Balázs Ferenczi, BSc.

Simulation of Critical Modules for a CCSDS Modem

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Electrical Engineering

submitted to

Graz University of Technology

Supervisor

Assoc. Prof. Dipl.-Ing. Dr. techn. Wilfried Gappmair

Institute of Communication Networks and Satellite Communications

Dipl.-Ing. Dr. techn. Johannes Ebert

Graz, February 2015

AFFIDAVIT

Danksagung

Dank gilt meinem Betreuer Assoc. Prof. Dipl.-Ing. Dr. techn. Wilfried Gappmair für die fachkundige Unterstützung während des Diplomarbeitsprozesses. Ich danke allen Beraterinnen und Beratern bei Joanneum Research, die mich bei diesem Vorhaben unterstützt haben. Besonders hervorheben möchte ich meine Ansprechperson Dipl.-Ing. Dr. techn. Joannes Ebert, der mir jederzeit bei Fragen und Anliegen zur Verfügung stand. Danken möchte ich außerdem Dipl.-Ing. Eral Türkyilmaz, Dipl.-Ing. Michael Schmidt, Dipl.-Ing. Dr. techn. Michael Schönhuber, Mag. Sabine Kastner-Puschl und Dipl.-Ing. Harald Schlemmer für deren Hilfe und Beratung.

Kurzfassung

Kommunikation ist eine der wichtigsten Aufgaben zwischen Satelliten und Bodenstationen, wie sie zum Beispiel bei Telemetrie und Telekommandos zum Einsatz kommt. Die erlaubten Kommunikationsschemata sind im CCSDS-Standard definiert. Das Ziel der vorliegenden Arbeit ist es, eine praktische Übersicht für die auf Software basierende Modulation bzw. Demodulation von Signalen zu geben. Zudem soll ein PC-basiertes System für die Signalverarbeitung in Echtzeit empfohlen werden. Das System soll auch fähig sein, den Erfordernissen mit hoher Zuverlässigkeit gerecht zu werden, die unterstützend für Weltraummissionen benötigt werden.

Diese Diplomarbeit enthält die grundsätzlichen Strukturen wie auch die detaillierte Beschreibung von verfügbaren digitalen Kommunikationsmethoden. Ein auf MATLAB basierter Simulator wurde entwickelt und eingesetzt, um die Performance und die Eigenschaften von Algorithmen für Kanalkodierung, Modulation, Demodulation sowie eine datengestützte und nicht datengeschützte Synchronisation zu evaluieren.

Darüber hinaus soll diese Arbeit die Planung und Dimensionierung von PC-basierten Systemen veranschaulichen. Um die nötige Rechenleistung schätzen zu können, wird die Rechenkomplexität für alle erwähnten Algorithmen untersucht. Unglücklicherweise ist momentan keine CPU leistungsfähig genug, um die ganze Signalverarbeitung in Echtzeit seriell durchführen zu können. Deshalb ist die Untersuchung der Parallelisierung ein wichtiger Teil dieser Arbeit.

Die Ergebnisse zeigen, dass es möglich ist, ein auf Software basiertes System für ein CCSDS-Modem mit einer Leistung von bis zu 40 Millionen Symbolen pro Sekunde zu entwickeln.

Abstract

Communication is one of the major tasks between space and ground segments, most prominently exemplified by telemetry and telecommand. The allowed communication schemes are defined by the CCSDS standard. The purpose of this work is to get a practical overview of software defined radio-based modulation and demodulation of these signals, and then recommend a PC-based system for real-time signal processing. Furthermore, the system has to be able to satisfy the requirements of high reliability required to supply space missions.

The thesis contains the basic structures as well as a detailed description of the available digital communication methods. A MATLAB-based simulator has been developed and used to evaluate the performance as well as the properties of the usable algorithms for channel coding, modulation, and synchronization.

In addition, this thesis presents the planning and dimensioning of a PC-based system. To be able to estimate the needed resources, the computational complexity is evaluated for all of the above mentioned algorithms. Unfortunately, none of the today available CPUs is strong enough to be able to do the complete signal processing in real time. Therefore, the examination of the possibilities for parallel organization is an important part of this work as well.

The results show that to date it is possible to build a software-based system for a CCSDS modem up to 40 Msymbols per second.

Acronyms

AVX……………	Advanced vector extension
AWGN	Additive white Gaussian noise
BER	Bit error rate
BPSK…………...	Binary phase shift keying
CCSDS…………	Consultative Committee for Space Data Systems
CPU…………….	Central processing unit
CRLB	Cramer-Rao lower bound
DA	Data-aided
DD	Decision-directed
ECSS…………..	European Cooperation for Space Standardization
FB	Feedback
FEC	Forward error correction
FF	Feedforward
FLOPS…………	Floating point operations pro second
GMSK…………..	Gaussian minimum shift keying
GPU…………….	Graphical processing unit
ISI	Inter-symbol interference
LUT	Look-up table
L&R	Luise-Reggiannini
MF	Matched filter
ML	Maximum likelihood
M&M	Morelli-Mengali
NCO	Numerically controlled oscillator
NDA	Non-data-aided
O&M	Oerder-Meyr
PLL	Phase-locked loop
PSK	Phase shift keying
QAM	Quadrature amplitude modulation
QPSK…………..	Quadrature phase-shift keying
RCos	Raised cosine
RRCos	Root-raised cosine
SNR	Signal-to-noise ratio
TCM…………….	Trellis coded modulation
TDMA	Time-division multiple access
V&V	Viterbi-Viterbi

Table of Contents

1. Introduction

Human presence in space has a history over 50 years. After some decades of reduced attention, in the last years space industries have a growing perspective.
Communication is one of the major tasks of all equipment in space. Without reliable communication the devices are not able to carry out the tasks they have been developed for. If the costs of space technologies are taken into consideration, it is easy to understand why the exact communication tests, and reliable communication devices are so important.
The standard modulation methods and coding techniques in space technologies are defined in CCSDS standard. Customer can choose from these modules, and build an own communication method optimized to individual needs, similar to LEGO.
The today used ground stations use in general standardized connections, and modules to be able to adapt the transceiver to consumer needs. Because of the changes in the hardware structure of the devices, and the need of high reliability, integration tests are needed all the time, which costs a lot in time and money too.
Today software radio can offer an alternative solution. Software based modular systems do not need integration tests all the time, because in this case the changes appear only in software. Another advantage of the SDR based system is that redundancy can be easier achieved. With the increased computational capacity of processors and the appearance of GPU based systems a decreased number of processing units are needed. The software based systems can be cheaper because of the usage of standard devices, and will be always more capable with time.
The purpose of this study is to get a practical sight of the usable algorithms and data structures, by taking into consideration the structure of modern CPUs GPUs, and the possibility of parallel organization.

2. Framework

2.1 Introduction

The purpose of this framework is to provide a powerful, easy to use platform, for development and evaluation of baseband transmitter, receiver algorithms, and structures relevant for satellite communication.

2.2 Basic structure of framework

The framework is based on Matlab/Simulink. The communication model is implemented in Simulink, with algorithms implemented in blocks. The configuration of the communication model blocks can be made in Matlab with global variables. All these variables are included by blocks_config.m file, they can be direct edited here. The basic communication simulation can be made by calling the Simulink simulation file from Matlab. The variables for the simulation (SNR, Timing error …) can be edited directly before the call of the simulation.

2.3 Basic files and usage of simulation framework

Framework.slx

The simulation can be separated to five sections followed by each other from left to right (Figure 1):

- Bit pattern generation
- Baseband transmitter structure
- Baseband channel model
- Baseband receiver structure
- Processing of received data

The basic used signal structure is frame based. Each signal is stored in a column vector, and the sample time is chosen to 1.
Each block can be configured through Matlab global variables. The recommended name convention is that each variable has a prefix (T_Transmitter, CH_Channel, R_Receiver), and a good readable unique name. Each used variable has to be stored with a default value in blocks_config.m.

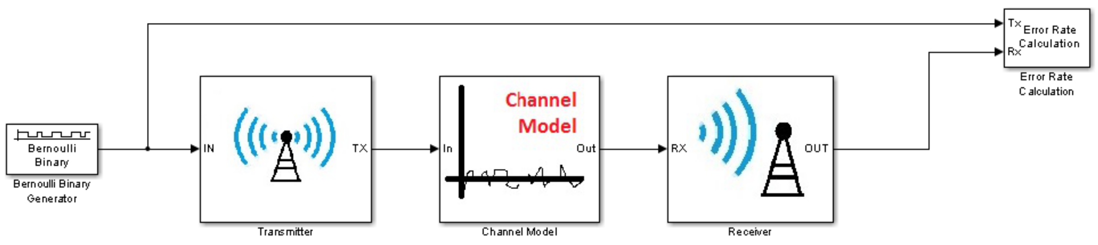


Figure 1: Framework

Blocks_config.m

In this file are all parameters needed for the framework. It is separated to four main structures (Transceiver, Channel model, Receiver, Other blocks), and has some longer notes in the end, these notes are quoted by blocks. Structure of the blocks:

```
% Filter: Root raised cosine
        T_Rolloff_factor=0.8;
        Group_delay=10; % delay in symbols (filterlength)
```

It has to be taken into consideration, that Simulink compiles the Matlab function blocks to C to significantly improve the simulation speed. It has some draw-backs; the main is that Simulink has to know the block size before compilation. With other words, the length of the output vector cannot depend on any input variable of the block! If it has to be, it is possible to have variable size blocks, but they are not supported from all Simulink blocks, and they must have a hard limit of maximal block size too. In general, if the output length depends on input variables, and no variable size block is usable, then the parameter has to be given in the file, and in blocks_config.m file has to be noted where it can be ed-ited.

How to run simulation from Matlab?

A basic simulation has this structure:

1. Call the default configuration

```
blocks_config;
```

2. Change some variable values:

```
SNR_values=15; % The SNR values to compute
```

3. Call the simulation:

```
sim('SIM_Framework');
```

4. Process the data from simulation.

   plot(signal);

Modules_to_ use.slx

All developed and used Simulink blocks can be found in this slx file. All the blocks are categorized into subsystems. The Channel model is the only exception; it is simply usable in this form.

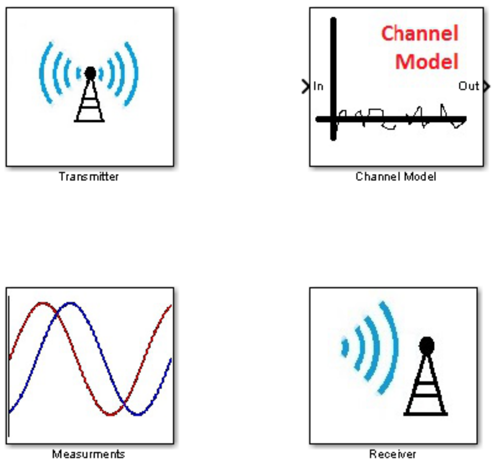


Figure 2: Modules

2.4 Main parts of a baseband digital communication system model

A baseband digital communication system model does not contain a model for the RF modulator, and the other parts of the RF chain (C. C. Bissell, D. A. Chapman, 1992) (red blocks in Figure 3). The main purpose of the baseband model is to make the computation faster by omitting the unnecessary RF parts from the simulation, and by replacing the RF channel model with an equivalent baseband channel model.

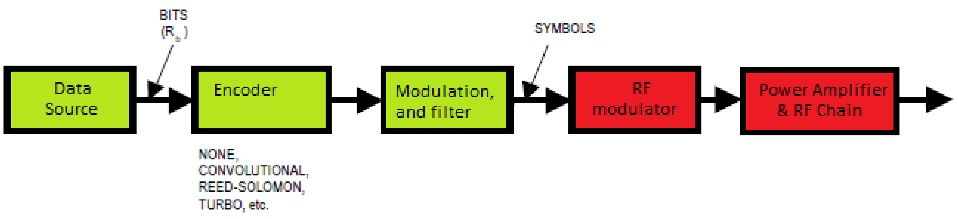

BITS
(R_b)
SYMBOLS
Data Source
Encoder
Modulation, and filter
RF modulator
Power Amplifier & RF Chain
NONE, CONVOLUTIONAL, REED-SOLOMON, TURBO, etc.

Figure 3:Main parts of a digital transmitter

Transmitter: The transmitter has a similar structure in almost all cases. It has an encoder to give some redundancy to the bit pattern. Then a modulator makes a baseband signal from the bit pattern, with some combination of pulse code modulation, modulation, and filtering.

Channel model: In baseband system model this baseband signal is directly distorted by a baseband model of the used channel.

Receiver: The purpose of the receiver is to recover the original data sequence. To reach this, it has to estimate the original signal from the received signal, and then demodulate it. The estimation is done by estimating the channel parameters, used for correction purposes. Although the structure of the receiver is similar to the reverse of the transmitter, because of often used feedback structures it cannot be so generally visualized.

2.5 Baseband channel model

The mostly used channel model for satellite communication is AWGN channel model (Figure 4). The model contains an AWGN noise generator, and some other main effects of the channel to the signal, which have to be corrected at receiver side. The sample time is considered to be 1. It has to be taken into consideration for all Hz based properties.

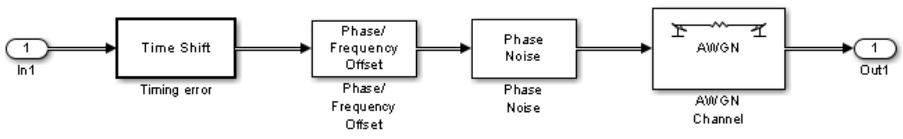


Figure 4: Baseband channel model

- **Timing offset**
  The purpose of this block is to generate timing offset. To lower the computation costs the timing offset accepts just integer numbers. Fractional timing offsets can be generated by oversampling at transceiver side, and build a downsample block into channel model before the receiver side. The Doppler effect makes some timing offset error, but it is significantly smaller than the caused frequency error, and in satellite communication it can be well propagated, and corrected. For that reason, this version of channel model does not include this effect.

- **Phase/frequency offset**
  The purpose of this block is to generate phase, and frequency offset. The work of model:
  $$out(t) = input(t) * e^{j*Frequency\ offset*t+phase\ offset}$$

- **Phase noise**
  Generates additionally phase noise. CH_Phase_noise_level describes the magnitude of the phase noise (dBc/Hz), and CH_Phase_frequency_offset describes the deviation of the noise. For more detailed information watch the block description.

- **AWGN channel**
  This block adds noise to the signal. The added noise has uniform power across the frequency band (white), and it has normal distribution in time domain (Gaussian). The CH_SNR variable contains the average Signal power (1) to Noise power ratio in dB.

2.6 Some examples of frequently used measurements with the framework

Basic measurements with one setting

In general it is easier to export the needed data to workspace, and there process it. The process is of course possible with the same code used in a Matlab function Simulink block too.

- Timing measurements

```
%Time

figure('name','Time')
plot(abs(signal_after_RRC(1:500)));
```

- Spectrum

```
%FFT
figure('name','FFT')
plot(abs(fft(signal_after_RRC)));
```

- Eye diagram (Figure 5.)

```
%Eye diagram
eyediagram(real(signal_after_RRC),Upsample_factor*2);
```

- Scatter plot

```
%Scatter plot
figure('name','Scatter plot')
plot(signal_downsampled,'.');
```

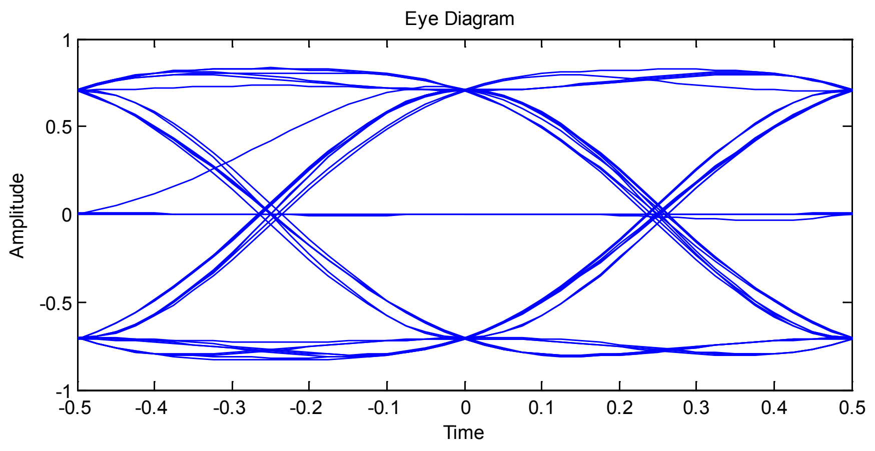


Figure 5: Eye diagram

3. Channel Coding

The main goal of channel coding is to reduce the BER for a given SNR to an acceptable level. In Channel coding we assume that we have perfect estimation of the channel parameters, and therefore a perfectly corrected channel.
In general we have to put some redundancy to the information, and with the knowledge of this redundancy we can get back our original bit sequence at the receiver.
Because this redundancy makes our data packets longer (code rate), and because with one symbol we can send more bits (modulation gain), it is a better idea to measure the effectiveness of coding not in terms of BER vs. SNR, but BER vs Eb/No.

3.1 Simulation Model

Because we assumed that the channel is perfectly corrected, we can leave timing, and phase error from channel model. If there is no timing error we can leave out upconversion, downconversion, and channel filter too, because they have no influence to received data in a memory less channel. These modifications make the simulation lot faster.

The curves for uncoded BPSK case
(Left from CCSDS standard, right simulation)

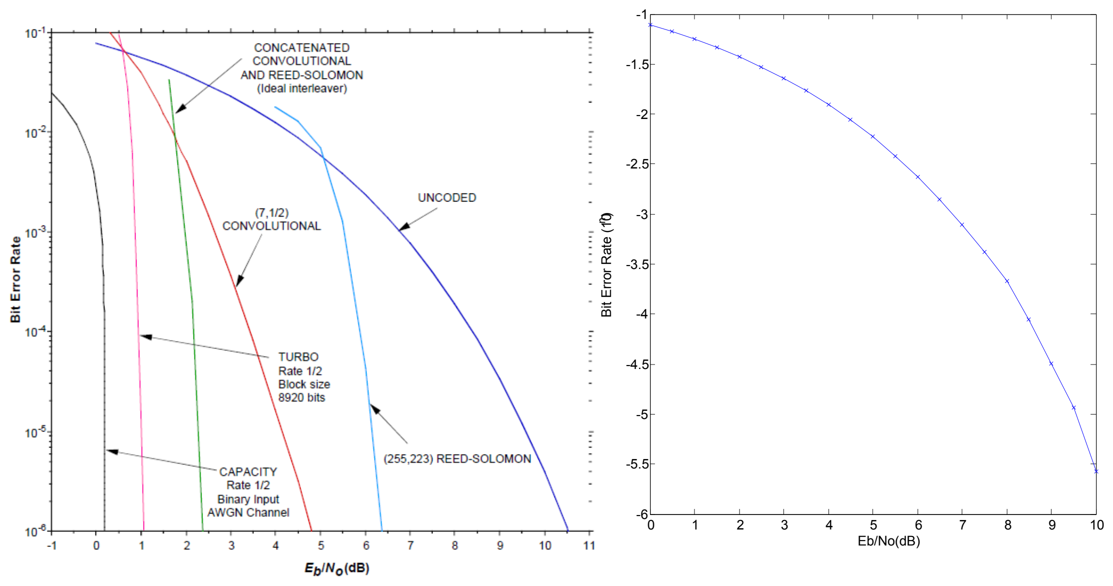


Figure 6: Uncoded BPSK BER performance

3.2 Main coding techniques used in CCSDS standard

CCSDS standard use the following techniques (CCSDS 130.1-G-1), (ECSS-E-ST-50-01C):

- Convolutional coding
- Reed-Solomon coding
- Concatenated codes: Reed-Solomon and convolutional
- Turbo Codes

This implementation model includes convolutional codes, Reed-Solomon and concatenated codes

3.3 Convolutional coding

Convolutional encoder

A rate r=1/n convolutional encoder is a linear finite-state machine with one binary input, n bit outputs and an m-stage shift register. Such a finite-state encoder has 2m possible states. The constraint length K of the convolutional code is defined as K=m+1, and the code is referred to as a (K,1/n) code. In comparison to block codes, convolutional codes encode the input data bits continuously rather than in blocks.

In general, a rate r=l/n convolutional encoder is a linear finite-state machine with l binary inputs and n binary outputs. A rate r=l/n code can also be produced by puncturing (leave some bit out) a convolutional code of rate r=1/n. (Figure 7)

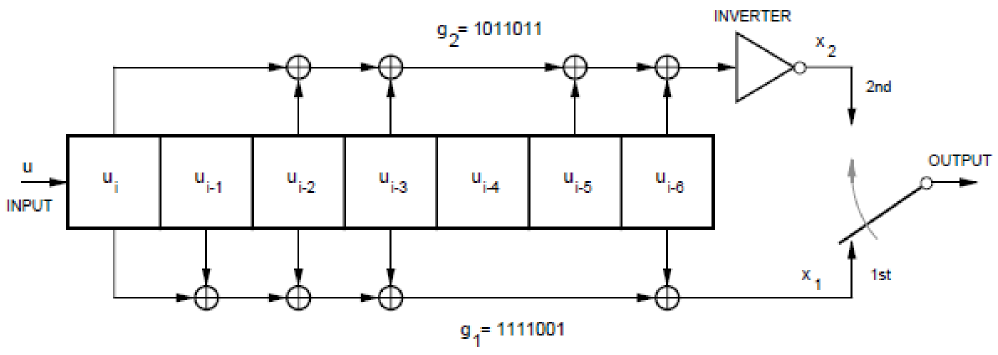


Figure 7: The basic structure of a (7,1/2) convolutional encoder (used in CCSDS standard)

Viterbi decoder

The used decoder is the so called Viterbi decoder structure (G. D. Forney, 1973). Viterbi decoder is a maximum likelihood decoder. Although Viterbi decoder is continuous the principle is easier understandable through sequences. Without bit failures not all variations of sequences is possible at the receiver, because of the convolutional encoder. Without any bit failure at the decoder, the convolutional code can be decoded with a simple state machine. If there are bit failures, then it has to be investigated which sequence from all possible is the nearest to the received sequence. The decoding process can be done with soft decisions too. The main idea behind this is that to each received symbol is related to a likelihood value that it is 1 or 0, and this information can be used to improve the quality of the decoder.

Figure 8 shows the K=3 trellis (four states) typically used for Viterbi decoding.

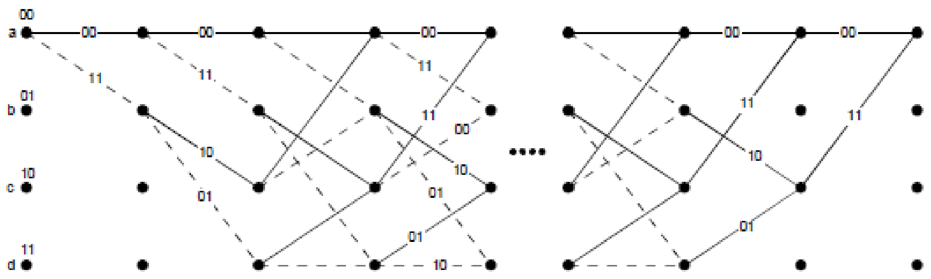


Figure 8: Viterbi decoder

In Figure 9 you can see the bit error rate vs. Eb/No from CCSDS standard with several decoding methods.

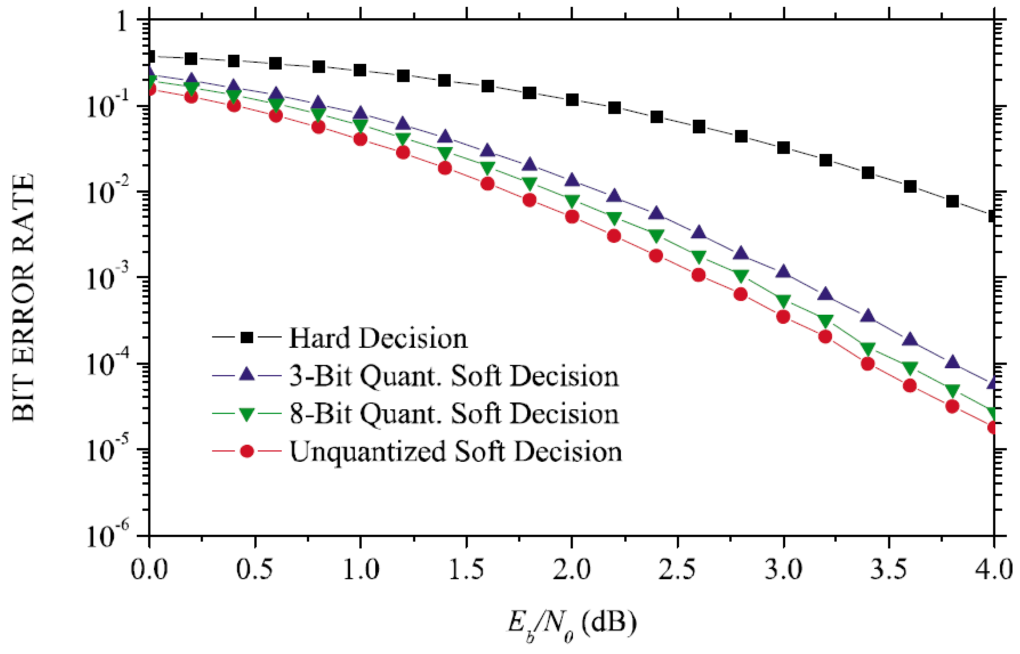


Figure 9: Error performance of a Viterbi decoder

Implementation of hard Viterbi decoder:

Hard decision means that the Viterbi decoder operates on binary digits delivered by the demodulator.(Figure 10)

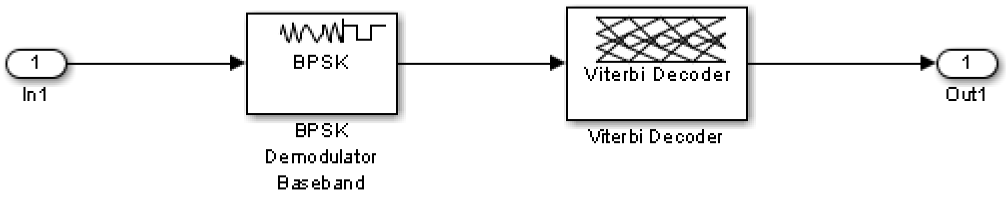


Figure 10: Hard Viterbi decoder

The model file of this structure:

`SIM_Convolution_hard.slx`

The used code in Matlab BER simulator:

```
% Convolutional code with hard decision
%
simname='SIM_Convolution_hard';
coderate=1/2;    %how much coded bit mean 1 bit information
modulation=1;    %how much bit is coded in one symbol
%}
```

The Graph:

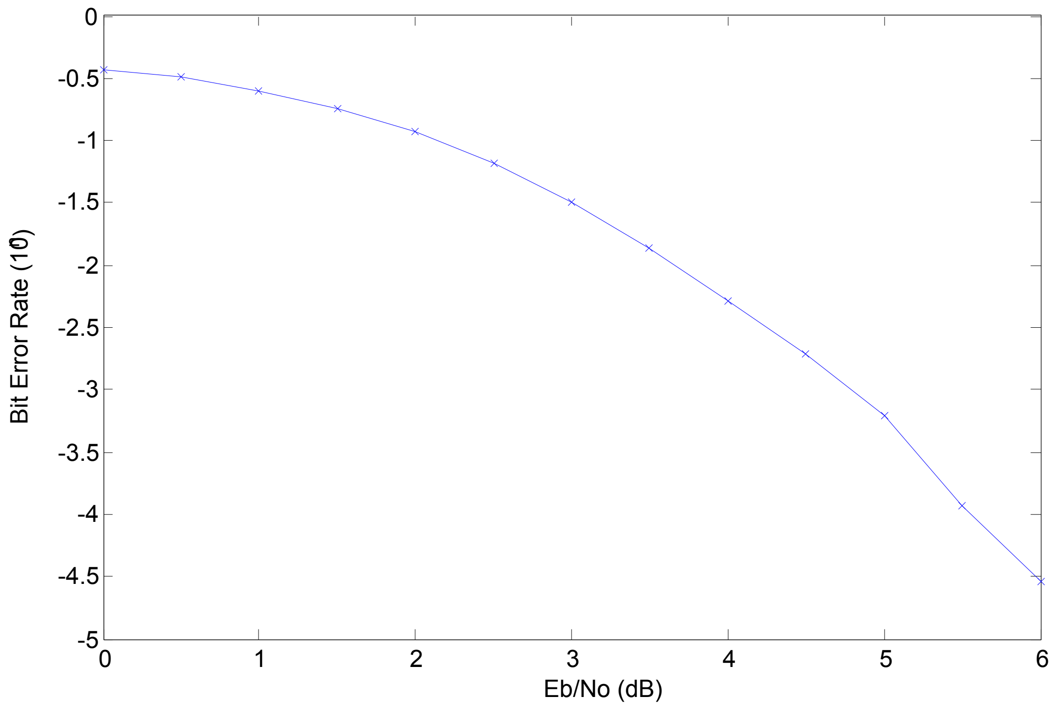
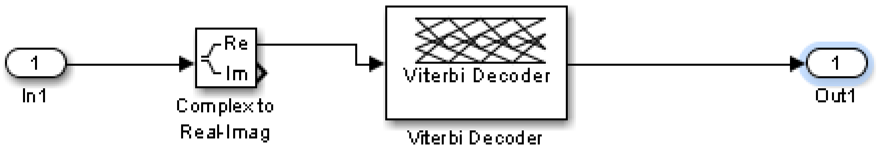


Figure 11: BER performance of hard decoder

Implementation of soft Viterbi decoder:

For soft decision decoding the demodulator does not make a decision. In this model there are no synchronization errors. Because of that a simple complex to real/imaginary block can be used as a demodulator.



The model file of this structure:

SIM_Convolution_unquantized.slx

The used code in Matlab BER simulator:

```
% Convolutional code with unquantized soft decision
%
simname='SIM_Convolution_unquantized';
coderate=1/2;    %how much coded bit mean 1 bit information
modulation=1;    %how much bit is coded in one symbol
%}
```

The Graph:

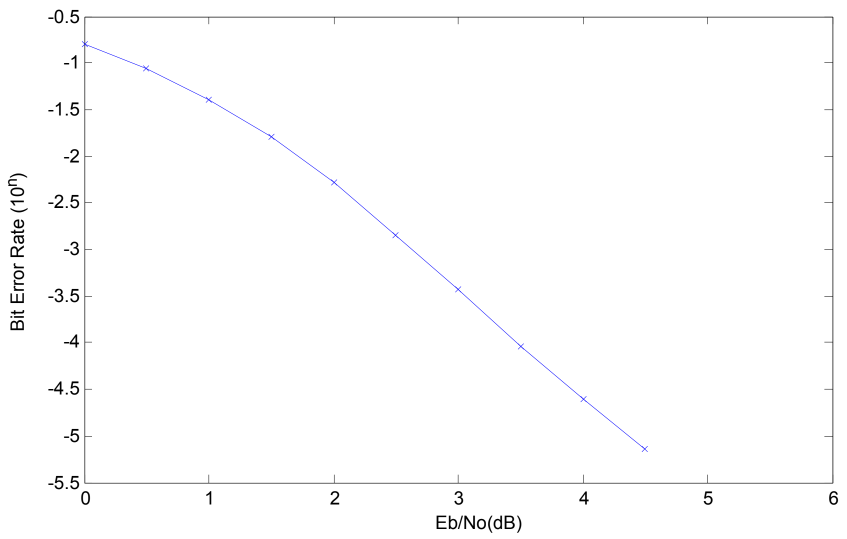


Figure 12: BER performance of non-quantized decoder

Implementation of quantized soft Viterbi decoder:

To reduce the computational cost, in general a quantized version of the signal is used for soft decision based decoding.

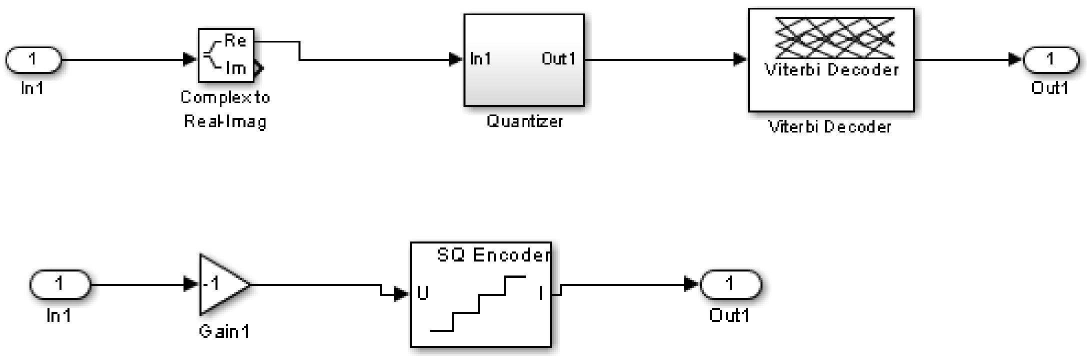


Figure 13:Quantized decoder

Boundary points of quantization: [-1:1/3:1]

The model file of this structure:

`SIM_Convolution_soft.slx`

The Graph:

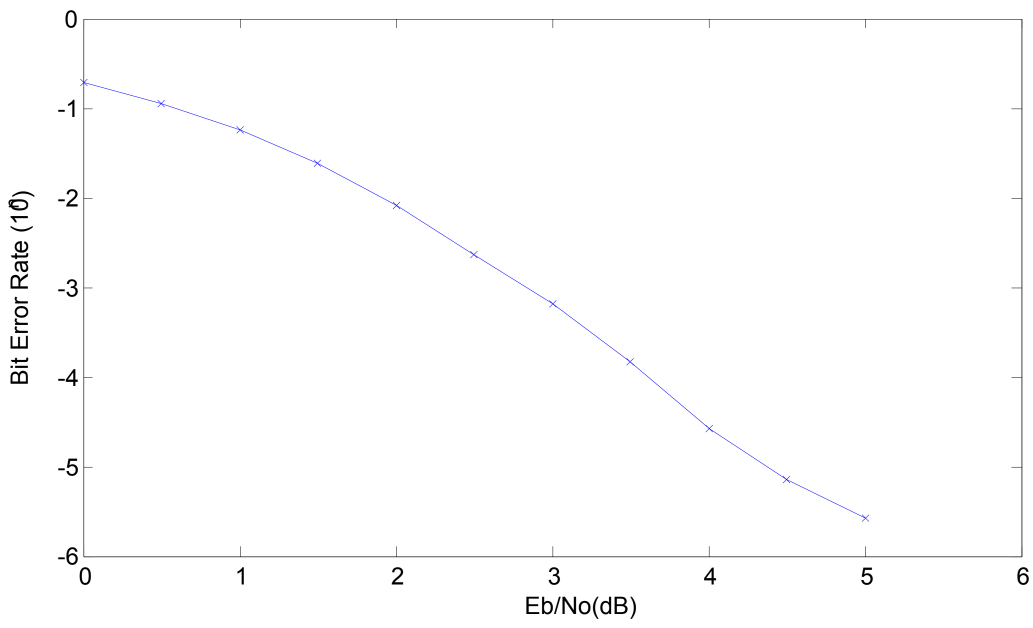


Figure 14: BER performance of quantized decoder

This case the graph has some small differences to the graph from CCSDS standard. It is because the boundary points are not the same.

3.4 Reed-Solomon code

Reed-Solomon codes are block codes. The (n,k) Reed-Solomon code code has an input sequence length of k, an output sequence length n, and a coderate k/n. The Reed-Solomon code can identify n-k errors, and can correct (n-k)/2 errors. The basics of Reed-Solomon codes are easier to understand in frequency domain. (Wikipedia, 2014)

Encoding: The sequence has to be Fourier-transformed, and then extended with a zero word, and then transformed back into time domain. (Figure 15)

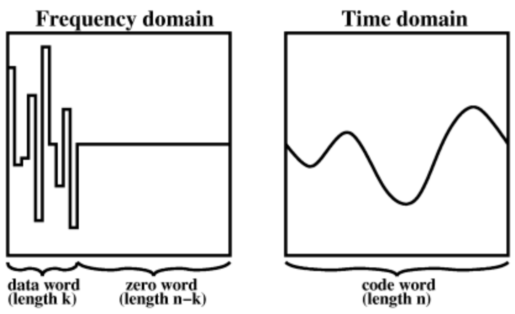


Figure 15: Reed-Solomon encoding (from: Wikipedia)

Decoding: The received sequence has to Fourier-transformed, and then from the last bits the error failures can be corrected. (Figure 16)

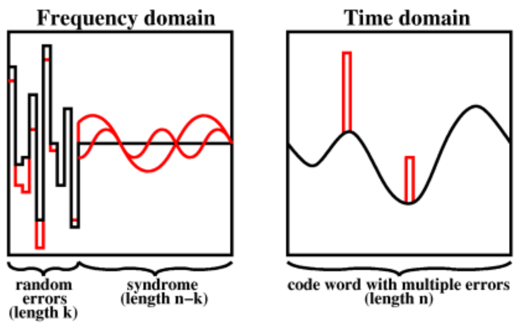


Figure 16: Reed-Solomon decoding (from: Wikipedia)

In the simulation (255,223) Reed-Solomon code is used, it can be set in blocks_config.m . It can correct 32/2=16 errors. The length of bit pattern has to be partible an integer multiple of k=223).

In figure 17 you can see the BER vs. Eb/No from CCSDS standard with (255,223) Reed-Solomon coding.

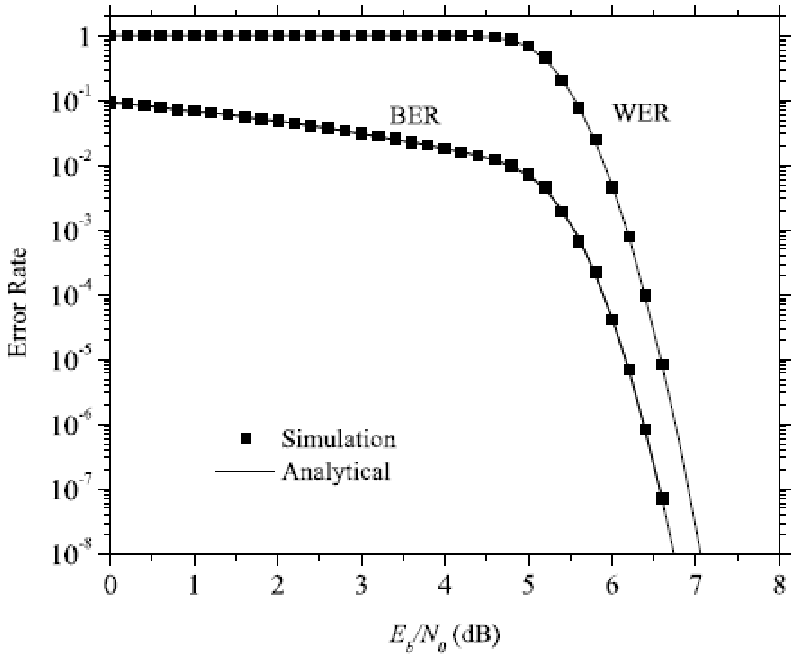


Figure 17: Reed Solomon error correction performance

The model file of this structure:

```
SIM_Reed_Solomon.slx
% Reed-Solomon
simname='SIM_Reed_Solomon';
coderate=RS_K/RS_N;   %how much coded bit mean 1 bit information
modulation=1;   %how much bit is coded in one symbol
```

The Graph:

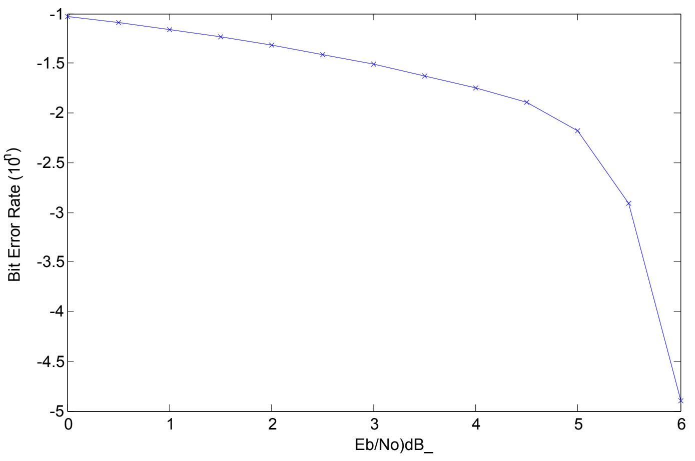


Figure 18: Simulated BER performance

3.5 Concatenated codes

Concatenated codes are useful methods to build scheme, with manageable decoding complexity. CCSDS standard use Reed-Solomon outer coder, and convolution inner coder with Viterbi decoding.

Transmitter model structure:

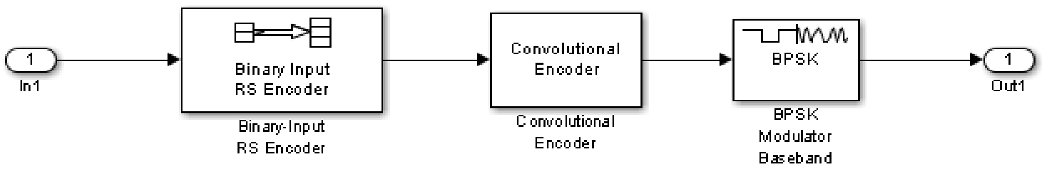


Figure 19: Transmitter model structure

Receiver model structure:

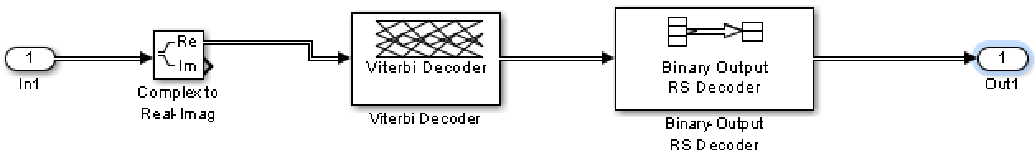


Figure 20: Receiver model structure

In Figure 21 you can see the BER to Eb/No from CCSDS standard with Reed-Solomon outer code, and convolution inner code with Viterbi decoding:

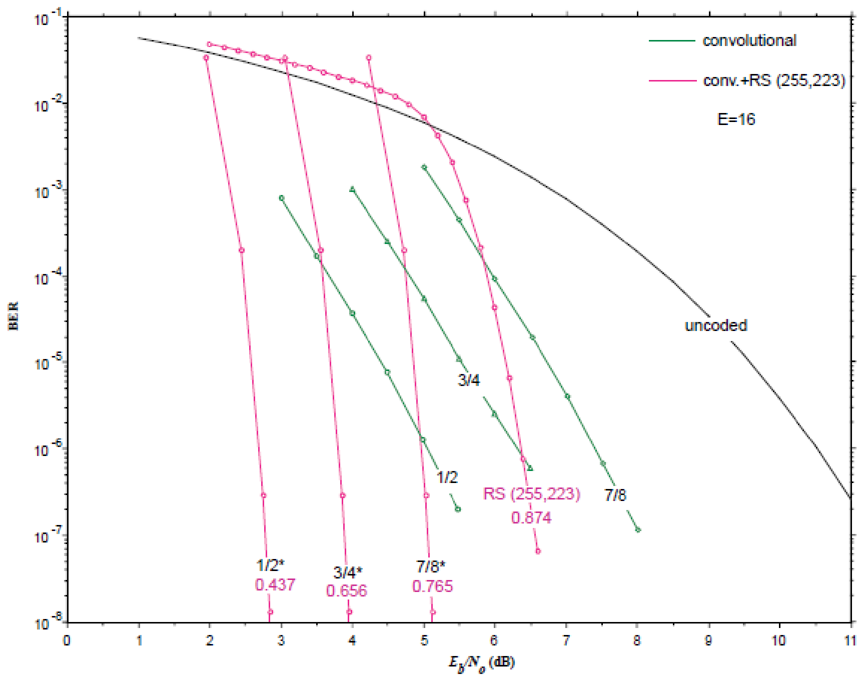


Figure 21: BER performance of concatenated codes

For simulation a (7,1/2) convolution code is chosen as inner coder, and (255,223) Reed-Solomon coder as outer code.

The models file of this structure:

```
SIM_Concatenated.slx
```

The used code in Matlab BER simulator:

```
% Concatenated codes: Reed-Solomon and Convolutional
%
simname='SIM_Concatenated';
coderate=RS_K/RS_N*1/2;   %how much coded bit mean 1 bit information
modulation=1;   %how much bit is coded in one symbol
%}
```

The Graph:

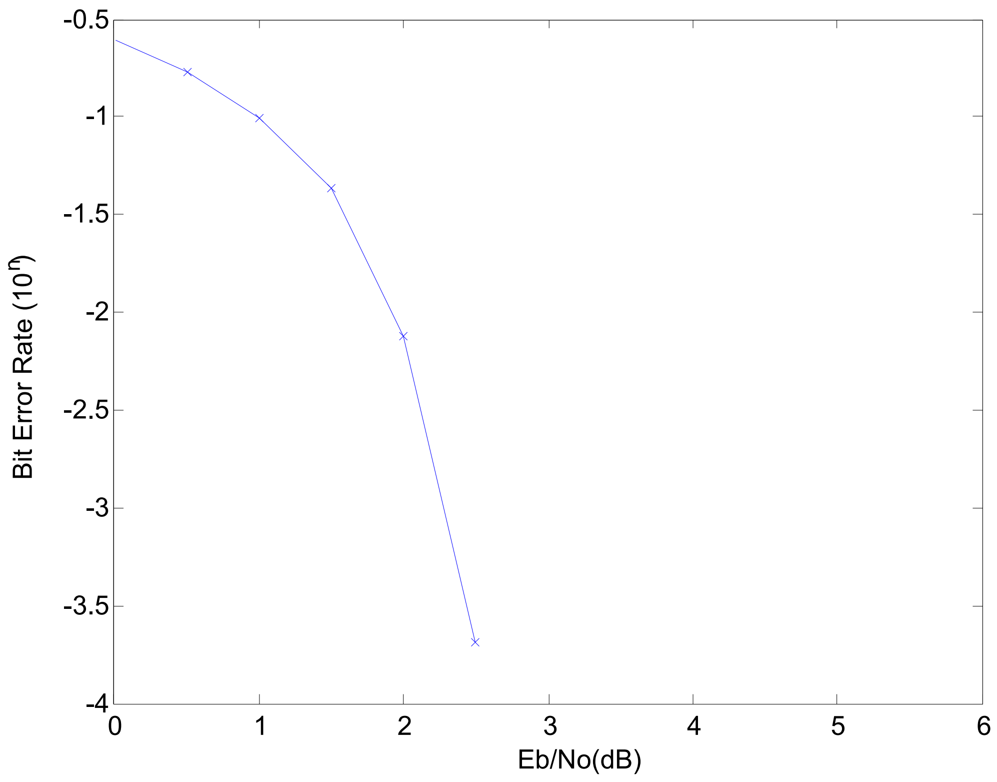
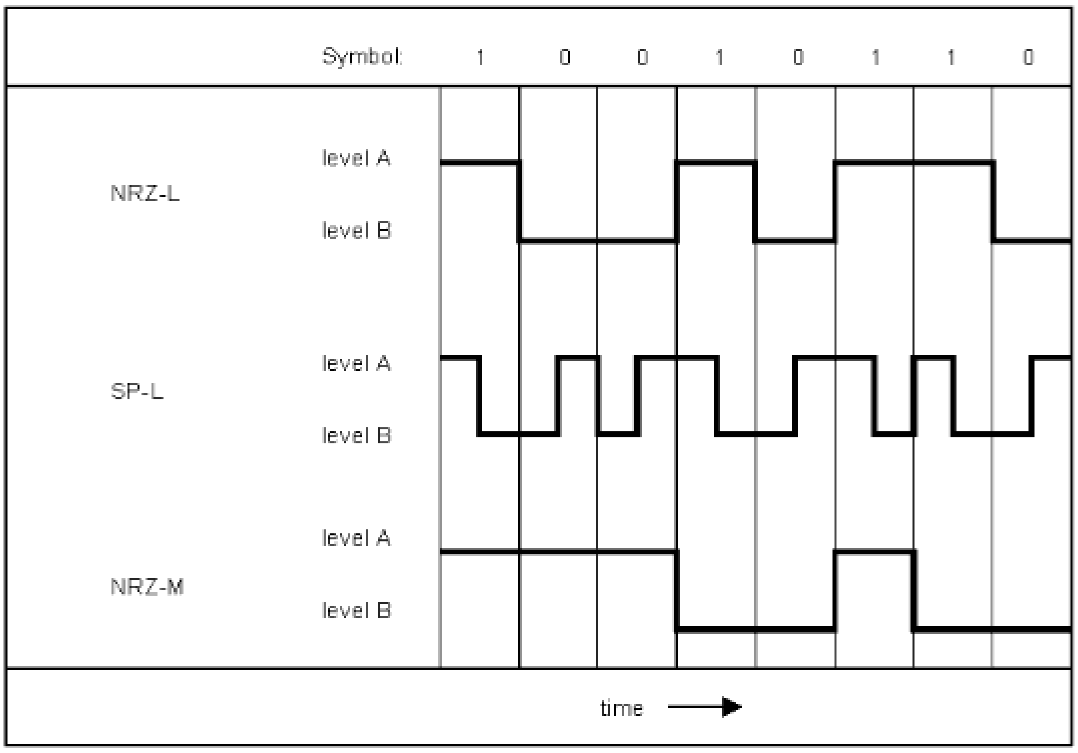


Figure 22: Simulated BER performance

4. Modulation

4.1 Pulse code modulation

To transmit a bit sequence, first it has to be associated with a signal. This is called pulse code modulation (PCM). The digital signal is represented by an amplitude-time-discrete signal. There are many solutions, from these the CCSDS standard use 3 methods (ECSS-E-ST-50-05C):

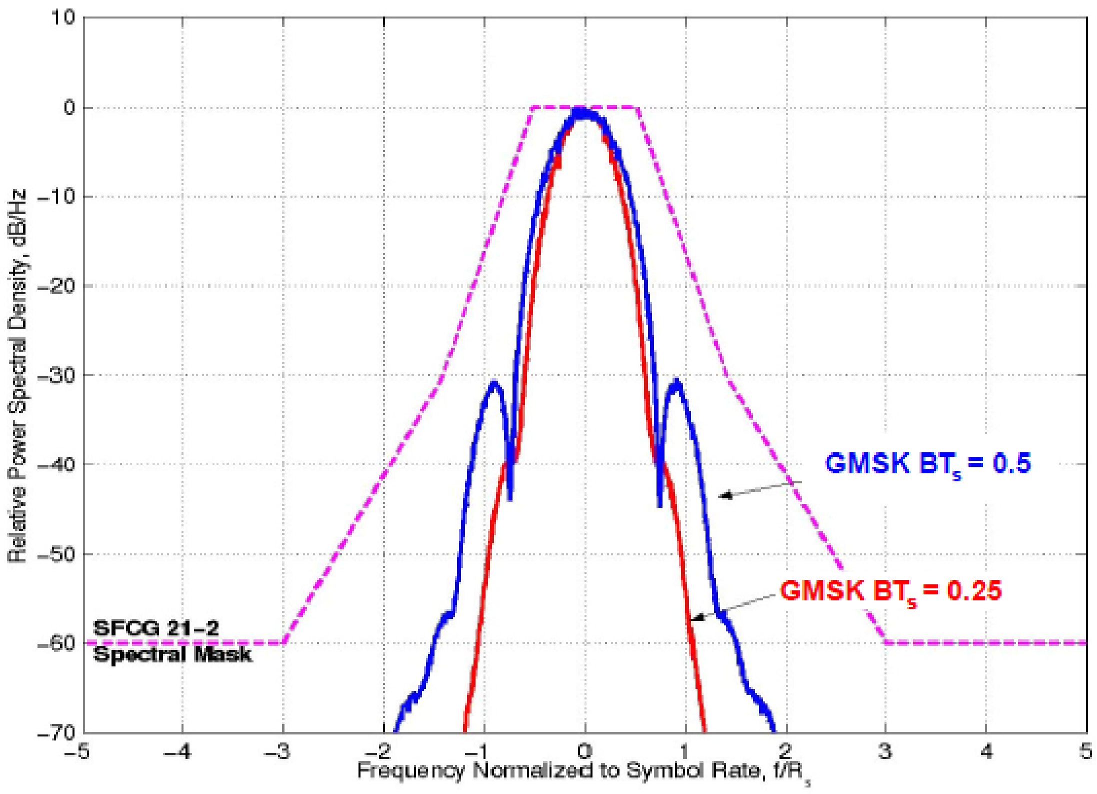


NRZ–L	level A signifies symbol "1" level B signifies symbol "0"
SP–L [14]	level A during the first half-symbol followed by level B during the second half-symbol signifies symbol "1" level B during the first half-symbol followed by level A during the second half-symbol signifies symbol "0"
NRZ–M	level change from A to B or B to A signifies symbol "1" no change in level signifies symbol "0"

23. Figure: Pulse-code modulation in CCSDS standard

4.2 Pulse shaping

In a transmission system, reduction of the used bandwidth is one of the most important tasks. To reduce the bandwidth, signal filters are used. The main purpose of these filters is that they have to reduce the needed bandwidth without disturbing the demodulation process (for example by intersymbol interference).
In space communication, the recommendations of Space Frequency Coordination Group (SFCG) maximizes the usable bandwidth for each band with spectral masks (http://www.sfconline.org). For an example see Figure 24.



24. Figure: Spectral mask

Interpolation

The basic method is to make a continuous baseband signal from the PCM pulses by usage of an ideal digital-analog converter (DAC), and an ideal low pass filter. The main problems are that no ideal DAC, and no ideal lowpass filter exists.
In practice the signal has to be upsampled to a higher sample per symbol rate. This can be done by inserting zeroes between the points of the signal. Then the signal can be shaped by a pulse shaping filter (Figure 25).

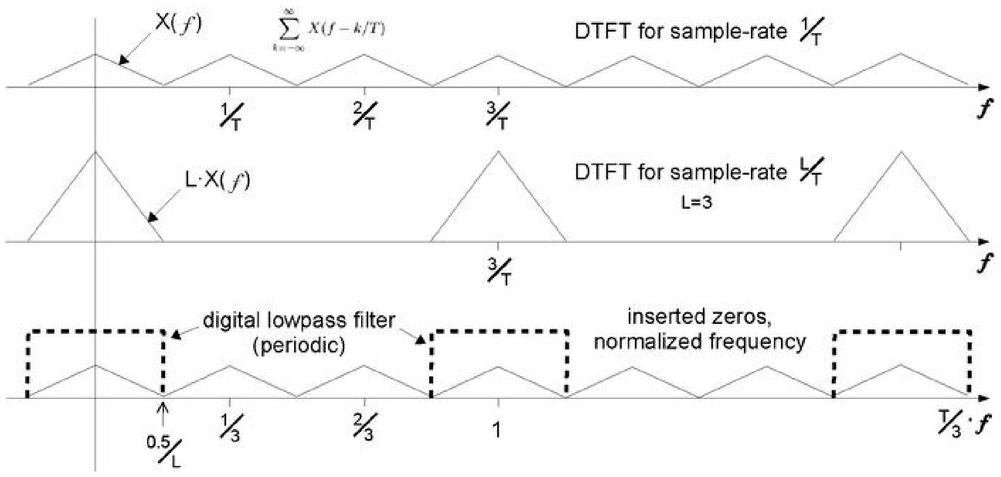


Figure 25:Interpolation

Method of simulation

All simulations can be find in Filter_simulation.m file.

For simulation of upsampling, and filter structures, the NRZ-L pulse code modulated signal is used:

```
T_bit_pattern=(T_bit_pattern-0.5)*2;
bit_pattern = [T_bit_pattern ; zeros(40,1) ];
```

The end of the signal has to contain zeros to be able to simulate the decay of the filter.

Configuration

The signal is constructed from T_bit_pattern with NRZ-L pulse-code modulation. For each filter, a signal is upsampled with zero-fill interpolation with number of zeroes Upsample_factor-1 between each symbol. In practice an upsample factor 2 to 8 is used.

```
    Upsample_factor=30;
    % Bit pattern to send uncomment the used one
    T_bit_pattern=[1]; %impulse
    %T_bit_pattern=[1;0;1;1;0;0;1;1;1];
```

Measures

The plot of the magnitude response of the filter is made with use of Matlab fvtool. The response of the filter with some input signal is simulated and plotted.

4.2.1 Root-raised cosine filter

The most frequently used filter for pulse shaping is a root-raised cosine filter. It is derived from the raised cosine filter. The basic idea behind the raised cosine filter is that first Nyquist criterion is satisfied (J. G. Proakis, 1995) Figures 26 and 27 show the impulse response of a raised cosine filter (upsample factor: 30, filter length: 15, rolloff factor: 0.8):

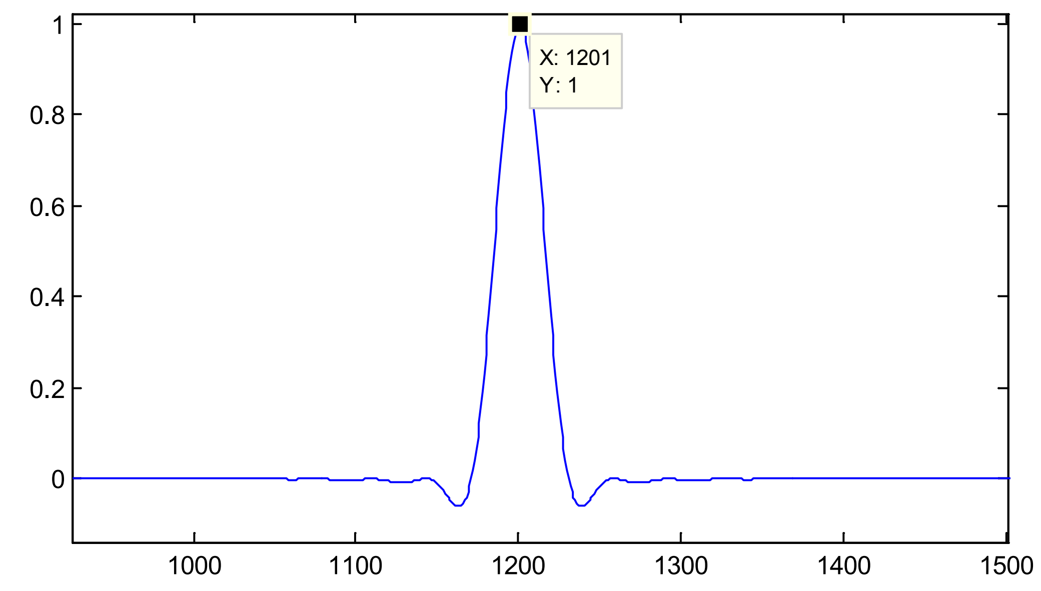


Figure 26: Impulse response of a raised cosine filter

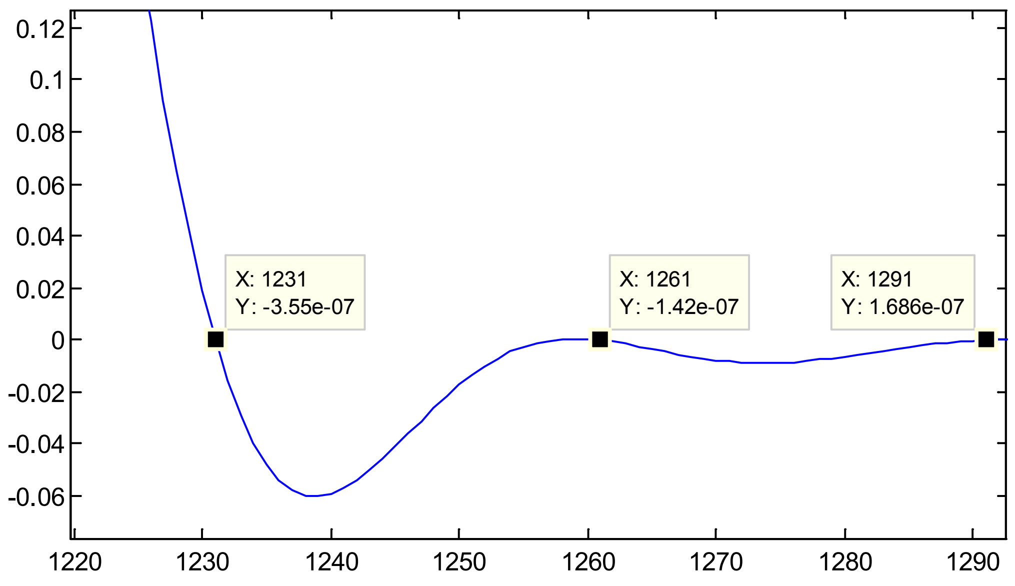


Figure 27: Intersymbol interference with raised cosine shaped pulses

The raised cosine filter is split into two filters to maximize SNR at receiver (matched filter theory). The resulting filter is called root-raised cosine (RRC) filter, because the frequency magnitude response is the root of the raised cosine filter (Figures 28 and 29).

Impulse response:

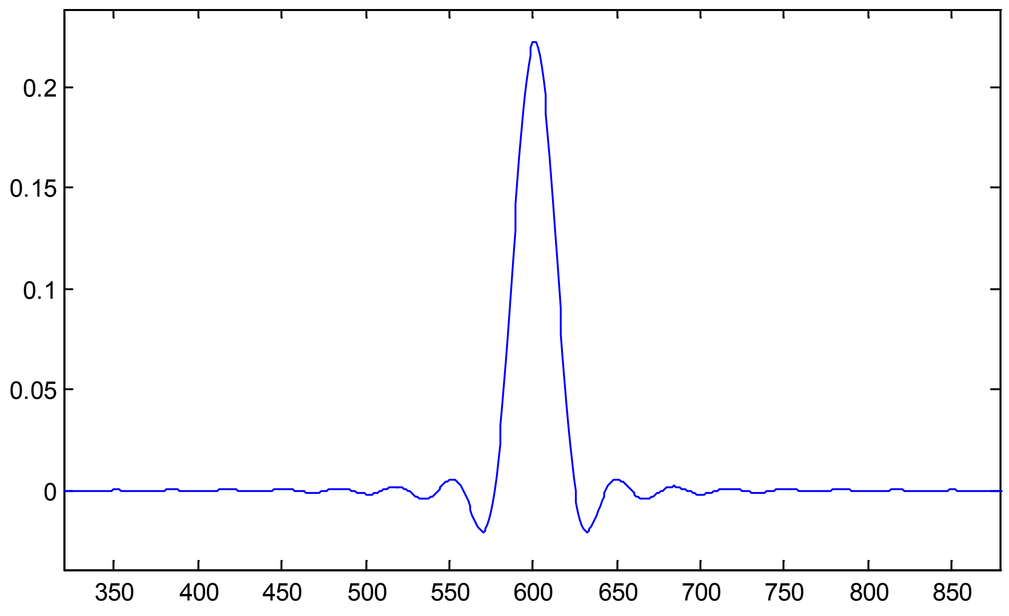


Figure 28: Impulse response of a RRC filter

Magnitude Response:

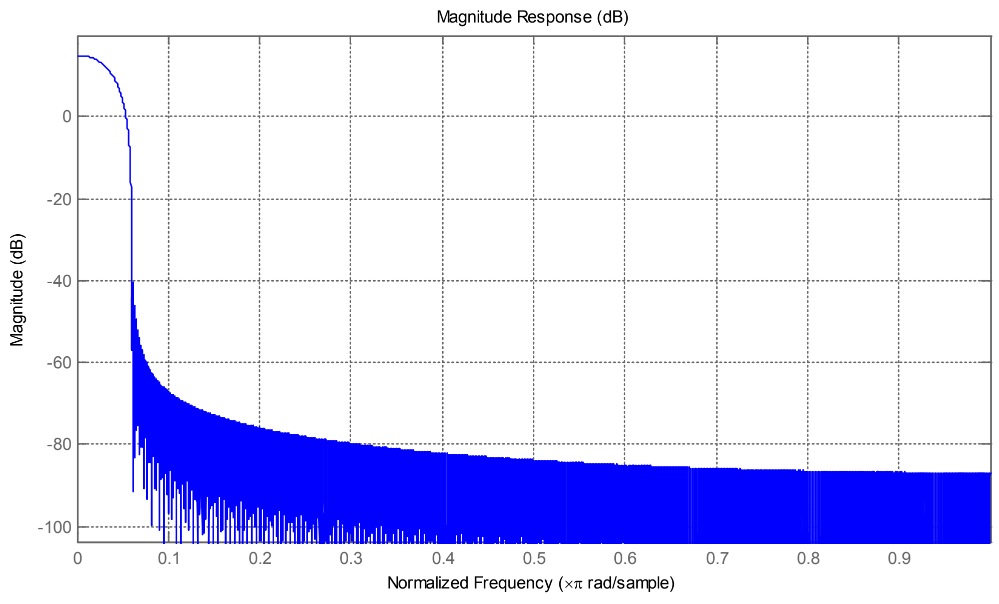


Figure 29: Magnitude response of a RRC filter

An example of the signal at receive filter, to prove the concept (after the root raised cosine filter) is provided in Figure 30:

(`T_bit_pattern=[1;0;1;1;0;0;1;1;1];, NRZ`)

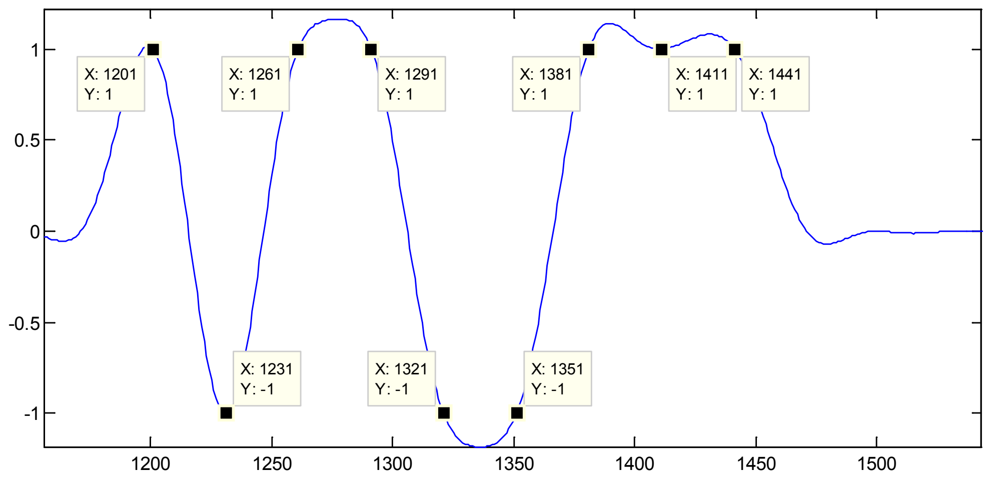


Figure 30: Signal after receiver RRC filter

The structure of the simulation:

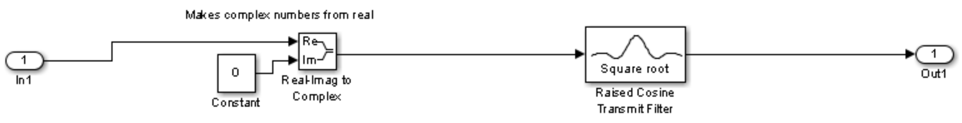

Makes complex numbers from real
1
In1
0
Constant
Re
Im
Real-Imag to Complex
Square root
Raised Cosine Transmit Filter
1
Out1

Figure 31: Structure of raised cosine simulation

The used code:

```
% Root Raised Cosine filter
%
simulation='SIM_Filter_RRC';
T_Rolloff_factor=0.8;
Group_delay=20; % delay in symbols (filterlength)


%}
```

4.2.2 IIR filter

For pulse shaping, all kinds of filter can be used. Although they are not as efficient as raised cosine, they are usable, and sometimes they have some advantage in practice. The continuous form of IIR filter is realizable with analog techniques.
This simulation (Figure 32) includes a simulator for IIR analogue filters. The coefficients are generated by [B,A] = butter(order,cutoff frequency). It has to be transformed to discrete domain by use of bilinear transformation.

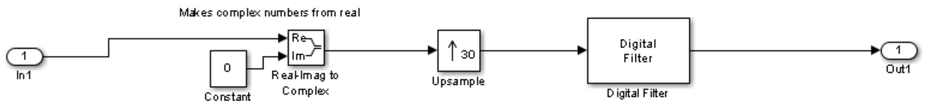


Figure 32: IIR filter simulation

The used code:

```
% IIR filter
%
  simulation='SIM_Filter_IIR';
  [TX_filter_coeff_b,TX_filter_coeff_a] = butter(3,0.03);
%}
```

The main purpose of the simpulation with this coeffients is to show the drawbacks of this structure. It is of course in this form not usable for any communication system. The answers are not normalized to transmitted signal power!

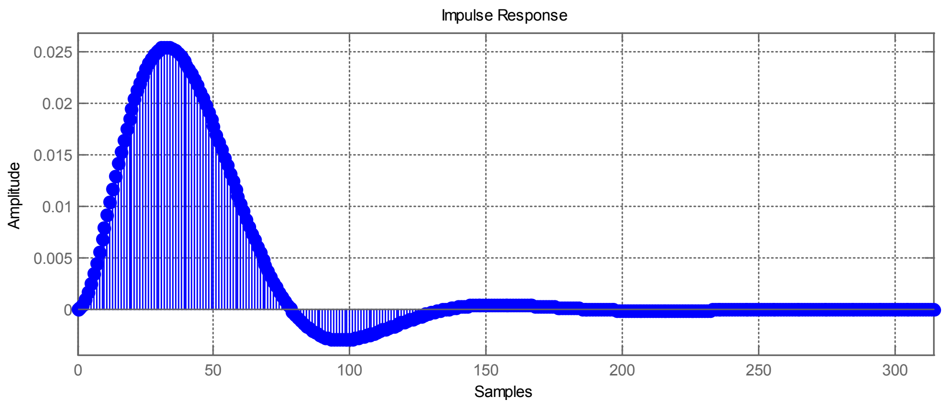


Figure 33: Impulse respond of an example IIR filter

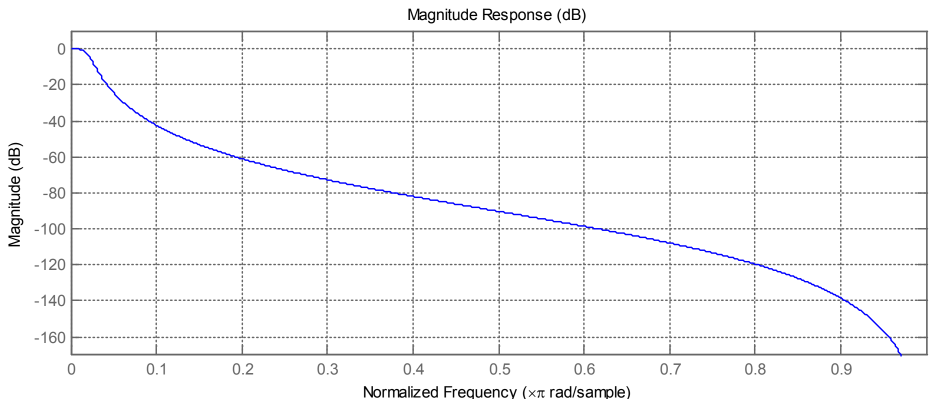


Figure 34: Magnitude response of an example IIR filter

The filtered signal illustrates the same bit pattern as in Figure 30. It can be seen that there is a great amount of ISI (Figure 35).

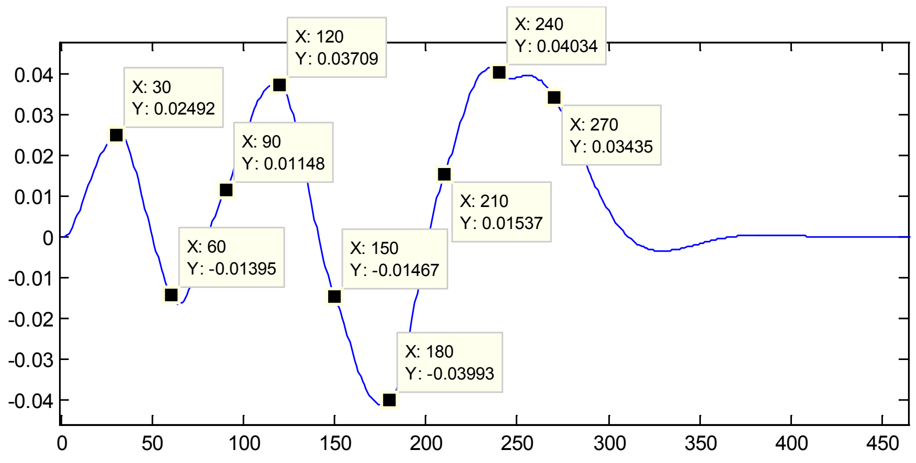


Figure 35: ISI of an example IIR filtered signal

4.2.3 Gaussian filter

The basic idea of Gaussian filter is to make a filter with the minimum possible group delay. The received signal is not ISI free, but the length of the filter has the minimal length in time domain. (H. J. Blinchikoff, A. I. Zverev, 2001)
The use of Gaussian filter as channel filter comes from structures, where the signal before the filter and the filter itself were analog. The signal before the filter in this structure has the form of square signals. In this simulation it is made by an FIR filter, which has an appropriately selected length by the upsample factor.

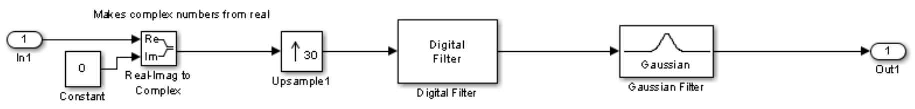


Figure 36: Simulation structure

The used code:

```
% Gaussian filter
simulation='SIM_Filter_Gauss';
T_Gauss_BTproduct=0.5;
T_Group_delay=2;
TX_filter_coeff_a=1;
```

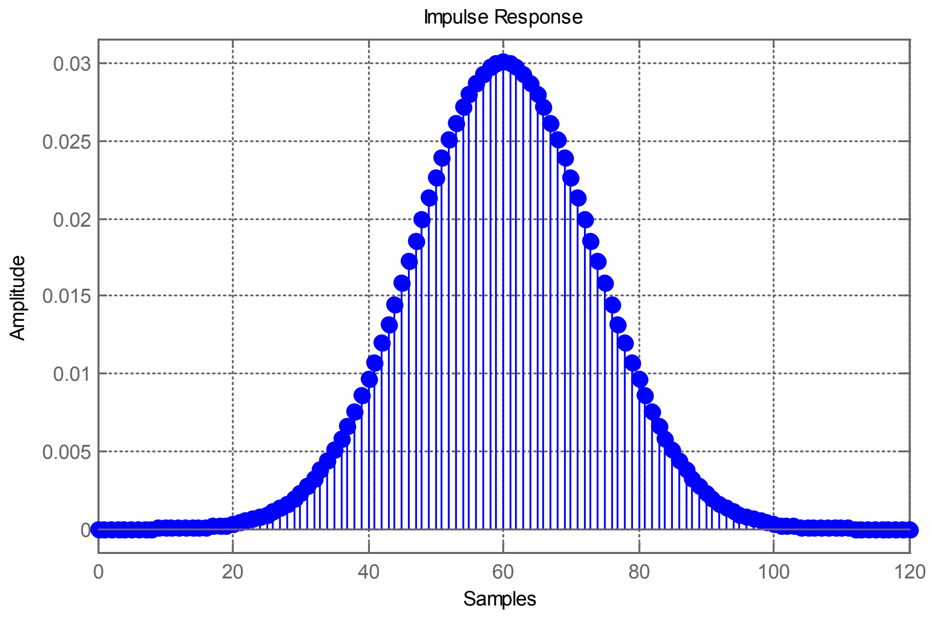


Figure 37: Impulse response of an example Gaussian filter

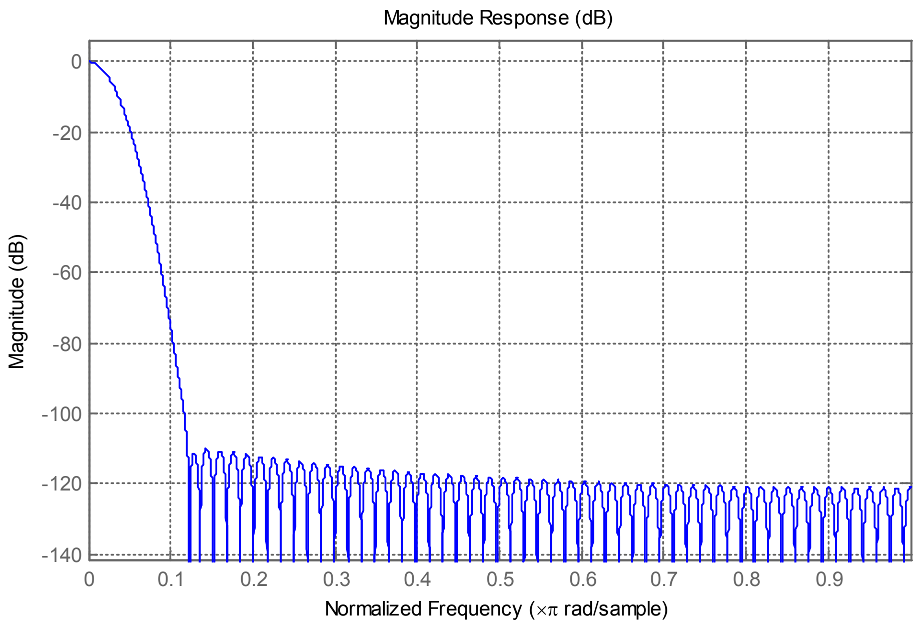


Figure 38: Magnitude response of an example Gaussian filter

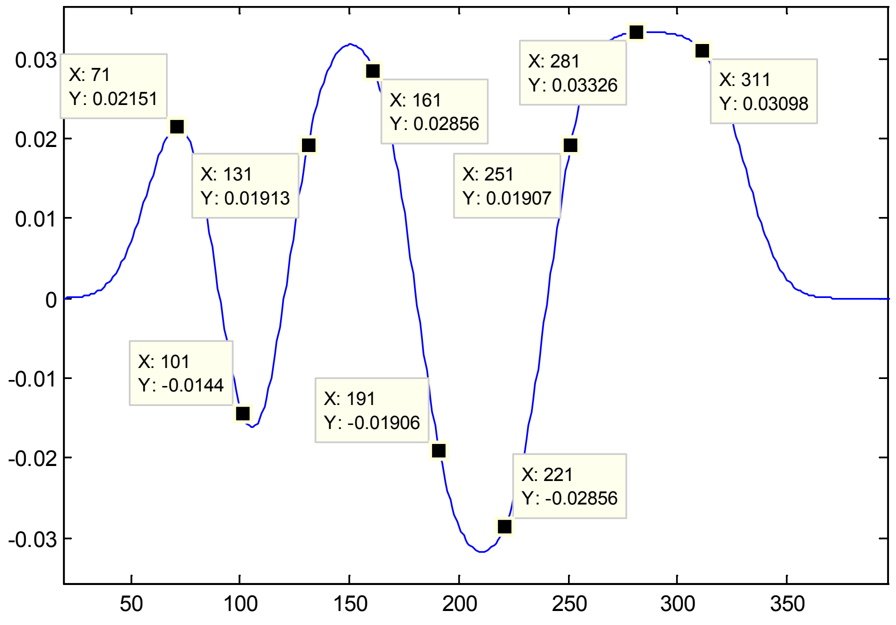


Figure 39: An example filtered signal (T_bit_pattern=[1;0;1;1;0;0;1;1;1];, NRZ)

4.3 Modulation

In CCSDS standard, phase modulations are used to reduce the effects of non-linear power amplifiers on the signal.

The basic modulation formats (CCSDS 413.0-G-2) (ECSS-E-ST-50-05C) are as follows:

- Phase shift based modulations :BPSK,QPSK,OQPSK, 8PSK TCM
- Continuous Phase modulations: MSK, GMSK

Basic properties of simulation

For simulation first a series of simulation are run on SIM file to measure SNR performance, then the last simulation is done with high SNR (no noise addition) to be able to see the signal properties.

Configuration

Each simulation needs an Upsample_factor, and the number of generated bits. For PSD BER measurements, a high number of bits is recommended, for time domain measurements there is no need for many symbols. Each simulation has its own variables too.

Measurements

For these measurements the signal after the transmitter is used without any noise or effects of the channel.

```
% Constellation
figure('name','Constellation')
plot(signal_at_transmitter(1:end),'.')
xlabel('Real');
ylabel('Imag');

% Time domain
figure('name','signal_at_transmitter')
plot(imag(signal_at_transmitter(1:end)))
ylabel('Imaginary axis of the signal');
xlabel('Sample');

% Phase in Time domain
figure('name','signal phase at transmitter')
plot(angle(signal_at_transmitter(1:end)))
```

```
ylabel('Phase');
xlabel('Sample');

% PSD
figure('name','PSD')
[Pxx,W] = pwelch(signal_at_transmitter);
W=(W-pi)*(Upsample_factor/(2*pi)); %normalized to 1/symbol
plot(W,10*log10(fftshift(Pxx))-max(10*log10(fftshift(Pxx))));
%Normalized to 0 dB maximum;
ylabel('PSD in dB');
xlabel('Frequency normalized to 1/Symbol');

% BER
figure('name','BER vs SNR')
plot(SNR_values, Error_ratio);
xlabel('SNR);
ylabel('Bit Error Rate');
```

4.3.1 M-ary phase shift keying

In PSK (L. W. Couch, 1997) systems the bit pattern is coded into the phase of a sinusoidal function. Each state belongs to a phase in the signal domain. CCSDS standard uses BPSK (1 bit per symbol, 2 states), and QPSK (2 bit per symbol, 4 states).

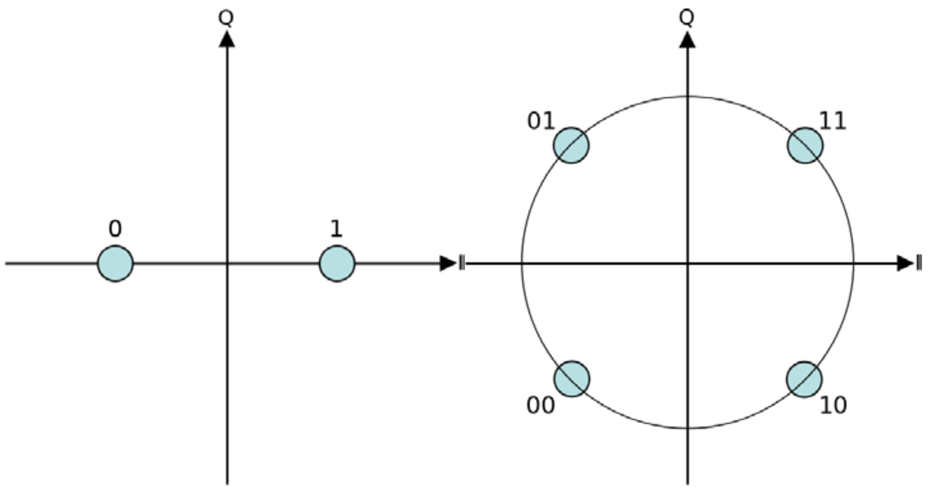


Figure 40: BPSK and QPSK symbol mapping (Wikipedia)

The signals are usually filtered by RRC filter. An example spectrum of a QPSK based transmitter can be found in Figure 41.

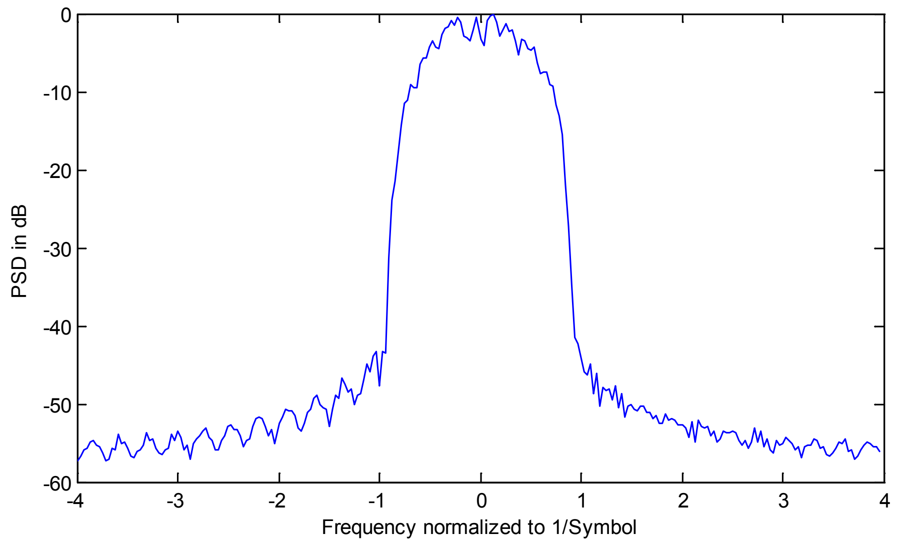
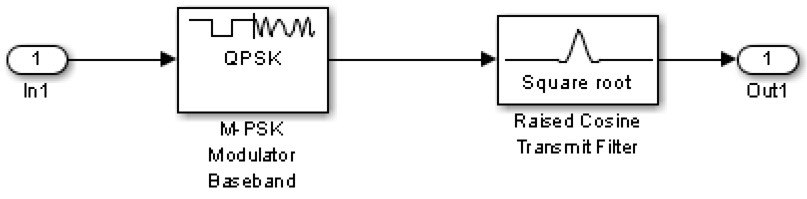


Figure 41: An example spectrum of a QPSK based transmitter

Code used for simulation:

```
% M-PSK
    simulation='SIM_MOD_M_PSK';
    M_PSK=4;
```

The model structure for transmitter:



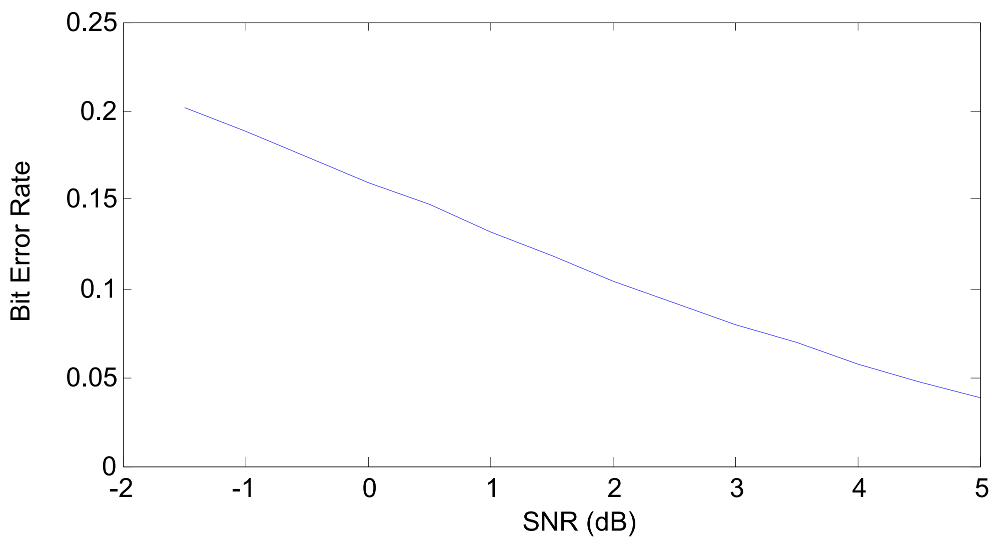


Figure 42: Example BER performance measurement

4.3.2 4D-8PSK-TCM

Usually coding and modulation are managed separately. The BER performance to noise ratio of the M-PSK and QAM like modulation methods depend on the Euclidian distance between the points. For coded schemes, Hamming distance between two possible bit sequences means about the same. The main idea behind TCM (Trellis Coded Modulation) (J. He, Z. Wang, H. Liu, 2010) is to manage these two together to reach a greater overall distance between two possible signal sequences.

The MPSK trellis coded modulation is based on the partitioning of one MPSK constellation into different subsets, each subset having a minimum Euclidian distance larger than the minimum distance of the original constellation. The structure of a 4D-8PSK TCM modulation used in CCSDS standard can be seen in Figure 43.

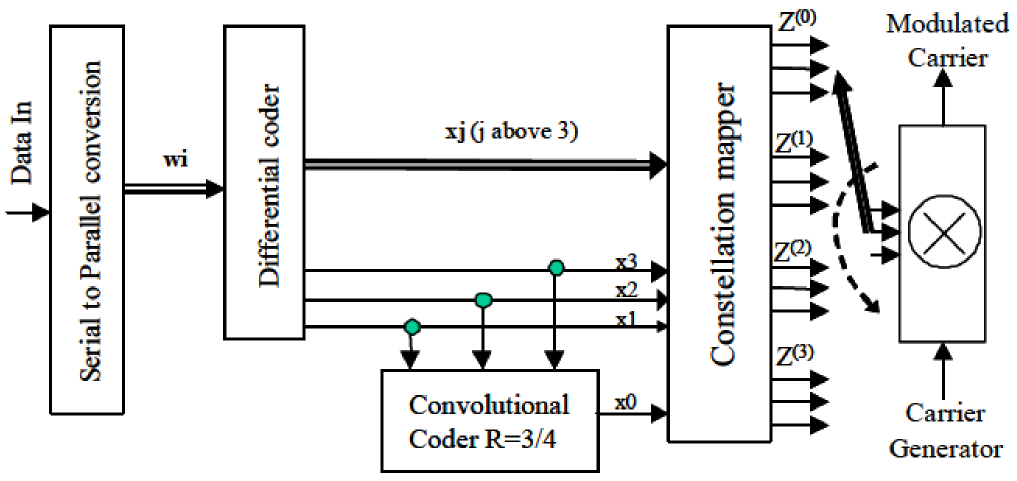


Figure 43: 4D-8PSK-TCM modulator

It has to be seen that 4D-8PSK-TCM is an 8PSK system with a special coding technique developed for 8PSK. Because of that, the transmitted signal, the pulse shaping methods, and the receiver structure is the same is as in 8PSK. (Figure 44)

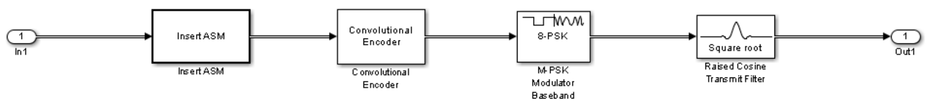


Figure 44: 4D-8PSK-TCM transmitter structure

4.3.3 Offset QPSK

One main drawback of QPSK is that there can be a 180° phase shift, which causes large amplitude fluctuations after the lowpass filter. One solution for this problem is to offset the signal in Q arm by half of the symbol rate. This method as it can be seen avoid the 180 grad jumps (Figure 45)

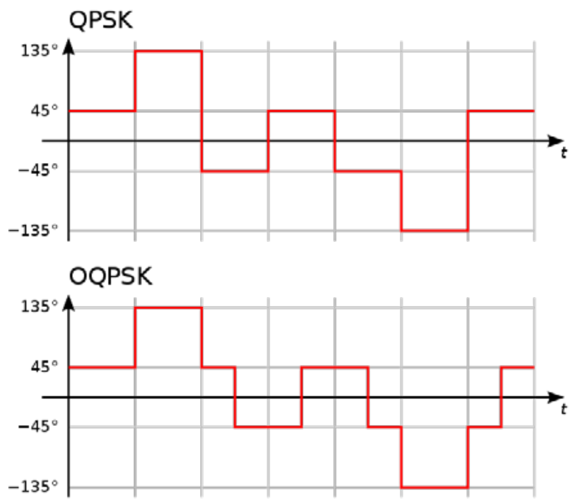


Figure 45: QPSK and OQPSK example

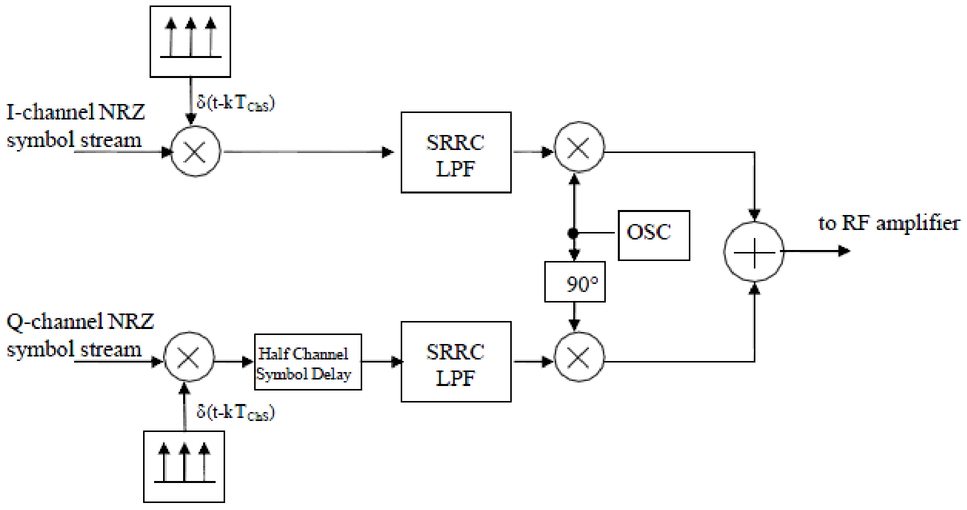

I-channel NRZ symbol stream
δ(t-kT_chs)
SRRC LPF
OSC
90°
to RF amplifier
Q-channel NRZ symbol stream
Half Channel Symbol Delay
SRRC LPF
δ(t-kT_chs)

Figure 46: Basic structure of OQPSK implementation (from CCSDS standard)

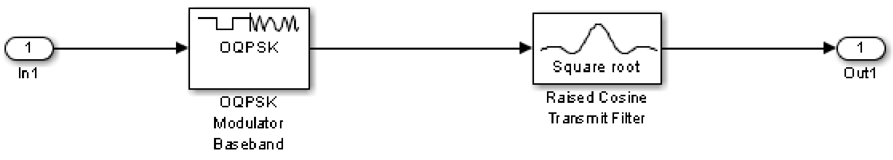


Figure 47: Basic structure of OQPSK implementation

Figure 48 comes from CCSDS standard, Figure 49 depicts the simulation results. It can be seen, that the OQPSK implementation, and my implementation produce comparable results, with RRC filter rolloff factor 0.5. Both implementations match the SFCG recommendations.

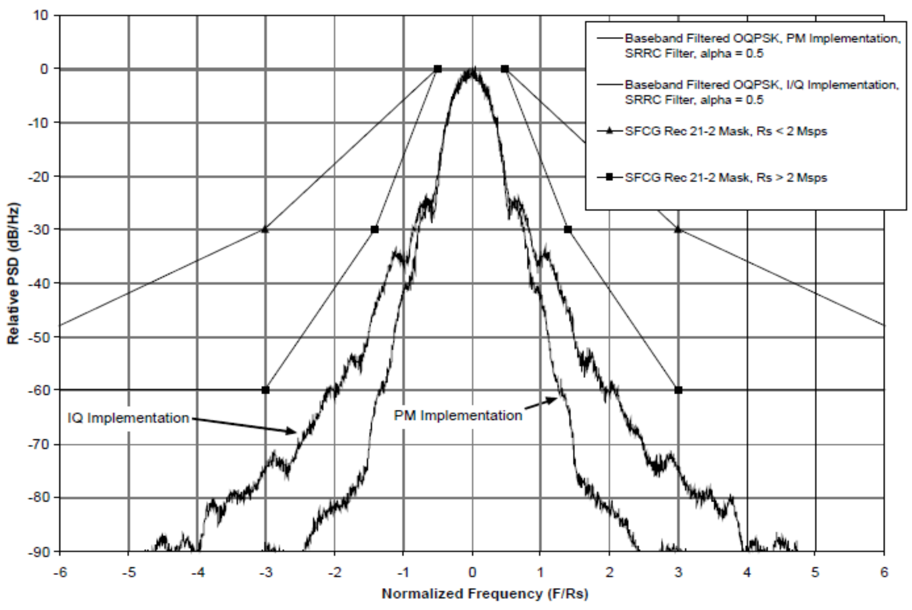


Figure 48: OQPSK PSD from CCSDS standard

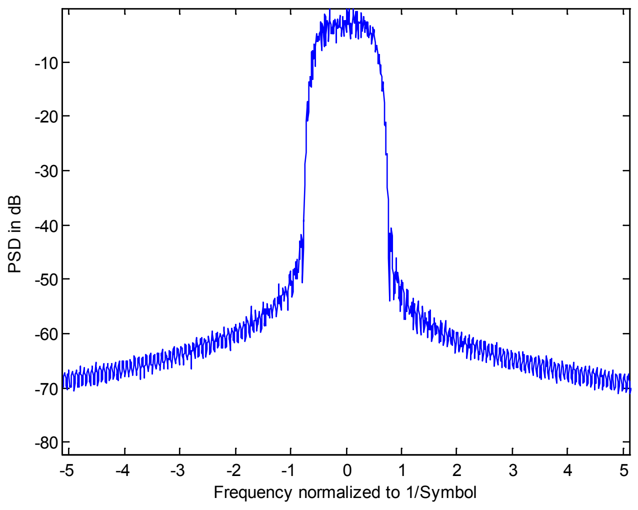
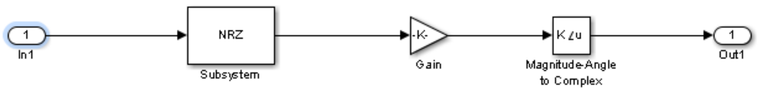


Figure 49: Floating point simulation with ideal SRRC filter (roll-off factor 0.5)

4.3.4 Phase modulation

In phase modulation, the PCM coded signal enters to a phase modulator. The advantage of this solution is that, that if the gain is smaller than π, the base-band signal has always a constant ingredient, which means the carrier is in the signal included.

The simulator:



The used code:

```
% PM
%
    simulation='SIM_MOD_PM';
    T_Phase_gain=1;
%}
```

The PSD of the transmitted signal:

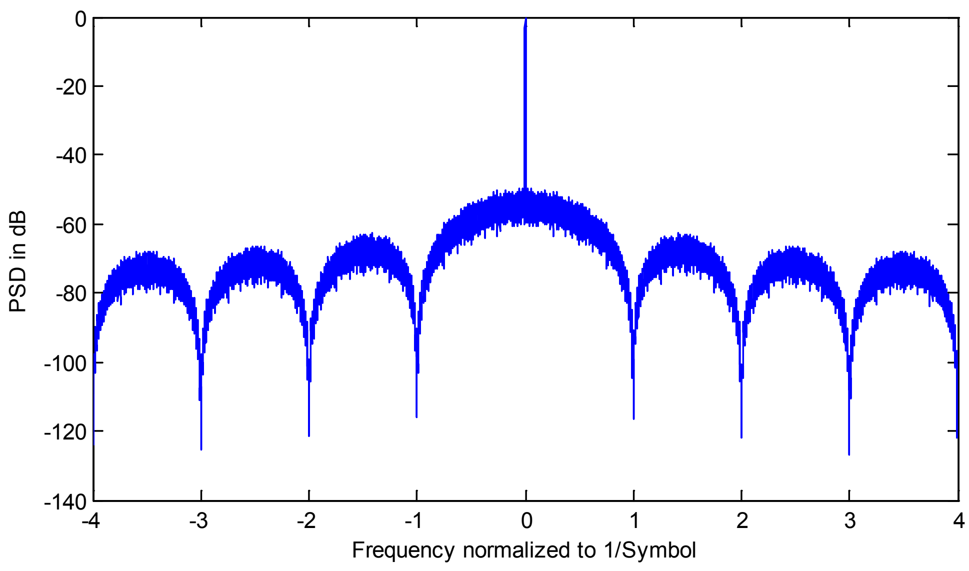


Figure 50: An example PSD of phase modulation

4.3.5 GMSK

GMSK is a sort of continuous phase modulation. GMSK is the filtered version of Minimal Shift Keying (MSK) (S. Pasupathy, 1979). In MSK each bit is coded with a half sinus. That is why there is no phase jump in the signal. The only difference in implementation of MSK and GMSK modulation is the use of a Gaussian filter in GMSK modulation. Some plots of MSK modulation in time domain can be seen in Figure 51 and Figure 52

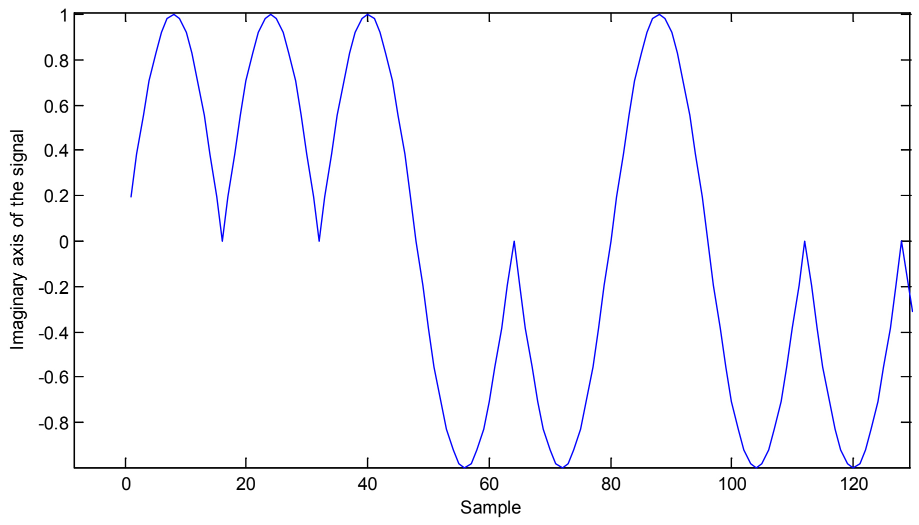


Figure 51: MSK modulation

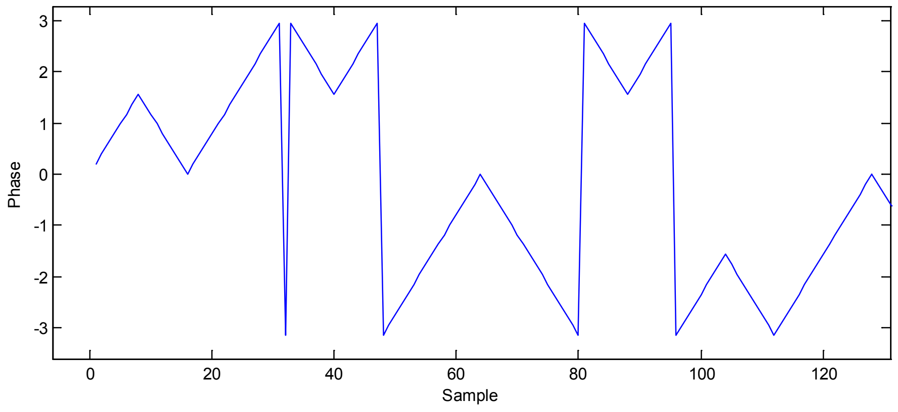


Figure 52: MSK modulation

The basic baseband implementation method (from CCSDS standard):

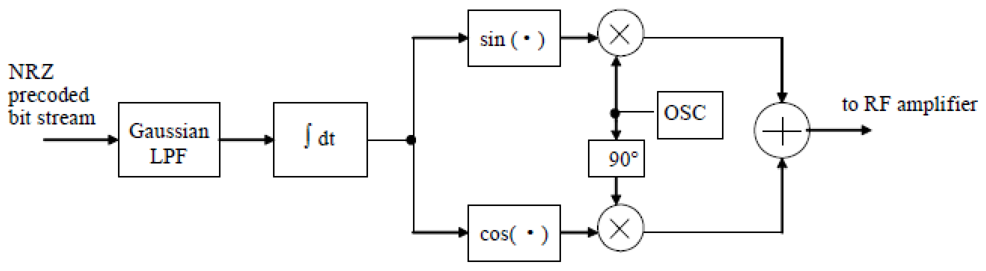


Figure 53: GMSK implementation (CCSDS standard)

The baseband implementation in simulator:

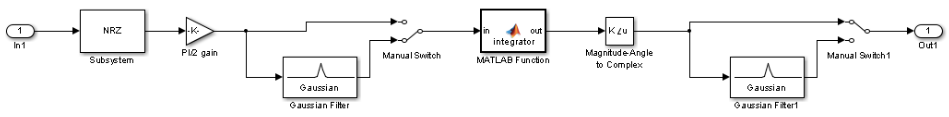


Figure 54: MSK and GMSK in the simulator

With the manual switches, we can switch the implementation between MSK, and two implementation forms of GMSK (more information below).

The used code:

```
% GMSK
%
    simulation='SIM_MOD_GMSK';


    T_Gauss_BTproduct=0.5;
    T_Group_delay=2;


    %Note: In transmitter block you can switch on/off Gaussian filters
    %Note2: To run BER measure with MSK the error rate calculation has
to
    %be set manually
%}
```

PSD:

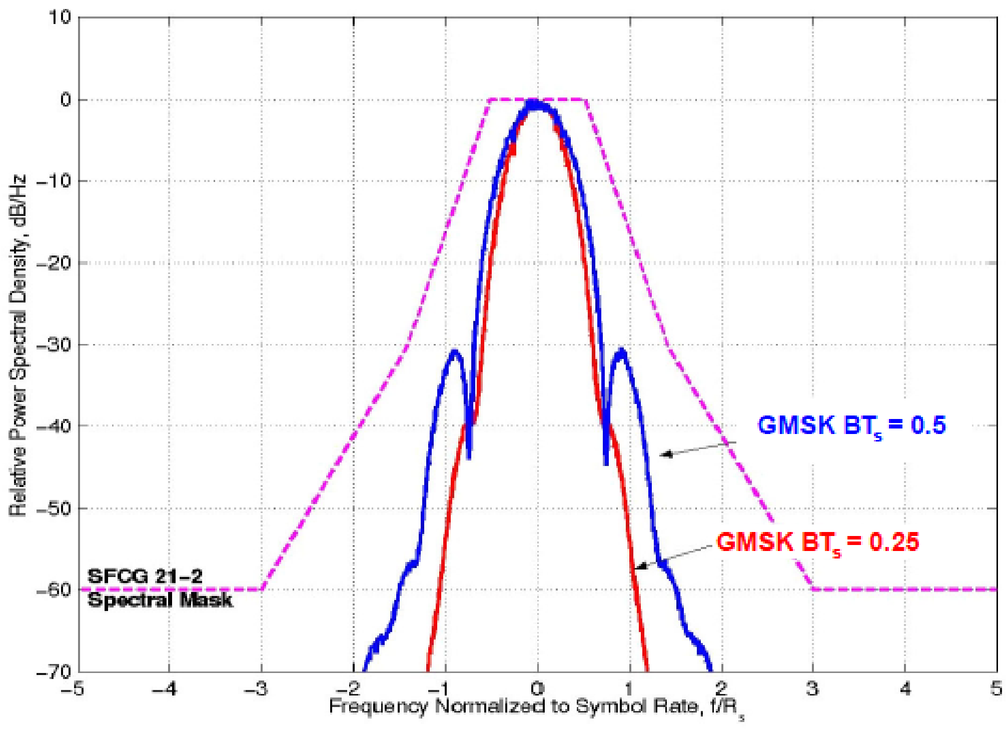


Figure 55: PSD of a GMSK signal from CCSDS standard:

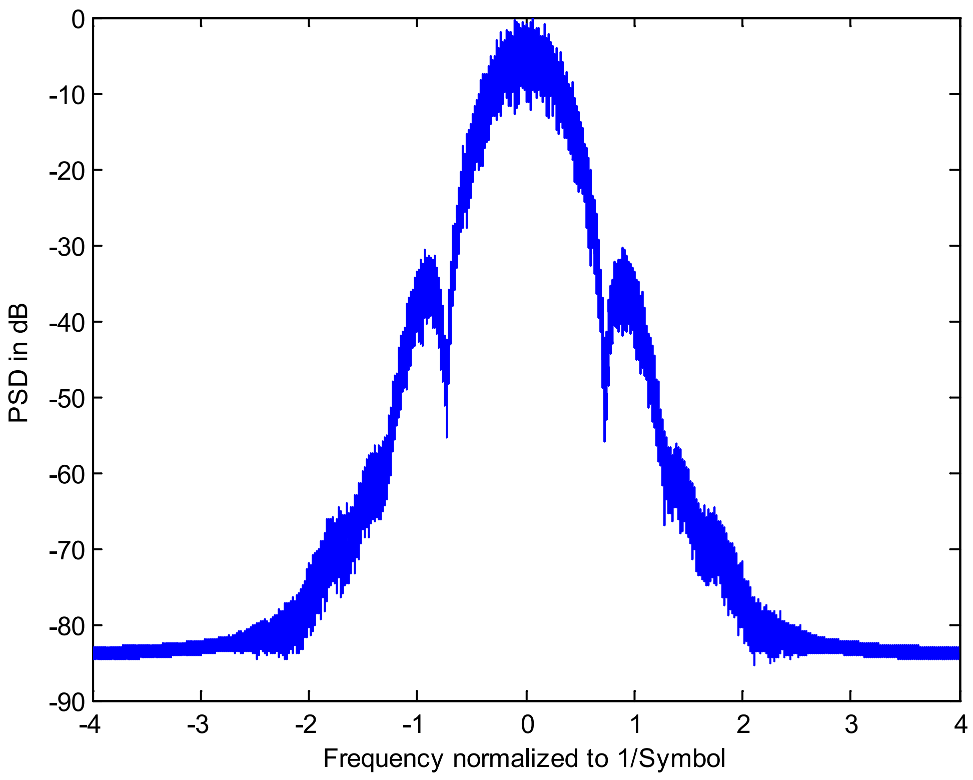


Figure 56: PSD of a GMSK signal from simulation (BT=0.5)

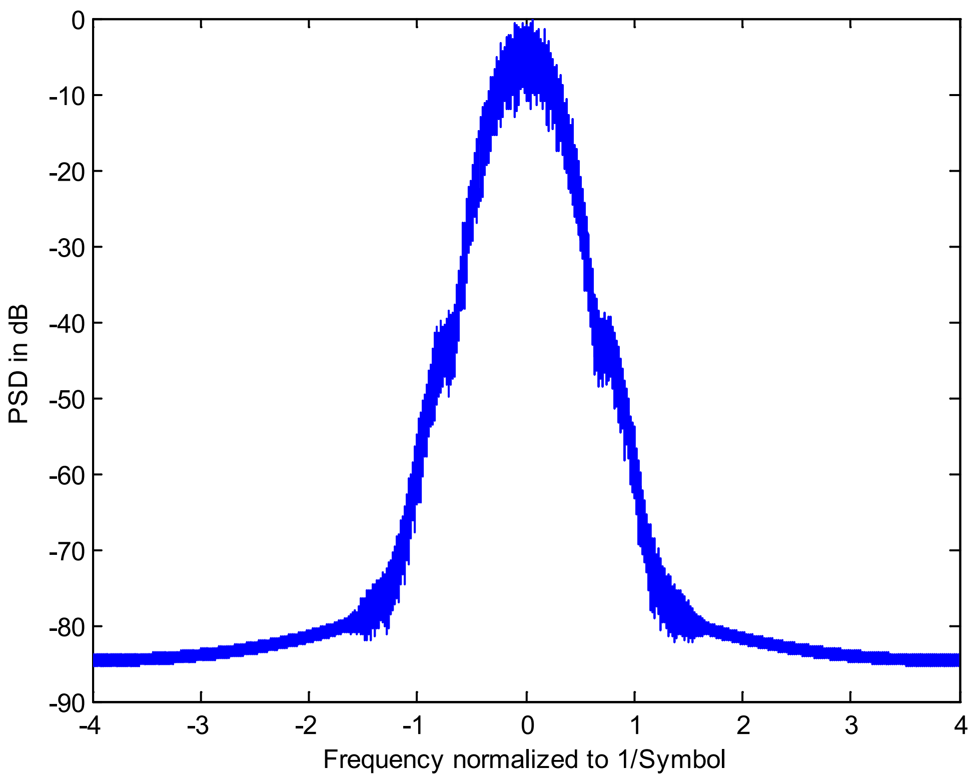


Figure 57: PSD of a GMSK signal from simulation (BT=0.25)

Scattering plots

Both plots where made with BT=0.25 Gaussian filter. In the first plot the filter is after the phase modulator, and in the second plot the filter is for the phase modulator.

Plots in Figure 57 are from simulator. In Figure 58 the same plots can be found from CCSDS standard.

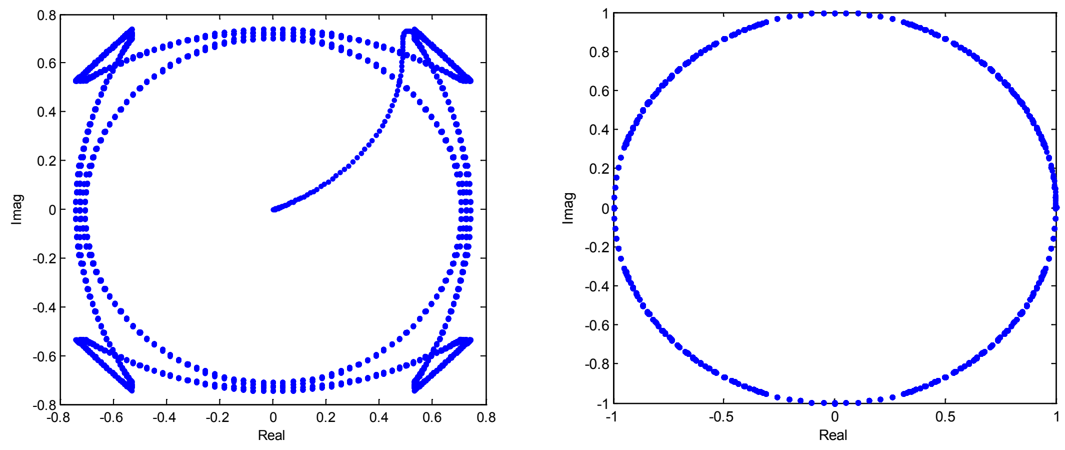


Figure 58: Scattering plots from simulator

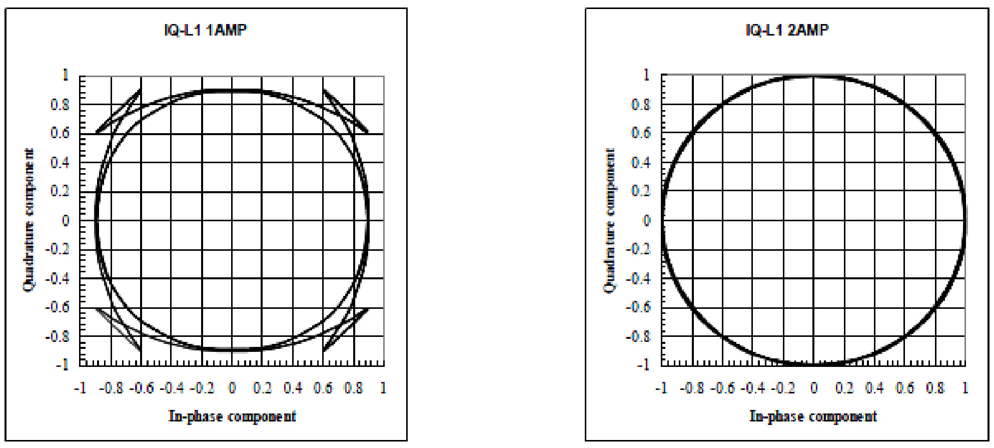


Figure 59: Scattering plots from CCSDS standard

5. Synchronization

5.1 Introduction

Synchronization is one of the most the most important tasks in modern communication. The main idea is to estimate the channel parameters from the received signal, and correct the signal with it. There is in general a lower bound for this estimation (Cramer-Rao lower bound CRLB), However, many algorithms which can reach it, are in general in practice not applicable most times, because of the computational complexity, or simply not exist. Because since many solutions exist for this problem, the development of a good receiver is a highly complex task.

There are two main categories for synchronization algorithms:

- NDA - Non Data Aided - This kind of algorithms does not require the data for correction
- DA – Data Aided – the knowledge of data is required for synchronization.

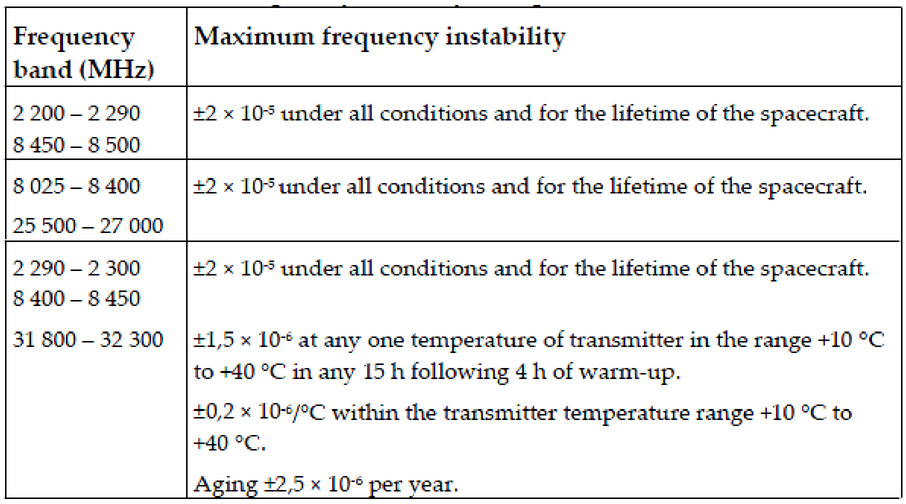
The purpose of this implementation is to show a possible solution of synchronization for modulations used in CCSDS standard.

5.2 Channel model parameters for space usage

The first task is to investigate the parameters for channel model and the limits where the algorithms have to work.

Frequency error

First the a priori known frequency error caused by Doppler effect has to be removed. It can be calculated by use of the known position and speed of the satellite. After this correction remains only the frequency instability caused by spacecraft transmitters. There is a maximum defined for allowed frequency instability in CCSDS standard (CCSDS 413.0-G-2):



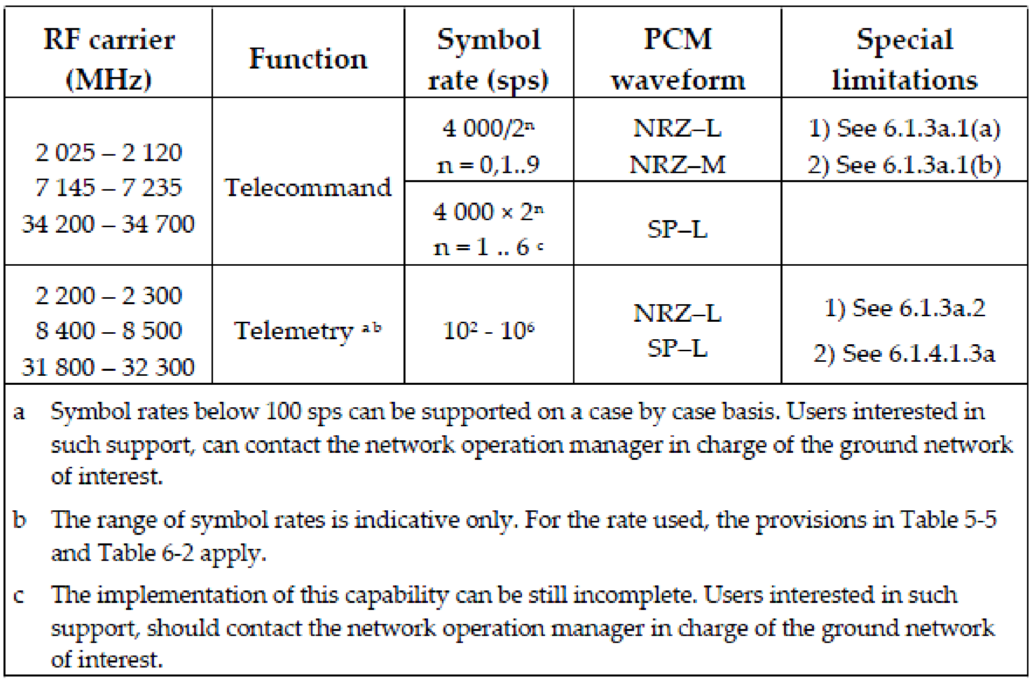
Frequency band (MHz)	Maximum frequency instability
2 200 – 2 290 8 450 – 8 500	±2 × 10^{-5} under all conditions and for the lifetime of the spacecraft.
8 025 – 8 400 25 500 – 27 000	±2 × 10^{-5} under all conditions and for the lifetime of the spacecraft.
2 290 – 2 300 8 400 – 8 450	±2 × 10^{-5} under all conditions and for the lifetime of the spacecraft.
31 800 – 32 300	±1,5 × 10^{-6} at any one temperature of transmitter in the range +10 °C to +40 °C in any 15 h following 4 h of warm-up. ±0,2 × 10^{-6}/°C within the transmitter temperature range +10 °C to +40 °C. Aging ±2,5 × 10^{-6} per year.

In addition for Category B missions the short term frequency stability shall be such that the resulting phase error when tracking the carrier with a second-order PLL with loop bandwidth $2B_L$ as specified for the mission, does not exceed 10 degrees peak in high signal to-noise conditions and in non-coherent mode.

NOTE 1 The "short term frequency stability" includes phase noise contribution and any "instantaneous" phase or frequency variations ("discontinuities") due to technological aspects and related to oscillator implementation.

NOTE 2 Depending on the link budget and on the ground station 2BL value, this requirement may have strong impact on the onboard subsystem architecture and the selection of the proper oscillator technology.

This model works in baseband, so the frequency error has to be taken into consideration in phase change per symbol period. The available symbol rates in CCSDS standard are:



RF carrier (MHz)	Function	Symbol rate (sps)	PCM waveform	Special limitations
2 025 – 2 120 7 145 – 7 235 34 200 – 34 700	Telecommand	4 000/2^n n = 0,1..9	NRZ–L NRZ–M	1) See 6.1.3a.1(a) 2) See 6.1.3a.1(b)
		4 000 × 2^n n = 1 .. 6 [c]	SP–L	
2 200 – 2 300 8 400 – 8 500 31 800 – 32 300	Telemetry [a b]	10^2 - 10^6	NRZ–L SP–L	1) See 6.1.3a.2 2) See 6.1.4.1.3a

a Symbol rates below 100 sps can be supported on a case by case basis. Users interested in such support, can contact the network operation manager in charge of the ground network of interest.

b The range of symbol rates is indicative only. For the rate used, the provisions in Table 5-5 and Table 6-2 apply.

c The implementation of this capability can be still incomplete. Users interested in such support, should contact the network operation manager in charge of the ground network of interest.

The worst case is the usage of a 34500 MHz carrier with $2*10^{-5}$ and 100 samples per second. In this case the frequency error is $34500*10^{6}*2*10^{-5}$=0.69 MHz, which means 6900 rotations of the vector per symbol period, which is more than 2 π of the signal. This means the possible frequency error can be overall between ±π radian per symbol. Normally, the used upsample factor is between 2 and 8, which much less than 6900. This means that this simulator can use ±π frequency error per sample too, which makes the simulation easier.

Phase noise

Limits from ECSS standard:

a. The phase noise of the unmodulated carrier shall be measured in noncoherent mode.

b. Phase noise of the unmodulated carrier, integrated between 100 Hz and 1 MHz shall be less than 2□ r.m.s. at UHF for the 2 200 MHz - 2 300 MHz band.

c. Phase noise of the unmodulated carrier, integrated between 100 Hz and 1 MHz shall be less than 6□ r.m.s. at SHF for the 8 025 MHz - 8 500 MHz band.

d. Phase noise of the unmodulated carrier, integrated between 100 Hz and 1 MHz shall be less than 10□ r.m.s. at EHF for the 25,5 GHz - 27,0 GHz and 31,8 GHz - 32,3 GHz. bands.

e. If specifications 6.2.6b, 6.2.6c and 6.2.6d are not met over the indicated integration ranges, integration shall be carried out taking into account the mission selected loop bandwidths and symbol rates.

This model use phase noise in dBc/Hz, which makes possible to model the phase noise in a more complex way. The implementation requires a worst case phase noise which can be calculated from the given limits in ECSS standard, this means -31 dBc/Hz in this case.

SNR

The enabled coding methods in CCSDS standard has been investigated in chapter 3. It can be seen, that less than 1 dB SNR the number of errors are too high, it does not make sense to use that low SNR for communication with these coding techniques. This means that there is no need for synchronization below this limit.

In the simulation model, SNR is given in energy/symbol, instead of energy/bit. QPSK signals have the same performance as BPSK, if they are computed in terms of Eb/No, but because one symbol contains 2 bits, QPSK needs twice more energy/symbol than BPSK for the same BER performance. Since in the simulation the energy/ symbol is normalized to 1, the limit for QPSK/OQPSK is about 4 dB SNR. GMSK has about the same BER performance as BPSK, so 1 dB is the chosen limit in this case for synchronization.

5.3 Possible transmitter structures

The second task is to implement some frequently used transmitter and receiver models without parameter estimation and correction. Although there are many possibilities, the CCSDS standard (CCSDS 413.0-G-2) has some recommendation to the used structures.

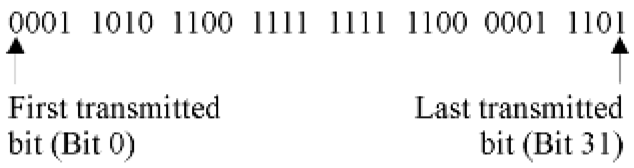
CCSDS recommendations

MODULATION METHODS FOR SRS, CATEGORY A
Because of the wide range of applications, ranging from the low-Earth orbiters to the science spacecraft at the edge of the Category A region (2×10^6 km), a number of different modulation schemes were retained in recommendation 401(2.4.17A) B-1:
– GMSK2 (BT_s=0.25) with precoding;
– Filtered OQPSK2 with various options:
• Baseband SRRC2, α=0.5;
• Baseband Butterworth 6 poles, BT_s=0.5;
• Other types of bandpass filters provided that the equivalent baseband BT_s is not greater than 0.5 and they ensure compliance with SFCG Recommendation 21-2R2 (or latest version) and interoperability with cross-supporting networks.
MODULATION METHODS FOR SRS, CATEGORY B
For SRS Category B missions, only one modulation was retained in recommendation 401(2.4.17B) B-1 and recommendation 401(2.4.20B) B-1:
– GMSK2 (BT_s=0.5) with precoding.
MODULATION METHODS FOR EESS AT 8 GHZ
Recommendation 401(2.4.18) B-1 contains three modulation options recommended for EESS missions:
– 4D 8PSK TCM;2
– GMSK2 (BT_s=0.25) with precoding;
– Filtered OQPSK2 with various options:
• Baseband SRRC2, α=0.5;
• Baseband Butterworth 6 poles, BT_s=0.5;
• Other types of bandpass filters provided that the equivalent baseband BT_s is not greater than 0.5 and they ensure compliance with SFCG Recommendation 21-2R2 (or latest version) and interoperability with cross-supporting networks.

For NDA based synchronization algorithms there is no impact of the coding to the estimation. To reduce computational complexity in the simulator, there is no need to use them. In DD algorithms there are two methods to use the information. First is to use the decoded signal as a reference assumed that there is no difference between the transmitted and the corrected bit pattern. The second idea is to use the failure detection mechanism of the coding. This means to try out some correction, and choose that where the number of the failures were minimal. This is the so called maximum likelihood principle.

Frame Synchronization

CCSDS standard use attached sync marker (ASM) for frame synchronization. The bit pattern is attached for the modulation/convolutional block. The format of the bit pattern is as follows:



The chosen transmitter structures and the chosen receiver structure without correction

This implementation contains the main models for the CCSDS recommendation by taking into consideration other possible structures too (for example BPSK).

<u>OQPSK</u>

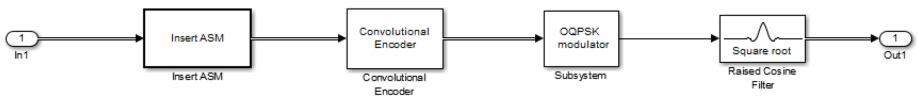


Figure 60: QPSK transmitter

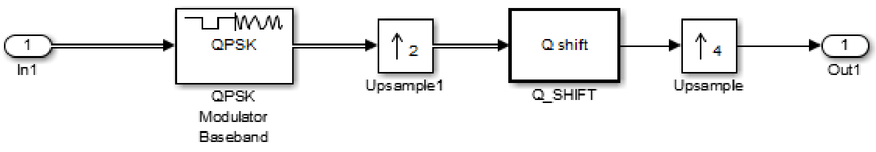


Figure 61: OQPSK modulator subsystem

The basic structure of the receiver of OQPSK is based on QPSK receiver. This is simply a downsampling, and demaping of the received signal. The only change is that, before downsampling, the signal needs to be shifted back.

8PSK

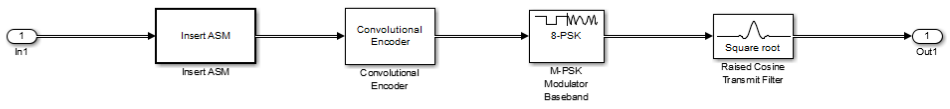


Figure 62: 8PSK based transmitter

Since the 4D-8PSK-TCM is a coded version of 8PSK, the synchronization of 4D-8PSK-TCM is the same task as the synchronization of 8PSK.

GMSK

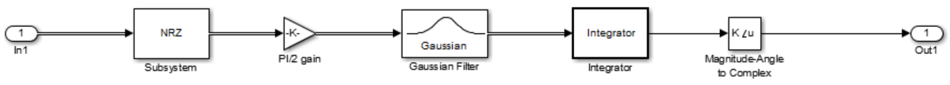


Figure 63: GMSK based transmitter

There are two main GMSK receiver structures:

- Coherent receiver : The receiver has complete knowledge of the carrier phase
- Noncoherent receiver : No knowledge of carrier phase needed

Coherent receiver: The standard high quality GMSK receivers are based on coherent structure, and Laurent decomposition of the GMSK signal. There is a study ordered by ESA (R. Abello, N. James, R. Madde, 2007), which shows a possible structure of this receiver optimized for requirements of space communication

BER performance of the cited coherent receiver can be seen in Figure 64.

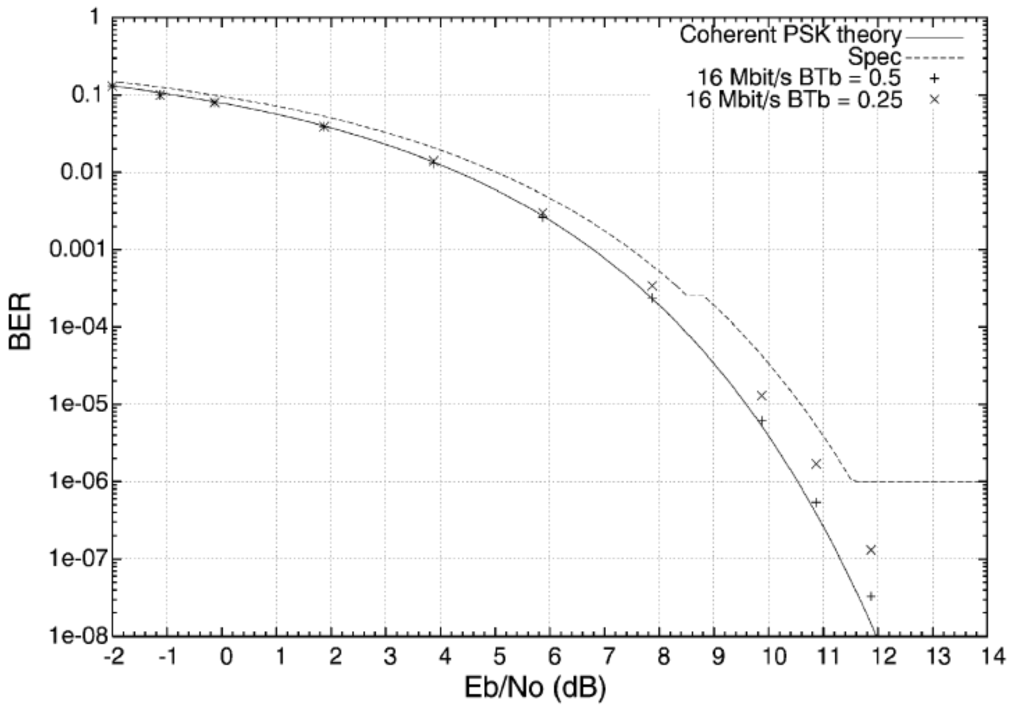
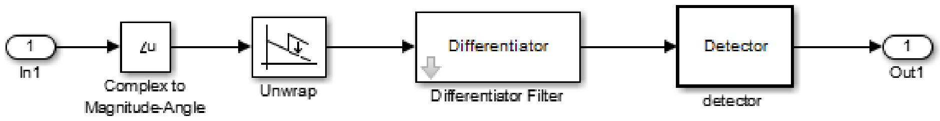


Figure 64: BER performance of the cited coherent receiver

Noncoherent receiver: The main drawback of the coherent receiver is the needed knowledge of carrier phase, and the lack of opportunity for parallel organization, because of the feedback structure. In this implementation a noncoherent receiver structure is used with maximum likelihood detector.



The main idea behind the detector is to reproduce all possible GMSK signal patterns and subtract them from the received signal. If there is a pass, then just the noise remains, which has less energy than the noise plus some remaining signal. This is a maximum likelihood detector for GMSK signal (Figure 65 and 66).

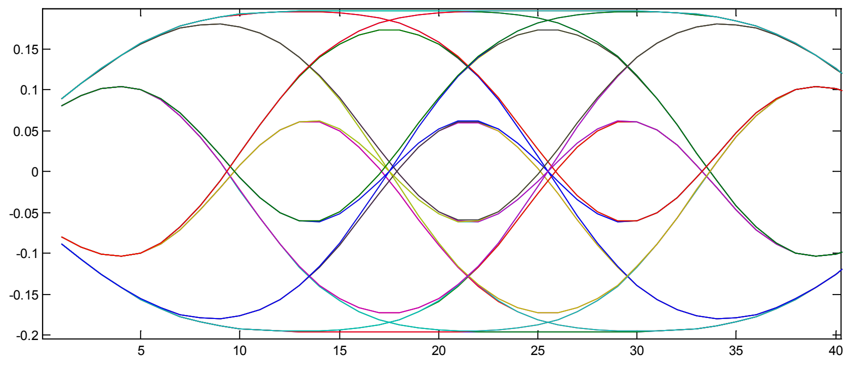


Figure 65: Possible GMSK signal patterns

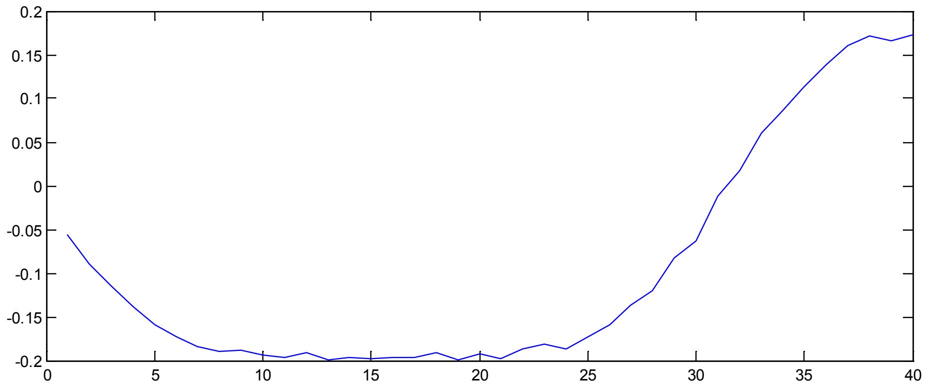


Figure 66: An example signal

A Matlab implementation of the receiver can be found in Appendix (Section 9.1). Because of the high number of bits, there are too many variations to compute. The number of variations has to be reduced, without losing too much performance. It can be seen that the nearby bits have more influence to a bit than other bits. (The bits cannot be analyzed alone because of ISI between bits, the influence of the bits to each other make a chain, which has to be taken into consideration in a maximum likelihood estimator). The main idea is to apply the algorithm just for several bits once, and calculate the most probably bit pattern. If it is made with a sliding window, there will be more estimation for each bit. From this estimations can be calculated an estimated value for the bit, in the simplest case with the calculation of mean. The quality of the receiver can be improved with a usage of a larger window, with investigate the possible shaping of result bits, or with the use the probabilities of each. The BER performance of the detector can be found in Figure 67 and 68 and the Matlab code in Appendix (Section 9.1).

BER performance:

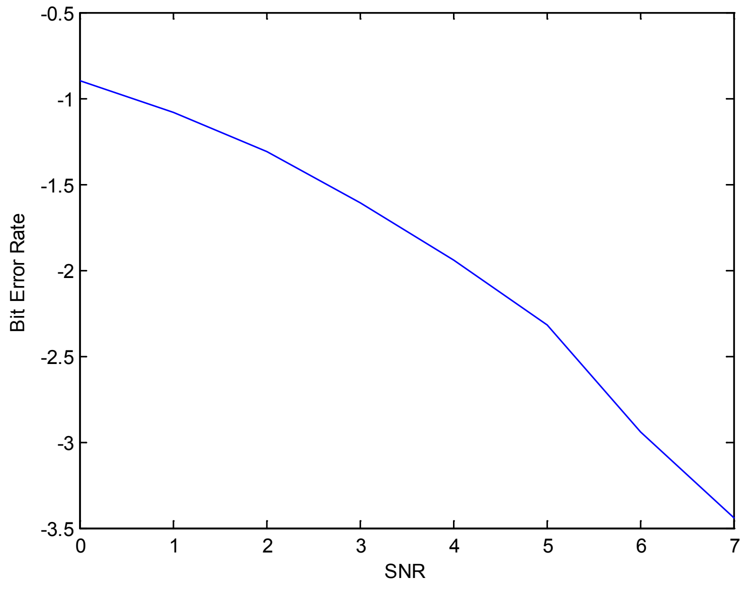


Figure 67: BER performance with a window length 5, without coding, BT= 0.5

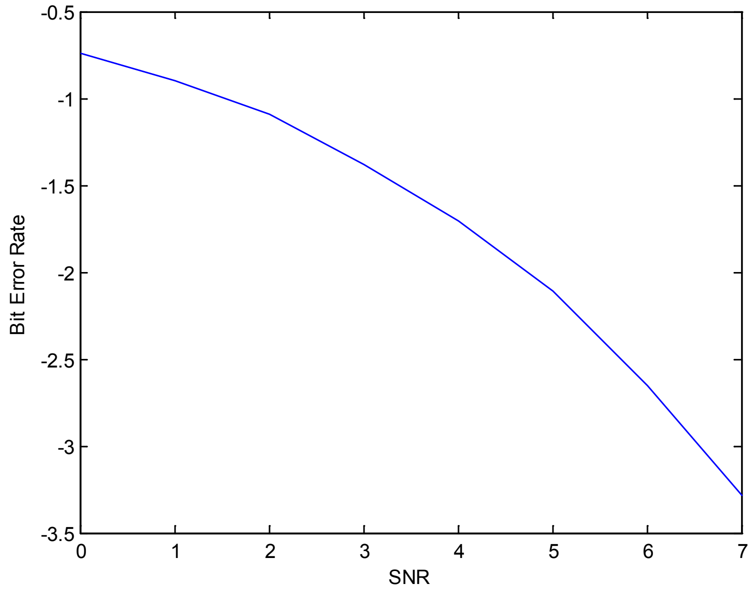


Figure 68: BER performance with window length 9, BT= 0.25

5.4 NDA synchronization algorithms for PSK based modulations

For non-data aided (NDA) synchronization, knowledge of the data is not needed (J. G. Proakis, 1995). Two different structures are used, feed-forward and feed-back algorithms. In general the advantages of feed-forward algorithms are that they are more organized in parallel, the performance of these algorithms is simpler to estimate and they have no stability problems. In contrast, feedback algorithms can follow small deviations of the parameters continuously.
The feedforward structure estimators usually estimate the parameters from a small numbers of symbols. Because these parameters are having just few variations in time, it is possible to make more observation in a large dataset, and then create average of these. With more observation the quality of the estimator can be highly improved. This structure can be very useful in this implementation because it is highly organized in parallel, and the parameters are having just a very small deviation in time, because of the high quality oscillators used in space communication.
The feedback algorithms are based on a loop-like structure. If the bandwidth of the loop filter is high, the synchronizers reaches the stable state rather quickly, but the fluctuations in the stable state are also high, which reduces the BER performance of the receiver.

5.4.1 Timing estimation (O&M, MaxE)

Basic idea used is that in principle RRC pulse shaped M-PSK systems have the maximal energy in the points, where they have to be sampled (M. Oerder, H. Meyr, 1988). The following algorithms are using this principle:

Maximal Energy: A simple solution for timing error estimation is the so called maximal energy method.
It simply makes sums of the signal energy in samples divided by upsample factor from each other. The estimated sampling point would be where this sum is the greatest.

$$\text{Timing\_estimation} = \text{place\_of\_max}(\sum_{q} bit\_pattern[n * upsample\_factor + q])$$

The used Matlab code can be found in Appendix (Section 9.2).

It has to be taken into consideration that the performance of this algorithm depends only on the length of the windows, it is calculated. Since Space communication use good quality oscillators, with only small drifts in time, the length for calculation can be large.

Oerder&Meyr: It is possible that the needed sampling point is between two samples. In this case the signal in the sampling point can be estimated with an interpolator. The estimation cannot be made by maximal energy method because it calculates the nearest sample to the sampling point. The usage of Oerder&Meyr algorithm is a possible solution for this problem.

$$X_L = \sum_{n=0}^{\substack{Constant* \\ Upsample\ factor}} |x[n]|^2 * e^{-\frac{i2\pi n}{upsample\_factor}}$$

$$sampling\ delay = \frac{Upsample\ factor}{2\pi} * argument(X_L)$$

The used Matlab code can be found in Appendix (Section 9.3).

If the upsample rate is much higher than the symbol rate, the maximal energy method has an acceptable performance with reduced computational cost.

Timing error with OQPSK: Offset QPSK needs a special implementation of timing error correction. Since there is no phase estimation available for the signal in this point, the time offset caused by the modulation cannot be undone. Since the signal has about the same energy in all phase, neither O&M nor Maximal Energy method can be used without modification.

To get a usable solution, the structure of the received signal has to be investigated. Basic idea is to search for the maximal energy sample separately for I and Q channel, and then summarize this information to estimate timing error. In an ideal case with no phase error, it is obvious that for example a maximal energy method used just for I channel does the estimation. If it is equipped to the Q channel, the estimation will include the half symbol delay caused by offset, which has to be removed. Since in this state there is no information about the phase error, the signal has a random rotation, which has to be taken into consideration. First there is no information if I and Q are swapped or not. For example, if there is 90° phase error, the I and Q are swapped.

In other phase errors there is a crosstalk between the two channels, coming from trigonometry. It causes problems if the phase error is about 45°, because the energy differences in I and Q channel vanish. To avoid false estimation, the idea is to calculate the method with 45° phase shift, and choose the biggest value from.

The used method is to calculate the energy separated for I and Q channel like in maximal Energy method, then summarize them with half symbol time spaces. Then it is possible to downsample the signal with upsample factor divided by 2. After that 2 samples per symbol remains before phase estimation.

After phase correction between the remaining two samples, the right one can be chosen with maximal energy method after correction.

The used Matlab code can be found in Appendix (Section 9.4, 9.5).

5.4.2 Phase error estimation (V&V)

The phase error of an M-PSK based system is usually corrected after downsampling. For phase estimation, this implementation use the Viterbi and Viterbi (V&V) algorithm. The basic idea behind V&V is the following:

M-PSK phase has this structure at the receiver: Phase= Phase_error+n*2π/M. If it is multiplied by M then: Phase=M*Phase_error+n*2π since the phase is repeated in every 2 π the result is equal to M*Phase_error, which has to be divided by M to get the phase error. The maximum estimation range of this method is 2π/M.

Viterbi and Viterbi algorithm

V&V algorithms (A. J. Viterbi, A. M. Viterbi, 1983) use the same idea, with a possibility to use the real part of the signal with a chosen nonlinearity:

$\theta_{estimation} = \frac{1}{M} * argument(\sum_{k=0}^{L-1} nonlinear(P[k]) * e^{iM\varphi[k]})$; x[k]= $P[k] * e^{iM\varphi[k]}$

In this implementation the nonlinear function P[k] is reduced to unity.
V&V can be used in feed-forward form, but in this implementation it is used in feedback form. In feedback form V&V can repair some frequency error and other deviations in signal too.
The performance of the V&V algorithm depends on the number of estimation averaged, in feedforward case, and the bandwidth of the feedback filter in feedback case.
The used Matlab code can be found in Appendix (Section 9.6).
Note: in feedback form if the filter bandwidth is limited, there is a need of a large dataset to reach stable phase estimation. Until this point is reached the BER performance of the receiver would be less than optimal, because of not accurate correction of the channel. To avoid this in a packet based system like this, a time reverse of the packet can be used to estimate the phase of the first symbol, and after it the signal can be tracked with a normal V&V feedback structure. (Figure 69)

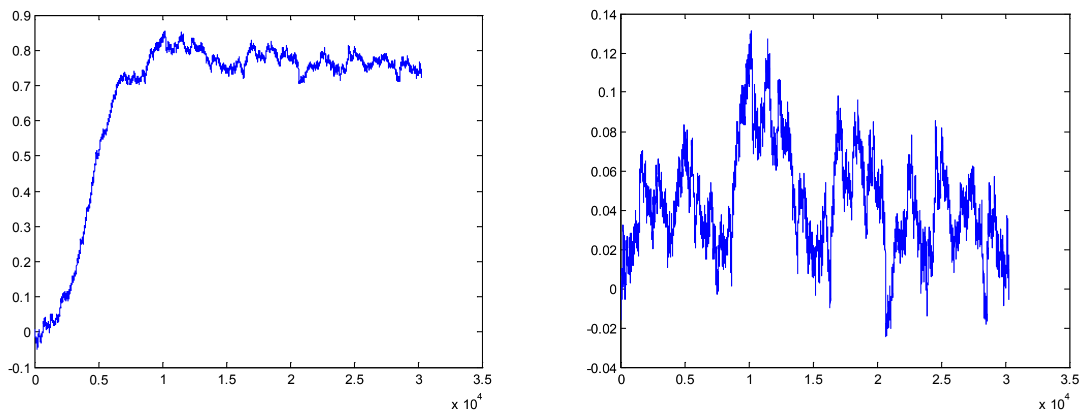


Figure 69: Time reverse V&V feedback example

5.4.3 Phase estimation in OQPSK case

As it can be seen the phase of OQPSK is the same as a QPSK signal with twice the symbol rate (Figure 70). It follows that V&V can be used for estimation in OQPSK case too without any change.

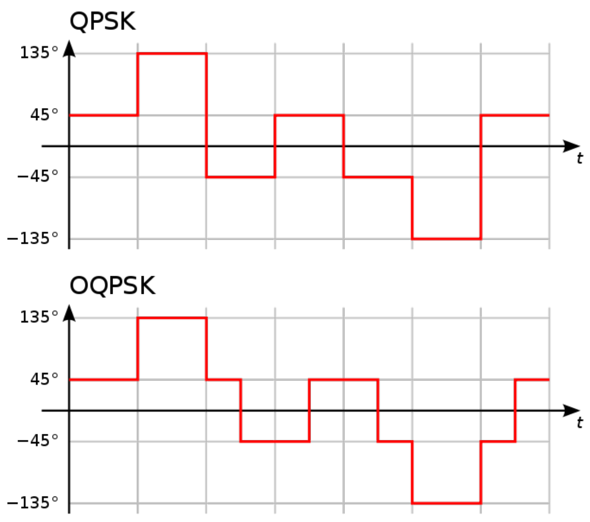


Figure 70: QPSK and OQPSK signal example (from: Wikipedia)

5.4.4 Frequency estimation (V&V)

In section 4.2 it has been investigated that there can be a huge frequency error, with small time variation. Because of these properties, the frequency error is in this implementation estimated and corrected by DA algorithms. The possibly remaining slow drifts can be corrected by V&V feedback structure.

5.5 NDA synchronization algorithms for GMSK

GMSK is a continuous phase modulation. Because of the usage of the derivation in GMSK demodulator, phase error has no, and frequency error has only a small influence on the receiver before the detector. From the detector structure comes that the constant deviation from frequency error has no influence too. That is why only timing error has to be taken into consideration.

Timing Error correction

Two possible methods are investigated for this problem. A DA and an NDA solution. More information about DA solution can be found in Chapter 4.6.

The main idea for a NDA solution comes from the observation, if a matched filter is used in the receiver structure; the filtered signal has usually peaks at sampling points (Figure 71).

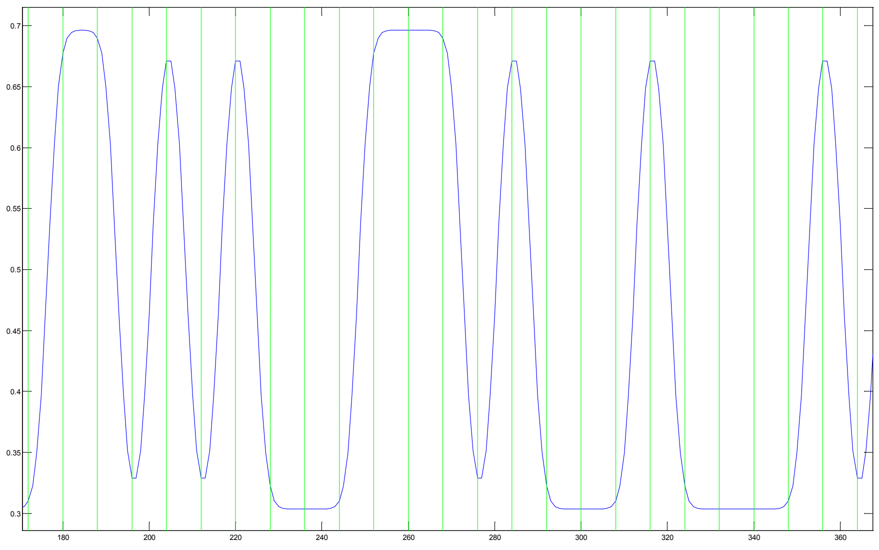


Figure 71: Filtered GMSK signal

Because GMSK have intersymbol interference, this peak is not always at the sampling point. This means that the performance of the estimator depends on the chosen BT. factor. In general with a huge number of symbols investigated, this method can be effectively used.

With this principle the estimation can be done for example with maximal energy method used in M-PSK case (Figure 72).

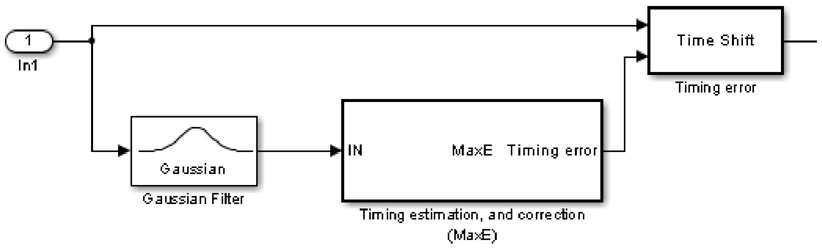


Figure 72: GMSK timing estimator in simulator

To demodulate the signal there is no need of the matched filter. That is why a separated time shift block is used.

5.6 DA synchronization – feedback structures

In CCSDS standard all possible transmitter structure includes an attached sync marker (ASM). This sync marker can be used for synchronization in several ways. This implementation consist of a solution for ASM based frame synchronization, phase estimation for M-PSK systems, a powerful frequency estimator, and a possible solution for low SNR timing estimation.

5.6.1 ASM based frame synchronization

Frame synchronization is needed in all possible communication solution in CCSDS standard. The beginning of each frame is marked with the ASM bit pattern. Frame synchronization:

1. Search ASM in the received signal.
2. Cut the frame from the signal.

The search of the ASM is made in this implementation by finding peaks in cross correlation of the received bit pattern and the known ASM. Since the frame size is known it is easy to cut it from the signal. The m file used for simulation can be found in the Appendix (Section 9.7).

5.6.2 Phase estimation in *M-PSK systems*

As it has been explained that V&V algorithms in an M-PSK system have an estimation range of 2π/M, since without any knowledge of the transmitted signal there is no possible solution to make a choice between M possible received signals. This implementation tries to decode all possible solutions, until ASM can be found in one signal (Figure 73 and 74).

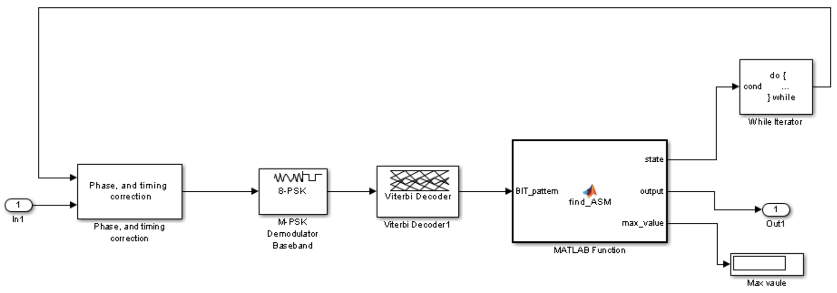


Figure 73: 8PSK decoder structure

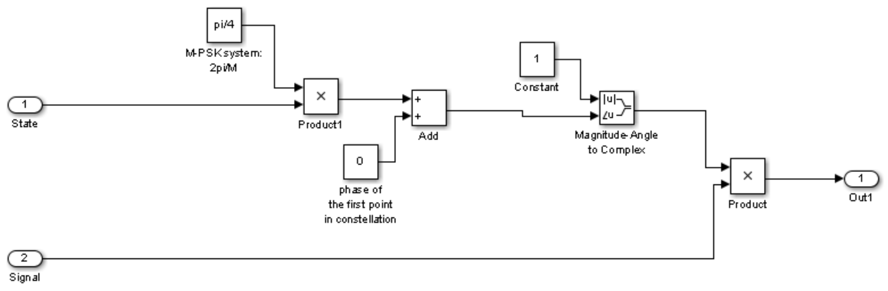


Figure 74: Phase correction block

It has to be taken into consideration that, if the sync marker is not coded, there are simpler solutions to find the phase of the signal.

5.6.3 Frequency estimation

It has been shown too in Section 4.2 that the frequency error can be large. The frequencies of the onboard oscillators of satellites have a small deviation of time. From these properties comes that a large sequence of the signal can be used, and the result is valid for a large set of signal. So the estimation can be calculated rarely from a chosen sequence of the signal.
To extend acquisition range there is a need of a decision aided estimator. As it has been demonstrated in Section 4.4 that the absolute maximum repairable frequency error of Viterbi and Viterbi in case of M-PSK system is 2π/M /symbol. In general a smaller bandwidth is recommended to reduce the effect caused by the noise.
The main idea is to try the receiver with all possible frequency correction, with a resolution specified by the feedback filter. If there is a match in received signal, then the remaining frequency error is smaller than the bandwidth of the V&V algorithm.
To find the match in the received signal only the ASM bit patterns are available, because there is not necessarily any information about the bit pattern of the frame. To get a good estimation it is recommended to use more packets to estimate. The chosen way to find the bit patterns is to calculate cross-correlation of the signal, and summarize the results with the known distance of markers (frame size+ASM size). If the result is greater or equal to ASM size minus ena-

bled failures multiplied with the number of frames, then there is a match, and the frequency error is found.
It makes sense to correct frequency error at least approximately before receiver filter is applied, to push the signal into the passband frequency of the filter. Implementation in simulator can be seen in Figure 75.

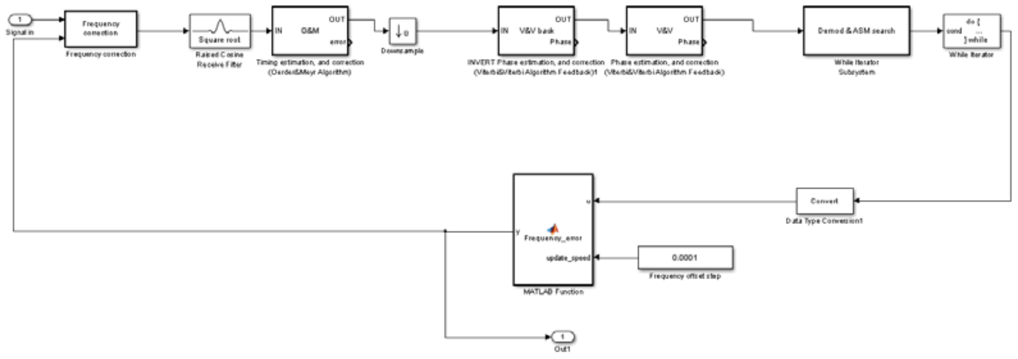


Figure 75: 8PSK receiver

The implementation of the ASM search can be found in Appendix (Section 9.8). For a real process, it is recommended to use successive approximation with different bandwidth estimators. It is more accurate with a reasonable speed improvement. The implementation can be found in Sim_freq_estimation.slx Simulink model file.

5.6.4 Timing estimator

In the case of presence of CRC coding it is possible to investigate the number of errors. This knowledge can be used to recover timing error in M-PSK, and GMSK case. The simplest method is to calculate all variations, and choose this one where the numbers of bit errors are the least. This implementation can work in all cases, when the signal can be decoded.

5.6.5 Timing estimator in GMSK

In GMSK systems there is a possibility to recover timing error by use of the maximum likelihood principle too. The main idea is to calculate the implemented estimator for several bits, with all possible bits, and all possible timing errors, and choose the best variation.

5.7 Basic receiver structures and demodulation examples

In this section there will be some example, and BER measurements of some full receiver structures. Because of the multi dimension structure of the task there is no chance to simulate all possibilities. The simulations where made with random phase, and timing error, with frequency error considered to be minimized by the frequency error estimator.
Since ASM patterns are not coded with Reed-Solomon or turbo codes, they have no impact to synchronization performance. There is no need to simulate them here.

5.7.1 OQPSK

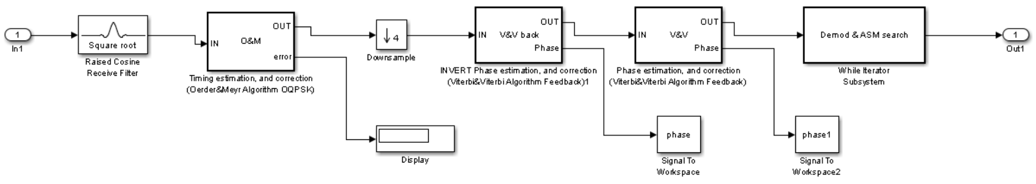

1
In1
Square root
Raised Cosine Receive Filter
IN
O&M
OUT
error
Timing estimation, and correction (Oerder&Meyr Algorithm OQPSK)
↓4
Downsample
Display
IN
V&V back
OUT
Phase
INVERT Phase estimation, and correction (Viterbi&Viterbi Algorithm Feedback)1
phase
Signal To Workspace
IN
V&V
OUT
Phase
Phase estimation, and correction (Viterbi&Viterbi Algorithm Feedback)
phase1
Signal To Workspace2
Demod & ASM search
While Iterator Subsystem
1
Out1

Figure 76: OQPSK receiver

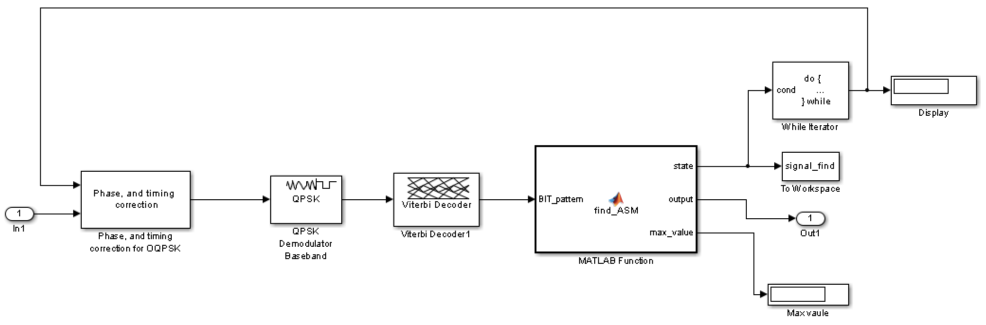


Figure 77: Demod&ASM subsystem search

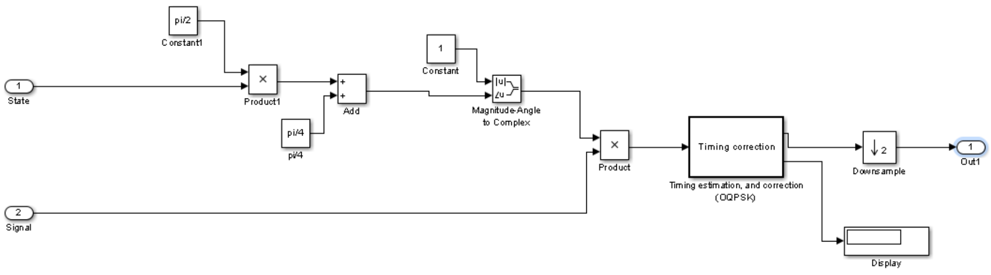


Figure 78: Phase and timing correction

BER performance:

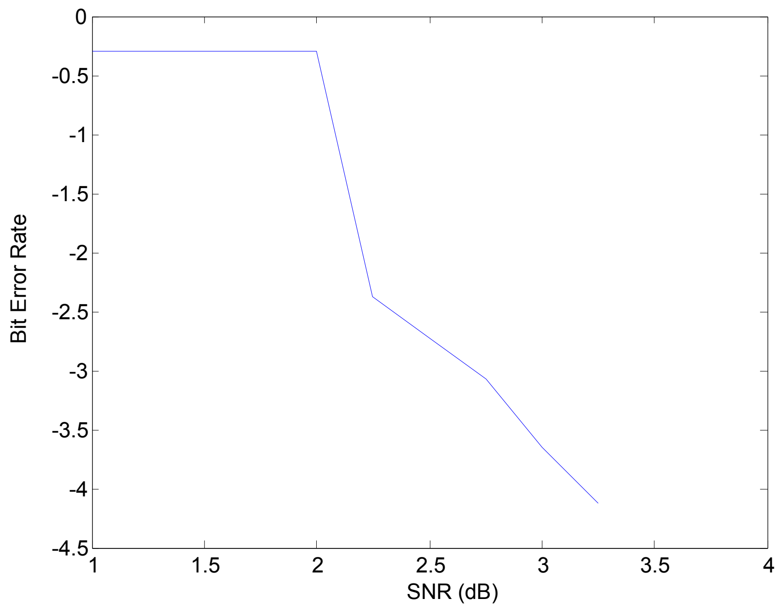


Figure 79: BER performance of OQPSK receiver

5.7.2 8-PSK

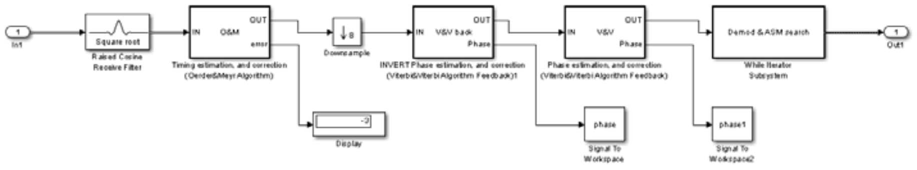

1
In1
Square root
Raised Cosine Receive Filter
IN
O&M
OUT
error
Timing estimation, and correction (Oerder&Meyr Algorithm)
↓8
Downsample
-3
Display
IN
V&V back
OUT
Phase
INVERT Phase estimation, and correction (Viterbi&Viterbi Algorithm Feedback)1
IN
V&V
OUT
Phase
Phase estimation, and correction (Viterbi&Viterbi Algorithm Feedback)
phase
Signal To Workspace
phase1
Signal To Workspace2
Demod & ASM search
While Iterator Subsystem
1
Out1

Figure 80: 8PSK receiver structure

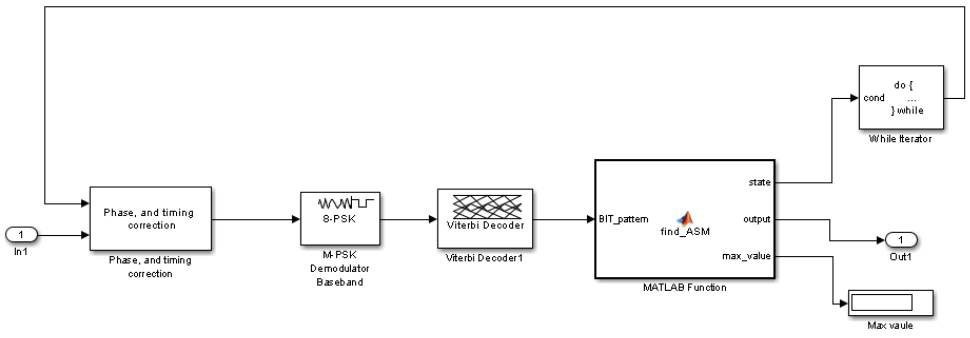


Figure 81: Demod&ASM subsystem search

BER performance:

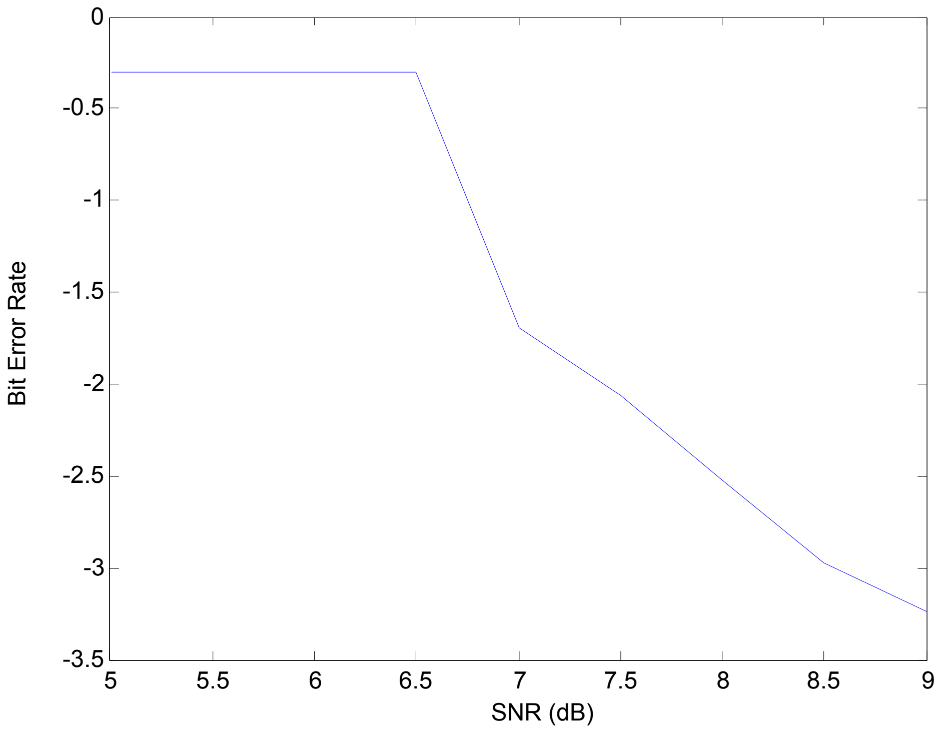


Figure 82: BER performance of 8PSK receiver

5.7.3 GMSK

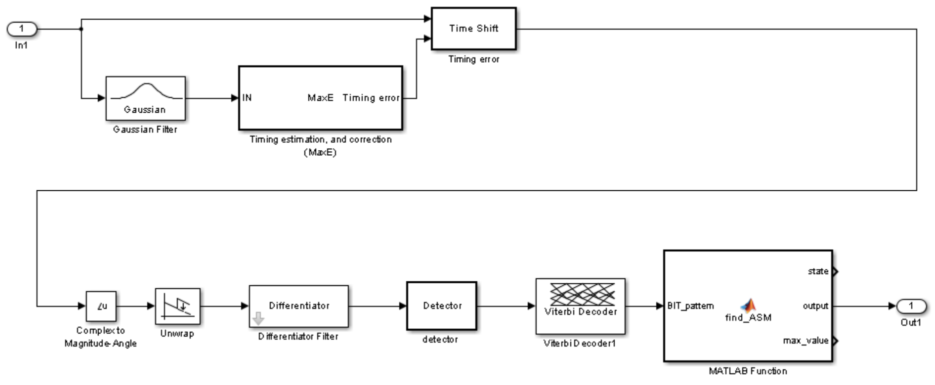


Figure 83: GMSK receiver structure

BER performance:

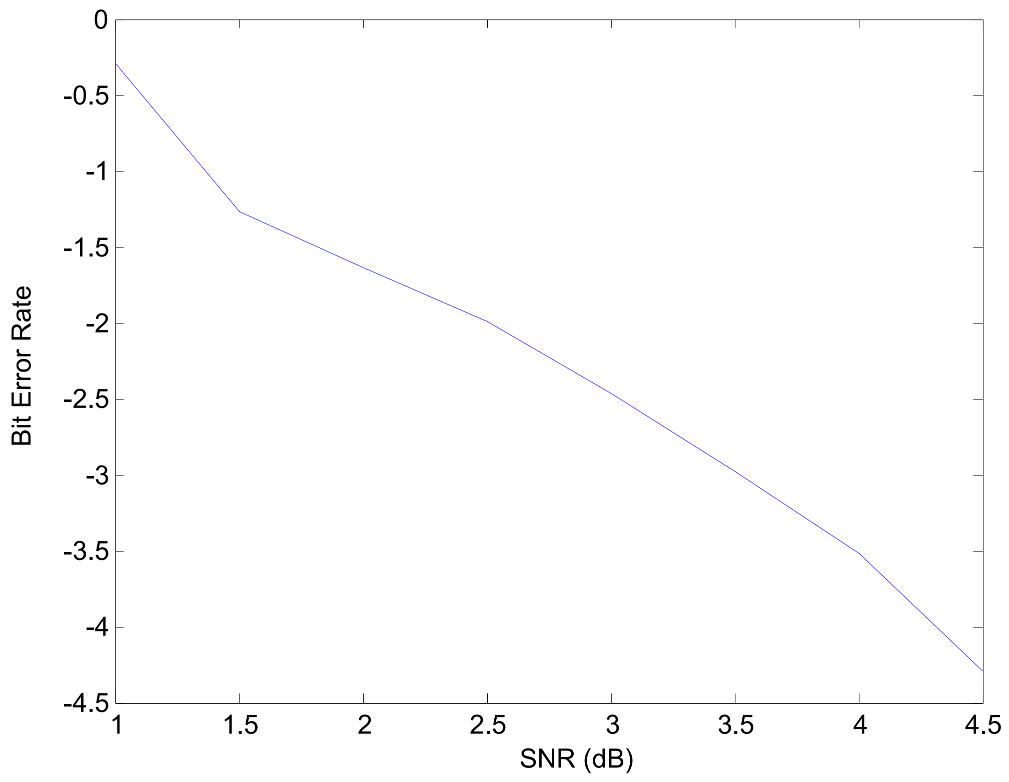


Figure 84: BER performance of GMSK receiver

6. Parallel Organization

6.1 Introduction

The main goal of this chapter is to give an estimation of the needed computation speed of the modem in floating point operations per second (FLOPS), and investigate the possible architectures to implement the modem, with the usage of the latest available hardware this time (CPU, GPGPU).

The chosen maximum rate: In this implementation the chosen maximum rate is 40 Msymbol per second, with 8 as upsample factor. The maximum throughput of the system is 80 Mbit, or 40 MBit with ½ rate convolutional coding. The chosen quantizing is 16 bit.

6.2 Parallel organization of the used algorithms

An effective implementation of an algorithm is a highly complex task, which depends on the chosen architecture. The purpose of this part is to give an estimation of the needed computation speed to each algorithm. As none of the modern CPUs have enough power to calculate the whole demodulation in serial operation, the possible parallel organization has to be investigated too. The investigated algorithms are as follows:

Transmitter:

Convolutional encoder, M-PSK modulation, FIR filter

Receiver:

Doppler correction, Best sample method, Differentiation, GMSK demodulation, MPSK demodulation, Viterbi algorithm

Convolutional encoder:

Convolutional encoding use, time delays and logical operators. CCSDS standard uses 7 bit delay and 2*7 XOR logical operators.

Flop/bit	Flops
14	0.56 G

It is not easily organized in parallel. But hardware specific implementation can be much faster, and compared to other parts of the implementation the needed number of computations is negligible.

MPSK modulation:

M-PSK modulation is simply an association of bits.

MPSK demodulation:

In BPSK, QPSK, 8PSK systems it can be done with simple comparison. With each comparison 1 bit can be demodulated.

Flop/bit	Flops
1	80 M

It is easily organized in parallel; each symbol can be processed alone.

FIR Filter:

A FIR filter is described by: $y_k = \sum_{i=1}^{m} c_i u_{k-i}$,

For each sample m multiplications, and m additions are needed. As an example, a 40 tap filter requires 80 Flop/sample, because of IQ structure it has to be multiplied by 2. to get Flops

Flop/sample	Flops
160	51.2 G

It is highly organized in parallel since each sample can be calculated alone.

Cartesian to polar coordinates

Since the investigated synchronization algorithms need less computation in polar coordinates it worth’s to change coordinate system:

The magnitude of the signal can be calculated by Pythagorean Theorem.

$Magnitude = \sqrt[2]{I^2 + Q^2}$. It requires 4 flop/sample

Flop/sample	Flops
4	1.3 G

The phase calculation can be done by inverse tangent. The computation speed of inverse tangent is highly dependent on the used approximation method, since the best approximation method highly depends on the architecture, it cannot be investigated separately from implementation. More information about architectures can be found in the next 2 chapters.

Look Up Table (LUT): practically not usable with PC-s, because of the memory access time, and lack of computability with vector based calculation.

Cordic: Since Cordic needs small lookup tables, it has similar problems as normal LUT solution.

Power series: it approximates the function with a polynomial approach:

$$f(x) = \sum_{n=0}^{\infty} a_n (x-c)^n$$

The first approximation is calculated via a polynomial with 10 steps. The exponential function in the first approximation is calculated with multiplication. In AVX-512 instruction set there are several methods to increase the speed of exponential calculation. In GPU an efficient solution is available with atanf() function.

Flop/sample	Flops
74	23.7 G

It is highly organized in parallel since each sample can be calculated alone.

Doppler Correction

Frequency correction in polar coordinates:

$$\varphi_n = mod(Error_{\frac{dphase}{sample}} * n + \varphi_n, 2\pi)$$

Timing corrections- since this implementation does not use interpolation, timing error can be corrected approximately by removing samples, or inserting zeroes

Flop/sample	Flops
4	1.3 G

It is highly organized in parallel since each sample can be calculated alone.

Best Sample Method

In polar coordinates the calculation of best sample method is a simple square, and addition.

Flop/sample	Flops
2	0.7 G

It is highly organized in parallel since each sample can be calculated alone. The sum of the results can be calculated in groups to be able to parallel calculation.

Viterbi and Viterbi

Since a complex multiplication uses 6 Flop, it is more efficient to calculate the error in polar coordinate system.

1. Multiply phase with M, and calculate the modulus of the result with 2 π – 2 Flop/sample
2. Feedback filter -2 Flop/sample

Flop/sample	Flops
4	1.3G

It is not organized in parallel, because of the feedback structure.

Differentiation:

3 Flop/sample, subtraction, compare, addition if needed

Flop/sample	Flops
3	1G

It is highly organized in parallel each sample can be calculated alone**.**

GMSK Demodulation

The chosen demodulator scheme for GMSK use N as a parameter. With higher N the bit error rate increases. The needed computation is 2^N subtraction, and N addition per sample:

for example if N=10:

Flop/sample	Flops
2058	659G

It is highly organized in parallel since each sample can be calculated alone**.**

Viterbi decoder:

To build the trellis diagram each bit requires normally M^2 multiplication, and M comparison, where M is the number of possible states. In this case it is 2^7. Because of the special structure of rate 1/2 convolutional codes, arrive to a state is only possible in 2 different way.

The required computation: 128*2 addition, 128 compare

Flop/bit	Flops
384	123G

The idea for parallel organization, which can be followed in this project, is to split the stream into n parts (or windows). Each window can be processed in parallel by a thread running on a CPU or GPU. In order to perform the reception at each window the initial synchronization has to be performed independently and the output of each thread has to be merged seamlessly together(Figure 85). To enable initial synchronization at each window without losing data, the windows must be overlapped in time.

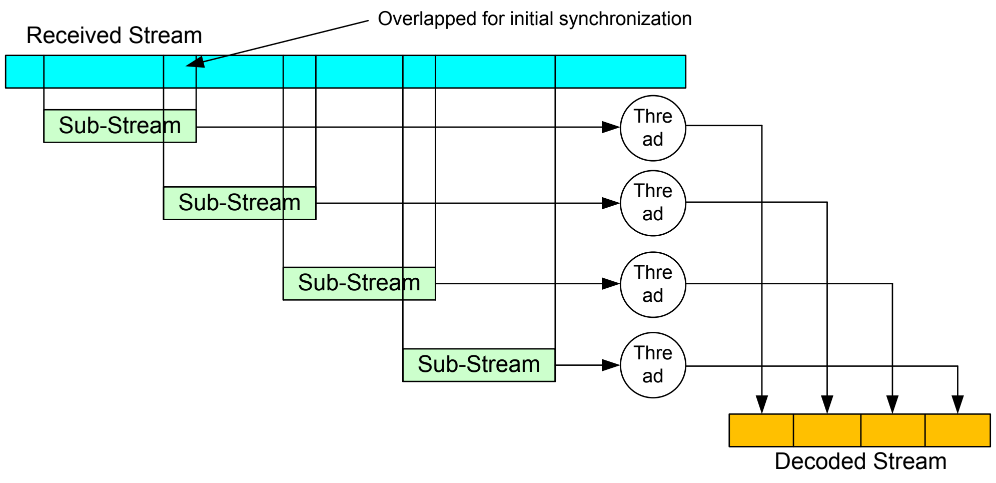


Figure 85: Parallel organization

6.3 CPU implementation

In this and in the next chapter, a 40 tap FIR filter is implemented in a modern CPU, and GP-GPU platform. The implementation shows an example of the main architectural aspects, and the connection between theoretical, and practical speed of the system.

The basic properties of a CPU:

- Usually high speed memory access due to multistage cache system, and efficient page management
- High serial computation capabilities

- Low number of threads (2-8 usually), costly switch between treads (state save needed)
- Capability to calculate with multiple data, or vectors in one time (MMX, SSE, AVX) (D. A. Patterson, J. L. Hennessey, 1998)

Assuming that we do not use an i7-4xxx Pentium, the AVX instruction set is the most performing one. The code using AVX intrinsics is shown below for a single sample. The filter coefficients c and the “zero”-vector are preloaded in AVX registers. The load instructions (_mm256_loadu_ps) are placed such that one cycle is between the load and the use of the loaded data in order to hide memory latency.

Please note that it is important to use the arch: AVX compiler option. Typically the C++ compiler use SSE for optimization as default and embed the AVX intrinsics into the SSE code, which results in expensive state switching of the CPU. The used algorithm:

```
        a0=_mm256_loadu_ps(&data[i]);              // load 8 unaligned
float
        sum=_mm256_mul_ps(a0, c0);
        a1=_mm256_loadu_ps(&data[i+8]);
        q=_mm256_mul_ps(a1, c1);
        a2=_mm256_loadu_ps(&data[i+16]);
        sum=_mm256_add_ps(sum, q);
        q=_mm256_mul_ps(a2, c2);
        a3=_mm256_loadu_ps(&data[i+24]);
        sum=_mm256_add_ps(sum, q);
        q=_mm256_mul_ps(a3, c3);
        a4=_mm256_loadu_ps(&data[i+32]);
        sum=_mm256_add_ps(sum, q);
        q=_mm256_mul_ps(a4, c4);
        sum=_mm256_add_ps(sum, q);
        sum=_mm256_hadd_ps(sum, zero);                  // horizontal
add pairwise
        sum=_mm256_hadd_ps(sum, zero);
        sum=_mm256_hadd_ps(sum, zero);
```

```
data[i]=*(float*)&sum;
```

Time measurement on an Intel i7-3820QM processor showed that 18 lines of code are executed exactly in 18 CPU cycles on a single core. This means that no performance impairments from memory latency and throughput. It could also be verified that, when using a single core, the CPU runs on its Turbo frequency of 3.7GHz (see Intel "Turbo Boost Technology" for more details).

In a second measurements, all 4 cores of the i7-3820QM processor where utilized with the filter calculation by means of 4 parallel threads. Again it was possible to verify practically the full use of the CPU cycles for filter calculation, this time on the normal clock speed of 2.7GHz. The measurements results are summarized in the table below.

Number of core in use on the CPU	Measured throughput in Msamp/s	Consumed CPU clock speed in GHz
1	203	3,65
2	361	3,25
3	475	2,85
4	649	2,6

The results have been also compared with an implementation using just for-loops and a standard optimizer. A throughput of 30Msamples/s has been achieved with a single core in use.

The results allow us to derive a performance formula for FIR filters on CPUs supporting AVX. Assuming that the filter order m is an integer multiple of 8, calculation in the AVX instruction set requires

- m/8 loads
- m/8 muls
- m/8-1 adds
- 3 horizontal adds (for adding the elements of the resulting vector)
- 1 store

resulting in **3m/8+3 cycles/sample** (In this case 18). Note that this is for a real sample. For example, a 40 tap filter in I and Q requires 2*18=36 CPU cycles per complex sample.

Comparison to theory

The maximum theoretically throughput of the system, based on the last chapter:

Flops	Msample/S
86.4 G	1080

Theoretical capability of the system can be found:

http://download.intel.com/support/processors/corei7/sb/core_i7-3800_m.pdf

Measured real throughput of the system: 649 Msample/sec

As it can be seen, about 60 percent of the theoretical maximum computation capabilities are achieved. It has to be taken into consideration, that theoretical computation speed is only reachable with special problems.

6.4 Cuda

The basic properties of a GP-GPU:

- GP-GPU has high computation capability, with extremely parallel organization, and some architectural restrictions
- There are two main implementation on market: OPENCL, and Cuda. Since today Cuda is more frequently used, and high optimized solutions are always hardware dependent, this trial uses Cuda.

In a Cuda (M. Garland, S. Le Grand, J. Nickolls, J. Anderson, 2008) system each thread is extremely lightweight; each has just several registers. To change between threads, there is no need, to store the registers manually. Because of that the change between threads has practically no delay.

The main drawback of a CUDA system is the high memory latency. There is just a small cache block available to each CUDA core, and the read delay from global memory is high (400-800 cycles). This latency can be hided with the run of a lot of threads in a core in one time, so something can be done until the other threads are waiting for the global memory.

In Fermi architecture on each Cuda streaming processor (SM) is 32 cores, the same code runs in one time in each core of an SM (Wrap). To each streaming processor belong a shared memory, which all threads can read and write.

The 40-tap FIR filter has been also realized in CUDA. In the test program, a block of 2^{18} samples has been used for filtering. After some experimentation it turned out that the use of 2^{18} threads optimizes performance on the Kepler architecture. So the kernel function is very simple calculating just the 40 taps of a single sample. Several versions of this Kernel have been implemented to test out different aspects of the hardware and CUDA compiler. The simplest version is given below to illustrate such a CUDA kernel.

```
__global__ void kernel_gpuTest_3(float* x)
{
        const float c[40]={...};

        int idx = blockIdx.x * blockDim.x + threadIdx.x;

        x[idx+262144]=x[0]*c[0];
        for(int i=1; i<40; i++)
                x[idx+262144]+=x[idx-i]*c[i];
}
```

Other kernel versions try to enhance performance to optimize the use of registers, the caches, access to the memory, use of texture memory for the coefficients, loop unrolling, etc. The best version has been used for comparison with the simple kernel.

It is also to mention that intentionally the measurements do not take the transfer times between CPU and GPU into account, since it is assumed that a real implementation will always try to add additional functionality and code to the GPU in order to hide the transfer overhead.

In the following, the kernels has been executed on 3 different GPU platforms. The specification (from “deviceQuery”) is given below:

Quadro K1000M

```
CUDA Driver Version / Runtime Version          5.5 / 5.5
CUDA Capability Major/Minor version number:    3.0
Total amount of global memory:                 2048 MBytes (2147483648 bytes)
( 1) Multiprocessors, (192) CUDA Cores/MP:     192 CUDA Cores
GPU Clock rate:                                851 MHz (0.85 GHz)
Memory Clock rate:                             900 Mhz
Memory Bus Width:                              128-bit
L2 Cache Size:                                 262144 bytes
Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
Total amount of constant memory:              65536 bytes
Total amount of shared memory per block:      49152 bytes
Total number of registers available per block: 65536
Warp size:                                    32
Maximum number of threads per multiprocessor:  2048
Maximum number of threads per block:          1024
Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
Maximum memory pitch:                         2147483647 bytes
Texture alignment:                            512 bytes
Concurrent copy and kernel execution:         Yes with 1 copy engine(s)
Run time limit on kernels:                    Yes
Integrated GPU sharing Host Memory:           No
Support host page-locked memory mapping:      Yes
Alignment requirement for Surfaces:           Yes
Device has ECC support:                       Disabled
CUDA Device Driver Mode (TCC or WDDM):        WDDM (Windows Display Driver Model)
Device supports Unified Addressing (UVA):     No
Device PCI Bus ID / PCI location ID:          1 / 0
memory bandwidth: 28.8 GB/Sec
```

GeForce GTX 680

```
CUDA Driver Version / Runtime Version         6.0 / 5.5
CUDA Capability Major/Minor version number:   3.0
Total amount of global memory:                2048 MBytes (2147483648 bytes)
( 8) Multiprocessors, (192) CUDA Cores/MP:    1536 CUDA Cores
GPU Clock rate:                               1059 MHz (1.06 GHz)
Memory Clock rate:                            3004 Mhz
Memory Bus Width:                             256-bit
L2 Cache Size:                                524288 bytes
Maximum Texture Dimension Size (x,y,z)        1D=(65536), 2D=(65536, 65536), 3D=(4096, 4096, 4096)
Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
Total amount of constant memory:              65536 bytes
Total amount of shared memory per block:      49152 bytes
Total number of registers available per block: 65536
Warp size:                                    32
Maximum number of threads per multiprocessor:  2048
Maximum number of threads per block:          1024
Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
Max dimension size of a grid size    (x,y,z): (2147483647, 65535, 65535)
Maximum memory pitch:                         2147483647 bytes
Texture alignment:                            512 bytes
Concurrent copy and kernel execution:         Yes with 1 copy engine(s)
Run time limit on kernels:                    Yes
Integrated GPU sharing Host Memory:           No
Support host page-locked memory mapping:      Yes
```

```
 Alignment requirement for Surfaces:            Yes
 Device has ECC support:                        Disabled
 CUDA Device Driver Mode (TCC or WDDM):         WDDM (Windows Display Driver Model)
 Device supports Unified Addressing (UVA):      Yes
 Device PCI Bus ID / PCI location ID:           3 / 0
 memory bandwidth: 192 GB/Sec
```

GeForce GTX 480

```
 CUDA Driver Version / Runtime Version          6.0 / 5.5
 CUDA Capability Major/Minor version number:    2.0
 Total amount of global memory:                 1536 MBytes (1610612736 bytes)
 (15) Multiprocessors, ( 32) CUDA Cores/MP:     480 CUDA Cores
 GPU Clock rate:                                1401 MHz (1.40 GHz)
 Memory Clock rate:                             1848 Mhz
 Memory Bus Width:                              384-bit
 L2 Cache Size:                                 786432 bytes
 Maximum Texture Dimension Size (x,y,z)         1D=(65536), 2D=(65536, 65535), 3D=(2048, 2048, 2048)
 Maximum Layered 1D Texture Size, (num) layers  1D=(16384), 2048 layers
 Maximum Layered 2D Texture Size, (num) layers  2D=(16384, 16384), 2048 layers
Total amount of constant memory:                65536 bytes
 Total amount of shared memory per block:       49152 bytes
 Total number of registers available per block: 32768
 Warp size:                                     32
 Maximum number of threads per multiprocessor:  1536
 Maximum number of threads per block:           1024
 Max dimension size of a thread block (x,y,z): (1024, 1024, 64)
 Max dimension size of a grid size    (x,y,z): (65535, 65535, 65535)
 Maximum memory pitch:                          2147483647 bytes
 Texture alignment:                             512 bytes
 Concurrent copy and kernel execution:          Yes with 1 copy engine(s)
 Run time limit on kernels:                     Yes
 Integrated GPU sharing Host Memory:            No
 Support host page-locked memory mapping:       Yes
 Alignment requirement for Surfaces:            Yes
 Device has ECC support:                        Disabled
 CUDA Device Driver Mode (TCC or WDDM):         WDDM (Windows Display Driver Model)
 Device supports Unified Addressing (UVA):      Yes
 Device PCI Bus ID / PCI location ID:           2 / 0
 memory bandwidth: 133 GB/Sec
```

The measured values represent the 40-tap filter kernel for 2^{18} samples run 10 times. In the following table, the measured results are compared. In addition, a relation to the theoretical computational power (CUDA cores * GPU Clock rate) of the GPU is given.

GPU	Kernel	Measured time in ms	Measured throughput in Msamp/s	GPU theoretical performance in Gflops/s	Flops/s/sample
Quadro K1000M	simple	28.68	91	163.4	1796
	optimized	9.91	265		617
GeForce GTX 680	simple	3.39	773	1626.6	2104
	optimized	1.42	1846		881
GeForce GTX 480	simple	2.12	1237	672.5	544
	optimized	0.54	4855		139

As shown in the table the GTX 480 delivers the best performance. The interesting aspect is that the GTX 480 is a Fermi architecture, which obviously performs for this algorithm better than the Kepler architecture used by the other two GPUs. The reason for that has been identified as a memory bottleneck due to unaligned memory access, which is twice the size at the Kepler architecture. Here, it was not possible to hide this memory latency with the 40 tap filter.
The intended GPU cards are Nvidia Tesla cards. For instance the Tesla K10 has a memory bandwidth of 320Gbit/s and 3072 CUDA cores. This is also true for Fermi based Tesla cards. So at least twice the performance of the GTX 680 can be expected.
In addition, the memory transfer speeds between CPU and GPU have been measured. The average duration of a transfer for 2^18 samples was on each platform and each direction around 0.4ms. (Please note that the kernel function has been invoked 10 times on the transferred data.) This means that the transfer rate is around 600Msamples/s. It was possible to increase this value by use of a pinned memory and by use of “half floats” (16 bit numbers), resulting in around 2000Msamples/s. Furthermore it is possible to hide the resulting latency by concurrent execution of transfer and GPU processing, which is possible on

today’s GPUs. However, compared to the processing in the GPU, this is still a relevant time factor.

The conclusion of the GPU evaluation was that

- Implementation of existing algorithms from normal C++ code requires significant redesign. It is not sufficient to port just some small critical sections to the GPU. If a GPU is used, a bigger portion of functionality must be shifted to the GPU in order to hide transfer bottlenecks between CPU and GPU.
- In addition, for the GPU a memory design must be developed carefully in order to optimize transfer speeds between CPU and GPU as well as memory access in the GPU core.
- Optimization with respect to the GPU hardware requires a lot of detailed GPU architecture knowledge and experimentation. The achievable performance cannot be really estimated beforehand, which makes planning of the required resources risky. “Speed ups” of a factor of 5-10 compared to 4-core Intel CPUs seems to be typical for signal processing application.

As it can be seen, the best implementation use 139 Flop/sample, and the theoretical minimum is 80. So about the 58 percent of theoretical maximum performance is achieved in this case.

6.5 Recommended system plan

Worst case computation needs are about a maximum of 1000 GFlops. As it has been shown, the modern PC with GPU-s have enough resources to demodulate the signal from software.

Modern PC performance:

CPU -86.4 GFlops – I7-3820M

GPU- 1626 GFlops – GTX680

It has to be taken into consideration that the maximum performance cannot be reached; the plan of an efficient implementation of an algorithm is a highly complex task, so it is recommended to plan the system architecture in a scalable manner. The real size of the system can be easily calculated after the implementation of the algorithms.

A recommended scalable architecture:

Basic idea:

- A master PC records the signal, and cuts it to frames
- The slave PCs demodulate the frames and return them to master
    - communication between the nodes with Ethernet 10 Gbit

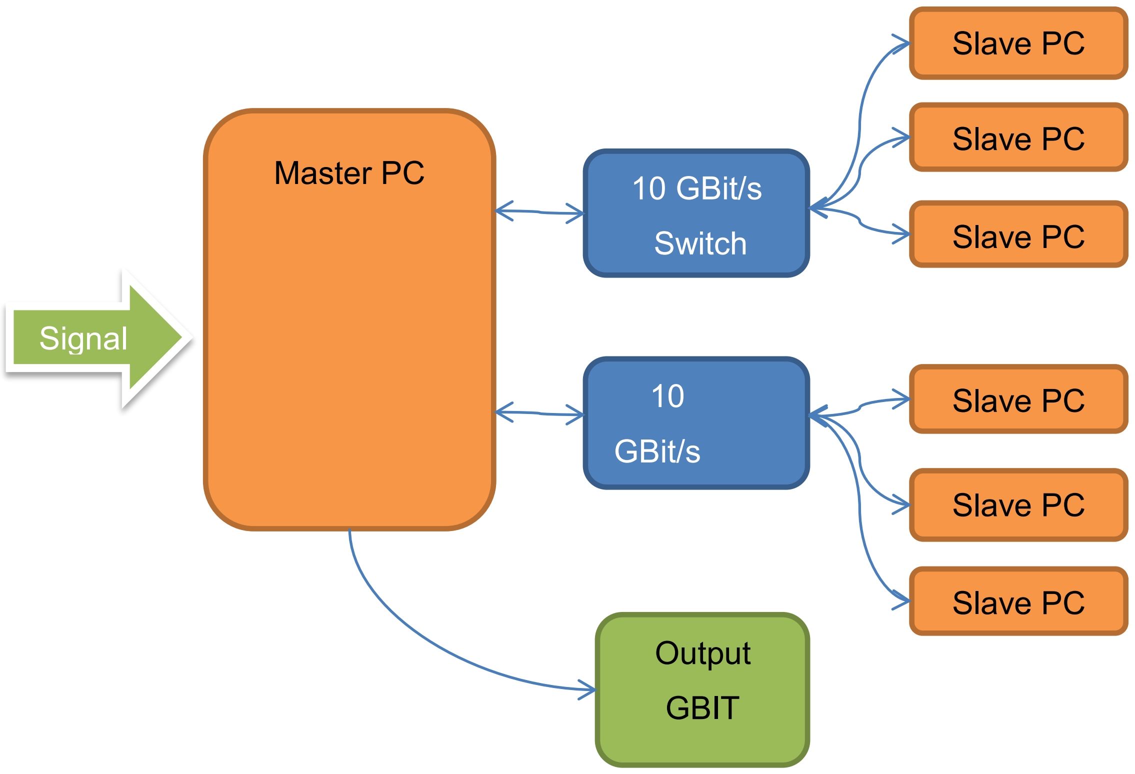


Figure 86: Recommended scalable architecture

2*10 GBit switches are needed, because the theoretical maximum bandwidth of the signal is 320Msample*2(IQ)*16 bit/sample= 10240 Mbit

Capture method in Master PC:

PCI-E FPGA based capture card

Master PC software

- Doppler correction
- Channel filter (FIR)
- Search for frequency error, and ASM bit pattern
- Cut the signal to frames, each frame has a whole sequence from ASM to ASM
- Send the numbered frames to slave PCs for demodulation
- Collect the results from slave PCs and save them
- For request sends the data's, statistics to a separated network

Slave PCs software

- Demodulates each frame, and sends the results back.

In many cases a simple PC can be enough for all tasks. With this architecture redundancy can be also achieved without doubling all resources. In calculating the system size it has to be taken into consideration that until frequency error is not corrected, and frames are not found by main computer, the demodulation cannot be done. The throughput of the system has to be slightly greater as theoretically needed.

7. Conclusion

This thesis presents a possible solution for developing a software defined radio based modem for low speed space communication. The simulated communication schemes are defined by the CCSDS standard.

A Matlab based simulator has been developed to evaluate algorithms, used for the simulation of a communication systems. Algorithms have been evaluated for channel coding (convolutional code, Reed-Solomon code, concatenated codes), pulse code modulation, pulse shaping filters (RRC, IIR, Gaussian), for modulation, and demodulation (PSK, 4D-8PSK-TCM, OQPSK, GMSK), and for synchronization (attached synch marker, non-data aided, and data-aided synchronization algorithms for PSK as well as GMSK systems).

The individual simulations show that a system built from these algorithms performs well enough to satisfy the requirements. Some tests of the whole systems also confirm this statement (OQPSK, 8-PSK, GMSK).

The worst case computation needed for real-time processing is about 1000 GFlops. It has been shown that a modern CPU or GPU alone is not enough, that’s why a parallel system with more CPUs, GPUs, and the parallel organization of the above mentioned algorithms is inevitable. A system with more individual PCs and high speed network is recommended over a single multi CPU, GPU system, to be able to provide redundancy, and a possibility for on the fly maintenance.

8. Literature

A. J. Viterbi, A. M. Viterbi. (1983). Nonlinear Estimation of PSK-Modulated Carrier Phase with Application to Burst Digital Transmission. *IEEE Trans. Inform. Theory, 29*(7), 543-551.

C. C. Bissell, D. A. Chapman. (1992). *Digital Signal Transmission.* Cambridge University Press.

CCSDS 130.1-G-1. (n.d.). *TM Synchronization and Channel Coding - Summary of Concept and Rationale.*

CCSDS 413.0-G-2. (n.d.). *Bandwidth-Efficient Modulations.*

D. A. Patterson, J. L. Hennessey. (1998). *Computer Organization and Design: The Hardware/Software Interface.* Morgan Kaufmann Publishers.

ECSS-E-ST-50-01C. (n.d.). *Space Data Links - Telemetry Synchronization and Channel Coding.*

ECSS-E-ST-50-05C. (n.d.). *Radio Frequency and Modulation.*

G. D. Forney. (1973). The Viterbi Algorithm. *Proceedings of the IEEE*(61), 268–278.

H. J. Blinchikoff, A. I. Zverev. (2001). *Filtering in the Time and Frequency Domains.*

J. G. Proakis. (1995). *Digital Communications (3rd ed.).* McGraw-Hill.

J. He, Z. Wang, H. Liu. (2010). An Efficient 4-D-8PSK-TCM Decoder Architecture. *IEEE Trans. on Very Large Scale Integration (VLSI) Systems, 18*, 808-817.

L. W. Couch. (1997). *Digital and Analog Communication Systems.* Prentice-Hall.

M. Garland, S. Le Grand, J. Nickolls, J. Anderson. (2008). Parallel Computing Experiences with CUDA. *IEEE Micro, 4*(28), 13-17.

M. Oerder, H. Meyr. (1988). Digital Filter and Square Timing Recovery. *IEEE Trans. Commun., 36*(5), 605–612.

R. Abello, N. James, R. Madde. (2007). GMSK Demodulator Implementation for ESA Deep-Space Missions. *Proceedings of the IEEE, 95*(11), 2132-2141.

S. Pasupathy. (1979). Minimum Shift Keying: A Spectrally Efficient Modulation. *IEEE Communications Magazine*, 14-22.

Wikipedia. (2014, 10 13). *www.wikipedia.org*. Retrieved 10 13, 2014, from http://en.wikipedia.org/wiki/Reed%E2%80%93Solomon_error_correction

9. Appendix

1. GMSK detector

```
function out = detector(signal_in, TX_filter_coeff_b)
%#codegen

%R_search_depth: sliding window size
R_search_depth=5;
shape_mask=ones(R_search_depth,1); % mask for shaping the estimation

%Initialisation
Upsample_factor=8;
T_Group_delay=2;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate the bit patterns
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Generate all possible bit pattern
s=0:2^R_search_depth-1;
bits = de2bi(s,R_search_depth);
bits=transpose(bits);

% NRZ
bits=(bits-0.5).*2;

% Add zeros to the end
sequences=[bits/Upsample_factor*pi/2 ; zeros(T_Group_delay,numel(s))];

% Upsample, and filter

%Zero padding
sequences = upsample(sequences,Upsample_factor);
%Upsample filter
sequences = filter(ones(1,Upsample_factor),1,sequences);
%Gauss filter
sequences = filter(TX_filter_coeff_b,1,sequences);
```

```
%cut the sequences to the same size as cutted from the signal
sequences=sequences(T_Group_delay*Upsample_factor:end,:);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Liklehood of the signal
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%initialize the received vector
received_vector=zeros(floor(numel(signal_in)/Upsample_factor),1);


vector=(zeros(numel(s),1));
for i=0:floor(numel(signal_in)/Upsample_factor)-R_search_depth-3;
    %multiply, and sum the sequence with the signal
   ma-
trix=repmat(signal_in((1:Upsample_factor*(R_search_depth)+1)+Upsample_
factor*i),1,numel(s))-sequences;
    %vector square
    for q=1:numel(s)
      vector(q)= transpose(matrix(:,q))*matrix(:,q);
    end
    %choose the minimal
    [value_of_max,position_of_min] = min((vector));
    %save the coeffitiens multiplied with the shape mask
    re-
ceived_vector(i+1:i+R_search_depth)=received_vector(i+1:i+R_search_dep
th)+bits(:,position_of_min).*shape_mask;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Detection
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%sign of the signal, or bring coeffitiens to the covolutional decoder

out=(sign(received_vector)+1)/2;



end
```

2. Maximize energy timing estimator

```
function [signal_out , R_timing_error] = Timing_recovery(signal_in)
%Estimates Timing error with O&M algorythm (NDA), and corrects it

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Timing estimation with MaxE algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Upsample_factor=8;

%save the signal
signal_out=signal_in;
%calculates the energy of the samples
signal_in=abs(signal_in).^2;
%generates mask for the signal to choose the needed values
% for example if Upsample factor is 8 the elements are: 0,8,16,24,32
...
mask=0:Upsample_factor:numel(signal_in)-Upsample_factor;
estimation=zeros(Upsample_factor,1); % initialize estimation

%calculates the sum of the energy of the signal points in the distance
of
%the upsample factor
for i=1:Upsample_factor
    estimation(i)=sum(signal_in(mask+i));
end

%looks for the position of the biggest value
[~,R_timing_error]=max(estimation);
%removes the bias cased by the element numbering of a vector in Matlab
R_timing_error=R_timing_error-1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Timing Correction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
signal_out= circshift(signal_out,-1*R_timing_error);

end
```

3. Oerder&Meyr timing estimator

```
function [signal_out , R_timing_error] = Timing_recovery(signal_in)
%Estimates Timing error with O&M algorythm (NDA), and corrects it

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Timing estimation with Oerder and Meyr algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Upsample_factor=8;

%The description of the algrithm can be found in the 4. documentation

temp1=[1:1/Upsample_factor:(length(signal_in)-
128)/Upsample_factor]*2*pi;
temp2=abs(signal_in(1:(length(signal_in)-127-Upsample_factor)));
temp1=temp2.*temp2.*transpose(exp(-1*1i*temp1));
temp1=sum(temp1);

R_timing_error=Upsample_factor/(2*pi)*angle(temp1);



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Timing Correction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
signal_out= circshift(signal_in,round(R_timing_error));

end
```

4. Maximize energy method for OQPSK

```
function [signal_out , R_timing_error] = Timing_recovery(signal_in)
%Estimates Timing error with O&M algorythm (NDA), and corrects it

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Timing estimation with Oerder and Meyr algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Upsample_factor=8;
```

```
%Initialisation

signal_out=signal_in;
signal_i=real(signal_in).^2;
signal_q=imag(signal_in).^2;
mask=0:Upsample_factor:numel(signal_in)-Upsample_factor;

estimation=zeros(Upsample_factor*2,1);
estimation_i=zeros(Upsample_factor,1);
estimation_q=estimation_i;

%Calculate maxE method separately for I, and Q

for i=1:Upsample_factor
    estimation_i(i)=sum(signal_i(mask+i));
    estimation_q(i)=sum(signal_q(mask+i));
end

% summarizes the elements with half symbol difference

for i=1:Upsample_factor
    estima-
tion(i)=estimation_i(i)+estimation_q(mod((i+Upsample_factor/2),Upsampl
e_factor)+1) ;
end

%%%%%%%%%%%%%%%%%%%%
% Pi/4 shift
%%%%%%%%%%%%%%%%%%%%
% calculates the same with pi/4 shift of the signal

%pi/4 shift
 signal_in=signal_in*exp(-1i*pi/4);
%energy
 signal_i=real(signal_in).^2;
 signal_q=imag(signal_in).^2;

%Calculate maxE method separately for I, and Q
```

```
for i=1:Upsample_factor
    estimation_i(i)=sum(signal_i(mask+i));
    estimation_q(i)=sum(signal_q(mask+i));
end

% summarizes the elements with half symbol difference

for i=1:Upsample_factor
    estima-
tion(i+Upsample_factor)=estimation_i(i)+estimation_q(mod((i+Upsample_f
actor/2),Upsample_factor)+1) ;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Estimation of timing error
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[~,R_timing_error]=max(estimation);
R_timing_error=mod(R_timing_error-1,Upsample_factor/2);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Timing Correction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

signal_out= circshift(signal_out,-1*R_timing_error);



end
```

5. Correction of the remaining symbol timing error after phase error is corrected

```
function [signal_out, out]  = Timing_recovery(signal_in)
%Estimates Timing error with O&M algorythm (NDA), and corrects it

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Timing estimation with Oerder and Meyr algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Upsample_factor=2;



mask=1:Upsample_factor:numel(signal_in)-Upsample_factor;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Timing Correction
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%generates the two possible signal variation
signal_1=signal_in;
signal_2= circshift(signal_in,-Upsample_factor/2);

%shifts the offset bac in both case
signal_1=         real(signal_1)+         1i*circshift(imag(signal_1),-
1*Upsample_factor/2);
signal_2=         real(signal_2)+         1i*circshift(imag(signal_2),-
1*Upsample_factor/2);

% Choose the right one with maximize Energy method

if sum(abs(signal_1(mask))) > sum(abs(signal_2(mask)))
    signal_out=signal_1;
    out=1;
else
    signal_out=signal_2;
    out=2;
end



end
```

6. <u>Viterbi&Viterbi with feedback structure</u>

```
function   [signal_out   ,   R_Phase_error]   =   VV_and_phase(signal_in,
R_Phase_M, R_Phase_A, R_Phase_B)
%Estimates  phase  error  with  V&V  algorythm  (NDA),  and  corrects,  the
signal
```

```
%ABOUT:
% The argument of the signal will be raised to M-th.(in case N-PSK
N=M)
% Note: the maximal area for estimation is 2*pi/M

% Define variables
R_Phase_error=zeros(numel(signal_in),1);
signal_out=signal_in;
Phase_error_at_previous_symbol=0;

% Estimates the Phase Error, and corrects it in each points
for i=1:numel(signal_in)-1
    % Corrects next symbol phase with the previous estimated phase
    signal_out(i)=signal_in(i)*exp(-1i*R_Phase_error(i));
    % Phase estimation, and filter

Phase_error_at_symbol=(1/R_Phase_M)*angle(signal_out(i).^R_Phase_M);

      R_Phase_error(i+1)=R_Phase_error(i)+R_Phase_A*Phase_error_at_sym
      bol+( R_Phase_B-R_Phase_A)*Phase_error_at_previous_symbol;

    % Saves the states for next cycle
    Phase_error_at_previous_symbol=Phase_error_at_symbol;
 end
end
```

7. Find ASM

```
function                                        [state,output,max_value]=
find_ASM(BIT_pattern,Number_of_bits,R_Enabled_bit_failures)
% Looks for ASM pattern. If finds, sets the state to 0
% The output is always the frame cutted after the sync pattern

%%%%%%%%%%%%%%%
%Constants
%%%%%%%%%%%%%%%
```

```
ASM_bit_pattern=[0 0 0 1  1 0 1 0  1 1 0 0  1 1 1 1  1 1 1 1  1 1 0 0
0 0 0 1  1 1 0 1];
num_of_zeros=128; % number of zeros inserted in sequence, read notes
                  % at Transmit/insert ASM to more info before edit!



state=1;


output=BIT_pattern(1:Number_of_bits);
%Output has to be generated in this form because with just zeros there
are no limits to size for matlab compiler
output=output*0;
%%%%%%%%%%%%%%%
%Calculations
%%%%%%%%%%%%%%%


% creates NRZ values from signal (0,1)
ASM_bit_pattern=(ASM_bit_pattern-0.5)*2;
BIT_pattern_signed=(BIT_pattern-0.5)*2;



% calculates the max value, and index of the crosscorrelation
% of ASM, and input Bit pattern


[max_value, index] = max(xcorr(BIT_pattern_signed,ASM_bit_pattern));


% If ASM is detected returns with 0
% The sequence has 32 bit lenght, so the without failure the
% max_value of cross correlation is 32. Each bit failure makes
% 2 lesser maximum in cross correlation


if (max_value >= (32-2*R_Enabled_bit_failures))
    state=0;
end


% Generates the cutted bit pattern
% The xcorr function of Matlab does zero padding,
% that is why the cutted Bit_pattern started with
% index-Number_of_bits+1 -num_of_zeros is in all place required be-
cause
% zero padding for timing error makes the signal longer.

if(index+1-num_of_zeros > numel(BIT_pattern)) % failure correction
    state=index;
else
    max_value=index-Number_of_bits+1-num_of_zeros;
    output=BIT_pattern((index-Number_of_bits+1-num_of_zeros):(index+1-
num_of_zeros));
end
end
```

8. Find ASM for DA frequency estimator

```
function [state,max_value,search]= find_ASM(BIT_pattern)
% Looks for ASM pattern. If finds, sets the state to 0
% The output is always the frame cutted after the sync pattern

%%%%%%%%%%%%%%%
%Constants
%%%%%%%%%%%%%%%

ASM_bit_pattern=[0 0 0 1  1 0 1 0  1 1 0 0  1 1 1 1  1 1 1 1  1 1 0 0
0 0 0 1  1 1 0 1];
%ASM_bit_pattern=transpose(ASM_bit_pattern);

%threshold of the signal find
treshold=0.3;

%init value for state
state=1;

%%%%%%%%%%%%%%%
%Calculations
%%%%%%%%%%%%%%%

% creates NRZ values from signal (0,1)
ASM_bit_pattern=(ASM_bit_pattern-0.5)*2;
BIT_pattern_signed=(BIT_pattern-0.5)*2;
```

```
% calculates the the crosscorrelation of ASM, and input Bit pattern

search= xcorr(BIT_pattern_signed,ASM_bit_pattern);

% Defining variables
mask=0:numel(ASM_bit_pattern):numel(search)-numel(ASM_bit_pattern);
estimation=zeros(numel(ASM_bit_pattern),1);

% In crosscorrelation vector the peaks are in a distance from frame
size
% from each other. The following algorithm summarize the peaks, and
looks
% for the correlated signal with some threshold.

for i=1:numel(ASM_bit_pattern)
    estimation(i)=sum(search(mask+i));
end

%choose the biggest value
max_value=max(estimation);
if (max_value>numel(search)*treshold)
    state=0;

end
end
```