MHz$ to $2.2\,GHz$. The 14 bit AD converted samples are then the input of the receiver frontend, which is implemented on the FPGA of the USRP N210.

5.1.2. Receiver Frontend

The first part of the receiver structure is the receiver frontend module `rx_frontend` implemented on the FPGA, which gives the opportunity to remove unwanted DC components from the input signal as well as to compensate for I/Q imbalance due to non-ideal mixers in the analog receiver frontend.
The two input signals are the outputs of the AD converters which are extended to a 16 bit two's complementary number and represent the real part and the imaginary part of the complex valued input signal.
First of all, the possibility of swapping the real and the imaginary part of the input signal is implemented using two multiplexers. The enable signal for these multiplexers is stored in a settings register. The value of the settings register is set by the firmware running on the USRP N210 which can be controlled by the host PC.
For removing the unwanted DC component of the input signal, a DC removal filter is implemented which uses fixed-point quantization to avoid data overflow. Figure 5.1.3 shows the basic block diagram of the implemented DC removal filter which is taken from [Lyo11]. Additional information on the DC removal filter can also be found in [Lyo11].

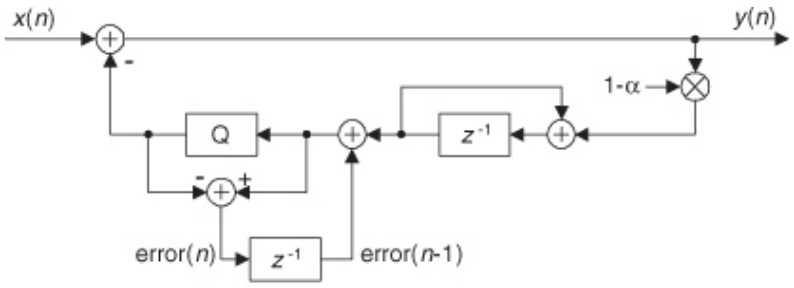


Figure 5.1.3.: DC removal filter using fixed-point quantization[Lyo11]

The final part of the module is used to compensate for any I/Q imbalance. This feature was added by the developers of Ettus Research during the end of this thesis and is still in a testing state and therefore the I/Q imbalance compensation part is omitted by default in the proposed architecture by Ettus Research.

5.1.3. DSP Core

The second part of the receiver structure is the DSP core module `dsp_core_rx`, which is implemented to apply fine tuning as well as the decimation of the input signal.
This module also gives the opportunity to swap the real and the imaginary part of the input signal using simple multiplexers. The value of the enable signal for the multiplexers is again stored in a settings register which is controlled by the firmware.

5.1.3.1. Digital Down-Conversion

Due to the limited resolution of the local oscillators on the analog receiver frontend ($25\,kHz$ steps) which is responsible for the mixing procedure, the desired frequency shift may not be realized at once. So if the desired center frequency cannot fully be achieved by the analog mixing stage, the residual frequency shift is implemented using a CORDIC DDC (Digital Down-Converter). This module is controlled by a simple counter which is used in an NCO where the phase increment is again stored in a settings register. The firmware calculates the residual frequency that could not be realized by the analog mixer and generates the corresponding phase increment in order to achieve the residual tuning in the CORDIC DDC. This phase increment then is stored in the settings register that is the input of the NCO.
The successive clipping module is used to reduce the bit width of the signals. If the input value is higher than the desired maximal output value, the output is set to the maximal output value. For negative input values, the output is set to the minimal output value if the input is smaller than the desired minimal output value. Otherwise the input value is not altered. This procedure leads to a bias-free output.

5.1.3.2. CIC Decimation

For the first part of the decimation, a four-stage CIC filter is used as a decimation filter. CIC stands for cascaded integrator comb which is a computationally efficient way of implementing lowpass filters, which is in this case used as an anti-aliasing filter prior to decimation. Due to its non-flat passband magnitude response and its low sidelobe attenuation which is claimed in [Lyo11], two half-band filters are followed by the CIC decimation stage in order to compensate for these drawbacks. Benefits of this CIC filters are the narrow-band lowpass characteristic and the absence of any multiplication which is proposed in [Lyo11]. Only additions and subtractions are used in CIC filters which can be seen in figure 5.1.4 which is taken from [Lyo11]. This figure shows a single stage CIC filter followed by a downsampling block. The simple arithmetic makes it quite popular in hardware devices and the downsampling also reduces the computational effort of the successive filter structures.
The CIC filter implemented in the existing block performs a decimation factor in the range of 1 to 128. The value of the decimation parameter is stored in a settings register which is calculated and set by the firmware according to the sampling frequency parameter set on the host PC.
Figure 5.1.5 shows the magnitude response of the four stage CIC filter with a decimation factor of 25. In this configuration the CIC filter is used for the measurements, since a $1\,\frac{MS}{s}$ signal needs to be four times oversampled which leads to a decimation of $\frac{100\,MHz}{4\,MHz} = 25$.

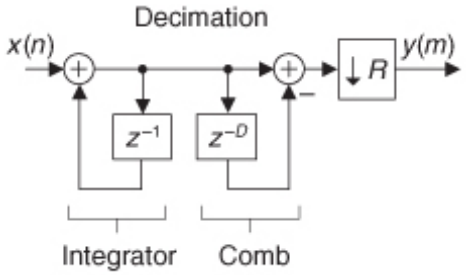


Figure 5.1.4.: Single stage decimation CIC filter[Lyo11]

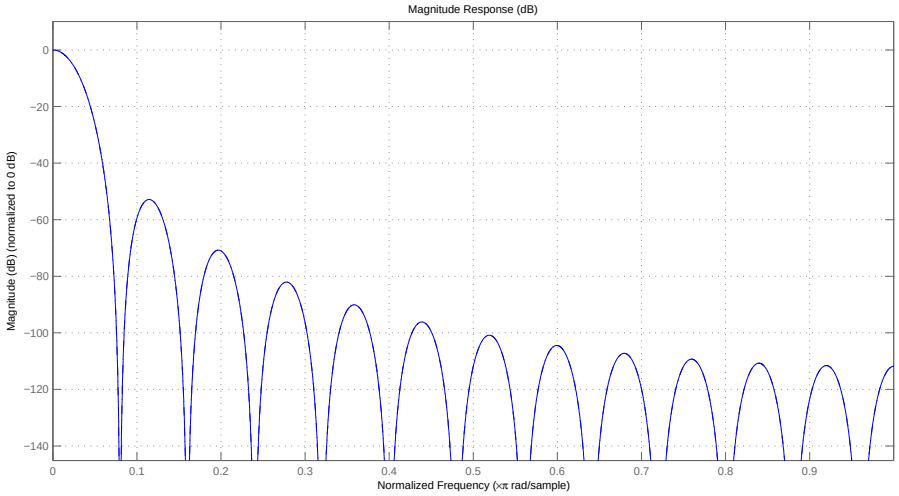


Figure 5.1.5.: Magnitude response of the four stage CIC filter with decimation factor of 25

When we take a closer look at the magnitude response of the four stage CIC filter with a decimation factor of 25, we can see that at the cut-off frequency of a root-raised-cosine filter with a symbol rate of $1\,MBaud$ and a sampling frequency of $f_s = 100\,MHz$, the attenuation is approximately $1\,dB$.

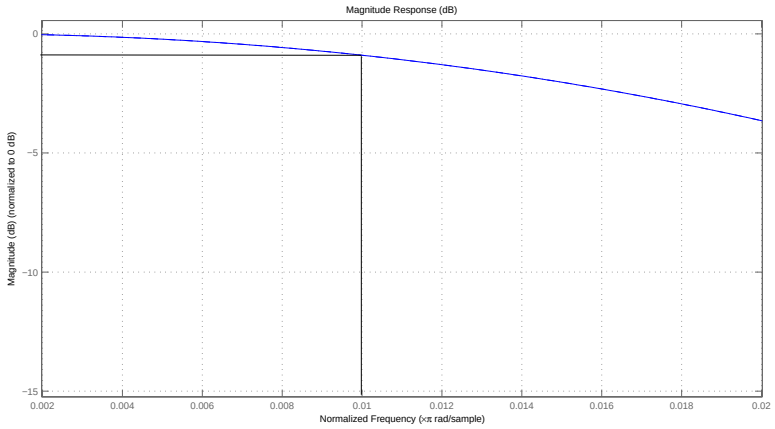


Figure 5.1.6.: Zoom into the magnitude response of the four stage CIC filter with decimation factor of 25

5.1.3.3. Half-Band Decimation

The second decimation module is a small half-band decimator. This small half-band filter must be able work at quite high sample rates and has therefore only 7 taps and decimates the input signal by 2. Another half-band filter is following which consists of 31 filter taps and also decimates the input signal by 2.
The magnitude responses of these two filters can be seen in figure 5.1.7 and 5.1.8 respectively. As it can be seen in these figures, the 3 dB attenuation is at half the sampling frequency and therefore they are called half-band filters and can be used as anti-alias filters prior to decimation by 2. When comparing the two magnitude responses, we can see that the larger

number of taps in the second half-band filter leads to steeper decay of the magnitude response as well as to higher sidelobe attenuation, but with the drawback of higher computational effort. Hence the second filter with the higher effort is placed at the lower sampling rate.

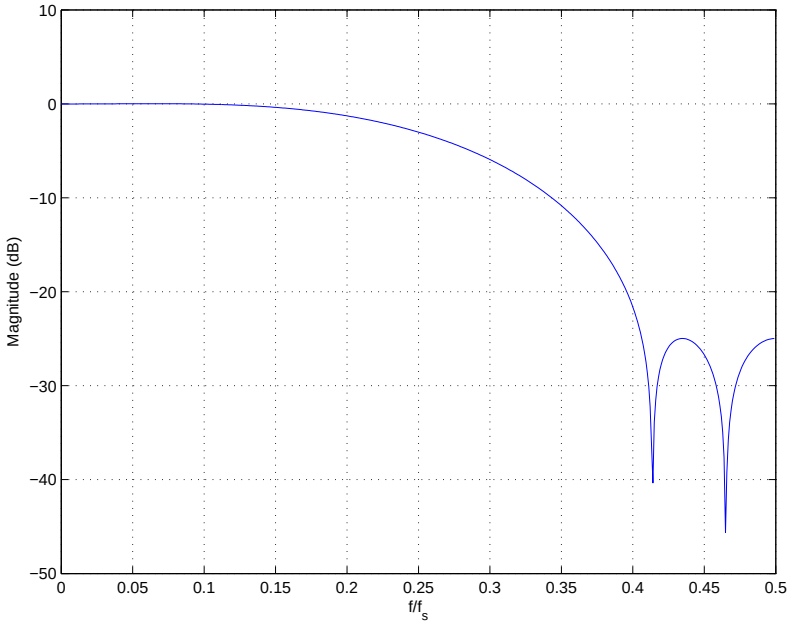


Figure 5.1.7.: Magnitude response of the small half-band filter

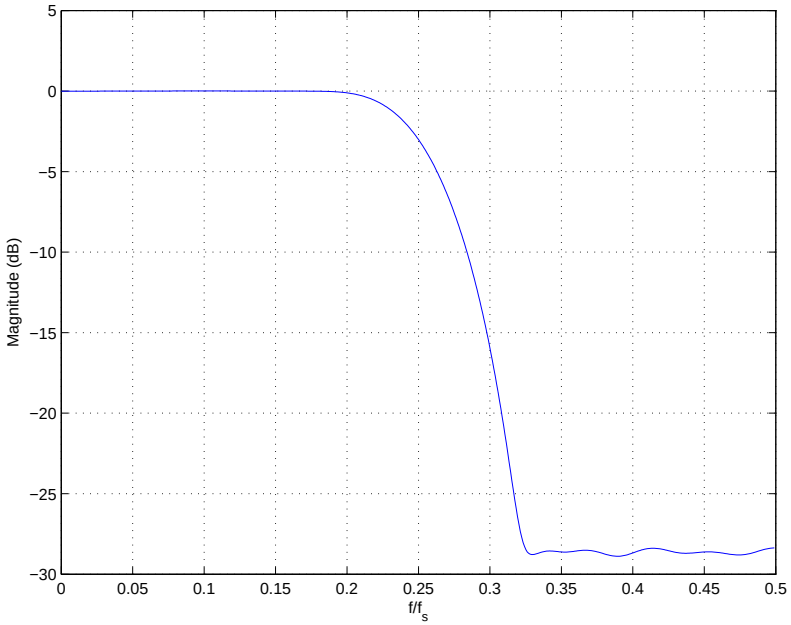


Figure 5.1.8.: Magnitude response of the second half-band filter

A strober module is also included to generate strobe signals which indicate valid input samples during the decimation procedure. This is necessary since the clock frequency is no longer equal to the sample frequency and the decimation modules need to know when a valid sample is at the input.

The combination of these three decimation modules leads to a minimum decimation factor of 4 and a maximum decimation factor of 512. The two half-band filters can also be bypassed separately to achieve every possible decimation rate in the valid range of $[4, 512]$. The setting of the decimation rate for the strober and the CIC decimation as well as the bypass signals are calculated and set by the firmware and depend on the desired decimation factor set on the host PC.
Finally the output of the decimation modules is rounded to achieve the 16 bit output signal using the round modules. These modules apply bias-free rounding towards zero.
This output is now the 16 bit two's complement input of the timing estimation and interpolation module that is developed in this thesis and presented in the next section.

5.2. Root-Raised-Cosine Module

In order to implement the root-raised-cosine filtering, a filter module needs to be implemented. Since filter operations are a fundamental procedure in digital signal processing, Xilinx provides an IP core especially for FIR filters.
Due to the fact that these available IP cores are highly optimized for the given FPGA architecture in terms of area, this IP core from Xilinx is used in this thesis instead of implementing a separate filter module.
Therefore the filter taps for the root-raised-cosine filter are produced using the Matlab `fdatool`. Due to the fact that the filter works with a $N = 4$ times oversampled signal and the filtering is performed over 8 symbols a root-raised-cosine filter with $N \cdot 8 + 1 = 33$ taps is generated. The roll-off factor for the filter is chosen with $\alpha = 0.35$ since no bandwidth limitations are given.
These coefficients are then quantized to 16 bit two's complementary binary numbers which leads to a deviation compared to the ideal impulse response. Additionally, the option of scaling the coefficients for better range utilization is enabled, which leads to a passband gain in the magnitude response of the filter. The impulse response and the magnitude response of this realized root-raised-cosine filter are shown in the figures 5.2.1 and 5.2.2.

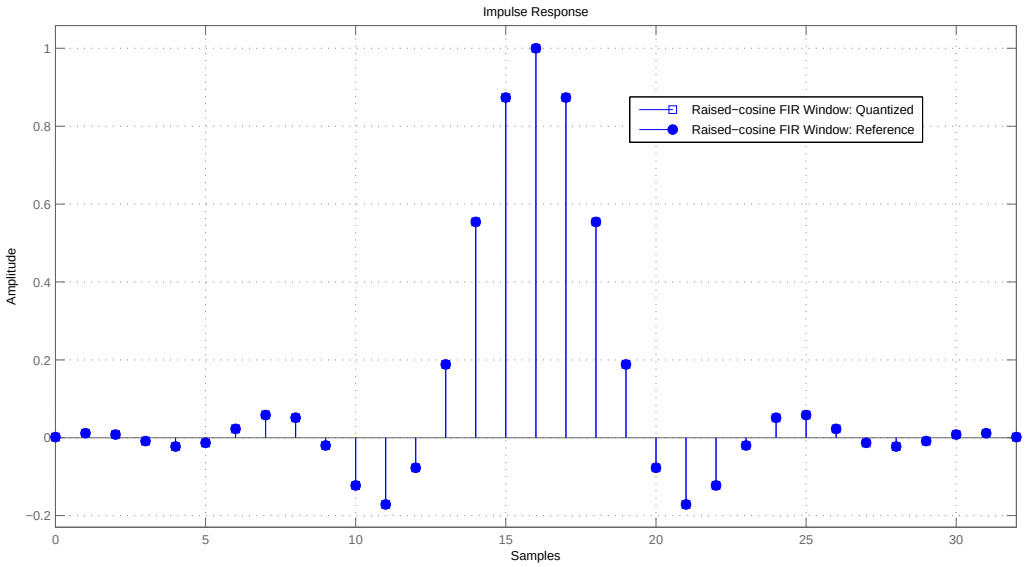


Figure 5.2.1.: Impulse response of the implemented root-raised-cosine filter

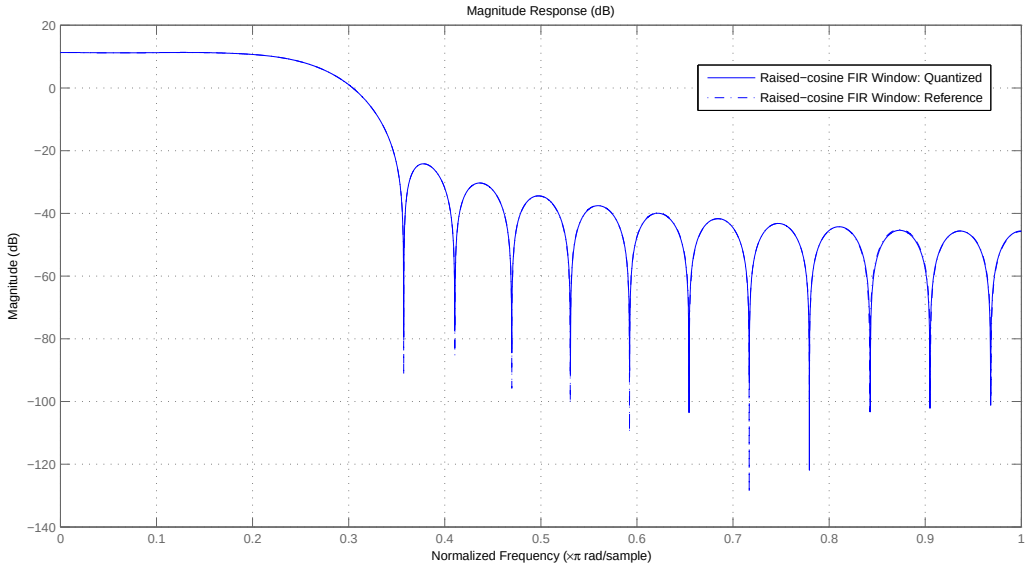


Figure 5.2.2.: Magnitude response of the implemented root-raised-cosine filter

5.3. Timing Estimation Module

The timing estimator module is the hardware module that calculates the value of the timing estimates presented in section 3. The latter are separated into an integer part and a fractional part. The integer part defines the delay to the optimal sampling time in integer values of samples and the fractional part defines the delay in fractional values of samples as mentioned in section 3.4.

Figure 5.3.1 shows the timing estimator module which consists of five input signals. `clk` is the clock signal and `reset` is the reset signal which is used to bring the module in an initial state. Due to better representation, these signals are not shown in the figures of this section. The signal `clk_en` is an enable signal which indicates valid data at the input buses `real_i` and `imag_i`. These two signals are 16 bit signed values in two's complement binary representation. `real_i` is the real part and `imag_i` is the imaginary part of the complex valued input sample.

Three output signals are produced in this module. The first one is the `int_frac_rdy_o` signal which indicates valid data on the data buses `frac_o` and `int_o`. `frac_o` is a 7 bit unsigned value from 0 to 127, which indicates the fractional delay to the optimal sampling time. `int_o` is a three bit signed value from -2 to 2, which indicates the integer delay to the optimal sampling time. The input and output signals are also summarized in table 5.3.1.

The detailed description of the containing submodules is handled in the next sections.

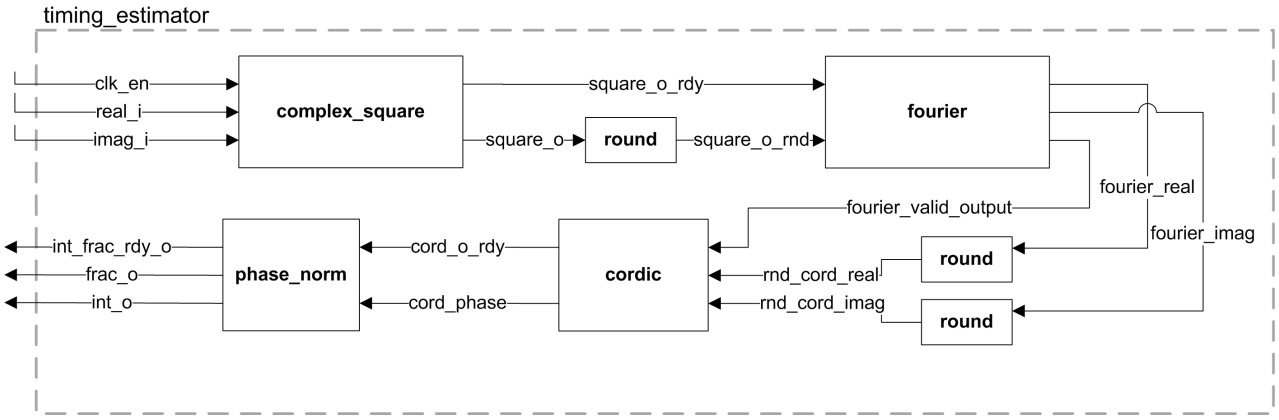


Figure 5.3.1.: Block diagram of the timing estimator module

Name	Type	Bit Width	Format
clk	input	1	-
clk_en	input	1	-
reset	input	1	-
real_i	input	16	2's complement
imag_i	input	16	2's complement
int_o	output	3	2's complement
frac_o	output	7	unsigned
int_frac_rdy_o	output	1	-

Table 5.3.1.: Input and output signals of the timing estimation module

5.3.1. Complex Square Module

This module is used to compute the square of the absolute value which is needed in equation 3.1.4. It squares the real part and the imaginary part separately and adds the two products. So the output of the module is given by

$$y[i] = |x[i]|^2 = x_{real}^2 + x_{imaginary}^2$$

Figure 5.3.2 shows the basic structure of this module. It consists of five input signals. `clk_en` indicates valid data on the data buses `real_i` and `imag_i` which represent the complex valued input sample. Both signals are in 16 bit two's complement binary representation. The two data signals get squared using two multipliers and the two products are added and stored in a 32 bit register `x_reg`. The register stores the new input values at the positive edge of `clk` and when `clk_en` is high. The `reset` signal is used to reset the register.

The output is a 32 bit signed value using two's complement.

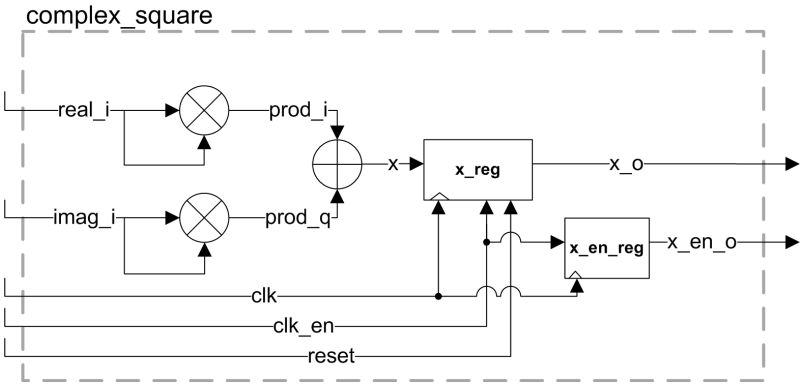


Figure 5.3.2.: Block diagram of the complex_square module

Name	Type	Bit Width	Format
clk	input	1	-
clk_en	input	1	-
reset	input	1	-
real_i	input	16	2's complement
imag_i	input	16	2's complement
x_o	output	32	2's complement
x_en_o	output	1	-

Table 5.3.2.: Input and output signals of the complex_square module

The hardware estimation for the module is shown in table 5.3.3.

Register	1 x 32 bit	1 x 1 bit
Adders	1 x 32 bit	
Multipliers	2 x 16 bit	

Table 5.3.3.: Hardware estimation of the complex_square module

5.3.2. Fourier Module

The Fourier module is responsible for the calculation of the spectral component of the signal at the symbol rate (see equation 3.1.9). Figure 5.3.3 shows the block diagram of this module which consists of four input signals and three output signals. The signal `clk` is the clock signal and `clk_en` indicates valid data on the `in_i` signal. The `reset` signal is used to bring the module into a defined initial state. The input signal `in_i` is in 16 bit two's complement representation.
The control block basically consists of two counters. The first one is a two-bit counter from 0 to $N-1$, which is used to select the addresses of the two multiplexers as well as to send the enable signals to the two registers `real_acc` and `imag_acc`. This counter decides whether the input signal is multiplied with ± 1 or $\pm j$. The second counter is a six bit counter from 0 to $L-1$ for counting the number of samples per section. During one section, the registers `real_acc` and `imag_acc` store the intermediate results of the summation in equation 3.1.9. If a section has passed, the `output_rdy_o` signal is set high, which indicates that the signals

`real_o` and `imag_o` have valid data and the two accumulator register are set to zero. Now these two output signals represent the complex valued Fourier coefficient.

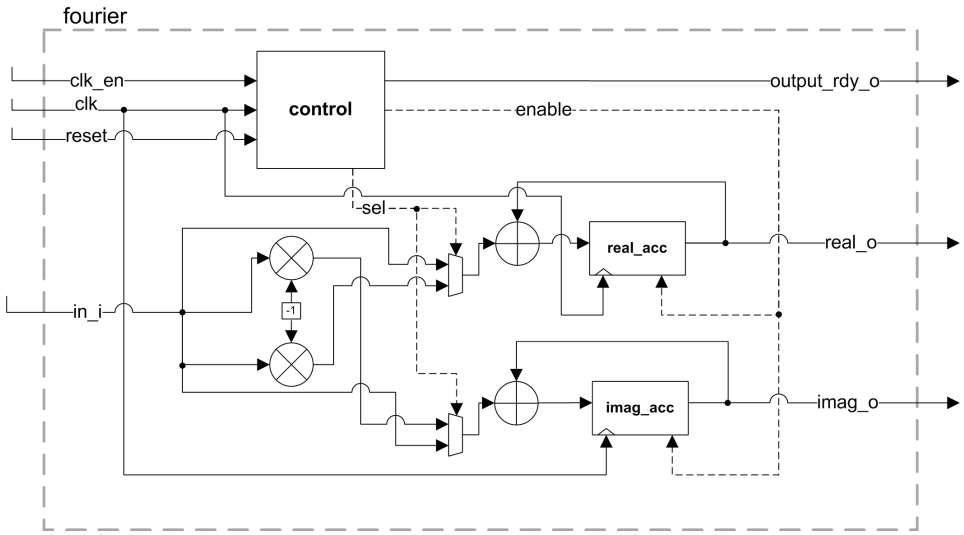


Figure 5.3.3.: Block diagram of the Fourier module

Name	Type	Bit Width	Format
clk	input	1	-
clk_en	input	1	-
reset	input	1	-
in_i	input	16	2's complement
real_o	output	20	2's complement
imag_o	output	20	2's complement
output_rdy_o	output	1	-

Table 5.3.4.: Input and output signals of the Fourier module

The two multipliers in figure 5.3.3 are only used to perform a sign switch. This sign switch and the following addition can also be realized using two adders/subtractors on which addition or subtraction can be selected. The resulting hardware estimation for the Fourier module is shown in table 5.3.5.

Register	2 x 20 bit	1 x 1 bit
Adders/Subtractors	2 x 20 bit	
Multipliers	2 x 16 bit	
Counters	1 x 2 bit	1 x 6 bit

Table 5.3.5.: Hardware estimation of Fourier module

5.3.3. CORDIC Module

After calculating the complex valued Fourier coefficient, equation 3.1.8 requires the calculation of the argument of this complex number. The angle Θ of a complex valued number $x = I + j \cdot Q$ is given by

$$\Theta = \arctan\left(\frac{Q}{I}\right)$$

One method of calculating the arc tangent is using look-up tables. The value of $\frac{Q}{I}$ can then be used as an address to this look-up table which contains an approximate value for Θ. If high accuracy is requested in the arc tangent calculation, this look-up table may require a large amount of memory.
Another method is the CORDIC (**co**ordinate **r**otation **di**gital **c**omputer) algorithm where only binary shifts and additions are used. Unfortunately this algorithm requires more time than reading from a look-up table. But since time is not the optimization goal in this thesis, this algorithm is very suitable. More information on this algorithm is presented in section 3.2.
Since the CORDIC algorithm is a very common way of calculating trigonometric functions, Xilinx provides an IP core for this algorithm. The description of the IP core is provided in [Xil11].
In the arc tangent configuration, the CORDIC module consists of four input signals and two output signal. The signal `clk` is the clock signal and `nd` is the signal that indicates valid data at the inputs `real_i` and `imag_i`. Where `real_i` represents the real part and `imag_i` the imaginary part of the complex valued input signal. The block diagram can be seen in figure 5.3.4 and the overview on the input and output signals is summarized in table 5.3.6.

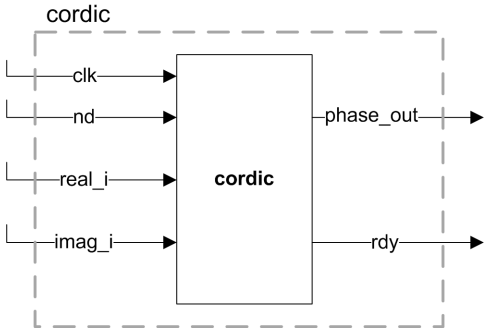


Figure 5.3.4.: Block diagram of the CORDIC module

Name	Type	Bit Width	Format
clk	input	1	-
nd	input	1	-
real_i	input	16	2's complement
imag_i	input	16	2's complement
phase_out	output	8	signed 2Q5
rdy	output	1	-

Table 5.3.6.: Input and output signals of the CORDIC module

5.3.4. Phase Normalization Module

The result of the CORDIC output is the angle of the complex Fourier coefficient. This angle now needs to be normalized according to equation 3.1.8. Additionally, the resulting

normalized phase has to be separated into an integer part and a fractional part as presented in equation 3.4.1 and 3.4.2 in order to use the result in the interpolator module.

The module consists of four input signals and three output signals. The signal `clk` is the clock signal of this module and `clk_en` indicates valid data at the input `phase_i`. The `reset` signal is used to reset the registers in the module.

The input signal `phase_i` gets multiplied with either `mult_factor` or `neg_mult_factor` according to the sign of the input signal, which is indicated by the leftmost bit of `phase_i` and the signal `prod_sign`. The two signals `mult_factor` and `neg_mult_factor` have the value of the normalization factor $\pm\frac{1}{2\cdot\pi}\cdot 4$ quantized to the signed $0Q7$ format. The additional factor of 4 is used to fully utilize all bits of these two signals. The reason for distinguishing between a positive and negative normalization factor is that a positive multiplication result leads to easier separation in hardware. Therefore a negative input signal gets multiplied with a negative normalization factor and a positive input signal gets multiplied with a positive normalization factor to achieve a positive product. This multiplication results therefore in a 16 bit signed $3Q12$ signal `prod` which gets then rounded to obtain the signed $3Q7$ signal `prod_rnd`.

This signal `prod_rnd` is effectively a signed $1Q9$ number due to the additional multiplication factor of 4 which equals a binary shift to the left by 2. Hence the signal `prod_rnd` consists of 1 sign bit, 1 integer bit and 9 fractional bits. The integer bit is always zero because the result is always in the interval $[-0.5, 0.5]$. Using this knowledge, the first two fractional bits can be used to distinguish the range of the value, because the first fractional bit indicates the value $2^{-1} = 0.5$ and the second fractional bit indicates the value $2^{-2} = 0.25$. Using these two fractional bits and the `prod_sign` signal, the fractional value and the integer value are then calculated according to equation 3.4.1 and 3.4.2 and finally stored in the two registers `int` and `frac`. The `rdy` register is needed since the `clk_en` signal needs to be delayed by one clock cycle until the output value is valid.

The block diagram of the phase_norm module can be seen in figure 5.3.5 and the resulting hardware estimation is shown in table 5.3.8.

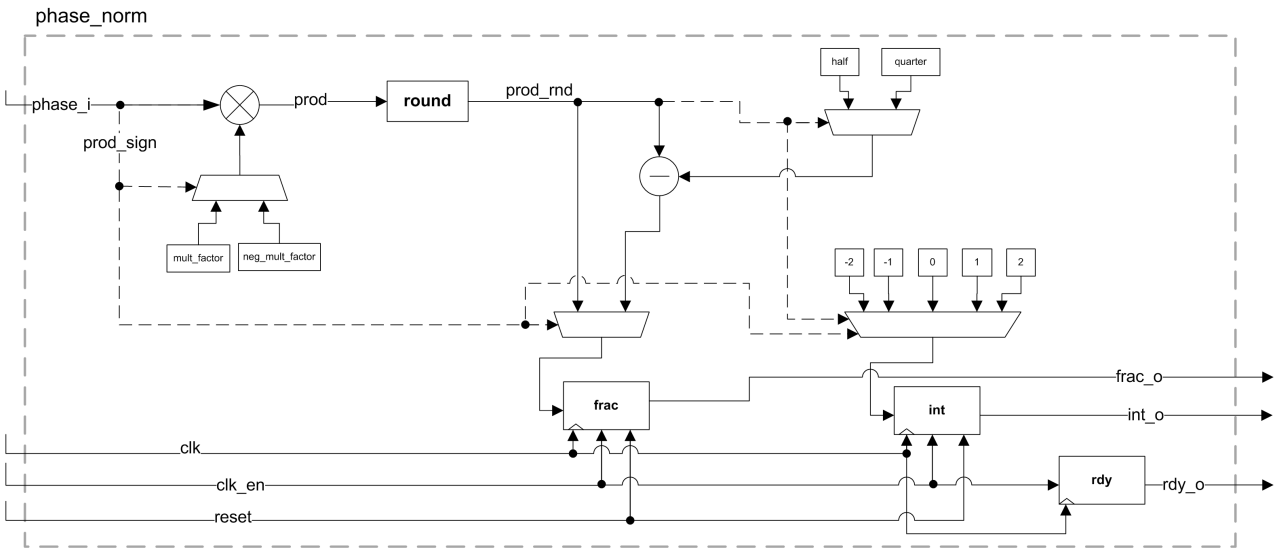


Figure 5.3.5.: Block diagram of the phase_norm module

Name	Type	Bit Width	Format
clk	input	1	-
clk_en	input	1	-
reset	input	1	-
phase_i	input	8	signed 2Q5
frac_o	input	7	unsigned integer
int_o	output	3	2's complement
rdy_o	output	1	-

Table 5.3.7.: Input and output signals of the phase_norm module

Register	2 x 20 bit	1 x 1 bit
Adders/Subtractors	2 x 20 bit	
Multipliers	2 x 16 bit	
Counters	1 x 2 bit	1 x 6 bit

Table 5.3.8.: Hardware estimation of the phase_norm module

5.3.5. Rounding Module

Since a high bus width leads to high effort on hardware, rounding is needed. Simple truncation of the additional bits would lead to a bias, because there is no discrimination between positive or negative input value. In this case, truncation would equal a rounding towards negative infinity. In order to avoid adding a bias to the signal, the modules in this thesis use symmetric rounding towards zero. Which means that if a value $x \in \mathbb{R}$ is greater than 0, it gets rounded down and if it's less than 0, it gets rounded up:

$$y = \begin{cases} \lfloor x \rfloor & x > 0 \\ x & x = 0 \\ \lceil x \rceil & x < 0 \end{cases}$$

In the hardware module this means that if the input value is negative and the truncated bits are not zero, 1 is added to the input value and the residual bits get truncated. This can be implemented by XOR-ing all the bits that would be truncated to check if they are unequal to zero and then AND it with the sign bit of the input signal `in_i`. If this results in a true value, a 1 gets added to the input signal an then the output signal `out_o` is the truncated result. If the input signal is positive or all the truncated bits are zero, the output signal `out_o` is simply the truncated input signal `in_i`. The block diagram can be seen in figure 5.3.6.

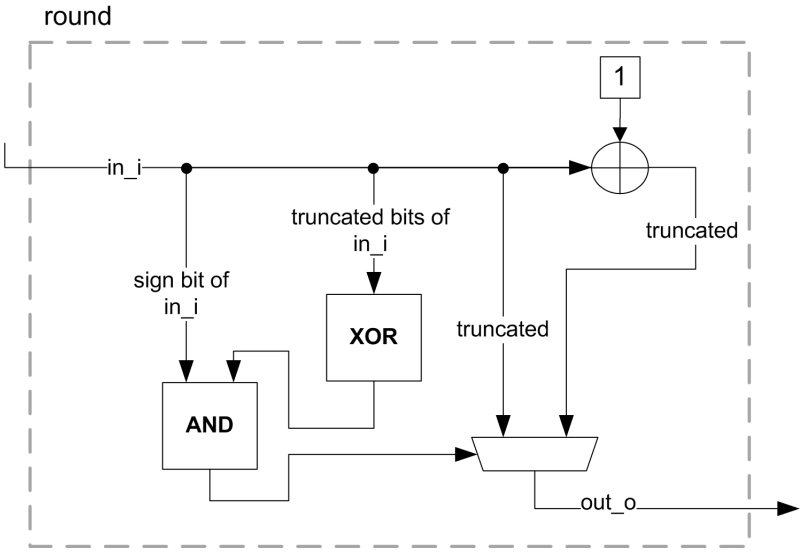


Figure 5.3.6.: Block diagram of the rounding module

5.4. FIR Interpolation Module

The interpolation module consists of two submodules. First of all, the delay_line is needed since the input samples need to be delayed until the result of the timing estimator module is ready. The second submodule is the sinc_fir module which computes the filter operation that is used for interpolation. Also the control logic is integrated in the interpolation module.

This module consists of eight input signals and three output signals. The input signals `clk` and `reset` are used for the clock signal and resetting the whole module for reaching an initial state. The signal `clk_en` indicates whether valid data is at the input signals `real_i` and `imag_i`, while the input signal `mue_rdy_i` is used to indicate valid data at the input signals `mue_frac_i` and `mue_int_i`.

The data signals `real_i` and `imag_i` represent the real part respectively the imaginary part of the complex valued input sample. `mue_frac_i` and `mue_int_i` represent the fractional value and the integer value of the timing estimation result.

Figure 5.4.1 shows the block diagram of this module and table 5.4.1 summarizes the input and output signals. Due to better representation the signals `clk` and `reset` are excluded in this figure.

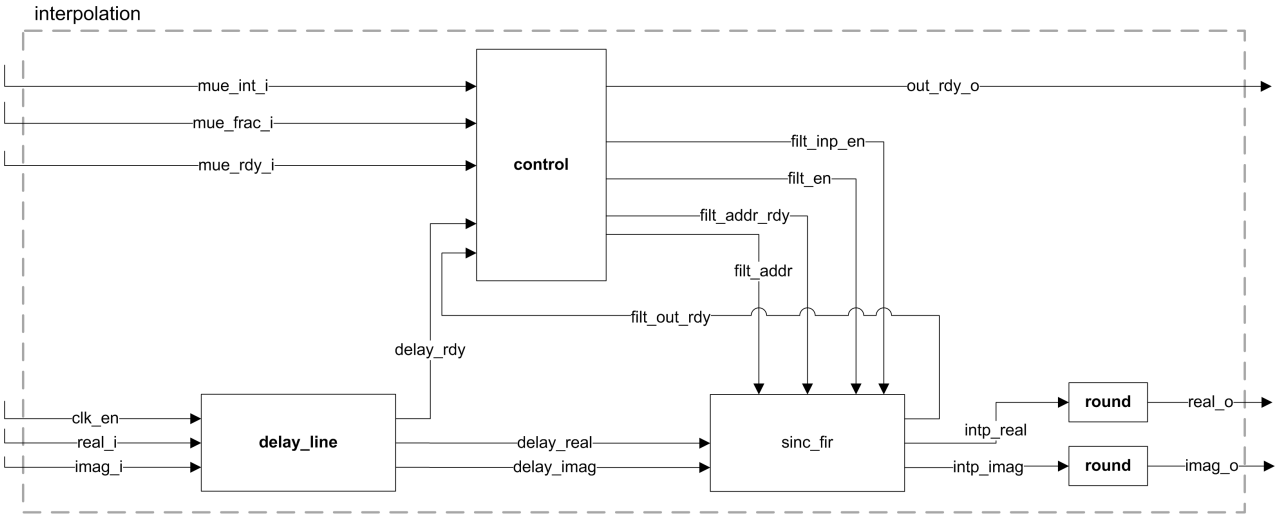


Figure 5.4.1.: Block diagram of the interpolation module

Name	Type	Bit Width	Format
clk	input	1	-
clk_en	input	1	-
reset	input	1	-
real_i	input	16	2's complement
imag_i	input	16	2's complement
mue_int_i	input	3	2's complement
mue_frac_i	input	7	unsigned
mue_rdy_i	input	1	-
real_o	output	16	2's complement
imag_o	output	16	2's complement
out_rdy_o	output	1	-

Table 5.4.1.: Input and output signals of the interpolator module

5.4.1. Delay Line Module

Since the timing estimator module presented in section 5.3 needs some time until the result is available, the input signal needs to be delayed so that the result of the timing estimator matches to the corresponding input samples. This delay D is implemented in the module delay_line using shift registers for the signals `real_i` and `imag_i`. The output signals `delay_real` and `delay_imag` are therefore the input signals delayed by D samples. The output signal `delay_rdy` is equal to the input signal `clk_en` which indicates valid input data. The block diagram of the delay_line module can be seen in figure 5.4.2 and the input and output signals are summarized in table 5.4.2.

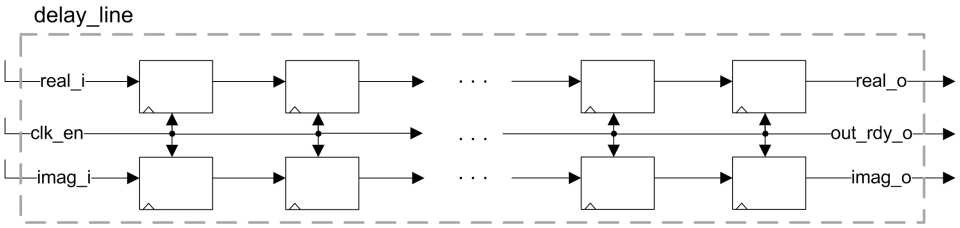


Figure 5.4.2.: Block diagram of the delay_line module

Name	Type	Bit Width	Format
clk	input	1	-
clk_en	input	1	-
reset	input	1	-
real_i	input	16	2's complement
imag_i	input	16	2's complement
real_o	output	16	2's complement
imag_o	output	16	2's complement
out_rdy_o	output	1	-

Table 5.4.2.: Input and output signals of the delay_line module

5.4.2. Control Logic of the Interpolator

The key function of the interpolator module is to apply the output of the timing estimator module to calculate the desired output symbol. Therefore an own process is implemented in which the controller stores the current `mue_int_i` and `mue_frac_i` values from the timing estimator every time the `mue_rdy_i` signal is high. Also the previous `mue_int` value is stored in a register because the difference between the old and the new `mue_int` value is needed when an update of the symbol timing is performed which is explained in section 3.4.1.

5.4.2.1. State Machine

For setting the control signals for the sinc_fir module there is also a state machine implemented which consists of three states.

INIT State

In this state the controller counts the number of samples that arrive from the delay_line module which is signalled by a high `delay_rdy` signal. Every time a sample arrives, the controller sets the `filt_inp_en` and the `filt_en` signals high so that the sinc_fir module stores the incoming samples from the delay_line module.
When the reference sample from the timing estimator module arrives at the sinc_fir module, the controller sets the `filt_addr` signal to the current `mue_frac` value and sets the `filt_addr_rdy` signal high. After that the controller jumps to the next state CONT_OUTPUT.

CONT_OUTPUT State

In this state the controller sets the signals `filt_inp_en` and `filt_en` high every time a new input sample arrives from the delay_line module. When the sinc_fir module has finished

the filter operation, it sets the signal `filt_out_rdy` high. The controller counts these filter outputs and sets the `out_rdy_o` signal high on every $N = 4$'th sinc_fir output.

When new values arrive from the timing estimator, the controller jumps to the next state UPDATE_MUE.

UPDATE_MUE State

This state is needed since new values from the timing estimator have to be considered. The `filt_addr` signal is set to the new `mue_frac` value and the `filt_addr_rdy` signal is set high. The controller stays in this state until the new reference sample of the timing estimator is at the sinc_fir module and then the controller jumps back to the state CONT_OUTPUT.

5.4.3. FIR Module

This module implements the filter procedure presented in section 2.2.4. Due to the small number of available slices in the used FPGA, only eight taps are stored of the filter impulse response. Also the possibility of linear interpolation between successive filter taps is discarded due to the high complexity and low number of available slices, because for the linear interpolation function an additional table of differences would be necessary. Also instead of calculating the table index online, there is a filter function stored for every possible μ value.

Since only interpolation and no actual decimation is needed, the filter is designed such that the cut-off frequency of the resulting filter is $\frac{f_s}{4}$. This ensures that also a signal with an excess bandwidth, like it is the case when using a root-raised-cosine filter, is only slightly attenuated by the filter. Therefore the interpolation filter is designed with the parameters $N_z = 2$ and a decimation of $\rho = \frac{1}{2}$.

For implementing $N_z = 2$ and a decimation of $\rho = \frac{1}{2}$, a filter of length $2 \cdot N_z \cdot \frac{1}{\rho} + 1 = 9$ filter taps is needed for every μ value. Since the last tap of all these impulse responses is always zero, only $N_{Taps} = 8$ taps are needed. The interpolation value μ is stored in a 7 bit signal, hence there are $2^7 = 128$ possible values for μ. This means that the filter taps are stored in a ROM with $2^7 \cdot N_{Taps} = 128 \cdot 8 = 1024$ 16 bit values.

The basic impulse response of the implemented filter is shown in figure 5.4.3. Figure 5.4.4 shows the corresponding magnitude response.

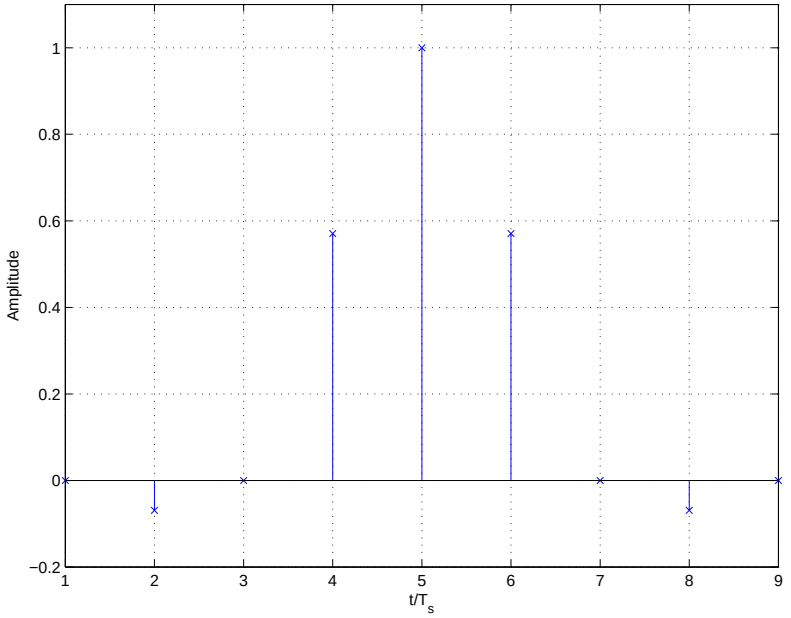


Figure 5.4.3.: Impulse response of the implemented Kaiser windowed sinc filter with $N_z = 2$ and $\beta = 4$

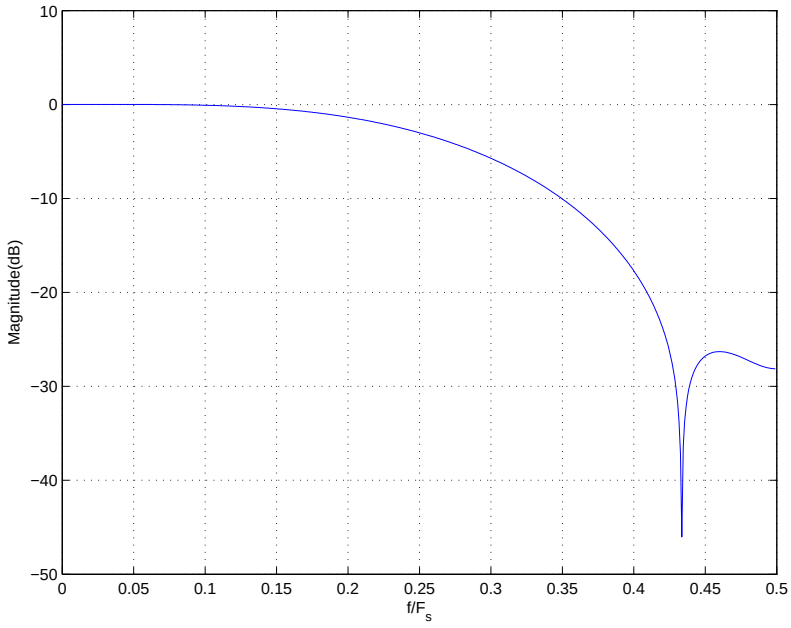


Figure 5.4.4.: Magnitude response of the implemented Kaiser windowed sinc filter with $N_z = 2$ and $\beta = 4$

One possibility for the implementation on the FPGA would be to use the FIR IP core from Xilinx, just like in the case of the root-raised-cosine filter module. This core also provides reloadable filter coefficients which is necessary for this application. But since the IP core uses filter architectures which need some time until the output engages to new filter coefficients (transposed form), the decision was made to implement this module without the use of the Xilinx core generator.

This FIR module consists of eight input signals and three output signals. `clk` and `reset` are the clock signal and the reset signal correspondingly, which is used to define an initial state of the registers. Figure 5.4.5 shows the simplified block diagram of this module where the clock signal (`clk`), the enable signals (`clk_en`, `inp_en`), the ready signal (`out_rdy_o`) and the `reset` signal are omitted due to better illustration. Also only the real part path is shown, since the imaginary path is equal to the real one.

The registers for the input samples are enabled on the `inp_en` signal and the registers for storing the addition results are enabled on the `clk_en` signal. This means that if the `inp_en` signal is high, the input samples are stored and shifted in the input registers. And if the `clk_en` signal is high, the results of the additions are stored in the registers.

The signal `addr_i` is used to select the address of the filter taps that are used for the current filter operation. When the filter output calculation is finished, the signal `out_rdy_o` is set high to indicate valid data at the signals `real_o` and `imag_o`.

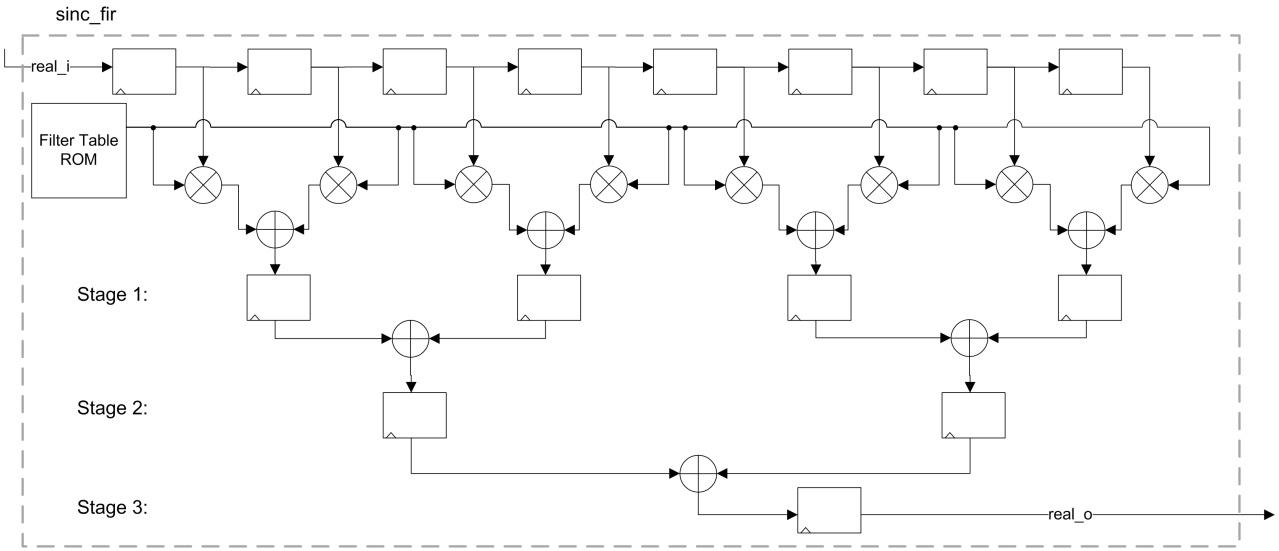


Figure 5.4.5.: Block diagram of the sinc_fir module

Name	Type	Bit Width	Format
clk	input	1	-
clk_en	input	1	-
reset	input	1	-
inp_en	input	1	-
addr_i	input	7	unsigned
addr_rdy_i	input	1	-
real_i	input	16	2's complement
imag_i	input	16	2's complement
real_o	output	33	2's complement
imag_o	output	33	2's complement
out_rdy_o	output	1	-

Table 5.4.3.: Input and output signals of the sinc_fir module

5.5. Device Utilization

The utilization of the FPGA was one major issue in this thesis, since the implemented modules on the USRP N210 FPGA already occupy a large number of slices. Some of these modules could be removed, like the transmitter module or the second receiver module, but the existing device utilization was still high.

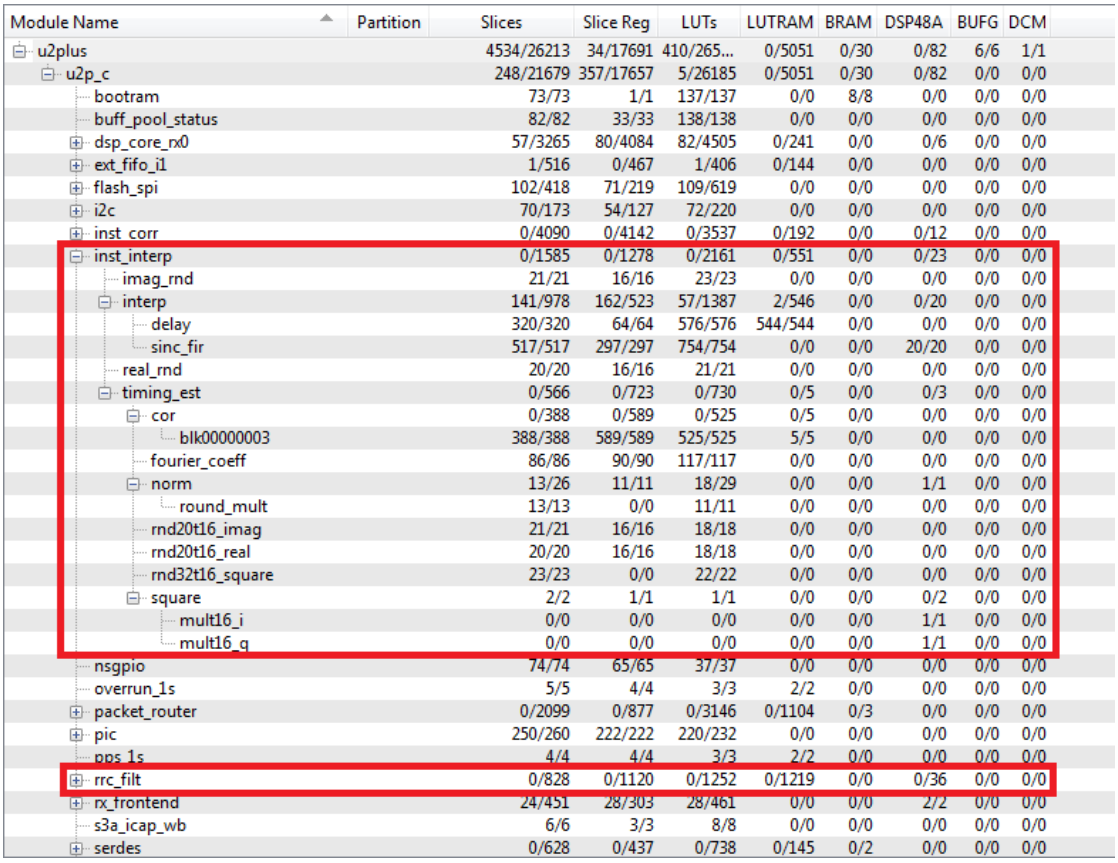
During the Place&Route process of the FPGA image creation, this high utilization leads to a much higher effort. If the utilization is too high, the timing constraints cannot be fulfilled and the created FPGA image is corrupted.

Figure 5.5.1 shows the device utilization summary where the number of occupied slices is highlighted. This 74% device utilization leads to no failing constraints and a working FPGA image. Hence it was necessary to keep the implemented module as small as possible in order to achieve a succeeding Place&Route process and a working FPGA image.

Device Utilization Summary				[-]
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	17,691	47,744	37%	
Number of 4 input LUTs	26,595	47,744	55%	
Number of occupied Slices	17,687	23,872	74%	
Number of Slices containing only related logic	17,687	17,687	100%	
Number of Slices containing unrelated logic	0	17,687	0%	
Total Number of 4 input LUTs	27,752	47,744	58%	
Number used as logic	21,544			
Number used as a route-thru	1,157			
Number used for Dual Port RAMs	512			
Number used as Shift registers	4,539			
Number of bonded IOBs	338	469	72%	
IOB Flip Flops	323			
IOB Master Pads	4			
IOB Slave Pads	4			
Number of ODDR2s used	2			
Number of BUFGMUXs	6	24	25%	
Number of DCMs	1	8	12%	
Number of ICAPs	1	1	100%	
Number of DSP48As	82	126	65%	
Number of RAMB16BWERs	30	126	23%	
Number of ICAP_SPARTAN3As	1	1	100%	
Average Fanout of Non-Clock Nets	3.38			

Figure 5.5.1.: Device utilization of the FPGA generated by Xilinx ISE Design Suite

A detailed utilization of the implemented modules is shown in figure 5.5.2. The interpolator instance `inst_interp` which is the top module of the interpolator `interp` and the timing estimator `timing_est`, as well as the root-raised-cosine filter `rrc_filt` is highlighted. It is shown in figure 5.5.1 that the number of available slices on the FPGA is 23872. Since the interpolator and the root-raised-cosine module occupy $1585 + 828 = 2413$ slices, the device utilization for the implemented modules is 10.1%.



Module Name	Partition	Slices	Slice Reg	LUTs	LUTRAM	BRAM	DSP48A	BUFG	DCM
u2plus		4534/26213	34/17691	410/265...	0/5051	0/30	0/82	6/6	1/1
u2p_c		248/21679	357/17657	5/26185	0/5051	0/30	0/82	0/0	0/0
bootram		73/73	1/1	137/137	0/0	8/8	0/0	0/0	0/0
buff_pool_status		82/82	33/33	138/138	0/0	0/0	0/0	0/0	0/0
dsp_core_rx0		57/3265	80/4084	82/4505	0/241	0/0	0/6	0/0	0/0
ext_fifo_i1		1/516	0/467	1/406	0/144	0/0	0/0	0/0	0/0
flash_spi		102/418	71/219	109/619	0/0	0/0	0/0	0/0	0/0
i2c		70/173	54/127	72/220	0/0	0/0	0/0	0/0	0/0
inst_corr		0/4090	0/4142	0/3537	0/192	0/0	0/12	0/0	0/0
inst_interp		0/1585	0/1278	0/2161	0/551	0/0	0/23	0/0	0/0
imag_rnd		21/21	16/16	23/23	0/0	0/0	0/0	0/0	0/0
interp		141/978	162/523	57/1387	2/546	0/0	0/20	0/0	0/0
delay		320/320	64/64	576/576	544/544	0/0	0/0	0/0	0/0
sinc_fir		517/517	297/297	754/754	0/0	0/0	20/20	0/0	0/0
real_rnd		20/20	16/16	21/21	0/0	0/0	0/0	0/0	0/0
timing_est		0/566	0/723	0/730	0/5	0/0	0/3	0/0	0/0
cor		0/388	0/589	0/525	0/5	0/0	0/0	0/0	0/0
blk00000003		388/388	589/589	525/525	5/5	0/0	0/0	0/0	0/0
fourier_coeff		86/86	90/90	117/117	0/0	0/0	0/0	0/0	0/0
norm		13/26	11/11	18/29	0/0	0/0	1/1	0/0	0/0
round_mult		13/13	0/0	11/11	0/0	0/0	0/0	0/0	0/0
rnd20t16_imag		21/21	16/16	18/18	0/0	0/0	0/0	0/0	0/0
rnd20t16_real		20/20	16/16	18/18	0/0	0/0	0/0	0/0	0/0
rnd32t16_square		23/23	0/0	22/22	0/0	0/0	0/0	0/0	0/0
square		2/2	1/1	1/1	0/0	0/0	0/2	0/0	0/0
mult16_i		0/0	0/0	0/0	0/0	0/0	1/1	0/0	0/0
mult16_q		0/0	0/0	0/0	0/0	0/0	1/1	0/0	0/0
nsgpio		74/74	65/65	37/37	0/0	0/0	0/0	0/0	0/0
overrun_1s		5/5	4/4	3/3	2/2	0/0	0/0	0/0	0/0
packet_router		0/2099	0/877	0/3146	0/1104	0/3	0/0	0/0	0/0
pic		250/260	222/222	220/232	0/0	0/0	0/0	0/0	0/0
pps_1s		4/4	4/4	3/3	2/2	0/0	0/0	0/0	0/0
rrc_filt		0/828	0/1120	0/1252	0/1219	0/0	0/36	0/0	0/0
rx_frontend		24/451	28/303	28/461	0/0	0/0	2/2	0/0	0/0
s3a_icap_wb		6/6	3/3	8/8	0/0	0/0	0/0	0/0	0/0
serdes		0/628	0/437	0/738	0/145	0/2	0/0	0/0	0/0

Figure 5.5.2.: Module level utilization of the FPGA generated by Xilinx ISE Design Suite

6. Simulation Results

For the simulation results presented in this chapter, the implemented software modules are adapted to a simulation environment developed by Joanneum Research. By using this simulation environment, the generation of desired output values for different parameters like SNR is simplified compared to the GNU Radio environment.

6.1. Timing Estimation

6.1.1. Variance of the Timing Estimator

The variance of the timing estimation result is a suitable measurement for the quality of the estimator. Figure 6.1.1 shows the variance of the timing estimator for different window lengths L as a function of the signal-to-noise ratio. As it can be seen, the variance gets lower the higher the window size L and the SNR get. Since the higher SNR leads to less noise on the signal and the higher window size leads to more samples that are available for the estimation, this result is no surprise.
Worth mentioning is the high difference in the variance between low window lengths. If we have a look at the $L = 64$ and the $L = 128$ curve, we can see that the difference in the variance is significant. Whereas the difference between the curves for $L = 832$ and $L = 1024$ is much smaller. As a result, it can be seen that in practice it is recommended to use higher window lengths. But when reaching a certain window length, an increase is not reasonable when looking at the high computational effort compared to the quite low decrease of the variance. For practical implementation a trade-off between a acceptable variance and possible hardware effort must be found.

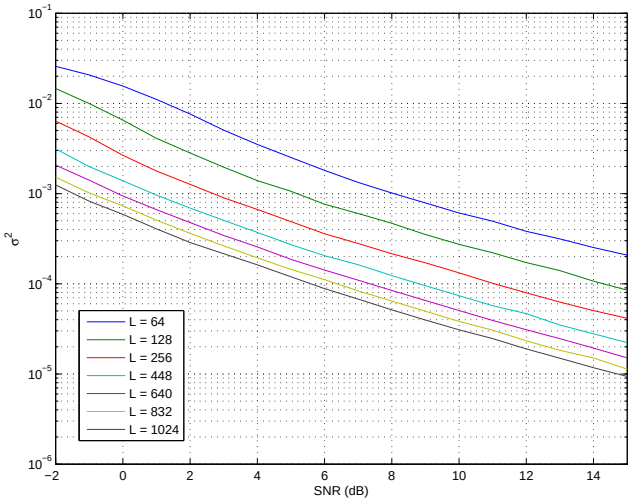


Figure 6.1.1.: Variance of the timing estimator for different SNR values and window lengths L

In figures 6.1.2 - 6.1.3 the variance of the timing estimator is compared with the modified Cramér-Rao lower bound (MCRLB) and with the lower bound for the Meyr and Oerder algorithm (MOLB) presented in section 3.1.1 for different window lengths L.

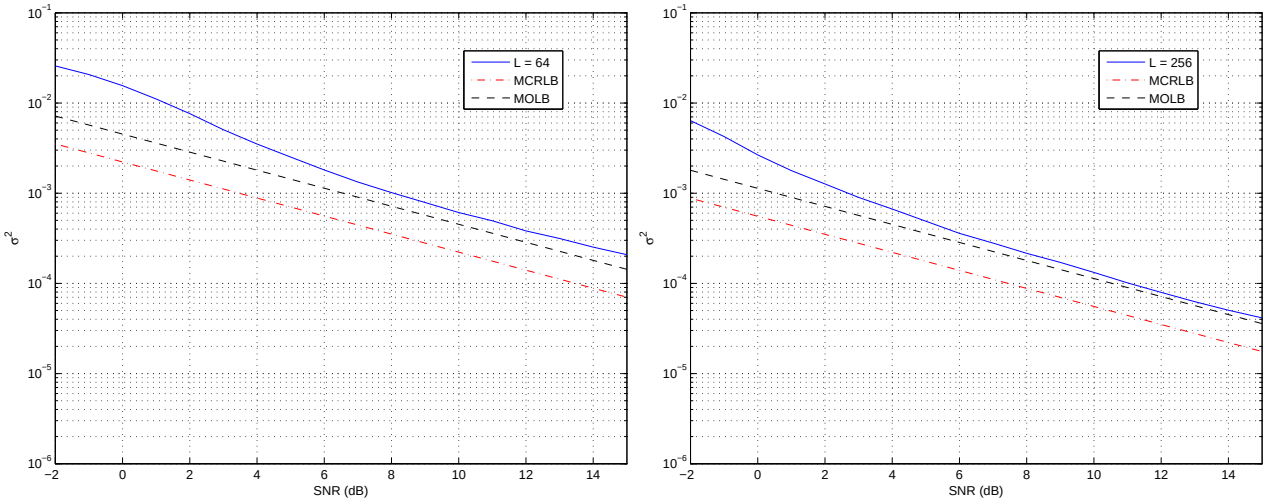


Figure 6.1.2.: Variance of the timing estimator compared to the lower bounds for window lengths $L = 64$ (left plot) and $L = 256$ (right plot)

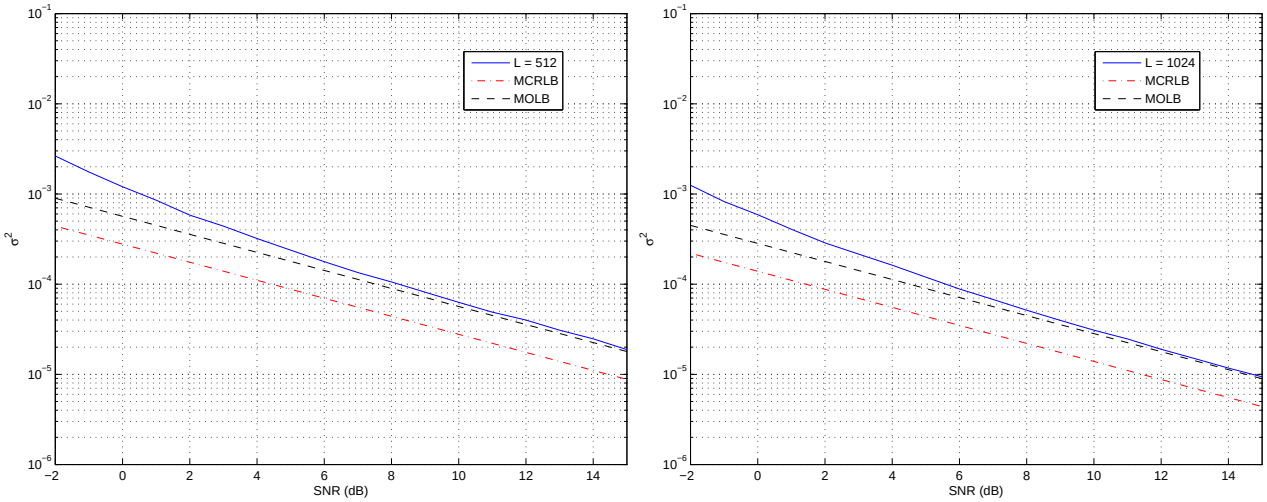


Figure 6.1.3.: Variance of the timing estimator compared to the lower bounds for window lengths $L = 512$ (left plot) and $L = 1024$ (right plot)

When looking at the figures 6.1.2 - 6.1.3, it can be seen that for higher window lengths L the variance curve converges towards the lower bound for the Meyr and Oerder algorithm.

6.2. FIR Interpolator

The most important parameters of the interpolator module and its impact on the output signal are discussed in this section.

6.2.1. Influence of Tap Resolution on a Single Carrier Signal

When implementing the interpolator module in real hardware, the resolution of the sinc filter samples is important. When more bits are used for the number representation, the quantization noise gets lower. But also the computational effort is increasing. Additionally, more bits for the filter coefficients results in larger filter tables and the multiplications and additions of the filter operations are getting more complex.
For illustrating the influence of the filter tap resolution of the stored filter function, the spectrum of the interpolator output is shown in a couple of figures. The quantization of the filter samples leads to additional quantization noise that affects the signal. By sending a complex valued sinusoidal signal to the interpolator and calculating the spectrum of the output of a decimation by $\rho = 0.984$ when using $N_z = 4$ zero crossings, the additional noise due to the quantization can be visualized. By using this decimation factor, the spectral component of the input signal can be visualized by one spectral component of the DFT after the decimation. Additionally, it is one possible use case for the satellite emulation application. In the simulations the binary two's complement is used for the representation of the filter coefficients, just like in the hardware implementation.
The spectrum of the input signal for this simulation can be seen in figure 6.2.1.

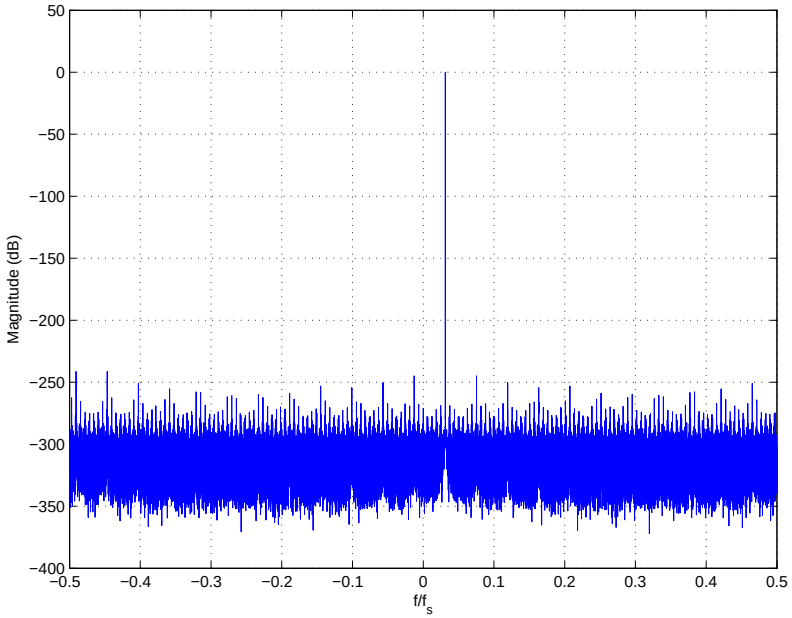


Figure 6.2.1.: Spectrum of the complex sinusoidal input signal

Figure 6.2.2 shows the spectrum of the output signal when floating point values are used for the filter samples. The highest spectral component is the complex valued sinusoidal input signal. For comparison reasons, the SNR is estimated for the output signal. Therefore the ratio of the power of the spectral component and the sum of the other spectral components is calculated. This SNR is calculated by:

$$SNR = 10 \cdot \log_{10} \left(\frac{X_{k_{Signal}}}{\sum_{k=0, k \neq k_{Signal}}^{N_{FFT}-1} X_k} \right)$$

In the floating point case, the SNR value is $SNR_{float} = 84.4\,dB$. This value can be seen as the maximal possible value for the following simulations.

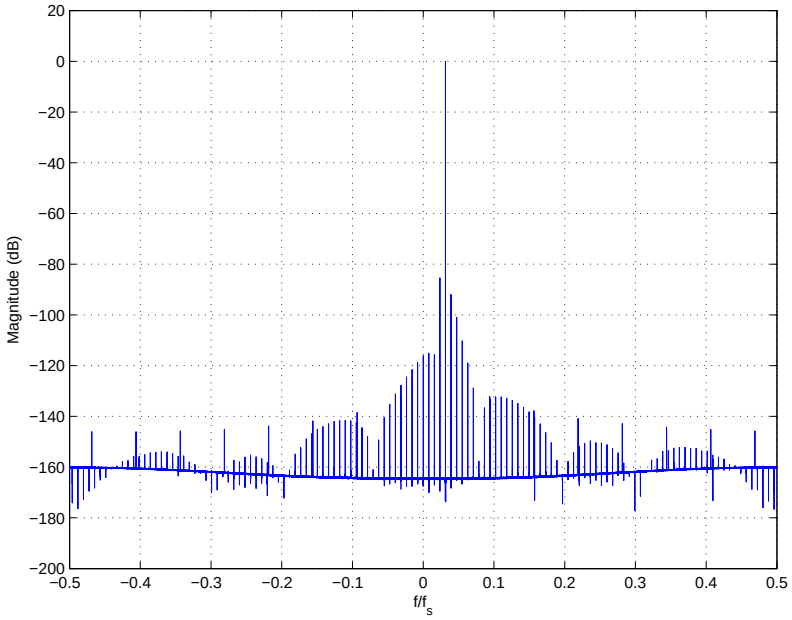


Figure 6.2.2.: Spectrum of the interpolator output signal using floating point values for the filter tap resolution

For the hardware implementation we don't want to use the floating point case due to its high complexity. Therefore we need to quantize these filter coefficients to a certain bit width. Figure 6.2.3 shows the output spectrum of the filter when only 4 bit are used for the filter tap representation. We can see that the additional noise due to this quantization is extremely high. In fact, the estimated SNR is only $SNR_{4bit} = 13.5\,dB$. This low value is mostly undesired in real applications.

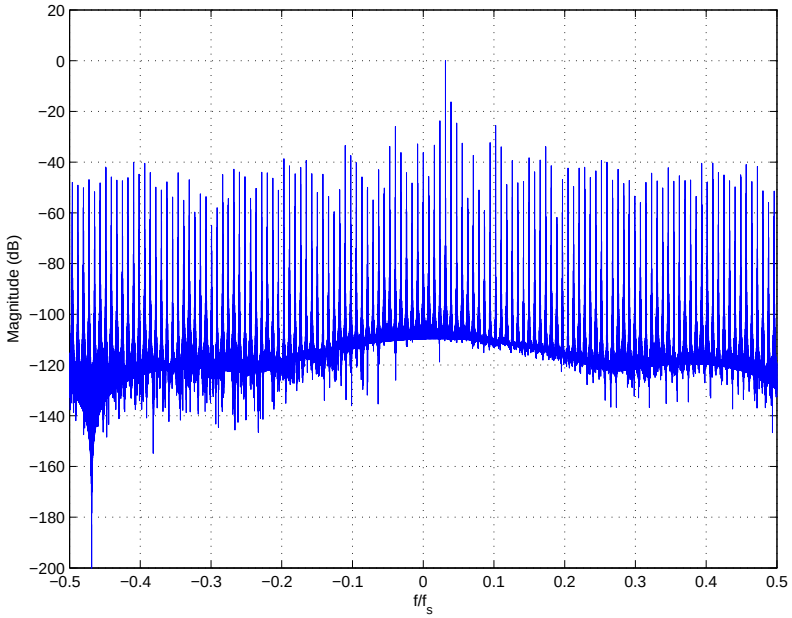


Figure 6.2.3.: Spectrum of the interpolator output signal using an amplitude resolution of 4 bit

When using 8 bit for the filter tap resolution, the SNR increases considerably. Now the SNR value is about $SNR_{8bit} = 44.8\,dB$. Figure 6.2.4 show the corresponding output spectrum of the interpolator.

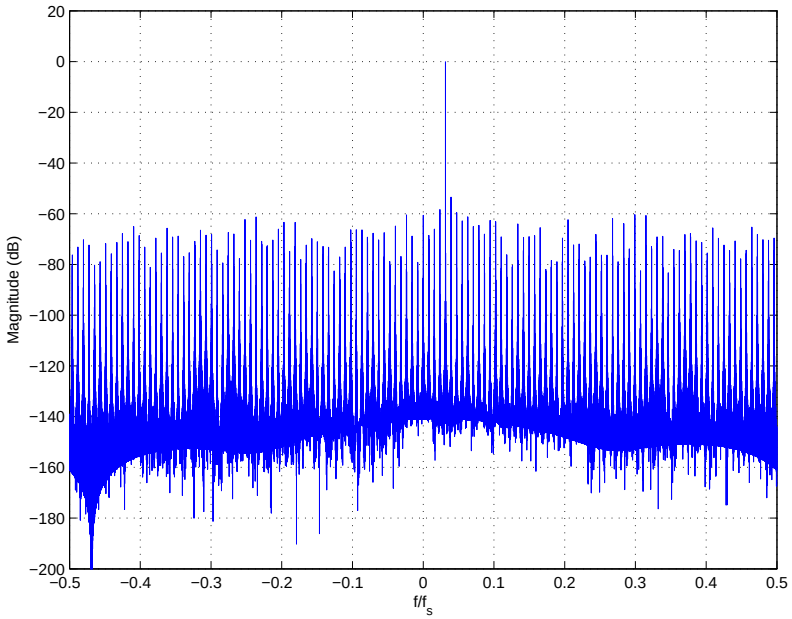


Figure 6.2.4.: Spectrum of the interpolator output signal using an amplitude resolution of 8 bit

When increasing the filter tap resolution again, the SNR also increases due to the lower quantization noise. This can be seen in figure 6.2.5, where 12 bit are used for the filter taps representation and a SNR of $SNR_{12bit} = 69\,dB$ is estimated.

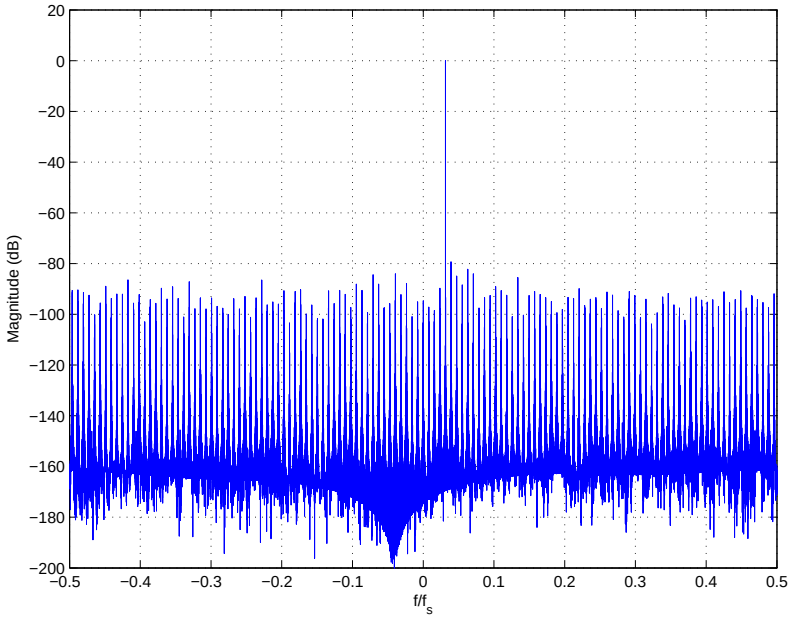


Figure 6.2.5.: Spectrum of the interpolator output signal using an amplitude resolution of 12 bit

In the hardware implementation 16 bit are used for the binary representation of the filter coefficients. The SNR value of this quantization is almost as high as in the floating point case. Figure 6.2.6 shows the corresponding output spectrum and the estimated SNR value of $SNR_{16bit} = 84.1\,dB$.

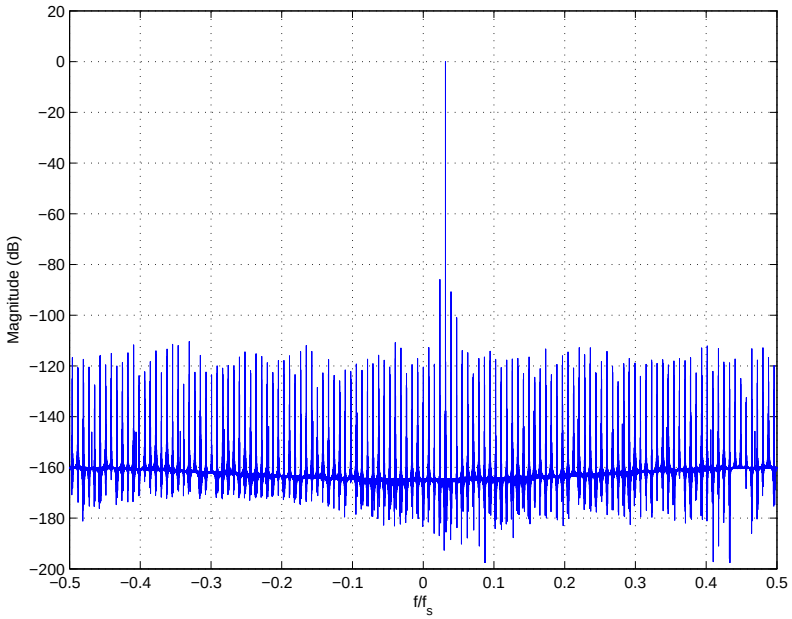


Figure 6.2.6.: Spectrum of the interpolator output signal using an amplitude resolution of 16 bit

So the used filter tap resolution of 16 bit is a good choice in terms of quantization noise, because it almost approaches the floating point case. The resulting hardware effort is acceptable since the 16 bit multiplication can be achieved by built in multiplication units. For

the filter table, which stores the filter taps, an internal ROM module is used where enough memory for the 16 bit taps is available.

6.2.2. Influence of Tap Resolution and the Number of Zero Crossings on a QPSK Signal

Another important parameter of the interpolation module is the number of zero crossings N_z which defines the number of filter taps. As mentioned in section 2.2.5, the number of zero crossing has a major influence on the suppression of the unwanted frequency components for avoiding aliasing after the resampling. In this section, a QPSK signal is generated and root-raised-cosine filtered. This input signal is then resampled only by slight decimation of $\rho = 0.992$, which is a possible scenario for the satellite emulator application.
The following figures visualize the output spectrum of the interpolation module with different filter tap resolutions as well as with different numbers of zero crossings N_z. Since the input signal is bandlimited and therefore no unwanted frequency components are present, the impact of the resulting filter and therefore the quantization noise and the number of filter taps is visible in the output spectrum.
If we choose such a decimation factor near 1, the cut-off frequency of the resulting filter is near $\frac{f_s}{2}$. When using only a small number of zero crossings, there is a slow decay of the magnitude response in the passband as it can be seen in figure 2.2.17 for low numbers of zero crossings. The more zero crossings are used, the less passband attenuation occurs. Also the quantization noise gets higher, the lower the filter tap resolution gets. As it can be seen in figure 6.2.6 of the previous simulation, the quantization noise shows a wavelike spectrum. Since for low numbers of zero crossings a passband attenuation is given, the quantization noise in this band gets also suppressed. Therefore the noise level is lower for low numbers of zero crossings in this simulation.
The spectrum of the according QPSK and root-raised-cosine input signal can be seen in figure 6.2.7.

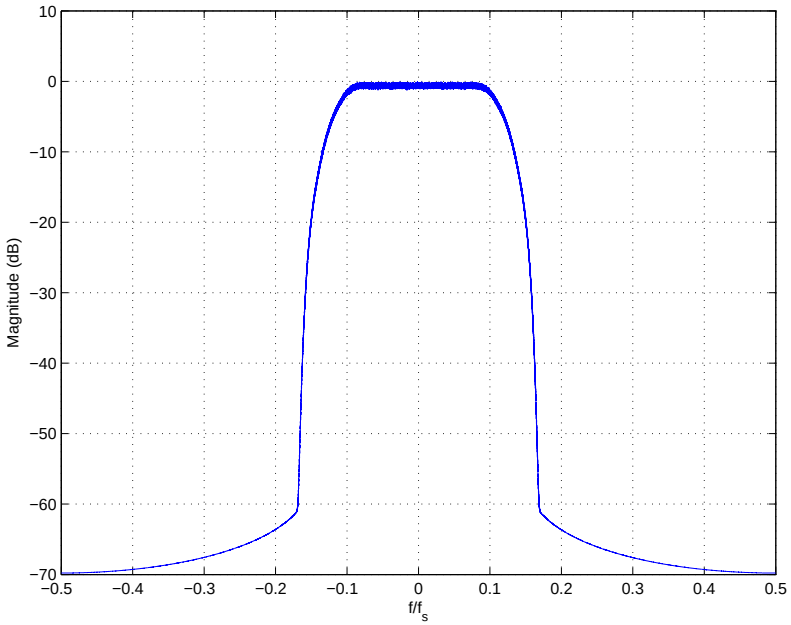


Figure 6.2.7.: Spectrum of the QPSK input signal

If we have a look at the comparison of three and 11 zero crossing for the 8 bit resolution case in figure 6.2.8, we can see that the noise ripples are visible in the spectrum. In the case of the lower number of zero crossings, the spectrum apart from the QPSK band gets decayed smoothly due to the passband attenuation and therefore the noise level is lower than in the $N_z = 11$ case.

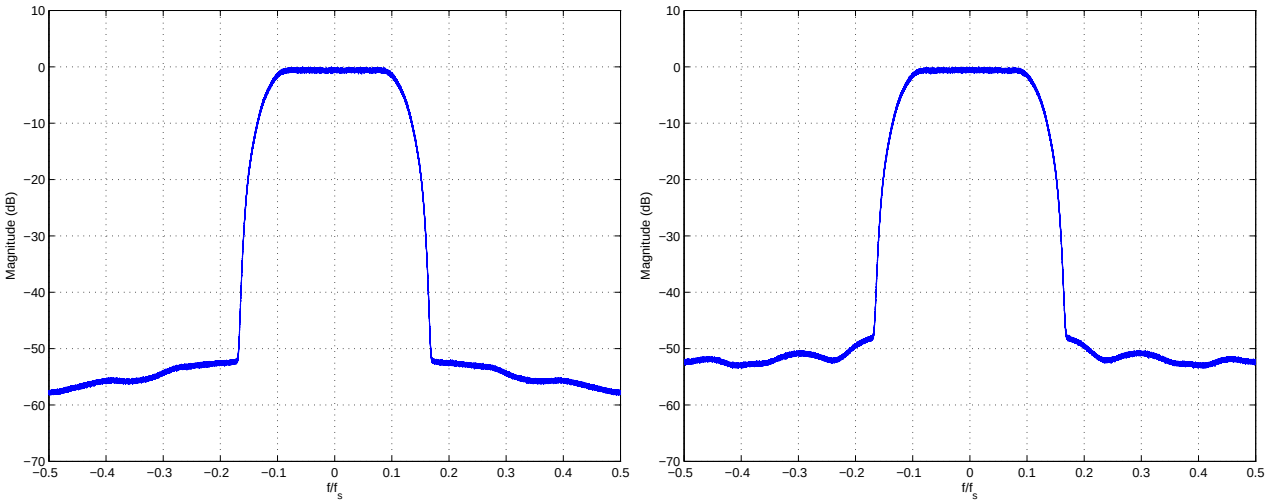


Figure 6.2.8.: Spectrum of the interpolator output signal using a filter tap resolution of 8 bit and three zero crossings for the left plot and 11 zero crossing for the right plot

Finally the output spectrum of the 16 bit case in shown in figure 6.2.9. Just like in the previous simulation case with a single sinusoidal input signal, the high filter tap resolution leads to a low quantization noise. When comparing the $N_z = 3$ and $N_z = 11$ plots, the noise ripples are no longer visible due to the lower quantization noise. Additionally we can see a higher noise floor due to the lower number of filter taps in the $N_z = 3$ case.

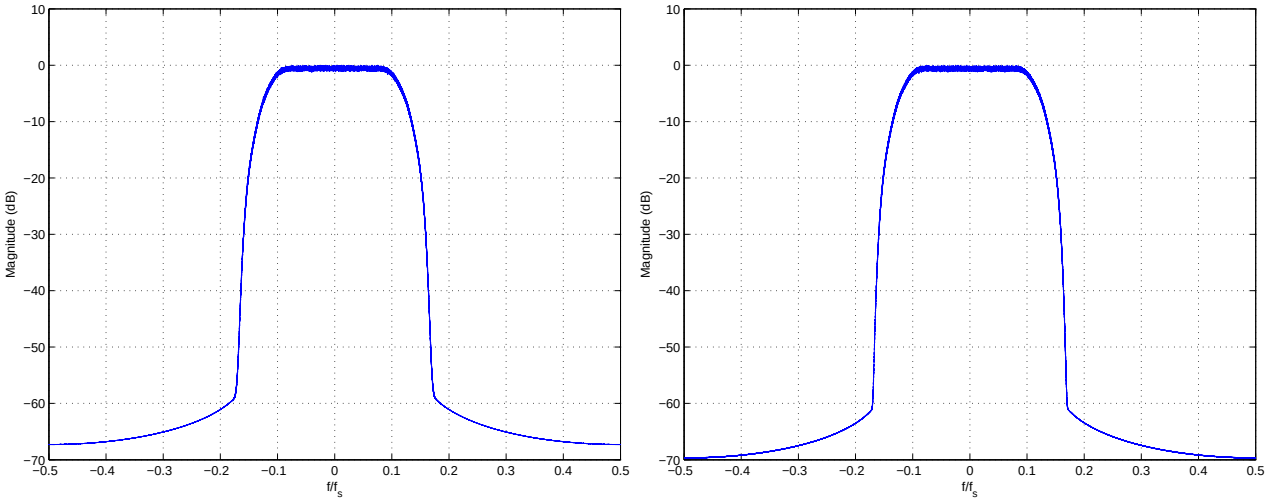


Figure 6.2.9.: Spectrum of the interpolator output signal using a filter tap resolution of 16 bit and three zero crossings for the left plot and 11 zero crossing for the right plot

When the application gets more complex and unwanted frequency components are present in the input signal, the higher number of zero crossings leads to a better suppression of

these frequencies. Also the hardware effort needs to be taken into account. The more zero crossings N_z are used, the more filter taps need to be stored in the filter table. As well as the higher complexity of filtering for more filter taps due to more multiplications and additions for producing one output sample.

6.2.3. Influence of the Number of Zero Crossings for the Decimation

When the interpolator is used for decimation like in the timing recovery application, the cut-off frequency and the steepness of the resulting filter is important. According to section 2.2.4, the cut-off frequency is set to $f_c = \frac{f_{s_{in}}}{2} \cdot \rho$ in the case of decimation. In this simulation, the rate conversion is set to $\rho = \frac{1}{2}$ which results in a cut-off frequency of $f_c = \frac{f_{s_{in}}}{2} \cdot \frac{1}{2} = \frac{f_{s_{in}}}{4}$. Since the new sampling rate equals $f_{s_{new}} = f_{s_{in}} \cdot \rho = \frac{f_{s_{in}}}{2}$, the cut-off frequency is at the borders of the illustrated output spectra. Figure 6.2.10 shows the input spectrum of the interpolator, where a single complex valued sinusoidal signal plus a SNR of $10\,dB$ is used.

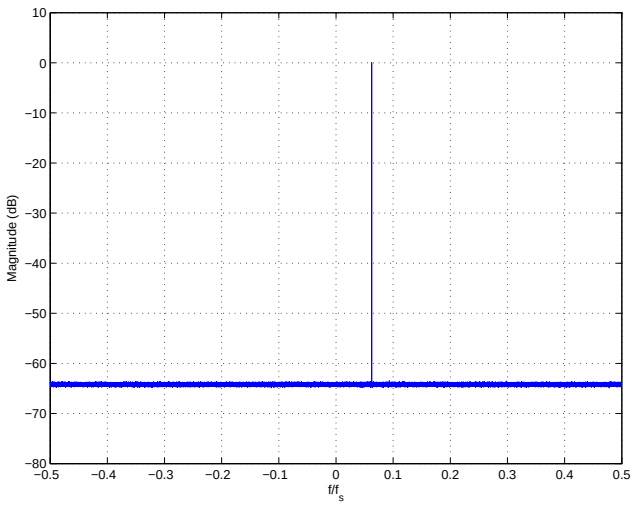


Figure 6.2.10.: Spectrum of the sinusoidal input signal for the decimation process affected by noise

This input signal is now decimated by 2, which results in a re-scaling of the frequency axis for the output spectrum. This can be seen by the shift of the input signal frequency component in the spectrum. In figures 6.2.11 - 6.2.13 the resulting output spectra are visible. Since the number of zero crossings and therefore the number of filter taps influences the steepness of the magnitude response (see figure 2.2.18), this effect can be observed in the output spectra. It can be seen that a higher number of zero crossings results in a lower passband attenuation as well as in a steeper decay of the noise at the spectral borders. Hence a higher number of zero crossings is desired when using the interpolator in the decimation application, since the usable frequency band is higher. We can see that for $N_z = 1$ in figure 6.2.11 the attenuation at $f = 0.2 \cdot f_s$ is already about $1\,dB$. When using for example $N_z = 11$ zero crossings like shown in figure 6.2.13, this point has moved to approximately $f = 0.45 \cdot f_s$ which expands the usable frequency band considerably.

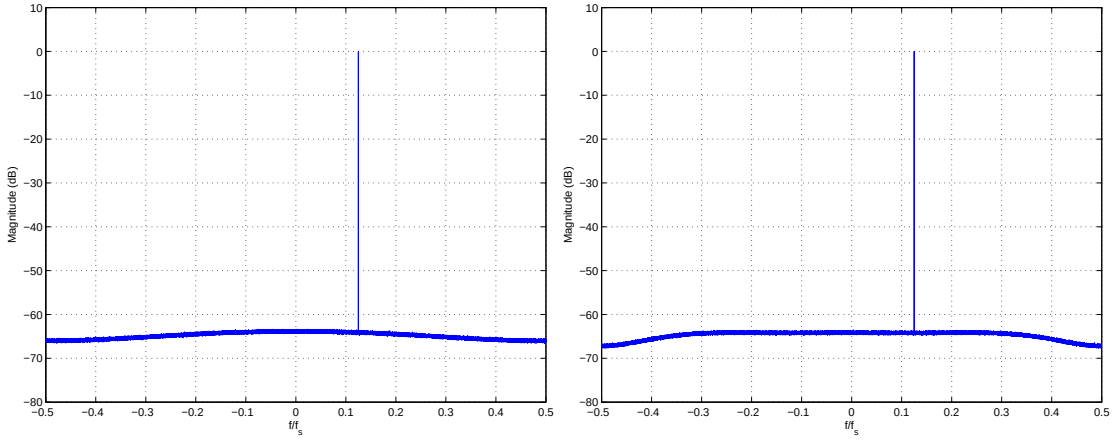


Figure 6.2.11.: Output spectrum of the decimator using one zero crossing (left plot) and three zero crossings (right plot)

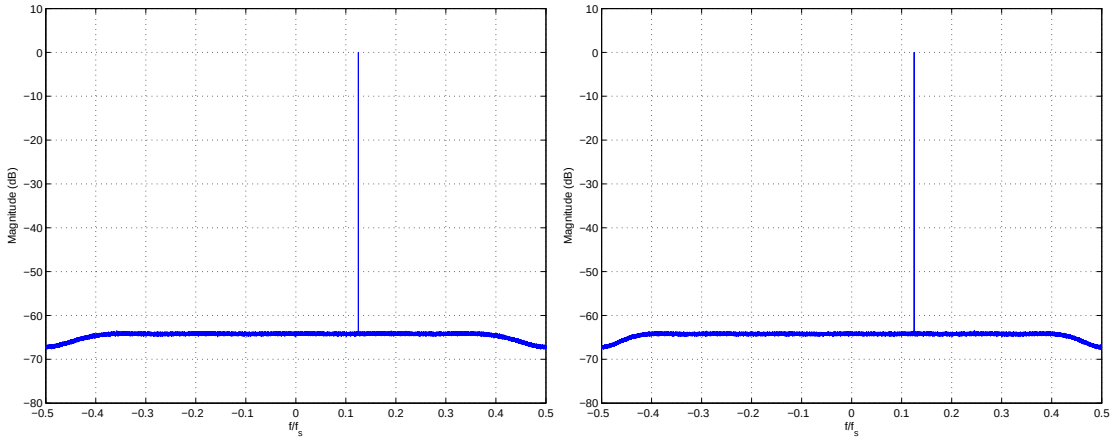


Figure 6.2.12.: Output spectrum of the decimator using five zero crossings (left plot) and seven zero crossings (right plot)

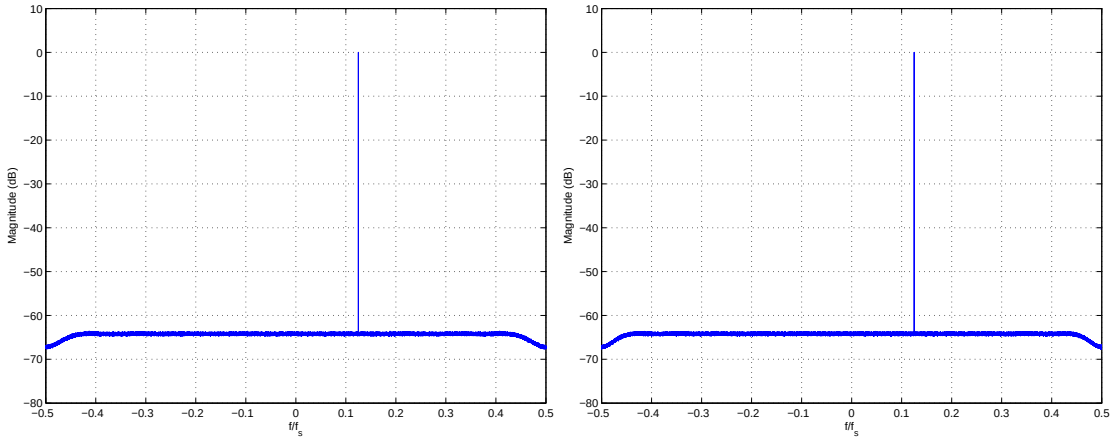


Figure 6.2.13.: Output spectrum of the decimator using nine zero crossings (left plot) and 11 zero crossings (right plot)

If we simulate the same decimation procedure with the cubic interpolator, we can see that there is no attenuation at the spectral borders of the output signal. Since the magnitude response of the cubic interpolator (see figure 2.1.1) shows no filtering in order to avoid aliasing for decimation, prefiltering would be necessary when using the cubic interpolator for decimation purpose.

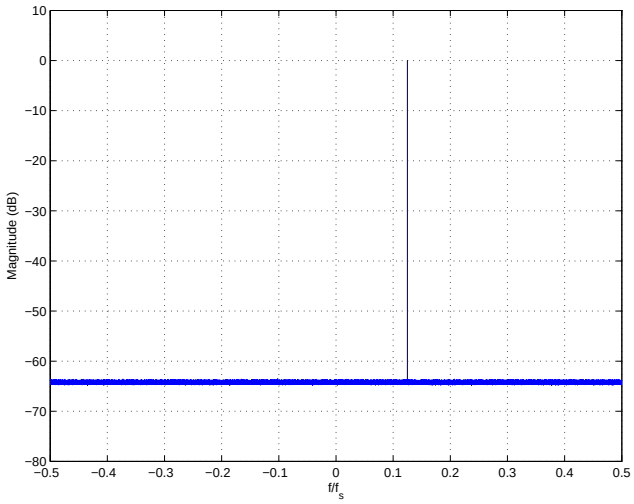


Figure 6.2.14.: Output spectrum of the cubic interpolator for decimation by 2

6.2.4. SFDR Analysis for the Filter Parameters

The spurious free dynamic range (SFDR) describes the ratio of the signal power to the strongest spurious frequency component. This measure is used to demonstrate the interpolator performance at a sampling rate conversion of $\rho = 0.99$. A single complex valued sinusoidal signal is used as input. This conversion factor is chosen such that it leads to a resulting spectral component that can be represented by the DFT without leakage.
The resulting output spectrum of the interpolator is used to measure the SFDR. Figure 6.2.15 shows such an example output spectrum for the SFDR calculation.

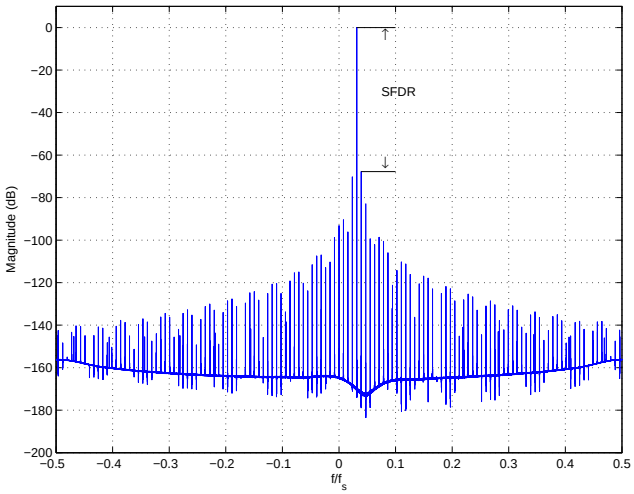


Figure 6.2.15.: Example of an interpolator output spectrum for the measurement of the SFDR

6.2.4.1. SFDR for Cubic Interpolator

The same sampling rate conversion of $\rho = 0.99$ is applied using the cubic interpolator. The resulting output spectrum can be seen in figure 6.2.16. The SFDR for the cubic interpolator is $SFDR_{cubic} = -44.38\,dB$. Since this result is quite low and the FIR interpolator can achieve much higher values which is shown in the next section, the cubic interpolator is not suited for the satellite emulator application.

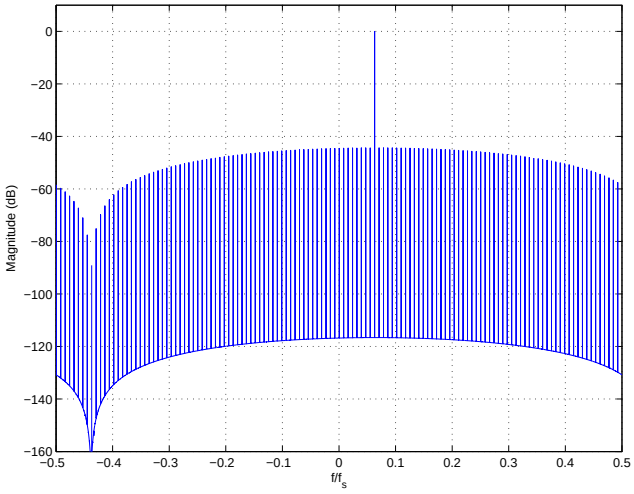


Figure 6.2.16.: Example of a cubic interpolator output spectrum for the measurement of the SFDR

6.2.4.2. SFDR as a Function of the Amplitude Resolution and the Numbers of Zero Crossings

In Figure 6.2.17 this SFDR is shown for different amplitude resolutions of the filter impulse response in dependency of the number of zero crossings N_z of the filter. It can be seen that the number of zero crossing has a major impact on the SFDR, but for $N_z = 9$ zero crossings, the result is already at its maximum value. Also the amplitude resolution affects the SFDR only up to 14 bit. This means that for an amplitude resolution of 14 bit and $N_z = 9$ zero crossing, the maximum SFDR can be reached in this simulation which is at $SFDR = -76\,dB$.

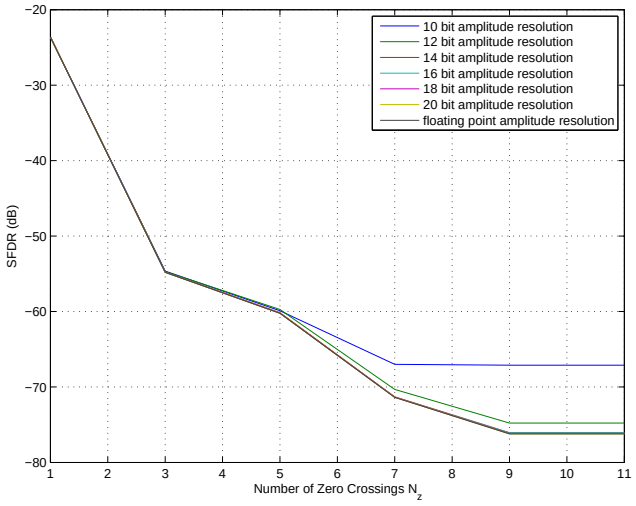


Figure 6.2.17.: SFDR for different amplitude resolutions in dependency of the number of zero crossings N_z

6.2.4.3. SFDR as a Function of the Number of Bits for Filter Resolution and Linear Resolution

The same simulation is used to measure the SFDR for different linear resolutions nb_η of the filter impulse response in dependency of the filter resolution nb_l in bit. As already explained in section 2.2.4, nb_l defines the number of filter taps per zero crossing. This means that the delay μ can be quantized with nb_l bit by the filter impulse response. Additionally the filter impulse response can be linearly interpolated by nb_η bit. This means that the delay μ can effectively be quantized with $nb_l + nb_\eta$ bit.
If we have a look at figure 6.2.18 where $N_z = 4$ zero crossings are used, we can see that the SFDR is highly influenced by nb_l, but only up to 7 bit. The linear interpolation resolution nb_η has only a minor influence on the SFDR in this simulation. The result shows that the maximum value of $SFDR = -71\,dB$ for $N_z = 4$ zero crossings can be achieved by using $nb_l = 7$ bit for the filter resolution.

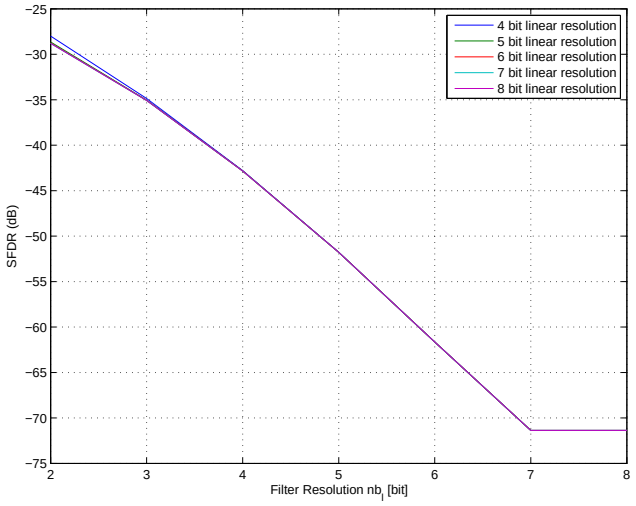


Figure 6.2.18.: SFDR for different linear resolutions nb_η in dependency of the filter resolution nb_l

6.2.4.4. SFDR as a Function of the Filter Resolution and the Number of Zero Crossings

From the two figures 6.2.17 and 6.2.18 we can see that the filter resolution and the number of zero crossings have the strongest influence on the SFDR. Therefore the same procedure is used to measure the SFDR of different numbers of zero crossings N_z in dependency to the filter resolution nb_l. This result is visualized in figure 6.2.19. Here we can see that the SFDR gets better for higher numbers of zero crossings up to $N_z = 9$, just like shown in figure 6.2.17. For this simulation the filter resolution nb_l affects the SFDR only up to 7 bit like in figure 6.2.18. This means that the best SFDR of $-76\,dB$ for this simulation can be achieved by using $N_z = 9$ zero crossing and a filter resolution of $nb_l = 7$ bit.

When comparing this result to the SFDR of the cubic interpolator which is $SFDR_{cubic} = -44.38\,dB$, we can see that for $N_z = 3$ and $nb_l = 5$ the result of the FIR interpolator is already better than for the cubic interpolator.

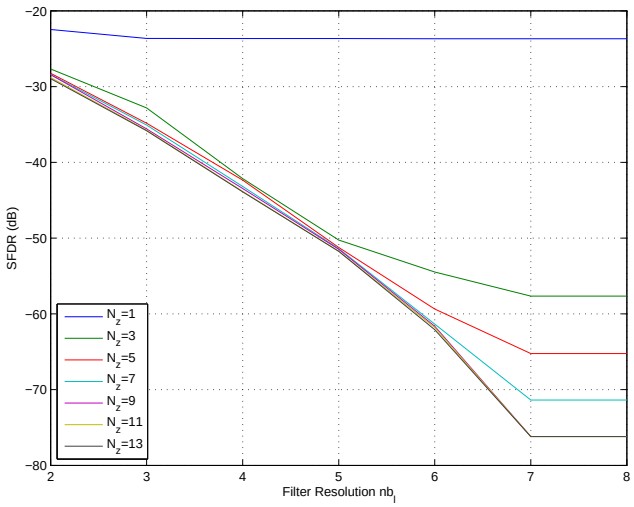


Figure 6.2.19.: SFDR for different numbers of zero crossings N_z in dependency of the filter resolution nb_l

6.2.5. Symbol Error Rate

Another useful measurement for the performance of the interpolator and timing estimator is the bit error rate that results from the timing recovery process. Therefore a four times oversampled QPSK signal is generated on which timing recovery is performed using the interpolator and timing estimator implemented in this thesis.

For the first plot in figure 6.2.20, $L = 64$ and different number of zero crossings N_z are used to show the influence on the symbol error rate. As it can be observed, the number of zero crossings has no major influence on the symbol error rate. Only for $\frac{E_s}{N_0} = 15\,dB$ a small deviation is visible, where the higher number of zero crossings shows a better result.

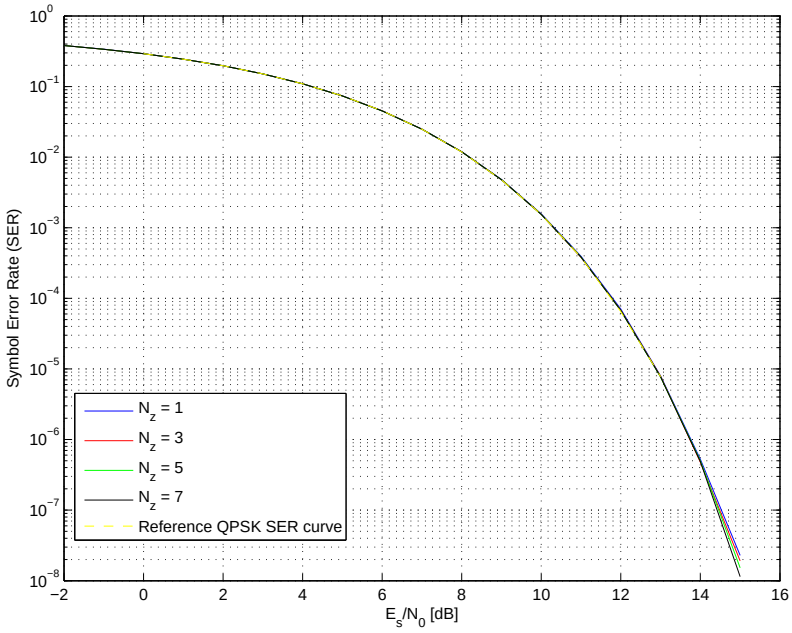


Figure 6.2.20.: Symbol error rates of a QPSK signal using the implemented timing recovery for different numbers of zero crossings N_z

In figure 6.2.21, the timing recovery is performed for different window lengths L of the timing estimator. In this simulation, the window length L has no major influence on the symbol error rate. Again, only for $\frac{E_s}{N_0} = 15\,dB$ the curves deviate due to the different window lengths L of the timing estimator. It can be observed that for a higher window length L, the symbol error rate improves for high $\frac{E_s}{N_0}$ values due to the better variance of the timing estimator.

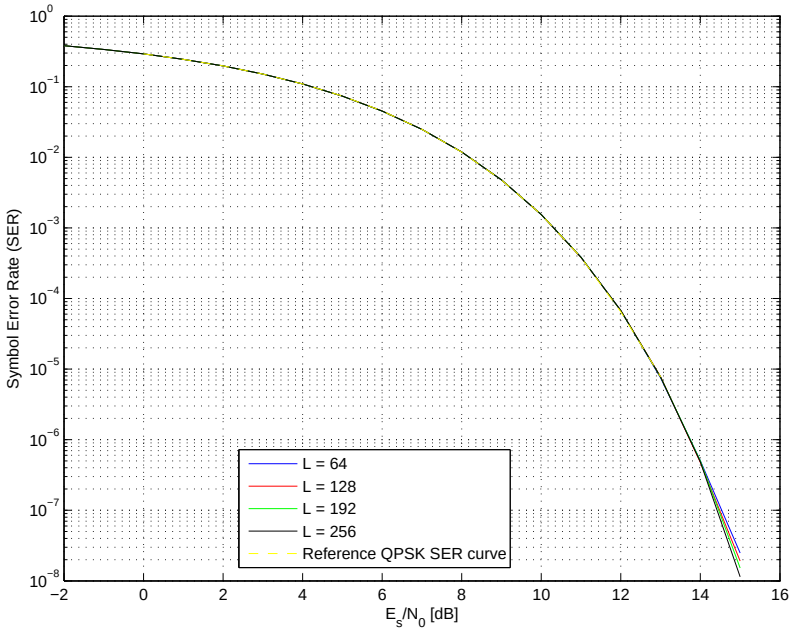


Figure 6.2.21.: Symbol error rates of a QPSK signal using the implemented timing recovery for different window sizes L of the timing estimator and $N_z = 4$

6.2.6. Folding of Spectral Images when using Cubic Interpolator for Decimation

The high sidelobes in the continuous magnitude response of the cubic interpolator (see figure 2.1.1 and [EGH93]) results in folding images according to [Gar93]. So if the decimation rate is chosen badly (decimation rate is a non-rational number), this impairment effect is visible in the output spectrum of the cubic interpolator.

This simulation shows this effect and uses a root-raised-cosine filtered QPSK signal which is eight times oversampled as the input signal. This input spectrum can be seen in figure 6.2.22. In figure 6.2.23 the output spectra of the cubic interpolator and the FIR interpolator are illustrated with a sampling rate conversion factor of $\rho = \frac{1}{2}$, which equals a decimation rate of $\frac{1}{\rho} = 2$. Due to the choice of this conversion factor, the spectral images at the new sampling rate $f_s' = \rho \cdot f_s$ get highly suppressed by the magnitude response of the cubic interpolator and no impairments occur.

But if a sampling rate conversion of $\rho = 0.4706$ is chosen as an example (non-rational decimation factor), the spectral components at integer multiples of the new sampling rate $f_s' = \rho \cdot f_s$ are no longer at integer multiples of the original sampling rate f_s and therefore these spectral images fold onto the output signal of the cubic interpolator. This effect can be seen in figure 6.2.24, which also shows the output of the FIR interpolator where the spectral images get suppressed sufficiently.

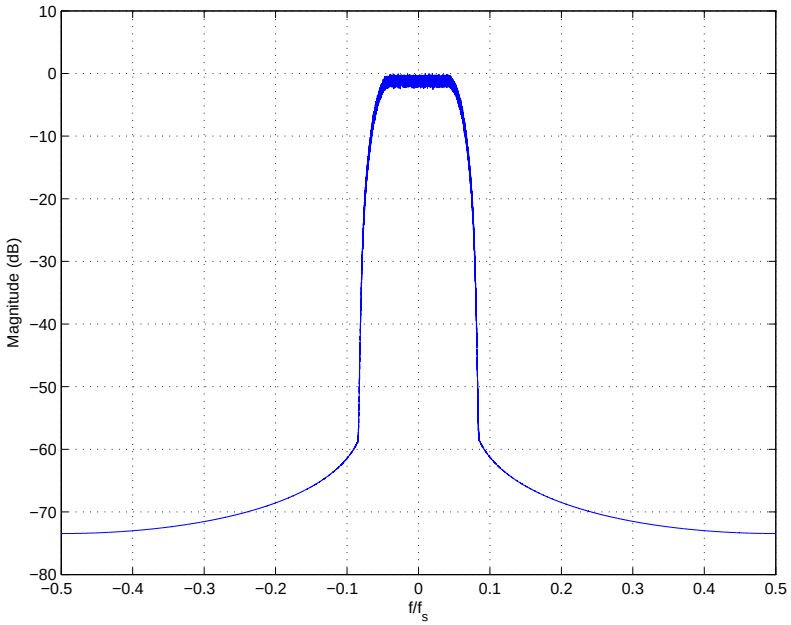


Figure 6.2.22.: Input spectrum for the aliasing illustration of the cubic interpolator

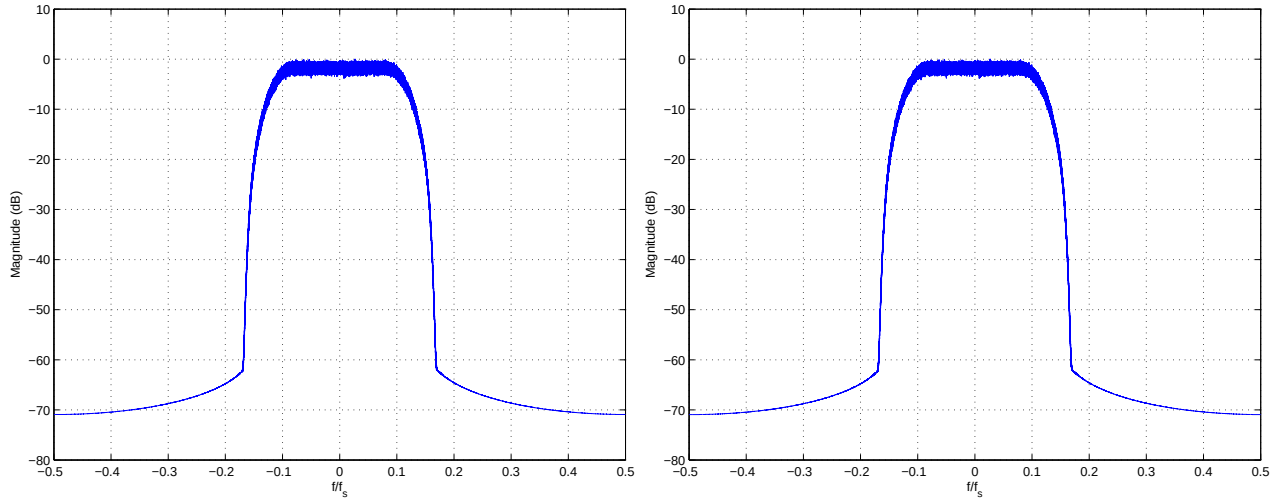


Figure 6.2.23.: Output spectrum of the cubic interpolator (left plot) and FIR interpolator (right plot) using a sampling conversion factor of $\rho = \frac{1}{2}$

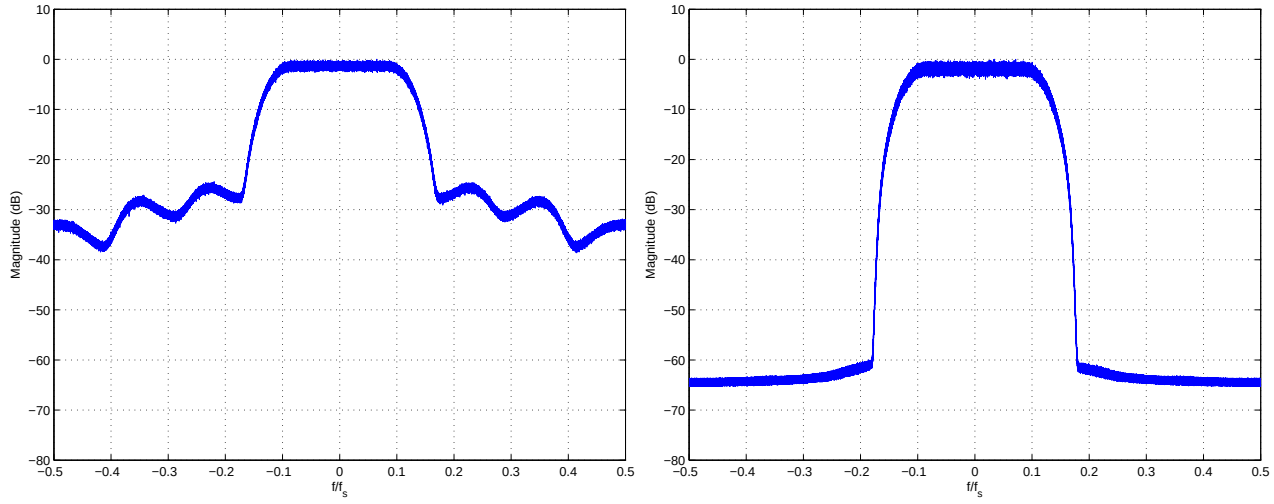


Figure 6.2.24.: Output spectrum of the cubic interpolator (left plot) and FIR interpolator (right plot) using a sampling conversion factor of $\rho = 0.4706$

7. Practical Measurements

7.1. Measurement Setup

For performing the measurements of the implemented hardware modules, the USRP N210 is integrated into a measurement setup. A block diagram of this setup is illustrated in figure 7.1.1. Since the implemented hardware is used in an environment for demodulating a DVB-S2 signal, the Newtec EL470 is used to generate this DVB-S2 signal. The payload is generated by an application on a PC which is also used for the configuration of the modulator (PC_TX). For the demodulation process another Newtec modem is integrated in the measurement setup. Additionally a second PC is necessary for configuration of the demodulator as well as for optional packet error rate calculations (PC_RX). The modulator generates a DVB-S2 signal from the payload provided by PC_TX using the following parameters:

- Center frequency $f_{center} = 1\,GHz$
- Symbol rate $f_{sym} = 1\,MBaud$
- Output level of $-20\,dBm$

This configuration can also be seen in figure 7.1.2 which shows the parameters of the modulator. Additionally the demodulator state is illustrated in figure 7.1.3.

A programmable noise generator is used to add AWGN to the signal. The input signal as well as the noise source can be attenuated here in order to achieve a desired SNR value. A power splitter is used to connect the noisy signal with the demodulator. Combined with the signal directly from the modulator, the PC_RX can now optionally be used to determine packet error rates.

Another power splitter is used to connect the signal with the Rohde&Schwarz signal analyzer. This analyzer is used to visualize the spectrum of the signal that is sent to the USRP N210 by Ettus Research. An example spectrum is shown in figure 7.1.4. This spectrum analyzer is used to measure the SNR of the signal for the measurements in this section. Due to inaccuracy in this measurement, the SNR values in this section should only be seen as guidance values. Another problem was the non flat spectrum of the programmable noise generator which results in an additional error.

The USRP N210 which contains the implemented hardware modules is connected via Ethernet to the host PC where GNU Radio is used for further signal processing. Also some debug signals are sent to a logic analyzer. Figure 7.1.5 shows a screenshot of this logic analyzer where the clock and the strobe signals of the implemented hardware modules are illustrated. In figure 7.1.6 the output waveform of the ADC on the USRP N210 is shown.

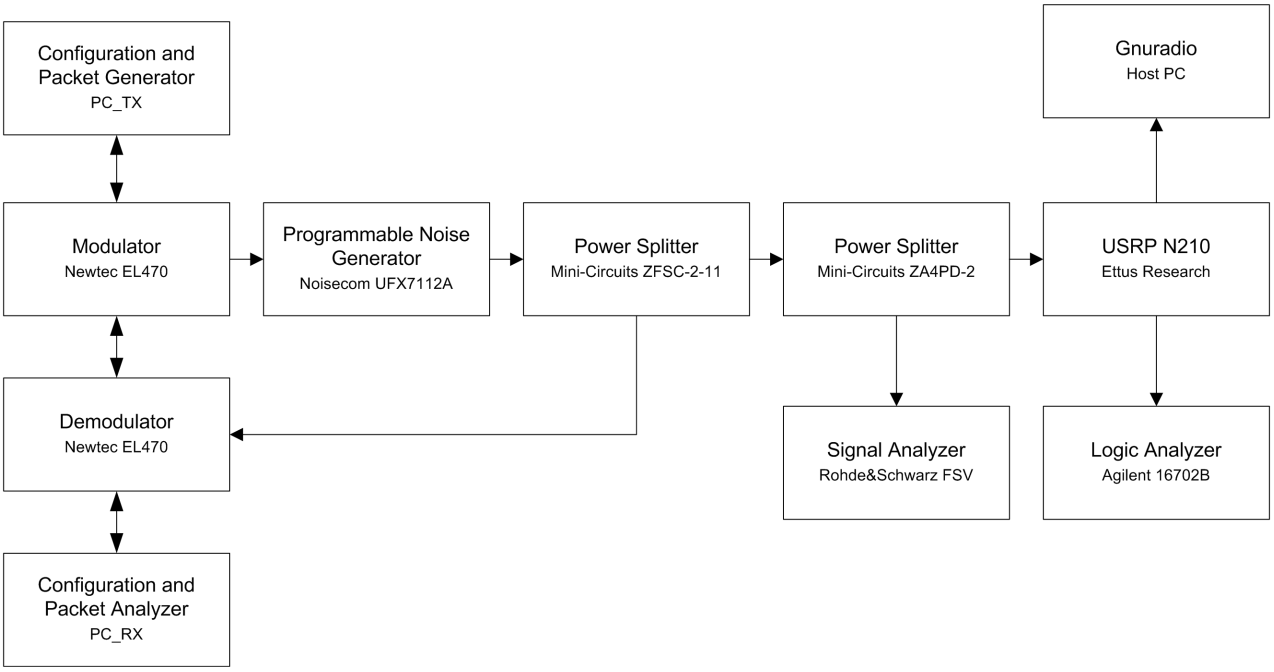


Figure 7.1.1.: Setup of the components for the measurements on the implemented modules in hardware

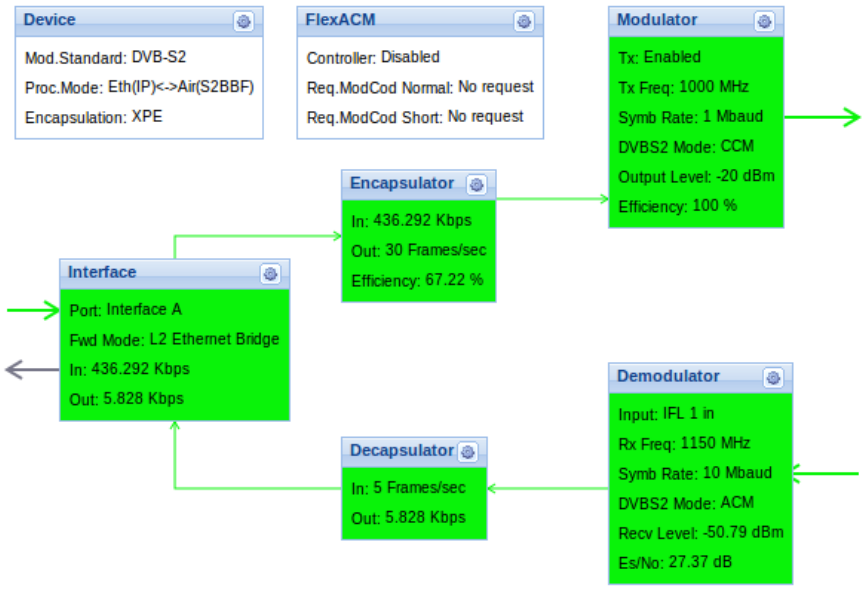


Figure 7.1.2.: Configuration overview of the modulator

State

State:	OK

Demod Lock:	Locked
Receive Level:	-47.81 dBm
Power spectral density:	-107.84 dBm/Hz
Es/No:	2.96 dB
Link Margin:	3.63 dB
Input:	IFL 1 in
Input Frequency:	1000 MHz
Symbol Rate:	1 Mbaud
Interface Rate:	1.452076 Mbps
DVB-S2 Mode:	ACM
LNB Power Supply 1:	Disabled
ODU Power Control 1:	Power off
LNB Power Supply 2:	Disabled
ODU Power Control 2:	Power off

Figure 7.1.3.: Demodulator state

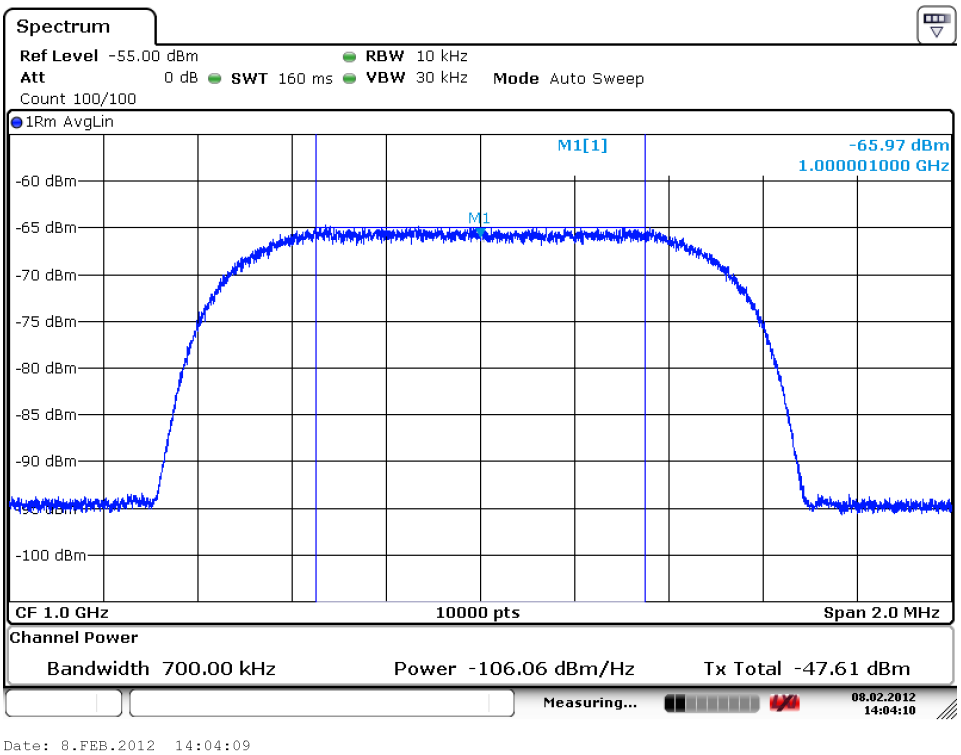


Figure 7.1.4.: Screenshot of the signal analyzer showing a spectrum of a $f_{sym} = 1\,MBaud$ DVB-S2 signal at a center frequency of $f_{center} = 1\,GHz$

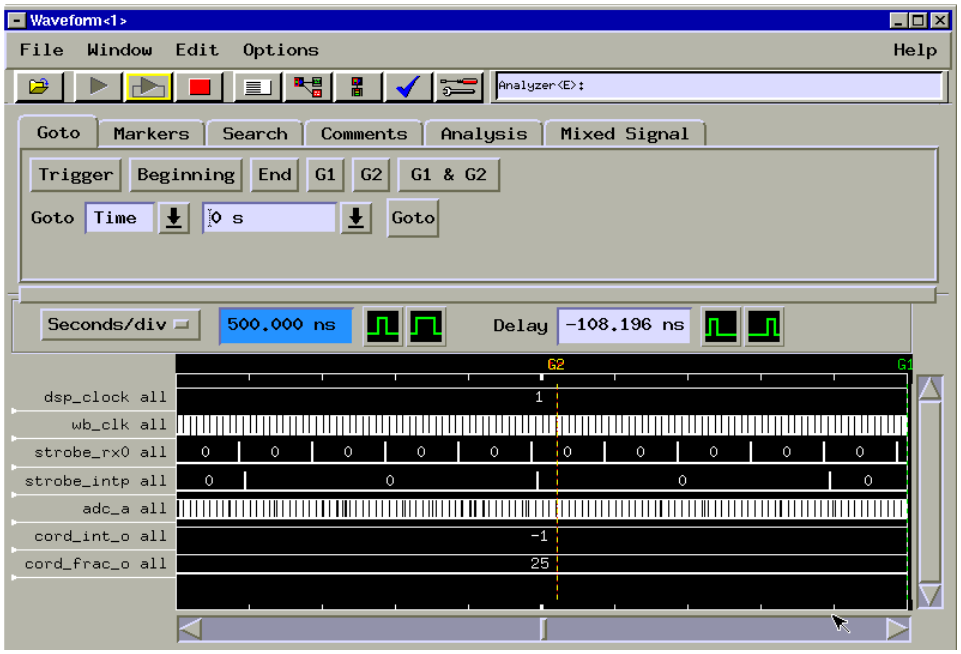


Figure 7.1.5.: Screenshot of the logic analyzer showing clock and strobe signals of the implemented hardware modules on the USRP N210

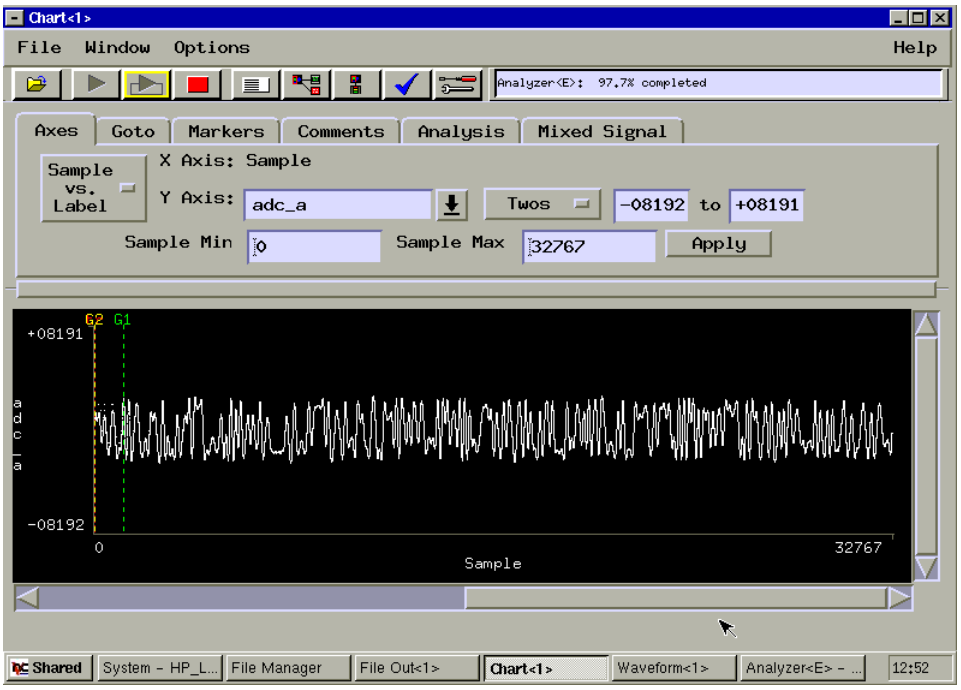


Figure 7.1.6.: Screenshot of the logic analyzer showing the output of the ADC on the USRP N210

7.1.1. Measurement Points

In order to calculate the signal-to-noise ratio, the power in the signal band is measured first without and then with the actual carrier. By subtracting these two values which are given in dB, we get the signal-plus-noise-over-noise ratio $x_{dB} = 10 \cdot \log_{10}\left(\frac{S+N}{N}\right)$. The actual SNR, which is $SNR_{dB} = 10 \cdot \log_{10}\left(\frac{S}{N}\right)$, can now be calculated using the following formula

$$SNR_{dB} = 10 \cdot \log_{10}\left(10^{\frac{x_{dB}}{10}} - 1\right) \tag{7.1.1}$$

For all the measurements points in this section the configuration of the programmable noise generator is shown in table 7.1.1.

$\left(\frac{S+N}{N}\right)_{dB}$	SNR_{dB}	$Noise\ Attenuation_{dB}$	$Signal\ Attenuation_{dB}$
45	45	OFF	0
29	29	20	0
15,1	15	6	0
10,4	10	6	5
7,8	7	2	4,3
6,2	5	2	6,1
5,5	4	2	7,3
4,7	3	2	8,3
4,1	2	2	9,1
3,5	1	2	10,2
3	0	2	11,2

Table 7.1.1.: Configuration of the programmable noise generator and the corresponding SNR values

7.2. Output Spectra of the USRP N210

For these measurements a special FPGA image is generated, which implements the existing receiver structure of the USRP and also the root-raised-cosine filter used for Nyquist pulse shaping. So the image provides two output signals which can be used to illustrate the spectrum before and after the root-raised-cosine filter. The existing receiver structure can be seen in figure 5.1.2. Figure 7.2.1 shows the spectrum a DVB-S2 input signal with $SNR = 5\,dB$ before and after the root-raised-cosine filter.

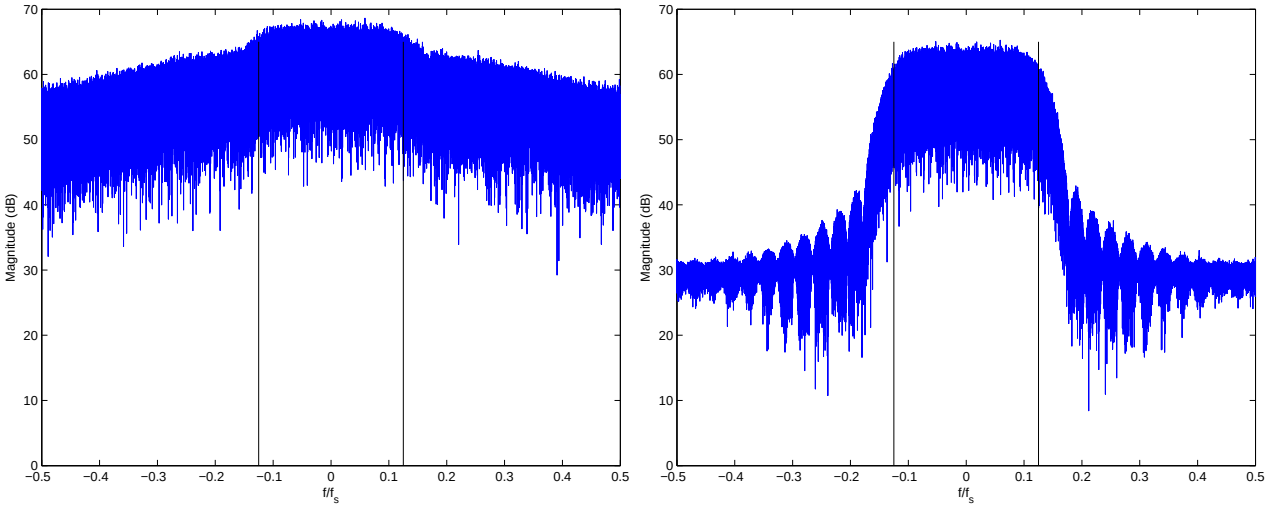


Figure 7.2.1.: Spectrum of the received QPSK signal before (left) and after (right) the root-raised-cosine filter at a signal-to-noise ratio of $SNR = 5\,dB$

When looking at figure 7.2.2 where only noise is sent to the USRP, the left plot shows the influence of the CIC filter which is used for the decimation process. Since the input sampling frequency is $f_{s_{in}} = 100\,MHz$ and the output frequency is $f_{s_{out}} = 4\,MHz$ the decimation factor is $\frac{f_{s_{in}}}{f_{s_{out}}} = 25$ which is implemented only by the CIC filter where the half-band filters are bypassed. The magnitude response of the four stage CIC filter with a decimation of 25 is already shown in figure 5.1.5. At the indicated cut-off frequency of the root-raised-cosine filter the attenuation is approximately $1\,dB$ which also corresponds to the observation in figure 5.1.5.

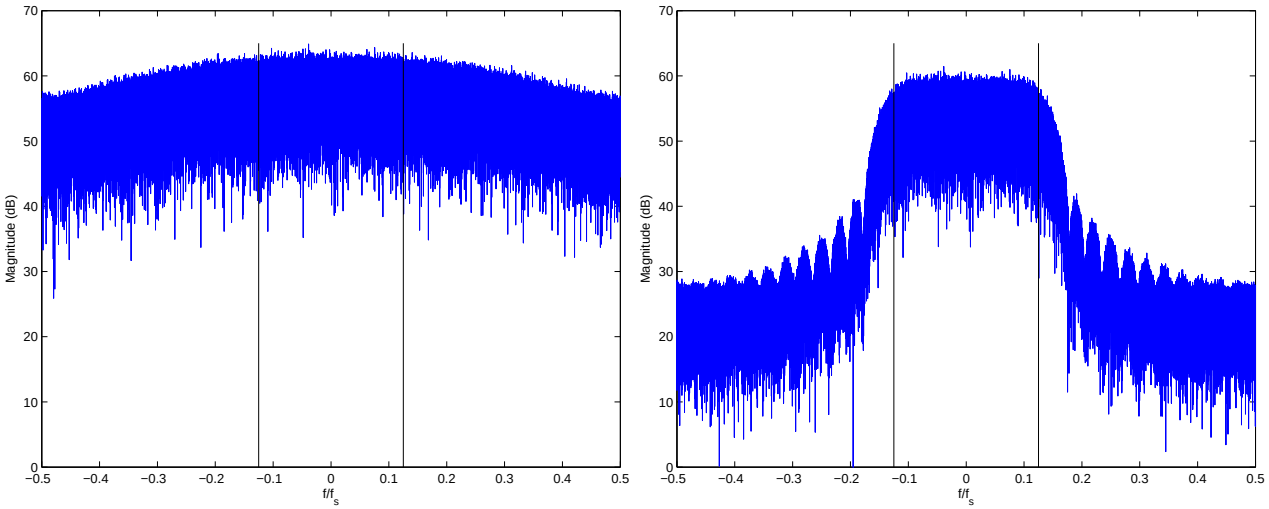


Figure 7.2.2.: Spectrum of a noise-only-signal before (left) and after (right) the root-raised-cosine filter

7.3. Timing Estimator

The variance of the timing estimator for a window size of $L = 64$ is already simulated in section 6.1.1. Now these simulation results are compared with the actual variance of the timing estimator implemented in hardware. Additionally, a bit-accurate simulation is performed. These variances are illustrated in figure 7.3.1 where also the lower bounds are presented. When looking at this figure, we can see that the performance of the implemented estimator in hardware is similar to the simulation results. Since the variance of the hardware module is only measured once for some SNR values, the curve is just a qualitative information of the performance. The variance curve of the measured hardware module shows a self-noise floor for higher SNR values just as expected.

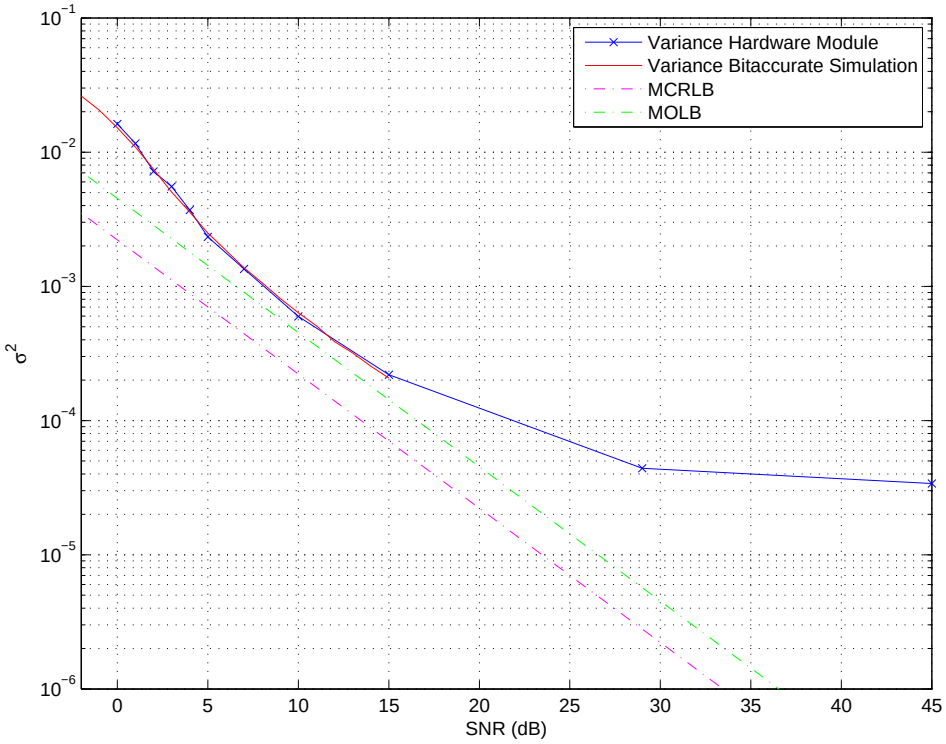


Figure 7.3.1.: Timing estimator variance of the hardware module and the simulation

7.3.1. Influence of the Clock Mismatch

Due to the mismatch of the internal clock of the USRP N210, the symbol rate of the modem and the symbol rate of the USRP N210 differ. Hence $T_{s_{TX}} \neq T_{s_{RX}}$ which results in a time varying optimal sampling point. As a result, the estimated value of the sampling time is also time varying. For the illustration of the influence of this clock mismatch this time drift is analyzed in this section.
The following measurement was done without any external clock reference but with the internal clock which mismatches the clock from the TX modem.
Figure 7.3.2 shows the averaged drift of the estimated timing value μ where T_L equals the time duration of one window L. Since the symbol rate in this measurement was set to $f_{sym} = 1\,MBaud$ and the window length is set to $L = 64$, we can calculate the window duration which results in $T_L = \frac{L}{f_{sym}} = 64\,\mu s$. Hence the duration of the illustrated drift is about $18000 \cdot T_L = 1.15\,s$.

The variation of μ in this time is approximately $\triangle\mu = 90$ which can now be used to determine the speed of the drift. Since the sampling duration T_s equals a estimated time value of $\mu = 128$ due to the 7 bit resolution of T_s, the estimated value of μ drifts by approximately

$$\frac{\triangle\mu}{1.15\,s \cdot 128} = 0.61\,\frac{samples}{s}$$

which means that in

$$\frac{1.15\,s}{\triangle\mu} \cdot 128 = 1.63\,s \tag{7.3.1}$$

the estimated sampling time is drifted by one sample.

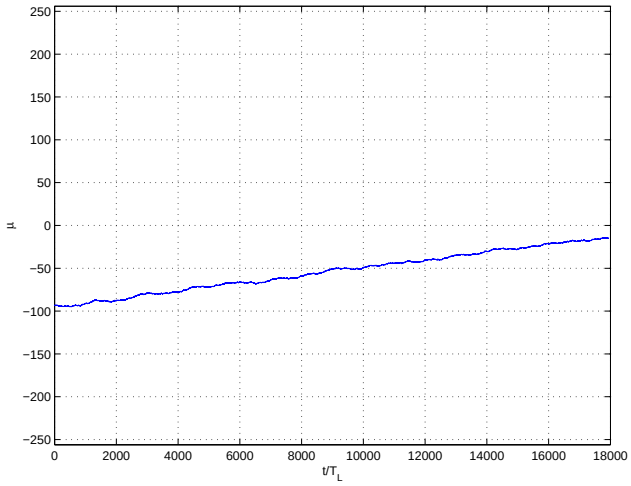


Figure 7.3.2.: Averaged drift of μ due to the clock mismatch

In order to avoid this clock mismatch, the USRP N210 can be connected to an external reference clock. This external clock ($10\,MHz$) is generated by the modem which generates the input signal. Figure 7.3.3 shows the averaged drift of the timing estimation value μ when the external reference clock is used. As it can be seen, no drift is occurring in this time duration of about $1.15\,s$.

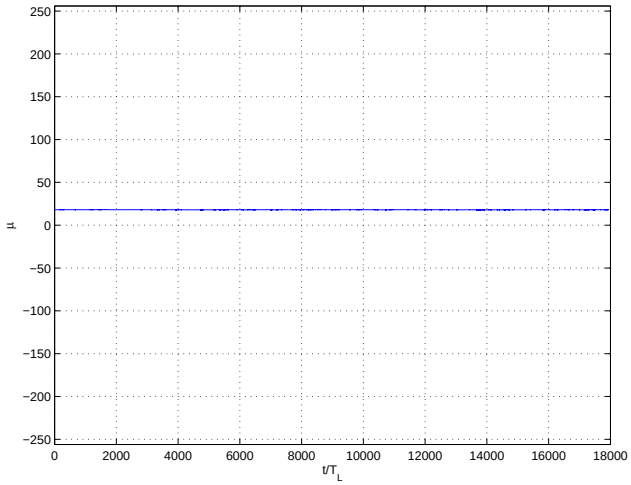


Figure 7.3.3.: Averaged drift of μ using the external reference clock reference

7.4. Output Symbols

The constellation diagrams for some specific SNR values are plotted in this section. Since no phase correction is applied to the signal, the constellation points are rotated by this phase error.

The symbols at the output of the hardware module are illustrated in the constellation diagram in figure 7.4.1 where an SNR of $29\,dB$ is applied. A QPSK signal is shown in the left plot and a 16-APSK signal is shown in the right plot. When using 16-APSK as modulation scheme, DVB-S2 sends additional pilot symbols with $\frac{\pi}{2}$-BPSK modulation scheme. Hence it looks like there is an additional QPSK constellation in the right plot. When setting this quite high SNR value, the certain constellation points can be seen clearly.

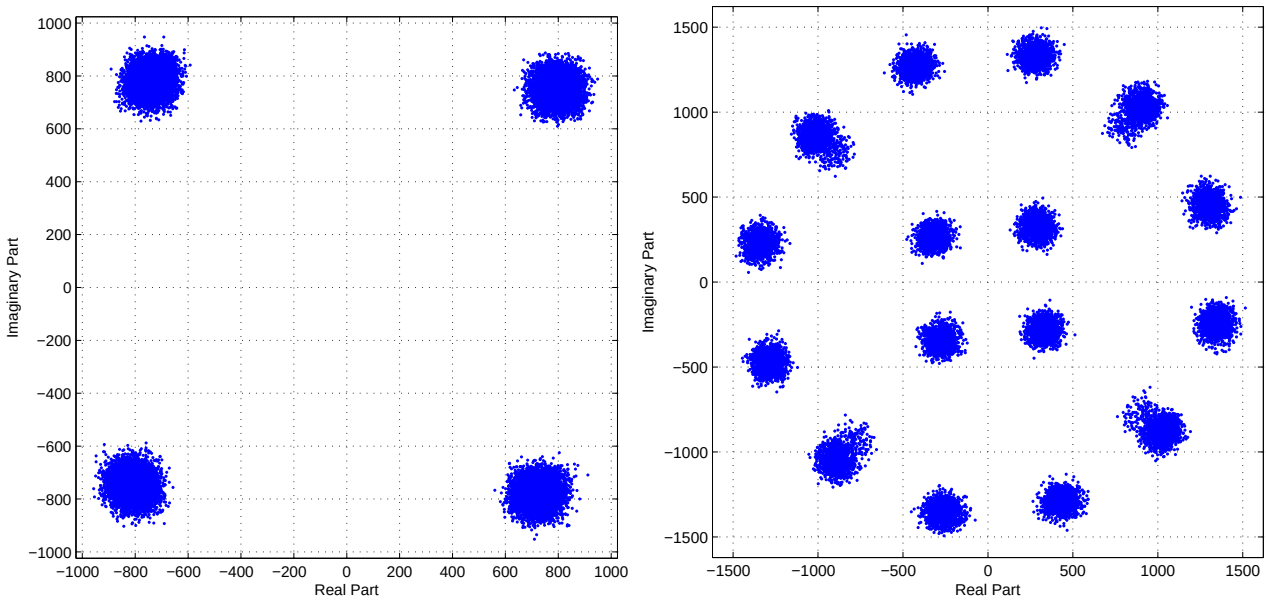


Figure 7.4.1.: Constellation diagram of the received QPSK (left) and 16-APSK (right) symbols with external clock reference and $SNR = 29\,dB$

When setting an SNR of $15\,dB$, the constellation points in figure 7.4.2 are still distinguishable whereas in figure 7.4.3 where a SNR of $10\,dB$ is applied, there are only the two rings of the 16-APSK signal visible.

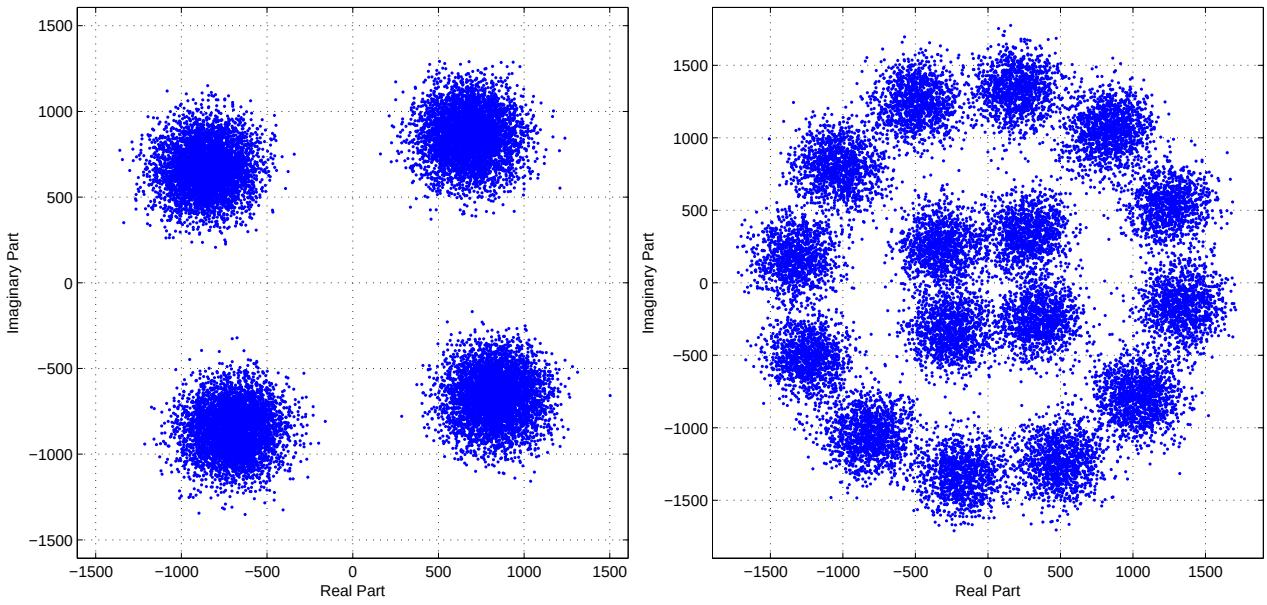


Figure 7.4.2.: Constellation diagram of the received QPSK (left) and 16-APSK (right) symbols with external clock reference and $SNR = 15\,dB$

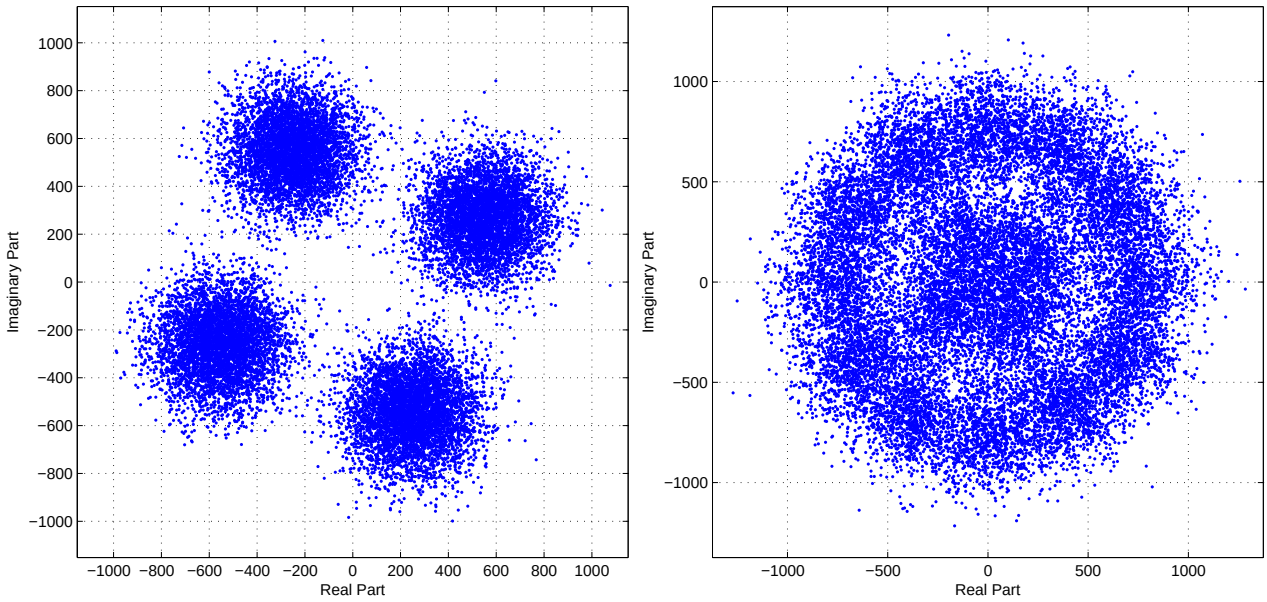


Figure 7.4.3.: Constellation diagram of the received QPSK (left) and 16-APSK (right) symbols with external clock reference and $SNR = 10\,dB$

7.4.1. Influence of the Clock Mismatch

Earlier in this thesis, the influence of the clock mismatch to the timing estimation was mentioned. Unfortunately this mismatch also leads to a carrier frequency offset due to wrong mixing in the analog stage of the USRP N210. This leads from the fact that the mixing frequency is also derived from the internal clock. This carrier frequency offset leads to a rotation of the symbols by the angular speed $\varphi = 2 \cdot \pi \cdot \triangle f$, where $\triangle f$ is the carrier frequency offset. Since the estimation and correction of this offset is not part of this thesis, but it might

be interesting to see the performance of the USRP N210, only an approximate estimation of the carrier frequency offset is done here.
Therefore the constellation diagram of a QPSK signal, where no external clock reference is attached, is used to demonstrate this rotation.
For the plot in figure 7.4.4, $N = 2000$ symbols are illustrated and a phase rotation of about $0.4688\,rad$ can be observed in the left plot where a QPSK signal is used. Since this rotation is no exact measurement for the phase rotation, this gives only a rough estimation of the carrier frequency offset.
The observation results in a rotation approximately given by

$$\varphi = \frac{0.4688\,rad}{N} = 234 \cdot 10^{-6} \frac{rad}{symbol}$$

The input symbols c_{in} are rotated by the angular speed φ which leads to

$$c_{out}[n] = c_{in}[n] \cdot e^{j \cdot \varphi \cdot n}$$

Now the carrier frequency offset $\triangle f$ can be calculated using the following rearrangement

$$\varphi = 2 \cdot \pi \cdot \triangle f \rightarrow \triangle f = \frac{\varphi}{2 \cdot \pi} = 37.3 \cdot 10^{-6} \frac{1}{symbol}$$

When multiplying this result with the symbol rate of $f_{sym} = 1\,MBaud$ we get a frequency offset of $\triangle f = \varphi \cdot f_{sym} = 37.3\,Hz$.
If we have a look at the right plot in figure 7.4.4, where the modulation scheme is 16-APSK, we can see that the rotation due to the carrier frequency offset results in constellation points that are not distinguishable with the naked eye. Only the two amplitude levels of the 16-APSK signal are visible.

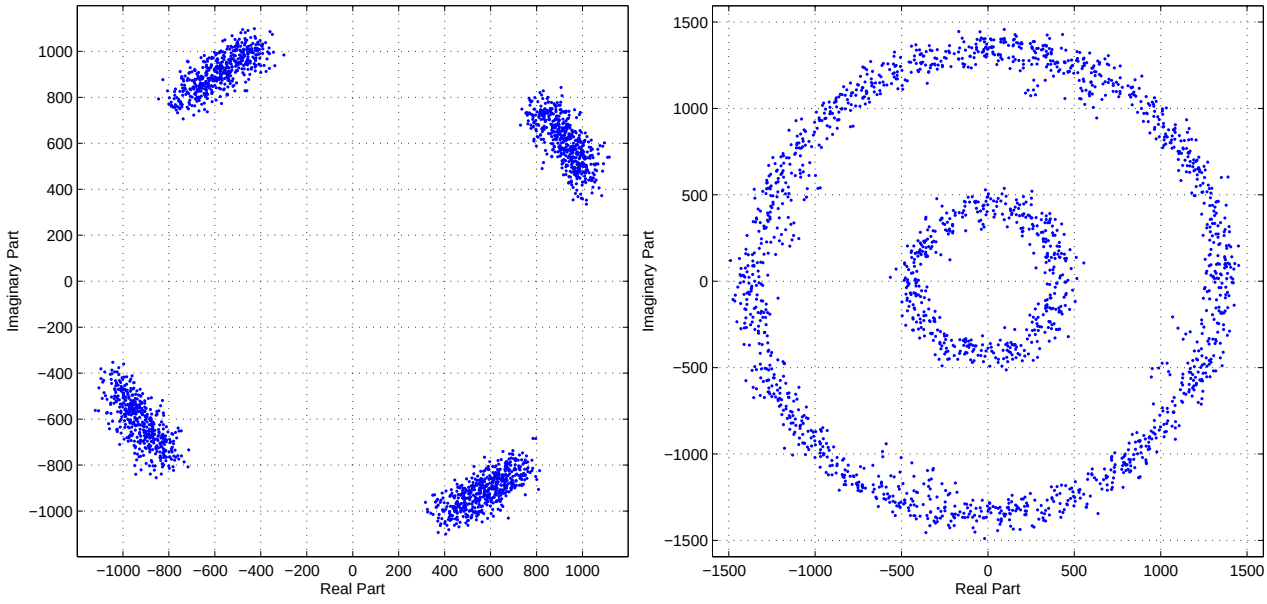


Figure 7.4.4.: Constellation diagram of the received QPSK (left) and 16-APSK (right) symbols with no external clock reference and $SNR = 29\,dB$

8. Conclusion

This thesis presents and analyzes a sampling rate conversion technique that can be employed for fractional conversion rates near 1 using a FIR interpolator. The implementation of this sampling rate converter uses a pre-calculated FIR filter which reduces the effort for online computation of the required filter coefficients. This FIR structure is highly oversampled providing this way a good quantization for the delay of the output sample. Additionally, a linear interpolation is introduced which improves this quantization even more. Using both high oversampling and linear interpolation, very fine delays can be applied to the input signal. Several simulations illustrate the performance of the FIR interpolator showing better results than a cubic interpolator.
It has also been shown that the filter design is very important for this sampling rate conversion technique. Hence, the most important filter parameters are analyzed and illustrated to present reference values for the delay variation application.

For the timing recovery procedure, the simulated modules provided good results in terms of the symbol error rate. The FPGA implementation of the investigated FIR interpolator requires less multiplications than the cubic interpolator in Farrow structure, but needs an additional table for the filter coefficients instead. Hence, there is no real advantage in using the FIR interpolator instead for the purpose of timing recovery.
The practical measurements on the implemented FPGA modules showed no significant differences to simulation results and in the actual SNR estimation experiment the whole system worked properly.
For the timing estimation, several improvements are suggested for future work. Since cycle slips occurred quite often in the final system, an improved unwrapping algorithm is recommended. Also the window length of the estimator algorithm can be increased in order to minimize the jitter variance. But to apply a larger window length, a larger FPGA should be available on the SDR, because the hardware effort also increases. Another possibility for future work is to consider a tracking algorithm for timing recovery.

One of the biggest issues in this thesis was the implementation of the modules on the USRP N210 FPGA. Since the number of available slices was very low, several trade-offs in terms of area had to be made to ensure a working system. When using a larger FPGA, more filter taps of the interpolator and a larger window length for the timing estimator might lead to a better performance. However, for the mentioned ALPHASAT TDP5 experiment the performance is sufficient.
The lack of documentation on the existing hardware structure of the USRP N210 also complicated the insertion of appropriate modules and made an intensive study of the existing sources necessary.

A. GNU Radio and UHD

A.1. Installation

For the installation of GNU Radio and UHD (Universal Hardware Driver) both projects need to be downloaded from the GNU radio website http://gnuradio.org[1]. Any sources or scripts that are mentioned in this section as well as more detailed instructions can also be found on this website.
There are pre-built binaries of GNU Radio available for the Linux distributions Ubuntu and Fedora. But when using the USRP devices, which require also the installation of UHD, the most common way is to manually compile everything directly from the sources.
When installing from the sources, it is mandatory to start with UHD before GNU Radio since the latter checks for the presence of UHD on the system. The installation is based on makefiles which are, in case of UHD, created by using CMake.
Fortunately, there is an install script available for the Linux distributions Ubuntu and Fedora. This script downloads the newest required sources, checks for dependencies and, if necessary, installs missing packages. The manual installation on a Linux platform will be described in the next section.

A.1.1. Installation of UHD on Linux

When using the GNU Radio toolkit with UHD, the latter has to be installed previously to the actual GNU Radio installation. Since Linux Ubuntu 10.10 was used for the implementation in this thesis, the following instructions refer to this operating system. The UHD sources can be downloaded using the git version control tool by entering the command:

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

Now enter the new directory `uhd` by typing the command:

```
cd uhd
```

create a build directory, enter it and start the CMake script by entering the commands:

```
mkdir build
cd build
cmake ../
```

This will generate the necessary makefiles and the build process can now be started with the commands:

[1] This URL was checked for validity in February 2012

```
make
sudo make install
sudo ldconfig
```

The directories `/usr/local/share/uhd` and `/usr/local/bin` now contain all the installed files. Also the firmware and the FPGA image are available in the git directory.

A.1.2. Installation of GNU Radio on Linux

For this thesis the Linux distribution Ubuntu 10.10 was used and therefore this installation guide refers to this operating system. First of all, the required packages for building the source need to be installed. On the GNU Radio website these packages are listed and they can be installed using the Synaptic Package Manager. Please note the indicated versions of these packages.
In order to download the sources, GNU Radio can be downloaded from the using git version control system by typing the command:

```
git clone http://gnuradio.org/git/gnuradio.git
```

This will create a directory called `gnuradio`. Now enter this directory by entering the command:

```
cd gnuradio
```

Now the source can be configured and built by typing the commands:

```
./bootstrap
./configure
make
```
`make check` (this is an optional self-check of GNU-Radio)
```
sudo make install
sudo ldconfig
```

All files will be installed into the directories `/usr/local/share/gnuradio` and `/usr/local/bin`. After the `./configure` part of the installation, make sure that the component `gr-uhd` is enabled if you want to use UHD.
For instructions on special configurations like enabling individual components please visit the GNU Radio website.

A.2. UHD Tools

There are three very useful tools that are worth mentioning when working with the USRP devices.

A.2.1. UHD Find Devices

This tool can be used to find all USRP devices that are connected to the host PC. The tool can be called by typing the command:

```
/usr/local/bin/uhd_find_devices
```

A.2.2. UHD USRP Probe

In order to get all necessary information of the attached USRP board and of the daughterboards, this tool can probe the device. To execute this tool enter the following command:

```
/usr/local/bin/uhd_usrp_probe
```

A.2.3. UHD USRP Net Burner

This Python script is a tool which is used to burn new firmware and FPGA images onto the USRP devices:

```
/usr/local/share/uhd/utils/usrp_n2xx_net_burner.py
```

There is also a graphical tool which can be used for the same purpose:

```
/usr/local/share/uhd/utils/usrp_n2xx_net_burner_gui.py
```

A.3. GNU Radio Companion

When starting with GNU Radio, the easiest way of working with the toolkit is the graphical tool GNU Radio Companion (GRC). The basic concept of GNU Radio are the flow graphs where nodes are called blocks and the edges between these blocks represent the data flow direction. The blocks are responsible for the actual signal processing and are written in the programming language C++, whereas the actual application is written in Python. These blocks should be kept as modular as possible to keep the software flexible, which means that each block should ideally perform one single signal processing operation.

GRC now gives users the opportunity to create such flow graphs without the need of any actual programming work. A large number of common signal processing block are already provided by GNU Radio, such as many different kinds of sources, sinks, synchronizers, modulators and filters.

Figure A.3.1 shows an example flow graph containing an USRP source, a root-raised-cosine filter and a FFT sink for displaying the spectrum of the signal. This three main blocks are connected by arrows which also indicate the direction of the data flow. For passing the data between blocks, many data types can be defined, like complex floating point, or short values. The data types are indicated by specific colours of the sockets. On the right side of the window the available blocks can be selected and inserted into the current graph. At the bottom of the window, console prints are displayed and at the top there are shortcut symbols for creating and starting the graph as well as for common operations like saving graphs and deleting blocks.

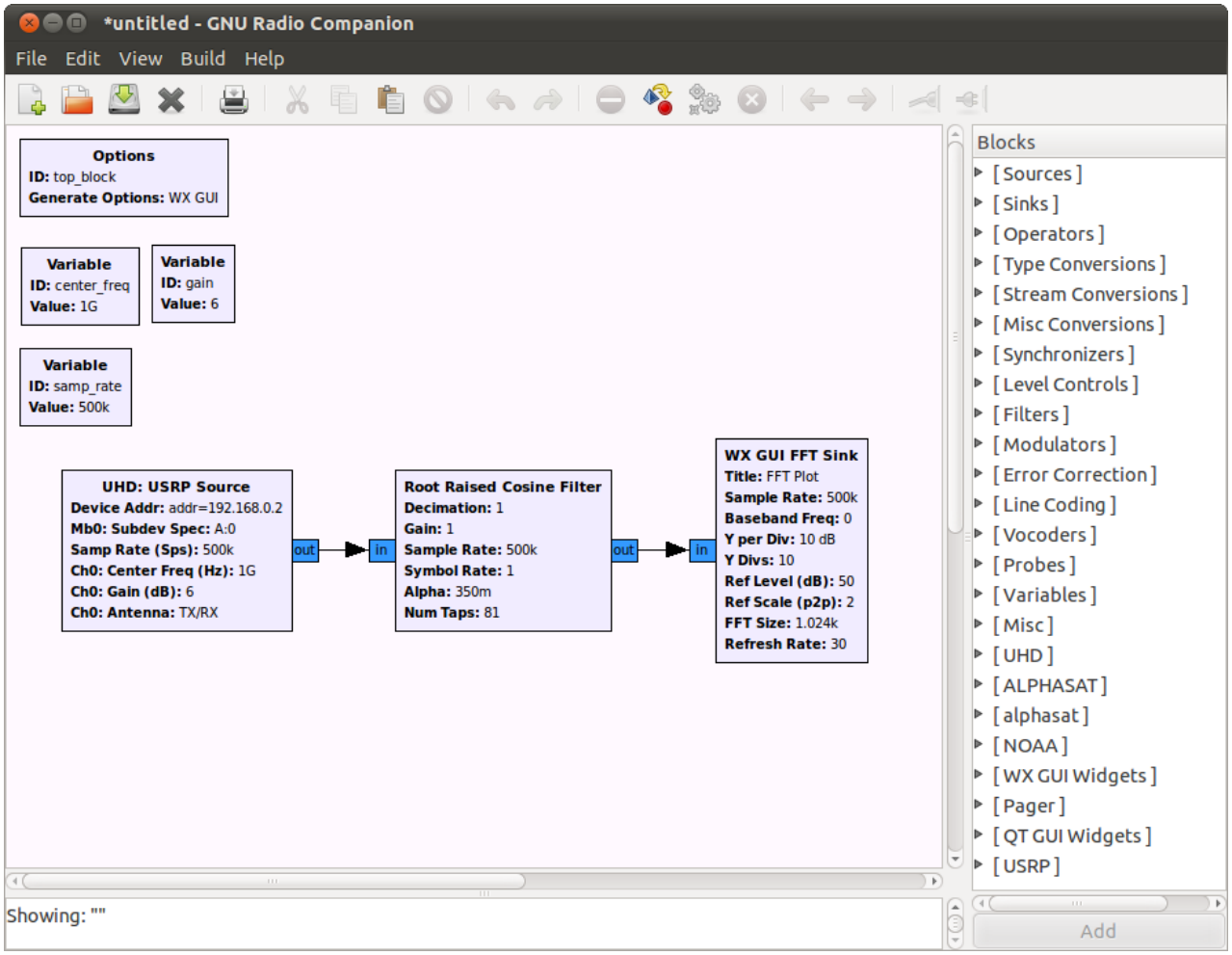


Figure A.3.1.: Sample graph of GNU Radio Companion

A.3.1. USRP Source

The first block `UHD: USRP Source` is responsible for the communication with a USRP device that is attached to the system. In case of the USRP N210, which is used in this thesis, UDP packets over the Gigabit Ethernet interface are used for the communication between the host PC (UHD) and the USRP device. Figure A.3.2 shows the property window of this block where all the necessary parameters can be selected appropriately. When a parameter is set, the UHD sends the corresponding command to the USRP. The firmware running on the USRP then receives this command and reacts correspondingly, like setting the values of particular registers of its hardware.
The parameters are now presented in a brief manner to explain the functionality of the USRP N210.

- `ID` represents a unique identity name for the block in the current graph.

- `Output Type` defines the type of data passed to the connected blocks. Valid types are complex, which corresponds to complex floating point, and short, which corresponds to a vector of two short values. The first one indicates the real part and the second one indicates the imaginary part of the complex valued output signal.

- `Device Addr` is used to specify the IP address of the USRP device. The prefix `addr=` is mandatory for this parameter. Also a list of addresses can be entered if more than one USRP device is used.

- `Ref Clock` is used to define whether the clock of the USRP is set to the internal oscillator or to an external clock that can be connected to the device. If more than one USRP device is connected, they need to be synchronized. This can be set by the parameter `Sync`.

- `Clock Rate` can be used to set the clock rate of the FPGA on the USRP device.

- `Num Mboards` defines the number of USRP motherboards that are used in the current configuration.

- `Subdev Spec` is used to set the specifications for the daughterboards that are attached to the USRP devices used in the current configuration. To see the valid specifications, run the Python script `usrp_probe` in order to probe the USRP and see its configuration. This script is included in the installation of GNU Radio.

- `Num Channels` defines the number of receive channels of the attached devices. The USRP N210 can be used to receive up to two channels.

- `Samp Rate` defines the sampling rate of the received signal. When using the USRP N210 the 100 $MSps$ are decimated to the desired sampling rate.

- `Center Freq` defines the frequency of the mixer and therefore the center frequency of the signal that is mixed into the baseband.

- `Gain` can be used to set the gain in dB of the amplifiers and attenuators of the daughterboard.

- `Antenna` defines the name of the antenna that is used for the current channel, if multiple antennas are available. To see the list of all available antennas run the Python script `usrp_probe`.

- `Bandwidth` can be used to configure the bandwidth filters available on the daughterboards that are attached to the USRP devices.

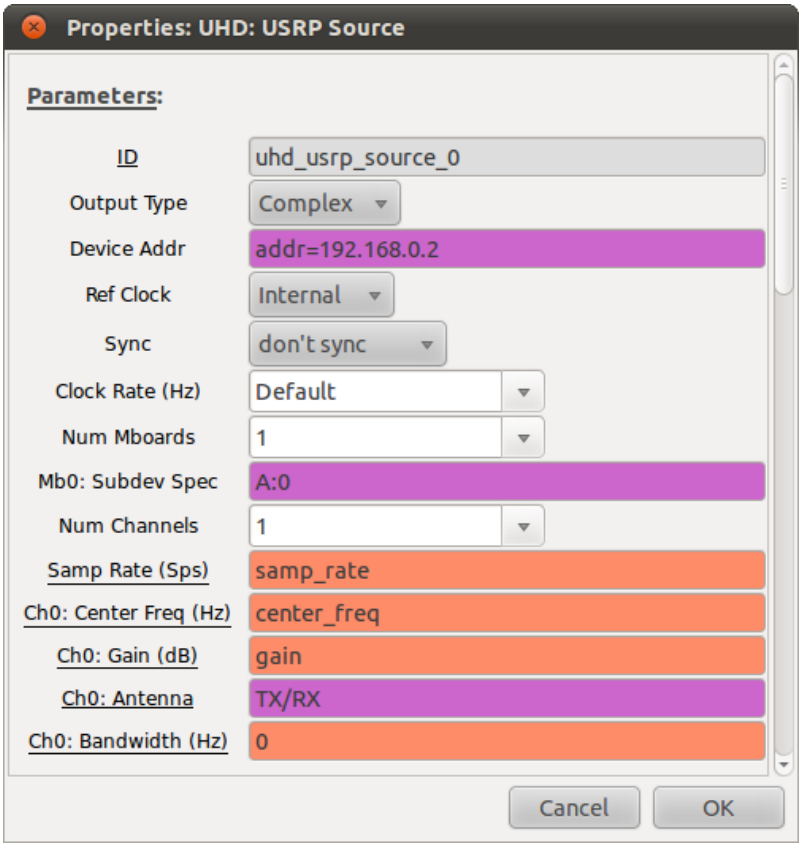


Figure A.3.2.: Parameters for USRP source block

A.4. Developing in GNU Radio

As already mentioned, there are a large number of existing blocks already included into the GNU Radio toolkit. But if one needs to develop a signal processing block that is not available in the existing toolbox, GNU Radio allows you to add your own blocks to the existing ones. The main signal processing of one block can be implemented in C++, whereas the connection between these block, the flow graph, is implemented in Python. When writing a block, it has to be compiled into a shared library object. This way the block can be imported into Python. We will now talk about the main points of adding own blocks into GNU Radio. For a detailed manual and code examples please visit the GNU Radio website http://gnuradio.org[2].
In GNU Radio, each block will be executed by one thread. Therefore, the order of execution is not fixed. Especially for debugging a flow graph, this fact needs to be considered.
The data passing between the blocks is performed by the data buffers `gr_buffer` for each connection of the blocks.

A.4.1. Writing Signal Processing Blocks in GNU Radio

The basic structure of a signal processing block is the corresponding C++ class (`.cc` and `.h` file) that needs to be implemented. Templates of the source- and makefiles can be found in the downloaded directory of GNU Radio:

[2]This URL was checked for validity in March 2012

gnuradio/gr-howto-write-a-block

The directory layout can also be seen in this template. This new class needs to be derived from the mother `gr_block` of all signal processing blocks. This base class contains several methods and member variables that need to be implemented in order to work in the GNU Radio toolkit and communicate with the other blocks in the flow graph.

gr_make_io_signature

First of all, the signature of the new block needs to be defined. The signature describes the input port and output port of the new block. It is stored in the class `gr_io_signature`, which defines the number of input streams as well as the data size of the input streams. Pointers to these objects can be created by the function

```
gr_io_signature_sptr gr_make_io_signature(int min_streams,
    int max_streams, int sizeof_stream_item)
```

One object of this class needs to be created for the input port and one object for the output port by calling this function passing the resulting pointers to the constructor of the new block class.

general_work

For the execution of the actual signal processing, the virtual method

```
int general_work(int noutput_items, gr_vector_int &ninput_items,
    gr_vector_const_void_star &input_items, gr_vector_void_star &output_items)
```

needs to be implemented. This method will be executed by the scheduler of GNU Radio when enough input data are available.
The first parameter `noutput_items` is an integer value that defines the number of output samples that can be produced in the current call of the method. This means that, if the method is called with `noutput_items=1024` for example, the function can return at most that amount of output samples. Producing less samples is also possible.
The second parameter is the pointer to the vector `ninput_items` which defines the number of available input samples for each input port. A pointer to a vector of input items is the third parameter `input_items`. It contains a pointer to a vector for each input stream. Using this input samples, the output samples can be calculated. These output samples then are stored into a vector for each output stream. The pointer to this vectors is given by the last parameter `output_items`.
When the output samples are produced, the scheduler needs to know how many of them are produced and also how many input samples are consumed. The number of produced output samples can be set by the return value of the method `general_work`. When -1 is returned by the method, the system aborts the execution of the whole flow graph. For telling the scheduler of GNU Radio how many input items are consumed, the next function needs to be called.

consume

The function

```
void consume(int which_input, int how_many_items)
```

is used to tell the GNU Radio system how many input items are used on which input port. If every port consumed the same amount of input samples, the function

```
void consume_each(int how_many_items)
```

can be called instead. These consumed input samples are then removed from the buffer and are no longer available in the next call of the `general_work` method.

forecast

Another important virtual method that needs to be implemented is

```
void forecast(int noutput_items, gr_vector_int &ninput_items_required)
```

The call of this function is needed to tell the GNU Radio system the input-to-output relation of the signal processing block. This function is also called by the GNU Radio system in order to check how many input items are required. Therefore, the function has the parameter `noutput_items` just like the method `general_work`. In the second parameter `ninput_items_required` which is a pointer to a vector, the programmer has to tell the required number of input samples for each input stream when `noutput_items` need to be produced. Hence, the system can always provide enough input samples on each call of the `general_work` method.

set_output_multiple

Another function worth mentioning is

```
void set_output_multiple(int multiple)
```

This function is the setter method of the member variable `d_output_multiple`. In signal processing the calculation in sample blocks is sometimes required for easier calculations. If blocks of output samples have to be produced, this can be done by calling this setter function. After this call, the scheduler will ensure that the argument `noutput_items` for the methods `general_work` and `forecast` is an integer multiple of this given value. The default value is set to one, but for certain computations other values are much more desirable.

A.4.2. SWIG

In case the C++ class for this signal processing block is implemented by creating the corresponding `.cc` and `.h` file, another file is necessary to tell the system how Python can use the implemented C++ code. This can be done by creating a `.i` file, which contains all necessary information for a special wrapper called SWIG (Simplified Wrapper and Interface) that does the binding between C++ and Python. The most easy way of creating this file is to alter an

existing template file.

When the flow graph with this new block is executed, the system first checks the input-to-output relation by calling the `forecast` method. Then it waits until enough samples are available on each input stream to produce a certain amount of output items. When enough items are available, the method `general_work` is called and the produced items are sent to the buffer of the following block.

A.4.3. GNU Radio Applications

Applications or flow graphs can either be created by using the graphical interface provided by GRC, or by writing suitable Python classes. In these Python classes, signal processing blocks can be added and connections between these block can be created.
When creating flow graphs, loop calls of the graph can be performed automatically which is not possible in GRC. This makes several calls of the same graph with different parameters possible without the need of a manual start of the whole graph. Also the reaction to specific return values can be implemented.
Especially for complex simulations with lots of parameters, this kind of creating flow graphs is preferred. A corresponding template of such a Python class file (`.py`) can also be found in the directory

`gnuradio/gr-howto-write-a-block/apps`

A.4.4. Building and Installation of Blocks

The creation, installation and linking of the new signal processing block into the existing toolkit is again done by calling makefiles. This means that after altering the source, the commands:

```
make
sudo make install
sudo ldconfig
```

need to be executed from the source directory of the current block in order to build and install all the necessary shared object files.
When installing GNU Radio, a lot of example files are included, which can either be used as manuals or as templates for creating the signal processing blocks.

Bibliography

[BLM04] J. R. Barry, E. A. Lee, and D. G. Messerschmitt. *Digital Communication*. Springer, 3rd edition, 2004.

[EGH93] L. Erup, F.M. Gardner, and R.A. Harris. Interpolation in digital modems II: implementation and performance. *IEEE Transactions on Communications*, 41(6):998 –1008, June 1993.

[ETS05] ETSI 102.376(V1.1.1). Digital Video Broadcasting (DVB) User guidelines for the second generation system for broadcasting, interactive services, news gathering and other broadband satellite applications (DVB-S2) . Technical report, February 2005.

[Gar90] F.M. Gardner. Timing adjustment via interpolation in digital demodulators. Technical report, Gardner Research Company, June 1990.

[Gar93] F.M. Gardner. Interpolation in digital modems I: fundamentals. *IEEE Transactions on Communications*, 41(3):501 –507, March 1993.

[Kai74] J.F. Kaiser. Nonrecursive digital filter design using the I0-sinh window function. *IEEE Symp. Circuits and Systems*, pages 20–23, April 1974.

[Lyo11] R. G. Lyons. *Understanding Digital Signal Processing*. Prentice Hall, 3rd edition, 2011.

[MB07] U. Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer, 3rd edition, 2007.

[MD97] U. Mengali and A. N. D'Andrea. *Synchronization Techniques for Digital Receivers*. Plenum Press, 1997.

[MdJ92] M. Moeneclaey and G. de Jonghe. Tracking performance comparison of two feedforward ML-oriented carrier-independent NDA symbol synchronizers. *IEEE Transactions on Communications*, 40(9):1423 –1425, September 1992.

[MMF98] H. Meyr, M. Moeneclaey, and S. A. Fechtel. *Digital Communication Receivers*. Wiley, 1998.

[OM88] M. Oerder and H. Meyr. Digital filter and square timing recovery. *IEEE Transactions on Communications*, 36(5):605 –612, May 1988.

[PM96] J. G. Proakis and D. G. Manolakis. *Digital Signal Processing - Principles, Algorithms, and Applications*. Prentice-Hall, 3rd edition, 1996.

[Resa] Ettus Research. TX and RX daughterboards. Retrieved March, 2012. Available from: http://www.ettus.com/downloads/ettus_daughterboards.pdf.

[Resb] Ettus Research. USRP N200 series. Retrieved March, 2012. Available from: http://www.ettus.com/downloads/ettus_ds_usrp_n200series_v3.pdf.

[Skl01] B. Sklar. *Digital Communications - Fundamentals and Applications*. Prentice-Hall, 2nd edition, 2001.

[Smi11] J. O. Smith. Digital audio resampling, November 2011. Retrieved March, 2012. Available from: https://ccrma.stanford.edu/~jos/resample/.

[Tür12] E. Türkyilmaz. Development, implementation and assessment of DVB-S2 frame synchronization and SNR estimation on the GNU Radio platform. Master's thesis, Graz University of Technology, 2012.

[WMLY08] R. Woods, J. McAllister, G. Lightbody, and Y. Yi. *FPGA-based Implementation of Signal Processing Systems*. Wiley, 2008.

[Xil11] Xilinx. LogiCORE IP CORDIC v4.0, March 2011. Retrieved March, 2012. Available from: http://www.xilinx.com/support/documentation/ip_documentation/cordic_ds249.pdf.