Technical University of Crete

Department of Electronics and Computer Engineering
Telecommunications Division

Quadrature Amplitude Modulation, Synchronization and Viterbi Equalization for experimental wireless ftp radio link over analog FM

Supervisor: **Nikolaos Sidiropoulos**

Committee: **Athanasios Liavas**

Alexandros Potamianos

Mpervanakis Markos

Devoted to Chaos

Abstract

This thesis is a part of team effort to implement a real-time, low cost wireless communication system through analog FM, capable of reliable data transmission. Due to the simple hardware, most of the signal processing is performed in software, which makes the WFTP versatile since it allows easy integration of any digital signal processing methods and algorithms. Consequently, the system is broken into several modules that are assigned to each member of the team, thus team management and smooth cooperation were key factors for the design and development of this prototype.

TABLE OF CONTENTS

TABLE OF FIGURES

1 The Wireless FTP Communication System

1.1 Introduction

The objective of this project was to design and develop a software modem combined with appropriate low cost and commonly used hardware that shall be mainly used for simple file transfer between two personal computers. The actual wireless data transfer is achieved through an analog FM radio link using an FM transmitter and receiver while the digital to analog (D/A) and A/D conversion is performed through PC compatible sound cards. The remaining operations like modulation, encoding etc… are handled by software. The lack of a single transceiver unit in both ends, due to budget restrictions, restricts the wireless communication to be only from the transmitter to the receiver thus, in order to develop a closed loop system an existing wired network configuration is used. Therefore, the general components of the WFTP communication system consist of:

- PC as server
- PC as client
- Server software
- Client software
- FM transmitter
- FM receiver
- Wired network infrastructure

The following figure (Fig 1.1) depicts a general block diagram of the WFTP components configuration. Notice that the wired network of choice is the Ethernet since the range of the radio link is 10 to 15 meters at most, but in case a stronger transmitter is used, any available data communication link will suffice.

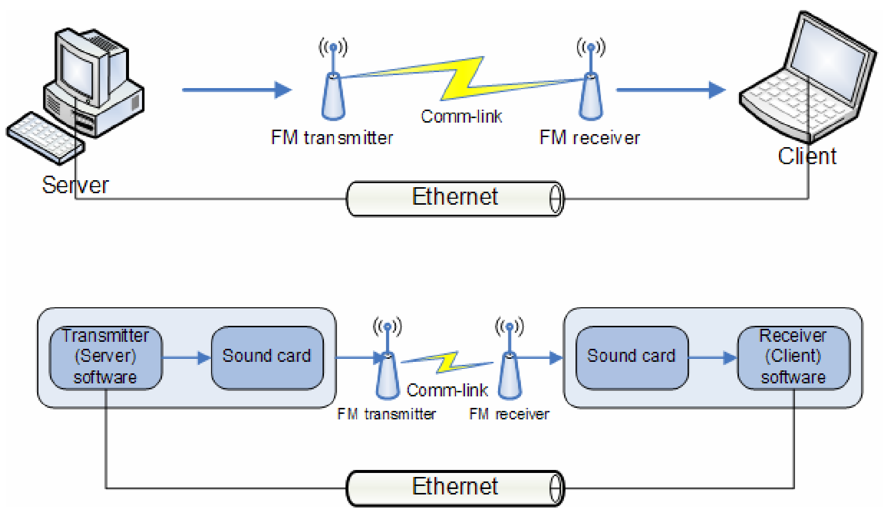


Figure 1.1: An abstract WFTP system configuration.

The subsequent sections provide a description of the main supported features of our system regarding the software – signal-processing capabilities, the intergraded hardware specifications and usage, while the last section presents a complete description of the WFTP system design and operation.

1.2 WFTP Software features

The WFTP software performs the digital signal processing routines except digital to analog (D/A) or analog to digital (A/D) conversion. Although software implemented systems of such scale luck in operation speed, they provide an extremely versatile and configurable platform. The WFTP software consists of the following general sets of procedures that are common in any digital communication system:

- Data segmentation and assembly
- Modulation, Demodulation and Detection
- Error Control Coding and Decoding
- Interleaving
- Channel Estimation and Equalization
- Cyclic Redundancy Check (CRC)
- Synchronization
- Phase Recovery
- Automatic Repeat Request (ARQ)

The versatility of the WFTP software derives from the fact that for each discrete procedure several schemes, algorithms or protocols can easily be integrated, swapped, removed or modified on the fly while the extended parameterization of each unit offers a highly configurable system. The remainder of this section presents an overview of each stage, the functionality and the possible alternative implementations supported by the WFTP prototype.

1.2.1 Data segmentation and assembly

In any communication system, the first stage is the processing of the input information and its conversion to binary data streams of some fixed length referenced as binary data segments. The inverse process, the data assembly is performed at the last stage of the receiver after the redundant information has been removed.

The WFTP system can be used in open loop configuration, without ARQ and CRC, for evaluation purposes or in closed loop when used for reliable file transfer (ftp). Consequently, the binary data streams should be generated randomly, for the first case, or originate from actual data files, for the latter. To cover both operation modes the *WFTP* supports randomized binary data streams creation and simple file decomposition.

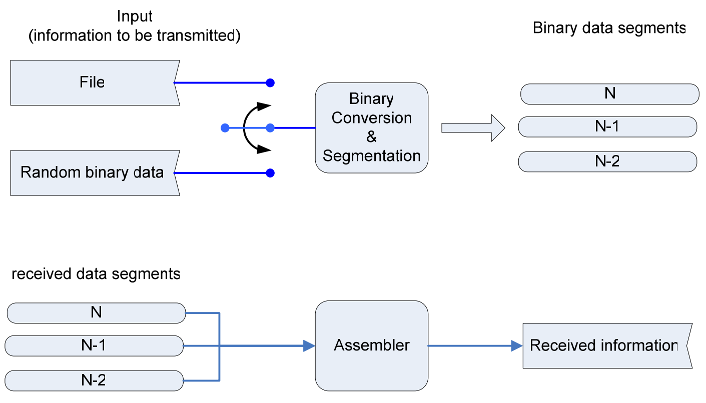


Figure 1.2: Segmentation & assembly operations.

Furthermore, another aspect of WFTP system is the usage of training sequences for frame alignment, symbol clock recovery, adaptive channel equalization or estimation and phase recovery. This procedure is also supported. The binary training stream can either be imported or randomly generated. Note that it must be the same for every data segment and is usually set after the system's first initialization.

1.2.2 CRC

The Cyclic Redundancy Check is a powerful error detection algorithm. The WFTP system uses the CRC-16. It is applied on the binary data stream producing a 16-bit frame check sequence (FCS) witch is concatenated at the end of the data stream (fig. 1.3). From this stage on, the data segment is enriched with additional information until it forms a possibly much larger sequence, referenced as data packet. The receiver applies the CRC to the received data packet, after it has been converted to binary form, decoded and de-interleaved, producing a 16-bit binary sequence. The new FCS is then compared with the encapsulated FCS within the packet. If they differ then the packet is considered corrupted and is discarded.

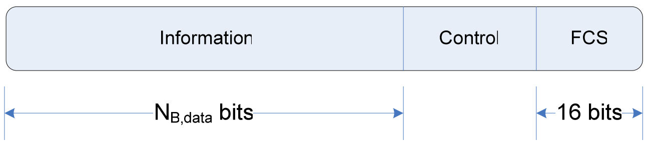


Figure 1.3: CRC-16 generated frame check sequence. CRC covers both data information and control bits.

Use of CRC is optional for the open loop system configuration but it is necessary for a closed loop system.

1.2.3 Error Control Coding

Error control coding, or widely referenced as Forward Error Correcting codes (FECC), is a method of real time error correction or error reduction at the far end receiver, while no return channel is required as in the case of CRC. The concept is based on adding systematic redundancy at the transmit end of a link such that errors caused by the transmission medium can be corrected at the receiver end by means of a decoding algorithm. The amount of redundancy is dependent on the type of code selected and the level of error correction capability desired.

Forward error correction uses channel coding which can be broken down into two broad categories of codes: *convolutional codes* and *block codes*. The WFTP software includes *non-systematic convolutional* coders paired with hard decision Viterbi decoder with available code rates ranging from ¼ to ¾, while for the second category; *BCH*, *Reed Solomon* and *Hamming* coding are also available. Finally, FEC is applied on the data packet including the control information bits and the CRC checksum (FCS). The use of FECC is optional.

1.2.4 Interleaving

Interleaving is the process of systematic reshuffling of the binary data stream, encoded or not. The data packet is partitioned in blocks and then it is rearranged. Interleaving is especially useful when burst errors occur which limit or prohibit the error correction process. Since consecutive blocks are separated, the bit errors can be considered independent at the receiver and the burst errors do not propagate to the neighboring symbols. Thus, using FECC it is possible to correct a long error burst, which might have destroyed all the information in the original block, at the expense of additional delay of the deinterleaving process. Although there is a great range of interleaving algorithms the WFTP supports *random* and *matrix* block interleaving of variable block length. Interleaving is also optional, but in any case is highly recommended.

1.2.5 Automatic Repeat Request (ARQ)

ARQ is another error correction scheme such as FEC but its operation relies on the request of retransmission of a corrupted packet. An error detection algorithm is necessary. There are three variants of ARQ:

- Stop and wait ARQ
- Continuous or selective ARQ
- Go Back N ARQ

The last two schemes require a pipelined infrastructure, or full duplex operation, which in our case is unrealizable due to the Matlab platform and the nature of the sound card handlers. Therefore, the only possible solution provided is the Stop ‘n wait ARQ. Finally, the ARQ is implemented over an Ethernet link utilizing the UDP protocol.

1.2.6 Modulation, Demodulation and detection

The modulation is the last stage of the transmitter where the resulting packet from the previous stages and the addition of a training sequence, is converted into waveforms – symbols that convey the binary information. Note that the modulated symbols are sampled sequences of baseband analog carriers. The actual digital to analog conversion is performed through the PC sound card and it is fed to the FM transmitter through the line out port.

The detection (or correlation detection) is the second stage of the receiver processor, where the sampled received data, from the PC sound cards Line in port, is converted into a sequence of complex symbols. Also, note that the training sequence is separated from the data frame. The received frame after this stage is further processed (equalized etc…), and is fed into the demodulator that will convert it into binary form.

The modulation/ demodulation schemes supported by the WFTP include:

- M-ary Quadrature Amplitude Modulation *(M-QAM*)
- M-ary Phase Shift Keying (*M-PSK*)
- M-ary Pulse Position Modulation (*M-PPM*).

The QAM modulation scheme is also covered in depth in chapter 2.

1.2.7 Synchronization

The synchronization unit of the WFTP system is the interface between the hardware and the software part of the receiver. Usually synchronization is used for symbol clock recovery; however, for the WFTP system its function is twofold. The first role is channel activity detection and handshaking, necessary to establish a communication attempt or data reception. The second role is to isolate the received frame, which in fact is symbol clock recovery. This process is also referred to as framing. Nevertheless both operations rely on the same principles; the use of known sequences at the transmitter and receiver, pattern matching and thresholding, so it is reasonable to share the same structure. The synchronization is covered thoroughly in chapter 4.

Handshaking can also be performed through the ARQ where the transmitter informs the receiver though simple UDP messaging but the incurred delays are unpredictable due to the nature of UDP. However, it remains as an option.

1.2.8 Channel Estimation and Equalization

The communication systems channel impulse response is time varying. In order to combat the channel distortion adaptive structures are the best choice. Channel equalizers depend on estimating an inverse channel impulse response or the channel itself. WFTP supports the following linear recursive equalizers:

- Least Mean Squares (*LMS*)
- Recursive Least Squares (*RLS*)
- Constant Modulus Algorithm (*CMA*)

LMS and *RLS* are initialized using the training sequence to readapt to the varying channel response while the *CMA* belongs to the family of blind equalizers.

In addition, WFTP also supports the non-linear MMSE Viterbi Equalizer based on the Viterbi algorithm (VA). Unlike the *LMS* and *RLS*, the *VE* needs an estimation of the channel impulse response, provided by the Least Squares solution, as referenced in Chapter 3.

1.2.9 Phase Recovery

Phase recovery uses the training sequence to apply a linear transformation on the received data frame. In a constellation diagram, the added phase results in a rotation and as a result simple geometric N-dimensional transformation can restore the original phase. Phase recovery is also optional.

1.3 WFTP Equipment

The equipment of the WFTP communication system consists mainly of two personal computers, a low power FM transmitter, a radio receiver and a dipole antenna.

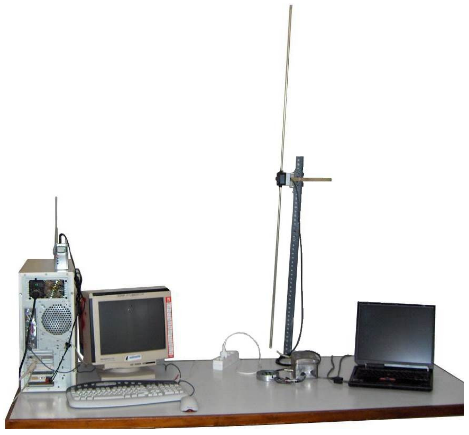


Figure 1.4: The WFTP prototype communication system

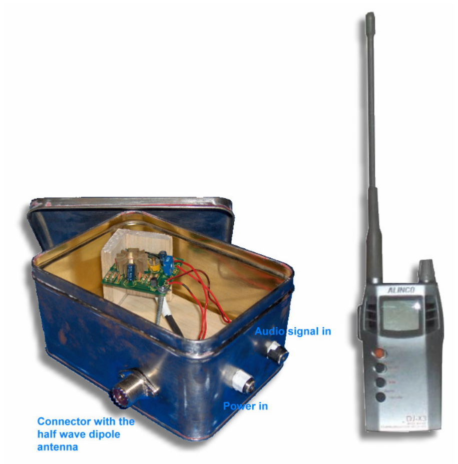


Figure 1.5: FM transmitter & radio receiver

The FM transmitter and the antenna are built from members of the team, while the wide band communications receiver (ALINCO Dj-X3) was recommended by out advisor, Mr. N. Sidiropoulos.

FM transmitter specifications:

- Power supply voltage: DC4.5 V ~ DC12 V
- Frequency Range: 88 MHz ~ 108 MHz
- Output power 1 W @ 12V
- Half wave dipole antenna (also see fig.1.6).

Remark: The casing of the transmitter should consist of a conductive metal or grid to provide a faraday globe. In addition, the casing, the transmitter and the antenna must be grounded. Often the casing acts as a common ground contact.

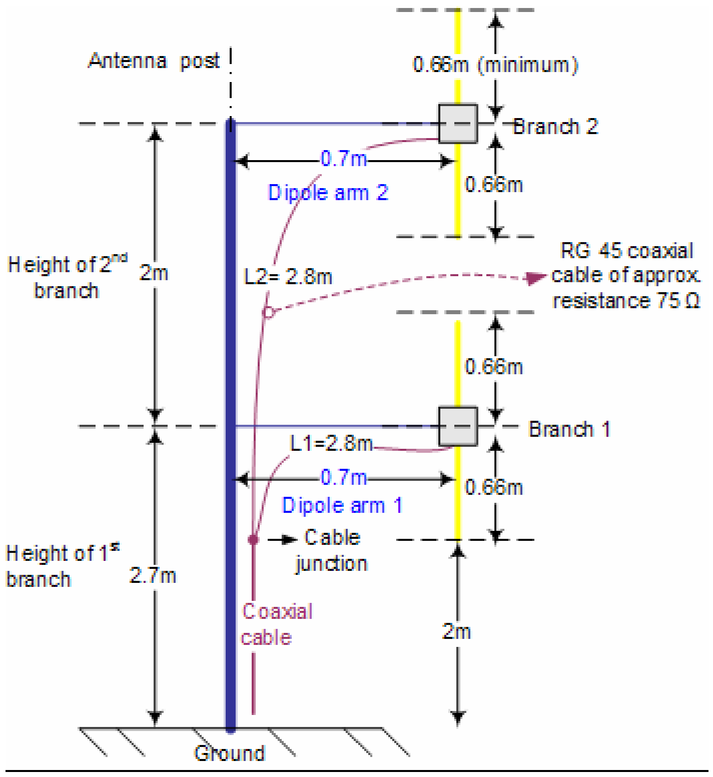


Figure 1.6: Dual branch half wave dipole antenna schematic.

The dipole antenna schematic is depicted in figure 1.6. Note that a single branch is sufficient, but a generic 2 branch setup emphasized the requirements and specifications. In addition the antenna metrics are configured at the minimum 66cm for transmission at 107.9 MHz but with larger lengths the transmitter is tuned at lower frequencies. Consideration should also be given at the cable fastening on the antenna branches where it should be in direct contact with the aluminum dipoles.

Wide Band Communications Receiver specifications:

- RX frequency range: 0.1 ~ 1300MHz
- Antenna impedance: 50Ω
- Battery voltage: DC3.6V ~ DC6V
- External power source: DC4.5V ~ 16V
- Freq. stability: ± 5PPM (-10^{o}C ~ +60^{o}C)

- RX sensitivity: less than 0db @ [FM] 30~500 MHz
- Stereo output

The last major component is the sound card, which is usually part of any personal computer. However, it is useful to post a set of specifications for this hardware as well.

Sound card specifications:

- 24-bit Analog-to-Digital conversion of analog inputs at 96KHz sample rate
- 24-bit Digital-to-Analog conversion of digital sources at 96KHz
- 16-bit and 24-bit recording with sample rates of 8~96KHz
- Line level out (Front/ Side/ Rear/ Center/ Subwoofer)
- Line In
- AC97 compliant
- Signal-to-noise Ratio < 100dB
- Frequency Response at -3dB <50Hz ~ 40KHz

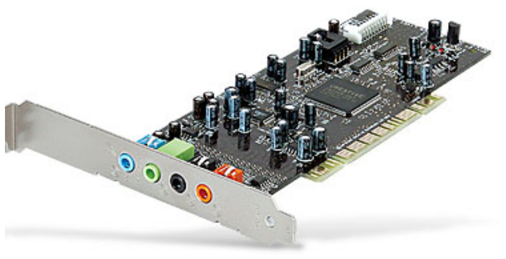
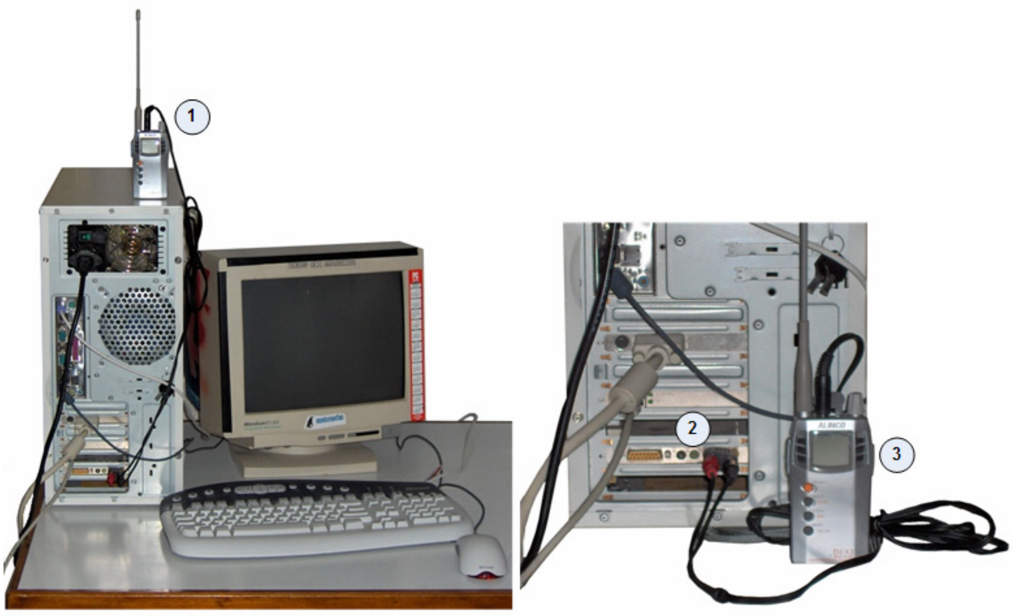


Figure 1.7: Typical PCI soundcard.

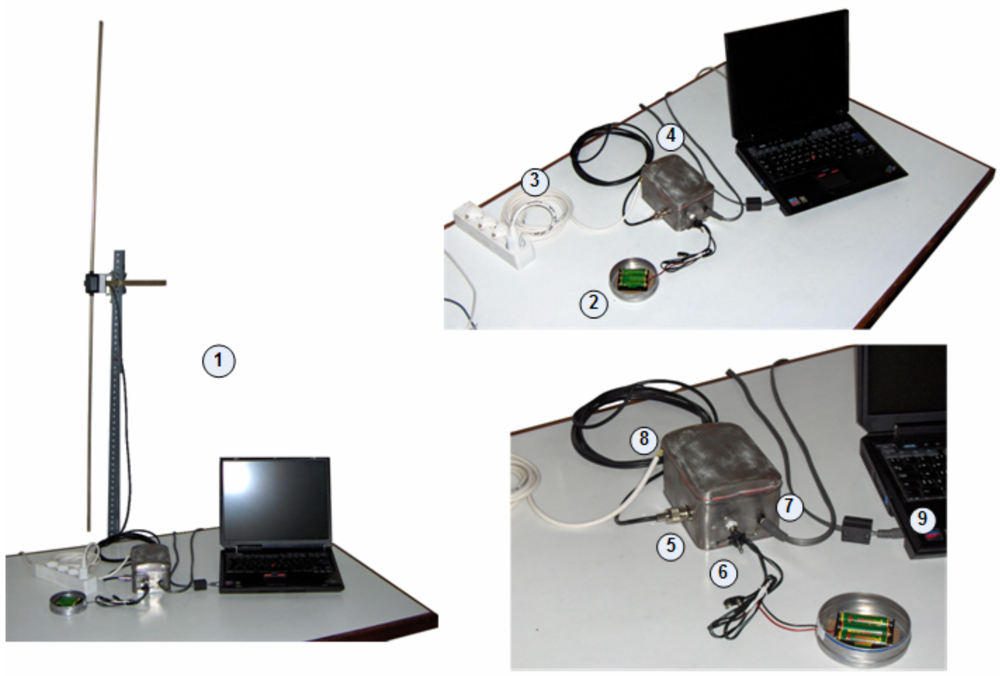
Note that in case the sound card supports multi-channel output only the *Front line out* port is connected to the FM transmitter *audio signal port* (fig 1.5). In addition, Line In port or auxiliary (*Aux*) *Line In* is necessary for stereo recording from a radio receiver.

Figures 1.8 and 1.9 illustrate the receiver and transmitter as well as the system assembly. Notice, though, that the selected power supply of the FM transmitter is provided by batteries instead of a transformer. The reason is that the transformer introduces scramble hum due to the frequency (50Hz ~ 60Hz) of the AC current.



1. Receiver : Desktop PC , Wide band communications receiver
2. Soundcard
3. Wide band communications receiver connected with the Personal Computer via the "line in" of the soundcard

Figure 1.8: Receiver



1. Transmitter : FM transmitter , Laptop , Half wave dipole antenna , Battery case
2. Battery case : 3 AA batteries 1.5V
3. Ground cable
4. Low power FM transmitter
5. Cable to the antenna
6. Power in : DC 4.5V
7. Line in (audio signal to be transmitted)
8. Proper case grounding
9. Line out from laptop

Figure 1.9: Transmitter

1.4 WFTP Design and Operation

The scope of this section is to provide a complete presentation of the WFTP communication system architecture, involving both hardware and software configuration and system operation.

1.4.1 Data link Design

Recall, from the introductory section, that the major components of the WFTP are the radio hardware, the personal computers and the related software. In particular, since the hardware setup is self-explanatory, we are interested in the software. To begin with, the software sub-modules at the receiver and the transmitter are grouped into four main modules witch run as processes using the Matlab platform. These are the:

- *Transmitter Unit*
- *Transmitter Server*
- *Receiver Unit*
- *Processor Unit*

The *Transmitter Unit* module incorporates all the sub-processes that operate on the data from the packet creation to data modulation and transmission through the sound card. The transmitter server is the UDP based Stop and Wait ARQ interface. The Receiver Unit consists of the sub-modules that handle negotiation and synchronization operations as well as the implementation of the ARQ interface at the receiver, while the Processor Unit is responsible for filtering, demodulation etc… of the received packet. The implementation of these modules defines the data link architecture and, subsequently, the communication system architecture. To be more specific, we are faced with two scenarios regarding the main processes interaction, the *full* and *half duplex* data link operation.

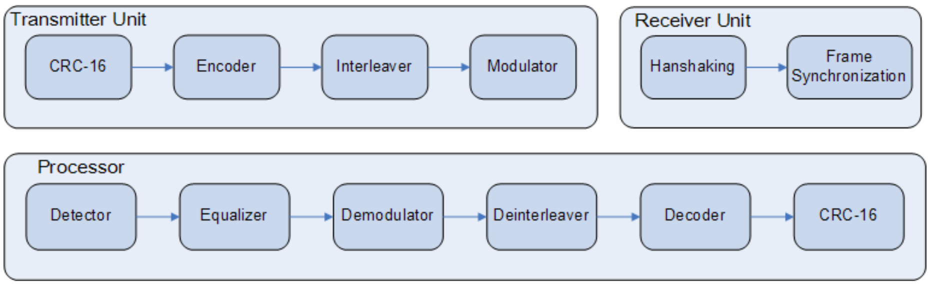


Figure 1.10: Main software units and their sub-modules.

1.4.1.1 Full Duplex design

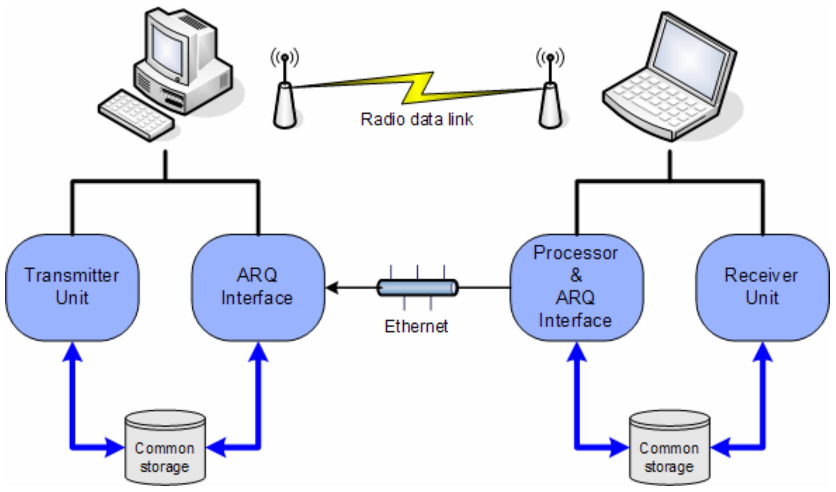


Figure 1.11: Full duplex configuration. The common storage could be visualized as a binary data packet buffer. Note also that each process is autonomous and operate in parallel.

In the first scenario, depicted in figure 1.11, the *Receiver Unit* and the *Processor Unit* become two different processes sharing common memory. With this configuration, the system becomes pipelined or full duplex meaning that while the receiver accepts incoming packets, the processor will also operate on the previously received packets. In this manner, Go Back N or Selective ARQ can be applied. This also means that the transmitter will also operate continuously and its processes must run in parallel. Therefore, both options require multiple instances of the Matlab platform to be active in each PC. However, due to the intensity of the digital signal processing operations as well as the audio operations are CPU and memory intensive and a single processor cannot handle that load. The solution of course is to use multi processor machines or PC clusters, which we lacked.

1.4.1.2 Half-Duplex Design

Since a full duplex data link cannot be applied, the remaining solution is to use half-duplex link, illustrated in figure 1.12, where both main functions of the receiver and the transmitter execute in series. Under these limitations, the applicable ARQ is the Stop and Wait that may be simple, but inefficient due to the long dead timing periods where the receiver and the transmitter remain inactive.

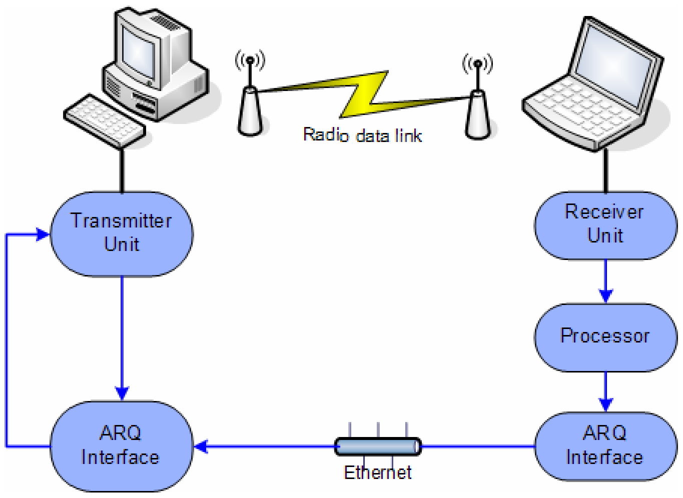


Figure 1.12: Half duplex configuration. Note that only one active process is permitted.

1.4.2 The WFTP Packet structure

The WFTP communication is packet based. A packet is the package where the exchanged information is encapsulated, enriched with redundant information used by the receiver to identify, extract, reconstruct and deliver the transmitted information. A WFTP binary data packet, illustrated in figure 1.13, consists of four fields:

- **Data segment**: The original information is partitioned in blocks of binary data, the useful information, which can be encoded or/and interleaved. It includes the CRC sequence.
- **Header**: Contains information about the sender and the recipient, such us IP address, the packet number, the binary stream and position the data segment belongs etc. However, Stop and Wait ARQ delivers in order and only one acknowledged packet at a time, thus packet numbering is unnecessary. In addition, the current WFTP does not support multi-user communication so the IP addresses are fixed. Under these conditions, the *Header* can be omitted.

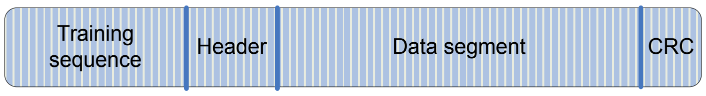


Figure 1.13: Binary packet fields.

- **CRC segment**: 16-bit checksum (FCS) of the current data partition (prior encoding or interleaving).

- **Training sequence**: Predefined sequence, known at the receiver and transmitter, used for synchronization, equalization and phase recovery. It does not undergo any operation other that modulation prior transmission.

1.4.3 WFTP Software structure and operation

The main software modules presented in the preceding section 1.4.1, consist of sub-modules or sub-processes, each implementing a specific stage of the software defined WFTP modem. The system configuration is controlled by a main unit, the *SystemSetup* module. Instead of module-by-module description, a schematic representation is much more helpful. Figure 1.14, illustrates the entire modem structure and the configurable parameters regarding each part of the WFTP system.

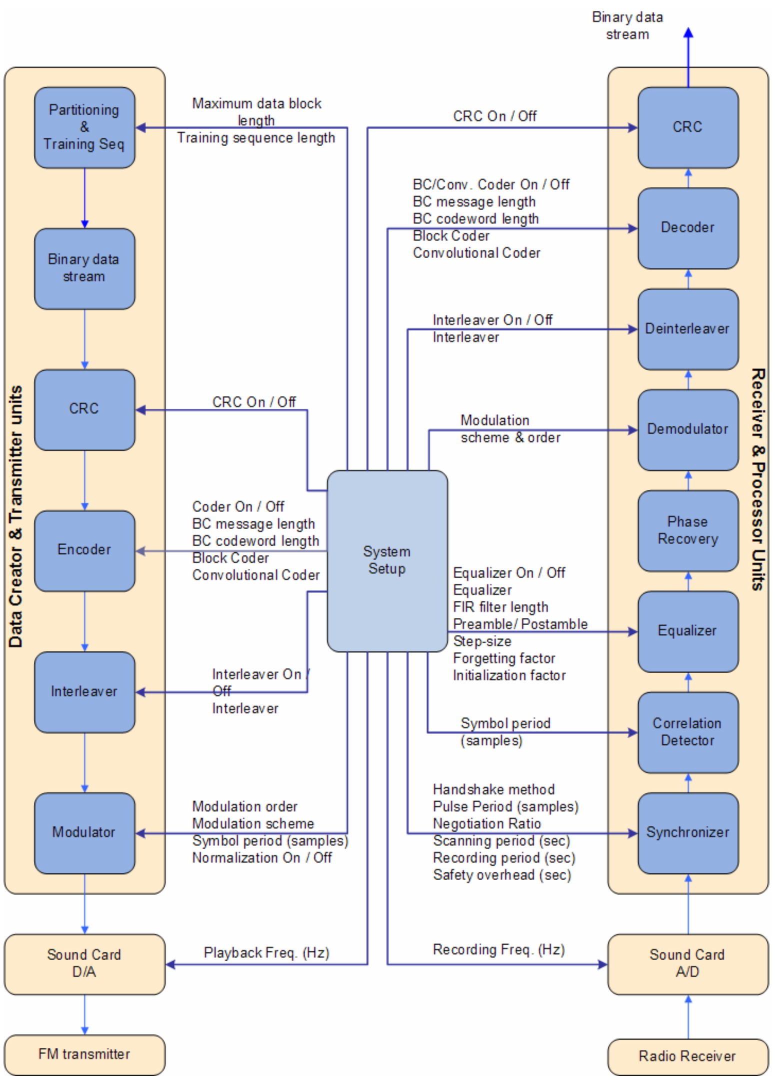


Figure 1.14: Block diagram of the WFTP system.

When the WFTP is used for file transfer, the first stage is to load the selected file and convert it to set of binary streams. Note that the number of resulting blocks from the file segmentation equals the number of packets eventually transmitted. Keep in mind that the training sequence, independent of the data stream, is generated at the same stage at the first initialization of the system or when the length is reset. Each data segment is stored in memory, which, in our case, is a structured file. The *CRC* stage operates on the data stream generating a 16-bit checksum (*FCS*), appended in the data sequence. The new binary sequence (data and FCS), goes through the *Encoder* module. Depending from the system configuration related with this stage, the resulting encoded sequence is much larger since it contains redundant information for error correction. The following *Interleaver* does not affect the packet length, just "reshuffles" the input stream. The last stage at the transmitter software is the *Modulator*. Prior the reforming of the binary data to symbols, the training sequence is concatenated at the head of the data stream, shaping the packet. The final form of the modulated sequence consists of digital waveforms of duration T_S samples. These steps cover the creation of the transmitted packet. For convenience, all the packets are precomputed and stored. Prior each transmission the corresponding packet is simply loaded from a database and then outputted through the line out port of the sound card. The output of the sound card is fed into the *FM transmitter* who handles the frequency up conversion and analog FM modulation. Down conversion and FM demodulation are achieved by the *radio receiver* who also feeds the received analog signal in the corresponding sound card line in port. The sound card records, samples, and quantizes the input signal with the same frequency used in the transmitter, *Fs*.

The data transmission, explained so far, is a simple process; when the packet is ready, it is driven into the sound card interface. In the same manner, the receiver initiates a data reception session, but since the transmitter and receiver are independent, the latter must be notified of any data activity. The problem is solved through negotiation. The interface between the sound card and the software, at the receiver end, is the *Synchronizer unit*. The synchronizer controls the recording process, initiating data reception after a successful negotiation or handshake (see chapter 4), and isolates the transmitted packet, the part of the signal that carries the useful information, from the recorded sequence. The frame detection is achieved using the training sequence, known at both ends also at the *Synchronizer*. The isolated frame is then passed through a *signal demodulator* who converts the sampled signal into complex representation regardless the modulation scheme used at the transmitter (except PPM). The *Equalizer Unit* filters the detected sequence to invert the distortion introduced by the systems channel while the phase shifting that can also occur is removed though a linear transformation at the *Phase Recovery* unit. The filtered data packet, without the training sequence, is then demodulated (symbol detector – *Demodulator Unit*) to produce a binary data stream. The binary data packet must be deinterleaved (*Deinterleaver unit*) and decoded (*Decoder unit*). The resulting bit stream contains the transmitted data bit stream and the 16-bit CRC checksum. Even though decoding performs error correction, the result is not guaranteed to be error free. A possibly corrupted packet is identified by the *CRC unit*.

The above short description summarizes the main operation of the WFTP software defined radio. The subsequent sections refer to specific parts of the WFTP communication system that could be considered as transparent concerning the software operation.

1.4.4 Stop and Wait ARQ

The implemented ARQ relies on the CRC error detection and simple messaging using the UDP protocol over Ethernet. The operation of the stop 'n wait ARQ is summarized in some simple steps:

1. Packet and acknowledgement numbering requires only one bit (0/1).
2. The transmitter (Transmitter Unit) sends PACK0 and enters the server mode for a specific amount of time – the timeout period. During this state the transmitter waits for a reply from the receiver (Receiver Unit) with the same number (ACK/NACK0). If a timeout occurs or a NACK message is received then the same packet is retransmitted. If a positive acknowledgement is received (ACK0) with the same numbering, then it loads and transmits the next packet PACK1.
3. The receiver checks each packet (*Processor Unit - CRC*) and transmits a corresponding message through UDP (as a client) with the same numbering ACK0/1 for correct packet or NACK0/1 for corrupted one.
4. Duplicate packets or responses are discarded.

1.4.5 Audio Playback & Audio Recording

The hardware interface between the receiver and transmitter software is the sound card that performs analog to digital and digital to analog conversions respectively. In practice the transmission is performed with audio playback using the *wavplay* matlab function while the reception though audio recording and the *wavrecord* function. The input and output is a baseband sampled waveform, as described in section 1.4.3. Even though this selection substantially limits the overall hardware complexity, however it has some serious drawbacks.

The sound card, as a digital device, accepts (playback) and outputs (recording) vectors with amplitudes limited in the range of [-1, 1]. If the modulated signal exceeds these limits it is simple chopped, thus normalization should be performed prior transmission. The side effect is that it limits the symbol energy or the signals power degrading the noise immunity capabilities of our system.

Another significant drawback is that the maximum playback and recording sampling frequency is limited to 96 KHz at maximum supported by only modern sound cards. This implies that we the minimum propagation delay is approximately 96msec. To emphasize on the transfer time (playback time) consider the transmission of a packet of 10kbits modulated with 4-PSK (M=4), symbol period Ts=10 samples and playback frequency Fs=96 KHz. The transfer time is given as the ratio of the samples of the modulated packet and the Fs:

$$t_{transfer} = \frac{N_{bits}}{\log 2(M).F_S}.T_S = \frac{10000*10}{2*96000} \approx 0.5\sec$$

The *transfer time* is also associated with the recording timing period at the receiver; witch is adjusted dynamically based on the previous relation. Unfortunately these operations import random delays during their initiation due to memory and resources management of the operating system. Therefore the recording period during transmission is set as $t_{transfer} + t_{safe}$ where t_{safe} is a timing constant ranging from 0.1 to 0.5 seconds depending on the handshaking configuration.

1.4.6 Handshaking

Handshaking informs the receiver that a communication session has initiated or terminated. Since both ends are independent the receiver must know when to start recording. This is achieved by the transmission of a wake up signal prior each packet. The *Receiver Unit* monitors the channel for small periods for this pattern. When the receiver accepts this specific pattern, starts to record for a specific amount of time which is dynamically configured based on the system settings. The termination of the session is performed though a UDP message transmitted from the *Transmitter Unit* notifying the *Receiver Unit* that the last packet was transmitted. Note that implemented WFTP protocol is *connectionless* thus, a session must be set for each transmission.

1.5 Executive Summaries

The design and development of the WFTP communication system prototype was a team effort where each member was assigned with the development and evaluation of one or more specific modules. The team members and their associated contribution are:

Iliakis Evangellos. Vangelis was assigned primarily with the development and evaluation of the convolutional coding and Viterbi decoding. In addition he implemented the modules for CRC and PSK modulation as well as the UDP based Stop and Wait ARQ.

Kardaras Georgios. His primary responsibility was the implementation of the Block Coding modules and the PPM modulation scheme. He also contributed the Block Interleaving/ Deinterleaving modules**.**

Kokkinakis Chris. He was assigned with the implementation and evaluation of the RLS, LMS and CMA equalizers.

Mpervanakis Markos. Mark was assigned with the development of the Viterbi Equalization and QAM modulation scheme. Moreover he implemented the Frame Synchronization and Handshaking modules. He also contributed to the overall system structure design and assembly of the software modules developed by the team to a completely operating and highly configurable application.

The WFTP Team. The team as a collective contributed with brainstorming and solutions to overcome the obstacles that emerged. However, the most important part was the support we all needed at difficult times.

2 Quadrature Amplitude Modulation

2.1 Introduction

This chapter is concerned with the M-ary QAM modulation; and its counterpart, the demodulation scheme, as is implemented and integrated in our communication system referred to as the FM wireless FTP prototype. To begin with, a theoretical perspective of QAM is presented followed by practical implementations in order to make the transition from theory to practice as smooth as possible. Finally, experimental results are presented through real time simulation of the prototype using the current modulation scheme.

2.2 QAM Modulator

As indicated, briefly, in the previous chapter; the modulator is the interface that maps a sequence of binary data into a set of corresponding signal waveforms suitable for transmission through a communication system. The QA modulator consists of several distinct parts – processes that handle the mapping from bits to symbols, energy normalization of the constellation and finally the generation of suitable waveforms which will be transmitted through a sound card to an FM transmitter.

2.2.1 Mapping

2.2.1.1 Background

M-QAM signals are obtained by simultaneously impressing two separate k-bit symbols, on two quadrature carriers, *cos2πft* and *sin2πft*. The corresponding signal waveforms may be expressed as:

$$\begin{aligned} s_m(t) &= \mathrm{Re}[(A_{mc} + jA_{ms})g_T(t)e^{j2\pi f_c t}], m = 1,2,\ldots,M, 0 \le t \le T \\ &= A_{mc}g_T(t)\cos 2\pi f_c t + A_{ms}g_T(t)\sin 2\pi f_c t \end{aligned} \qquad (2.1)$$

where A_{mc} and A_{ms} are the information bearing amplitudes of the quadrature carriers and $g_T(t)$ the is signal pulse [*6 pg. 400*].

It’s obvious from the above equation that until the signal is to be transmitted we need not consider with neither pulse shaping nor carrier multiplexing. It’s also apparent that the baseband equivalent signal-space representation can be expressed as complex or Cartesian coordinates of 2 independent quadrature components, in-phase (I) and quadrature (Q) corresponding to x and y axis, respectively:

$$s_m = A_{mc} + jA_{ms}, m = 1, 2..., M \tag{2.2}$$

Several constellation diagrams have been proposed for QAM transmission over Gaussian channels. However, the three constellations shown in figure 2.1 are often preferred. The essential problem is to maintain a high minimum distance between points whilst keeping the average power required for the constellation to a minimum. Calculation of minimum distance and average power is a straightforward geometric procedure and has been performed by Proakis [6] on a range of constellations. The results show that the square constellation is optimal for Gaussian channels. For the remaining of this chapter the discussion will be restricted to square and generally orthogonal constellations.

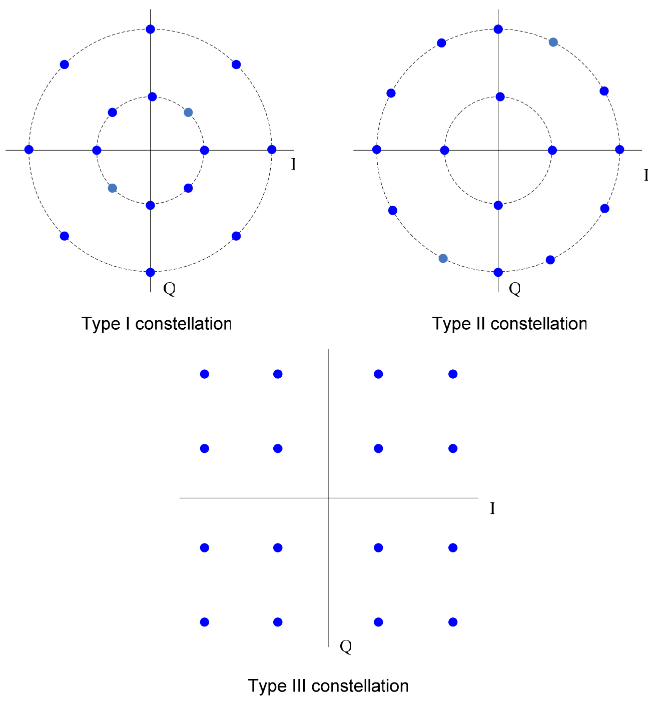


Figure 2.1: Examples of Type I, II, and III QAM constellations.

2.2.1.2 Orthogonal Constellations

The most obvious constellation diagrams are the orthogonal, if each symbol carries odd number of bits, and the square, provided an even number of bits. Orthogonal constellations are implemented by keeping the minimum Euclidean distance, between each message point, constant. To design the constellation, the first step is to implement a lookup table –matrix that directly maps k bits into symbols. Recall from the previous paragraph, that a symbol can be represented as

$$s_{IQ} = (A_{mc}, A_{ms}) \tag{2.3}$$

So the first step is to define a set of discrete values –amplitudes, for each of the in-phase (A_{mc}) & quadrature (A_{ms}) components. The k bits are distributed independently to each symbol component (A_{mc}, A_{ms}) to k_1 & k_2 bits respectively providing $L_1=2^{k1}$ and $L_2=2^{k2}$ possible values. The range of each component, then, is defined as:

$$\begin{aligned} A_{mc} &= \{(2m_1 - 1 - L_1), m_1 = 1,2...,L_1\} \\ A_{ms} &= \{(2m_2 - 1 - L_2), m_2 = 1,2...,L_2\} \end{aligned} \tag{2.4}$$

where

$$\begin{aligned} L_2 &= 2^{k_2} \\ L_1 &= 2^{k_1} \\ k_1 + k_2 &= k \\ M = 2^k &= L_1 L_2 \end{aligned} \tag{2.5}$$

The resulting $L_1 x L_2$ of the (Amc, Ams) element matrix is then:

$$s_{ij} = (A_{ic}, A_{js}) = \begin{bmatrix} (-L_1+1, L_2-1) & (-L_1+3, L_2-1) & \cdots & (L_1-1, L_2-1) \\ (-L_1+1, L_2-3) & (-L_1+3, L_2-3) & \cdots & (L_1-1, L_2-3) \\ \vdots & \vdots & \cdots & \vdots \\ (-L_1+1, -L_2+1) & (-L_1+3, L_2+1) & \cdots & (L_1-1, -L_2+1) \end{bmatrix}_{L_1 x L_2} \tag{2.6}.$$

For example the constellation matrix of a 16-QAM is:

$$(x,y) = \begin{bmatrix} (-3,3) & (-1,3) & (1,3) & (3,3) \\ (-3,1) & (-1,1) & (1,1) & (3,1) \\ (-3,-1) & (-1,-1) & (1,-1) & (3,-1) \\ (-3,-3) & (-1,-3) & (1,-3) & (3,-3) \end{bmatrix}_{4x4}$$

The resulting square constellation is depicted in the scatter plot of figure 2.2.

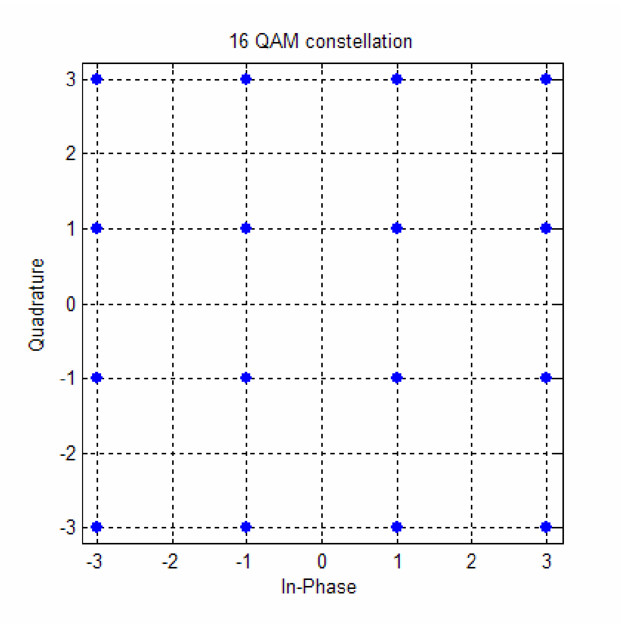


Figure 2.2: 16 QAM constellation.

2.2.1.3 Bit Mapping

Having created the constellation matrix, or in other words, specified the signal amplitudes discrete range, remains the mapping of binary data to symbols. This procedure requires 3 steps (*Fig*. 2.3); a serial to parallel bit-stream conversion, coding (specifically Gray coding) witch is optional and finally the corresponding symbol assignment through the lookup table (constellation matrix).

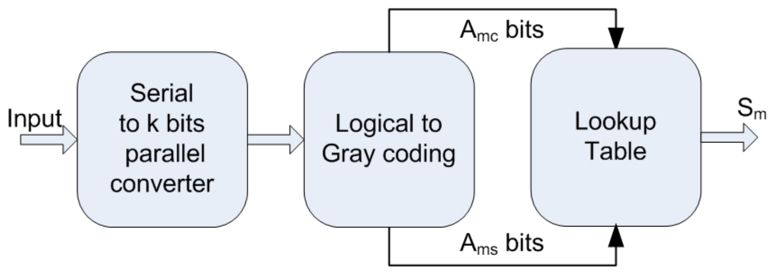


Figure 2.3: Bit mapping process.

The incoming binary data stream is passed through a parallel converter of appropriate width (k or $\log_2$ (M) bits where M is the QAM size). The data are then passed through an optional logic block which converts them to the equivalent gray coded codeword.

This is achieved using the following logic equations or the equivalent logic circuit of fig. 2.4:

$$\begin{aligned}
&F_N = MSB\\
&F_{N-1} = MSB\ XOR\ Bit_{N-1}\\
&F_{N-2} = Bit_{N-1}\ XOR\ Bit_{N-2}\\
&\ldots\\
&F_1 = Bit_2\ XOR\ LSB\\
&where\\
&word : [Bit_N\ Bit_{N-1}\ Bit_{N-2} \ldots Bit_2\ Bit_1]\\
&Bit_N : MSB\ (most\ significant\ bit)\\
&Bit_1 : LSB\ (less\ significant\ bit)
\end{aligned} \qquad (2.7)$$

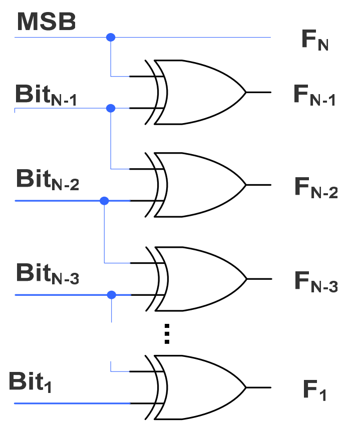


Figure 2.4: Binary to Gray converter

The gray coded word is then split, assigning k_1 bits as index to the first dimension of the lookup table, and k_2 bits for the second. For example, consider the previous example of the 16-QAM. The parallel converter will buffer 4 bits and then feed a 4bit codeword to the Gray converter. The result is also a 4 bits wide binary word which is split to half, assigning the 2 MSB bits as the A_{mc} index and the remaining 2 LSB bits to the A_{ms} or vice versa. It's easier to understand the indexing process by converting the binary indexes (00, 10 etc.) to their decimal equivalent value. Figure 2.5 displays both binary and decimal indexing while figure 2.6 shows the resulting mapping of 4bit words to symbols. Of course the indexing method analyzed so far is the same whether Gray coding is used or not.

binary	$A_{mc} \Leftrightarrow$	00	01	10	11
$A_{ms} \Updownarrow$	*decimal*	0	1	2	3
00	0	$(-3,3)$	$(-1,3)$	$(1,3)$	$(3,3)$
01	1	$(-3,1)$	$(-1,1)$	$(1,1)$	$(3,1)$
10	2	$(-3,-1)$	$(-1,-1)$	$(1,-1)$	$(3,-1)$
11	3	$(-3,-3)$	$(-1,-3)$	$(1,-3)$	$(3,-3)$

Figure 2.5: Lookup table indexing. If a binary word is 0011 then its Gray equivalent is 0010 so the indexing is [00 10] = [-3 1] = [A_{mc}, A_{ms}].

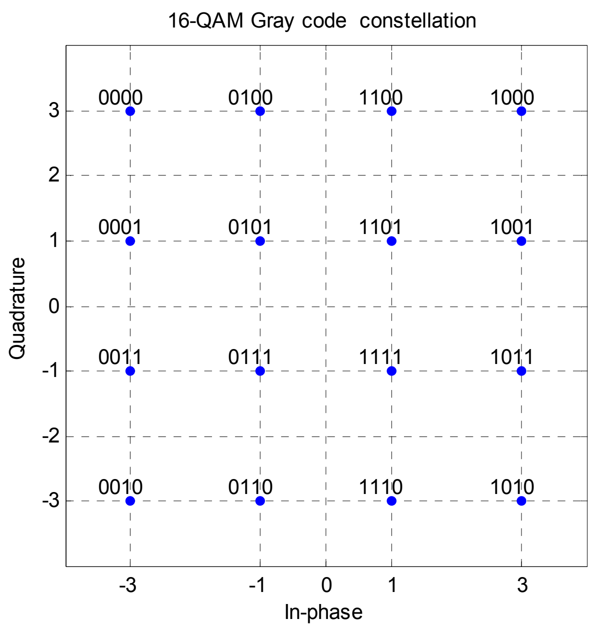


Figure 2.6: 4-bit data mapping using the lookup table of Fig 2.5.

2.2.1.4 Implementation

The most important and complicated unit of the *Mapper* is the constellation matrix. The process of converting each binary word to its equivalent gray code could be an easy task in hardware but from a programming style approach; it's redundant since it can be integrated with the lookup table. Prior the implementation analysis let's set some definitions which shall be used throughout this section (also see equation 2.5):

- *N_b: number of bits.*
- *N_D: number of symbols N_D=N_b/k.*
- $\mathbf{C}$ *: Constellation matrix of size L_1xL_2.*
- $\mathbf{i}$ *: In-phase components vector of size N_Dx1.*
- $\mathbf{q}$ *: Quadrature components vector of size N_Dx1.*

The concept of integrating the Gray mapping process is to achieve the same results through immediate indexing. Reordering the lookup table in a consistent way will provide the solution. For example, interchanging the last 2 columns & rows of the matrix depicted in Fig.2.5 will result in a gray coded matrix (Fig 2.7) that provides the same results. Borrowing a term from digital design the resulted matrix is a *Karnaugh* map [*7 pg. 325*]. The invert process of retrieving the binary word using the symbols is just a matter of reconfiguring the look up table.

$$\left[\begin{array}{c|c|c|c|c|c}
binary & & 00 & 01 & 11 & 10 \\
\hline
 & decimal & 0 & 1 & 3 & 2 \\
\hline
00 & 0 & (-3,3) & (-1,3) & (3,3) & (1,3) \\
01 & 1 & (-3,1) & (-1,1) & (3,1) & (1,1) \\
11 & 3 & (-3,-3) & (-1,-3) & (3,-3) & (1,-3) \\
10 & 2 & (-3,-1) & (-1,-1) & (3,-1) & (1,-1)
\end{array}\right] \qquad
\left[\begin{array}{c|c|c|c|c|c}
binary & & 00 & 01 & 10 & 11 \\
\hline
 & decimal & 0 & 1 & 2 & 3 \\
\hline
00 & 0 & (-3,3) & (-1,3) & (3,3) & (1,3) \\
01 & 1 & (-3,1) & (-1,1) & (3,1) & (1,1) \\
10 & 2 & (-3,-3) & (-1,-3) & (3,-3) & (1,-3) \\
11 & 3 & (-3,-1) & (-1,-1) & (3,-1) & (1,-1)
\end{array}\right]$$

(a) (b)

Figure 2.7: Reordered constellation matrices, (a) is the reordered map of Fig 2.6, while (b) is the final Gray coded, lookup table.

The operations described so far are applied to blocks of data, in a "packaging" fashion so it's most appropriate to express the Bit mapping process likewise. From figure 2.3, a data stream (packet) of N_b bits enters a converter that splits the stream into chunks of k bits each. The latter are used as indexes of a lookup table which, in the end, outputs the corresponding symbols.

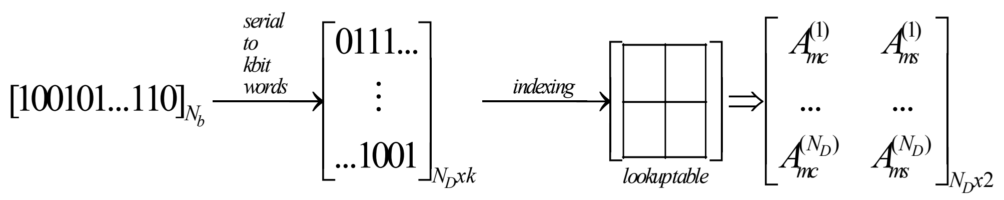


Figure 2.8: Mapping process of data blocks.

2.2.2 Modulation

Recall from the previous chapter that the output port is the Line Out of a PC sound card. This imports two restrictions; first the output modulated data must be a waveform (sampled signal), and last but not least the waveforms amplitude must be bounded within the range of [-1, 1].

A typical digital QAM modulator is depicted in fig. 2.9. After the modulation amplitudes have been generated through the bit mapping process, digital to analog conversion is required followed by pulse shaping prior the frequency up conversion.

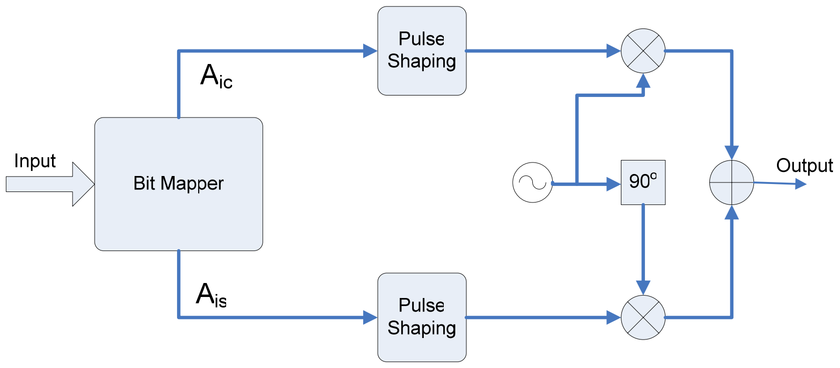


Figure 2.9: Standard QAM modulator.

Let's ignore the pulse shaping and frequency change for now and concentrate on the fact that QAM modulated signals consist of two orthogonal components. Since the frequency change is handled by the FM transmitter, the only consideration is to create a sampled equivalent waveform witch can be transmitted through the sound card. Under these assumptions the entire process of the QAM modulation can be integrated with the pulse shaping by selecting two orthogonal pulses reaching the modified modulator of fig. 2.10 that happens to be the same for the PSK module.

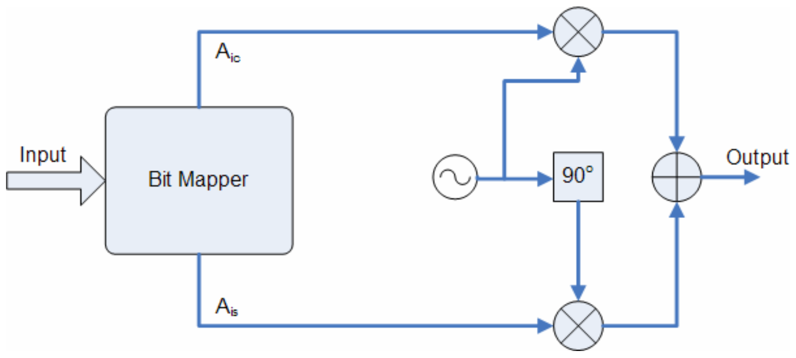


Figure 2.10: Modified QAM modulator.

2.2.2.1 Implementation

The mapping process, as implemented in the previous section, provides the symbols that carry the information bits. Since the information data are processed by blocks the output consist of two vectors **I** and **Q** of the consecutive components (A_{ic}, A_{is}) of each symbol S_i, respectively. Prior the analysis of the implementation it's useful to set some definitions:

- *N_D: number of bits per information block.*

- *N: number of samples per basis function* $\underline{y}(n)$.
- *M: QAM size.*
- *Ns: number of symbols.* $N_S = N_D / \log_2(M)$
- *S: information bearing symbol.* $S_m = A_{mc} + jA_{ms}$ *or* $S = [A_{mc}\ A_{ms}]$.
- A_{mc}*: in phase component of symbol at time m.*
- A_{ms}*: quadrature component.*
- $\mathbf{y}_1, \mathbf{y}_2$ *: sampled orthogonal (*$\mathbf{y}_1^T \mathbf{y}_2 = 0$*) basis functions of size Nx1.*
- $\mathbf{i}_{N_D x1}$ *:vector of in-phase components,* $\mathbf{i} = [A_{1c}\ A_{2c}\ ..., A_{mc} ..., A_{N_D c}]^T$
- $\mathbf{q}_{N_D x1}$ *:vector of in-phase components,* $\mathbf{q} = [A_{1s}\ A_{2s}\ ..., A_{ms} ..., A_{N_D s}]^T$

A single modulated waveform (also transmitted) at time *m* is defined as:

$$tr_m(n) = A_{mc} y_1(n) + A_{ms} y_2(n)\ , n = 1, 2..., N \tag{2.8}$$

Relation (2.8) is expressed in vector notation as:

$$\mathbf{Tr}_{N_S x N} = \mathbf{i}\mathbf{y_1}^T + \mathbf{q}\mathbf{y_2}^T \tag{2.9}$$

Note that the basis functions are produced using the **Gram – Schmidt** procedure *[6]*. Each row of the matrix ***Tr*** is a modulated waveform, of N samples, that can finally be transmitted. The following figures (2.11, 2.12) represent the modulation process of a 15 bits data stream using 8-QAM and 20 samples per symbol. The output of the *mapper* module consists of 5 symbols, each bearing 3 bits of information. The second figure depicts the modulator output using as basis functions **y₁**, **y₂**:

$$\begin{aligned}
\underline{\mathbf{y}}_{\mathbf{1}} &= \frac{1}{\sqrt{E_{f1}}} \underline{\mathbf{f}}_{\mathbf{1}}\text{, where } f_1(n) = \sin(\frac{2\pi n}{N}), n = 1, 2..., N \\
\underline{\mathbf{y}}_{\mathbf{2}} &= \frac{1}{\sqrt{E_{f2}}} \underline{\mathbf{f}}_{\mathbf{2}}\text{, where } f_2(n) = \sin(\frac{\pi n}{N}), n = 1, 2..., N \\
E_{fi} &= \sum_{n=1}^{N} f_i(n)^2 = \underline{\mathbf{f}}_{\mathbf{i}}^T \underline{\mathbf{f}}_{\mathbf{i}}.
\end{aligned} \tag{2.10}$$

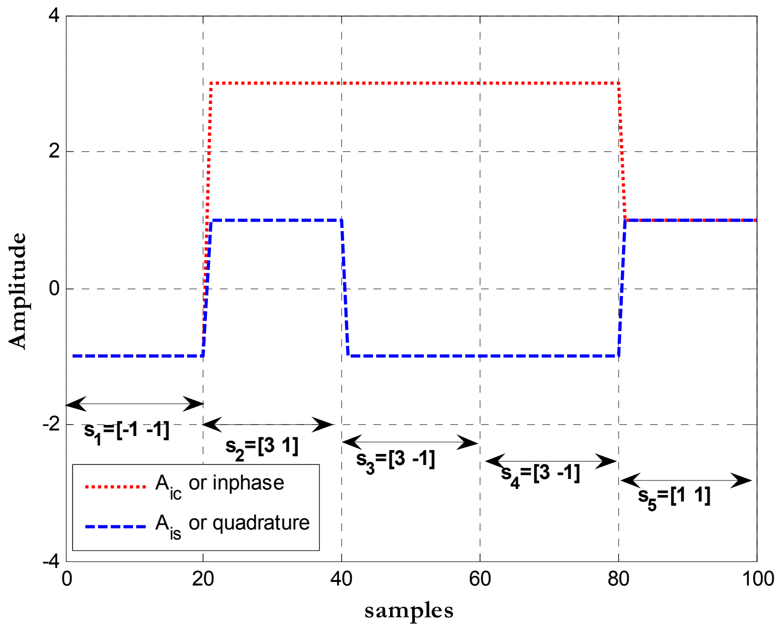


Figure 2.11: Up sampled bit mapper output for an 8-QAM modulator. The dotted line represents the in-phase component, while the dashed line the quadrature. For the duration of each (N=20 samples), the corresponding symbol is also noted (s_1 etc…).

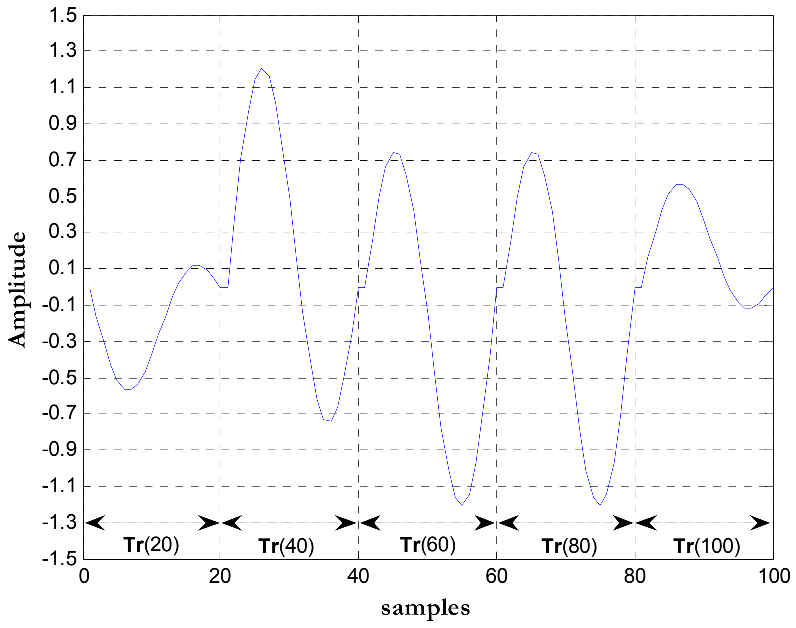


Figure 2.12: 8-QAM modulator output. The basis waveforms are modulated by the symbols of Fig 2.9. The output per time interval of N=20 samples, is also annotated for ease of comparison.

2.2.2.2 Energy Normalization

The previous section provided a compatible way to transmit the modulated data. Recall that the second restriction of this system is that the output must remain bounded at [-1 1] amplitudes. Levels above these values are literally chopped witch means immediate loss of information. Another problem that could occur, is when the peak amplitude levels are too low, for example [-0.2 0.2], resulting to a faint signal more acceptable to noise or other distortions. Both cases are dealt through means of waveform scaling, or energy normalization as will be made clear shortly.

A first example of a waveform that exceeds the maximum amplitude level constraint is that of Fig 2.12, higher QAM levels can have more severe loss at the *Line Out* port, as Fig 2.13 demonstrates.

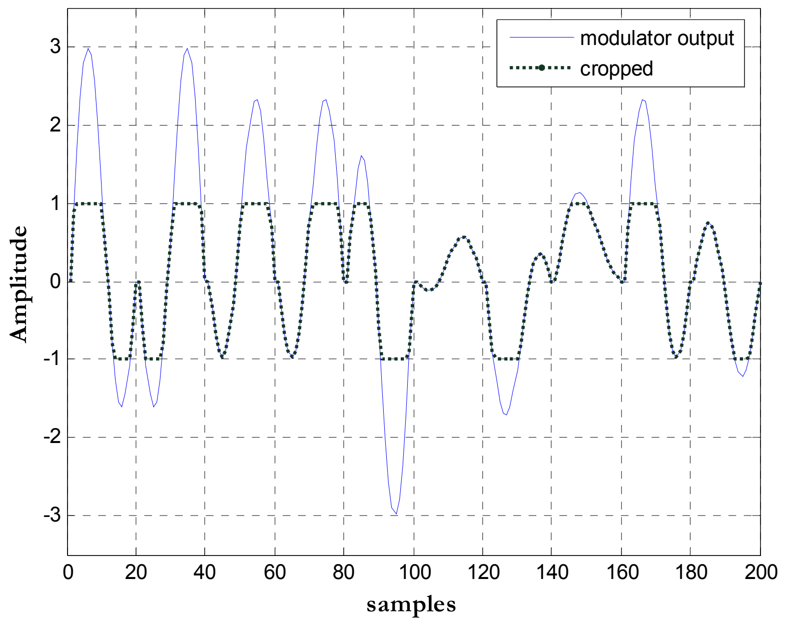


Figure 2.13: 32-QAM modulated signal, with sampling frequency 20 samples/ symbol, and the cropped version that occurs due to hardware limitations.

To circumvent this issue, the obvious solution is to scale the waveform amplitudes to acceptable levels. In addition, since the symbols of the orthogonal constellations must have equal minimum distances, the scaling should be uniform. Removing the last constrain could offer better performance but at a higher complexity to both transmitter and receiver. Considering the uniform case, the scaling factor is:

$$scale = \frac{\max\_\text{limit}}{|\,peak\_level\,|} = \frac{1}{|\,peak\_level\,|} \tag{2.11}$$

where *max_limit* the maximum output amplitude and *peak_level* the absolute maximum modulated signal amplitude. Due to symmetry peak positive and negative levels are equal in absolute value. Applying the scaling factor, the equivalent modulated and scaled signal (depicted in Fig 2.14) is:

$$\begin{aligned} scale.\mathbf{tr}_m &= [A_{mc}\mathbf{y}_1 + A_{ms}\mathbf{y}_2].scale,\ \ for\, m = 1,2...,N_S \\ &= (A_{mc}.scale)\mathbf{y}_1 + (A_{ms}.scale)\mathbf{y}_2 \end{aligned} \tag{2.12}$$

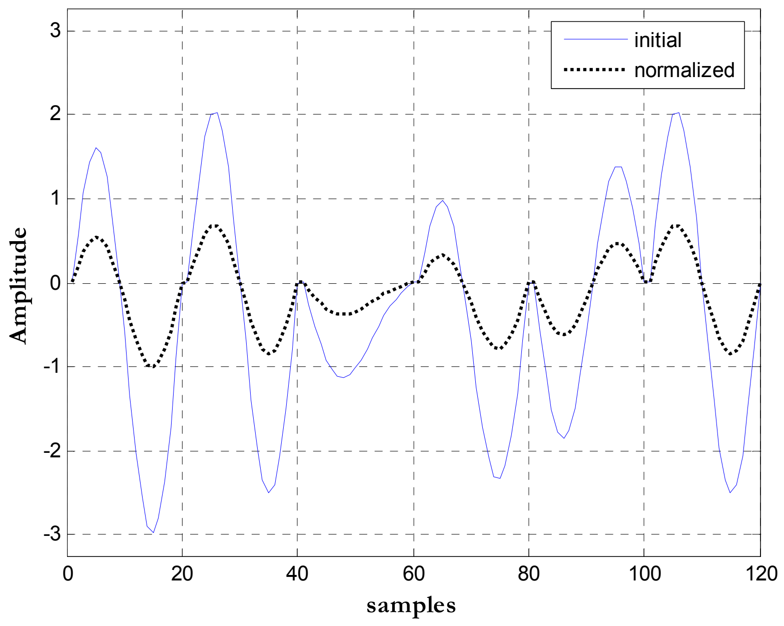


Figure 2.14: Normalization of a 32 QAM modulated signal.

Scaling can also be useful when the signal is weak. This case occurs for high sampling frequencies due to normalization side effects of the Gram – Schmidt method. Higher sampling frequencies downgrade the transfer rate performance, since they increase the transmission time, but never the less, can be useful due to the increased resolution of the transmitted symbols. Figure 2.15 represents this case for N=100 samples using 4 - QAM. The procedure is the exact but with the difference that the signal is in fact amplified.

The result (2.12) allows the scaling to be implemented along with the construction of the constellation, in other words in the *bit mapping* module. So a revised form of the equation (2.4) that will also consider the output normalization is:

$$
\begin{aligned}
&A_{m1c} = \{(2m_1 - 1 - L_1)d, m_1 = 1, 2..., L_1\} \\
&A_{m2s} = \{(2m_2 - 1 - L_2)d, m_2 = 1, 2..., L_2\} \\
&d : scaling\ factor
\end{aligned}
\tag{2.13}
$$

The average energy of the constellation is:

$$
E_{av} = \frac{1}{M}\sum_{m=1}^{M} \| ds_m \|^2 = \frac{d^2}{M}\sum_{m=1}^{M} \| s_m \|^2, m = 1, 2..., M \tag{2.14}
$$

Eventually, scaling results to symbol energy normalization, either increasing the transmitted energy, when d>1 or the opposite, when d<1. Further references to the effects of the energy normalization are left to the demodulation section of this chapter.

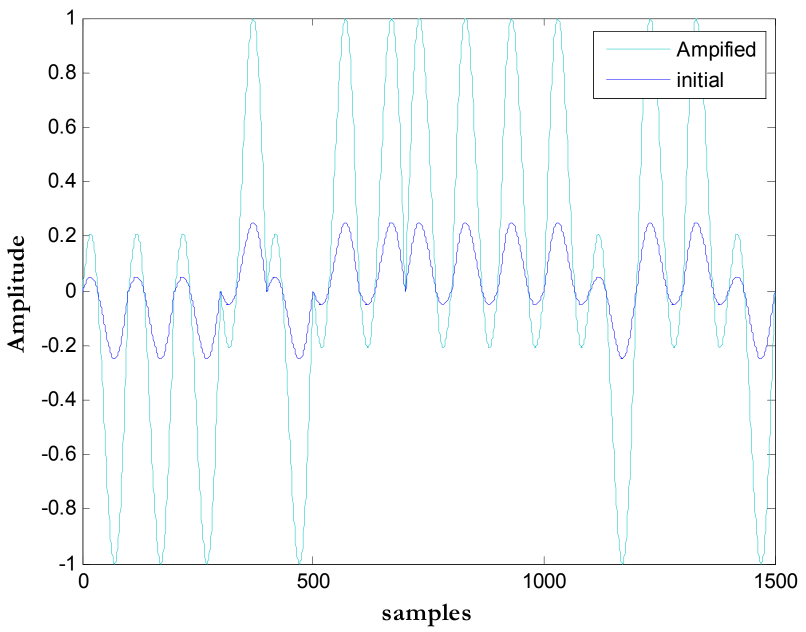


Figure 2.15: 4 QAM modulated and boosted signal.

2.2.2.3 Implementation

Currently the system supports 3 different ways of normalization:

- Average energy normalization:

  If the desired mean energy output is E_d then the scaling factor (*d*) is

$$d = \sqrt{\frac{\widehat{E}_d}{\widehat{E}_{av}}} \tag{2.15}$$

- Peak energy or peak power normalization:

  Defines the maximum symbol energy E_{peak}:

$$d = \sqrt{\frac{E_{peak}}{\arg\max_m \{\| s_m \|^2\}}} \tag{2.16}$$

- Automated:

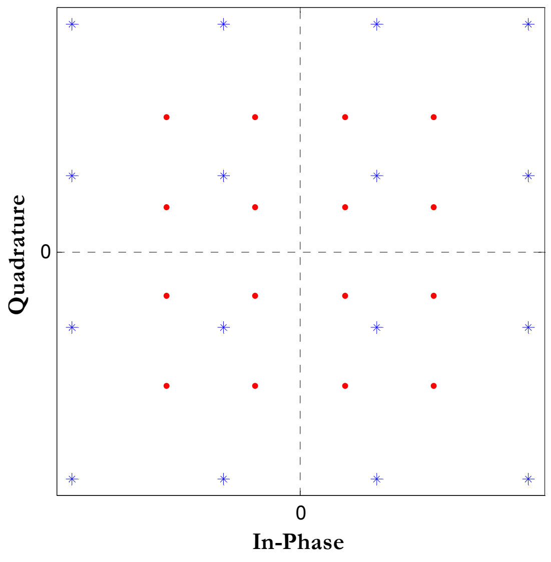
  The automated normalizing method implements the process discussed at the previous paragraphs and does not require user input. A necessary step is to calculate a priori the maximum possible amplitude of a generated modulated signal.

  Define the matrix **IQ** as $\mathbf{IQ} = \left[\mathbf{Ac\ As}\right]_{Mx2}$ and *peak_level* the peak amplitude of all possible modulated waveforms. Then:

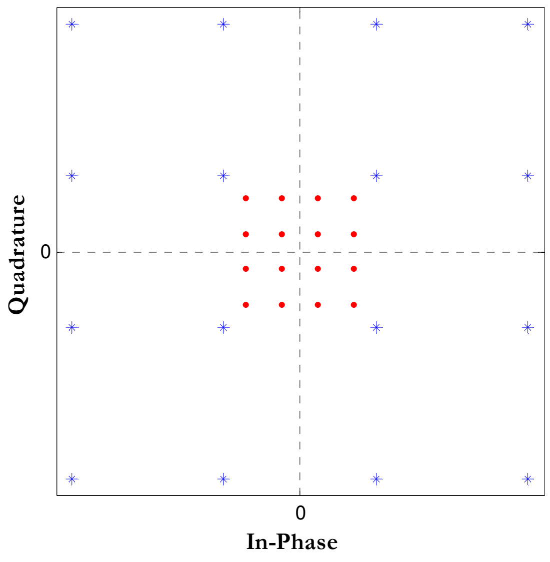
$$peak\_level = \max\left\{\mathbf{IQ}.\begin{bmatrix}\mathbf{y}_1 \\ \mathbf{y}_2\end{bmatrix}_{2xN}\right\} \tag{2.17}$$

  where M the QAM size , $\mathbf{y_1}$, $\mathbf{y_2}$ the basis functions and N the number of samples of the basis functions. It actually implements all possible occurrences of equation (1.8). The scaling factor then is d=1/*peak_level.* The resulting peak level depends on three factors, as can be observed from relation (1.17); the constellation or QAM size (M), the basis functions ($\mathbf{y_1}$, $\mathbf{y_2}$) and the sampling frequency (N) used to generate the discrete set of basis functions. Note that the third method optimizes mean and peak energy in to a maximum allowed limit as mentioned previously.

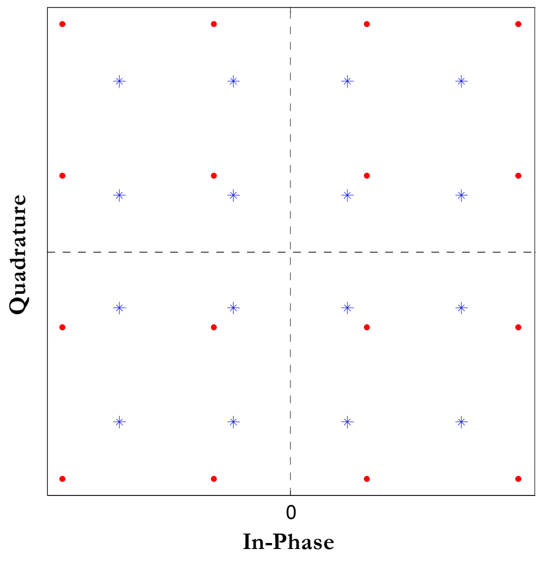
If the constellation matrix (1.6) is defined as **C,** then the scaling is straightforward as **Cn** = d***C**, where **Cn** the new elements matrix or lookup table. Figures 2.16 present the resulted constellations for each operation.



(a)



(b)



(c)

Figure 2.16: (a) Mean energy normalization to E_{av}=1. (b) Peak normalization to E_{peak}=1. (c) Automated normalization to maximum allowed peak / mean energy. The star like marker represents the default symbols of a 16 – QAM constellation and the dotted, the normalized.

2.3 QAM Demodulator

The demodulator is, as its name implies, the inverse process of the modulator. Prior this stage the sampled received signal goes through the FM receiver, the sound cards Line in port, frame synchronization and possibly, equalization. For the purposes of this section, it's assumed that the communication channel is simply AWGN, meaning that the previous stages are performing perfectly. It consists of two discrete parts or modules. These are, in order, the signal demodulator and the symbol detector. The demapper is implemented within the symbol detector and is the invert process of mapping thus it shall not be treated as a discrete process.

2.3.1 Signal Demodulator

2.3.1.1 Case 1: AWGN channel

The signal demodulator module, simply extracts the information from the carrier signal which for this first step is a symbol, as defined previously. To conform to the previously used notation, one instance (*x*) of the received signal *Rx* shall be:

$$\begin{aligned} &Rx(n) = Tx(n) + v(n),\ n = 1,2...N \ \mathit{and}\ N : \mathit{samples} / T \\ &v(.) : \mathit{white\ gaussian\ noise} \end{aligned} \tag{2.18}$$

For this stage, there are 2 implementations thought the literature, the correlation demodulator and the matched filter demodulator. Although they are equivalent, our implementation matches the design of the correlation demodulator *[5&10].*

Recall that the transmitted signal Rx is in fact a linearly weighed sum of the orthogonal basis functions $\mathbf{y}_1, \mathbf{y}_2$. In fact each instance of the received signal goes through a bank of cross-correlators that compute the projection of *Rx(n)* onto the basis functions, as illustrated in fig. 2.17. The result of each correlator is a noisy estimate of the in-phase and quadrature components (see Fig 2.19 below).

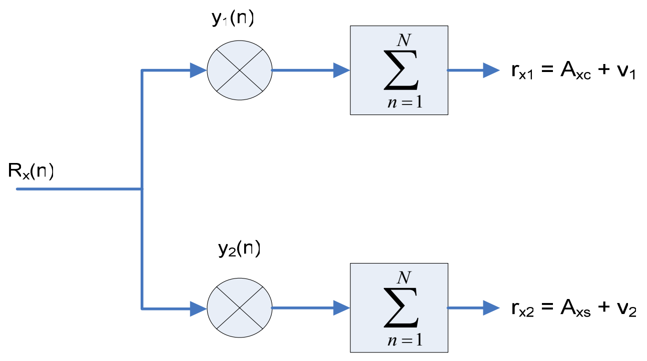


Figure 2.17: Correlation demodulator, where v_1, v_2 are the noise components.

2.3.1.2 Case 2: ISI

Previously, was assumed that the signal has already been equalized prior the correlation, but that is not always the case. In fact, due to speed problems of the LMS and RLS equalizers for large packets, i.e. 100KB, the correlators precede the equalization stage, therefore reducing substantially the execution and convergence timing requirements. Considering the channel (joint responses of sound card, FM transmitter, RF channel, FM receiver and sound card input) as linear FIR of L length then the received signal is:

$$\begin{aligned}&R_x(n) = T_x(n) * h(n) + v(n),\ n = 1,2...N\\&or\\&h:\{h_k\}_{k=0}^{L-1}\\&r_x = h_0 s_x + \sum_{k=1}^{L-1} h_k s_{x-k} + v_x, where\, r_x = r_{Ix} + jr_{Qx}\end{aligned} \tag{2.19}$$

The last relation is the output from the correlator where each component is a weighted sum of previous transmitted symbols –short termed *ISI*.

2.3.1.3 Implementation

Consider that the received signal is in fact a sequence of N_D noisy transmitted modulated waveforms:

$$R_x(n) = T_x(n) + v(n), for\, x = 1,2..., N_D\ and\ n = 1,2..., N \tag{2.20}$$

The process of correlation with a bank of basis functions that "extracts" the noisy components:

$$r_{xk} = s_{xk} + v_k = \sum_{n=1}^{N} y_k(n) R_x(n), \; k = 1,2 \tag{2.21}$$

Using vector notation:

$$\mathbf{R}_{NxN_D} = [R_1(n)\, R_2(n) \cdots R_{N_D}(n)] \tag{2.22}$$

$$\mathbf{r}_{1xN_D} = \mathbf{y}_1{}^T\mathbf{R} + j\mathbf{y}_2{}^T\mathbf{R} \tag{2.23}$$

So the resulting vector $\mathbf{r}^T$ is the complex representation of each received symbol. Figure 2.18 illustrates a part of a 4 QAM modulated signal, transmitted trough an AWGN channel and the correlator outputs. Note that the process is the same for the 2nd case of ISI presence, although the results will differ (Fig. 2.19) until ISI is removed.

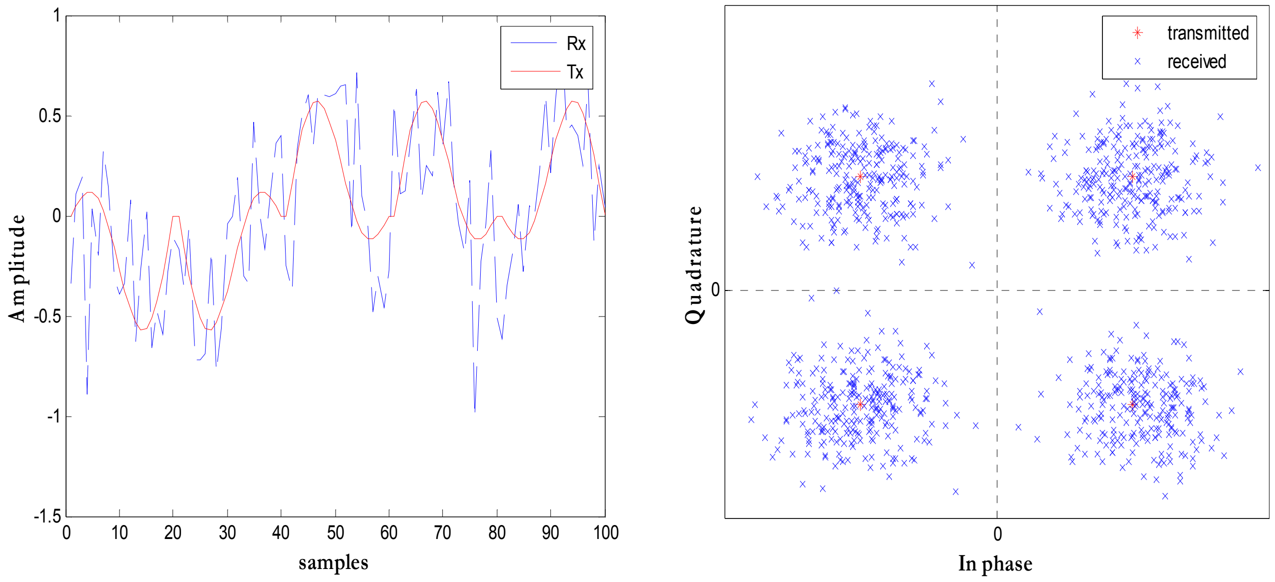


Figure 2.18: Received waveform and correlator output for simple AWGN channel.

2.3.2 Symbol Detector

Symbol detection sums up the process of estimating the transmitted symbol based on the observation vector **r** (eq. 2.23). For the case of simple AWGN channel or when the ISI is removed through equalization, therefore memoryless modulation signals, the optimum detector is the symbol by symbol detector. In the presence of memory the optimum detector is the Maximum Likelihood Sequence Estimator. The latter is examined in Chapter 3, under the scope of the *Viterbi* equalizer.

The decision rule is based upon finding the symbol that is closest in distance to the received vector **r,** referred to as minimum Euclidean distance detection. The distance metrics for any orthogonal modulation scheme are:

$$D(\mathbf{r},\mathbf{s}_m)=\sum_{k=1}^{2}(r_k-s_{mk})^2, m=1,2...,M \tag{2.24}$$

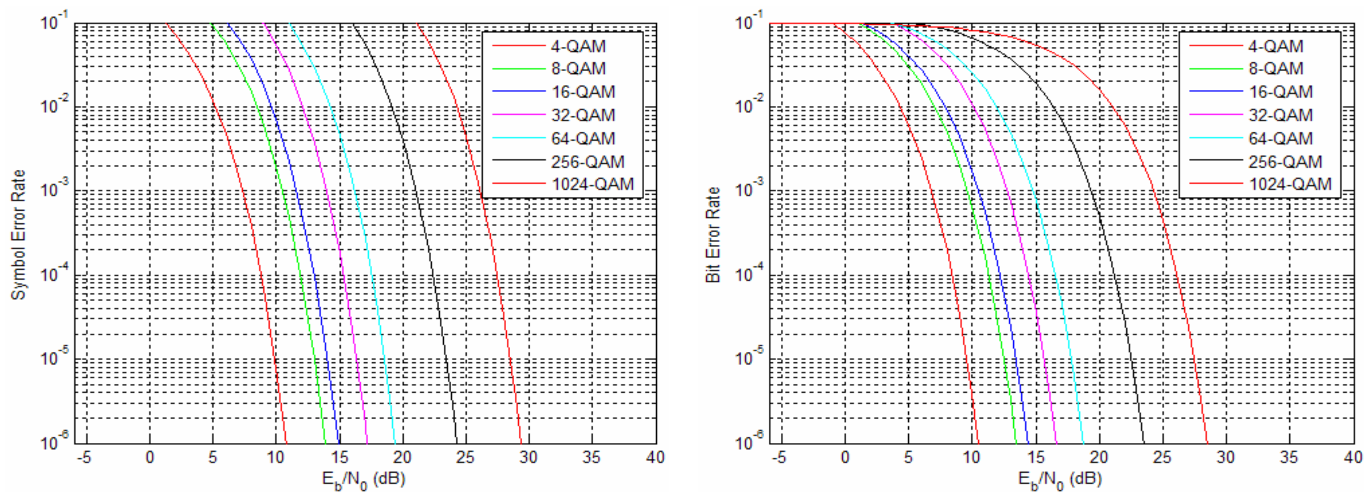
which divide the constellation into decision regions making the process of estimation quite straightforward.

The resulting error probability for m-ary QAM modulation schemes:

$$\begin{aligned} P_M &= 1-\left[1-2\left(1-\frac{1}{\sqrt{M}}\right)Q\left(\sqrt{\frac{3E_{av}}{(M-1)N_o}}\right)\right]^2, E_{av,bit}=kE_{av} \\ Q(x) &= \frac{1}{\sqrt{2\pi}}\int_x e^{-t^2/2}dt \end{aligned} \tag{2.25}$$

In the case of Gray coded modulation, assuming that only one bit error occurs per symbol, which is the most likely form of error, the probability of bit error:

$$P_{M,biterr}=\frac{1}{k}P_M \ , k=\log_2(M) \tag{2.26}$$



(a) (b)

Figure 2.19: Symbol (a) and bit (b) error rates of M-ary QAM versus bit SNR.

2.3.2.1 Implementation

The detector is implemented in block form, performing estimation per packets. Therefore, to be consistent with the previous stage, the input to the detector is a vector of all received symbols (**r**) for each transmitted packet. Then the metrics are calculated using the matrix(**S**) and the symbol which gives the minimum distance is considered the most probable.

$$\mathbf{r}^T = \begin{bmatrix} r_{1I} + jr_{1q} \\ \vdots \\ r_{xI} + jr_{xq} \\ \vdots \\ r_{N_DI} + jr_{N_Dq} \end{bmatrix}, \ \mathbf{S} = \begin{bmatrix} A_{1c} + jA_{1s} & \ldots & A_{Mc} + jA_{Ms} \\ A_{1c} + jA_{1s} & & A_{Mc} + jA_{Ms} \\ \vdots & \cdots & \vdots \\ A_{1c} + jA_{1s} & & A_{Mc} + jA_{Ms} \end{bmatrix}_{N_D x M}$$

where x denotes t the x^{th} transmitted symbol and **S** is formed from the set of all possible symbols {A_{mc}+jA_{ms}}, m=1,2…M.

Each element of the distance vector **D** of size N_DxM:

$$\mathbf{D} = \begin{bmatrix} (r_{1I} - A_{1c})^2 + (r_q^1 - A_{1s})^2 & \ldots & (r_{1I} - A_{Mc})^2 + (r_{1q} - A_{Ms})^2 \\ (r_{2I} - A_{1c})^2 + (r_{2q} - A_{1s})^2 & \ldots & (r_{2I} - A_{Mc})^2 + (r_{2q} - A_{Ms})^2 \\ & (r_{xI} - A_{1c})^2 + (r_{xq} - A_{ms})^2 & \\ (r_{N_DI} - A_{1c})^2 + (r_{N_Dq} - A_{1s})^2 & & (r_{N_DI} - A_{1c})^2 + (r_{N_Dq} - A_{1s})^2 \end{bmatrix}.$$

Finally the index of the minimum per row of the matrix **D** is retrieved and used as pointer to a modified lookup table (inverse lookup table (*LT*)), like a sorted version of the constellation matrix, to retrieve the corresponding bits.

$$\begin{aligned} &index_x = \{m : \arg\min_m (\mathbf{D}(x, m))\} \\ &binary\_word_x = inverse\_LT(index_x) \end{aligned} \tag{2.27}$$

3 Viterbi Equalization

3.1 Introduction

The second chapter presented the optimum symbol-by-symbol detector, under the scope of quadrature amplitude demodulation and memoryless signals. However, when the communication channel induces memory, meaning that consecutive symbols are interdependent, optimality is achieved when the decisions of the detector are based on observations of received symbols sequences, over multiple signaling intervals. Therefore, it is necessary to readapt the channel model, to include the intersymbol interference (ISI). Afterwards, we present briefly the maximum likelihood sequence detector and its implementation, the Viterbi algorithm in more depth. Finally, the derived Viterbi equalizer structure is presented.

3.2 CHANNEL MODEL

The channel generally is the medium through the information travels until it reaches its destination. In previous chapters, we assumed that it is simply AWGN. That is not entirely true, although for various modulation schemes can be considered as such since ISI effects are not severe. Recall, also, that for the purposes of this project, the WFTP prototype, the channel impulse response is the joint responses of each intermediate stage from the sound cards line out until the line in:

$$c_i = [f_{LO}(t) * f_{tr}(t) * f_c(t) * f_{re}(t) * f_{LI}(t)]_{t=iT} \qquad (3.1)$$

where T is the symbol interval.

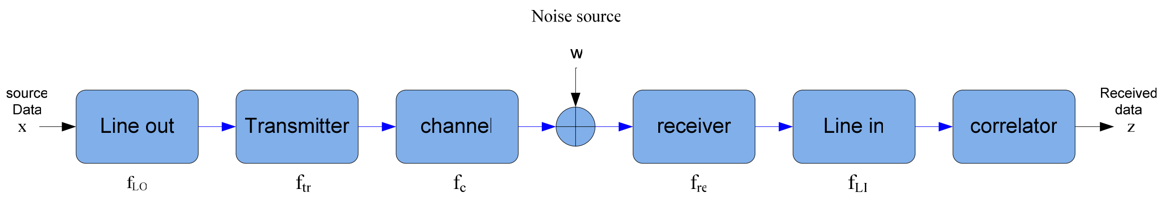


Figure 3.1: The data transmission system.

Let us consider as the discrete time channel the combined result of each stage until the correlators output. Then, the resulted system model is depicted below.

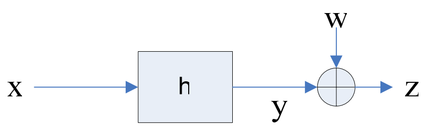


Figure 3.2: Discrete-time channel model.

From now on, we concentrate on the discrete time model of figure *3.2*, which satisfies the following assumptions:

- The channel h is linear.
- W is Gaussian with zero mean and σ^2 variance.
- Each source symbol x_i can take one of the m integer values 0, 1…, m-1 independently with equal probability, where m is the alphabets size.

Now we can define the notation used throughout this chapter. Let *h* be the channel impulse response of length L=*v*+*1,* where *v* is the channel memory length or equivalently the number of interfering symbols. Let *x* be the source symbol sequence of length *l.* Denote *z* and *w* as the received symbol sequence and noise sequence, respectively, each of length *l*+*v*-1. Also, define y as the transmitted symbol sequence. Besides, define $\hat{x}$ as the estimated symbol sequence and $\hat{y}$ the corresponding transmitted symbol sequence. By definition we have that *z=y* + *w* and *y=h*x.* Each transmitted symbol is then:

$$y_i = h_o + \sum_{j=1}^{v} h_j x_{i-j} \tag{3.2}$$

where the second term is the induced ISI.

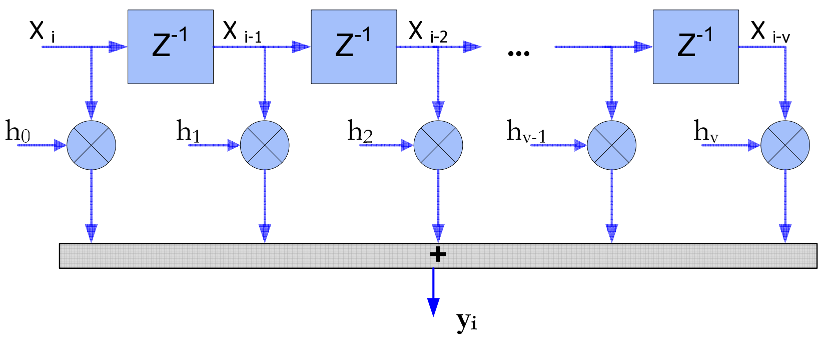


Figure 3.3: Finite state machine (FSM) model.

The above equation, allows us to describe the channel as a finite-state machine (*FSM*), as shown in figure *3.3*. Regarding the stored elements as the state of the FSM, each transmitted symbol can be treated as the output due to a specific state transition. Letting the state sequence at time i as $s = \{s_i\}$, where $s_i = (x_{i-1}, x_{i-2} ..., x_{i-v})$ then, the transmitted symbol $y_i = (s_i, s_{i+1})$ is solely determined by the state transition (s_i, s_{i+1}). Also, note that the state s_i depends on the previous state s_{i-1} and the current source symbol x_i that enters the FSM.

3.3 *MAXIMUM LIKELIHOOD SEQUENCE ESTIMATION*

In the receiver, only the received sequence z can be observed. Decision on which one of many permissible source sequences being transmitted is based on probabilistic argument. Denote ***X*** as the set of all possible source sequences. We want to maximize the a posteriori probability P(x|z) for all x in X. This maximum a posteriori (MAP) rule minimizes the error probability in detecting the whole sequence, and is thus optimum in this sense. A receiver detecting signals using the MAP rule is referred to as a MAP receiver.

Under the condition that all source sequences are equi-probable (i.e. the a priori probability P(x) is the same for all x in X), maximizing P(x|z) is equivalent to maximizing P(z|x). This is termed the maximum likelihood (ML) rule. A receiver detecting signals using the ML rule is referred to as a ML receiver or a MLSE. Note that the MLSE can be treated as a special case of the MAP receiver. Since the source sequence and the state sequence are one-to-one correspondent and the noise terms w$_i$ are independent, the log likelihood function

$$\ln P(z \mid x) = \ln P(z \mid s) = \sum_i \ln P(z_i \mid s_i, s_{i+1}) = \sum_i \ln P(z_i - y(s_i, s_{i+1})), \qquad (3.3)$$

where $y(s_i, s_{i+1})$ is the transmitted symbol corresponding to the state transition (s_i, s_{i+1}). As the noise components are independent and Gaussian, the joint probability density of the noise sequence w is

$$p(w) = \prod_i p(w_i) = \prod_i \left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{w_i^2}{2\sigma^2}}\right) = \mathrm{K} e^{-\frac{\sum_i w_i^2}{2\sigma^2}}, \qquad (3.4)$$

where K is a constant. Therefore, it is only needed to minimize the $\sum_i w_i^2 = \sum_i (z_i - y_i)^2$, which is the Euclidean distance between the received and the possible output of the FSM.

3.4 THE VITERBI ALGORITHM

A brute force approach to the problem of MLSE is to enumerate all valid source sequences. This would require m^ℓ calculations, each of ℓ squaring operations and $2\ell - 1$ additions. Thus, is unacceptable as the computational time increases exponentially with the sequence length ℓ.

Algorithms that are more efficient can derive through the representation of a channel as a finite state machine. Usual representation of FSM is with state diagrams. An alternate representation is with the *trellis* diagrams. For example, figures 1.4 (a) and (b) illustrates the state and trellis diagrams respectively of a binary PAM system with channel impulse response $h = [h_0\ h_1]^T = [1\ 0.5]$. Note that state s_i is defined as (x_{i-1}) and each transition (s_i, s_{i+1}) is associated with a weight $y(s_i, s_{i+1}) = h_0 x_i + h_1 x_{i-1} = x_i + 0.5 x_{i-1}$.

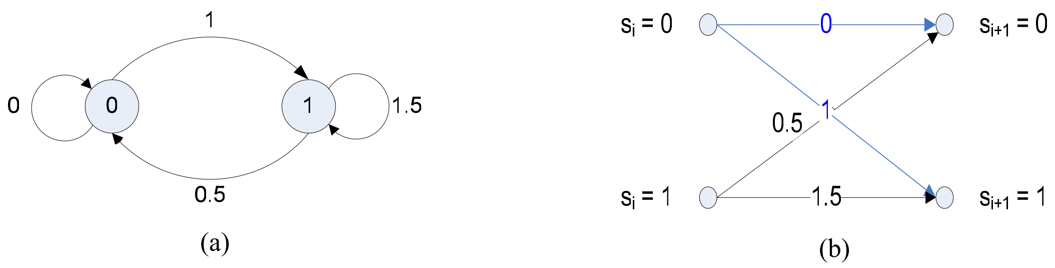


Figure 3.4: (a) State diagram of the channel h= (1, 0.5) for binary transmission. (b) One stage of the corresponding trellis diagram.

Since the trellis can be regarded as a graph, from now on, a state is called a *node*, a transition from predecessor node to its successor, or generally between nodes, is called a *branch* and a state sequence is called a *path*. The weight associated with each branch is termed *branch metric* and the accumulated weight associated with a path is termed as *path metric*.

The criterion in MLSE is minimized to the calculation of the Euclidean distance $|z_i - y(s_i, s_{i+1})|^2$ thus, is defined as the branch metric for each branch (s_i, s_{i+1}) of the trellis. We can find the shortest path, the one with the minimum path metric, by computing the branch metrics stage-by-stage. Each node has m incoming branches, except a few stages in the beginning or the end of the trellis diagram, due to the advent of a new source symbol in the FSM. Of the m incoming branches, only the one connected with the minimum partial path metric is retained, forming the survivor path. Partial paths associated with the other m-1 incoming branches are discarded since the shortest path must contain the survivor path if it goes through that node. Therefore, the number of survivor paths is the same as the number of nodes at each stage, m^v where v is the channels memory length. After all stages of the trellis have gone through, the resulting shortest path corresponds to the maximum likelihood sequence.

The details of how the VA calculates the shortest path are best demonstrated with an example. Consider a binary transmission with channel impulse response h^T=[1 0.8 0.3] of v=2 and m=2. The number of possible states is m^v = 4, with each state being a sequence of length v of source symbol combinations, S= {00, 01, 10, 11}. The state diagram illustrates the valid transitions, driven by the current source symbol and the associated weights (x_i / y_i). For the trellis diagram, assume that the observed sequence at the receiver is z = [0.8, 0.5, 1.2, 0.6]. The starting state is assumed to be the s_0 = 00 or that $x_{-1} = 0, x_{-2} = 0$. The weight of each branch is the branch metric and the path metric is shown inside each node. The solid lines represent the survivors and the bold lines the resulting shortest path.

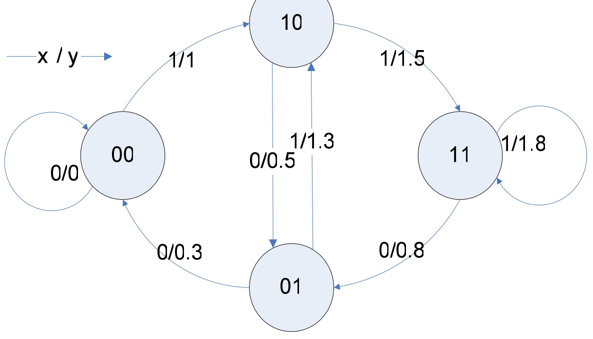


Figure 3.5: State transition diagram for binary transmission x_m:{0,1}, for m = 0,1 and channel impulse response h^T=[1 0.8 0.3]

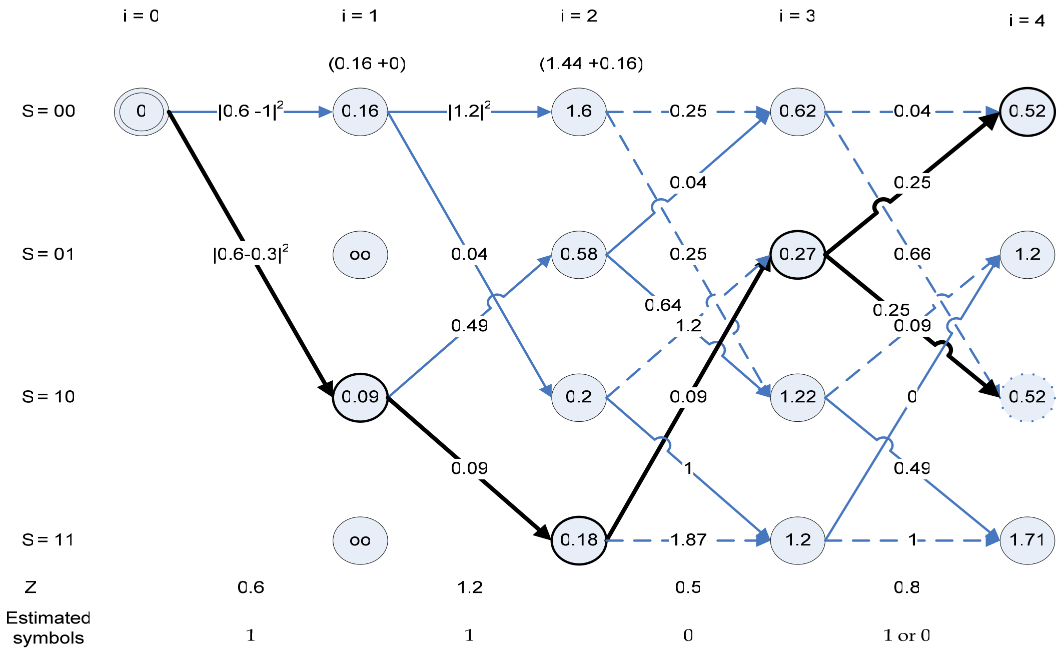


Figure 3.6: VA algorithm for channel impulse response $h^T=[1\ 0.8\ 0.3]$ **and received sequence** $z = [0.8, 0.5, 1.2, 0.6]$.

The process calculating the path metrics stage-by-stage is the forward evolution of the algorithm. Having reached an end, starts the back propagation where the optimum path and accordingly the most probable, in the ML sense, sequence is estimated. Even though the trellis diagram could be calculated using trees; simpler but more efficient structures avoid the costs of storing pointers and can be implemented much easier. For the purposes of this chapter, we shall analyze two main structures / matrices used to store only required information.

The first is a container for the path metrics. Notice, though, that at each stage only the metrics of the previous are required. Therefore, denote the path costs matrix $\mathbf{PC}_{m^v x2}$. The first column stores the old metrics while the second the new. At the next stage, the most recent calculated become the old etc..., until the end where only the new values are needed. The **PC** is indexed through the states in some proper form (binary or decimal representation), for example, using the example of figure 3.6 for the stages $i, i+1$ $c_{old} = \mathbf{PC}(s_i^{(00)} = 00,1)$ and regarding as survivor the 00 state, the new value is stored $\mathbf{PC}(s_{i+1}^{(00)} = 00,2) = c_{new}$.

Notice that we need to know during the back propagation (backward stage) the sequences of nodes that lead to the shortest node or equivalently the nodes of the shortest path. These nodes are of course the survivors at each stage. Therefore, denote the survivor matrix or map, $\mathbf{M}_{m^v x(l-1)}$ indexed the same way as the costs matrix. Each element is the survivor node s_{i-1} at stage i for node s_i. For example, from figure 3.6 the surviving node of state $s_2^{(00)}$ is $s_1^{(00)}$ whilst for $s_3^{(00)}$ is $s_2^{(01)}$ and the matrix is updated as M(00,2)=00 , M(00,3)=01 correspondingly. Thus, each row consists of surviving nodes that point to states 00, 01, 10 and 11. Both matrices are depicted in figure 3.7 with values corresponding to the trellis of figure 3.6.

PC (costs matrix)

Si	OLD	NEW
00	0.62	0.52
01	0.27	1.2
10	1.22	0.52
11	1.2	1.71
	1	2

Map

Si	i=1	i=2	i=3
00	00	10	01
01	10	11	11
10	00	01	01
11	10	10	10

Figure 3.7: *PC* and *Map* structures at the final stage of the VA of figure 3.6.

The first step of back propagation is to select the node with the minimum path metric. Afterwards the map is accessed in order to derive the optimum path:

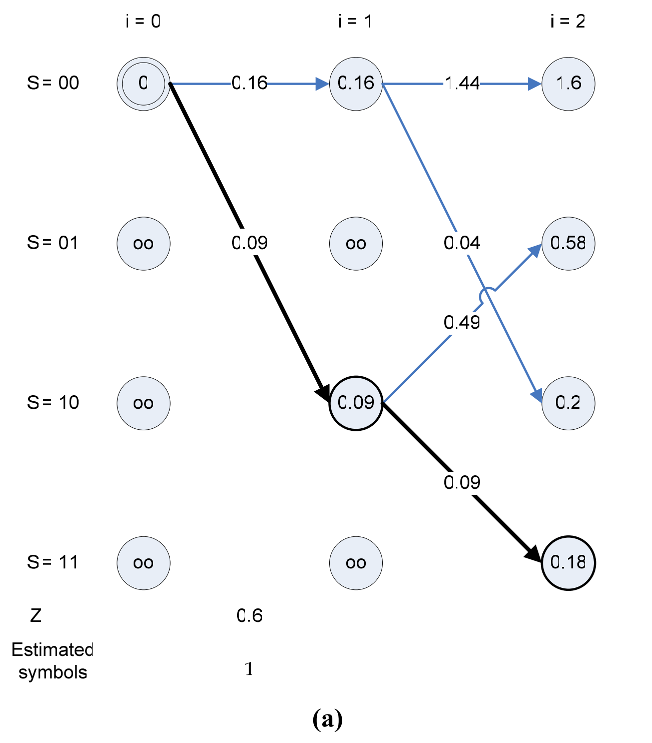
$$\begin{aligned}
S_{i,\min} &= \min\{PC(:,NEW)\} \\
S_{i-1,\min} &= M(S_{i,\min}, i-1) \\
S_{i-2,\min} &= M(S_{i-1,\min}, i-2) \qquad (3.5) \\
&\ldots \\
S_{1,\min} &= M(S_{2,\min}, 1)
\end{aligned}$$

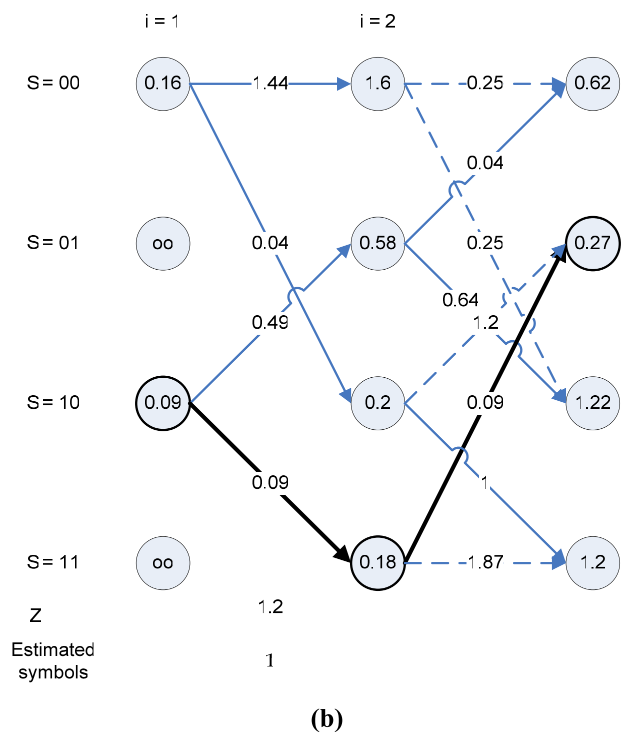
The initial state is anyway known so it does not need to be stored or retrieved. For our example, there are two possible estimated sequences of transmitted source symbols $\hat{x} = (1,1,0,1)$ *or* $(1,1,0,0)$. In general, when the path metrics are equal, the decision is taken in random.

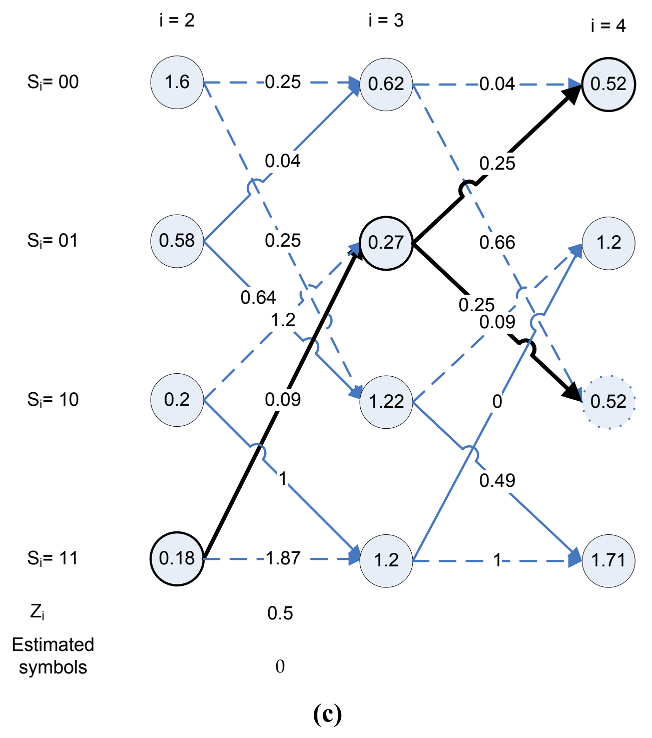
Let us look at the algorithms complexity. In general, there are m^v states per stage of totally ℓ stages. Assuming that the branch metrics are precomputed then one multiplication, one addition and m-1 binary comparisons are needed for each node. Thus, the time complexity of the VA is $O(m^{v+1})$ operations per detected symbol but it is increasing linearly with ℓ rather than exponentially. Its main disadvantage though is the huge storage it requires. Each survivor path requires $\ell v \log_2(m)$ bits and each metric p bits resulting in a space complexity of $m^v(\ell v \log_2(m) + 2p)$ bits. For large channel memory or symbol alphabet, result to prohibitive memory requirements. Besides, the incurred detection delay of the entire sequence could be undesirable. Therefore, some modifications must be made in order to make VA more practical.

A first approach is to reduce the trellis search depth to a manageable one; δ, called the truncation depth. Applying the truncation depth, a decision on input symbols $x_{i-\delta}$ are made at time i. After the decision path history before or at that time are discarded and the next stage is computed. Such modified algorithm is called the truncated VA or TVA. Note that the time complexity is not affected. The exact performance degradation due to the truncation is analytically intractable and is normally found by experimentation. However if δ is large enough the performance loss is negligible. Figure 3.8 shows the TVA with truncation depth Notice that the node associated with the shortest survivor is released and all path history at or before time i is retained without recalculation. In this way only the path information involving the last stage is computed.

In all examples, the initial state was arbitrarily chosen to be the state 00. This usually is not the case as the data segment follows a training sequence. Therefore, to create a known past and to "lead", at the same time, the first steps of the VA we inject a sequence into the data stream to be used as a *preamble*, known also to the algorithm, avoiding erroneous start-ups that could propagate. In the same manner, we also use a terminating sequence or *postamble.* Usually for the sake of simplicity, the length of both sequences is the same, ranging from 4 to 10 symbols maximum.







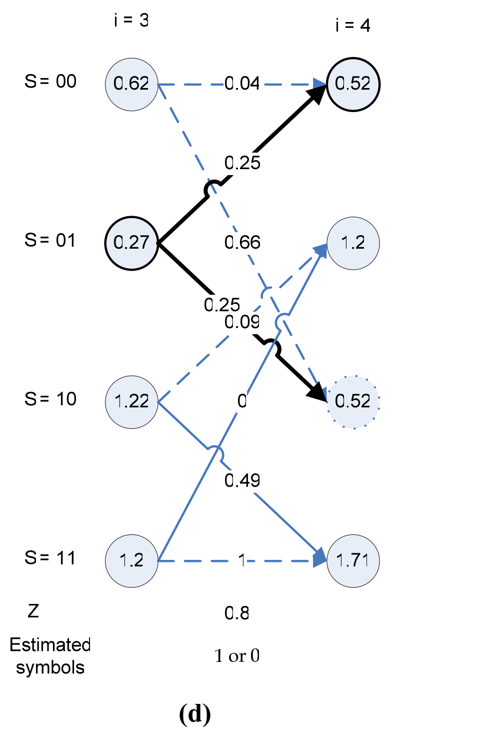


Figure 3.8: The truncated Viterbi Algorithm with truncation depth of 2 is applied to the example of figure 3.6, the estimated symbols are: (a) $\hat{x}_0 = 1$, (b) $\hat{x}_1 = 1$, (c) $\hat{x}_2 = 0$, (d) $\hat{x}_3 = 0$.

3.5 Viterbi Equalizer

The Viterbi equalizer is in fact the combination of a channel estimator unit and the Viterbi algorithm usually the truncated implementation depicted in figure 3.9.

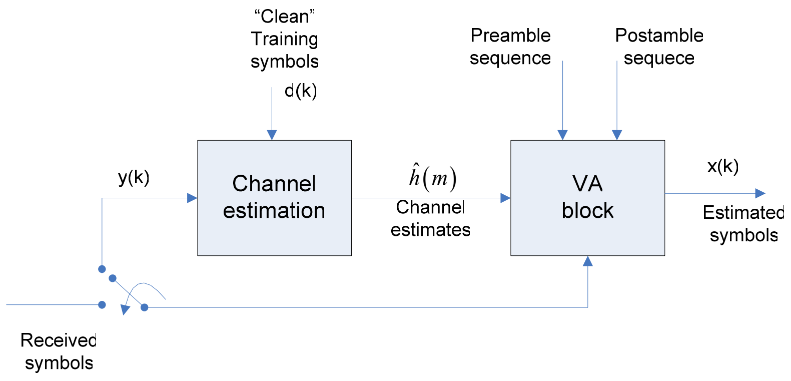


Figure 3.9: Viterbi Equalizer block diagram

In wireless applications the channel is time varying so the channel estimator is an adaptive algorithm such as RLS or LMS. The estimation is based on the training sequence ahead of the data segment performed at a symbol rate meaning that instead of operating on the received sampled training signal, it uses the symbol outputs from the correlator. The accuracy of the estimated channel taps, feedbacked to the VA detector unit, depends on the length of the training data, and the channel length. Another important issue is the speed the channel changes, since the estimation will not be valid for all the data sequence.

However, due to that the receiver operates in a block basis the channel estimation is performed though implementing direct LS estimation [10]. The channel coefficients are provided by the solutions of the Least Squares equations implemented in block form:

$$\underline{\mathbf{y}} = [y_1\, y_2 \ldots y_N]^T \textit{ received symbols vector}$$

$$\underline{\mathbf{d}} = [d_1, d_2 \ldots d_N]^T \textit{ training symbols vector}$$

Then the least squares solution of L taps and channel estimation is:

$$\hat{\underline{\mathbf{h}}}_{LS} = (\mathbf{A}^H \mathbf{A})^{\dagger} \mathbf{A}^H \underline{\mathbf{y}} \tag{3.6}$$

where **A** is the toeplitz matrix of the training symbols:

$$\mathbf{A}_{LxN+L-1} = \begin{bmatrix} d_L & \dots & d_1 \\ \vdots & \dots & \vdots \\ d_1 & \dots & d_{N+L-1} \end{bmatrix}.$$

The indexes H, $\dagger$ denote the Hermitian and Penrose Moore pseudo inverse operations respectively.

3.6 PERFORMANCE EVALUATION

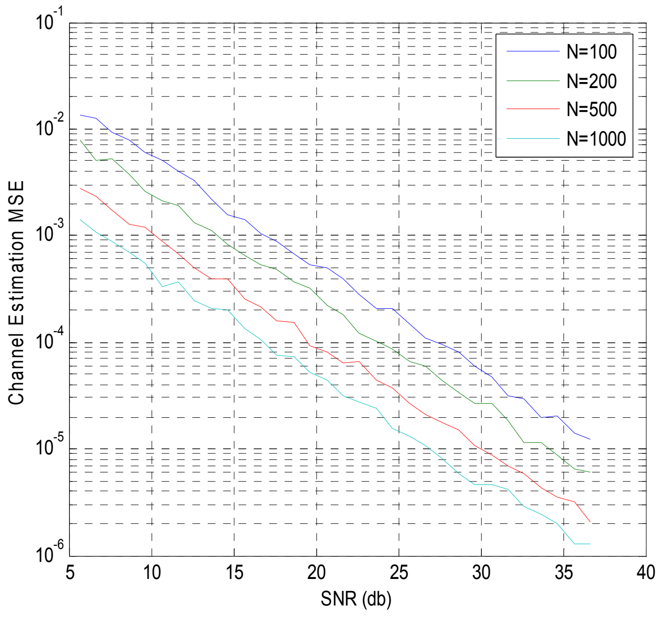
The scope of this section is to evaluate parameters or algorithms through simulation or experimentation. The goal is to either eliminate or limit the range of values of involved parameters as well as to identify possible dependencies and finally to estimate the impact each unit causes when applied to the system. For this part, the inspection shall begin from the channel estimation units and their timing requirements and afterwards the Viterbi Equalizer is evaluated.

3.6.1 Channel Estimation

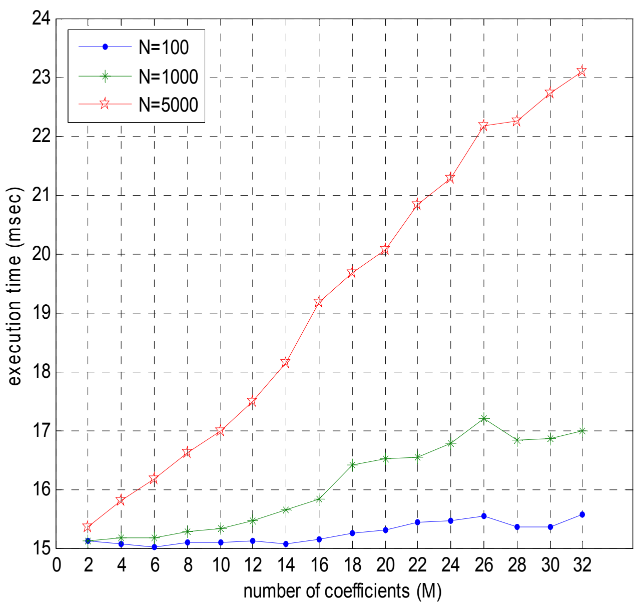
The first part evaluates both estimators for both channel and sequence lengths versus SNR though Monte Carlo simulation. The channel is generated randomly while the additive noise is white Gaussian. The performance metric is the mean square error of the estimated channel impulse response ($\hat{\underline{h}}$) and the original applied to the training data ($\underline{c}$):

$$chan\_mse = \frac{\| \underline{c} - \hat{\underline{h}} \|_2^2}{L} \quad \begin{array}{l} \hat{\underline{h}}_{Mx1} : estimated\ channel \\ \underline{c}_{Mx1} : channel \end{array} \tag{3.7}$$

The second metric comprises of the execution time for various channel and training sequence lengths. The simulation results are illustrated in figures 3.10 (a) and (b) respectively.



(a)



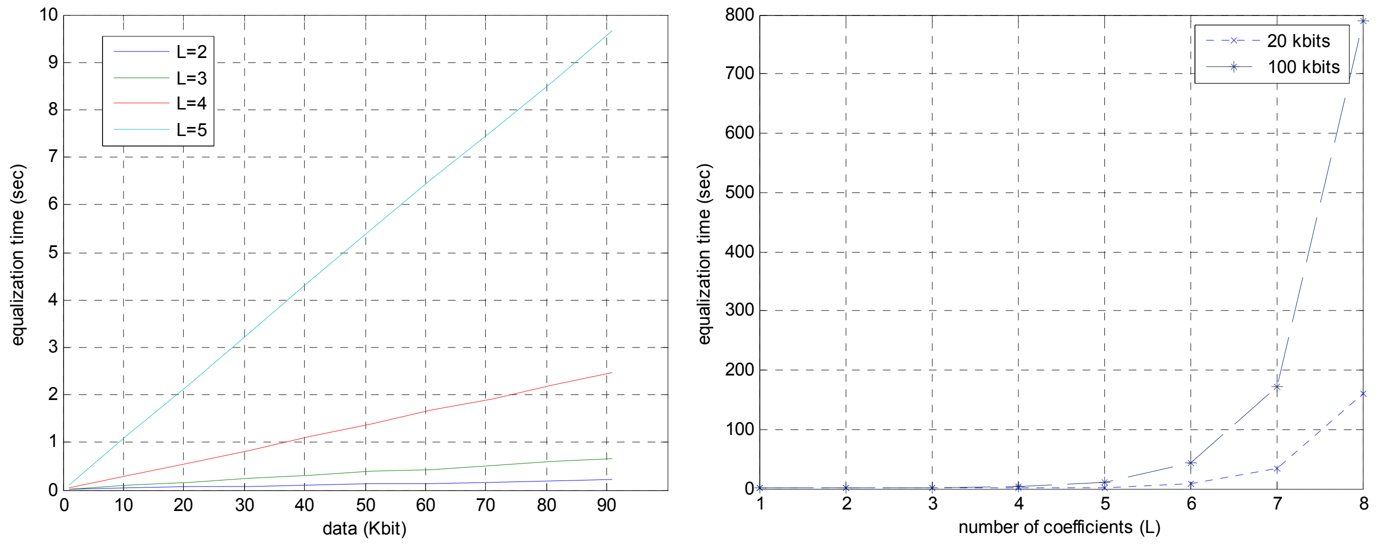
(b)

Figure 3.10: (a) Mean square error of the channel and the estimated versus the SNR for channel length L=10. (b) Timings of LS algorithm for channel lengths up to 32 taps and sequences of 100, 1000 and 5000 4-QAM modulated symbols.

The simulation proves that the estimation is dependent of the training sequence length, as always suspected; however, what remains to be seen is the equalization performance of the VA using the estimated channel response feedback from the LS unit. Finally, figure (b) shows that the delay the estimator adds to the receiver is quite small, meaning that large training sequences will not cause noticeable overall performance degradation due to processing delay of the LS estimator.

3.6.2 Viterbi Equalizer

The Viterbi equalizer is implemented using the TVA referenced previously due to memory limitations. Therefore, the parameters under consideration are the traceback length (*Tblen*), channel impulse response length (*L*) and the size of the training sequence (*N*). The quality performance metric is the symbol error rate while the speed performance is, as always, given by the execution time. Due to its time complexity though, it is prudent to analyze the time delay prior the error rate for the latter results to be of any practical usage. Figures 3.11(a), (b) translate the theoretic complexity of VA to timing requirements relative to the application platform (Matlab). Notice that the alphabet is consisted of a 4-QAM constellation, thus *m=4*, while the sequence's length is measured in bits.



(a) **(b)**

Figure 3.11: (a) Time delay with respect to sequence length (in kbits) and channel length (L), for m=4 (4-QAM) and traceback length (Tblen) 20 stages. (b) Time delay versus channel length (L).

Figure 3.11(a) illustrates the TVA's time complexity as time delay versus the data size and the estimated channel's length, while 3.11(b) emphasizes the exponential delay impact of channels with length L > 5. To keep the analysis simple we use as common ground the resulting total number of nodes-states per stage, affected by both estimated channel impulse response length (L) and the size of the alphabet (m), which is a power of 2. Thus the total nodes per stage expressed likewise are:

$$N_{nodes} = m^{L-1} = 2^{v\log 2(m)} \qquad (3.8)$$

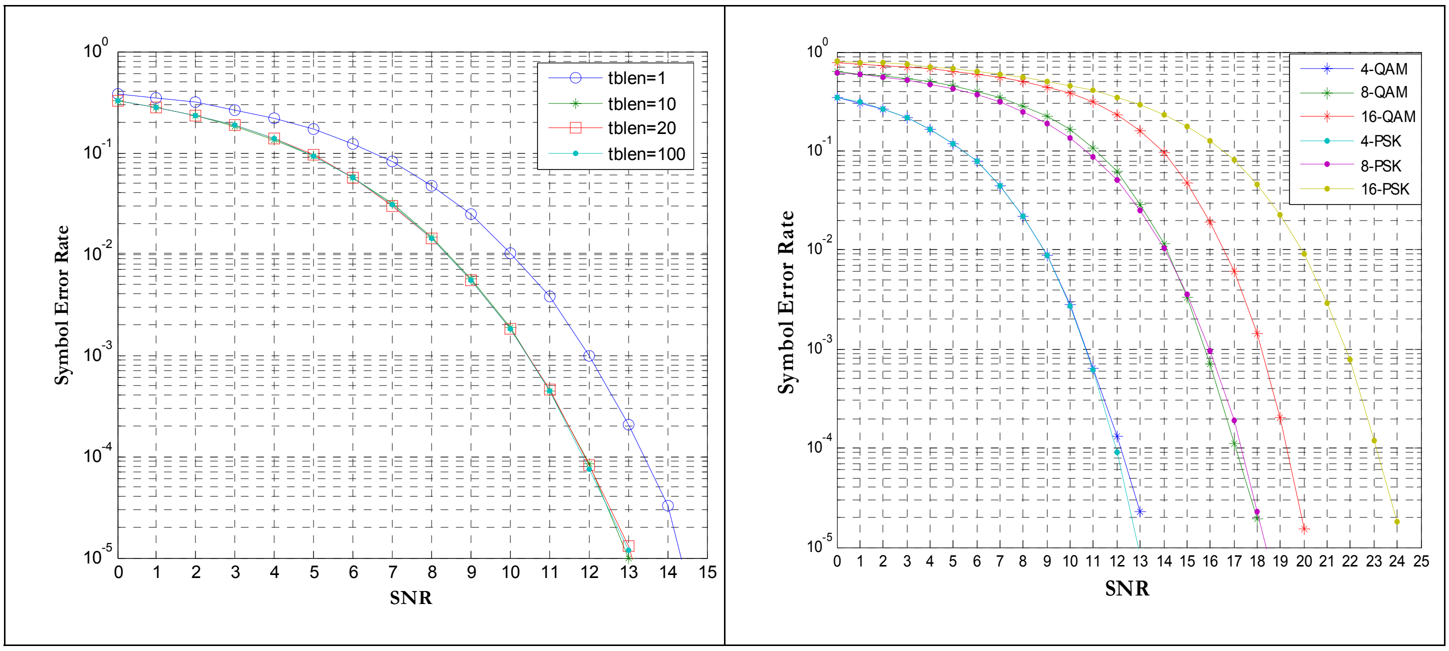
Figure 3.11(a) confirms the linear relation of VA and input sequences length and provides the time needed for the add-compare-select and memory access operations per node, which is approximately 1μsec with the current hardware. The choice of trace back length theoretically does not reduce the total time of equalization of a sequence but in a pipelined system limits the delay per symbol estimation to $Tb_{len}N_{nodes}10^{-6}$ seconds. Unfortunately, the receiver structure cannot operate in such continuous mode thus we sustain the full delay of sequence equalization. Practically though, it limits the memory usage so it can be entirely performed in the much faster physical memory (RAM), limiting the virtual memory access, providing a great improvement over large sequences.

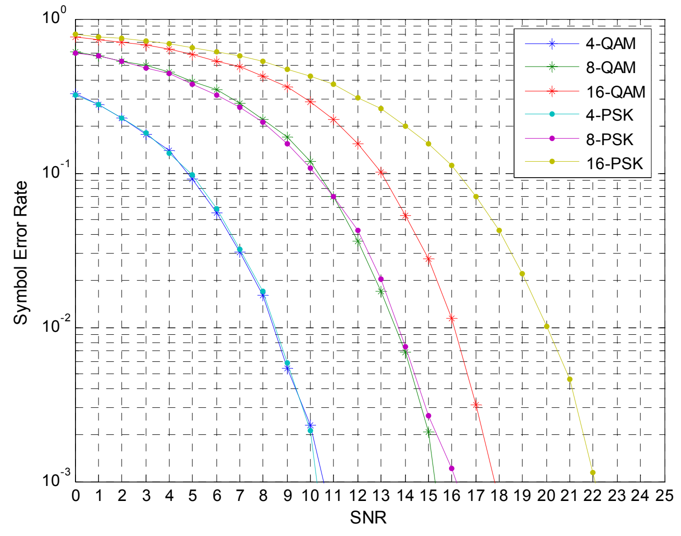
Figure 3.11(b) clearly states that the incurred delay for $N_{nodes} > 2^{12}$ is unacceptable and prohibits the use of Viterbi Equalization if the resulting states per stage exceed that limit. Also, note that the maximum is 2^{16} nodes. Any combination of ISI length and alphabet size beyond that limit completely depletes the memory needed to store just each stage or requires more than 2GB of either physical or virtual space. Therefore, we set a strict upper bound of the possible number of nodes per stage to

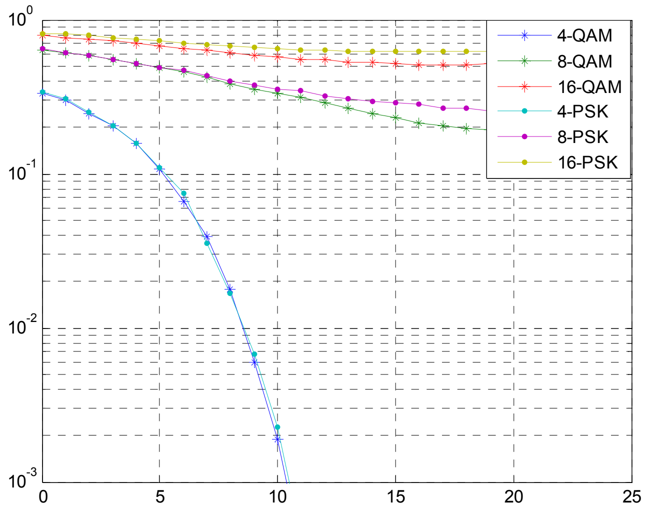
$$\max N_{nodes} = 2^{12} \tag{3.9}$$

To overcome, we can either select modulation schemes with small alphabets or /and truncate the estimated channel to an acceptable length when possible.

Having defined a set of parameters to allow a realizable application of the TVA, we can now proceed with evaluation of the equalizer. Implicated parameters are the estimated channel, the length of ISI, the traceback length as well as the modulation scheme.




Symbol Error Rate
SNR
10^0
10^-1
10^-2
10^-3
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
4-QAM
8-QAM
16-QAM
4-PSK
8-PSK
16-PSK


10^0
10^-1
10^-2
10^-3
0
5
10
15
20
25
4-QAM
8-QAM
16-QAM
4-PSK
8-PSK
16-PSK

4 Synchronization

4.1 Introduction

In a wireless communication system, there is the problem of generally detecting a signal and some essential characteristics for reliable data transmission. Synchronization techniques can be used for clock recovery, which attempts to synchronize the receiver clock with the transmitter’s symbol rate clock, and for “waking up” the receiver, causing an interrupt so that reception can be initiated. Specifically, for this wireless prototype, transmission is achieved through playback, while reception, through recording; therefore, there are unpredictable delays in both operations due to the nature of these processes (memory management, threading etc…). The following paragraphs present two techniques, implemented in this prototype to trace the incoming activity inside a hostile environment and control the aftermath of delay.

4.2 Framing – Packet detection

Clock recovery, in a broader sense, helps to detect the recorded sequence –packet inside a larger sampled signal of unknown delay, by searching for a known sequence, a specific pattern. The source of the delay is of non-importance. For this purpose, each data stream is injected with a sounding sequence, followed by actual data. Since it is adding a frame of symbols, the process is also referred to as framing. The receiver scans for this sequence by performing an autocorrelation and the sampling point that the maximum correlation occurs is deemed the correct, from which the modulated data follow (fig. 3.1). This header is also used for training channel estimators or equalizers, so for the rest of the chapter it will be referred to as the training sequence. For the same reason, also, the training sequence is a fixed number of bits of multiple time slots, randomly chosen in order to produce a strong auto correlated peak and provide enough training information. It is important to note that no assumptions are made regarding the channel meaning that it should perform regardless of the distortion; it is caused either by simple additive noise or due to channel memory (ISI) as well. Figure 3.1 illustrates a packet –the transmitted sequence, inside a larger one, the recorded. Being able to identify where the training frame starts or ends, is the key to retrieving the transmitted packet and operating on it. The margins prior and after the packet are caused by early and prolonged recording respectively, of tsafe seconds combined. Therefore, the recorded sequence has a length of packet transmission time plus a safety margin of t_{safe} seconds.

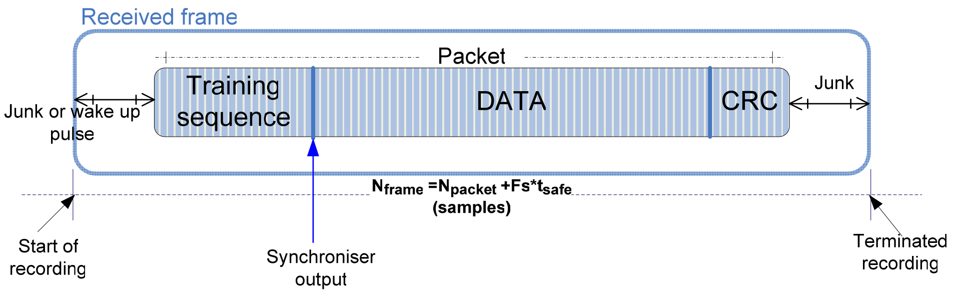


Figure 4.1: The packet at the receiver.

Let us make some definitions witch will be used throughout this chapter. The packet prior to transmission is measured in bits while after modulation, in samples or in symbols. It consists of three fields, the training sequence witch, as mentioned before, is concatenated with the data stream, the data stream and finally the CRC, that is used for error checking and is standard 16 bits wide. The notation for each field's size in bits is $B_{train}, B_{data}, B_{crc}$ respectively. The sampling and transmission frequencies are F_S, F_T but quite often are equal, so only F_S is used implying this equality, and the symbol frequency F_{symbol} as samples per modulated symbol. Therefore if M is the modulations size, then the size of a packet in samples is:

$$N_{packet} = \frac{B_{packet}}{\log_2(M)} F_{symbol} = \frac{B_{train} + B_{data} + B_{crc}}{\log_2(M)} F_{symbol} \tag{4.1}$$

The excess samples of the received frame are:

$$N_{excess} = F_S t_{safe} \tag{4.2}$$

Thus, a frame is consisted of $N_{frame} = N_{packets} + N_{excess}$ samples.

Having defined some crucial parameters lets see how the position of the packet can be determined. Correlation calculates the similarity of two sequences, possibly shifted in time, thus it is a simple form of pattern matching. Because the known sequence shall be compared with a larger sequence, cross correlation will be used. Cross-correlation in discrete time is a function of the time shift (lag) between two sequences that for our case are the known training sequence (d) and the received (r_x):

$$R_{du}(j) = \frac{1}{T}\sum_{k=0}^{N_{frame}} d[k]u[k+j] \tag{4.3}$$

The cross correlation function is maximized when the most similar sequence is found. Recall that matched filtering also produces similar results thus can also be used, providing as filter the known training sequence but flipped:

$$r[n] = u[n] * d[N_{train} - n] \tag{4.4}$$

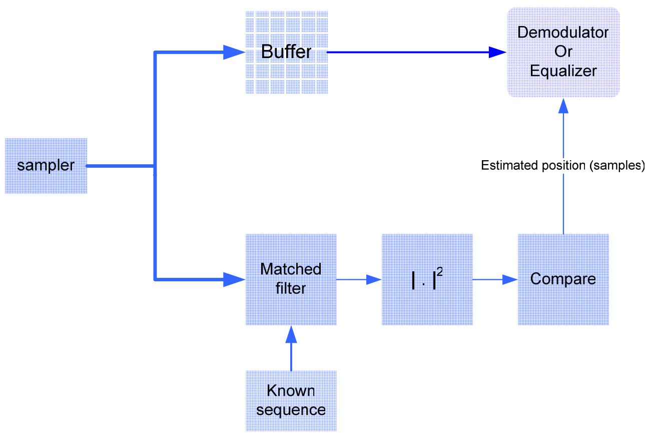


Figure 4.2: Proposed synchronizer.

In our implementation, matched filtering is used and the output is squared. The index of the maximum outcome is considered the end of the training sequence. This is simply the position of the training sequence inside the packet increased by a delay due to factors previously mentioned. Figure 3.3 illustrates the synchronizer of figure 3.2 in action. Knowing the length of the training sequence, it is a simple process to extract the entire packet from the received frame.

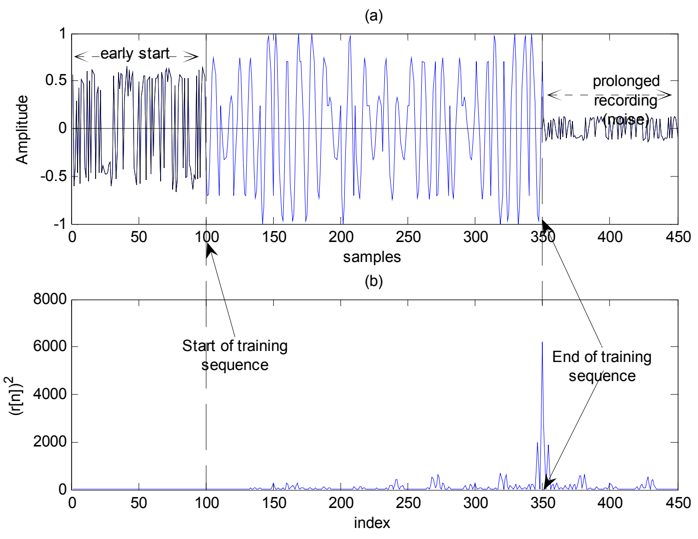


Figure 4.3: Sequence detection using matched filtering.

4.3 Handshaking

In most communication systems, such as TCP/IP, in order to initiate a data transfer, some kind of negotiation must take place, so that the transmitter will know the receiver's status and contrarily. The process of initiation a conversation among the host and a client is referred to as handshaking. In our WFTP prototype, the transmitter and receiver front-end is a PC sound card (line in, line out respectively). Therefore, as mentioned before (*section 1-8*), both processes require a timing period prior their initialization and furthermore, the receiver must continuously monitor for any communication activity because there are no other means of doing so.

The nature of the recording, however, does not allow constant large chunks of info due to resource draining (memory and CPU), thus only small timing intervals are acceptable. Prior the transmission of a packet, the transmitter sends some pulses of equal duration. After this step, it starts transmitting. The receiver on the other hand, should detect this kind of activity and will start receiving or "wake up". The rest of the conversation, (acknowledgement etc…) is handled through wired protocols such as UDP, discussed at Chapter 1. For the communication to succeed the receiver must be able to recognize these patterns as a negotiation attempt; otherwise the packet will be lost. Consequently the problem is twofold; the receiver has to be able to distinguish the handshaking pulses from noise, otherwise will result to "false alarms", and to actually record them, since each monitoring time-slot is followed by a delay due to buffer purging, initialization and data processing.

The solution to recognizing the negotiation pulses is to use the pattern-matching module -the synchronizer, developed before. The differences are that the used sequences must be shorter, and thresholding is used for identification. The second problem is treated by transmitting the same sequence multiple times, so the receiver will identify at least some of them. Beyond a threshold of identified pulses, the receiver will enter the state of packet reception. Unfortunately negotiating prior to transmission imports a fixed delay, depending on the pulse train duration, and the ratio of the transmitted versus to the required to be identified, thus leading to a tradeoff between accuracy and delay.

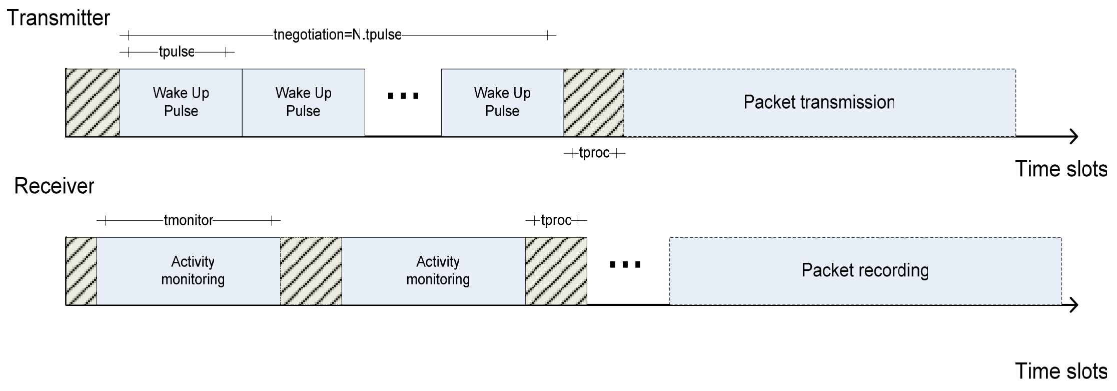
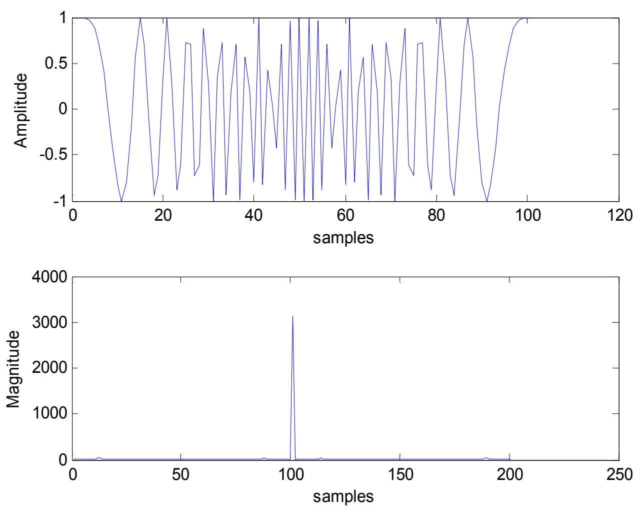
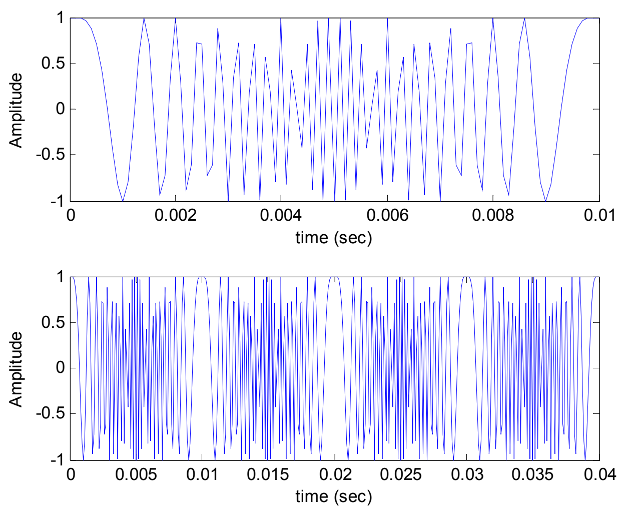


Figure 4.4: Handshaking events representation.

Figure 3.4 illustrates the actions of the receiver and the transmitter. The negotiation attempts on behalf of the transmitter consist of N identical linear swept-frequency signal pulses of equal duration t_{pulse} (fig. 3.5). The distinct gaps represent the system delays mentioned before, which in general are considered as processing time slots of duration t_{proc}. The receiver scans for the transmitted pulses periodically for $t_{monitor}$ time slots. Finally, the ratio of transmitted vs. required is simply denoted as (N / N_{req}) and referred to as *negotiation ratio*. These are the crucial parameters for this part of this system, which are examined in the following sections in order to derive specific guidelines and parameter settings.



(a)



(b)

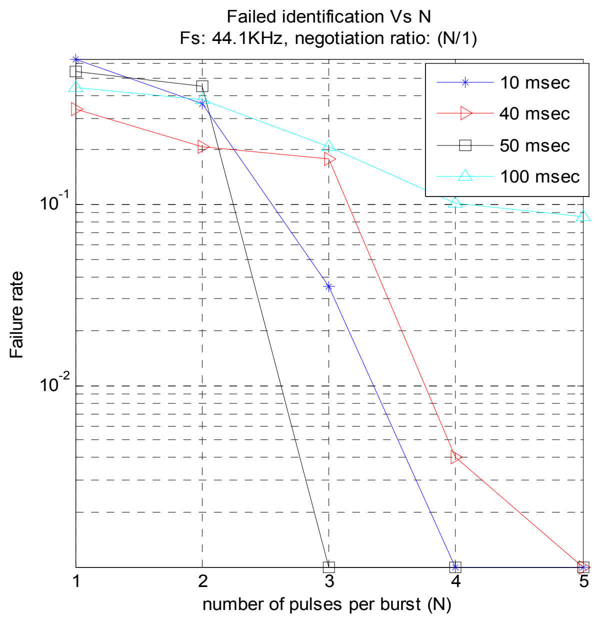
Figure 4.5: (a) depicts a chirp signal and its autocorrelation. (b) Chirp pulse of 0.01 sec and the resulting "wake up" pulse train of 0.04 sec.

4.4 Performance analysis

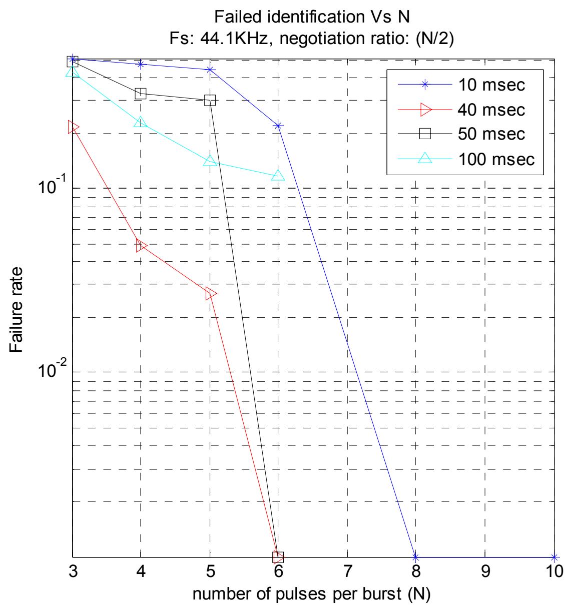
The evaluation is limited to the negotiation sub-system since frame detection or alignment depend on the length of the training sequence witch in turn is defined mostly by the equalizers. Suffice to note, although, that the detection error for sequences of 100 bits or more is smaller that 10^{-6} meaning that detection errors are caused only by negotiation delays or other system malfunctions.

The performance metrics are limited to the negotiation failure (failed attempts) versus the negotiation ratio (N / N_{req}) for various pulse periods (t_{pulse}). Since the delays must be kept at a minimum, at least when possible, the monitoring period is set to 0.05 seconds, as is the minimum Matlab can achieve. The experiments are performed at sampling frequencies (Fs) 44.1 KHz and 88.2 KHz, while the threshold is the 10 to 20 percent of the squared pulse energy ($(0.1 \sim 0.2) | E_{pulse} |^2$).

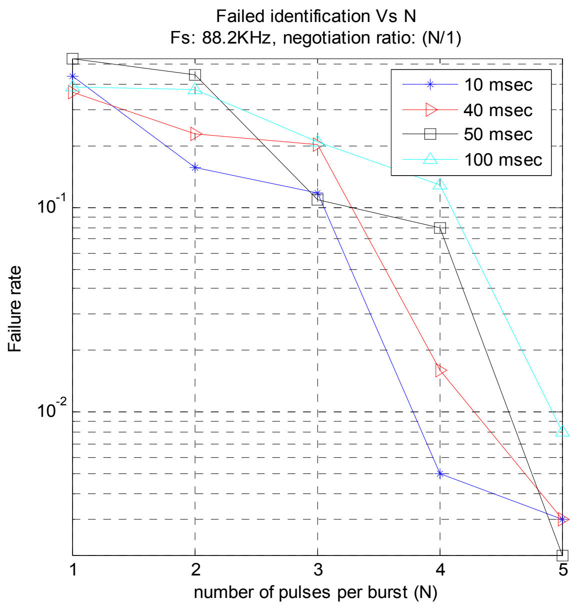
The results of the experiments depicted in figures 3.6 (a) to (d), provide with useful deductions regarding the parameters presented through this chapter as well as useful guidelines which are applied in the final setup of the WFTP prototype. Let us present the derived observations in respect with the involved parameters.



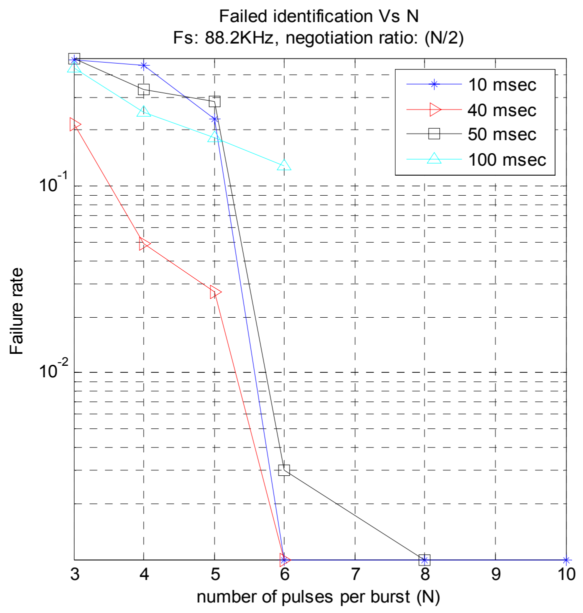
(a)



(b)



(c)



(d)

Figure 4.6: Failure rate versus negotiation ratio. Note that the legends refer to the chirp pulse period, therefore a pulse train burst of N periods takes N*t_{pulse} seconds. Plots (a) and (b) refer to 44.1KHz and ratios (N/1) and (N/2) respectively, while (c), (d) to 88.2KHz.

Chirp period (t_{pulse})

The chirp wave period affects both the resolution of the wake up signal thus its resilience to interference and the possibility of the receiver to record it. The experiments reveal that period of 40msec and sampling frequency of 44 KHz provides adequate pulses that can easily be identified. The mismatch error illustrated in figure 3.6 can easily derive from the fact that the receiver monitors periodically for 50msec after witch follows a processing time interval. If the handshake attempt falls into this time slot, it will be ignored or only a part of it will be recorded. Too short pulses, like 10msec, are most affected, as well as those of duration much larger than the monitoring period, like 100msec, since a maximum of 50% at best can be recorded. Therefore, the most reasonable solution lies within the range of 10msec to 40msec. This conclusion can also derive from figure 3.4 in the previous section.

Negotiation Ratio (N / N_{req})

Recall that the negotiation ratio denotes the number of chirp signal periods transmitted; versus the successful identifications the receiver must accomplish prior entering the state of packet reception. Clearly the higher the ratio the most likely is to be detected. Of course, that also affects the induced delay prior each transmission. Therefore, more appropriate metrics for comparison would be the error rate in conjunction with the total duration, noted in figure 3.4 where $t_{negotiation} = Nt_{pulse}$. For ratios of (*N/1*), best performance is achieved with a period of *10msec* and ratio *5/1* adding a maximum delay (worst case scenario) of additional 40msec. The same applies when the playback frequency is 88.2 KHz, but with slightly worst results due to the increased processing delay. To even lower the probability of identification failure and that of a false alarm scenario, ratios of *N/2* are best suited but at a higher cost in delay. Consequently, from figures 3.6(b) and 3.6(d), ratios of *(10/2)* at *10msec* and *44.1/88.2* KHz provide best results with maximum delay of *80msec*, although a ratio (*6/2*) with pulse period of *40msec* could also be selected at slower machines or when the host happens to operate under heavy load. The latter is a common case during successive transfers thus is preferred to combat unexpected system behavior without any loss of performance.