ENSEIRB-MATMECA / LSE

Second Year Internship Report

Author:
Maxime Descos

Abstract

L'ENSEIRB-MATMECA requiert d'effectuer un stage dans un des domaines de l'électronique à la fin de la deuxième année. De plus, durant la scolarité, 12 semaines à l'étranger sont obligatoires.

Ces deux exigences sont là pour permettre aux élèves d'acquérir une expérience pratique et une meilleure compréhension des méthodes de travail dans les autres pays. Cela permet d'apprendre à travailler dans un milieu donné (laboratoire, entreprise, start-up) mais aussi d'apprendre les différences entre nos systèmes de travail et de pensée par rapport à ceux qui se fait ailleurs.

J'ai donc choisi de remplir ces deux conditions lors de mon stage de deuxième année. C'est ainsi que j'ai effectué ce stage dans le laboratoire de systèmes embarqués (LSE) de la Faculté d'Ingénieur de l'Université de Buenos Aires (FIUBA) en Argentine.

Durant ce stage, j'ai appris comment fonctionne un système de communication numérique et comment le modéliser sur ordinateur. J'ai précisément utilisé Octave comme environnement de développement.

Ce stage m'a permis de continuer le cours d'introduction aux systèmes de communication que j'ai eu à l'ENSEIRB-MATMECA. J'ai aussi pu améliorer mec connaissances en Matlab/Octave. Enfin, j'ai eu la chance de découvrir un nouveau pays et une nouvelle culture. Ce fut une expérience extrèmement enrichissante qui m'a donné envie de continuer de travailler à l'étranger ou du moins, en collaboraton avec d'autres pays.

Contents

1 Acknowledgments

First of all, I would like to thank Mr Frederico Zacchigna for the opportunity to work and live in Argentina and for the help he gave me during my internship. I also would like to thank Mr Ariel Lutenberg for accepting me in the laboratory. Finally, I have to thank my parents for all the help they gave and give me for all my projects.

2 Introduction

My second-year internship was conducted in the laboratory of embedded systems of the university of Buenos Aires, Argentina. The subject of the internship was to develop a digital communication system using QAM modulation.

To work and live in South America was one thing I wanted to do, so naturally I tried to find a internship at the end of my second year in ENSEIRB-MATMECA in this area. With high school, I learned a bit of spanish and I thought it will be easier to communicate if I knew the language, at least a bit, so I remove Brasil from the country list where I could do my internship. After, I did some researches on the other countries, I learned that there is an important engineering university in Argentina. Furthermore, I am really interested by embedded systems, I would like to specialize myself in this field and the university of Buenos Aires has a laboratory of embedded systems, so I sent some emails to the people who work in this university and Mr Frederico Zacchigna has accepted my internship.

My internship was conducted from June 1st until August 13th. The short duration of my internship is due to my exchange year at the Illinois Institute of Technology in Chicago where classes start mid-August.

He gave me the following subject : "To design a wireless digital communication system". I had a course on digital communication during this second year in my school, so I knew on what I would have to work. The work plan was firstly to understand exactly how a communication system is composed of. That is the first step to design the all system. After that, I had to found an algorithm for each block and simulate it with all the other blocks of the system

I used the Octave sofware to simulate the system.

3 Laboratorio de Sistemas Embebidos

The Laboratorio de Sistemas Embebidos is a part of the Facultad de Ingeneria of the Universidad de Buenos Aires. It was created in 2010 and there is currently 40 workers in it. This work fields are digital system design and sensor network. This laboratory mostly does researches and teaching in the university. It is financed

by the government of Argentina and by companies that use the laboratory as a consultant such as NXP or ARM.

I only worked with Mr Zacchigna, an engineer and my tutor, and Mr Lutenberg, a researcher and the director of the laboratory.

The laboratory is composed of several computers and electronic boards with all the tools to work on them.

4 Goal of the internship

4.1 Subject

As said before, my subject was to design a digital communication system using QAM modulation. To design such system, every part has to be understand and the ability to see the all system is also mandatory to see where the project goes. Specifications will be told in this report.

This project can be used to study a more complex digital communication system and to implement the simulation on an electronic board to understand the differences between the model and the reality.

4.2 Specifications

There is several kind of modulation : QAM, QPSK, OFDMA. For this internship, QAM was used because it is a simple modulation but however, it allows to demonstrate the good behaviour of the system and works very well.

Quadrature amplitude modulation (QAM) is both an analog and a digital modulation scheme. It conveys an analog message signal, or a digital bit stream, by changing (modulating) the amplitudes of two carrier waves, using the amplitude-shift keying (ASK) digital modulation scheme or amplitude modulation (AM) analog modulation scheme. The two carrier waves of the same frequency, usually sinusoids, are out of phase with each other by 90 degrees and are thus called quadrature carriers or quadrature components — hence the name of the scheme. The modulated waves are summed, and the final waveform is a combination of both phase-shift keying (PSK) and amplitude-shift keying (ASK), or, in the analog case, of phase modulation (PM) and amplitude modulation.

4.3 Work plan

The work plan was easy to describe. The first step is to study the modulations, how it works and particularly the QAM modulation. After a good understanding of this modulation, the next step is to study how a digital communication system is

made of. The number of blocks, the interactions between this blocks, the input and output of the system, the tests to check if the system works well, all of this should be designed before coding the system.

5 Project progress

5.1 Modulation

Quadrature Amplitude Modulation or QAM is a form of modulation which is widely used for modulating data signals onto a carrier used for radio communications. It is widely used because it offers advantages over other forms of data modulation such as PSK, although many forms of data modulation operate alongside each other.

When using QAM, the constellation points are normally arranged in a square grid with equal vertical and horizontal spacing and as a result the most common forms of QAM use a constellation with the number of points equal to a power of 2 i.e. 4, 16, 64 . . . .

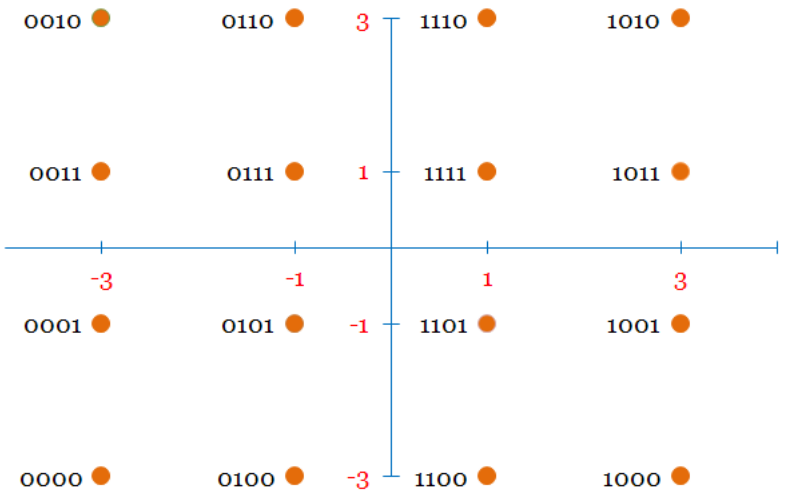

Figure 1: 16-QAM Constellation

Although QAM appears to increase the efficiency of transmission for radio communications systems by utilising both amplitude and phase variations, it has a number of drawbacks. The first is that it is more susceptible to noise because the states are closer together so that a lower level of noise is needed to move the signal to a different decision point.

5.2 Study of Digital Communication Systems

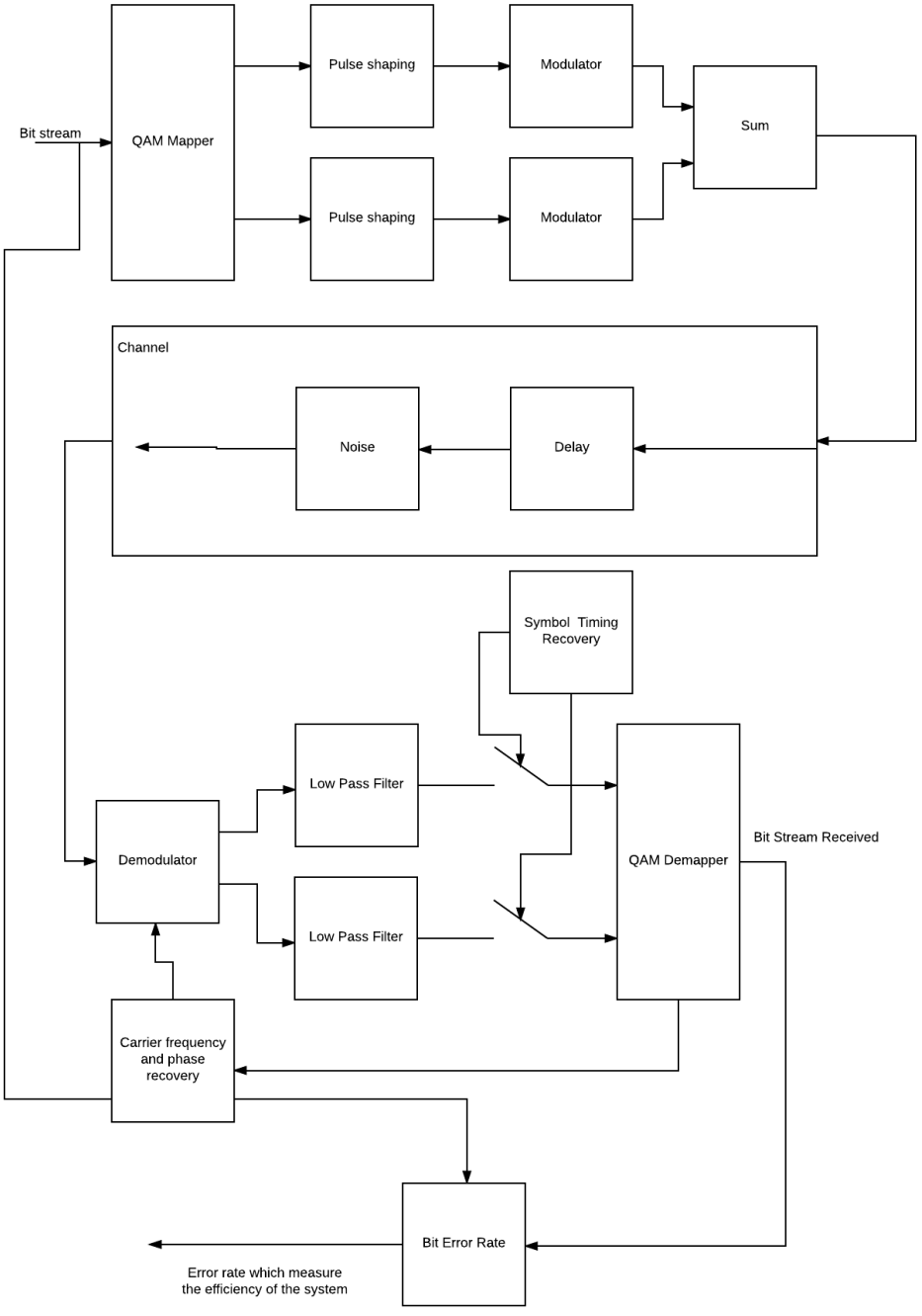

Figure 2: Digital Communication System

A digital communication system can be divided in three subsystems: transmitter, channel, receiver. Each subsystems is composed of different blocks.

The transmitter will receive a bit stream. Each group of bits (2 for a 4QAM, 4 for a 16QAM) will be transformed in a point on the constellation by the QAM Mapper (as seen on figure 1). Now, the system can be decomposed in two chains: one for the real mathematical part of the point and one for the imaginary mathematical part. These two chains will be the same. After that, we have a pulse shaping which make the transitions between the each point smoother and convert the digital signal in analog. Hence, the modulator which put the signal in the right frequency for the transmission. And finally, the real and imaginary parts are summed into one signal and can be transmitted.

Now the signal will travel in a particular channel. Each different channels add a different delay and a different noise. This parameter should be acknowledged to design a system which can retrieve the bit even with a noise and a delay.

After that, there is the most difficult system to design: the receiver. It works almost as the transmitter reversed. Firstly, there is the demodulator to divide the system in the real and imaginary part. Then a low pass filter to eliminate the noise and the wrong harmonics. The QAM Demapper will convert the constellation point into a group of bits. Two other blocks are used. One to know which point of the analog signal should be take as the digital signal, it is the symbol timing recovery. And the other which should retrieve the frequency and the phase of the carrier to separate the useful signal from the carrier.

Finally, a block will test the system by finding how many differences there is in the transmitter bit stream and the receiver bit stream. This block is useful to describe the system and the channel. This error rate should be small, even null if the system is well designed and a channel without too much noise.

5.3 QAM Mapper

As said before, the bit stream has to be change in constellation points. This block was implemented in the draft1.

The bit stream is firstly shrink into a smaller array with one group of bits in each cell of the array. There is two bits for a 4QAM and 4 bits for a 16QAM. Each group of bits is transformed in a decimal number and this number will be used as the index of the symbol representing the bits group in the mapping table. A symbol is the name of a constellation point and the mapping table is a table with the value of all the symbol.

After that, a bit group is now a symbol with a real and a imaginary part. This symbol can be visualised in a constellation. So, the two parts are given to two identical subsystems to do the work in the same time.

5.4 Pulse Shaping

First of all, the two parts are only numbers. So, to have a signal, the two parts have to become a signal on a defined time period. To do that, the pulse shaping multiplies the symbol with a pulse. This pulse can be only flat signal with a amplitude of one to begin. This pulse is really simple and not really good but is perfect to implement the other blocks.

Also, modulation of a carrier sinusoid results in constant transitions in its phase and amplitude. The sharp transitions in any signal result in high-frequency components in the frequency domain. It is clear that phase/amplitude transitions occur at every two periods of the carrier and sharp transitions occur, when filtering is not applied.

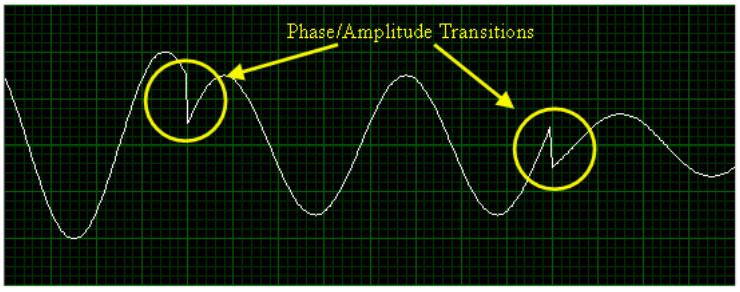

Figure 3: With a flat pulse

By applying a pulse shaping filter to the modulated sinusoid, the sharp transitions are smoothed and the resulting signal is limited to a specific frequency band. As a result, the frequency information of the sinusoid becomes more concentrated into a specified frequency band.

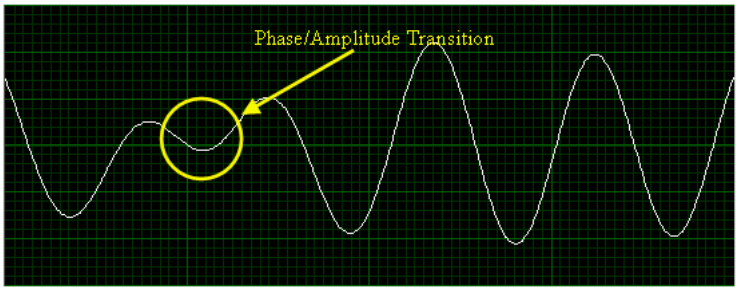

Figure 4: With a sinusoidal pulse

The sharp transitions do cause high frequency components in the frequency domain. Now, once a filter has been applied to the carrier signal, these high frequency

components of the signal have been removed. Thus, the majority of the channel power is now limited to a specific defined bandwidth. It is clear that the required bandwidth for a channel is directly related to the symbol rate and is centered at the carrier frequency.

5.5 Modulator And Sum

This part is really simple. There is two inputs : the two subsignal : real and imaginary. Each part are multiplied by a cosine/sine at the carrier frequency and phase.

The output will be the final signal which can be transmitted with a wire or an antenna.

5.6 Channel

The channel can be a lot of different things. Air, coaxial cable, water, space void. Besides the different kind of channel, many other things can disrupt the channel : other transmission, strong magnetic field, obstacle. All of this cause an imperfect channel. The channel will disturb the signal. Noise will be added and delay too obviously.

Physically, the additive noise process may arise from electronic components and amplifiers at the receiver of the communication system or from interference encountered in transmission (as in the case of radio signal transmission). Hence, the signal will not be the same as what the transmitter sent. The receiver will have to work with a new signal and try to recover the initial signal.

To model this behaviour, noise is added with the Octave awgn function. The input x is a complex voltage signal. The returned value y will be the same form and size as x but with Gaussian noise added.

After the channel, the signal can be retrieve by the receiver with an antenna or another way.

5.7 Demodulator and Low Pass filter

The QAM demodulator is very much the reverse of the QAM modulator.

The signals enter the system, they are split and each side is applied to a mixer. One half has the in-phase local oscillator applied and the other half has the quadrature oscillator signal applied.

In practice, there is an unknown phase delay between the transmitter and receiver that must be compensated by synchronization of the receivers local oscillator.

Now, the signal is in base band but the multiplication gives also another harmonic. To remove this one, a low pass filter is needed. The technique used here is

to go into the frequency domain using a Fast Fourier Transform, and after, a low pass filter can be easily applied to the signal.

This filter is rather simple, it doesn't modify the low frequency (1 is the multiplication coefficient) and cut all the other frequencies (0 is the multiplication coefficient). It works well but it is only good for a model. In the physical reality, a low pass filter which cut perfectly all the non wanted frequencies and keep unchanged the right frequency is impossible.

Furthermore, the influence of the pulse shaping as to be removed. The simplest way to do that is to divide the signal by the pulse shaping. It means that the receiver should know what pulse shape has been used but it is mandatory. In reality, the same pulse shape will always be used, so it can be hard coded in the transmitter and the receiver.

5.8 Symbol Timing Recovery

Now, the real part and the imaginary part should be almost the same as in the transmitter. But it is still an analog signal and the system has to choose which point take as the digital value. It is the role of the Symbol Timing Recovery.

Different algorithms exist for symbol timing recovery and synchronization. An "Early/Late Symbol Recovery algorithm" is used here.

The algorithm starts by selecting an arbitrary sample at some time (denoted by T). It captures the samples before and after T. The samples before T are called Early Samples and the samples after are called Late Samples. The timing error is generated by comparing the amplitudes of the sum of the early and late samples. The next symbol sampling time instant is either advanced or delayed based on the sign of difference between the early and late samples.

1) If the $EarlySample = LateSample$: The peak occurs at the on-time sampling instant T. No adjustment in the timing is needed.

2) If $EarlySample > LateSample$: Late timing, the sampling time is offset so that the next symbol is sampled before the current sampling time.

3) If $EarlySample < LateSample$: Early timing,the sampling time is offset so that the next symbol is sampled seconds after the current sampling time.

The figure only uses one sample before and after. In the system, all the samples before and after are used to have a better evaluation.

But, to do that, dummy symbols have to be send before the bit stream to be sure to take the right sample after, during the real communication.

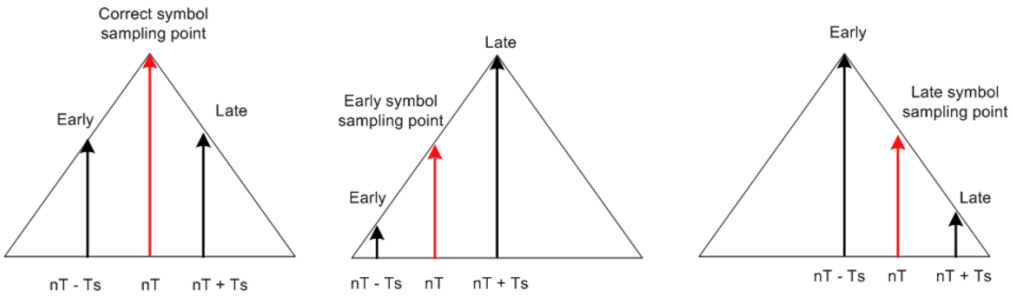

Figure 5: Late Early Algorithm

5.9 QAM Demapper

As in the transmitter, symbol now should be change into a bit stream. It is the reverse algorithm, the symbol is converted in a decimal number, this number will be the index of the demapping table and the bit stream is retrieved.

But the noise may have modified the exact value of the the amplitude and phase. So, the symbols are now like the figure below:

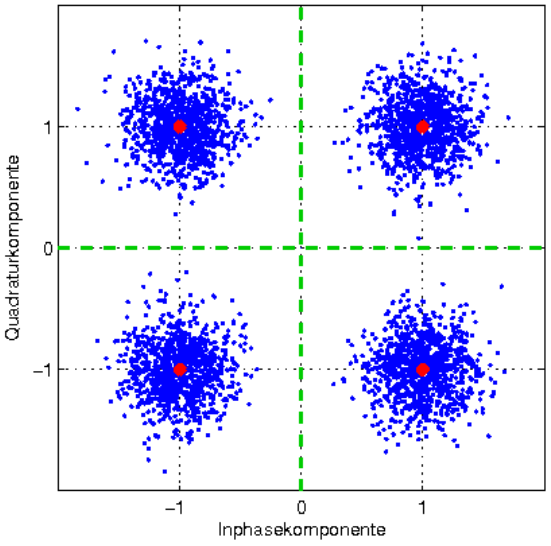

Figure 6: Receiver constellation points

An algorithm to decide which is the corresponding symbol from the point received is needed. Many algorithms exist. The simplest algorithm was used here. The goal was to move foreward to have a complete system. The differences between the point

and all the possible symbol values are calculated and the symbol with the minimal difference was taken as the symbol received.

It is not the best way to achieve that but with rather small noise, this algorithm works well.

5.10 Carrier Frequency and Phase Recovery

A carrier recovery system is a circuit used to estimate and compensate for frequency and phase differences between a received signal's carrier wave and the receiver's local oscillator for the purpose of coherent demodulation.

In the transmitter of a communications carrier system, a carrier wave is modulated by a baseband signal. At the receiver the baseband information is extracted from the incoming modulated waveform.

In an ideal communications system, the carrier signal oscillators of the transmitter and receiver would be perfectly matched in frequency and phase thereby permitting perfect coherent demodulation of the modulated baseband signal.

However, transmitters and receivers rarely share the same carrier oscillator. Communications receiver systems are usually independent of transmitting systems and contain their own oscillators with frequency and phase offsets and instabilities. Doppler shift may also contribute to frequency differences in mobile radio frequency communication systems.

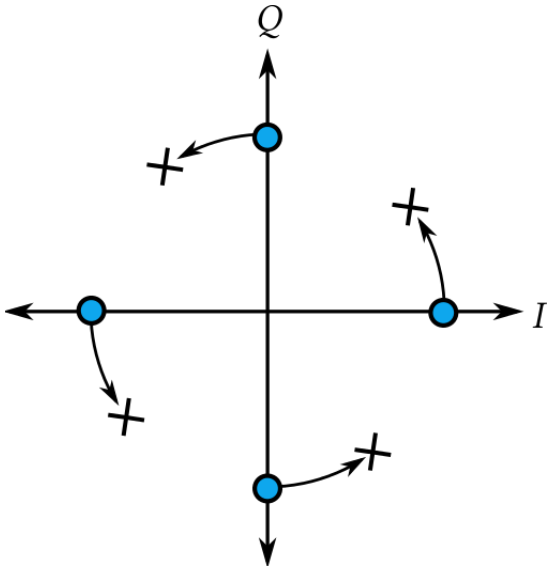

Figure 7: Example of QPSK carrier recovery frequency error causing rotation of the received symbol constellation, relative to the intended constellation

All these frequency and phase variations must be estimated using information in the received signal to reproduce or recover the carrier signal at the receiver and permit coherent demodulation.

Carrier frequency and phase recovery as well as demodulation can be accomplished using a Costas loop of the appropriate order. A Costas loop is a cousin of the PLL that uses coherent quadrature signals to measure phase error. This phase error is used to discipline the loop's oscillator. The quadrature signals, once properly aligned/recovered, also successfully demodulate the signal.

In the classical implementation of a Costas loop, a local voltage-controlled oscillator (VCO) provides quadrature outputs, one to each of two phase detectors, e.g., product detectors. The same phase of the input signal is also applied to both phase detectors and the output of each phase detector is passed through a low-pass filter. The outputs of these low-pass filters are inputs to another phase detector, the output of which passes through noise-reduction filter before being used to control the voltage-controlled oscillator. The overall loop response is controlled by the two individual low-pass filters that precede the third phase detector while the third low-pass filter serves a trivial role in terms of gain and phase margin.

An example of a VCO can be found in the vco.m file. This VCO was working but not used in the final design.

Finally, the phase error was given by the multiplication of the signal and a sin at the supposed frequency and phase but in quadrature. The phase is now known with the last phase and the error. The frequency error is the difference between the actual phase and the last phase. So, the right frequency is known too.

All of this is done at the beginning during a synchronization period when only dummy symbols are send to calibrate the frequency and phase. A better system will be to change the frequency and phase during the communication.

5.11 Bit Error Rate

When data is transmitted over a data link, there is a possibility of errors being introduced into the system. If errors are introduced into the data, then the integrity of the system may be compromised. As a result, it is necessary to assess the performance of the system, and bit error rate, BER, provides an ideal way in which this can be achieved.

Unlike many other forms of assessment, bit error rate, BER assesses the full end to end performance of a system including the transmitter, receiver and the medium between the two. In this way, bit error rate, BER enables the actual performance of a system in operation to be tested, rather than testing the component parts and hoping that they will operate satisfactorily when in place.

The bit error rate compare the bit stream from the transmitter and the receiver

and counts the difference. With a small noise and a working system, the BER should be almost null.

6 Designs used and progress

A first design with the essential parts can be found in the draft1 file. This system only has a basic transmitter without pulse shaping, a channel and a receiver with the demodualtion and the demapper. It is a theoretic system but was used to create the mapper and demapper.

The second design (draft4) is much more complete. Almost all the blocks are in it. The transmitter is the final transmitter and has everything. It is the same for the channel. The receiver has some carrier frequency recovery but not totally working.

The VCO file implements a VCO used in a Costas loop and in a Phase Locked Loop. This VCO is working and can be used in different system to lock a signal.

The draft5 is a almost complete communication system. It is working if there is not too much noise on the signal. The only thing missing is the symbol timing recovery. The digital value is given by averaging the signal. It is not perfect but works to prove that the system is working.

Finally, the final design is the systemWithPLL file. This system has all the parts of a digital communication.

7 Conclusion

The final design is working good in the simulation. It was the goal to achieve. There is a lot of flaws in the current system but it is a first step in the digital communication model.

Apart from the fact that this internship was perfect to continue my course of digital communication, it allowed me to work on rather important Matlab/Octave project. Even though I worked on this software during my studies, it was only for small and short duration projects. It was great to understand the basics but a full system is better to see how to design.

Even if I have many things to learn on digital communication systems, I now gave a better understanding of the theoretical bases of this field.

I also learned how to work on a project in a laboratory environment. I t was the first time I worked in a university and it was a great experience.

For the future, it will be a good idea to work on another algorithm for the carrier frequency recovery. An algorithm which works even during the communication and not only with dummy symbols at the beginning. The next step will be to implement this system on a FPGA device and test the system in real conditions and not only in simulation.

8 Bibliography

Digital Communications, John G. Proakis & Masoud Salehi, McGrawHill Higher Education

Introducion a la teoria y sistemas de communicacion, B. Lathi, Limusa Noriega Editores

Modern Digital and Analog Communication Systems, B. Lathi, Oxford University Press

Digital Communication Signal Processing, John Cioffi

9 Appendix

9.1 draft1

```
clear all;
close all;
%clf;

%------------------ Initial Parameters ------------------

choice1 = menu ("Which number of bits in the bitstream do you want?", "
    12", "64", "102", "1020", "10200"); %number of bits in the FIFO
if choice1 == 1
  n = 12;
elseif choice1 == 2
  n = 64;
elseif choice1 == 3
  n = 102;
elseif choice1 == 4
  n = 1020;
elseif choice1 == 5
  n = 10200;
endif

choice2 = menu ("Which QAM do you want to use?", "4", "16", "64"); %
    number of symbols in the constellation
if choice2 == 1
  M = 4;
elseif choice2 == 2
  M = 16;
elseif choice2 == 3
  M = 64;
endif
k = log2(M); %number of bits in one symbol
w = 2*pi*10;  % angular frequency

if mod(n, k) ~= 0
    error('numberOfBits must be a multiple of log2(M).');
end

numSamplesPerSymbol = 1;     % Oversampling factor

%------------------Creation of the bitstream ----------

dataIn = randi(2,n,1) - 1  % Generate vector of binary data

%stem(dataIn,"filled");
%title('Random Bits');
```

```
%xlabel('Bit Index');
%ylabel('Binary Value');

%------------------- QAM Mapper ----------------------

numberOfSymbols = length(dataIn)/k; %number of symbols in the FIFO

% Define mapping table applying Gray mapping

if M == 4
  mappingTable(1) =  1 + 1*j;
  mappingTable(2) = -1 + 1*j;
  mappingTable(3) =  1 - 1*j;
  mappingTable(4) = -1 - 1*j;

elseif M == 16
  mappingTable(1:4) = -3;
  mappingTable(5:8) = -1;
  mappingTable(9:12) = +3;
  mappingTable(13:16) = +1;
  for i = 0:15
    if mod(i,4) == 0
      mappingTable(i+1) = mappingTable(i+1) -3*j;
    elseif mod(i+3,4) == 0
      mappingTable(i+1) = mappingTable(i+1) -1*j;
    elseif mod(i+1,4) == 0
      mappingTable(i+1) = mappingTable(i+1) +1*j;
    elseif mod(i+2,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +3*j;
    endif
  endfor


elseif M == 64
  mappingTable(1:8)   = + 7*j;
  mappingTable(9:16)  = + 5*j;
  mappingTable(17:24) = + 1*j;
  mappingTable(25:32) = + 3*j;
  mappingTable(33:40) = - 7*j;
  mappingTable(41:48) = - 5*j;
  mappingTable(49:56) = - 1*j;
  mappingTable(57:64) = - 3*j;

  for i = 0:63
    if mod(i+2,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +1;
    elseif mod(i+1,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +3;
    elseif mod(i+3,8) == 0
```

```
      mappingTable(i+1) = mappingTable(i+1) +5;
    elseif mod(i+4,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +7;
    elseif mod(i+6,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -1;
    elseif mod(i+5,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -3;
    elseif mod(i+7,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -5;
    elseif mod(i,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -7;
    endif
  endfor
endif

mappedSymbols = zeros(1, numberOfSymbols);

% Map bits to symbols
for i = 1:k:length(dataIn)

    symbolBits = dataIn(i:i+k-1);

    if M == 4
      symbolIndex = 2^1 * symbolBits(1) + 2^0 * symbolBits(2);
    elseif M == 16
      symbolIndex = 2^3 * symbolBits(1) + 2^2 * symbolBits(2) + 2^1 *
          symbolBits(3) + 2^0 * symbolBits(4);
    elseif M == 64
      symbolIndex = 2^5 * symbolBits(1) + 2^4 * symbolBits(2) + 2^3 *
          symbolBits(3) + 2^2 * symbolBits(4) +  2^1 * symbolBits(5) +
          2^0 * symbolBits(6);
    endif

     % Mapping
    mappedSymbols((i - 1)/k + 1) = mappingTable( symbolIndex + 1);

endfor

%%-------------------------------Pulse shaping ------------------------
%
%
%
%%-------------------------------Construction of the signal-----------
%
%signal = mappedSymbols;
%
%t = 0:1/k:length(dataIn)/(k*k)-1/k;
%modulation = real(signal).*cos(w.*t) + imag(signal).*sin(w.*t);
%
```

```
%%-----------------------------------Channel with noise addition
    -----------------
%
%EbNo = 10;
%snr = EbNo + 10*log10(k) - 10*log10(numSamplesPerSymbol);
%
%signalAfter = awgn(signal,snr,'measured');
%
%%---------------------------------- Deconstruction of the signal
    ---------------
%
%phi = 0;
%
%signalAfterDeconstructionI = signalAfter.*cos(w*t + phi);
%signalAfterDeconstructionQ = signalAfter.*sin(w*t + phi);
%
%%-----------------------------------FIR
    ------------------------------------------------
%
%% Low pass filtering with a Butterworth filter
%[b,a]=butter(2,0.3);
%Yi=filter(b,a,signalAfterDeconstructionI);
%Yk=filter(b,a,signalAfterDeconstructionQ);

%---------------------------------- QAM demapper
    ------------------------------------

receivedSymbols = mappedSymbols;
%sPlotFig = scatterplot(signal,1,0,'k*');
%hold on
%scatterplot(signalAfter,1,0,'g.', sPlotFig);
%hold on
%scatterplot(receivedSymbols,1,0,'r+', sPlotFig);
%grid;
%xlabel('I');
%ylabel('Q');
%if M == 4
%  axis([-2 2 -2 2]);
%elseif M == 16
%  axis([-4 4 -4 4]);
%elseif M == 64
%  axis([-8 8 -8 8]);
%endif
%title('Signal before sending, after sending and after filtering');
%legend('Before Sending','After Sending', 'After Filtering');


for i = 1:length(receivedSymbols)
  [mindiff minIndex] = min(receivedSymbols(i) - mappingTable);
```

```
  symbolIndexAfter(i) = minIndex - 1;
endfor

for i=1:length(symbolIndexAfter)
  bitString = dec2bin(symbolIndexAfter, k);
endfor
bitString
bitString(1)
bitString(2)
for i=1:length(bitString)
    pack =bitString(i);
%    for j=1:k
%      dataOut((i*k)-k+j ) = str2double(pack(j));
%    endfor
  endfor
```

9.2 draft4

```
close all;
clear all;
graphics_toolkit ("gnuplot");

  %---------- Choice of parameters -------- %
  choice1 = menu ("Which number of bits in the bitstream do you want?",
      "12", "64", "102", "1020", "10200"); %number of bits in the FIFO
    if choice1 == 1
      n = 12;
    elseif choice1 == 2
      n = 64;
    elseif choice1 == 3
      n = 102;
    elseif choice1 == 4
      n = 1020;
    elseif choice1 == 5
      n = 10200;
    endif

  choice2 = menu ("Which QAM do you want to use?", "4", "16", "64");  %
     number of symbols in the constellation
    if choice2 == 1
      M = 4;
    elseif choice2 == 2
      M = 16;
    elseif choice2 == 3
      M = 64;
    endif

  bitsBySymbol = log2(M); %number of bits in one symbol
```

```
%------------------- QAM Map Table ----------------------

numberOfSymbols = n/bitsBySymbol; %number of symbols in the FIFO

% Define mapping table applying Gray mapping

if M == 4
  mappingTable(1) =  1 + 1*j;
  mappingTable(2) = -1 + 1*j;
  mappingTable(3) =  1 - 1*j;
  mappingTable(4) = -1 - 1*j;

elseif M == 16
  mappingTable(1:4) = -3;
  mappingTable(5:8) = -1;
  mappingTable(9:12) = +3;
  mappingTable(13:16) = +1;
  for i = 0:15
    if mod(i,4) == 0
      mappingTable(i+1) = mappingTable(i+1) -3*j;
    elseif mod(i+3,4) == 0
      mappingTable(i+1) = mappingTable(i+1) -1*j;
    elseif mod(i+1,4) == 0
      mappingTable(i+1) = mappingTable(i+1) +1*j;
    elseif mod(i+2,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +3*j;
    endif
  endfor


elseif M == 64
  mappingTable(1:8)   = + 7*j;
  mappingTable(9:16)  = + 5*j;
  mappingTable(17:24) = + 1*j;
  mappingTable(25:32) = + 3*j;
  mappingTable(33:40) = - 7*j;
  mappingTable(41:48) = - 5*j;
  mappingTable(49:56) = - 1*j;
  mappingTable(57:64) = - 3*j;

  for i = 0:63
    if mod(i+2,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +1;
    elseif mod(i+1,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +3;
    elseif mod(i+3,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +5;
    elseif mod(i+4,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +7;
```

```
    elseif mod(i+6,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -1;
    elseif mod(i+5,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -3;
    elseif mod(i+7,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -5;
    elseif mod(i,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -7;
    endif
  endfor
endif

% ---- ---- Simulation Paramters ---- ---- %
last_phase = 0;
sync_flag = -2;
k=1;
end_k = numberOfSymbols +1;
Ts = 1e-3;
dt = 1e-5;
end_t = 1e-3;
t = 0:dt:Ts-dt;
lt = length(t);
window_size = 5;
PLOT_TX = 1 ;
PLOT_RX = 1;

% ---- ---- Transceiver Paramters ---- ---- %
fc = 1e3;
fs = 1/dt;
dataIn = randi(2,n,1) - 1; % Generate vector of binary data
dataOut = zeros(n,1);
  % ---- ---- Bad Paramters ---- ---- %
  delay = 0.005e-3;
  phase_tx = 0;
  phase_rx = 2*pi*0.13;
  fc_tx = fc;
  fc_rx = fc*1.01;
  SNR = 0;
  % ---- ---- Good Paramters ---- ---- %
  delay = 0.0;
  phase_rx = 0;
  phase_tx = 0.451;
  fc_tx = fc;
#   fc_rx = fc;
  SNR = 100;

#   vco_p(1) = 0;
#   vco_f = fc;
#
```

```
#    maf_l = 500; %Moving average filter - Low Pass filter but with a
    simpler algorithm without all the multiplications
#    fifo = zeros(1,maf_l);
#
#    maf_f_l = 100;
#    fifo_f = zeros(1,maf_f_l);
#
#    maf(1) = 0;
#    maf_f(1) = 0;

  % ----- ----- Pulse shaping ----- ----- %
   pulse_shaping = sin(2*pi*(0:dt:Ts-dt)/(Ts-dt));
   pulse_shaping = ones(1,lt);

% ----- ----- END Transceiver Paramters ----- ----- %



if PLOT_TX
  figure(1);
  hold on;
endif;
if PLOT_RX
  figure(2);
  hold on;
endif;


# xi_aux_t = zeros(1,floor(Ts/dt));
# xq_aux_t = zeros(1,floor(Ts/dt));
xi_aux_t = zeros(1,lt);
xq_aux_t = zeros(1,lt);

x2_t = zeros(1,2*lt);

% ----- ----- Main Loop ----- ----- %
while(k < end_k)


  % ----- ----- Transmitter ----- ----- %
#    xi_k = -xi_k;
#    xq_k = 1;

   if sync_flag <= 0 %Dummy symbol to lock the PLL
     xi_k = 1;
     xq_k = 1;
   else
     symbolBits = dataIn((k-1)*bitsBySymbol+1:(k-1)*bitsBySymbol+
         bitsBySymbol);
```

```
    if M == 4
      symbolIndex = 2^1 * symbolBits(1) + 2^0 * symbolBits(2);
    elseif M == 16
      symbolIndex = 2^3 * symbolBits(1) + 2^2 * symbolBits(2) + 2^1 *
          symbolBits(3) + 2^0 * symbolBits(4);
    elseif M == 64
      symbolIndex = 2^5 * symbolBits(1) + 2^4 * symbolBits(2) + 2^3 *
          symbolBits(3) + 2^2 * symbolBits(4) +  2^1 * symbolBits(5) +
          2^0 * symbolBits(6);
    endif
    symbolIndex;
     % Mapping
    symbol = mappingTable(symbolIndex + 1);
    xi_k = real(symbol);
    xq_k = imag(symbol);
  endif

  xsi_t = pulse_shaping .* xi_k;
  xsq_t = pulse_shaping .* xq_k;

  xi_t = xsi_t .* cos(2*pi*fc_tx*t+phase_tx);
  xq_t = xsq_t .* sin(2*pi*fc_tx*t+phase_tx);

  x_t = xi_t + xq_t;
  x2_t = [x2_t(1+lt:2*lt) x_t];
  % ----- ----- END Transmitter ----- ----- %


  % ----- ----- Receiver ----- ----- %;

  %CHANNEL
  y_t = awgn(x_t, SNR);
  delay_idx = mod(floor(delay/dt),lt)+1;
  idx = [delay_idx:(delay_idx+lt-1)];
  y_t = awgn(x2_t(idx), SNR);

  %CARRIER FREQUENCY AND PHASE RECOVERY
#   if k == 0  %TODO : send the carrier only at the beginning to
    recover the frequency and the phase

    k
    aux_vco_t = -sin(2*pi*fc_rx*t + phase_rx - pi/4);
    aux_phase_error_t = y_t .* aux_vco_t;
    phase_error = mean(aux_phase_error_t)

    fc_error = (phase_rx - last_phase);
    last_phase = phase_rx;
```

```
  if sync_flag  <= 0  %TODO : send the carrier only at the beginning to
      recover the frequency and the phase
    phase_rx = phase_rx + phase_error
#     fc_rx = fc_rx + fc_error * 10

    if phase_error < 0.001
      sync_flag ++;
    endif;
  endif;


    figure(10); hold on;
    plot(t,aux_phase_error_t);
    plot([t(1) t(1)+Ts],phase_error*[1 1],'k');
    plot([t(1) t(1)+Ts],phase_rx*[1 1],'g');
    plot([t(1) t(1)+Ts],fc_error*[1 1],'r');
#     plot(t,fc_rx*ones(1,length(t)),'c');




#
#
#
#     delta_t = 1e-5;
#     max_t = 5e-1;
#     t_sync = 0:delta_t:max_t;
#     y_t = cos(2*pi*fc_rx*t_sync + phase_rx);
#
#     for i = 2:length(t_sync)
#
#       vco_t(i) = -sin(2*pi*vco_f(i-1)*t_sync(i) + vco_p(i-1));
#       aux_t(i) = y_t(i) * vco_t(i);
#       maf(i) = maf(i-1) + aux_t(i);
#       maf(i) -= fifo(maf_l);
#
#       fifo(2:maf_l) = fifo(1:maf_l-1);
#       fifo(1) = aux_t(i);
#
#       vco_p(i) = vco_p(i-1) + maf(i-1)/maf_l/100;
#
#       maf_f(i) = maf_f(i-1) + (vco_p(i) - vco_p(i-1))/delta_t; %
    Derivate the phase to have the frequency
#       maf_f(i) -= fifo_f(maf_f_l);
#       fifo_f(2:maf_f_l) = fifo_f(1:maf_f_l-1);
#       fifo_f(1) = (vco_p(i) - vco_p(i-1))/delta_t;
#
#       if (mod(i,10000) == 0)
```

```
#         vco_f(i) = vco_f(i-1) + maf_f(i)/pi/2/maf_f_l;
#       else
#         vco_f(i) = vco_f(i-1);
#       endif;
#
#   %    carrier_inph(step-1) = cos(2*pi*vco_f(step-1)*step*dt+vco_p(
    step-1));
#   %    carrier_quad(step-1) = sin(2*pi*vco_f(step-1)*step*dt+vco_p(
    step-1));
#     endfor
#     k++;
#
#     p_step = 100;
#
#     figure(4); hold on;
#     plot(t_sync(1:p_step:end),aux_t(1:p_step:end));
#     plot(t_sync(1:p_step:end),maf(1:p_step:end)/maf_l*2,'r;phase
    error;');
#     plot(t_sync(1:p_step:end),vco_p(1:p_step:end),'g;vco phase;');
#      %plot(t(1:p_step:end),vco_f(1:p_step:end)-rx_f(1:p_step:end),'c;
    vco f - rx f;')
#     plot(t_sync(1:p_step:end),maf_f(1:p_step:end)/1000/pi,'k;freq
    error;')
#
#     figure(5); hold on;
#     plot(t_sync(1:p_step:end),fc_rx*ones(1,length(t_sync(1:p_step:end
    ))),'b;rx f;');
#     plot(t_sync(1:p_step:end),vco_f(1:p_step:end),'c;vco f;');

#   else

#    carrier_inph = cos(2*pi*vco_f(end)*t+vco_p(end));
#    carrier_quad = sin(2*pi*vco_f(end)*t+vco_p(end));

    %RECOVERY OF YI_T AND YQ_T
    carrier_inph = cos(2*pi*fc_rx*t + phase_rx);
    carrier_quad = sin(2*pi*fc_rx*t + phase_rx);

    yi_t = y_t .* carrier_inph;
    yq_t = y_t .* carrier_quad;
    % TODO The LBP filter is missing
  %   lbp_length = 4;
  %   lbp_filter = [ones(1,lbp_length) zeros(1,lt-lbp_length)];
  %   lbp_filter /= sum(lbp_filter);
  %   yi_t = filter(lbp_filter, [1 zeros(1,19)], [xi_aux_t yi_non_lbp_t
      ])(lt+1:2*lt);
  %   yq_t = filter(lbp_filter, [1 zeros(1,19)], [xi_aux_t yq_non_lbp_t
      ])(lt+1:2*lt);
  %   xi_aux_t = yi_non_lbp_t;
```

```
%    xq_aux_t = yq_non_lbp_t;

% FILTERING BY FFT
   aux = yi_t + j*yq_t;
   aux = fft(aux) .* [1 1 zeros(1,lt-2)];
   aux = ifft(aux);
   yi_filter_t = 2*real(aux);
   yq_filter_t = 2*imag(aux);
   yi_k = yi_filter_t(1);
   yq_k = yq_filter_t(1);

% TODO : FIR filter but for the mean time, FFT is good to implement
    the carrier recovery
%  %figure(3)
%  b = fir1(100,0.001);
%  %freqz(b,10e6);
%  yi_filter_t = 2*filter(b,1,yi_t);
%  yq_filter_t = 2*filter(b,1,yq_t);
%  yi_k = yi_filter_t(50)
%  yq_k = yq_filter_t(50);

%QAM DEMAPPER
   receivedSymbols = yi_k + j*yq_k;
   [mindiff minIndex] = min(receivedSymbols - mappingTable);
   symbolIndexAfter = minIndex - 1;

if sync_flag == 1
   for i = 1:bitsBySymbol
     bits(i) = mod(symbolIndexAfter,2);
     symbolIndexAfter = floor(symbolIndexAfter/2);
   endfor
   bits = fliplr(bits);

   for i = 1:bitsBySymbol
     dataOut((k-1)*bitsBySymbol +i) = bits(i);
   endfor
endif;

  % ---- ---- END Receiver ---- ---- %



  % ---- ---- Plot Transmitter Signals ---- ---- %
   if (PLOT_TX == 1)
     figure(1);
     plot([t(1) t(1)+Ts],[0,0],'k-');
     stem(t(1), xi_k,'b','linewidth',3);
     stem(t(1), xq_k,'r','linewidth',2);
```

```
    plot([t(1) t(1)+Ts],[-3,-3],'k-');
    plot(t,xsi_t-3,'b:','linewidth',2);
    plot(t,xsq_t-3,'r-','linewidth',1);

    plot([t(1) t(1)+Ts],[-6,-6],'k-');
    plot(t,xi_t-6,'b:','linewidth',1);
    plot(t,xq_t-6,'r-','linewidth',1);

    plot([t(1) t(1)+Ts],[-9,-9],'k-');
    plot(t,x_t-9,'g-','linewidth',1);
    figure(1);
    axis([t(1)-window_size*Ts t(1)+Ts -11 2]); %TODO: implement a
        plotting buffer with a windown from 5 to 10 Ts
    drawnow ("expose");
    title('xik(blue)/xqk(red), xsit/q, xit/q, xt');
  endif;
% ---- ---- END Plot Transmitter Signals ---- ---- %


% ---- ---- Plot Receiver Signals ---- ---- %
  if (PLOT_RX == 1)
    figure(2);
    plot(t,y_t-9,'g-','linewidth',1)
    plot([t(1) t(1)+Ts],[-9,-9],'k-');
    axis([t(1)-window_size*Ts t(1)+Ts -11 2]); %TODO: implement a
        plotting buffer with a windown from 5 to 10 Ts
    drawnow ("expose");

    plot([t(1) t(1)+Ts],[-6,-6],'k-');
    plot(t,yi_t-6,'b:','linewidth',1)
    plot(t,yq_t-6,'r-','linewidth',1)

    plot([t(1) t(1)+Ts],[-3,-3],'k-');
    plot(t,yi_filter_t-3,'b:','linewidth',2)
    plot(t,yq_filter_t-3,'r-','linewidth',1)

    plot([t(1) t(1)+Ts],[0,0],'k-');
    stem(t(1), yi_k,'b','linewidth',3);
    stem(t(1), yq_k,'r','linewidth',2);

    title('yik(blue)/yqk(red), yifiltert/q, yit/q, yt');
  endif;
% ---- ---- END Plot Receiver Signals ---- ---- %

% ---- ---- Update time and plots ---- ---- %
  t = t + Ts;
  if sync_flag == 1
    k++;
```

```
    endif;
    fflush(stdout);
    % ----- ----- END Update time and plots ----- ----- %
#   endif
#   vco_p(end)
#   k
endwhile;
% ----- ----- END Main Loop ----- ----- %
dataIn;
dataOut;

%---------------- BER ----------------

total_error = 0;

for i = 1:n-2
    if dataIn(i) != dataOut(i+2)
      total_error ++;
    endif
endfor;

% Calculation of BER to return the result
BER = total_error/n;

% Showing final results
disp(['Total wrong bits = ' num2str(total_error)]);
disp(['BER = ' num2str(BER)]);
%
%figure(3);
%plot(vco_f,'c;vco f;');
```

9.3 vco

```
close all;
clear all;
clf;
graphics_toolkit ("gnuplot");

 % ----- ----- Simulation Paramters ----- ----- %
Ts = 1e-3;
dt = 1e-6;
fs = 1/dt;
end_t = 1e-3;
t = 0:dt:Ts-dt;
lt = length(t);
fc = 1e3;
delay = 0.0;
phase_rx = 0;
phase_tx = 0;
```

```
fc_tx = fc;
fc_rx = fc;
SNR = 100;

% first order pll
alpha = 0.005;

% VCO voltage which controlls the frequency. at v=0 its exactly at f.
v = 0;
% cosine output
c = 1;
% delayed cosine by one timestep
c_delay = 0;
% sine output
s = 0;
% delayed sine by one timestep
s_delay = 0;

% data from the VCO for debugging purposes
sine = zeros(Ts,1);
cosine =zeros(Ts,1);
vco = zeros(Ts,1);

for step = 1:dt/dt:Ts/dt

  % this is part of the PLL
  % "voltage" controlled oscillator
  f0 = fc_rx/fs + v*alpha;
  c_delay = c;
  s_delay = s;
  c = c_delay * cos(2*pi*f0) - s_delay * sin(2*pi*f0);
  s = s_delay * cos(2*pi*f0) + c_delay * sin(2*pi*f0);
  % lets save everything in handy vectors for plotting
  sine(step) = s;
  cosine(step) = c;
  vco(step) = v
  % end VCO

  % save our carriers
  % !!! 90 degree phase shift so the sine becomes the inphase
  %     signal and the cosine the quadrature signal
  carrier_inph(step) = c;
  carrier_quad(step) = s;

endfor

figure(1);

subplot(3,1,1);
```

```
plot(cosine);
ylabel('Amplitude');
title('cosine');
grid on;

subplot(3,1,2);
plot(sine);
ylabel('Amplitude');
title('sine');
grid on;

subplot(3,1,3);
plot(vco);
ylabel('Amplitude');
title('vco');
grid on;
```

9.4 draft5

```
close all;
clear all;
graphics_toolkit ("gnuplot");

# figure; hold on;
# t=0:0.01:1;
# ph=0;
# for i = 1:30
#   stem(i,mean(sin(2*pi*t).*sin(2*pi*t+ph)));
#   stem(i,mean(cos(2*pi*t).*sin(2*pi*t+ph)),'r');
#   ph+=0.1;
# endfor;
# return;

  %---------- Choice of parameters -------- %
  choice1 = menu ("Which number of bits in the bitstream do you want?",
      "12", "64", "102", "1020", "10200"); %number of bits in the FIFO
    if choice1 == 1
      n = 12;
    elseif choice1 == 2
      n = 64;
    elseif choice1 == 3
      n = 102;
    elseif choice1 == 4
      n = 1020;
    elseif choice1 == 5
      n = 10200;
    endif

  choice2 = menu ("Which QAM do you want to use?", "4", "16", "64");  %
```

```
      number of symbols in the constellation
    if choice2 == 1
      M = 4;
    elseif choice2 == 2
      M = 16;
    elseif choice2 == 3
      M = 64;
    endif

  bitsBySymbol = log2(M); %number of bits in one symbol

  %------------------- QAM Map Table ----------------------

numberOfSymbols = n/bitsBySymbol; %number of symbols in the FIFO

% Define mapping table applying Gray mapping

if M == 4
  mappingTable(1) =  1 + 1*j;
  mappingTable(2) = -1 + 1*j;
  mappingTable(3) =  1 - 1*j;
  mappingTable(4) = -1 - 1*j;

elseif M == 16
  mappingTable(1:4) = -3;
  mappingTable(5:8) = -1;
  mappingTable(9:12) = +3;
  mappingTable(13:16) = +1;
  for i = 0:15
    if mod(i,4) == 0
      mappingTable(i+1) = mappingTable(i+1) -3*j;
    elseif mod(i+3,4) == 0
      mappingTable(i+1) = mappingTable(i+1) -1*j;
    elseif mod(i+1,4) == 0
      mappingTable(i+1) = mappingTable(i+1) +1*j;
    elseif mod(i+2,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +3*j;
    endif
  endfor


elseif M == 64
  mappingTable(1:8)   = + 7*j;
  mappingTable(9:16)  = + 5*j;
  mappingTable(17:24) = + 1*j;
  mappingTable(25:32) = + 3*j;
  mappingTable(33:40) = - 7*j;
  mappingTable(41:48) = - 5*j;
  mappingTable(49:56) = - 1*j;
```

```
  mappingTable(57:64) = - 3*j;

  for i = 0:63
    if mod(i+2,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +1;
    elseif mod(i+1,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +3;
    elseif mod(i+3,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +5;
    elseif mod(i+4,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +7;
    elseif mod(i+6,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -1;
    elseif mod(i+5,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -3;
    elseif mod(i+7,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -5;
    elseif mod(i,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -7;
    endif
  endfor
endif



% ----- ----- Simulation Paramters ----- ----- %
k=1;
end_k = numberOfSymbols +1;

dt = 1e-8;
end_t = 1e-3;

window_size = 5;

PLOT_TX = 0;
PLOT_RX = 0;

dataIn = randi(2,n,1) - 1; % Generate vector of binary data
dataOut = zeros(n,1);

if PLOT_TX
  figure(1);
  hold on;
endif;
if PLOT_RX
  figure(2);
  hold on;
endif;
% ----- ----- Simulation Paramters ----- ----- %
```

```
% ----- ----- Transceiver Paramters ----- ----- %
fc                  = 1e6;
# fc_tx             = fc*1.0001;
# fc_tx             = fc*1.00002; % typical error 20ppm
fc_tx               = fc;
ph_tx               = 0.4135;
fc_rx               = fc;
ph_rx               = 0.0;
# Tsymb             = 1e-4;      % Symbol period
Tsymb               = 10/fc;     % Symbol period = 100/fc = 100 fc periods
samples_per_symb    = 16;
Ts                  = Tsymb/samples_per_symb; % Sampling period: 16
    samples/symbol
alpha_param         = 1;
beta_param          = 0.25;
delay               = Tsymb/3;
delay               = 0.0;
SNR                 = 100;
  % ----- ----- Pulse shaping ----- ----- %
  t = 0:dt:Tsymb-dt;
  tl= length(t);
#   pulse_shaping = sin(pi*(0:dt:Tsymb-dt)/(Tsymb-dt));
  pulse_shaping = 0.5+sin(pi*(0:dt:Tsymb-dt)/(Tsymb-dt)) + 0.5;
#   pulse_shaping = ones(1,tl);
# SNR = 100;
% ----- ----- Transceiver Paramters ----- ----- %
```

```
# % ----- ----- Transceiver Paramters ----- ----- %
#   % ----- ----- Bad Paramters ----- ----- %
#   delay = 0.005e-3;
#   ph_tx = 0;
#   ph_rx = 2*pi*0.13;
#   ph_tx = 0.451;
#   fc_tx = fc;
#   fc_rx = fc*1.01;
#   SNR = 0;
#   % ----- ----- Good Paramters ----- ----- %
# #   delay = 0.0;
# #   ph_tx = 0;
#   ph_rx = 0;
#   fc_tx = fc;
# #   fc_rx = fc;
```

```
#    SNR = 100;
#
#
#
#    maf_f_l = 100;
#    fifo_f = zeros(1,maf_f_l);
#    maf_f(1) = 0;
#
#    % ----- ----- Pulse shaping ----- ----- %
#    pulse_shaping = sin(2*pi*(0:dt:Tsymb-dt)/(Tsymb-dt));
#    pulse_shaping = ones(1,tl);
#
% ----- ----- END Transceiver Paramters ----- ----- %




xi_aux_t = zeros(1,tl);
xq_aux_t = zeros(1,tl);

x2_t = zeros(1,2*tl);
y3_t = zeros(1,3*tl);
yi2_t = zeros(1,2*tl);
yq2_t = zeros(1,2*tl);

aux_error = 0;
last_phase = 0;
sync_flag = -20;
freeze = 1;

sync_sample = 0;

% ----- ----- Main Loop ----- ----- %
while(k < end_k)

  % ----- ----- Transmitter ----- ----- %

  if sync_flag <= 0 %Dummy symbol to lock the PLL
    xi_k = 1;
    xq_k = 1;
  else
    symbolBits = dataIn((k-1)*bitsBySymbol+1:(k-1)*bitsBySymbol+
        bitsBySymbol);

    if M == 4
      symbolIndex = 2^1 * symbolBits(1) + 2^0 * symbolBits(2);
    elseif M == 16
      symbolIndex = 2^3 * symbolBits(1) + 2^2 * symbolBits(2) + 2^1 *
```

```
            symbolBits(3) + 2^0 * symbolBits(4);
        elseif M == 64
          symbolIndex = 2^5 * symbolBits(1) + 2^4 * symbolBits(2) + 2^3 *
            symbolBits(3) + 2^2 * symbolBits(4) +  2^1 * symbolBits(5) +
            2^0 * symbolBits(6);
        endif
        symbolIndex;
         % Mapping
        symbol = mappingTable(symbolIndex + 1);
        xi_k = real(symbol);
        xq_k = imag(symbol);
      endif

      xsi_t = pulse_shaping .* xi_k;
      xsq_t = pulse_shaping .* xq_k;

      xi_t = xsi_t .* cos(2*pi*fc_tx*t+ph_tx);
      xq_t = xsq_t .* sin(2*pi*fc_tx*t+ph_tx);

      x_t = xi_t + xq_t;
      x2_t = [x2_t(1+tl:2*tl) x_t];
      % ----- ----- END Transmitter ----- ----- %


      % ----- ----- Receiver ----- ----- %;

      %CHANNEL
      delay_idx = mod(tl-floor(delay/dt),tl)+1;
      idx = [delay_idx:(delay_idx+tl-1)];
      y3_t = [y3_t(1+tl:3*tl) awgn(x2_t(idx), SNR)];
      y2_t = y3_t(1:2*tl);
#     y_t = awgn(x_t, SNR);
#     y_t = awgn(x2_t(idx), SNR);
#     y_t = awgn(x2_t, SNR);

      %CARRIER FREQUENCY AND PHASE RECOVERY

#     carrier_inph = cos(2*pi*fc_rx*t + ph_rx);
#     carrier_quad = sin(2*pi*fc_rx*t + ph_rx);

#     yi_t = y_t .* carrier_inph;
#     yq_t = y_t .* carrier_quad;
#     % FILTERING BY FFT
# #     aux = yi_t + j*yq_t;
# #     aux = fft(aux) .* [1 1 zeros(1,tl-2)];
# #     aux = ifft(aux);
# #     y_t = aux;
#     yi2_t = [yi2_t(1+tl:2*tl) yi_t];
#     yq2_t = [yq2_t(1+tl:2*tl) yq_t];
```

```
  t2 = t(1):dt:(t(end)+Tsymb);
  carrier_inph = cos(2*pi*fc_rx*t2 + ph_rx);
  carrier_quad = sin(2*pi*fc_rx*t2 + ph_rx);
  yi2_t = y2_t .* carrier_inph;
  yq2_t = y2_t .* carrier_quad;


#   sample_number = mod(floor(1/fc/4/dt) + floor(ph_rx/2/pi*Tsymb/dt),
    Tsymb/dt);
#   sample_number = mod(floor(1/fc/4/dt), Tsymb/dt);
  indx = Ts*sync_sample:Ts:(Tsymb-Ts)+Ts*sync_sample;
  indx /= dt;
  indx = floor(indx);
  indx += 1;
  yi_s = yi2_t(indx);
  yq_s = yq2_t(indx);

  indx = floor(mod(indx-1, Tsymb/dt)+1) ;

  yi_k = mean(yi_s./pulse_shaping(indx));
  yq_k = mean(yq_s./pulse_shaping(indx));

  if sync_flag <= 0
    sync_flag ++;
    freeze = 1;
    symbolIndexAfter = 0;
  else
    receivedSymbols = yi_k + j*yq_k;
    [mindiff minIndex] = min(receivedSymbols - mappingTable);
    symbolIndexAfter = minIndex - 1;

    aux_symbolIndexAfter = symbolIndexAfter;
    for i = 1:bitsBySymbol
      bits(i) = mod(aux_symbolIndexAfter,2);
      aux_symbolIndexAfter = floor(aux_symbolIndexAfter/2);
    end
    bits = fliplr(bits);
    for i = 1:bitsBySymbol
      dataOut((k-1)*bitsBySymbol +i) = bits(i);
    end;
    sync_sample = mod(sync_sample, samples_per_symb);
  endif;

  mappingTable(symbolIndexAfter+1)
  yi_k + j * yq_k
  arg(mappingTable(symbolIndexAfter+1))
  arg(yi_k + j * yq_k)
  phase_error = arg(mappingTable(symbolIndexAfter+1)) - arg(yi_k + j *
      yq_k)
```

```
    aux_error += phase_error;
    phase_correction = alpha_param * phase_error + beta_param * aux_error
        ;

    ph_rx = mod(ph_rx + phase_correction, 2*pi)


      figure(10); hold on;
#       plot(t,aux_phase_error_t);
      plot([t(1) t(1)+Tsymb],phase_error*[1 1],'k');
      plot([t(1) t(1)+Tsymb],ph_rx*[1 1],'g');
#       plot([t(1) t(1)+Tsymb],fc_error*[1 1],'r');
%       plot(t,fc_rx*ones(1,length(t)),'c');
      plot([t(1) t(1)+Tsymb],aux_error*[1 1],'m','linewidth',2);
      plot([t(1) t(1)+Tsymb],phase_correction*[1 1],'c','linewidth',2);


      % ----- ----- END Receiver ----- ----- %



      % ----- ----- Plot Transmitter Signals ----- ----- %
      if (PLOT_TX == 1)
        figure(1);
#         subplot(4,1,1); hold on;
#         plot([t(1) t(1)+Tsymb],[0,0],'k-');
        stem(t(1), xi_k,'b','linewidth',3);
        stem(t(1), xq_k,'r','linewidth',2);
        axis([t(1)-window_size*Tsymb t(1)+Tsymb -1.5 1.5]); %TODO:
            implement a plotting buffer with a windown from 5 to 10 Tsymb

#         subplot(4,1,2);  hold on;
        plot(t,xsi_t-3,'b:','linewidth',2);
        plot(t,xsq_t-3,'r-','linewidth',1);
        axis([t(1)-window_size*Tsymb t(1)+Tsymb -4.5 1.5]); %TODO:
            implement a plotting buffer with a windown from 5 to 10 Tsymb

#         subplot(4,1,3);  hold on;
#         plot(t,xi_t-6,'b:','linewidth',1);
#         plot(t,xq_t-6,'r-','linewidth',1);
#         axis([t(1)-window_size*Tsymb t(1)+Tsymb -7.5 1.5]); %TODO:
    implement a plotting buffer with a windown from 5 to 10 Tsymb

#         subplot(4,1,4); hold on;
#         plot(t,x_t,'g-','linewidth',1);
#         axis([t(1)-window_size*Tsymb t(1)+Tsymb -1.5 1.5]); %TODO:
    implement a plotting buffer with a windown from 5 to 10 Tsymb
```

```
        drawnow ("expose");
      title('xik(blue)/xqk(red), xsit/q, xit/q, xt');
    endif;
    % ---- ---- END Plot Transmitter Signals ---- ---- %


    % ---- ---- Plot Receiver Signals ---- ---- %
    if (PLOT_RX == 1)
      figure(2); hold on;
#       subplot(4,1,1); hold on;
#       plot(t,y_t,'g-','linewidth',1)
## #       plot([t(1) t(1)+Tsymb],[0,0],'k-');

#       subplot(4,1,2); hold on;
#       plot(t,yi_t-6,'b:','linewidth',1)
#       plot(t,yq_t-6,'r-','linewidth',1)

#       subplot(4,1,3); hold on;
#       stem(t(indx),yi_s,'m:','linewidth',2)
#       stem(t(indx),yq_s,'g-','linewidth',1)
      plot(t(indx),yi_s,'m:','linewidth',2)
      plot(t(indx),yq_s,'g-','linewidth',1)
      axis([t(1)-window_size*Tsymb t(1)+Tsymb -1.5 1.5]); %TODO:
          implement a plotting buffer with a windown from 5 to 10 Tsymb

#       figure(3); hold on;
#       subplot(4,1,4); hold on;
      stem(t(1), yi_k,'bx','linewidth',3,'linewidth',4,'markersize',15)
          ;
      stem(t(1), yq_k,'rx','linewidth',2,'linewidth',4,'markersize',15)
          ;
#       stem(t(indx), yi_s./pulse_shaping(indx),'bx','linewidth',3,'
    linewidth',4,'markersize',15);
#       stem(t(indx), yq_s./pulse_shaping(indx),'rx','linewidth',2,'
    linewidth',4,'markersize',15);
      axis([t(1)-window_size*Tsymb t(1)+Tsymb -1.5 1.5]); %TODO:
          implement a plotting buffer with a windown from 5 to 10 Tsymb

#       axis([t(1)-window_size*Tsymb t(1)+Tsymb -11 2]); %TODO:
    implement a plotting buffer with a windown from 5 to 10 Tsymb
#       title('yik(blue)/yqk(red), yifiltert/q, yit/q, yt');
      drawnow ("expose");
    endif;
    % ---- ---- END Plot Receiver Signals ---- ---- %

    % ---- ---- Update time and plots ---- ---- %
    t = t + Tsymb;
    if freeze == 0
      k++;
```

```
    elseif sync_flag == 1
      freeze = 0;
    end
    fflush(stdout);
    % ----- ----- END Update time and plots ----- ----- %
endwhile;
% ----- ----- END Main Loop ----- ----- %

dataInTest = dataIn(1:end-6);
dataOutTest = dataOut(7:end);

%---------------- BER ----------------

total_error = 0;

for i = 1:n-6
    if dataIn(i) != dataOut(i+6)
      total_error ++;
    end
end

% Calculation of BER to return the result
BER = total_error/n;

% Showing final results
disp(['Total wrong bits = ' num2str(total_error)]);
disp(['BER = ' num2str(BER)]);
%
%figure(3);
%plot(vco_f,'c;vco f;');
```

9.5 systemWithPLL

```
close all;
clear all;
graphics_toolkit ("gnuplot");

  %---------- Choice of parameters -------- %
  choice1 = menu ("Which number of bits in the bitstream do you want?",
      "12", "64", "102", "1020", "10200"); %number of bits in the FIFO
    if choice1 == 1
      n = 12;
    elseif choice1 == 2
      n = 64;
    elseif choice1 == 3
      n = 102;
    elseif choice1 == 4
      n = 1020;
    elseif choice1 == 5
```

```
      n = 10200;
    endif

  choice2 = menu ("Which QAM do you want to use?", "4", "16", "64"); %
     number of symbols in the constellation
    if choice2 == 1
      M = 4;
    elseif choice2 == 2
      M = 16;
    elseif choice2 == 3
      M = 64;
    endif

  bitsBySymbol = log2(M); %number of bits in one symbol

  %---------------------- QAM Map Table ----------------------

numberOfSymbols = n/bitsBySymbol; %number of symbols in the FIFO

% Define mapping table applying Gray mapping

if M == 4
  mappingTable(1) =  1 + 1*j;
  mappingTable(2) = -1 + 1*j;
  mappingTable(3) =  1 - 1*j;
  mappingTable(4) = -1 - 1*j;

elseif M == 16
  mappingTable(1:4) = -3;
  mappingTable(5:8) = -1;
  mappingTable(9:12) = +3;
  mappingTable(13:16) = +1;
  for i = 0:15
    if mod(i,4) == 0
      mappingTable(i+1) = mappingTable(i+1) -3*j;
    elseif mod(i+3,4) == 0
      mappingTable(i+1) = mappingTable(i+1) -1*j;
    elseif mod(i+1,4) == 0
      mappingTable(i+1) = mappingTable(i+1) +1*j;
    elseif mod(i+2,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +3*j;
    endif
  endfor


elseif M == 64
  mappingTable(1:8)   = + 7*j;
  mappingTable(9:16)  = + 5*j;
  mappingTable(17:24) = + 1*j;
```

```
  mappingTable(25:32) = + 3*j;
  mappingTable(33:40) = - 7*j;
  mappingTable(41:48) = - 5*j;
  mappingTable(49:56) = - 1*j;
  mappingTable(57:64) = - 3*j;

  for i = 0:63
    if mod(i+2,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +1;
    elseif mod(i+1,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +3;
    elseif mod(i+3,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +5;
    elseif mod(i+4,8) == 0
      mappingTable(i+1) = mappingTable(i+1) +7;
    elseif mod(i+6,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -1;
    elseif mod(i+5,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -3;
    elseif mod(i+7,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -5;
    elseif mod(i,8) == 0
      mappingTable(i+1) = mappingTable(i+1) -7;
    endif
  endfor
endif

% ---- ---- Simulation Paramters ---- ---- %
last_phase = 0;
sync_flag = -50;
freeze = 1;
state = 1;
i = 2;
k=1;
end_k = numberOfSymbols +1;
Ts = 1e-3;
dt = 1e-5;
end_t = 1e-3;
t = 0:dt:Ts-dt;
lt = length(t);
window_size = 5;
PLOT_TX = 0;
PLOT_RX = 0;

% ---- ---- Transceiver Paramters ---- ---- %
fc = 1e3;
fs = 1/dt;
dataIn = randi(2,n,1) - 1; % Generate vector of binary data
dataOut = zeros(n,1);
```

```
  % ----- ----- Bad Paramters ----- ----- %
  delay = 0.005e-2;
  phase_tx = 0;
  phase_rx = 2*pi*0.13;
  phase_tx = 0.451;
  fc_tx = fc;
  fc_rx = fc*1.001;
  SNR = 0;
  % ----- ----- Good Paramters ----- ----- %
  % delay = 0.0;
  %phase_rx = 0;
  %phase_tx = 0;
  fc_tx = fc;
  fc_rx = fc;
  SNR = 100;

  var_t = 0;
  ks = lt / 4;
  span = 5;
  shift = floor(lt / 8) + 1;
  timing_shift = 0;

  maf_f_l = 100;
  fifo_f = zeros(1,maf_f_l);
  maf_f(1) = 0;

  % ----- ----- Pulse shaping ----- ----- %
  pulse_shaping = sin(2*pi*(0:dt:Ts-dt)/(Ts-dt));
  pulse_shaping = ones(1,lt);

% ----- ----- END Transceiver Paramters ----- ----- %


if PLOT_TX
 figure(1);
 hold on;
endif;
if PLOT_RX
 figure(2);
 hold on;
endif;


% xi_aux_t = zeros(1,floor(Ts/dt));
% xq_aux_t = zeros(1,floor(Ts/dt));
xi_aux_t = zeros(1,lt);
xq_aux_t = zeros(1,lt);
```

```
x2_t = zeros(1,2*lt);

% ----- ----- Main Loop ----- ----- %
while(k < end_k)

  % ----- ----- Transmitter ----- ----- %

  if state == 1 %Dummy symbol to lock the PLL
    xi_k = 1;
    xq_k = 1;

  elseif state == 2 %Dummy symbol to recover the symbol timing
    xi_k = 1;
    xq_k = 1;

  else
    symbolBits = dataIn((k-1)*bitsBySymbol+1:(k-1)*bitsBySymbol+
        bitsBySymbol);

    if M == 4
      symbolIndex = 2^1 * symbolBits(1) + 2^0 * symbolBits(2);
    elseif M == 16
      symbolIndex = 2^3 * symbolBits(1) + 2^2 * symbolBits(2) + 2^1 *
          symbolBits(3) + 2^0 * symbolBits(4);
    elseif M == 64
      symbolIndex = 2^5 * symbolBits(1) + 2^4 * symbolBits(2) + 2^3 *
          symbolBits(3) + 2^2 * symbolBits(4) +  2^1 * symbolBits(5) +
          2^0 * symbolBits(6);
    end
    symbolIndex;
     % Mapping
    symbol = mappingTable(symbolIndex + 1);
    xi_k = real(symbol);
    xq_k = imag(symbol);
  end

  xsi_t = pulse_shaping .* xi_k;
  xsq_t = pulse_shaping .* xq_k;

  xi_t = xsi_t .* cos(2*pi*fc_tx*t+phase_tx);
  xq_t = xsq_t .* sin(2*pi*fc_tx*t+phase_tx);

  x_t = xi_t + xq_t;
  x2_t = [x2_t(1+lt:2*lt) x_t];
  % ----- ----- END Transmitter ----- ----- %


  % ----- ----- Receiver ----- ----- %;
```

```
%CHANNEL
y_t = awgn(x_t, SNR);
delay_idx = mod(floor(delay/dt),lt)+1;
idx = [delay_idx:(delay_idx+lt-1)];
y_t = awgn(x2_t(idx), SNR);

%SYMBOL TIMING RECOVERY

if state == 2

  before = 0;
  after = 0;

  for l = 1:(lt/4) + 1
    before += y_t(mod(ks - l - shift,100) + 1); %Sum of the samples
        before the sample used for demodulation
    after += y_t(ks + l - shift); %Sum of the samples after the
        sample used for demodulation
  end

  timing_error = abs(before) - abs(after) %Difference of the sum to
      see the timing error. It should be near zero if the sample used
       for demodulation is the highest point of the sine

  if timing_error > 1
    ks = ks - 1; %Shift of one sample before if the before sum is
        higher than the after sum

  elseif timing_error < -1
    ks = ks + 1; %Shift of one sample after

  else
    var_t++; %If timing error is near zero, var_t is incremented
  end

  if var_t == 5 %If there is 5 symbols with a timing error near zero,
       the real symbols can be send
    state = 3;
  end

  if ks < shift + 1 %To make sure than the samples place exists
    ks = shift + 1;
  end
end

%CARRIER FREQUENCY AND PHASE RECOVERY

  aux_vco_t = -sin(2*pi*fc_rx*t + phase_rx - pi/4);
  aux_phase_error_t = (y_t ./ pulse_shaping) .* aux_vco_t;
```

```
    phase_error = mean(aux_phase_error_t);

    if state == 1

      if sync_flag  <= 0  %TODO : send the carrier only at the
          beginning to recover the frequency and the phase
        freeze = 1;

        fc_error = (phase_rx - last_phase);
        last_phase = phase_rx;

       % maf_f(i) = maf_f(i-1) + fc_error/Ts; %Derivate the phase to
           have the frequency
        % maf_f(i) -= fifo_f(maf_f_l);
        % fifo_f(2:maf_f_l) = fifo_f(1:maf_f_l-1);
        % fifo_f(1) = fc_error/Ts;
        phase_rx = sign(phase_rx + phase_error) * mod(abs(phase_rx +
            phase_error),2*pi);
%     fc_rx = fc_rx + maf_f(i)/pi/2/maf_f_l;
%      fc_rx = fc_rx + fc_error * 10
        sync_flag ++;
        if abs(phase_error) < 0.001
          if i > 2
            sync_flag
            sync_flag = 1;
            state = 2;
          end
        end
      end
    end

    i++;
    fc_rx;
    phase_rx;
    figure(10); hold on;
    plot(t,aux_phase_error_t);
    plot([t(1) t(1)+Ts],phase_error*[1 1],'k');
    plot([t(1) t(1)+Ts],phase_rx*[1 1],'g');
    plot([t(1) t(1)+Ts],fc_error*[1 1],'r');
%     plot(t,fc_rx*ones(1,length(t)),'c');

    %RECOVERY OF YI_T AND YQ_T
    carrier_inph = cos(2*pi*fc_rx*t + phase_rx);
    carrier_quad = sin(2*pi*fc_rx*t + phase_rx);

    yi_t = y_t .* carrier_inph;
    yq_t = y_t .* carrier_quad;
    % TODO The LBP filter is missing
  %   lbp_length = 4;
```

```
%    lbp_filter = [ones(1,lbp_length) zeros(1,lt-lbp_length)];
%    lbp_filter /= sum(lbp_filter);
%    yi_t = filter(lbp_filter, [1 zeros(1,19)], [xi_aux_t yi_non_lbp_t
    ])(lt+1:2*lt);
%    yq_t = filter(lbp_filter, [1 zeros(1,19)], [xi_aux_t yq_non_lbp_t
    ])(lt+1:2*lt);
%    xi_aux_t = yi_non_lbp_t;
%    xq_aux_t = yq_non_lbp_t;

% FILTERING BY FFT
   aux = yi_t + j*yq_t;
   aux = fft(aux) .* [1 1 zeros(1,lt-2)];
   aux = ifft(aux);
   yi_filter_t = 2*real(aux);
   yq_filter_t = 2*imag(aux);


   yi_k = yi_filter_t(ks - shift);
   yq_k = yq_filter_t(ks - shift);

% TODO : FIR filter but for the mean time, FFT is good to implement
    the carrier recovery
%  %figure(3)
%  b = fir1(100,0.001);
%  %freqz(b,10e6);
%  yi_filter_t = 2*filter(b,1,yi_t);
%  yq_filter_t = 2*filter(b,1,yq_t);
%  yi_k = yi_filter_t(50)
%  yq_k = yq_filter_t(50);

%QAM DEMAPPER
   receivedSymbols = yi_k + j*yq_k;
   [mindiff minIndex] = min(receivedSymbols - mappingTable);
   symbolIndexAfter = minIndex - 1;

   if state == 3

     if symbolIndexAfter == 0
       if abs(phase_error) >= 0.001
         'Correction of the phase'
          phase_rx = sign(phase_rx + phase_error) * mod(abs(phase_rx +
              phase_error),2*pi);
       end
     end

     for i = 1:bitsBySymbol
       bits(i) = mod(symbolIndexAfter,2);
       symbolIndexAfter = floor(symbolIndexAfter/2);
     end
```

```
    bits = fliplr(bits);

    for i = 1:bitsBySymbol
      dataOut((k-1)*bitsBySymbol +i) = bits(i);
    end

  end

  % ----- ----- END Receiver ----- ----- %


if state > 1
 % ----- ----- Plot Transmitter Signals ----- ----- %
 if (PLOT_TX == 1)
   figure(1);
   plot([t(1) t(1)+Ts],[0,0],'k-');
   stem(t(1), xi_k,'b','linewidth',3);
   stem(t(1), xq_k,'r','linewidth',2);

   plot([t(1) t(1)+Ts],[-3,-3],'k-');
   plot(t,xsi_t-3,'b:','linewidth',2);
   plot(t,xsq_t-3,'r-','linewidth',1);

   plot([t(1) t(1)+Ts],[-6,-6],'k-');
   plot(t,xi_t-6,'b:','linewidth',1);
   plot(t,xq_t-6,'r-','linewidth',1);

   plot([t(1) t(1)+Ts],[-9,-9],'k-');
   plot(t,x_t-9,'g-','linewidth',1);
   figure(1);
   axis([t(1)-window_size*Ts t(1)+Ts -11 2]); %TODO: implement a
       plotting buffer with a windown from 5 to 10 Ts
   drawnow ("expose");
   title('xik(blue)/xqk(red), xsit/q, xit/q, xt');
 endif;
 % ----- ----- END Plot Transmitter Signals ----- ----- %


 % ----- ----- Plot Receiver Signals ----- ----- %
 if (PLOT_RX == 1)
   figure(2);
   plot(t,y_t-9,'g-','linewidth',1)
   plot([t(1) t(1)+Ts],[-9,-9],'k-');
   axis([t(1)-window_size*Ts t(1)+Ts -11 2]); %TODO: implement a
       plotting buffer with a windown from 5 to 10 Ts
   drawnow ("expose");

   plot([t(1) t(1)+Ts],[-6,-6],'k-');
```

```
        plot(t,yi_t-6,'b:','linewidth',1)
        plot(t,yq_t-6,'r-','linewidth',1)

        plot([t(1) t(1)+Ts],[-3,-3],'k-');
        plot(t,yi_filter_t-3,'b:','linewidth',2)
        plot(t,yq_filter_t-3,'r-','linewidth',1)

        plot([t(1) t(1)+Ts],[0,0],'k-');
        stem(t(1), yi_k,'b','linewidth',3);
        stem(t(1), yq_k,'r','linewidth',2);

        title('yik(blue)/yqk(red), yifiltert/q, yit/q, yt');
    end
    % ----- ----- END Plot Receiver Signals ----- ----- %
  end

    % ----- ----- Update time and plots ----- ----- %
    t = t + Ts;
    if freeze == 0
      k++;
    elseif state == 3
      freeze = 0;
    end
    fflush(stdout);
    % ----- ----- END Update time and plots ----- ----- %
endwhile;
% ----- ----- END Main Loop ----- ----- %
dataInTest = dataIn(1:end-2);
dataOutTest = dataOut(3:end);

%---------------- BER ----------------

total_error = 0;

for i = 1:n-2
    if dataIn(i) != dataOut(i+2)
      total_error ++;
    end
end

% Calculation of BER to return the result
BER = total_error/n;

% Showing final results
disp(['Total wrong bits = ' num2str(total_error)]);
disp(['BER = ' num2str(BER)]);
%
%figure(3);
%plot(vco_f,'c;vco f;');
```