



Analysis of audio coding algorithms for networked embedded systems

by

*Knut Ole Hauge
&
Svein Birger Skogly*

Masters Thesis in
Information and Communication Technology

Agder University College
&
The University of Queensland

Brisbane, May 04


Abstract

Today there are many different audio coding algorithms, some of them standardized by the ITU, like G.711, G.726 and G.728, and several other codecs by other organizations, like Fraunhofer (Mp3) and Microsoft (WMA). All these algorithms differ in several important features, like speech quality, bit or compression rate, robustness, delay, sampling frequency, complexity and range of use.

The past decade we have witnessed a great progress towards application of low-rate speech/music coders as well as computer related voice/music applications. Central to this progress has been the development of speech coders capable of producing high-quality speech and music at low rates. Most of these coders incorporate mechanisms to: represent the spectral properties of speech/music and optimize the coder's performance for the human ear.

The objective of this thesis is to compare audio coding algorithms for use in an embedded networked system: PCM and ADPCM for speech, and Mp3, WMA and Ogg-Vorbis for music. The thesis report starts with a theoretical background of the different transport protocols, some signal theory and information about all the audio coding algorithms we would like to analyse.

Numerical analysis with MatLab and a subjective assessment of audio quality are performed to find out which audio codec is best suited in an embedded VoIP system. Our main observations are; that the quantization noise increases when the quantization levels decreases, that is, lower bit rate result in higher quantization noise; codecs implemented in MatLab with lower than 40 kbit/s bit rate is not suited for an audio based embedded system; and that WMA 48 kbit/s is very well suited for a networked embedded radio system.

	Master thesis in Information and Communication Technology	Date: 31-May-04
	Analysis of audio coding algorithms for networked embedded systems	Version: 2.8NO

Preface

This master thesis is written at The University of Queensland (School of Information Technology and Electronic Engineering) in Brisbane, Australia. The thesis is performed to complete our Master of Science degree in Information and Communication Technology (ICT) at Ager University College (Faculty of Engineering and Science) in Grimstad, Norway.

We would like to use this opportunity to thank Associate Prof. Lars Line for contacting UQ and making it possible for us to come to Australia. Further we would like to thank Tor Erik Christensen, at AUC's International office, for his effort and help during the application period, Sunil Shukla for his help with MatLab. Last but not least we would like to thank our supervisor at UQ, Prof. Neil Bergmann for his effort during the application period and guidance throughout our project.

Knut Ole Hauge

Svein Birger Skogly



List of Figures

FIGURE 1.2:1 – IP-TELEPHONE TO PC COMMUNICATION	2
FIGURE 2.3:1 – TCP SEGMENT STRUCTURE	8
FIGURE 2.3:2 – UDP SEGMENT STRUCTURE	9
FIGURE 2.3:3 – IPV4 DATAGRAM FORMAT	11
FIGURE 2.7:1 – ILLUSTRATION OF QUANTIZATION [31]	14
FIGURE 2.8:1 – PCM CODEC	17
FIGURE 2.8:2 – G.726 ADPCM ENCODER BLOCK DIAGRAM [25]	18
FIGURE 2.8:3 – G.726 ADPCM DECODER BLOCK DIAGRAM [25]	19
FIGURE 2.8:4 – G.728 LD-CELP SIMPLIFIED ENCODER BLOCK DIAGRAM [26]	20
FIGURE 2.8:5 – G.728 LD-CELP SIMPLIFIED DECODER BLOCK DIAGRAM [26]	20
FIGURE 2.8:6 – MP3 FRAME HEADER [15]	21
FIGURE 2.8:7 – BLUETOOTH SUB-BAND ENCODER PROCESS [22]	24
FIGURE 2.8:8 – BLUETOOTH SUB-BAND CODEC DECODER PROCESS [22]	25
FIGURE 3.1:1 – RCM3400 CORE MODULE [29]	26
FIGURE 3.1:2 – THE PROTOTYPING BOARD [29]	27
FIGURE 3.3:1 – VIEW OF ELECTRICAL CIRCUITRY	27
FIGURE 3.4:1 – FLOWCHART FOR IMPLEMENTING AUDIO CODING ALGORITHMS	28
FIGURE 3.4:2 – FLOW CHART PCM A-LAW AND U-LAW	29
FIGURE 3.4:3 – DPCM (A) ENCODER (B) DECODER [32]	29
FIGURE 3.4:4 – ADPCM ENCODER [25]	30
FIGURE 3.4:5 – ADPCM DECODER [25]	32
FIGURE 3.4:6 – MENU SELECTION GUIDE	32
FIGURE 3.4:7 – NUMBER OF QUANTIZATION STEPS	33
FIGURE 3.4:8 – MODE OF ENCODER	33
FIGURE 3.5:1 – SCREENSHOT OF VB APPLICATION	34
FIGURE 3.5:2 – FLOWCHART FOR VB APPLICATION	35
FIGURE 4.1:1 – AWGN NOISE FILTER	36
FIGURE 5.1:1 – INPUT AND OUTPUT FOR DECODED UNIFORM PCM	40
FIGURE 5.1:2 – UNIFORM PCM QUANTIZED	40
FIGURE 5.1:3 – INPUT AND OUTPUT FOR DECODED A-LAW PCM	41
FIGURE 5.1:4 – INPUT AND OUTPUT FOR DECODED MU-LAW PCM	41
FIGURE 5.1:5 – QUANTIZED A-LAW PCM	41
FIGURE 5.1:6 – QUANTIZED MU-LAW PCM	41
FIGURE 5.1:7 – COMPARISON OF QUANTIZATION AND COMPRESSION	42
FIGURE 5.2:1 – INPUT AND OUTPUT FOR DECODED DPCM	42
FIGURE 5.2:2 – QUANTIZED DPCM	43
FIGURE 5.3:1 – INPUT AND OUTPUT FOR DECODED ADPCM 32KBIT/S	43
FIGURE 5.3:2 – QUANTIZED ADPCM 32 KBIT/S	44
FIGURE 5.3:3 – QUANTIZED ADPCM 16 KBIT/S	44
FIGURE 5.4:1 – QUANTIZATION ERROR FOR ALL PCM MODES	45
FIGURE 5.4:2 – QUANTIZATION NOISE FOR 64 LEVEL	46
FIGURE 5.4:3 – QUANTIZATION NOISE FOR 32 LEVEL	46
FIGURE 5.4:4 – QUANTIZATION NOISE FOR 16 LEVEL	46
FIGURE 5.4:5 – QUANTIZATION NOISE FOR 8 LEVEL	46
FIGURE 5.4:6 – BER AS A FUNCTION OF SNR	47
FIGURE 5.4:7 – SNR PERFORMANCE FOR 128 LEVELS	48
FIGURE 5.5:1 – COMPARISON OF AVERAGE QUALITY RATING	49
FIGURE 5.5:2 – RESULTS FROM THE TRANCE GENRE	50
FIGURE 5.5:3 – RESULTS FROM THE POP GENRE	51
FIGURE 5.5:4 – RESULTS FROM THE CLASSICAL GENRE	51
FIGURE 5.5:5 – RESULTS FROM THE SPEECH LISTENING TEST	52
FIGURE 5.5:6 – ERRORS FOUND BY THE PARTICIPANTS	53

List of Tables

TABLE 2.8:1 – AUDIO CODING ALGORITHMS-----	16 -
TABLE 2.8:2 – THE THIRTEEN HEADER FILES' CHARACTERISTICS [15] -----	22 -
TABLE 3.1:1 – SUMMARY OF RCM3400 FEATURES -----	26 -
TABLE 4.2:1 – EXPECTED QUALITY RATING FOR MUSIC SAMPLES -----	38 -
TABLE 4.2:2 – EXPECTED QUALITY RATING FOR SPEECH SAMPLES-----	39 -
TABLE 4.2:3 – ERRORS ADDED TO SPEECH SAMPLES -----	39 -
TABLE 4.2:4 – QUALITY SCALE -----	39 -
TABLE 5.4:1 – SNR PERFORMANCE -----	47 -
TABLE 5.5:1 – COMPARISON OF AVERAGE QUALITY RATING-----	48 -
TABLE 5.5:2 – TEST OF SPEECH SAMPLES WITH AUDIENCE-----	52 -
TABLE 5.6:1 – COMPUTATIONAL COMPLEXITY -----	54 -
TABLE 7.1:1 – ABBREVIATIONS AND ITS EXPLANATIONS -----	65 -



Table of Content

ABSTRACT	II
PREFACE	III
LIST OF FIGURES	IV
LIST OF TABLES	V
TABLE OF CONTENT	VI
TABLE OF CONTENT	VI
CHAPTER 1 - INTRODUCTION	1 -
1.1. THESIS INTRODUCTION	1 -
1.2. THESIS DESCRIPTION	2 -
1.3. THESIS OUTLINE	3 -
CHAPTER 2 - LITERATURE REVIEW	4 -
2.1. BASICS OF VOIP	4 -
2.2. VOIP RELATED PROTOCOLS	5 -
2.2.1. H.323	5 -
2.2.2. SIP	5 -
2.2.3. RTP	6 -
2.2.4. RTCP	7 -
2.3. TRANSPORT PROTOCOLS	7 -
2.3.1. TCP	7 -
2.3.2. UDP	9 -
2.3.3. UDP vs. TCP in real-time systems	10 -
2.3.4. IP	10 -
2.4. QUALITY OF SERVICE (QOS)	12 -
2.5. PACKET LOSS AND MISSING PACKETS	12 -
2.6. ECHO	13 -
2.7. SIGNAL THEORY	13 -
2.7.1. Sampling	13 -
2.7.2. Encode/Decode	13 -
2.7.3. A/D & D/A – conversion	13 -
2.7.4. Quantization	14 -
2.7.5. Compander/Expander	15 -
2.7.6. Aliasing	15 -
2.7.7. Signal-To-Noise Ratio and BER	15 -
2.8. AUDIO CODING ALGORITHMS	16 -
2.8.1. G.711 PCM	16 -
2.8.2. G.726 ADPCM	17 -
2.8.3. G.728 LD-CELP	19 -
2.8.4. MP3	20 -
2.8.5. WMA	23 -
2.8.6. Ogg-Vorbis	23 -
2.8.7. Bluetooth Sub-band Codec	24 -
CHAPTER 3 - DESIGN AND IMPLEMENTATION	26 -
3.1. INTRODUCTION TO RABBIT TRAINER	26 -
3.1.1. Rabbit core module	26 -
3.1.2. Prototyping board	26 -
3.2. DEVELOPMENT TOOLS	27 -
3.3. CIRCUITRY INTERFACES	27 -
3.4. IMPLEMENTATION OF AUDIO CODING ALGORITHMS	28 -
3.4.1. PCM implementation	28 -
3.4.2. DPCM implementation	29 -
3.4.3. ADPCM implementation	30 -
3.4.4. Implementation of noise module	32 -

3.4.5.	<i>The functionality of the MatLab program</i>	32 -
3.5.	IMPLEMENTATION OF VISUAL BASIC APPLICATION	33 -
CHAPTER 4 - EXPERIMENTS		36 -
4.1.	NUMERICAL EXPERIMENTS	36 -
4.1.1.	<i>Test motivation</i>	36 -
4.1.2.	<i>Simulation environment</i>	36 -
4.1.3.	<i>Experiment 1 – Quantization noise vs. Quantization levels</i>	36 -
4.1.4.	<i>Experiment 2 – BER vs. SNR</i>	36 -
4.1.5.	<i>Experiment 3 – SNR performance of coders</i>	36 -
4.1.6.	<i>Simulation issues</i>	37 -
4.2.	PERFORMANCE OF SUBJECTIVE AUDIO QUALITY TEST	37 -
4.2.1.	<i>Music samples</i>	37 -
4.2.2.	<i>Speech samples</i>	38 -
4.2.3.	<i>Errors added to speech samples</i>	39 -
4.2.4.	<i>Quality scale</i>	39 -
CHAPTER 5 - RESULTS		40 -
5.1.	G.711 PCM	40 -
5.1.1.	<i>Uniform PCM</i>	40 -
5.1.2.	<i>A-law PCM and μ-law PCM</i>	41 -
5.1.3.	<i>The effects of compression algorithms</i>	42 -
5.2.	DPCM	42 -
5.3.	G.726 ADPCM	43 -
5.4.	NUMERICAL EXPERIMENTS IN MATLAB	45 -
5.4.1.	<i>Quantization error for the different PCM modes</i>	45 -
5.4.2.	<i>Experiment 1 – Quantization noise vs. Quantization levels</i>	46 -
5.4.3.	<i>Experiment 2 - BER vs. SNR</i>	47 -
5.4.4.	<i>Experiment 3 – SNR performance of coders</i>	47 -
5.5.	SUBJECTIVE ASSESSMENT OF AUDIO QUALITY	48 -
5.5.1.	<i>Music samples</i>	50 -
5.5.2.	<i>Speech samples</i>	52 -
5.5.3.	<i>Speech samples with introduced errors</i>	53 -
5.6.	ELECTRICAL CIRCUITRY	54 -
5.7.	VISUAL BASIC APPLICATION	54 -
CHAPTER 6 - DISCUSSION		55 -
6.1.	THE IMPLEMENTATION OF THE CODECS	55 -
6.1.1.	<i>Evaluation methods</i>	55 -
6.1.2.	<i>Possible source of errors</i>	55 -
6.1.3.	<i>The results</i>	56 -
6.2.	THE NUMERICAL EXPERIMENTS IN MATLAB	56 -
6.2.1.	<i>Evaluation methods</i>	56 -
6.2.2.	<i>Possible source of errors</i>	56 -
6.2.3.	<i>Experiment 1 – Quantization noise vs. Quantization levels</i>	57 -
6.2.4.	<i>Experiment 2 – BER vs SNR</i>	57 -
6.2.5.	<i>Experiment 3 – SNR performance of coders</i>	58 -
6.3.	SUBJECTIVE ASSESSMENT OF AUDIO QUALITY	58 -
6.3.1.	<i>Evaluation methods</i>	58 -
6.3.2.	<i>Possible source of errors</i>	59 -
6.3.3.	<i>Music samples</i>	60 -
6.3.4.	<i>Speech samples</i>	60 -
6.3.5.	<i>Speech samples with introduced errors</i>	61 -
CHAPTER 7 - CONCLUSION		62 -
7.1.	RECOMMENDATIONS FOR FUTURE WORK	62 -
REFERENCES		63 -
ABBREVIATIONS		65 -

APPENDIX OVERVIEW ----- - 66 -

APPENDIX A – DATASHEET FOR THE ELECTRICAL CIRCUITRY ----- - 66 -

APPENDIX B – MATLAB CODE ----- - 66 -

APPENDIX C – VISUAL BASIC V.6.0 CODE ----- - 66 -

APPENDIX D – DYNAMIC C CODE ----- - 66 -

APPENDIX E – SUBJECTIVE ASSESSMENT OF AUDIO QUALITY ----- - 66 -

Chapter 1 - Introduction

1.1. Thesis introduction

The transmission of voice over packet switched networks, such as an IP network (like the Internet), is an area of active research. Much of the past work focused on using packet switching for both voice and data in a single network. Renewed interest in packet voice, and more generally, packet audio applications has been fuelled by the availability of supporting hardware, increased bandwidth throughout the Internet and the desire to integrate data and voice services in the networks.

Intercom is today used as a communication system linking different rooms within a building etc. VoIP, also called Internet telephony or IP telephony, is the transmission of voice telephony services over IP. Combining these two technologies this project aims to make an intercom, following on with an analysis of what audio codec algorithm is best suited for this purpose. For more efficient use of bandwidth it is interesting to investigate how different audio coding algorithms is encoded and decoded.

The motivation for transporting voice over IP networks is the potential cost saving achievable by eliminating or bypassing the circuit-switched telephony infrastructure. This form of intercom will have several advantages compared to normal telephony or intercom. These advantages include lower costs per call, especially for long-distance calls; lower infrastructure costs once IP infrastructure is installed, no or little additional telephony infrastructure is needed. Note that VoIP traffic does not necessarily have to travel over the public Internet: it may also be deployed on private IP networks. This may be important for a large corporation that has many industry secrets.

Intercom today, is used as an alternative communication inside a building and requires the installation of wires for each speaker. Making this IP based, you can integrate the speakers to the already existing infrastructure of CAT5 cables or WLAN infrastructure. This makes the system cheaper and easier to build. The usage area is almost endless for example public announcements can be heard over the built-in speaker, an emergency button triggers a remote alarm on a highway or on a gas station etc.

1.2. Thesis description

The transfer of voice traffic over packed network, and especially voice over IP (VoIP), is rapidly gaining acceptance. VoIP can significantly reduce the per minute cost, especially for long-distance bills. Today there are many different audio coding algorithms, standardized by the International Telecommunication Union (ITU-T), for use in VoIP networks. All these algorithms differ in several important features, like speech quality, bit or compression rate, robustness, delay, sampling frequency, and complexity.

This thesis aims to develop a networked embedded system (IP-telephone/Intercom), and undertake an analysis with different audio codec algorithms implemented. The codecs; G.711 PCM, G.726 ADPCM and G.728 LDCELP will be tested. Additionally it will be build interface circuitry for the Rabbit trainer. Then error coders will be added to the pc, measuring the effects. Finally, experiments with different audio codecs will be run, considering cost and performance. If time permits, experiments with other codecs, like MP3, WMA, Ogg-Vorbis and Bluetooth Sub-Band Codec will be investigated.

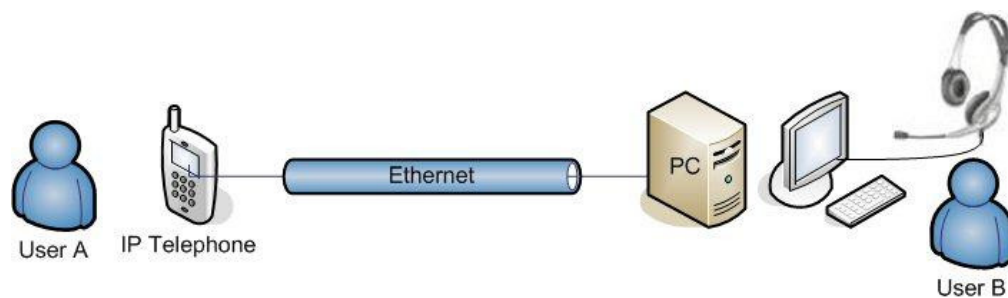


Figure 1.2:1 – IP-telephone to PC communication

1.3. Thesis outline

Chapter 2 - Literature review

This chapter is a little dive into some of the protocols, audio codec algorithms and some signal theory that will be useful for the thesis.

Chapter 3 - Design and implementation

This chapter gives the reader a short introduction to the Rabbit trainer, the development tools used, explains the circuitry interfaces, explains how the audio codecs were implemented and finally explains the implementation of the Visual Basic application.

Chapter 4 - Experiments

In this chapter all the experiments are presented and it is explained how they are performed.

Chapter 5 - Results

In this chapter all the results are presented so that the reader can make their own decisions based on our graphs and analysis of the results.

Chapter 6 - Discussion

This chapter presents an analysis of our results presented in *chapter 6 - results*. Further it explains our main observations and results from the analysis.

Chapter 7 - Conclusion

This chapter presents a clear conclusion based on our results and discussion. This chapter also explains our suggested improvements of the project and what work that remains.

Chapter 2 - Literature review

2.1. Basics of VoIP

Packet voice systems accept “analogue” voice signals from telephone handsets, digitize and compress the signal, placing the resulting series of bits into a short packet, send the packet over a network and then decode and reconstruct the signal at the remote end.

An IP phone perform the digitization, compression and packaging process directly within the phone and sends the resulting stream of packets over an Ethernet connection. [1]

Generally, the benefits of VoIP technology can be divided into the following four categories:

- **Cost Reduction:** Reducing long distance telephone costs is a good reason for implementing VoIP. Today flat rate long distance pricing is available with the Internet and can result in considerable savings for both voice and facsimile (at least currently). The sharing of equipment and operations costs across both data and voice users can also improve network efficiency since excess bandwidth on one network can be used by the other, thereby creating economies of scale for voice (especially given the rapid growth in data traffic).
- **Simplification:** An integrated infrastructure that supports all forms of communication allows for increased standardization and reduces the total equipment complement. This combined infrastructure can support dynamic bandwidth optimization and a fault tolerant design. The differences between the traffic patterns of voice and data, offer further opportunities for significant efficiency improvements.
- **Consolidation:** Since people are the most significant cost elements in a network, any opportunity to combine operations, to eliminate points of failure, and to consolidate accounting systems would therefore be beneficial. In the enterprise, SNMP-based management can be provided for both voice and data services using VoIP. Universal use of the IP protocols for all applications holds out the promise of both reduced complexity and more flexibility. Related facilities such as directory services and security services may be more easily shared.
- **Other Advanced Applications:** Even though basic telephony and facsimile are the initial applications for VoIP, the long term benefits are expected to be derived from multimedia and multiservice applications. For example, Internet commerce solutions can combine WWW access to information with a voice call button that allows immediate access to a call centre agent from the PC. Needless to say, voice is an integral part of conferencing systems that may also include shared screens, whiteboarding, etc. Combining voice and data features with new applications will provide the greatest returns over the long term. Videoconferencing can also be greatly enhanced. [2]

2.2. VoIP related protocols

2.2.1. H.323

H.323 is a standard [3] that specifies the components, protocols and procedures that provide multimedia communication services—real-time audio, video, and data communications over packet networks, including Internet protocol (IP)–based networks. H.323 is part of a family of ITU-T recommendations called H.32x that provides multimedia communication services over a variety of networks. [3]

Fundamentally, there are four entities in a general H.323 implementation. These are **terminals**, **gatekeepers**, **gateways**, and **multipoint control units (MCUs)**. Terminals, gateways, and MCUs are collectively known as endpoints. Not all these entities are required in an H.323 implementation. An H.323 implementation may consist of terminals alone, but the practical usefulness of such a network may be greatly reduced.

Terminals are H.323 client endpoints that provide real-time bi-directional multimedia communications. The terminal may either be a multimedia PC or a stand-alone device. It may merely be a simple telephone. All that the H.323 standard requires from the terminal is that it must support audio communications. Video and data communications are optional. All audio, data, and video processing occur in the terminal.

Gatekeepers are often referred to as the “brains” of an H.323 network. It is the most important component of an H.323 network. The H.323 standard requires 4 functions from the gatekeeper. These are Address Translation; Admissions Control, Bandwidth Control; and Zone Management. Optional gatekeeper functions include Call Control Signal Processing; Call Authorization; Bandwidth Management, and Call Management.

Gateways are optional components of an H.323 implementation. When communications between different networks (such as LAN and PSTN) is desired, gateways are needed at the interface. Gateways provide data format translation, control signalling translation, audio/video codec translation, and call setup/termination functionality on both sides of the network.

MCUs facilitate videoconferencing with more than two participants. It serves as the coordinator of all multimedia capabilities of the participants in a multiparty conference. It can even provide features such as audio mixing and video selection for participating terminals without these capabilities. It accomplishes these functions with the use of a multipoint controller, which is required, and zero or more multipoint processors. MCUs are not required in an H.323 implementation, unless multiparty conferences are desired. [4]

2.2.2. SIP

The Session Initiation Protocol (SIP) [33] is a signalling protocol for initiating, managing and terminating voice and video sessions across packet networks. SIP sessions involve one or more participants and can use unicast or multicast communication. Borrowing from ubiquitous Internet protocols, such as HTTP and

SMTP, SIP is text-encoded and highly extensible. SIP may be extended to accommodate features and services such as call control services, mobility, interoperability with existing telephony systems, and more. SIP is being developed by the SIP Working Group, within the Internet Engineering Task Force (IETF). The protocol is published as IETF RFC 2543 [33] and currently has the status of a proposed standard.

A SIP network is composed of four types of logical SIP entities. Each entity has specific functions and participates in SIP communication as a client (initiates requests), as a server (responds to requests), or as both. One “physical device” can have the functionality of more than one logical SIP entity. For example, a network server working as a Proxy server can also function as a Registrar at the same time. Following are the four types of logical SIP entities:

In SIP, a **User Agent (UA)** is the endpoint entity. User Agents initiate and terminate sessions by exchanging requests and responses. RFC 2543 defines the User Agent as an application, which contains both a User Agent client and User Agent server, as follows:

- **User Agent Client (UAC)** - a client application that initiates SIP requests.
- **User Agent Server (UAS)** - a server application that contacts the user when a SIP request is received and that returns a response on behalf of the user.

Some of the devices that can have a UA function in a SIP network are: workstations, IP-phones, telephony gateways, call agents, automated answering services.

A **Proxy Server** is an intermediary entity that acts as both a server and a client for the purpose of making requests on behalf of other clients. Requests are serviced either internally or by passing them on, possibly after translation, to other servers. A Proxy interprets, and, if necessary, rewrites a request message before forwarding it.

A **Redirect Server** is a server that accepts a SIP request, maps the SIP address of the called party into zero (if there is no known address) or more new addresses and returns them to the client. Unlike Proxy servers, Redirect Servers do not pass the request on to other servers.

A **Registrar** is a server that accepts REGISTER requests for the purpose of updating a location database with the contact information of the user specified in the request. [5]

2.2.3. RTP

RTP, the real-time transport protocol [6], provides end-to-end network transport functions suitable for applications transmitting real-time data, such as audio, video or simulation data, over multicast or unicast network services. RTP does not address resource reservation and does not guarantee quality-of- service for real-time services. The data transport is augmented by a control protocol (RTCP) to allow monitoring of the data delivery in a manner scalable to large multicast networks, and to provide minimal control and identification functionality. RTP and RTCP are designed to be independent of the underlying transport and network layers. The protocol supports the use of RTP-level translators and mixers.

2.2.4. RTCP

RTCP is the control protocol for RTP (Real-time Transport Protocol) [7]. It is used to periodically transmit control packets to participants in a streaming multimedia session. RTCP's primary function is to provide feedback on the quality of service being provided. This feedback may be used to scale back the sender for flow-control reasons or to keep from congesting the network. The sender may also use the information to change the current compression ratio. RTCP is outlined in 1889 (RTP: A Transport Protocol for Real-Time Applications, January 1996) [7].

2.3. Transport protocols

2.3.1. TCP

TCP, Transmission Control Protocol [8], fits into layered protocol architecture just above a basic Internet Protocol which provides a way for the TCP to send and receive variable-length segments of information enclosed in internet datagram “envelopes”. The internet datagram provides a means for addressing source and destination TCPs in different networks. The internet protocol also deals with any fragmentation or reassembly of the TCP segments required to achieve transport and delivery through multiple networks and interconnecting gateways. The internet protocol also carries information on the precedence, security classification and compartmentation of the TCP segments, so this information can be communicated end-to-end across multiple networks.

TCP provides multiplexing, demultiplexing, and error detection in exactly the same manner as UDP (to be explained in 2.3.2). The most fundamental difference between TCP and UDP is that UDP is connectionless, while TCP is connection oriented. TCP is connection oriented because before one application process can begin to send data to another, the two processes must first “handshake” with each other – that is, they must send some preliminary segments to each other to establish the parameters of the ensuing data transfer. The TCP “connection” is not an end-to-end TDM or FDM circuit as in circuit switched network. Nor is it a virtual circuit, as the connection state resides entirely in the two end systems. Because the TCP protocol runs only in the end systems and not in the intermediate network elements (routers and bridges), the intermediate network elements do not maintain TCP connection state. In fact, the intermediate routers are completely oblivious to TCP connections - they see datagrams, not connections. [9 p.207-208]

Connection establishment

The client first sends a special TCP segment, the server responds with a second special TCP segment, and finally the client responds again with a third special segment. The first two segments contain no “payload”, that is, no application-layer data. The third of these segments may carry a payload. Because three segments are sent between the two hosts, this connection establishment procedure is often referred to as a “three-way handshake”.

Once a TCP connection is established, the two application processes can send data to each other, because TCP is full-duplex they can send data at the same time.

TCP Segment structure

The TCP segment structure, shown in *Figure 2.3:1*, is defined in the RFC793 [8]. The TCP segment consists of header file and a data field. The data field contains a chunk of application data. The MSS (Maximum Segment Size) limits the maximum size of a segment's data field. When TCP sends a large file, such as an encoded image as part of a web page, it typically breaks the file into chunks of size MSS (except for the last chunk, which will often be less than the MSS). The TCP header is typically 20 bytes (12 bytes more than the UDP header).

The source and destination port numbers in the header are used for multiplexing/demultiplexing data to/from upper layer applications. Also, as with UDP, the header includes a **checksum** field. A TCP segment header also contains the following fields described below [9 p.210-211]:

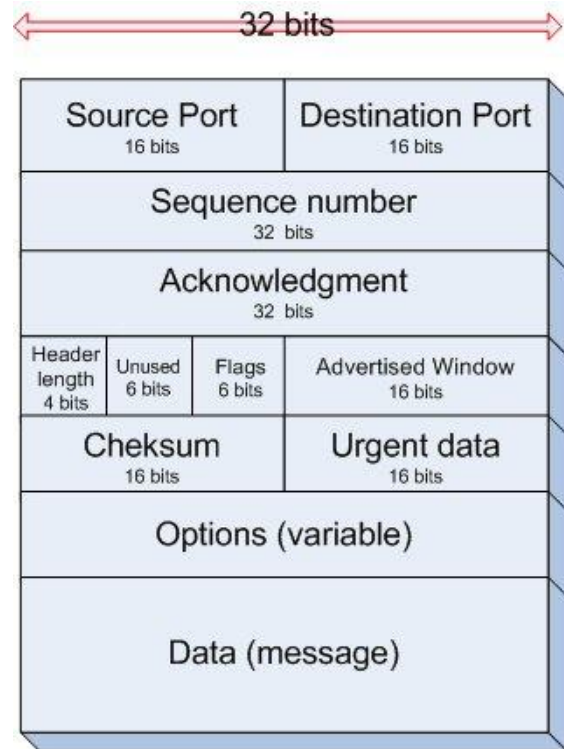


Figure 2.3:1 – TCP segment structure

- The 32-bit **sequence number** field and the 32-bit acknowledgment number field are used by the TCP sender and receiver in implementing a reliable data-transfer service.
- The 16-bit **window-size** (Advertised window) field is used for flow control. This is used to indicate the number of bytes that a receiver is willing to accept.
- The 4-bit **length** field specifies the length of the TCP header in 32-bit words. The TCP header can be of variable length due to the TCP options field, discussed below. (Naturally, the options field is empty, so that the length of the typical TCP header is 20bytes).
- The optional and variable length **options** field is used when a sender and receiver negotiate the maximum segment size (MSS) or as a window scaling factor for use in high-speed networks. A timestamping option is also defined. See RFC854 and RFC1323 for additional details.
- The **flag** field contains 6 bits. The ACK bit is used to indicate that the value carried in the acknowledgment field is valid. The **RST**, **SYN** and **FIN** bits are used for connection setup and teardown. When the **PSH** bit is set, this is an indication that the receiver should pass the data to the upper layer immediately. Finally, the **URG** bit is used to indicate that there is data in this segment that the sending-side upper layer entity has marked as “urgent”. The location of the last byte of this urgent data is indicated by the 16-bit urgent data pointer. TCP must inform the receiving-side upper-layer entity when urgent data exists and pass it a pointer to the end of the urgent data. (In

practice, the PSH, URG, and pointer to urgent data are not used. However, we mention these fields for completeness).

Using TCP you obtain a reliable connection; TCP uses the sequence number and the acknowledgment number to maintain a reliable connection. TCP views data as an unstructured, but ordered, stream of bytes. The sequence number for a segment is the byte-stream number of the first byte in the segment. The acknowledgement number that e.g. host A puts in its segment is the sequence number of the next byte host A is expecting from host B. If A misses one segment from B, B will notice this when B does not receive a correct ACK. B then retransmits the segment requested from A. Another reason for B to resend to a segment is when an ACK from A times out. [9 p.210-211]

2.3.2. UDP

The User Datagram Protocol (UDP) [10] is defined to make a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks available. This protocol assumes that the Internet Protocol (IP) is used as the underlying protocol. This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. [10]

UDP is a no-frills, lightweight transport protocol with a minimalist service model. UDP is connectionless, so there is no “handshaking” before the two processes start to communicate. UDP provides an unreliable data transfer service, that is, when a process sends a message into a UDP socket, UDP provides no guarantee that the message will ever reach the receiving socket. Furthermore, messages that do arrive to the receiving socket may arrive out of order.

On the other hand, UDP does not include a congestion-control mechanism, so a sending process can pump data into a UDP socket at any rate it pleases. Although all the data may not make it to the receiving socket, a large fraction of the data may arrive. [9 p.82]

UDP Segment structure

Aside from multiplexing/demultiplexing function and some light error checking, it adds nothing to the IP. In fact, if the application developer chooses UDP instead of TCP, then the application is almost talking with IP. UDP takes messages from the application process, attaches **source** and **destination port** number fields for the multiplexing/demultiplexing service, and adds (header + data) **length** and **checksum**, and the resulting segment to the network layer. The network layer encapsulates the segment

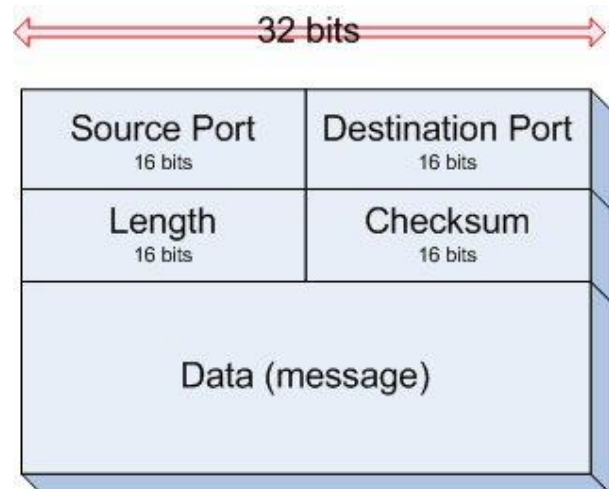


Figure 2.3:2 – UDP segment structure

into an IP datagram and then makes a best-effort attempt to deliver the segment to the receiving host. If the segment arrives at the receiving host, UDP uses the port numbers and the IP destination address to deliver the segment's data to the correct application process. Note that with UDP there is no "handshaking" between sending and receiving transport-layer entities before sending a segment. For this reason, UDP is said to be connectionless. [9. p.177]

The UDP segment, shown in *Figure 2.3:2*, is defined in the RFC768 [10]. The application data occupies the data field of the UDP datagram. For example, for a streaming audio application, audio samples fill the data field. The UDP header only has four fields, each consisting of two bytes. The port numbers allow the destination host to pass the application data to the correct process running on the destination (that is, the demultiplexing function). The checksum is used by the receiving host to check if errors have been introduced into the segment. In truth, the checksum is also calculated over a few of the fields in the IP header in addition to the UDP segment. [9. p.180]

2.3.3. UDP vs. TCP in real-time systems

No connection establishment: TCP uses a three-way handshake before it starts to transfer data. UDP just blasts away without any formal preliminaries. Thus UDP does not introduce any delay to establish a connection.

No connection state: TCP maintains connection state in the end system. This connection state includes receive and send buffers, congestion-control parameters.

Sequence and acknowledgment number parameters: UDP, on the other hand, does not maintain connection state and does not track any of these parameters. For this reason, a server devoted to a particular application can typically support many more active clients when the application runs over UDP rather than TCP.

Small packet header overhead: The TCP segment has 20bytes of header overhead in every segment, whereas UDP only has 8bytes of overhead.

Unregulated send rate: TCP has a congestion control mechanism that throttles the sender when one or more links between sender and receiver become excessively congested. This throttling can have a severe impact on real-time applications, which can tolerate some packet loss but requires a minimum send rate. On the other hand, the speed at which UDP sends data is only constrained by the rate at which the application generates data, the capabilities of the source (CPU, clock rate, and so forth) and the access bandwidth to the internet. We should keep in mind, however, that the receiving host does not necessarily receive all the data. When the network is congested, some of the data could be lost due to router buffer overflow. [9. p.178]

2.3.4. IP

The Internet Protocol [11] is designed for use in interconnected systems of packet-switched computer communication networks. The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks. [11]

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols. The internet protocol can capitalize on the services of its supporting networks to provide various types and qualities of service. [11]

The Options provide for control functions needed or useful in some situations but unnecessary for the most common communications. The Options include provisions for timestamps, security, and special routing. The Header Checksum provides a verification that the information used in processing internet datagram has been transmitted correctly. The data may contain errors. If the header checksum fails, the internet datagram is discarded at once by the entity which detects the error. The internet protocol does not provide a reliable communication facility. There are no acknowledgments either end-to-end or hop-by-hop. There is no error control for data, only a header checksum. There are no retransmissions. There is no flow control. Errors detected may be reported via the Internet Control Message Protocol (ICMP) which is implemented in the internet protocol module. [11]

IPv4 Datagram format

Version number: These four bits specify the IP protocol version of the datagram. By looking at the version number, the router can then determine how to interpret the remainder of the IP datagram. The current datagram format, IPv4, is shown in *Figure 2.3:3*, and is defined in the RFC791 [11].

Header length: Because an IPv4 datagram can contain a variable numbers of options, these four bits are needed to determine where in the IP datagram the data actually begins. Most IP datagrams do not contain Options, so the typical IP datagram has a 20byte header.

TOS: The Type of Service bits were included in the IPv4 header to allow different “types” of IP datagrams to be distinguished from each other, presumably so that they could be handled differently in times of overload. It would be useful to distinguish real-time datagrams (used by an IP-telephony application for example) from non-real-time traffic (for example FTP).

Length (datagram length): This is the total length of the IP datagram (header + data) measured in bytes. Since this field is 16bits long, the theoretical maximum size

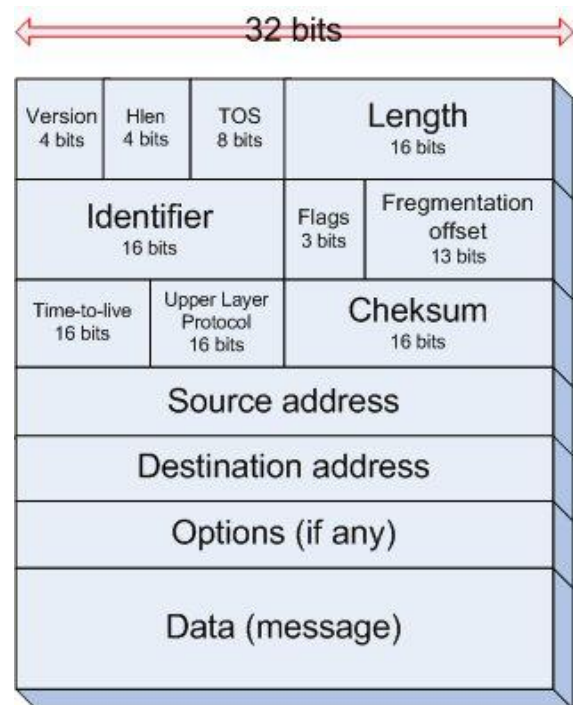


Figure 2.3:3 – IPv4 datagram format

of the IP datagram is 65535 bytes. However, datagrams are rarely greater than 1500 bytes and are often limited in size to 576 bytes.

Identifier, Flags, Fragmentation offset: These three fields have to do with so-called “IP fragmentation”.

Time-to-live: The time-to-live (TTL) field is included to ensure that datagrams do not circulate forever (due to a long-lived router loop for example) in the network. This field is decremented by one each time the datagram is processed by a router and is dropped when it reaches zero.

Upper layer protocol: This field is used only when an IP datagram reaches its final destination. The value of this field indicates the transport-layer protocol at the destination to which the data portion of this IP datagram will be passed.

Checksum (header checksum): The header checksum aids a router in detecting bit errors in a received IP datagram. Routers typically discard datagrams for which an error has been detected. Note that the checksum must be recomputed and restored at each router, as the TTL field and possibly options fields as well, may change.

Source and destination IP address: These fields carry the 32-bit IP address of the source and final destination for this IP datagram.

Options: The options fields allow an IP header to be extended.

Data (payload/message): In most circumstances, the data field of the IP datagram contains the transport-layer segment (TCP or UDP) to be delivered to the destination. However, the data field can carry other types of data, such as ICMP messages. [9 p.314 - 316]

2.4. Quality of Service (QoS)

The current Internet service model is flat, offering a classless and best-effort delivery service. The biggest problem faced by voice over packet networks is that of providing end users with the quality of service that they get in a traditional telephony network. Unlike the PSTN (Public Switched Telephone Network), where a dedicated end-to-end connection is established for a call, packet based networks use statistical multiplexing of network resources. Although sharing resources among multiple users promotes cost effectiveness (hence the attraction of voice over packet networks), it does not guarantee the overall quality of service offered to a user. The next generation of IP, version 6, includes support for the flow control of packets between one or more hosts. In conjunction with a hop-by-hop resource reservation protocol such as RSVP, end-to-end capacity can be set aside for real-time traffic. There are multiple parameters that determine the quality of service provided by a network. This subsection describes the barriers to the operation of these schemes, including requirements for codecs, bandwidth, delays, delay jitter and the packet loss experienced in a network. [9]

2.5. Packet Loss and Missing Packets

Voice packets routed through an IP network could be lost due to its best-effort nature. To provide reliable transmission of data in an IP network, a retransmission

scheme is used at the transport layer (TCP), which retransmits any packets for which an acknowledgement is not received from the destination (assuming that the packet gets lost). However, the same scheme cannot be applied to voice, as a retransmitted voice packet might reach the destination much later than when it is needed. Packets arriving late due to the delays described above are discarded at the receiver. There are also packets which are lost due to the network errors and the best-effort nature of IP networks. All these discarded and lost packets are considered as “missing packets”, and a good reconstruction algorithm is necessary to fill in these packets. [9]

2.6. Echo

Echo occurs as a result of transmitted signals being coupled into a return path and fed back to their respective sources. The returned signal occurs with noticeable delay. The subjective effect of echo is also a function of delay. On short connections, the delay is small enough for the echo to appear to the talker as a mere natural coupling in his ear. A telephone is purposely designed to couple some speech energy (called sidetone) in the earpiece. Otherwise, the telephone seems dead to the talker. If the delay is more than about 25 milliseconds, the caller can hear a distinct echo. Hence, long-distance circuits require significant attenuation to minimize echo annoyance due to this long round-trip delay or else require specialised signal processing circuits to remove echo. Echo affects the talker more than the listener. Because of this echo, one can hear one’s own voice in the receiver after a delay of more than about 25 ms - this can cause interruptions and break the cadence in a conversation. [9]

2.7. Signal theory

2.7.1. Sampling

Sampling is the process of encoding an analogue signal in digital form by reading (sampling) its level, at precisely spaced intervals of time. [31]

Nyquist theorem:

“An analogue signal must be sampled at a rate of twice the maximum frequency of the signal to produce an accurate representation of the signal. Conversely, for a given sampling rate, the maximum frequency of the analogue signal that can be accurately represented is one-half the sampling rate”.

2.7.2. Encode/Decode

An encoder is any program, circuit or algorithm which encodes an input signal, by converting electronic data into a given format, e.g. binary PCM, while the decoder converts the encoded signal back to the original format as a reconstruction of the original input signal. [31]

2.7.3. A/D & D/A – conversion

Analogue-to-Digital (A/D)

A A/D device is a device that changes the continuous fluctuations in voltage from an analogue device, such as a microphone, into digital information that can be stored or processed in a sampler, digital recording device. [31]

Digital-to-Analogue (D/A)

A D/A device is a device that changes the sample words put out by a digital audio device into analogue fluctuations in voltage that can be sent to a mixer, amplifier, or speaker for example. [31]

2.7.4. Quantization

Quantization is a way of representing an analogue signal by a vector of discrete values. The signal, after quantization, has a stepped shape rather than its original continuous curve, and the difference between this and the original signal is quantization error. [31]

Quantization error is the difference between the actual analogue value at the sample time and the nearest quantized (digitally encoded) value. At worst, the quantized value encoded will be no greater than one-half increment away from the actual analogue value. Quantization error is related to the SNR and the maximum number of quantization increments is related to dynamic range. [31]

Quantization noise is one of the types of error introduced into an analogue audio signal by encoding it in digital form. The digital equivalent of tape hiss, quantization noise, is caused by the small differences between the actual amplitudes of the points being sampled and the bit depth of the A/D converter. In the quantization of a sine wave whose frequency is a sub-multiple of the sampling frequency, the error will have a definite pattern which repeats at the frequency of the signal, having a frequency content consisting of multiples of this frequency, where it can be considered as harmonic distortion rather than noise. In music, however, the signal is constantly changing and no such regularity exists, resulting in quantization error, producing wideband noise, called quantization noise. [31]

Quantizing increments is the total number of stepped levels, from noise floor to saturation, that an A/D has available for assignment of the continuously varying analogue input voltage with each sample taken. [31]

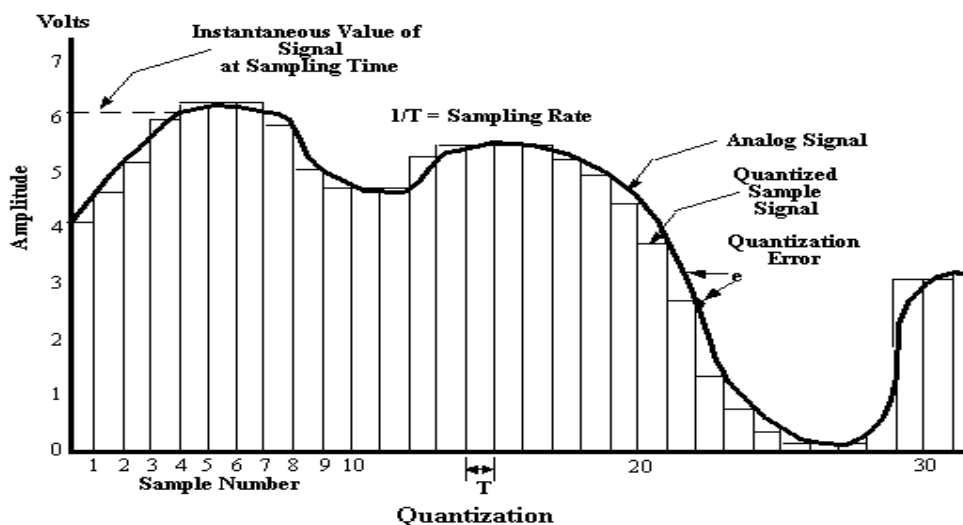


Figure 2.7:1 – Illustration of Quantization [31]

2.7.5. Compander/Expander

A **compander** is a device for noise reduction in audio devices such as recorders. The compander will reduce the dynamic range of the signal before sending it to be recorded. The compression makes the softer passages louder so the dynamic range recorded is less than it would be if it were not compressed. Then on playback, the signal is expanded, that is, the softer passages, which are too loud, are reduced in volume to match the original signal, restoring its dynamics. [31]

Expander is a signal processing device which is the inverse of a compressor, providing the gradual attenuation of signals that fall below a user-defined threshold. This process, known as expansion, reduces background noise and at the same time increases the dynamic range of the input signal. [31]

2.7.6. Aliasing

Aliasing is distortion that is produced when higher harmonic components of the input audio signal sampled by a digital recording device, or generated within a digital sound source, lie above the Nyquist frequency. The effects of aliasing differ from some other types of distortion in that its pitch changes radically when the pitch of the intended sound changes. [31]

Before a signal is subjected to the process of A/D conversion, it must be passed through a low-pass brick-wall filter to remove any components that are higher than the Nyquist frequency. This is because it requires at least two samples per cycle to determine the existence and strength of a frequency component, or the A/D process will create aliased signals. [31]

2.7.7. Signal-To-Noise Ratio and BER

Signal-to-Noise Ratio in analogue and digital communications is a measure of signal strength relative to background noise. The ratio is usually measured in decibels (dB).

If the incoming signal strength in microvolts is V_s , and the noise level, also in microvolts, is V_n , then the signal-to-noise ratio, S/N (in decibels) is given by the formula

$$S/N = 20 \log_{10}(V_s/V_n)$$

Equation 2.7:1 - SNR formula

If $V_s = V_n$, then $S/N = 0$. In this situation, the signal borders are unreadable, because the noise level severely competes with it. In digital communications, this will probably cause a reduction in data speed because of frequent errors that require the source computer or terminal to resend some packets of data. Ideally, V_s is greater than V_n , so S/N is positive. [31]

The Bit Error Rate (BER) in telecommunication is the transmission of the percentage of bits that have errors relative to the total number of bits received in a transmission, usually expressed as ten to a negative power. For example, a transmission might have a BER of 10 to the minus 6, meaning that, out of 1 000 000 bits transmitted, one bit was in error. The BER is an indication of how often a packet or other data units has to be retransmitted because of an error. Too high a

BER may indicate that a slower data rate would actually improve overall transmission time for a given amount of transmitted data since the BER might be reduced, lowering the number of packets that had to be resent. [31]

2.8. Audio coding algorithms

International Telecommunication Union (ITU-T) has standardized numbers of speech codecs that sample voice and convert it into digital code, like G.711 (PCM), G.721 ADPCM, G.726 (ADPCM) and G.728 (LD-CELP). CCITT (International Consultative Committee on Telecommunications and Telegraphy) changed its name to ITU-T on 1 March 1993.

Table 2.8:1 – Audio coding algorithms

Standard:	Standardized by:	Description:	Bit rate: (kbs)	Sampling rate: (kHz)	Typically end-to-end delay (ms) excluding channel delay:
G.711 PCM	ITU-T	Pulse code modulation	64	8	<< 1
G.726 ADPCM	CCITT (ITU)	Adaptive Differential Pulse Code Modulation	16, 24, 32, 40	8	60
G.728 LD-CELP	CCITT (ITU)	Low-Delay Code Excited Linear Prediction	16	8	<< 2
Mp3	Fraunhofer ISS	MPEG1 Layer III	32 – 320	8 – 48	N/A
WMA	Microsoft	Windows Media Audio	32 – 320	8 – 48	N/A
Ogg-vorbis	Xiph.org Foundation	Ogg-Vorbis	16 – 320	8 – 48	N/A
BSC	Bluetooth	Bluetooth Sub-band Codec	16 – 32	8 – 48	N/A

2.8.1. G.711 PCM

Without using any knowledge of how the signal to be coded was generated, waveform codecs attempt to produce a reconstructed signal whose waveform is as close as possible to the original. This means that in theory they should be signal independent and work well with non-speech signals. Generally they are low complexity codecs which produce high quality speech at rates above about 16kbps. When the data rate is lowered below this level the reconstructed speech quality that can be obtained, degrades rapidly.

PCM merely involves sampling and quantizing the input waveform. Narrow-band speech is typically band-limited to 4 kHz and sampled at 8 kHz. If linear quantization is used then, to give good quality speech, around 12 bits per sample are needed, giving a bit rate of 96kbts/s. This bit rate can be reduced by using non-uniform quantization of the samples. In speech coding an approximation to a logarithmic quantizer is often used. Such quantizers give a signal to noise ratio, which is almost constant over a wide range of input levels, and at a rate of 8 bits/sample (or 64kbts/s) give a reconstructed signal which is almost indistinguishable from the original uniformly quantised signal. [12]

The accuracy of the lower amplitude parts of the signal is more important than the larger amplitude parts, hence a non-linear method of quantization can be used to reduce the data rate. The idea here is to vary the distance between quantization reconstruction levels so that the distance increases as the amplitude of sample increases. To do this, the sampled signal is first passed through a logarithmic compressor and then uniformly quantized. In reverse the signal is passed through an expander with the inverse transform characteristics of the compressor. This process is referred to as “companding”. Two particular logarithmic quantization techniques for PCM, as defined by CCITT are in worldwide use. Essentially, the G.711 standard compresses a 13-bit A-law (Europe) or 14-bit μ -law (US) linear PCM sample to an 8 bit logarithmic representation. [23]

The definition, their relationship and translation between A-law and μ -law are given in the tables presented in the G.711 recommendation from ITU-T. [24]

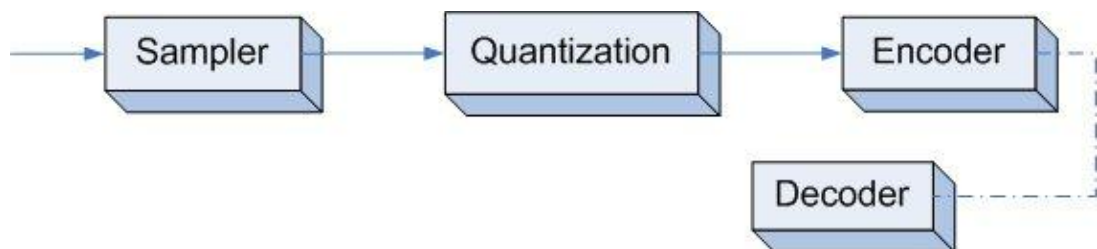


Figure 2.8:1 – PCM codec

2.8.2. G.726 ADPCM

ADPCM, Adaptive Differential Pulse Code Modulation, is a way an analogue signal can be converted to a digital signal. In the 1980s that was first standardized by the CCITT in G.721 for 32 kbps. Later came the standards G.726 and G.727 for 40, 32, 24 and 16 kbps. G.726 replaces G.721 and G.723 [13].

G.726 specifies how a 64 kbps A-law or μ -law PCM signal can be converted to 40, 32, 24 or 16 kbps ADPCM channels where the 24 and 16 kbps channels are used for voice in Digital Circuit Multiplication Equipment (DCME) and the 40 kbps is for data modem signals (especially modems doing 4800 kbps or higher) in DCME. [13]

Adaptive Differential Pulse Code Modulation (ADPCM) codecs are waveform codecs, which instead of quantizing the speech signal directly, like PCM codecs, quantize the difference between the speech signal and a prediction that has been made of the speech signal. If the prediction is accurate then the difference between the real and predicted speech samples will have a lower variance than the real speech samples, and will be accurately quantized with fewer bits than would be needed to quantize the original speech samples. At the decoder the quantized difference signal is added to the predicted signal to give the reconstructed speech signal. The performance of the codec is aided by using adaptive prediction and quantization, so that the predictor and difference quantizer adapt to the changing characteristics of the speech being coded. [12]

ADPCM encoder

Subsequent of the A-law or μ -law PCM input signal to uniform PCM, a difference signal is obtained, by subtracting an estimate of the input signal from the input signal itself. An adaptive 31-, 15-, 7-, or 4-level quantizer is used to assign five, four, three or two binary digits, respectively, to the value of the difference signal for transmission to the decoder. An inverse quantizer produces a quantized difference signal from these same five, four, three or two binary digits, respectively. The signal estimate is added to this quantized difference signal are operated reconstructed version of the input signal. Both the reconstructed signal and the quantized difference signal are operated upon by an adaptive predictor which produces the estimate of the input signal, thereby completing the feedback loop. Illustration of the encoding process is presented in *Figure 2.8:2*. The encoding process is further explained in the G.726 recommendation from CCITT (ITU). [25]

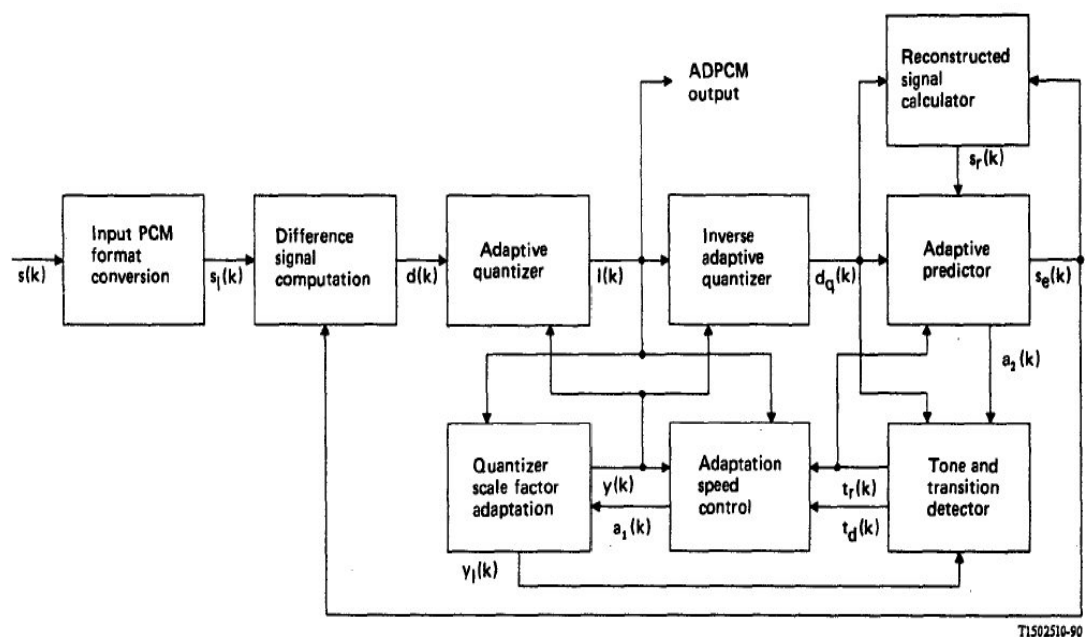


Figure 2.8:2 – G.726 ADPCM Encoder block diagram [25]

ADPCM Decoder

The decoder includes a structure identical to the feedback portion of the encoder, together with a uniform PCM to A-law or μ -law conversion and a synchronous coding adjustment.

The synchronous coding adjustment prevents cumulative distortion occurring on synchronous tandem coding (ADPCM-PCM-ADPCM, etc., digital connections) under certain conditions [see 25 §3.7]. The synchronous coding adjustment is achieved by adjusting the PCM output codes in a manner which attempts to eliminate quantizing distortion in the next ADPCM encoding stage. Illustration of the decoding process is presented in *Figure 2.8:3*. The decoding process is further explained in the G.726 recommendation from CCITT (ITU) [25].

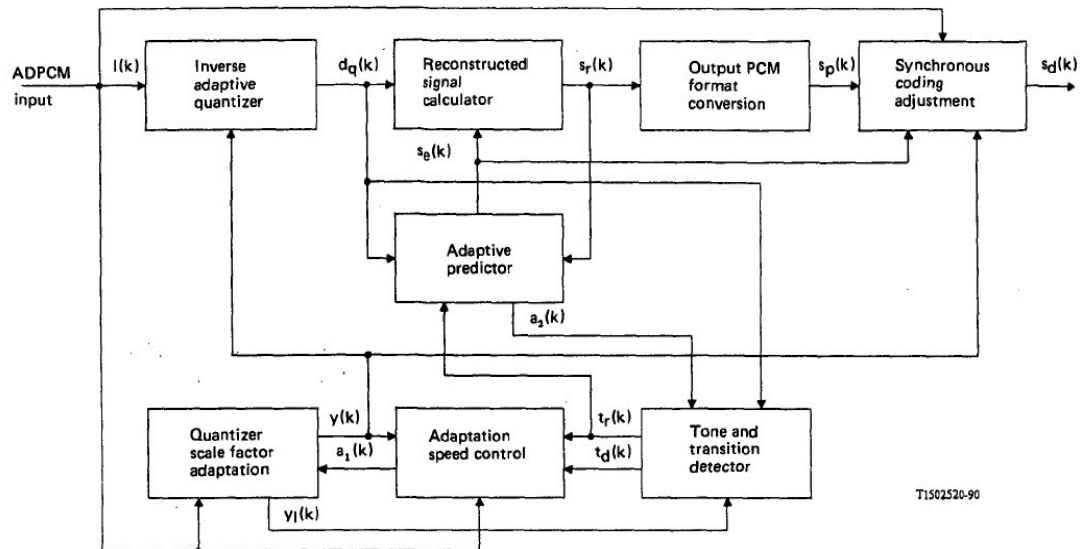


Figure 2.8:3 – G.726 ADPCM Decoder block diagram [25]

2.8.3. G.728 LD-CELP

G.728 Low-Delay Code Excited Linear Prediction (LD-CELP) compression is a 16 kbps compression. This has an algorithmic coding delay of 0.625ms. However, because of the forward adaptive determination of the short term filter coefficients used in most of these codecs, they tend to have high delays. The delay of a speech codec is defined as the time from when a speech sample arrives at the input of its encoder to when the corresponding sample is produced at the output of its decoder, assuming the bit stream from the encoder is fed directly to the decoder. For a typical hybrid speech codec, this delay will be in the order of 50 to 100 ms, and such a high delay can cause problems. [12]

Encoding Processes

After the conversion from A-law or μ -law PCM to uniform PCM, the input is portioned into blocks of five-consecutive input signal samples. For each input block, the encoder passes each of 1024 candidate codebook vectors (stored in excitation codebook) through a gain scaling unit and a synthesis filter. From the resulting 1024 candidate quantized signal vectors, the encoder identifies the one that minimizes a frequency-weighted mean-squared error measure with respect to the input signal vector. The 10-bit codebook index of the corresponding best codebook vector (or “codevector”), which gives rise to that best candidate quantized signal vector, is transmitted to the decoder. The best codevector is the passed through the gain scaling unit and the synthesis filter to establish the correct filter memory in preparation for the encoding of the next signal vector. The synthesis filter coefficients and the gain are updated periodically in a backward adaptive manner based on the previously quantized signal and gain-scaling excitation. Illustration of the encoding process is presented in *Figure 2.8:4*. The encoding process is further explained in the G.728 recommendation from CCITT (ITU) [26].

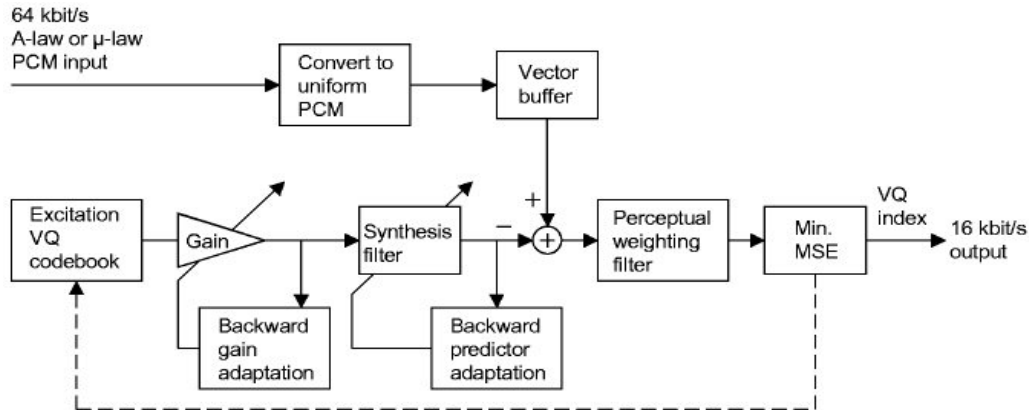


Figure 2.8:4 – G.728 LD-CELP simplified encoder block diagram [26]

Decoding Processes

Upon receiving each 10-bit index, the decoder performs a table look-up to extract the corresponding codevector from the excitation codebook. The extracted codevector is then passed through a gain scaling unit and a synthesis filter to produce the current decoded signal vector. The synthesis filter coefficients and the gain are then updated in the same way as in the encoder. The decoded signal vector is then passed through an adaptive postfilter to enhance the perceptual quality. The postfilter coefficients are updated periodically using the information available at the decoder. The five samples of the postfilter signal vector are next converted to five A-law or μ -law PCM output samples. Illustration of the decoding process is presented in *Figure 2.8:5*. The decoding process is further explained in the G.728 recommendation from CCITT (ITU) [26].

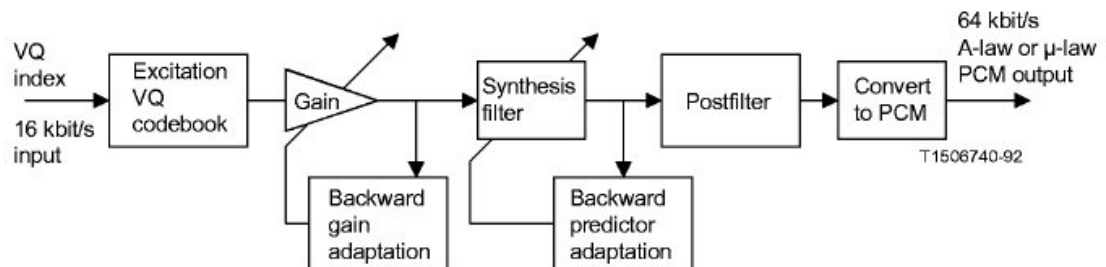


Figure 2.8:5 – G.728 LD-CELP simplified decoder block diagram [26]

2.8.4. MP3

The MPEG/audio compression algorithm is the first international standard for the digital compression of high-fidelity audio. MP3 stands for MPEG1 Layer III. This layer introduces increased frequency resolution based on a hybrid filter bank. The MP3 encoder compresses the data on the basis of Psychoacoustics Model. The compression is done in the frequency domain. The decoder decodes the MP3 streams by using the Huffman Decode tables. [14]

First the mp3 codec throws away what humans can't hear anyway (or at least it makes acceptable compromises), and then it encodes the redundancies to achieve further compression. It then breaks the signal into smaller component pieces called "frames," each lasting a fraction of a second. One could consider these frames to be similar to those in a movie film. The encoding bitrate is taken into account, and the maximum number of bits that can be allocated to each frame is calculated. For

instance, if you're encoding at 128 kbps, you would have an upper limit on how much data can be stored in each frame (unless you're encoding with variable bitrates). This step determines how much of the available audio data will be stored, and how much will be left on the cutting room floor. The frequency spread for each frame is compared to mathematical models of human psychoacoustics, which are stored in the codec as a reference table. From this model, it can be determined which frequencies need to be rendered accurately, seeing as though they'll be perceptible to humans, and which ones can be dropped or allocated fewer bits, as we wouldn't be able to hear them anyway. The collection of frames is assembled into a serial bitstream, with header information preceding each data frame. The headers contain instructional “meta-data” specific to that frame. [15]

Lossy compression, both used in Mp3 and Ogg-Vorbis, is any compression that causes information to be lost. Compressing and then uncompressing a file results in something similar, but not identical, to the original file. This is no good for things that must be interpreted by a computer, like executable programs/applications or most computer-readable data, but is more than sufficient for things where the interpretation is being done by a human (like photographs or sounds). The trick is to remove little bits of information in places where it cannot be perceived. Lossy audio compression works using a psychoacoustic model. That is, by modelling how your ears (and your brain) hear sound, it is possible to find places from which to remove information that you would not have perceived anyway. [16]

As mentioned earlier, MP3 files are segmented into zillions of frames, each containing a fraction of a second's worth of audio data, ready to be reconstructed by the decoder. Inserted at the beginning of every data frame is a “header frame”, which stores 32 bits of meta-data, related to the coming data frame. As illustrated in *Figure 2.8:6*, the MP3 header begins with a “sync” block, consisting of 11 bits. The sync block allows players to search for and “lock onto” the first available occurrence of a valid frame, which is useful in MP3 broadcasting, for moving around quickly from one part of a track to another, and for skipping ID3 or other data that may be living at the start of the file. However, note that it's not enough for a player to simply find the sync block in any binary file and assume that it's a valid MP3 file, since the same pattern of 11 bits could theoretically be found in any random binary file. Thus, it is also necessary for the decoder to check for the validity of other header data as well, or for multiple valid frames in a row. *Table 2.8:2* lists the total 32 bits of header data that are spread over 13 header positions.

Sync		
ID	Layer	Prot. bit
Bitrate		
Frequency	Pad. bit	Priv. bit
Mode	Mode extention	
Copy	Home	Emphasis
Audio Data		

Figure 2.8:6 – Mp3 frame header [15]

Table 2.8:2 – The Thirteen Header Files' Characteristics [15]

Position	Purpose	Length (Bits)
A	Frame sync	11
B	MPEG audio version (MPEG-1, 2, etc.)	2
C	MPEG layer (Layer I, II, III, etc.)	2
D	Protection (if on, then checksum follows header)	1
E	Bit rate index (lookup table used to specify bit rate for this MPEG version and layer)	4
F	Sampling rate frequency (44.1kHz, etc., determined by lookup table)	2
G	Padding bit (on or off, compensates for unfilled frames)	1
H	Private bit (on or off, allows for application-specific triggers)	1
I	Channel mode (stereo, joint stereo, dual channel, single channel)	2
J	Mode extension (used only with joint stereo, to conjoin channel data)	2
K	Copyright (on or off)	1
L	Original (off if copy of original, on if original)	1
M	Emphasis (respects emphasis bit in the original recording; now largely obsolete)	2

Following the **sync block** comes an **ID bit**, which specifies whether the frame has been encoded in MPEG-1 or MPEG-2. Two layer bits follow, determining whether the frame is Layer I, II, III, or not defined. If the **protection bit** is not set, a 16-bit checksum will be inserted prior to the beginning of the audio data.

The bit rate field, naturally, specifies the bit rate of the current frame (e.g., 128 kbps), which is followed by a specifier for the audio frequency (from 16,000Hz to 44,100Hz, depending on whether MPEG-1 or MPEG-2 is currently in use). The padding bit is used to make sure that each frame satisfies the bit rate requirements exactly.

The mode field refers to the stereo/mono status of the frame, and allows for the setting of stereo, joint stereo, dual channel, and mono encoding options. If joint stereo effects have been enabled, the mode extension field tells the decoder exactly how to handle it, i.e., whether high frequencies have been combined across channels.

The copyright bit does not hold copyright information per se (obviously, since it's only one bit long), but rather mimics a similar copyright bit used on CDs and DATs. If this bit is set, it is officially illegal to copy the track (some ripping programs will report this information back to you if the copyright bit is found to be set). If the data is found on its original media, the home bit will be set. **The “private” bit** can be used by specific applications to trigger custom events.

The emphasis field is used as a flag, in case a corresponding emphasis bit was set in the original recording. **The emphasis bit** is rarely used, although some recordings do still use it.

Finally, the decoder moves on through the **checksum** (if it exists) and on to the actual **audio data frame**, and the process begins all over again, with thousands of frames per audio file. [15]

2.8.5. WMA

The Microsoft Windows Media Audio codec is designed to handle all types of audio content, from speech-only audio recorded with a sampling rate of 8 kilohertz (kHz) to 48 kHz high-quality stereo music. This codec is very resistant to degradation due to packet loss because it does not use interframe memory. Its tolerance makes it excellent for use with streaming content. In addition, by using an improved encoding algorithm, this codec encodes and decodes much faster. The compression algorithm used, creates audio files that need much less disk space for storage than the same content created with other codecs. Content created using the Windows Media Audio codec is easily distributed over the Internet because the files can be downloaded more quickly. Therefore, if you are creating audio files for download, the Windows Media Audio codec is a great choice because it provides near-CD quality sound at half of the bandwidth required by most other codecs. [17]

2.8.6. Ogg-Vorbis

Ogg-Vorbis is a fully open compressed audio format for mid to high quality (8 kHz – 48.0 kHz, 16+ bits, polyphonic) audio and music at fixed and variable bitrates from 16 to 128 kbps/channel. This places Vorbis in the same competitive class as audio representations such as MPEG-4 (AAC), and similar to, but higher performance than MPEG-1/2 audio layer 3, MPEG-4 audio (TwinVQ), WMA and PAC.

The Vorbis CODEC design assumes a complex, psychoacoustically-aware encoder and simple, low-complexity decoder. Vorbis decode is computationally simpler than mp3, although it does require more working memory as Vorbis has no static probability model; the vector codebooks used in the first stage of decoding from the bitstream are packed, in their entirety, into the Vorbis bitstream headers. In packed form, these codebooks occupy only a few kilobytes; the extent to which they are pre-decoded into a cache is the dominant factor in decoder memory usage.

Vorbis provides none of its own framing, synchronization or protection against errors. It is solely a method of accepting input audio, dividing it into individual frames and compressing these frames into raw, unformatted “packets”. The decoder then accepts these raw packets in sequence, decodes them, synthesizes audio frames from them, and reassembles the frames into a facsimile of the original audio stream. Vorbis is a free-form VBR codec and packets have no minimum size, maximum size, or fixed/expected size. Packets are designed that they may be truncated (or padded) and remain decidable. This is not to be considered an error condition and is used extensively in bitrate management. Both the transport mechanism and decoder allow for a packet to be of any size, to end before, or after the packet decoder expects.

Vorbis packets are thus intended to be used with a transport mechanism that provides free-form framing, sync, positioning and error correction in accordance with these design assumptions, such as Ogg (for file transport) or RTP (for network multicast). For purposes of a few examples in this document, we will assume that Vorbis is to be embedded in an Ogg stream specifically, although this is by no means a requirement or fundamental assumption in the Vorbis design. [19]

2.8.7. Bluetooth Sub-band Codec

The Bluetooth SBC is a low computational complexity audio coding system designed for high quality audio at moderate bitrates. Bluetooth SBC is based on low-complexity. Block diagrams of the SBC encoder and decoder algorithms are shown in *Figure 2.8:7* and *Figure 2.8:8*. The SBC system uses a cosine-modulated filterbank (polyphase analysis) for analysis and synthesis. The filterbank can be configured for 4 or 8 sub-bands. The sub-band signals are quantized using a dynamic bit allocation scheme and block adaptive PCM quantization. The number of bits available and the number of blocks to quantize over are variable, making the overall bit-rate of the SBC system adjustable. This is advantageous for use in wireless applications where the available wireless bandwidth for audio, and hence the maximum possible bit-rate, may vary over time. [27]

In Sub-Band Coding (SBC) the input speech is split into a number of frequency bands, or sub-bands, and each is coded independently using, for example, an ADPCM like coder. At the receiver the sub-band signals are decoded and recombined to give the reconstructed speech signal. The advantages of doing this come from the fact that the noise in each sub-band is dependent only on the coding used in that sub-band. Therefore we can allocate more bits to perceptually important sub-bands so that the noise in these frequency regions is low, while in other sub-bands we may be content to allow a high coding noise because noise at these frequencies is less perceptually important. Adaptive bit allocation schemes may be used to further exploit these ideas. Sub-band codecs tend to produce communications to toll quality speech in the range 16-32 kbits/s. [28]

Encoding Processes

Via a polyphase analysis filter the input PCM is split into sub-band signals. For each sub-band a scale factor is calculated. On the basis of the scale factors and the bit allocation, levels are derived for each sub-band. Then the sub-band samples are scaled and quantized and finally, a bitstream is generated. This process is further explained in [22].

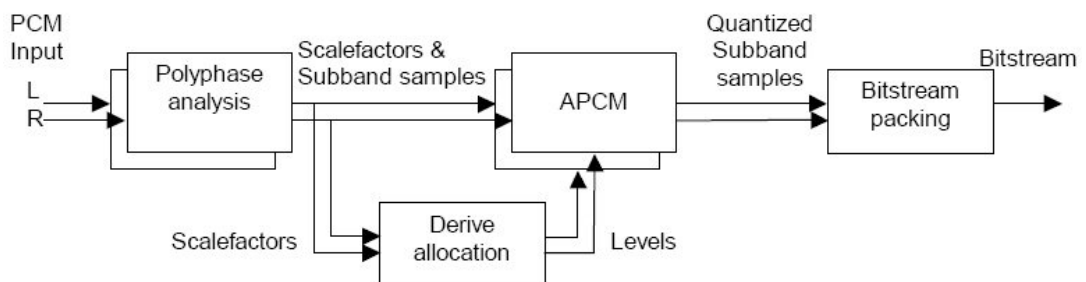


Figure 2.8:7 – Bluetooth sub-band encoder process [22]

Decoding Processes

In *Figure 2.8:8* the operation of the decoder is illustrated. On the basis of the scale factors, the bit allocation is calculated. For the MONO and DUAL_CHANNEL the bit allocation is calculated for each channel independently. For the STEREO and JOINT_STEREO channel modes, the allocation calculation for the two channels is combined. Then the numbers of quantization levels are derived for each sub-band. The sub-band samples are calculated and finally, via a polyphase synthesis filter, the PCM output is generated. This process is further explained in [22].

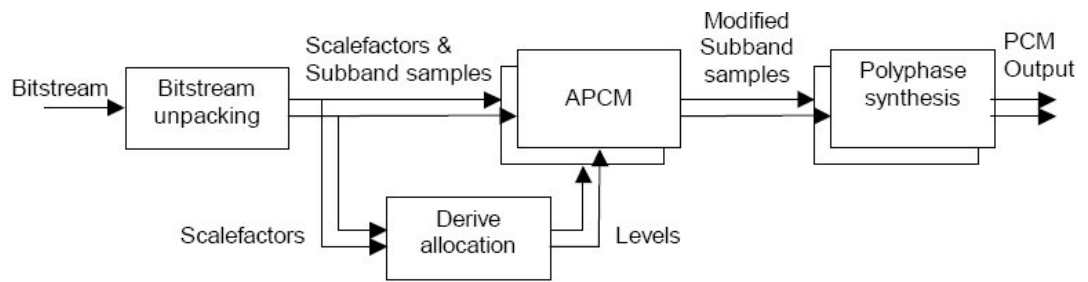


Figure 2.8:8 – Bluetooth sub-band codec decoder process [22]

Chapter 3 - Design and implementation

3.1. Introduction to Rabbit trainer

3.1.1. Rabbit core module

The microprocessor chosen for this thesis is the Rabbit 3000 (RCM3400) running at 29.4 MHz. Extracted from the datasheets provided by the manufacturer, the Rabbit has 47 I/O ports, 512K flash memory, 512K SRAM, PWM outputs, ten 8-bit timers and one 10-bit timer with two match registers and a 12-bit 8-channel A/D converter. It has also interfaces to the Prototyping board, giving the Rabbit a 10/100MB Ethernet connection, IrDA transceiver etc [29]



Figure 3.1:1 – RCM3400 Core Module [29]

A memory-access time of 55 ns suffices to support up to a 30 MHz clock with no wait states; with a 30 ns memory-access time, a clock speed of up to 50 MHz is possible with no wait states. This means that the Rabbit can execute the first instruction in the interrupt subroutine in about 1 μ s, which is very good considering that the Rabbit needs to be interrupted every 8 kHz in order to supply 64 Kbits voice quality. [29]

Table 3.1:1 – Summary of RCM3400 features

Feature:	RCM3400:
Microprocessor	Rabbit 3000 running at 29,4 MHz
Flash Memory	512 K
SRAM	512 K
Serial Ports	5 shared high-speed, CMOS-compatible ports: 5 are configurable as asynchronous serial ports; 4 are configurable as clocked serial ports (SPI); 2 are configurable as SDLC/HDLC serial ports; 1 asynchronous serial port is used during programming

3.1.2. Prototyping board

The Prototyping Board included in the Development Kit makes it easy to connect an RCM3400 module to a power supply and a PC workstation for development. It also provides some basic I/O peripherals (RS-232, RS-485, an IrDA transceiver, an Ethernet port, LEDs, and switches), as well as a prototyping area for more advanced hardware development. [29].

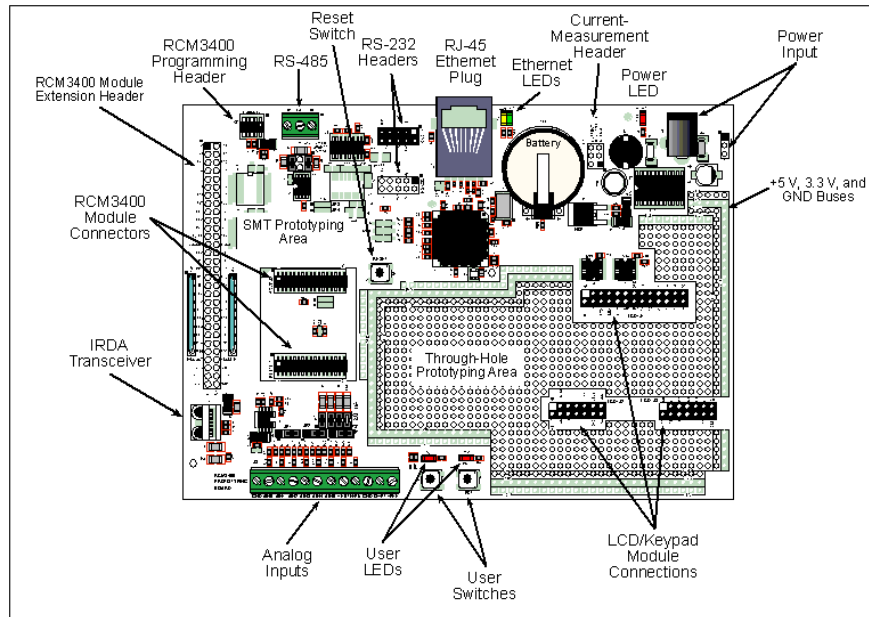


Figure 3.1:2 – The Prototyping Board [29]

3.2. Development tools

To develop the GUI prototype to send sound files from pc to the Rabbit trainer, Visual Basic V6.0 was used. V.6.0 was favoured over .Net since there are more example files available on the internet for VB6. On the Rabbit trainer Dynamic C 8.03 is used for communication with the pc on the other side.

To perform the analysis of the different audio codecs, G.711 PCM, G.726 ADPCM and DPCM, was implemented in MatLab V.6.5 R13. MatLab was also used to add noise and measure these noise effects placed on the audio files. Adobe Audition V.1.0 was used to make samples for a wider analysis, that is, samples that were not implemented in MatLab, e.g. Mp3 and WMA files. CDex V.1.3 was used to make Ogg-Vorbis files.

Protel 99 was used for designing the electrical circuitry and for drawing the PCB board.

3.3. Circuitry interfaces

To realise the hardware part in the specification it contains a built microphone circuitry and a speaker circuitry. The circuitry is designed to work on an external power supply $\pm 10V$ and GND, instead of on the Rabbit trainer.

Explanations, component list, circuitry overview, schematics and calculations can be found in *Appendix A – Datasheet for the electrical circuitry*. Figure 3.3:1 illustrates the electrical circuitry.

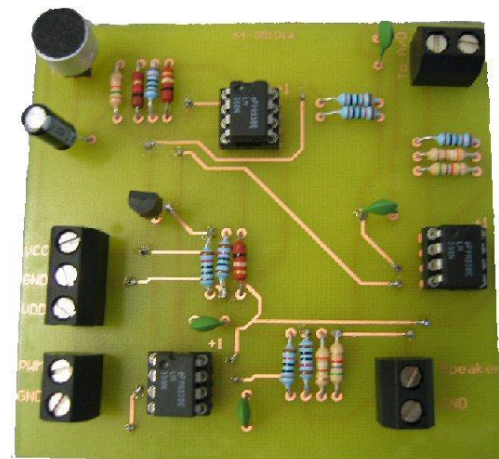


Figure 3.3:1 – View of electrical circuitry

3.4. Implementation of audio coding algorithms

Implementation of linear PCM, A-law PCM, μ -law PCM, DPCM and ADPCM audio coding algorithms is performed using MatLab programming language and environment, developed by Mathworks. Both encoder and decoder for all the algorithms are coded. There is also an implemented function to introduce error into the codecs where the effects are measured. An encoder, a decoder and an AWGN (Additive White Gaussian Noise) module is implemented. The encoders and decoders for the respective algorithms are coded in separate files like functions. Subsequently the main encoder and decoder file calls the codec-functions for the selected audio coding algorithm. The user can choose which codec to use to encode/decode the input signal.

Step 1: An input signal is generated by opening a file and reading the content into a vector. It is possible to choose how many characters one will read into the vector, which would then be appropriated according to testing and evaluation.

Step 2: The generated input signal is quantized using the round function in MatLab. Every sample is rounded to the nearest quantization level. The quantized signal is then plotted in a graph.

Step 3: The quantized samples are saved into a file and the encoded samples are displayed in a graph. The saved file is then passed to the decoder and the noise module. In the noise module error can be introduced to the encoded signal and the effects will then be measured.

Step 4: Dequantization is merely the inverse of quantization.

Step 5: The decoded signal is simply a reconstruction of the original signal. The decoded and the original signal are plotted to visualize the difference between them. Samples of the decoded signal are saved into a wav file. The wav file can then be played to give an indication of the quality of the codec.

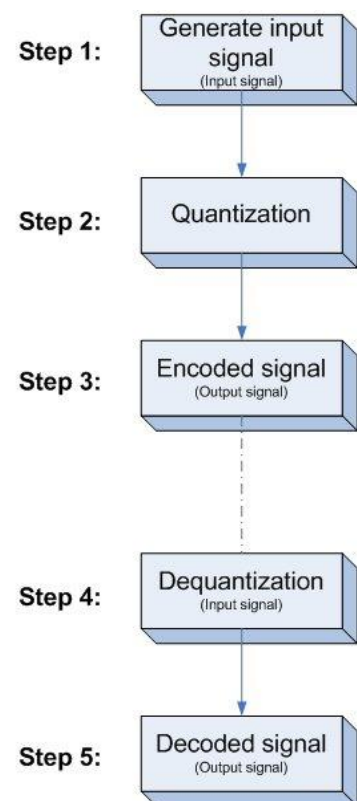


Figure 3.4:1 – Flowchart for implementing audio coding algorithms

3.4.1. PCM implementation

PCM is a sampling technique used to convert voice signals into digital code. Input signal is sampled at 8000 samples/sec. The PCM-codec samples the input waveform, quantizes the samples, and represents each quantizer level with a binary index. For the implementation of PCM, a uniform quantizer is used - each sample being rounded off to the nearest quantization level.

By choosing smaller quantization steps for small signals and larger steps for large signals, a more robust quantizer can be obtained. Non-uniform spacing between quantization steps can be achieved by using a non-linear compander technique. Companding can be described as the compression and expansion of the quantization levels. In the Recommendation G.711 [24], two companding laws are specified. The compression characteristics for the companding laws are described below:

PCM A-law: To encode a signal according to A-law compression, the following algorithm was used. A in Equation 3.4:1 is set to 86.7. V is set to 1

$$y = \begin{cases} \frac{A|x|}{1 + \log A} \operatorname{sgn}(x) & \text{for } 0 \leq |x| \leq \frac{V}{A} \\ \frac{V(1 + \log(A|x|/V))}{1 + \log A} \operatorname{sgn}(x) & \text{for } \frac{V}{A} \leq |x| \leq V \end{cases}$$

Equation 3.4:1 - A-law compressor [24]

PCM μ -law: μ -law encoding use the compression algorithm below, Equation 3.4:2. Here μ is set to 255. V is set to 1.

$$y = \frac{V \log(\mu|x|/V)}{\log(1 + \mu)} \operatorname{sgn}(x)$$

Equation 3.4:2 - μ -law compressor [24]

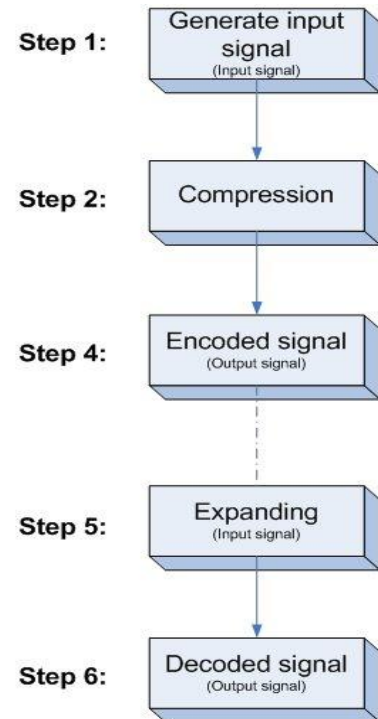


Figure 3.4:2 – Flow chart PCM a-law and u-law

3.4.2. DPCM implementation

Implementation of PCM using differential coding is called DPCM. Instead of coding the signal, differential coding is used to code the difference between two signals. By using this means of coding, as much short-term redundancy of the speech waveform is removed as possible. To accomplish this, a difference signal is formed by subtracting an estimate of the signal from the original signal. The estimate is generally obtained by a linear predictor that estimates the current samples from a linear combination of one or more past samples.

A first-order adaptive predictor is used in the implementation:

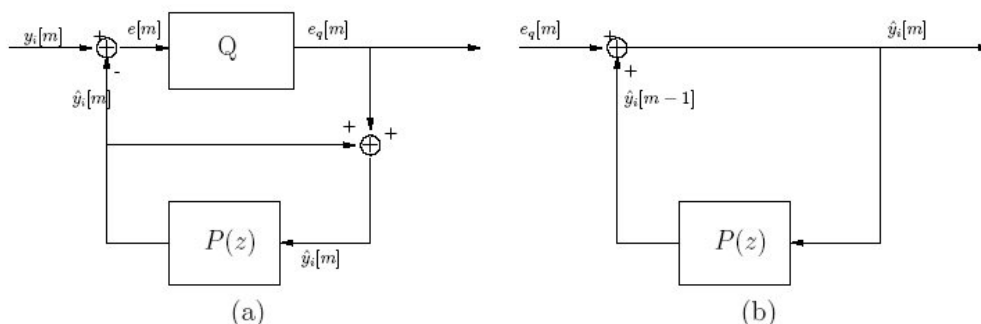


Figure 3.4:3 – DPCM (a) encoder (b) decoder [32]

3.4.3. ADPCM implementation

The ADPCM algorithm is implemented according to the block diagrams for encoder/decoder in the G 7.26 recommendation from CCITT in ITU. It is conceptually similar to DPCM but more sophisticated in that it uses an eight-order predictor, adaptive quantizer and adaptive predictor. The algorithm is also designed to recognise the difference between voice and data signals, and then use either a slow or fast quantizer adaptation mode. ADPCM provides greater levels of prediction gain than simple DPCM, depending on the sophistication of the adaptation logic and the number of past samples used to predict the next sample. The prediction gain is ultimately limited by the fact that only a few past samples are used to predict the input. Adaptation logic only adapts the quantizer, not the prediction-weighting coefficient. The ADPCM algorithm is implemented in the way that the user can choose between 40, 32, 24 and 16 kbit/s.

Encoder schematic:

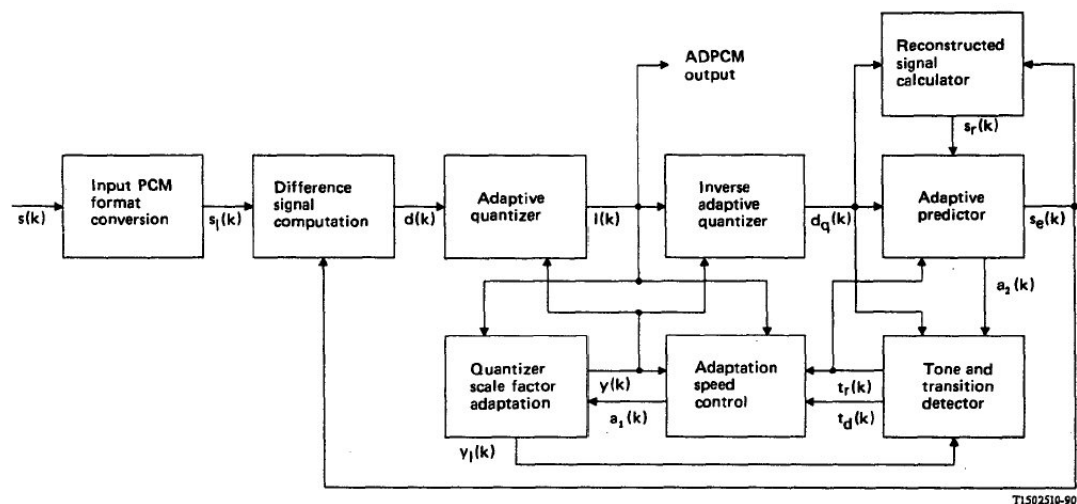


Figure 3.4:4 – ADPCM encoder [25]

Encoder principles [25]:

1. Input PCM format conversion

This block converts the input signal $s(k)$ from A-law or μ -law PCM to a uniform PCM signal $s_1(k)$.

2. Difference signal computation

This block calculates the difference signal $d(k)$ from the uniform PCM signal $s_1(k)$ and the signal estimate $s_e(k)$.

3. Adaptive quantizer

A 31-, 15-, 7- or 4-level non adaptive quantizer is used to quantize the difference signal $d(k)$ for operating at 40, 32, 24 or 16 kbit/s, respectively. Prior to quantization, $d(k)$ is converted to a base 2 logarithmic representation and scaled by $y(k)$ which is computed by the scale factor adaptation block. Normalized input/output characteristic values from tables in the recommendation for the quantizer are used.



4. Inverse adaptive quantizer

A quantized version $d_q(k)$ of the difference signal is produced by scaling, using $y(k)$, specific values selected from the normalized quantization characteristics in the recommendation and then transforming the result from the logarithmic domain.

5. Quantizer scale factor adaptation

In this block $y(k)$ is computed. This is the scaling factor for the quantizer and the inverse quantizer. The inputs are the 5-bit, 4-bit, 3-bit, 2-bit quantizer output $I(k)$ and the adaptation speed control parameter $a_1(k)$. The basic principle used in scaling the quantizer is bimodal adaptation. Fast for speech signal and slow for data signals. The speed of adaptation is controlled by a combination of fast and slow scale factors.

6. Adaptation speed control

The controlling parameter $a_1(k)$ can assume values in the range of $[0, 1]$. It tends towards unity for speech signals and towards zero for voice-band data signals. The parameter is derived from a measure of the rate-of-change of the difference signal values.

7. Adaptive predictor and reconstructed signal calculator

The primary function of the adaptive predictor is to compute the signal estimate $s_e(k)$ from the quantized difference signal $d_q(k)$. Two adaptive predictor structures are used, a sixth order section that models zeroes and a second order section that models poles in the input signal. This dual structure effectively caters for the variety of input signals that might be encountered.

8. Tone and transition detector

In order to improve performance for signals originating from frequency shift keying modems (FSK) operating in the character mode, a two-step detection process is defined. First, partial band signal detection is invoked so that the quantizer can be driven into the fast mode of adaptation. In addition, a transition from a partial band signal is defined so that the predictor coefficient can be set to zero and the quantizer can be forced into the fast mode of adaptation.

Decoder schematic:

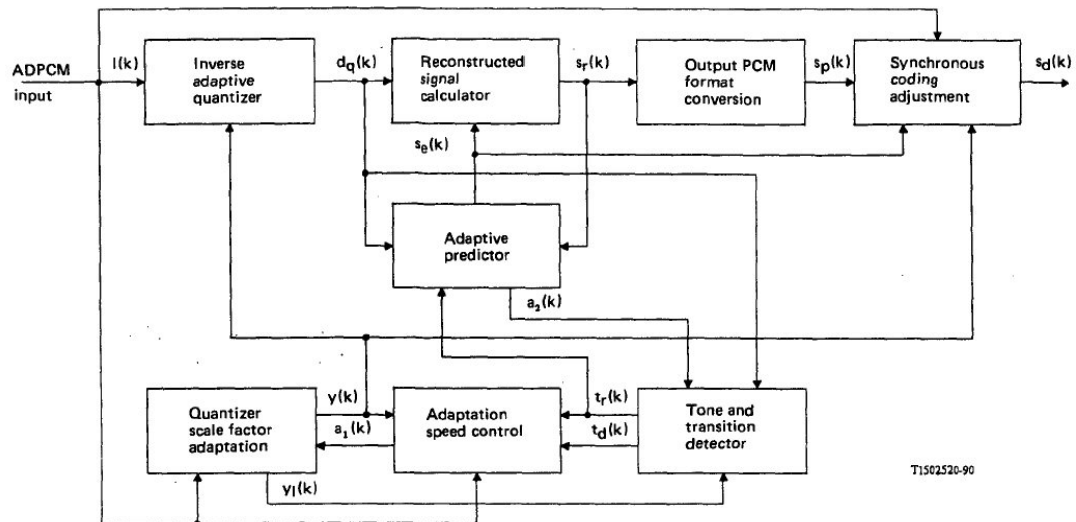


Figure 3.4:5 – ADPCM decoder [25]

Decoder principles [25]:

Most blocks in the decoder are the same as those in the encoder, and the functions are described in the encoder section. Only the blocks that differ from the encoder are described in this section.

1. Output PCM format conversion

This block converts the reconstructed uniform PCM signal $s_r(k)$ into an A-law or μ -law PCM signal $s_p(k)$ as required.

2. Synchronous coding adjustment

This function prevents cumulative distortion occurring on synchronous tandem coding.

3.4.4. Implementation of noise module

As proposed in the analysis component of the thesis, a noise module for adding errors to the different codecs was implemented. We wanted to introduce error and measure the effects. The main source of error in an end-to-end PCM voice-band channel is bit error and packet loss. BER for the different codecs are computed and plotted versus SNR for an AWGN channel.

3.4.5. The functionality of the MatLab program

A simple text based menu appears when running the MatLab program. Then the user can either choose to view the quantization noise for the different PCM modes, the effect of SNR on BER, or run the codec.

Menu Selection guide

- 1) View the quantization noise for different PCM modes
- 2) View the effect of SNR on BER
- 3) Run the codec

Enter your choice : 1

Figure 3.4:6 – Menu selection guide

When 1, 2 or 3 located within the menu are selected, the users will get the opportunity to choose the number of steps for quantization.

```
Enter the number of steps for quantization : 32
```

Figure 3.4:7 – Number of quantization steps

When choice 3 in the menu is selected, and the number of steps for quantization is entered, a new menu will be displayed. In this menu the user can choose which codec to use when coding the wav-file. The selected codec will start encoding the input-file and the quantized signal will appear in a window when the file is encoded.

```
Mode of encoder
```

```
Mode 1 : Uniform PCM
```

```
Mode 2 : A Law PCM
```

```
Mode 3 : Mu Law PCM
```

```
Mode 4 : DPCM
```

```
Mode 5 : ADPCM
```

```
Enter the mode of operation : 1
```

Figure 3.4:8 – Mode of encoder

The encoded samples are saved into a MatLab file called samples.mat. This file is passed to the decoder, and the decoding process can start. After decoding the file, figures of the input-signal and the decoded signal are displayed. The program will also save the decoded samples into a new wav-file. The user can then play this file and compare it with the original file.

3.5. Implementation of Visual Basic application

Visual Basic was used to record and send sound to the Rabbit trainer. Initially the program should have been sending in real-time, but due to lack of time it was chosen to record to a file, before sending the file and playing it in “real-time” at the receiver, Rabbit.

The implementation process were divided into several steps, these steps were;

- Play a wav file with use of GUI in VB.
- Send a wav file through a buffer to the pc speaker.
- Send sound from microphone to pc speaker.
- Send a wav file through a buffer over UDP (localhost) to the pc speaker.
- Send sound from microphone through a buffer over UDP to the pc speaker.
- Send sound from microphone through a buffer over UDP to the Rabbit speaker.

Before trying to send a file, it was sent a text string to the Rabbit trainer; this was done to test if the UDP socket where functioning in both ends. The extension of this was to send a sound file in real-time over the same socket. The first obstacle was to send sound directly through a buffer to the UDP socket (streaming), so it was decided to save the recorded signal to a file before sending the file. Winsock was used to send the file to the Rabbit; this is an easy and fast way to implement a UDP or a TCP socket in a VB application. See *Appendix D – Dynamic C Code* for the code implemented on the Rabbit. When adding the Winsock to the application, one set the host and remote IP address, remote port and binds the local port for listening

to incoming datagrams. Illustration of the application is given in *Figure 3.5:1 – Screenshot of VB application*.

Further a flowchart of the application is provided in *Figure 3.5:2 – Flowchart for VB application*

Functionality of the VB application:

The application is in run-mode when the program have set all necessary IP's, ports and opened a new wav, so that it's ready to record from microphone to memory.

Basically when the program is running the only choices remaining are Record, Choose File (to send), Statistics and Exit. The other buttons will become available when you need them. When recording, it uses the `mciSendString` function to record to memory, if an error occurs during recording, the program will end the recording. The Play function only plays the last recorded speech/song from memory, and not the chosen file you want to send.



When one have chosen a file to send and press the Send File button, the application automatically sends the file to a set IP on a set port. If one needs to change the receiver one should do so in VB code. When pressing Send File, a Sent or Error message box will appear, depending on the outcome. An illustration of the application is shown in *Figure 3.5:1*.

In the VB code it is possible to adjust the sample rate to 11025 (low quality), 22050 (medium quality) or 44100 (high quality, CD). One can also change how many bits per sample wanted, 16 or 8, and how many channels wanted to record from, 1 (mono) or 2 (stereo).

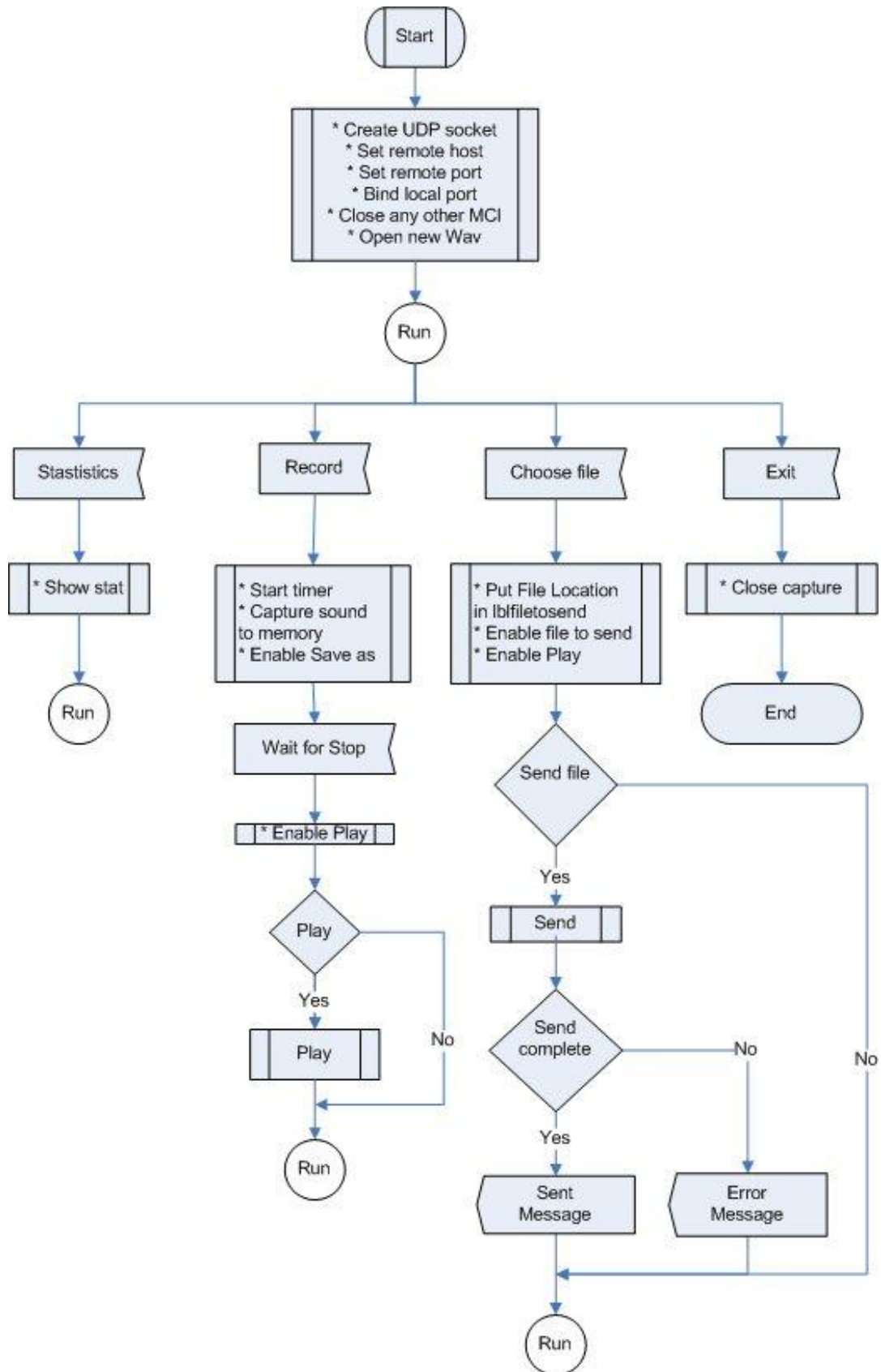


Figure 3.5:2 – Flowchart for VB application

Chapter 4 - Experiments

4.1. Numerical experiments

4.1.1. Test motivation

One of our goals with the master thesis is to run different experiments and to form comparisons with the theoretical background material available. This part of the testing is a numerical MatLab experiment with the implemented audio coding algorithms. The experiments are simulated in such a way so that they will show relevance to the use of codecs in VoIP or in an embedded system environment.

4.1.2. Simulation environment

The experiments were performed in MatLab. MatLab was chosen since all the codecs were implemented in this language. MatLab is also a great tool for analysis since it is an integrated technical computing environment that combines numeric computation, advanced graphics and visualization.

4.1.3. Experiment 1 – Quantization noise vs. Quantization levels

Since quantization noise will vary according to the different audio coding algorithms, it will be interesting to investigate the error in the quantization process for each algorithm. In order to recognise quantization noise, simulations with different quantization levels will be performed. 128, 64, 32, 16 and 8 quantization levels for the algorithms are chosen for the experiment. This will give us different errors in quantization for each simulation and the possibility to observe the effects of this noise pertaining to the levels.

4.1.4. Experiment 2 – BER vs. SNR

This experiment investigates the methods of modulation for the different audio coding algorithms by simulating a system through an AWGN channel. For the AWGN channel no distortion or effects other than the additive white gaussian noise is assumed. With this experiment we will see the effects of introduced errors for the respective algorithms. The outcome of the experiment considers the probability of bit error, as a function of SNR.



Figure 4.1:1 – AWGN Noise filter

4.1.5. Experiment 3 – SNR performance of coders

In order to evaluate the quality of the different audio coding algorithms implemented in MatLab, a comparative analysis of SNR of each algorithm was performed. For this experiment, SNR is calculated using *Equation 4.1:1 – SNR calculation*. The SNR vs. bit rate for Uniform PCM, A-law PCM, μ -law PCM and DPCM are examined in this experiment.

$$SNR = 10 * \log \left(\frac{\sum (Y_{norm})^2}{\sum (q_{noise})^2} \right)$$

Equation 4.1:1 – SNR calculation

4.1.6. Simulation issues

Taking into account the time factor involved in the encoding and decoding of the audio-coding algorithms, only a portion of the file was used in the simulations (as opposed to encoding and decoding the whole inputted wav-file.) Approximately 15 hours were spent to encode/decode the whole wav-file for a specific codec on our laptops. Therefore, 1000 characters of the wav-file were used in the simulations. Nevertheless, this should be sufficient to give simulation results with enough data to analyse the experiments.

4.2. Performance of subjective audio quality test

To perform a subjective audio quality test eight people were gathered to function as panellists. They were told to rate each played sample according to its audio quality (see *Table 4.2:4 – Quality scale*). Before commencing the test, for each of the speech samples and in each music genre, best and worst quality samples were played as models to the panel so as to give them an idea of what to expect in the various divisions. Errors were also introduced to the speech samples and played to the panel. The forms the panellists used for the test is shown in *chapter 1 of Appendix E* and the results are presented in *chapter two*. The results are also presented as graphs in *5.5 Subjective assessment of audio quality*.

For both music and speech, the music as well as the speech samples where played in random order, and the speech samples with introduced errors were played in amongst the other samples. If one or more of the panellists so wished, the samples were repeated.

4.2.1. Music samples

For the music part of the listening test, samples from three different genres (trance, pop and classical) were taken. To represent trance we used an extract from Paul Van Dyk – *We're Alive*, for pop we used Smashing Pumpkins – *Disarm* and for classical we used Johann Sebastian Bach – *Rejouissance*. Each sample lasted 10 seconds and we made samples of the following codecs and bit rates as listed in *Table 4.2:1*. This table also justifies why this rating is expected.

Table 4.2:1 – Expected quality rating for music samples

Codec:	Expected quality rating:	Justification of expected quality rating:
G.711 PCM:		
PCM A-law 64 Kbps	3	Simple codec, medium bit rate, speech
PCM μ -law 64 Kbps	3	Simple codec, medium bit rate, speech
G.726 ADPCM:		
ADPCM 40 Kbps	2	Simple codec, low bit rate, speech
ADPCM 32 Kbps	2	Simple codec, low bit rate, speech
ADPCM 24 Kbps	1	Simple codec, lowest bit rate, speech
ADPCM 16 Kbps	1	Simple codec, lowest bit rate, speech
Mp3:		
Mp3 320 Kbps	5	Advanced codec, high bit rate, music
Mp3 128 Kbps	5	Advanced codec, high bit rate, music
Mp3 80 Kbps	4	Advanced codec, medium bit rate, music
Mp3 32 Kbps	2	Advanced codec, low bit rate, music
WMA:		
WMA 320 Kbps	5	Advanced codec, high bit rate, music
WMA 128 Kbps	5	Advanced codec, high bit rate, music
WMA 80 Kbps	4	Advanced codec, medium bit rate, music
WMA 32 Kbps	2	Advanced codec, low bit rate, music
Ogg-Vorbis:		
Ogg 160 Kbps	5	Advanced codec, high bit rate, music

4.2.2. Speech samples

A source file of a speech sample, with duration of 4.5 seconds, was downloaded [34]. The sample file contained both a male and female voice. From the source file, we produced samples with an expected quality rating as listed in *Table 4.2:2*. Samples with introduced errors were also included, and all expected quality ratings are justified in the table presented below.

Table 4.2:2 – Expected quality rating for speech samples

Codec:	Expected quality rating:	Justification of expected quality rating:
G.711 PCM:		
PCM μ -law 64 Kbps	5	Designed for speech, used in phones
PCM A-law 64 Kbps	5	Designed for speech, used in phones
PCM A-law 64 Kbps With Error	4	Designed for speech, used in phones, introduced error
PCM A-law 64 Kbps With Error Burst	4	Designed for speech, used in phones, introduced error burst
G.726 ADPCM:		
ADPCM 40 Kbps	4	More advanced than PCM, low bit rate
ADPCM 32 Kbps	3	More advanced than PCM, low bit rate
ADPCM 24 Kbps	2	More advanced than PCM, low bit rate
ADPCM 24 Kbps With Error	2	More advanced than PCM, lowest bit rate, introduced error
ADPCM 24 Kbps With Error Burst	2	More advanced than PCM, lowest bit rate, introduced error burst
ADPCM 16 Kbps	1	More advanced than PCM, lowest bit rate
G.728 LD-CELP:		
LD-CELP 16 Kbps	1	More advanced than ADPCM, lowest bit rate
Mp3:		
Mp3 48 Kbps	3	Advanced codec, high/medium bit rate
WMA:		
WMA 48 Kbps	3	Advanced codec, high/medium bit rate

4.2.3. Errors added to speech samples

It was made one sample with bit errors and one with a burst of errors of respectively ADPCM 24 kbit/s and PCM A-law 64 kbit/s. The amount of errors added to each sample is shown in *Table 4.2:3*:

Table 4.2:3 – Errors added to speech samples

Sample:	Bit errors		Error burst	
	Total errors (sek)	%	Burst size (sek)	%
PCM A-law 64 kbit/s	0.300	6.7	0.302	6.7
ADPCM 24 kbit/s	0.243	5.4	0.237	5.3

4.2.4. Quality scale

When rating the music and speech samples the panellist's were told to use the scale as represented in *Table 4.2:4*.

Table 4.2:4 – Quality scale

Rating:	Explanation:	
1	Bad	Very annoying distortion which is objectionable
2	Poor	Annoying distortion but not objectionable
3	Adequate	Perceptible distortion that is slightly annoying
4	Good	Slight perceptible level of distortion but not annoying
5	Excellent	Imperceptible level of distortion

Chapter 5 - Results

In this chapter, results of the implemented codecs; results of the numerical experiments in MatLab; and the results of the subjective assessments of audio quality will be provided.

5.1. G.711 PCM

The results of the implementation of the codecs are presented as figures of the original signal and the decoded signal. This form of presentation gives an indication of the codecs ability to reconstruct the original signal. But because the quantized values are only approximates, it is impossible to recover the same signal which was sent. Graphs of input signal and decoded signal for uniform PCM, A-law PCM and μ -law PCM are presented. Quantization noise for these PCM modes is also visualized by figures.

5.1.1. Uniform PCM

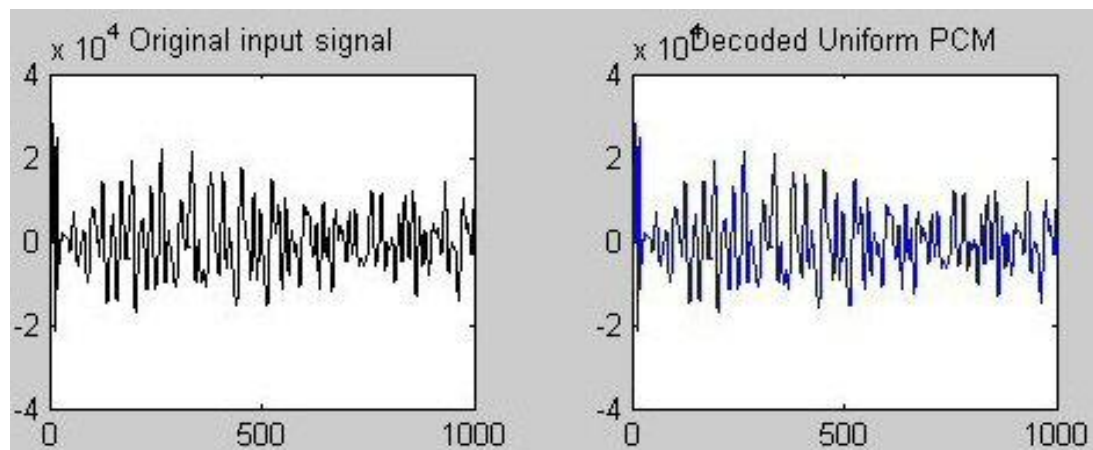


Figure 5.1:1 – Input and output for decoded uniform PCM

The wav-file decoded with 128 quantization levels, gives an output signal relatively close to the original input signal. The quantized uniform PCM shows that the quantization changes the continuously ranging signal to a range between +1 and -1. The sampled values are rounded off to the pre-determined discrete levels by the quantizer. The error occurred from the rounding off of the quantization is also presented in the graph as quantization noise.

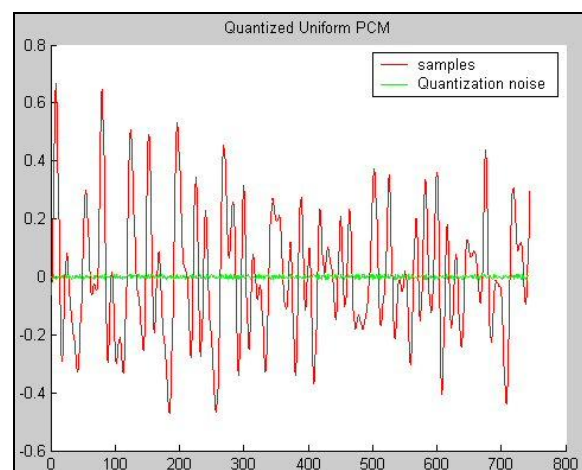


Figure 5.1:2 – Uniform PCM quantized

5.1.2. A-law PCM and μ -law PCM

The figures below illustrate the results of decoded A-law and μ -law PCM.

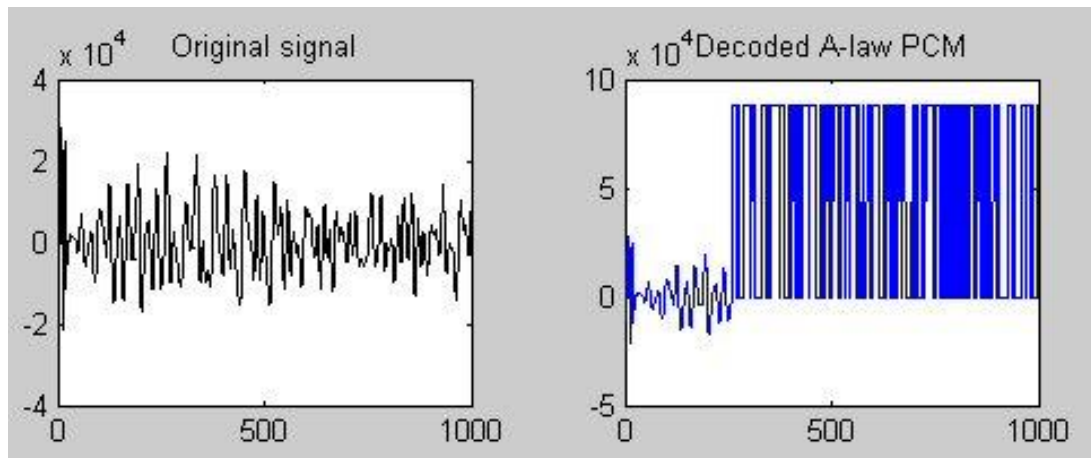


Figure 5.1:3 – Input and output for decoded A-law PCM

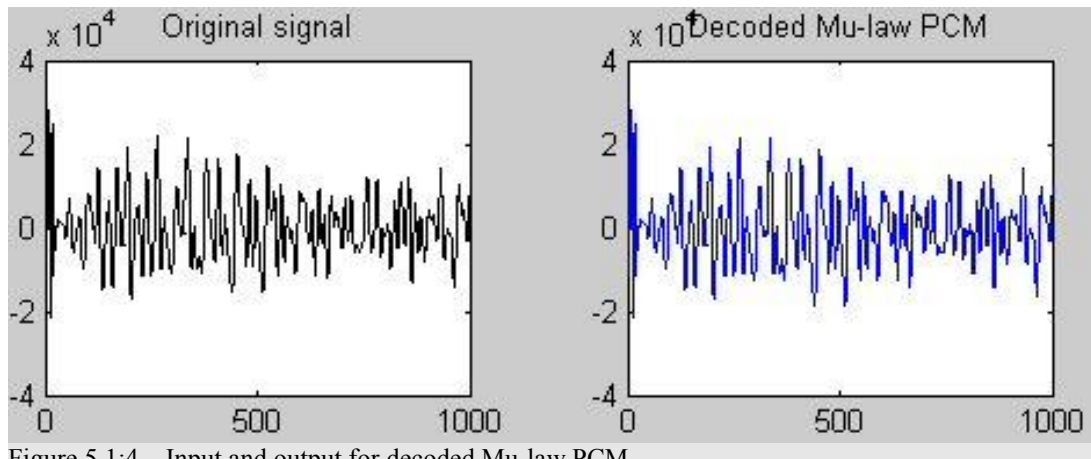


Figure 5.1:4 – Input and output for decoded Mu-law PCM

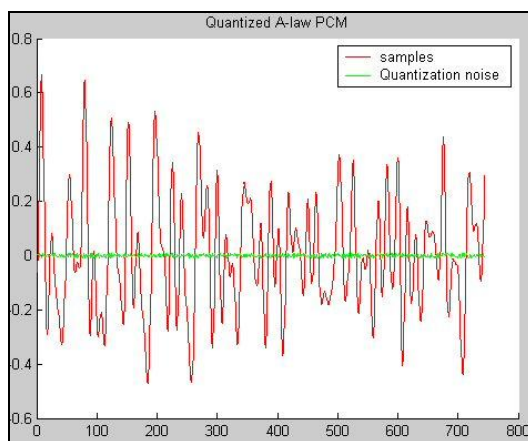


Figure 5.1:5 – Quantized A-law PCM

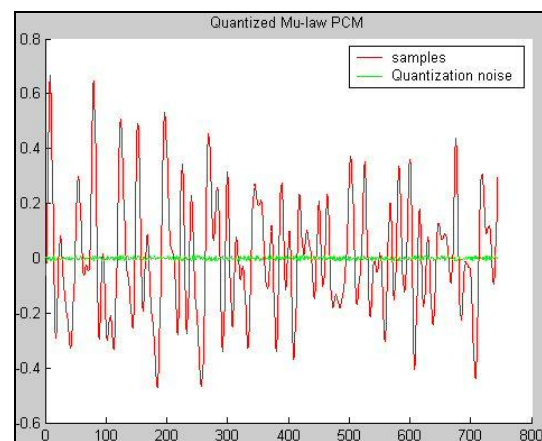


Figure 5.1:6 – Quantized Mu-law PCM

The A-law PCM decoding yielded an unexpected result, as the decoded version was quite different from the original signal, which suggested a coding error in the A-law decoder. There was insufficient time to find the source of this error. The μ -law decoding gave a result similar to that of the original signal.

5.1.3. The effects of compression algorithms

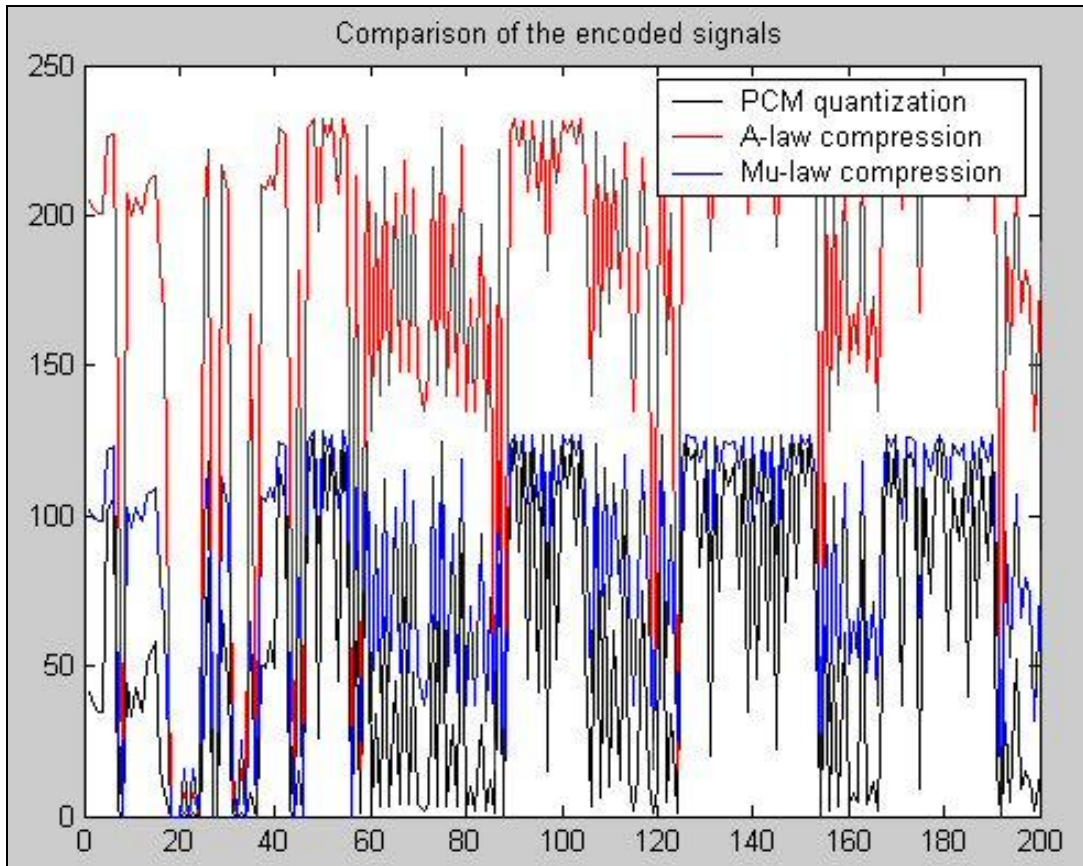


Figure 5.1:7 – Comparison of quantization and compression

Both PCM A-law and μ -law encoding use companding. The way the signal is compressed is illustrated in *Figure 5.1:7*, and the compression difference can be observed. The A-law compression algorithm amplifies all signals compared to the PCM signal. On the other hand it can be observed that the μ -law compression algorithm amplifies more low level than high level signals.

5.2. DPCM

This section covers the results from the DPCM implementation. Original and DPCM decoded signals are displayed, and a figure of the quantized DPCM is also presented.

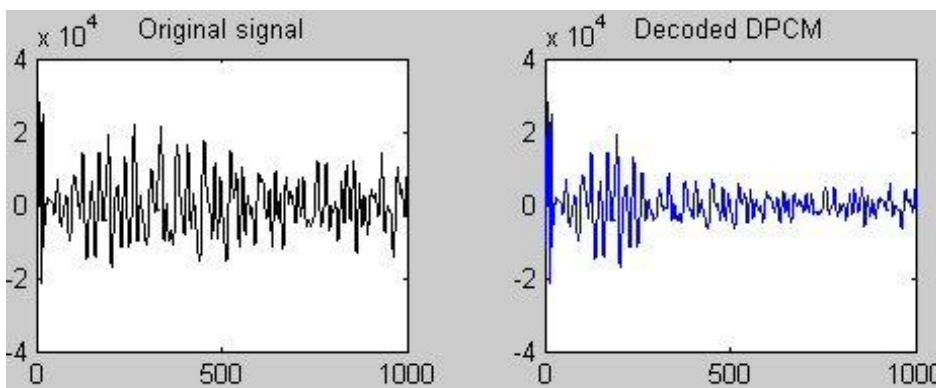


Figure 5.2:1 – Input and output for decoded DPCM

The DPCM algorithm gives an output decoded signal which is decreasing in range from 250 samples and above. The decoded DPCM wav-file was played after decoding, and the sound level of this file was much lower than the original wav-file. The difference in signal between each sample has been quantized with a predictor coefficient of $\alpha = 0.45$. The predictor transfer function, in its simplest form, simply create a weighted copy of the previous sample, i.e. $P(z) = \alpha z^{-1}$, where α is in the range of 0.9 for highly correlated sub-bands. In this way, only the prediction error signal needs to be quantized, and therefore this prediction error naturally has a smaller dynamic range than the signal itself.

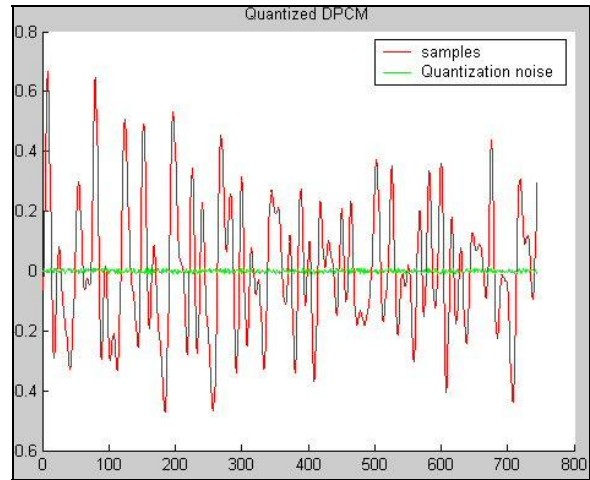


Figure 5.2:2 – Quantized DPCM

5.3. G.726 ADPCM

The implemented ADPCM algorithm, gives the user the possibility to choose between the following bit rates: 40, 32, 24 and 16 (kbit/s), all these rates were implemented. The results of 32kbit/s ADPCM are presented below.

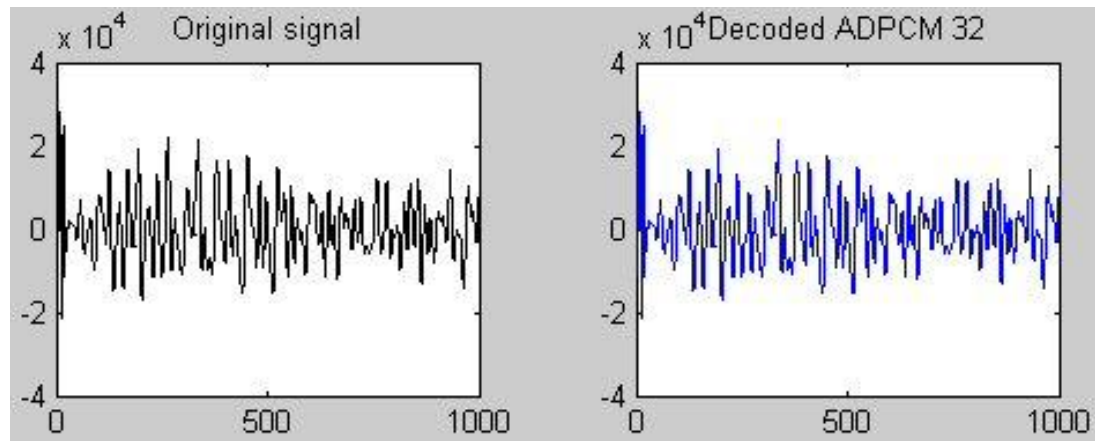


Figure 5.3:1 – Input and output for decoded ADPCM 32kbit/s

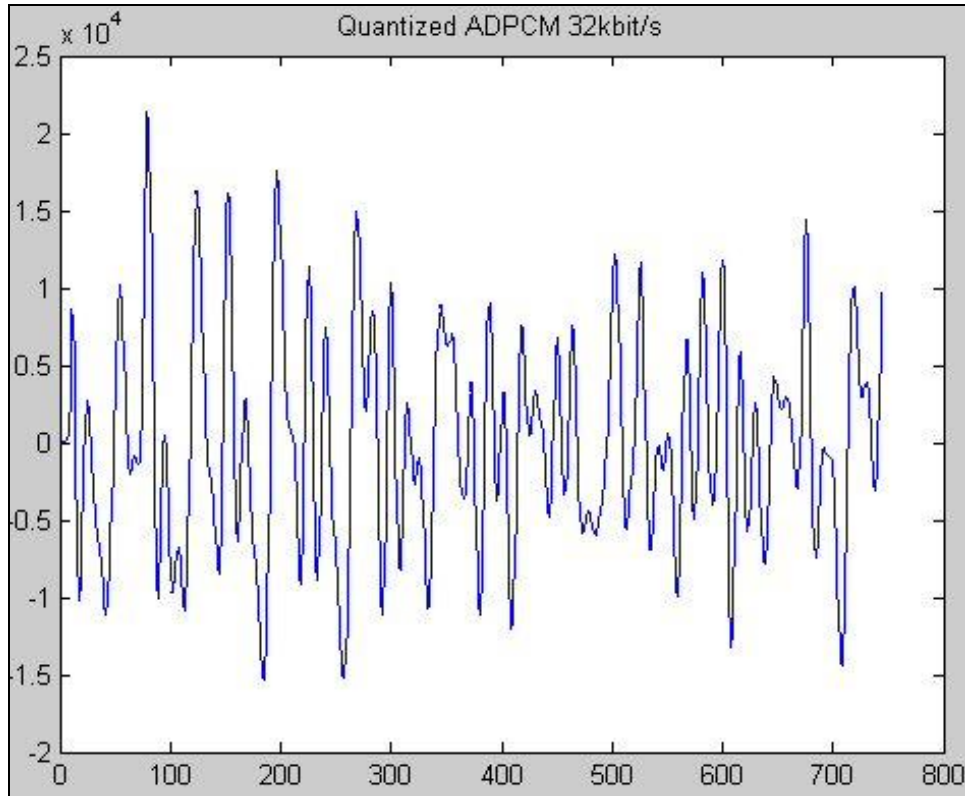


Figure 5.3:2 – Quantized ADPCM 32 kbit/s

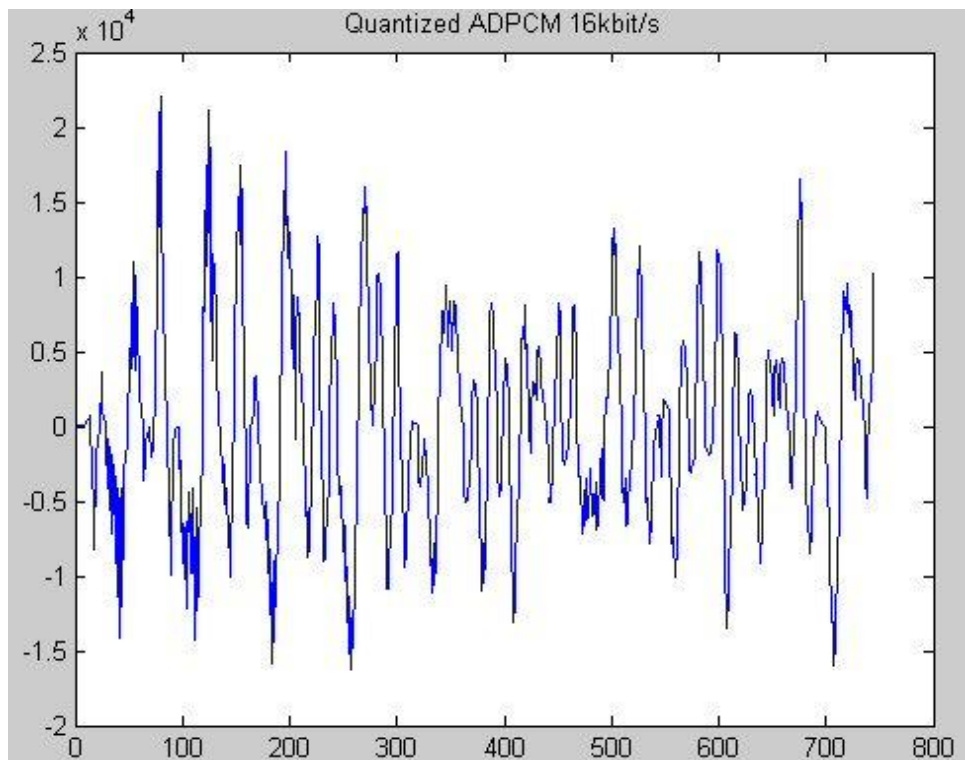


Figure 5.3:3 – Quantized ADPCM 16 kbit/s

Encoding and decoding of ADPCM were conducted for the above mentioned rates. The results, *Figure 5.3:2* and *Figure 5.3:3*, show that lowering the bit rates increased the amount of distortion.

5.4. Numerical experiments in MatLab

For the figure of quantization error for the respective PCM modes, 128 numbers of quantization levels are used. In order to investigate and evaluate the relation between quantization noise and the number of quantization levels, simulations with different quantization levels were run. The numbers of quantization levels can be selected from the 2^n formula, where n is the number of bits. So in the simulation of experiment 1, the numbers of quantization levels of 64, 32, 16 and 8 are used.

5.4.1. Quantization error for the different PCM modes

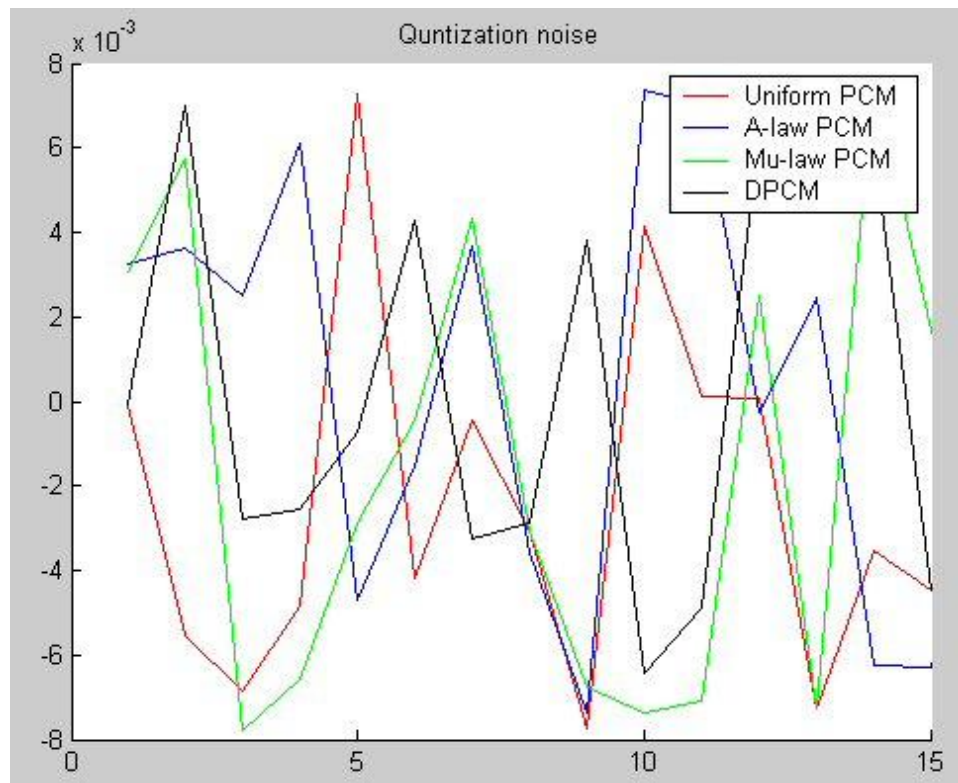


Figure 5.4:1 – Quantization error for all PCM modes

In the MatLab program, it is possible to view the quantization noise for all the different PCM modes. The results are presented in *Figure 5.4:1*. As shown in the figure, the quantization noise for the various PCM modes differs for each mode.

Since the quantization process leads to approximates of the input signal with the detected signal having some deviation in amplitude, this deviation gives rise to distortion. In order to better present the different quantization error, due to the number of quantization levels, experiment 1 was performed. The MatLab code below, is used to calculate the quantization noise.

MatLab code:

```
% calculation of quantization noise  
quantized_samples = round(Y / delta) * delta ;  
quantisation_noise(1,i) = Y - quantized_samples ;
```

5.4.2. Experiment 1 – Quantization noise vs. Quantization levels

The figures below illustrate the results collected using different quantization levels.

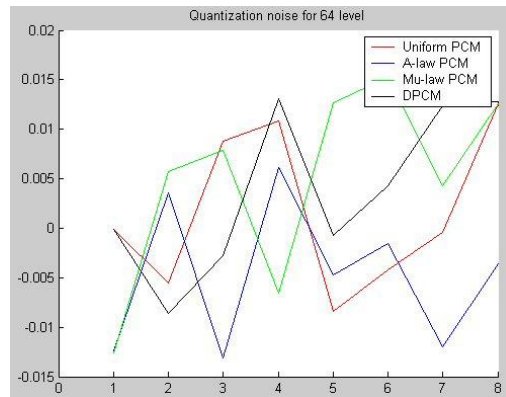


Figure 5.4:2 – Quantization noise for 64 level

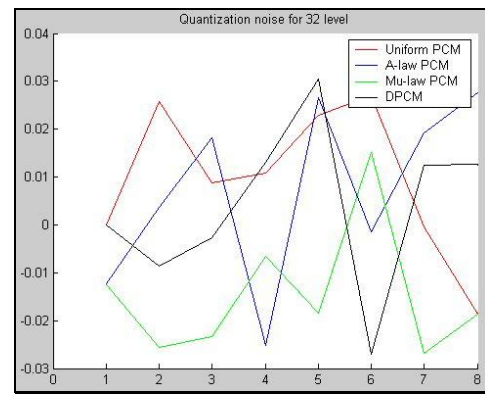


Figure 5.4:3 – Quantization noise for 32 level

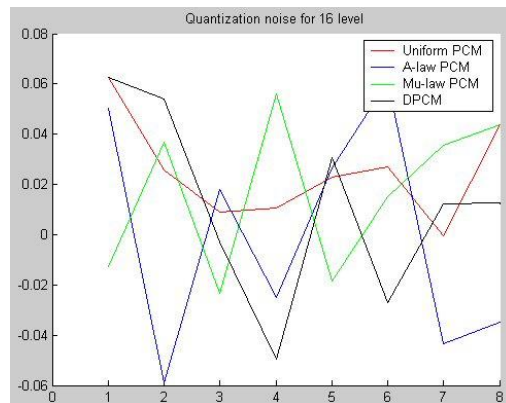


Figure 5.4:4 – Quantization noise for 16 level

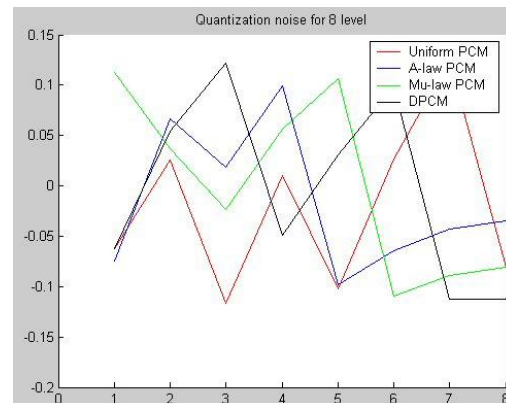


Figure 5.4:5 – Quantization noise for 8 level

The quantization noise which occurs while encoding the wav-file is in fact the error from the process of quantization itself. Here the samples are rounded off to the nearest quantization level, predefined by the number of quantization levels. The difference between the original sample and the encoded sample is known as the quantization error. Increasing the number of quantization levels reduces the amount of quantization noise, choosing a wider dynamic range therefore providing a more accurate result.

Observed from the figures above, the range of the quantization noise is increased from the range between 0.02 to -0.015 for 64 levels towards 0.15 to -0.2 for 8 levels. This observation indicates that for each decrease of quantization levels (64 – 32 – 16 – 8) the range of quantization noise appear to double. The quantization noise is in inverse ratio to the number of quantization levels.

5.4.3. Experiment 2 - BER vs. SNR

Simulations were run in order to investigate the effect of introduced error to the different codecs. The error is introduced randomly in an AWGN channel. Due to the shape, the plot of BER as a function of SNR, is called a “waterfall curve”. This waterfall curve is used to indicate the performance of the codecs in experiment 2.

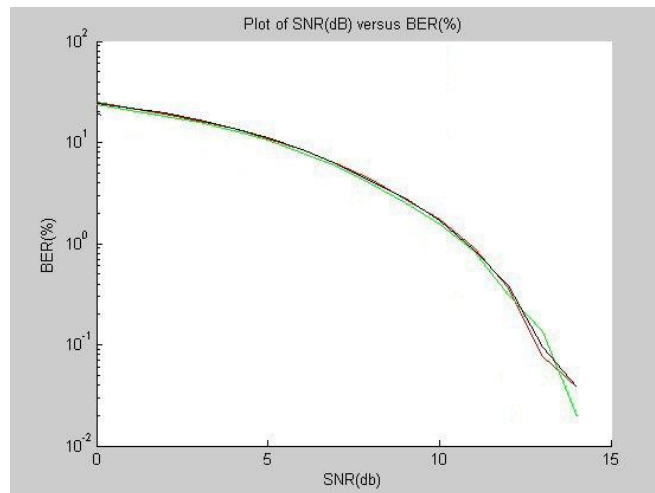


Figure 5.4:6 – BER as a function of SNR

Figure 5.4:6 shows BER as a function of SNR in the AWGN channel.

The failure to meet perfect performance is measured by the bit-error-rate (BER), which is computed for the PCM modes using MatLab, and displayed as a function of SNR. The graph establishes a close relationship between SNR and the amount of error. An increase in SNR would reduce the probability of error significantly. The probability of BER draws near to 0 around 15 dB SNR.

5.4.4. Experiment 3 – SNR performance of coders

Speech coders aim at reducing the bit rate, while introducing as little perceptual distortion as possible into the speech signal. Table 5.4:1 shows the decrease of SNR as the bit rate is reduced.

Table 5.4:1 – SNR performance

Bit rate:	64 kbit/s	56 kbit/s	48 kbit/s	40 kbit/s	32 kbit/s
Codec:	SNR (dB):	SNR (dB):	SNR (dB):	SNR (dB):	SNR (dB):
1. U-PCM	43.61	32.58	20.56	16.07	1.44
2. PCM A	35.93	35.93	35.93	14.21	8.05
3. PCM μ	40.59	39.39	22.39	12.21	10.89
4. DPCM	29.43	41.29	33.36	19.03	8.11

As observed in the table above, SNR performance is affected by the bit rate. The main observation is the considerable drop in SNR when reducing the bit rate from 40 to 32 kbit/s.

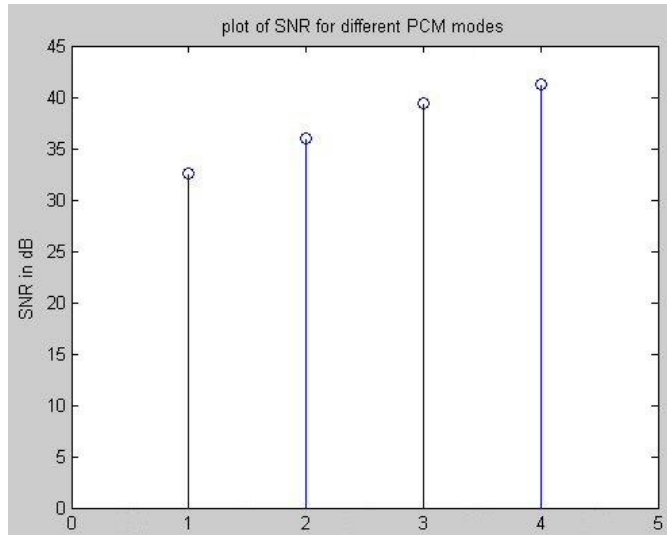


Figure 5.4.7 – SNR performance for 128 levels

Figure 5.4.7 – SNR performance for 128 levels, shows the plot of SNR performance for the different PCM modes, for 128 quantization levels. 1: Uniform PCM, 2: A-law PCM, 3: μ -law PCM and 4: DPCM.

5.5. Subjective assessment of audio quality

Table 5.5:1 – Comparison of average quality rating

Codec:	Average quality rating (Trance):	Average quality rating (Pop):	Average quality rating (Classical):	Total Average:	Expected quality rating:
G.711 PCM:					
PCM A-law 64 Kbps	2,0	1,8	3	2,3	3
PCM μ -law 64 Kbps	2,4	1,5	2,9	2,3	3
G.726 ADPCM:					
ADPCM 40 Kbps	2,2	1,5	2,6	2,1	2
ADPCM 32 Kbps	2,8	1,5	2,8	2,4	2
ADPCM 24 Kbps	1,7	1,4	2,1	1,7	1
ADPCM 16 Kbps	1,1	1,1	1,8	1,3	1
Mp3:					
Mp3 320 Kbps	3,3	4,4	4,4	4,0	5
Mp3 128 Kbps	3,9	4,6	4,5	4,3	5
Mp3 80 Kbps	3,7	3,5	3,9	3,7	4
Mp3 32 Kbps	1,9	1,7	2,8	2,1	2
WMA:					
WMA 320 Kbps	4,2	4,8	4,6	4,5	5
WMA 128 Kbps	4,1	4,6	4,4	4,4	5
WMA 80 Kbps	4,1	4,5	4,0	4,2	4
WMA 32 Kbps	3,0	3,3	3,5	3,3	2
Ogg-Vorbis:					
Ogg 160 Kbps	3,6	4,6	4,5	4,2	5

Table 5.5:1 and Figure 5.5:1 present the combined average quality rating from each music genre, and an expected quality rating for each audio codec. The expected quality rating is a comparison of the sound samples themselves, and the actual bit rate, as we assume they should perform.

Table 5.5:1 and Figure 5.5:1 also indicate that the expected quality rating was, for most of the codecs, not far from the total achieved average quality rating.

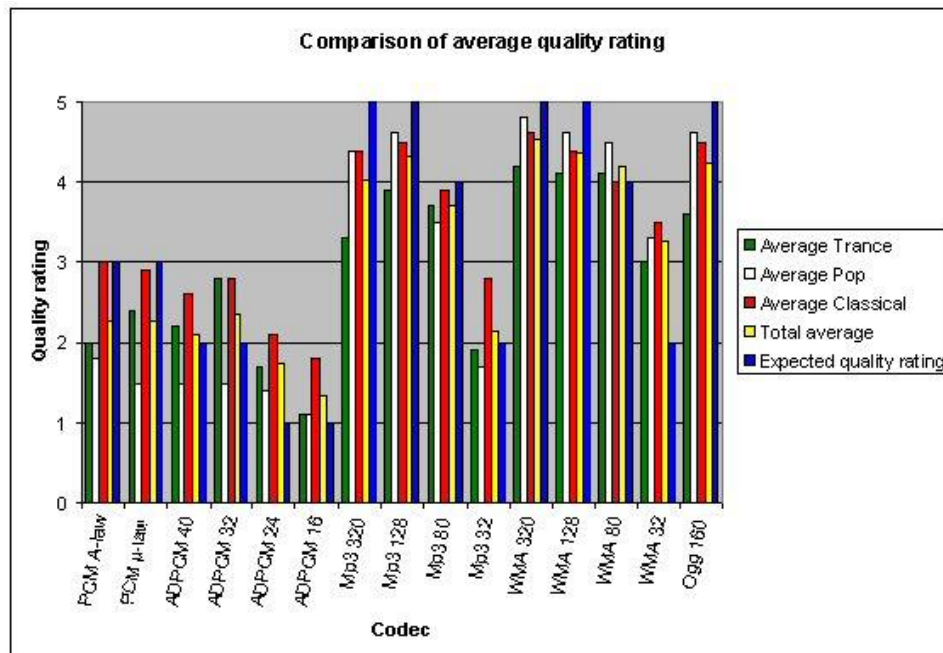


Figure 5.5:1 – Comparison of average quality rating

Figure 5.5:1 – Comparison of average quality rating presents a comparison of each genre’s quality rating, the total average and the expected quality rating.

G.711 – PCM

One can see that the A-law and μ-law algorithm has an equal total average, although it is less than expected. It is also showing that there is a clear difference between the Pop and Classical music genre. This could be attributed to several factors. We will look at some of those factors in *the discussion chapter*.

G.726 – ADPCM

One can see that the codecs perform almost as expected, but also in ADPCM as PCM the classical genre gets a higher rating than the pop genre. Furthermore, it is evident that the audience has rated ADPCM 32 higher than ADPCM 40.

Mp3

From the Mp3 codec one can see that it is the trance genre that gets the lowest rating and that both pop and classical are rated almost the same. One also see that Mp3 128 is rated slightly higher than Mp3 320 and that Mp3 80 is relatively good when compared to Mp3 128 and 320.

WMA

As illustrated in the table and graph above, one notices that WMA gets higher overall ratings in all genres than Mp3, and that as Mp3, the trance genre gets the worst rating. The difference between WMA 80 and WMA 320 is low in comparison to the expected rating.

Ogg-Vorbis

For Ogg-Vorbis the trance genre gets rated lower than the pop and classical genre.

Summary

It is interesting to observe that although most of the codecs performed as expected, the ratings for classical music are higher for all codecs with the exception of WMA and Ogg-Vorbis. Furthermore one see that the music codecs where rated higher than the speech codecs, even for those with lower bit rate than the speech codecs.

5.5.1. Music samples

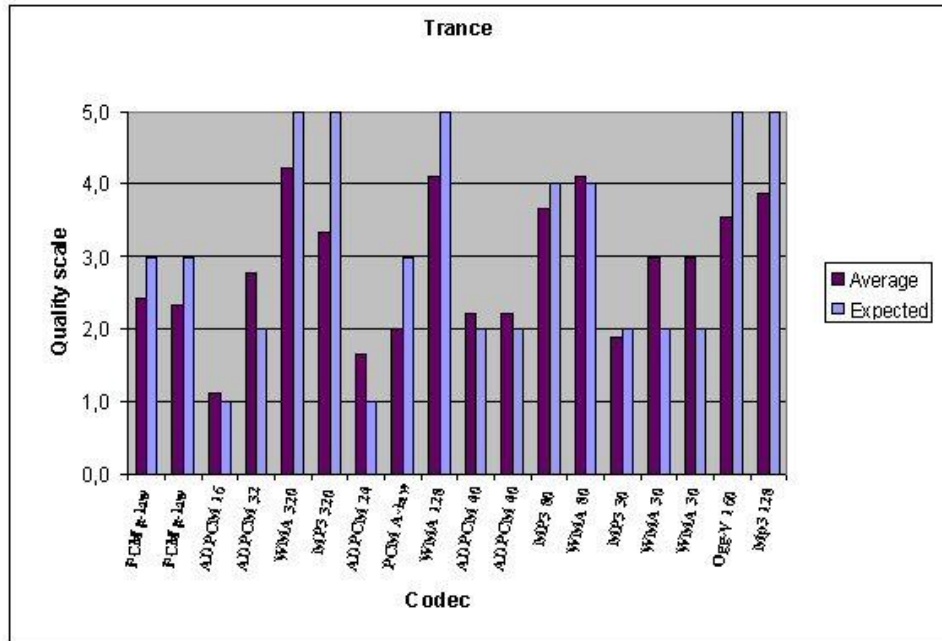


Figure 5.5:2 – Results from the Trance genre

Figure 5.5:2 presents the measured average quality rating and the expected quality rating from the Trance genre in the subjective assessment of sound quality test.

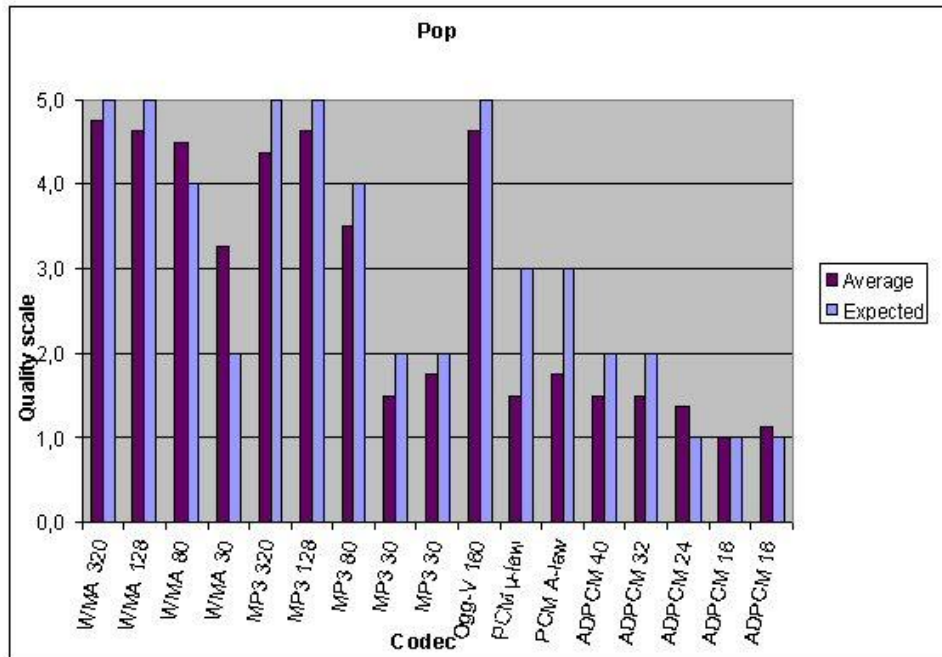


Figure 5.5.3 – Results from the Pop genre

Figure 5.5.3 presents the measured average quality rating and the expected quality rating from the Pop genre in the subjective assessment of sound quality test.

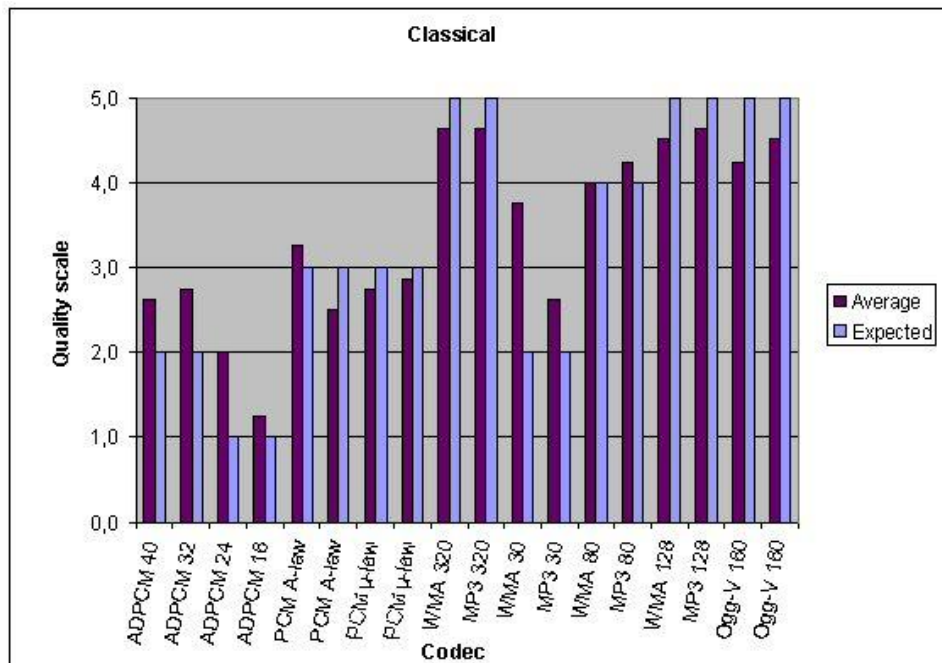


Figure 5.5.4 – Results from the Classical genre

Figure 5.5.4 presents the measured average quality rating and the expected quality rating from the Classical genre in the subjective assessment of sound quality test.

5.5.2. Speech samples

Table 5.5:2 – Test of Speech samples with audience

Codec:	Average quality rating:	Expected quality rating:
G.711 PCM:		
PCM μ -law 64 Kbps	4,7	5
PCM A-law 64 Kbps	4,4	5
PCM A-law 64 Kbps Error	3,1	4
PCM A-law 64 Kbps Error Burst	4,1	4
G.726 ADPCM:		
ADPCM 40 Kbps	3,7	4
ADPCM 32 Kbps	3,1	3
ADPCM 24 Kbps	2,2	2
ADPCM 24 Kbps Error	1,8	2
ADPCM 24 Kbps Error Burst	2,2	2
ADPCM 16 Kbps	1,1	1
G.728 LD-CELP:		
LD-CELP 16 Kbps	1,2	1
Mp3:		
Mp3 48 Kbps	3,7	3
WMA:		
WMA 48 Kbps	4,3	3

Table 5.5:2 presents all the measured average quality ratings and the expected quality ratings from the speech listening test. The expected quality rating is a comparison of the sound samples themselves, and the actual bit rate, as we assume they should perform.

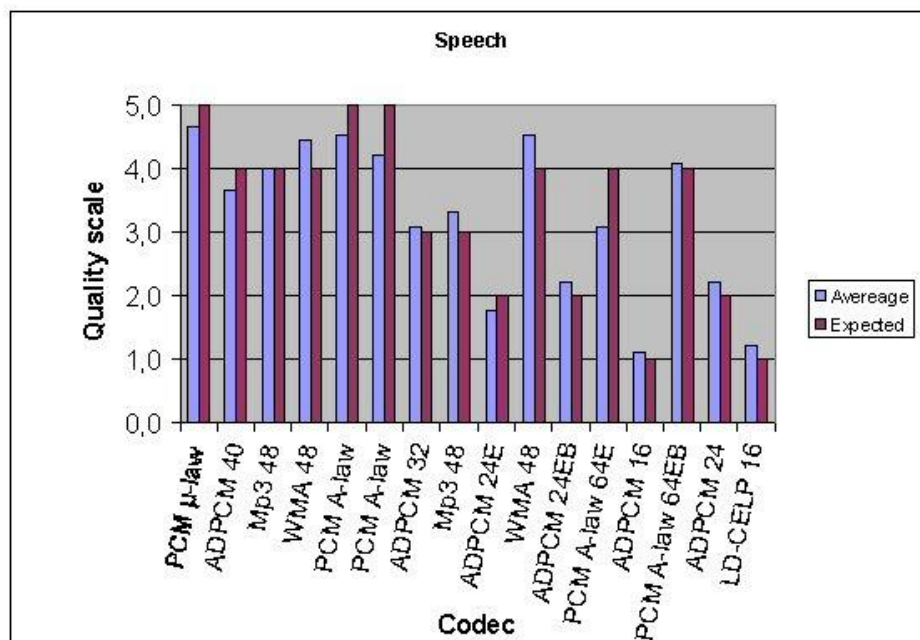


Figure 5.5:5 – Results from the Speech listening test

Figure 5.5:5 presents the measured average quality rating and the expected quality rating from the Speech part in the subjective assessment of sound quality test.

G.711 – PCM

From the table and graph above one notice that the PCM codec performs almost as expected, and those samples containing errors is rated lower than those with a burst of error.

G.726 – ADPCM

One can see that ADPCM performs almost as expected for all the different bit rates of the codec.

G.728 – LD-CELP

As illustrated from the table and graph above, LD-CELP performs almost as expected.

Mp3

One can see that Mp3 is rated just as expected.

WMA

As one can observe from the table and graph above, WMA is rated just above expected.

Summary

On average one can see that most of the codecs are rated as expected. Both Mp3 48 kbit/s and WMA 48 perform surprisingly good compared to PCM A-law and μ -law 64 kbit/s.

5.5.3. Speech samples with introduced errors

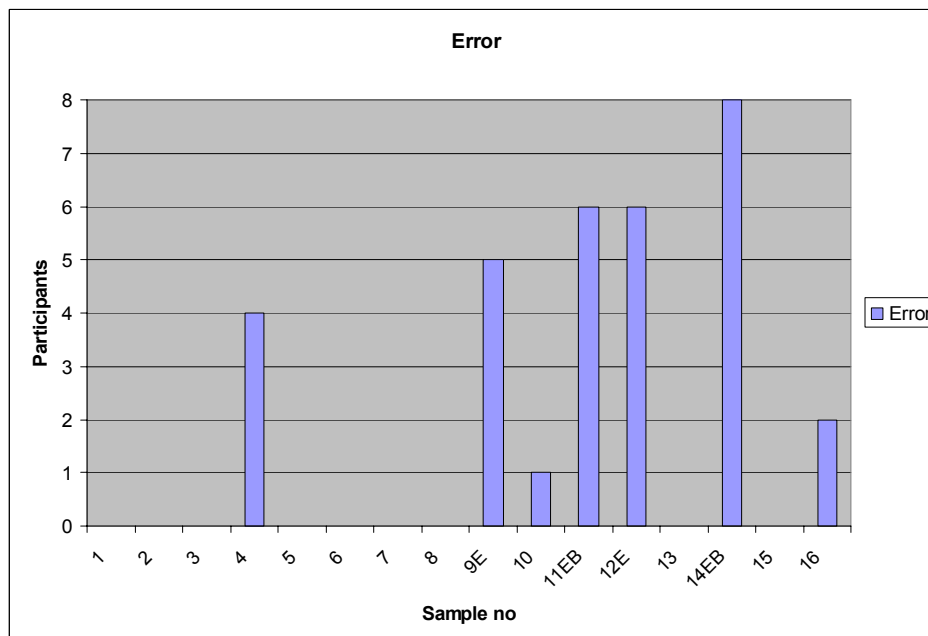


Figure 5.5:6 – Errors found by the participants¹

¹ E on the X-axis indicates that the sample contained errors. EB on the X-axis indicates that the sample contained a burst of error.

Figure 5.5:6 presents the number of participants who perceived an error or an error burst in the played sample. Note that 4 and 10 respectively are the same sample - without introduced errors, and that 16 is LD-CELP without introduced errors.

5.6. Electrical circuitry

The circuitry was tested in the laboratory and showed normal behaviour during performance testing. The tests performed were: input of different frequencies - to see if the LP-filter had the correct cut-off frequency (15 kHz). The second test dealt with the transfer of sound directly from microphone to speaker, this was performed in order to observe that the op-amp did not go into saturation.

One of the original goals was to measure computational complexity of codecs on the Rabbit. Since the codecs were not implemented on the Rabbit in an embedded system, measuring computational complexity due to lines of Kbytes used and complexity of the code could not be performed. However, the codecs were implemented in MatLab, and from the computational complexity of this code we have made an assumption of the complexity of each codec when implemented on an embedded system.

Table 5.6:1 – Computational complexity

Computational complexity:	Codec:
Very low	Uniform PCM, μ -law
Low	A-law, DPCM
Medium	ADPCM, LD-CELP
High	Mp3, WMA, Ogg-Vorbis, Bluetooth Sub-band codec

5.7. Visual Basic application

The steps performed to accomplish transfer of a file to the Rabbit are listed in 3.5 *Implementation of Visual Basic application*.

The first version of the application sent a text string from PC A to PC B. This version had a very simple GUI for the user to follow. In addition, a text string from PC A to the Rabbit was sent (to check that the UDP socket worked). The second version recorded sound from the microphone on the PC and made it possible to save the recorded signal to a file. It is also possible to play the recorded signal before saving it. During the third application, a file (up to 8 Kb) was able to be sent through a buffer over a UDP socket, to a given IP and PORT. Since the codecs were not implemented on the Rabbit it was not possible to see whether the file arrived on the Rabbit or not. Tests from PC A to PC B communication however, showed that it worked, that is, PC B played the signal when it arrived.

Chapter 6 - Discussion

The scope of this thesis was to do an analysis of different audio coding algorithms for networked embedded systems. Original the intention was to implement G.711 A-law and μ -law PCM, G.726 ADPCM and G.728 LD-CELP into an embedded system. Then we wanted to introduce errors and measure the effects. An analysis of all this should then be performed. After some time, it was clear that it was very much to do, if we should proceed with the original plan. Development of an application containing of encoders for all the audio coding algorithms, implementation of the corresponding decoders in the Rabbit processor, development of the electrical circuitry interface to the Rabbit trainer, were to time consuming. Since the scope of the thesis was to analyze the performance of audio coding algorithms, we had to do some adjustments to the original plan.

Along with the supervisor, we decided to implement the audio coding algorithms by themselves in MatLab. If we managed to implement them there, run numerical experiments, and analyze the effects, we would still have results attached to the scope of the thesis. To achieve a more profound analysis of the audio coding algorithms we also added a subjective assessment test to measure the audio quality. These actions would still give us material to analyze audio coding algorithms for networked embedded systems. But even with these adjustments we continued working with the development of a visual basic application and the circuitry interface.

The focus in our following discussion is based on the thoughts and views we have on the implementation of the codecs, the experiment results from numerical test and the subjective audio quality test. Our application and printed circuit board are evaluated as well. Issues and complications which occurred during the project period will be commented.

6.1. The implementation of the codecs

6.1.1. Evaluation methods

In order to evaluate the performance of the implemented audio coding algorithms, comparison of the input signal and the decoded output signal by figures were done. By evaluating the implemented algorithms this way, we can say something about the accuracy of the coding. But, comparison by figures only, is not a good way to estimate the quality. A signal can sound sufficient, even though the comparison of the inn and output signal by figures can differ. With this in mind, we also listened to the decoded wav files, to obtain a sense of the quality.

6.1.2. Possible source of errors

The way the audio coding algorithms are coded in MatLab can give possible errors in the behaviour of the codecs. Error in the MatLab programming code and the assumption of the fact that if to graphs look alike, they actually are.

6.1.3. The results

We were satisfied with the implementation and performance of the different audio coding algorithms. Uniform PCM, μ -law PCM, DPCM and ADPCM were fully implemented and gave results as expected. The A-law PCM compression algorithm in the encoder seems to work. But, the decoded signal of A-law PCM presents some strange results. This is probably caused by some kind of error in the expanding algorithm in the decoder.

Another observation worth mentioning is the decrease of range in the DPCM signal. Listening to the decoded DPCM wav-file also supports this observation. It sounded like the file had decreased in volume, compared to the original wav-file. The DPCM has a simple structure with a first order predictor. This will cause quantization noise accumulation. The stepsize for this codec is very important. Small stepsize results in slope overload distortion and large stepsize results in granular distortion.

However this should not result in that large distortion for our DPCM codec. Due to the theoretical behaviour of the codec, we suggest failure in implementation more likely than feature of codec. If given more time, this would have been investigated further.

6.2. The numerical experiments in MatLab

The experiments in MatLab were performed to analyze the different audio coding algorithms numerically. We wanted to investigate the effects of noise and errors related to the algorithms. Our first experiment considers the difference in quantization noise for the codecs. This is errors which occur in quantization process in the encoders. Experiment 2 considers the relationship between BER and SNR. These types of errors are related to transmission over a channel, e.g. voice over IP.

6.2.1. Evaluation methods

Several simulations with different numbers of quantization steps were run to evaluate difference in quantization noise. The figures from the simulations indicate that the quantization noise differs for the different algorithms, but we were more interested in the relationship between quantization noise and numbers of quantization levels.

For the other experiment simulation over an AWGN channel, with introduction of random errors, was performed. The method used for evaluation of this experiment was to display the BER as a function of SNR.

6.2.2. Possible source of errors

One possible source of error with these experiments is the selection of numbers of samples in the simulation. For experiment 2 the sequence of the wav-file could not be too short. By using a small sequence with few samples the BER versus SNR might not give results accurate enough. On the other hand, due to long simulation time, we had to come up with a sequence not too long, but still would provide an indication of BER vs. SNR.

The random function in the AWGN code should also be considered talking about things that would affect our results. Since the introduction of errors is randomly

distributed, different results could be observed related to where the errors are situated in the wav-stream for different simulations.

6.2.3. Experiment 1 – Quantization noise vs. Quantization levels

The quantization noise decreases in relation with an increasing number of quantization levels. These levels are related to the bit rate. The equation used to find the number of levels is 2^n where n is number of bits.

Example:

$$\begin{aligned}
 n &= 7 \text{ bits} \\
 2^7 &= 128 \text{ numbers of quantization levels} \\
 8000 \text{ samples/s} * 7 &= 56 \text{ kbit/s}
 \end{aligned}$$

Increasing number of levels means higher number of bits. In addition, the bit rate will increase. Increase of bit rate results in greater bandwidth requirement. Real time systems like VoIP sessions strive to reduce the required bandwidth, whilst still maintaining an adequate quality.

As one in real time system want to reduce both bandwidth and bit rate for audio transmission over an IP network, this involve a reduction of quantization levels. This results in an increase in quantization noise as it depends on the number of quantization levels. The challenge here will be to reduce noise, whilst at the same time maintaining a low bandwidth. With this in mind the wav-form codec's used in the simulation, might be inadequate. A solution could be a move towards hybrid coders like LD-CELP and LPC to reach a bandwidth of 2.4 kbit/s(LPC), as these codecs have lower bandwidth requirement for speech transfer.

6.2.4. Experiment 2 – BER vs SNR

The result of Experiment 2 indicated the relationship between BER and SNR. Randomly distributed errors were introduced. Several simulations were performed in order to investigate the AWGN module in MatLab. This examination confirmed that our noise module was able to create randomly distributed error patterns as desired. In addition it would be interesting to add a burst of error as well in this experiment. This could imitate the effects of e.g. loss of packets in transmission real time transmission.

A possible situation which could occur during the randomly insertion of bit error to the audio stream for the DPCM codec were observed. This situation did not occur during the simulations, but were found during thoughts around the experiment. The first difference sample in the DPCM decoder is set to be the same as the first sample received at the receiver end of the transmission system. If this sample contains error from the random introduction of errors, it will cause error in several cycles and then re-synchronise, otherwise it will just have a DC offset.

$$SNR = 10 \log \left(\frac{\sum (S_{org})^2}{\sum (S_{org} - S_{coded})^2} \right)$$

Equation 6.2:1 – SNR calculation

This experiment was not fully completed as intended, due to insufficient time. Just the first step of the original experiment was performed. A noise module is developed and randomly adds bit error to the encoded signal, according to the percentage of error selected by the user. The next step would be to measure the effects of these introduced errors. Evaluating and visualizing the SNR vs. BER for the different codecs using *Equation 6.2:1 – SNR calculation* to calculate SNR.

6.2.5. Experiment 3 – SNR performance of coders

From *Table 5.4:1* one can see an overall decrease in SNR from 48 to 40 kbit/s, though it is not that high for Uniform PCM, the other codec's almost halves their SNR from 48 to 40 kbit/s. The other considerable drop is from 40 to 32 kbit/s where for example Uniform PCM only performs 1.44 dB SNR. As mentioned before the quantization noise for the codecs increases as the number of quantization levels decreases. The PCM coder uses rounding and as a result of this the signal will contain some redundancy. As the performance of this type of speech coder degrades quickly for bit rates less than 40 kbits/s, moving towards linear predicted coders or low bit rate coders should be implemented to achieve adequate quality of audio with low bit rates.

A first-order adaptive predictor is used in our implementation of the DPCM. Considering that this coder remove some redundancy from the speech signal and are generally used for bit rates above 16 kbits/s, we observe that DPCM performs better than the other codecs for e.g. 40 kbit/s. This supports the statement above.

6.3. Subjective assessment of audio quality

6.3.1. Evaluation methods

As pointed out earlier, the user's perception of quality is not universal, and depends on many factors including sound quality and the user's content specific perception of sound quality. Another recognized determinant of quality is the purpose and level of user engagement.

This method uses a large number of human listeners to produce subjective quality scores with statistical confidence (it considers the mean of the obtained quality opinions). The method is applied to both one-way and two-way (conversational) listening scenarios, and it is applied under a controlled testing environment (method of scoring, properties of voice samples used in tests, etc.). The overall MOS (Mean Opinion Score) quality scores lie on a 1–5 scale, as shown in *Table 4.2:4 – Quality scale*. ITU Recommendation P.800 describes the techniques for performing MOS, while ITU Recommendation P.830 describes the methods for subjective evaluation of speech codecs. However, MOS quality assessment has several shortcomings:

- The setup of experiments can be quite costly and time-consuming, as it requires a large number of participants in controlled environment.
- Because these tests directly involve humans and their subjective judgements, they are influenced by uncontrollable aspects of the subjects (mood, level of engagement, etc.).
- They are unpractical if there is a requirement for frequent or on-line real-time testing. Such a requirement exists in the case of design and

configuration of networks and appliances. As if when we had implemented the codecs to the Rabbit and wanted to test the system on a large number of humans.

These drawbacks suggest that objective computational models can automatically and repeatedly estimate the ongoing quality of speech required to quantify the subjective clarity and quality of networked voice applications.

For some applications, such as IP-telephony, users typically require high audio quality. If IP-telephone conversation have too much packet loss or uses an audio codec that does not work sufficient for the user they would rather use the landline, even though using IP-telephony can save money.

It can be hard for the panellists to distinguish between the samples, since many of the samples are of high quality. To rate a adequate sample just after a excellent one might also be a problem, since the panellists may be uncertain of how low they should rate.

It is also worthwhile mentioning the distinction between sound quality as measured by mathematical procedures or computational models (i.e., the degree of errors or quantization noise) and the observed quality or sound fidelity.

6.3.2. Possible source of errors

- The speaker's ability to quote the samples played, that is, how much distortion or background noise that comes from the computer.
- The panellist's placement pursuant to the speakers, that is, listening distance and angle to the speakers.
- The volume the samples are played in.
- Acoustics of the room.
- Each panellist's comprehension of what good and bad sound is. Each human have their own psychoacoustics.
- If the panellists do not focus on their task, to rate the samples, they might not pay enough attention so that they can hear the difference of the samples. It should be mentioned that some samples can sound almost the same even though they aren't.

We also discovered that what order the samples were played in, had consequences for what rating the next sample got, if a adequate (3) sample were played after a excellent (5) sample it typically got a 2 as a rating. And when a poor (2) sample where played after the adequate (3) sample some panellist's were uncertain if they should put a 1 or a 2 as a rating.

The MOS (Mean Opinion Score) results were almost as expected for each music genre, except for the classical genre where the speech codecs and Mp3 got higher rated than expected. The audio codecs ability to quote the frequencies and the human psychoacoustics has effect on how the panellists experience the sound, and also if they actually like listening to the genre played. This also means that a music sample with no drastic changes of the frequencies could sound better even with a lower compression.

6.3.3. Music samples

It appears that sound down to 80 kbps appeal too the human listeners, even though, according to a mathematical interpretation of distortion, e.g. SNR, they are distorted in comparison to the originals, one should be able to hear the difference between 80 kbps and 320 kbps. It should also be mentioned that even though the output signal of a codec does not look exactly the same as the input, it might be inaudible for the listeners.

The so-called good samples were rated very harsh in the trance genre, this might be since they were played just after a poor or adequate sample. Or it might be that this was the first of the three tests. Only three out of five expected samples were rated above 4 in average. An all-over observation is that WMA gets higher ratings than Mp3, and that the speech codecs do not perform very well for music, even for those with lower bit rates. This is not an unexpected observation, since the speech codecs are designed to do their best performance for exactly speech.

Most of the ratings are approximately as expected, for all the genres. Another observation is that samples played more than once are rated approximately the same, which gives this type of rating method more validity.

For classical music the test started with the “worst” samples, ADPCM, which gave this codec a slightly higher rating than expected. Most of the classical samples were rated higher than the other two genres. The reason for this might be that the panellist normally do not listen to this type of music and therefore do not listen carefully enough when the samples are played. All over one can see that the rating of classical music is higher for all the codecs but WMA and Ogg-Vorbis.

6.3.4. Speech samples

Both the WMA 48 samples perform approximately just as well as PCM A-law/ μ -law and ADPCM 40. One should also note that WMA 48 is half the file-size of a PCM A-law/ μ -law sample and therefore requires less bandwidth when sending on an IP-based network.

One reason that Mp3 48 and WMA 48 performs as good as they do is that they have a more sophisticated encoding process than the decoding process, whilst PCM perform a decoding process which is the inverse of the encoding. In an embedded system the encoding process for Mp3, WMA and even Ogg-Vorbis would not be an advantage since the amount of processors time used will increase compared to other codecs. So for an embedded VoIP system a PCM or even an ADPCM 40 kbit/s coder would be preferred. On the other hand if you want to make an embedded “radio”, that is an embedded system that only require a decoder, WMA and Mp3 48 is a better alternative than PCM, especially in one want to play music on the system.

One should also be critical to the results given from this way of testing, as mentioned in possible source of errors, the order of which the samples are played in affect the next samples rating. One can observe from *figure 6.5:5* that Mp3 48 is played just after ADPCM 40 and one might expect a slightly higher rating, which it also got. But when comparing Mp3 48 to the next WMA 48 sample, one sees that WMA is rated higher, not only once, but twice.

6.3.5. Speech samples with introduced errors

In the speech test we also introduced some errors to some of the samples. This was done to see whether the panellists heard the errors and if they would rate the samples worse than the same sample without errors or a burst of errors.

From *Figure 5.5:6 – Errors found by the participants*, we observe that sample 4 and 10 are the same sample, WMA 48, and that 16 is LD-CELP - without introduced errors. As we discussed in last chapter WMA 48 got very high ratings compared to PCM, but still 50 % of the audience thought there were errors in the sample, the second time only 1 of the of the audience thought there were errors introduced to the sample. When a sample with an EB (error burst) is played the sample got higher rated, since the overall quality of the sample is interpreted as better. When a sample with randomly introduced errors was played, the audience rated the sample lower, since the overall quality was interpreted as worse than the “original”. The reason for the panellists to think it was errors introduced to LD-CELP is reasonable, since the quality of this sample were quite low, just as ADPCM 16.

Chapter 7 - Conclusion

The goal of this thesis was to implement some audio codec algorithms on an embedded system and perform several different tests on the system. Due to insufficient time no codecs were implemented on the Rabbit trainer, instead several tests were performed with G.711 – PCM, G.726 – ADPCM and DPCM implemented in MatLab. To achieve a more profound analysis of the codecs, a subjective assessment test was also added to measure the audio quality. In this listening test other music codecs and speech codecs were also tested.

As a result of the subjective assessment of audio quality we can conclude that the best suited audio algorithm for speech on an embedded system, e.g. VoIP system, is WMA 48 kbit/s, provided you have a processor that handles the complexity of this codec. For implementation on simpler processors, we recommend ADPCM 40 kbit/s, as this is sufficient for speech. For music, the scenario would be a radio based networked embedded system, which would only require a decoder on the embedded system. Then there is no need for the same complexity as on the encoding side, an implementation of WMA 80 kbit/s would be recommended for this system.

The results from the numerical experiments indicate that quantization noise increases when the quantization levels decreases meaning lower bit rate result in higher quantization noise. SNR for the different coding algorithms drop considerably below a bit rate of 40 kbit/s. In addition, the computational complexity will increase for codecs with lower bit rates. With this in mind the wav-form codec's used in the simulation, might be inadequate for bit rates below 40 kbit/s. A solution could be a move towards hybrid coders like LD-CELP and LPC to reduce the bandwidth requirement.

7.1. Recommendations for future work

The first step in which future projects could address, is to implement the audio codecs, in Dynamic C, on the Rabbit. Then it would be interesting to make the system work in real-time, and not only from PC to Rabbit. When this is done, one could make the PC application choose which audio codecs it should use. The application would then need to be able to find out what codecs that is available on the receiving side of the communication channel. If both ends support e.g. WMA 80 kbit/s, but are using PCM 64 kbit/s, it would be interesting for both parts to swap to WMA, since this would give a better audio quality and as a result of that more satisfied users of the system.

References

1. Troubleshooter.com, V., *Voice over IP Basics*. 2004, http://www.voiptroubleshooter.com/voiptr_basics.htm.
2. Deliver, I., *VoIP Knowledgebase*. 2003, http://www.ipdeliver.com/voip_k_advantages.php.
3. Education, O.-I., *H.323 Definition and Overview*. 2003, <http://www.iec.org/online/tutorials/h323/index.html>.
4. Forum, H., *H.323 Flash Tutorial*. 2004, <http://www.h323forum.org/>.
5. Radvision, *SIP: Protocol Overview*. <http://161.58.93.205/NR/rdonlyres/0F6C2E81-72B3-4C42-B33C-DDBC9115AC3D/234/SIPProtocolOverview.pdf>, 2004.
6. H. Schulzrinne, S.C., R. Frederick, V. Jacobson, *RTP/RTCP: A Transport Protocol for Real-Time Applications - rfc1889*, in *Audio-Video Transport Working Group*. 1996, <http://www.ietf.org/rfc/rfc1889.txt>.
7. Sur, T.S.a.B., *RTCP (Real-Time Control Protocol)*. 2001, <http://www.linktionary.com/t/rtcp.html>.
8. Postel, J., *RFC 793 - Transmission controll protocol*. 1981, <http://www.faqs.org/rfcs/rfc793.html>.
9. James F.Kurose, K.W.R., *Computer Networking A top-down approach featuring the internet*. 2001: Addison Wesley.
10. Postel, J., *RFC 768 - User Datagram Protocol*. 1980, <http://www.faqs.org/rfcs/rfc768.html>.
11. Postel, J., *RFC 791 - Internet Protocol*. 1981, <http://www.faqs.org/rfcs/rfc791.html>.
12. Woodard, J., *Speech Coding*. 1998, <http://www.ecs.soton.ac.uk>.
13. Platform, C.k., *ADPCM, G.721, G.726, G.727*. 2004, <http://ckp.made-it.com/adpcm.html>.
14. Mistral, *Mp3 Technologies*. 2003, <http://www.mistralsoftware.com/html/services/technologies/mp3.htm>.
15. Hacker, S., *MP3: The Definitive Guide*. 1 ed. 2000: O'Reilly.
16. Mitchell, G., *An Introduction to Compressed Audio with Ogg Vorbis*. 2003, http://grahammitchell.net/writings/vorbis_intro.html.
17. Andersen, M.C.b.S., *About Windows Media Audio Codec*. 2000, <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnwmt/html/msaudio.asp>.
18. Knispel, B., *The Prog_CDR FAQ, v 1.1*. 2001, <http://www.progcdr.org/progcdrfaq.htm#2:%20WMA>.
19. Foundation, X.O., *The Ogg Vorbis CODEC project*. 2003, <http://www.xiph.org/ogg/vorbis/>.
20. Impulsesoft, *Impulsesoft Bluetooth Audio Guide*. 2003, <http://www.impulsesoft.com/HomePage/btaudioguide.pdf>.
21. Jerry D. Gibson, T.B., Tom Lookabaugh, Dave Lindbergh, Richard L.Baker, *Digital Compression for multimedia (Principles & Standards)*. 1998: Morgan Kaufmann Publishers, Inc.

22. Bluetooth.com. *ADVANCED AUDIO DISTRIBUTION PROFILE SPECIFICATION*. 2003, <http://www.bluetooth.com/>.
23. Quicklogic, *G.711 PCM Codec*. 2004, http://www.quicklogic.com/images/amphion_g711_pcm_1.pdf.
24. ITU-T, *Pulse code modulation (PCM) of voice frequencies G.711*. 1972.
25. ITU-T, C.f., *40, 32, 24, 16 kbit/s Adaptive Differential Pulse Code Modulation (ADPCM) G.726*. 1990.
26. ITU-T, C.f., *Coding of speech at 16 kbit/s using Low-Delay Code Excited Linear Prediction G.728*. 1992.
27. D. Hermann, R.L.B., H. Sheikhzadeh, E. Cornu, *Low-Power implementation of the bluetooth subband audio codec*. 2004, <http://IEEE.org>.
28. Woodard, J., *Waveform Codecs*. 2004, Department of Electronics & Computer Science, University of Southampton, http://www-mobile.ecs.soton.ac.uk/speech_codecs/waveform.html.
29. Semiconductor, R., *RabbitCore RCM3400 - User's manual*, <http://www.rabbitsemiconductor.com/documentation/docs/manuals/RCM3400/index.htm>. 2002.
30. Hill, P.H.a.W., *The Art of Electronics*. 2nd ed. 1989.
31. Dictionary, D., <http://www.dilettantesdictionary.com/index.php?let=q>. 2004.
32. Department of Electrical Engineering, M.U., *Experiment 4 - Subband Audio Compression*. 2004, <http://www.ece.mcgill.ca/courses/304-490/Exp4/Exp4.pdf>.
33. M. Handley, H.S., E. Schooler, J. Rosenberg, RFC 2543 - SIP: Session Initiation Protocol. 1999, <http://www.faqs.org/rfcs/rfc2543.html>.
34. Speech Samples, Examples of Digitally Compressed Speech. 2004, <http://www.its.bldrdoc.gov/home/programs/audio/examples.htm>.

Abbreviations

Table 7.1:1 – Abbreviations and its explanations

Abbreviation:	Explanation:
A	
ACK	Acknowledgment (TCP)
ADPCM	Adaptive Differential Pulse Code Modulation
AUC	Agder University College (also see HiA)
AWGN	Additive White Gaussian Noise
B	
BSC	Bluetooth Sub-band Codec
C	
CCITT	Commite' Consultatif International de Telegraphique et Telephonique. (International consultative committee on telecommunications and Telegraphy).
Compander	Short for compressor/expander
D	
DPCM	Differential Pulse Code Modulation
F	
FDM	Frequency Division Multiplexing
FIN	Finish, No more data from sender (TCP)
G	
GUI	Graphical User Interface
H	
HiA	Høgskolen i Agder (also see AUC)
HTTP	Hypertext Transfer Protocol
I	
I/O	Input / Output
ICMP	Internet Control Message Protocol
ICT	Information and Communication Technology
IETF	Internet Engineering Task Force
IP	Internet Protocol
ITEE	Information Technology and Electrical Engineering
ITU-T	International Telecommunication Union
L	
LAN	Local Area Network
LDCELP	Low-Delay Code Excited Linear Prediction
M	
MCI	Media Control Interface
MCU	Multipoint Control Unit
Mp3	MPEG1 Layer III
MOS	Mean Opinion Score
MPEG	Moving Picture Experts Group
MSS	Maximum Segment Size (TCP)
P	
PCM	Pulse Code Modulation
PSH	Push Function (TCP)
PSTN	Public Switched Telephone Network

Q	
QoS	Quality of Service
R	
RFC	Request For Comments
RST	Reset the connection (TCP)
RTCP	Real-time Transport Control Protocol
RTP	Real-time Transport Protocol
S	
SIP	Session Initiation Protocol
SIP	Session Initiation Protocol
SMTP	Simple Mail Transfer Protocol
SYN	Synchronize sequence numbers (TCP)
T	
TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TOS	Type of Service (IP)
TTL	Time To Live (IP)
U	
UDP	User Datagram Protocol
UQ	The University of Queensland
URG	Urgent Pointer field significant (TCP)
V	
VB	Visual Basic
W	
WMA	Windows Media Audio codec

Appendix overview

Appendix A – Datasheet for the electrical circuitry

Appendix B – MatLab Code

Appendix C – Visual Basic V.6.0 Code

Appendix D – Dynamic C Code

Appendix E – Subjective assessment of audio quality



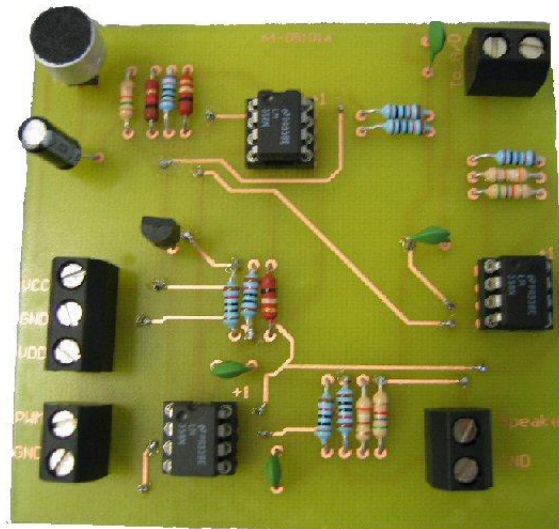
AGDER UNIVERSITY
COLLEGE

Faculty of Engineering and
Science



THE UNIVERSITY OF QUEENSLAND

School of Information Technology and
Electrical Engineering



Appendix A – Datasheet for the electrical circuitry

–
*Microphone amplifier
and
speaker circuitry with Low-Pass filter*

–
For
Agder University College
&
The University of Queensland

–
by
Knut Ole Hauge
Svein Birger Skogly



Table of content

APPENDIX A – DATASHEET FOR THE ELECTRICAL CIRCUITRY	1
TABLE OF CONTENT	2
LIST OF EQUATIONS	2
CHAPTER 1 - DESCRIPTION	3
1.1. OBJECTIVE	3
1.2. POWER SUPPLY	3
1.3. I/O PORTS	3
1.4. OVERVIEW	3
1.5. CONNECTIONS	3
1.6. DIMENSION	3
1.7. LIST OF COMPONENTS	4
1.8. CIRCUITRY OVERVIEW	5
1.8.1. <i>Top layer</i>	5
1.8.2. <i>Bottom layer</i>	5
1.9. 3D OVERVIEW	6
1.10. SCHEMATIC OVERVIEW	7
1.10.1. <i>Microphone</i>	7
1.10.2. <i>Speaker</i>	8
CHAPTER 2 - CALCULATIONS	9
2.1. VOLTAGE AMPLIFICATION	9
2.1.1. <i>Amplifier (op-amp)</i>	9
2.1.2. <i>Amplification after LP-filter</i>	9
2.2. CUT-OFF FREQUENCY	9
2.3. CURRENT AMPLIFIER	10

List of equations

EQUATION 1 - AMPLIFIER GAIN AFTER THE OP-AMPS	9
EQUATION 2 - LP-FILTER AND SPEAKER CIRCUITRY GAIN	9
EQUATION 3 - TOTAL GAIN OF MICROPHONE CIRCUITRY	9
EQUATION 4 – CAPACITORS VALUE	10

Chapter 1 - Description

1.1. Objective

The circuitry is an interface circuitry giving the rabbit trainer a microphone and a speaker.

1.2. Power supply

The power supply should be connected to $\pm 10V$ and GND.

1.3. I/O ports

- [TB1 PMW_in] – Input signal from PMW on the rabbit
- [TB2 To_A/D] – Output signal to A/D on rabbit
- [TB3 Power] – Power supply for the circuitry
- [TB4 Speaker] – Output signal to speaker

1.4. Overview

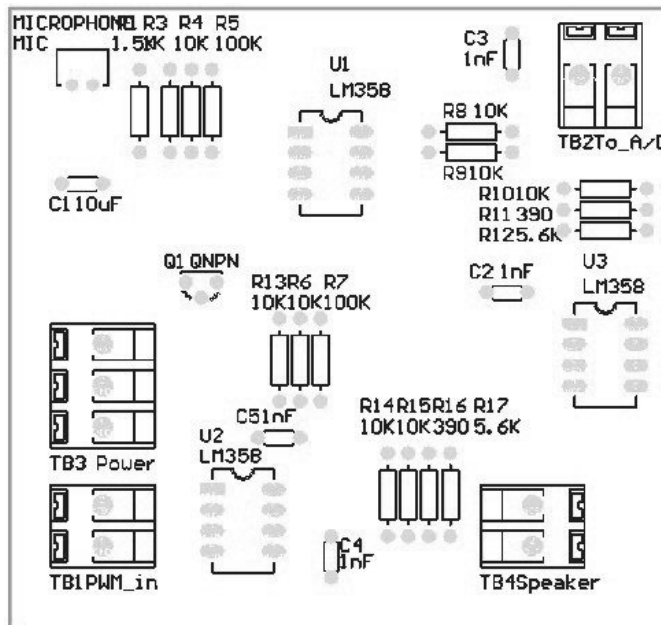


Figure 1.4:1 - Electrical circuitry component overview

1.5. Connections

Table 1.5:1 - Overview of connections

[TB1 PMW_in] 1. GND 2. In signal from PMW	[TB3 Power] 1. -10 V 2. GND 3. +10 V
[TB2 To_A/D] 1. Out signal to A/D 2. GND	[TB4 Speaker] 1. Out signal to speaker 2. GND

1.6. Dimension

Height: 75 mm
 Width: 81 mm

1.7. List of components

Table 1.7:1 - List of components

Part:	Value:	Package:	Library:	Footprint:
Power supply pin	$\pm 10V$ DC, GND	1 x 03	TB3	PCTB3
Op-amp	LM358	3 x 04	Op-Amp	DIP8
Microphone	34L1T	1 x 02	MIC01	MIC01
Speaker connection	426-2773 (Farnell)	1 x 02	TB2	PCTB2
I/O ports	TB2	2 x 02	TB2	PCTB2
Q ₁	PN 3565	QNP	QNP	TO92/C
C ₁	10 μ F	CAP	CAP	RAD0.2A
C ₂	1 nF	CAP	CAP	RAD0.2A
C ₃	1 nF	CAP	CAP	RAD0.2A
C ₄	1 nF	CAP	CAP	RAD0.2A
C ₅	1 nF	CAP	CAP	RAD0.2A
R ₁	1,5 K Ω	RES	RES	AXIAL0.4
R ₃	1 K Ω	RES	RES	AXIAL0.4
R ₄	10 K Ω	RES	RES	AXIAL0.4
R ₅	100 K Ω	RES	RES	AXIAL0.4
R ₆	10 K Ω	RES	RES	AXIAL0.4
R ₇	100 K Ω	RES	RES	AXIAL0.4
R ₈	10 K Ω	RES	RES	AXIAL0.4
R ₉	10 K Ω	RES	RES	AXIAL0.4
R ₁₀	10 K Ω	RES	RES	AXIAL0.4
R ₁₁	390 Ω	RES	RES	AXIAL0.4
R ₁₂	5,6 K Ω	RES	RES	AXIAL0.4
R ₁₃	10 K Ω	RES	RES	AXIAL0.4
R ₁₄	10 K Ω	RES	RES	AXIAL0.4
R ₁₅	10 K Ω	RES	RES	AXIAL0.4
R ₁₆	390 K Ω	RES	RES	AXIAL0.4
R ₁₇	5,6 K Ω	RES	RES	AXIAL0.4



1.8. Circuitry overview

1.8.1. Top layer

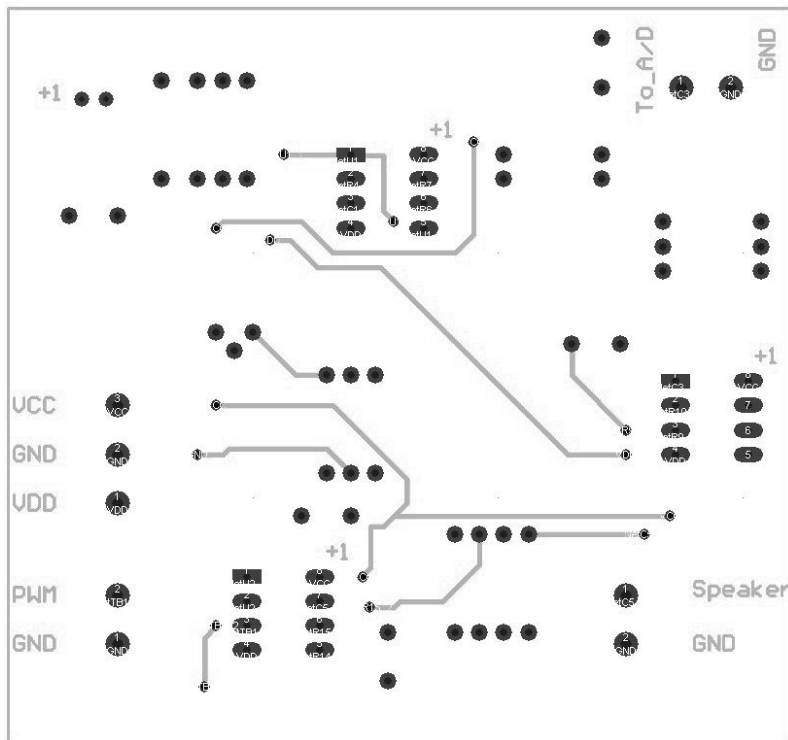


Figure 1.8:1 - Top layer

1.8.2. Bottom layer

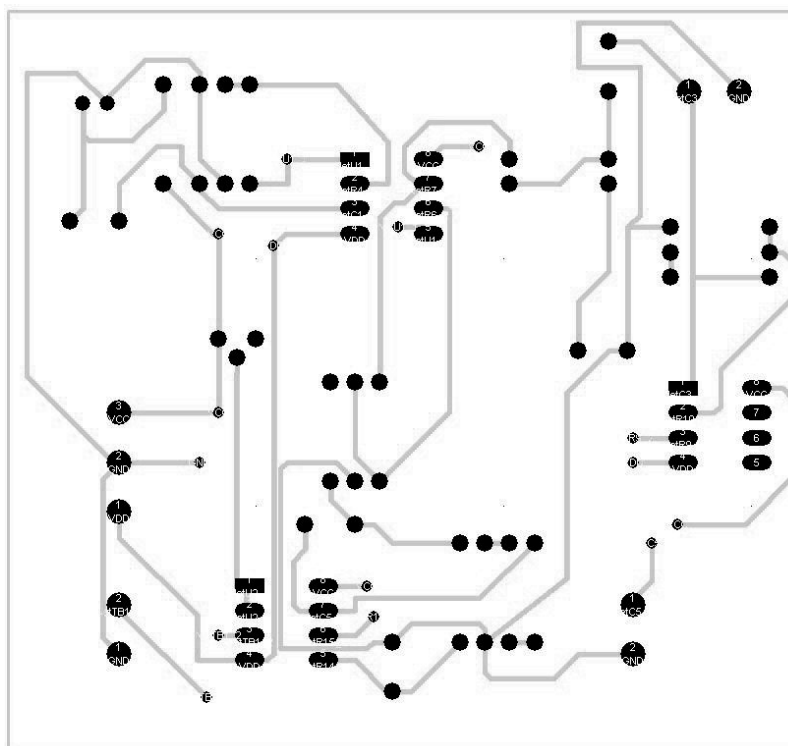


Figure 1.8:2 - Bottom layer



1.9. 3D Overview

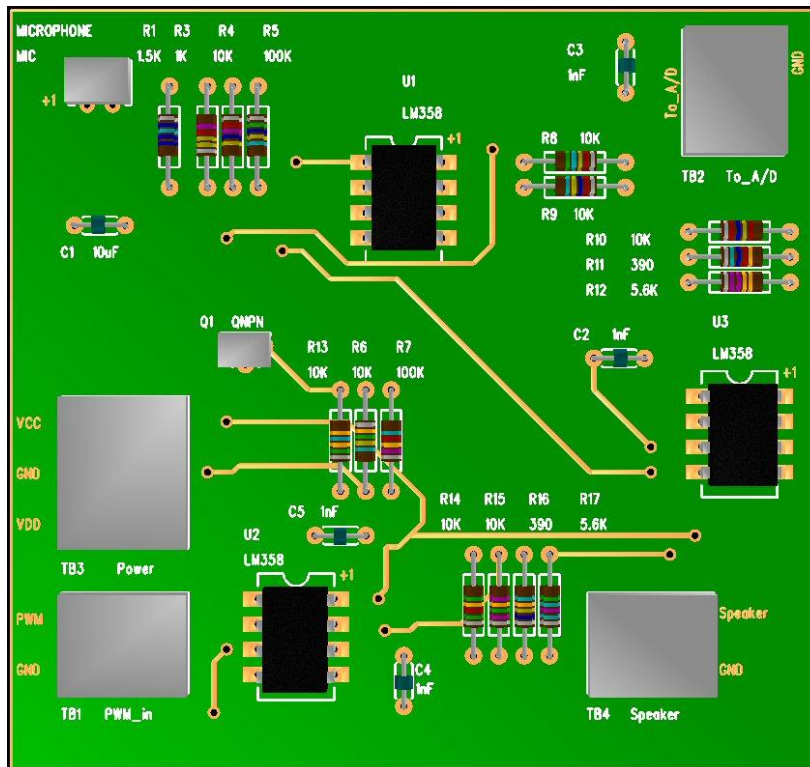


Figure 1.9:1 - 3D Overview of electrical circuitry

1.10. Schematic overview

1.10.1. Microphone

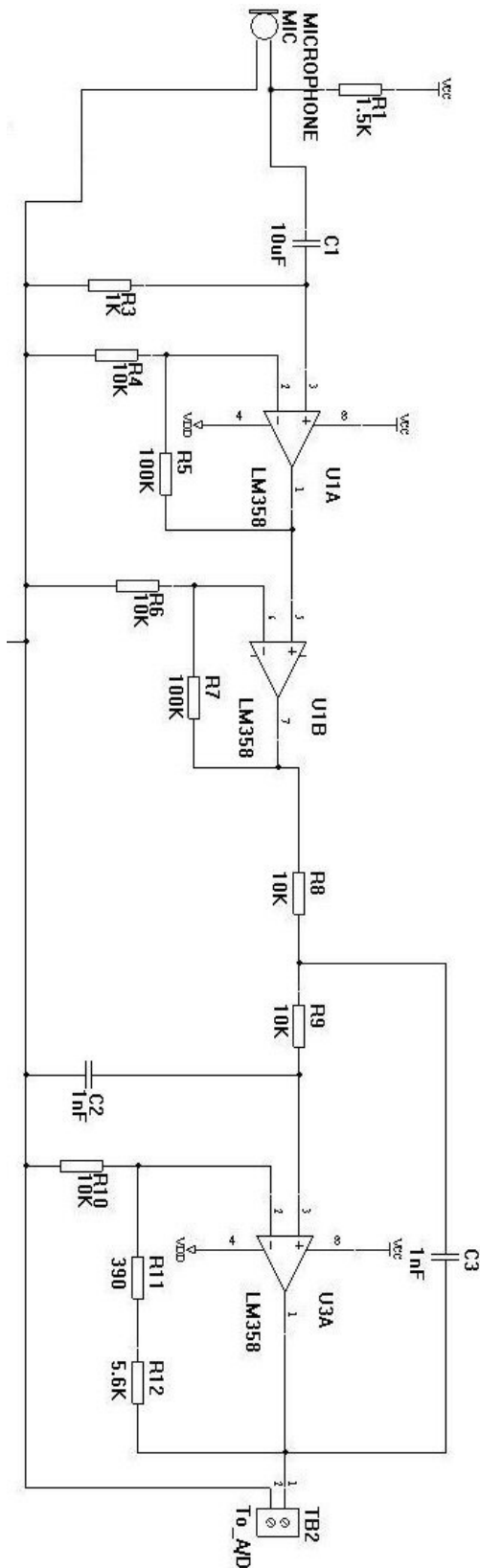


Figure 1.10:1 - Schematic overview of the Microphone circuitry

1.10.2. Speaker

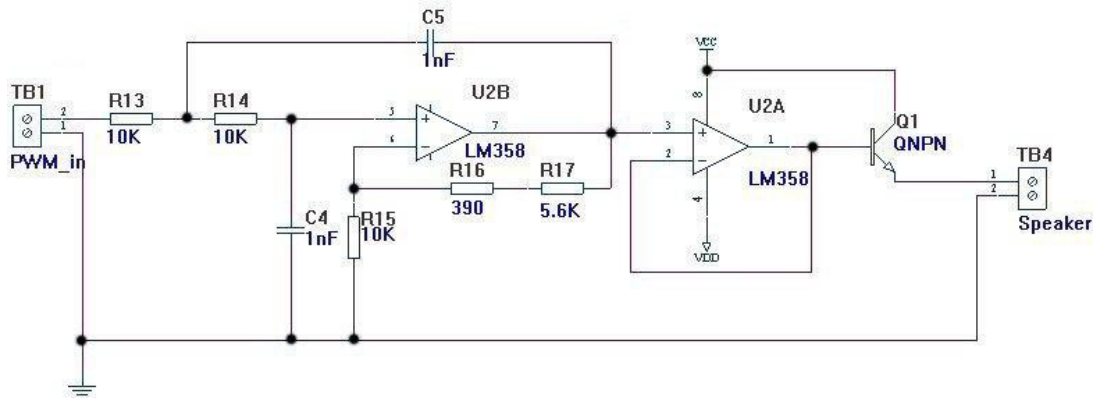


Figure 1.10:2 - Schematic overview of the Speaker circuitry



Chapter 2 - Calculations

2.1. Voltage amplification

The Low Pass-filter is a Butterworth filter of second order. Its function is to let all the low frequencies pass through, and block out the high frequencies. A second order filter cuts the frequency response with a “sharpness” of 12 dB / Octave. (A first order filter cuts of with 6 dB / Octave, a third order with 18 dB / Octave, and so on).

2.1.1. Amplifier (op-amp)

$$A_v = A_{v1} * A_{v2} = \left(\frac{R_4}{R_5} + 1\right) * \left(\frac{R_7}{R_8} + 1\right) = \left(\frac{100K\Omega}{10K\Omega} + 1\right) * \left(\frac{100K\Omega}{10K\Omega} + 1\right) = \underline{\underline{121times}}$$

Equation 1 - Amplifier gain after the op-amps

2.1.2. Amplification after LP-filter

$$A_{v_LP} = \frac{(5,6K\Omega + 390\Omega)}{10K\Omega} \approx \underline{\underline{1,6times}}$$

Equation 2 - LP-filter and Speaker circuitry gain

$$A_v + A_{v_LP} = \underline{\underline{122,6times}}$$

Equation 3 - Total gain of microphone circuitry

The same LP-filter is used for the speaker circuitry as the microphone circuitry. This means that some of the resistors and capacitors have equal values, those are listed below.

Table 2.1:1 - List of resistors and capacitors with equal value

	Microphone:	Speaker:	Value:
Res:	R _{8,9,10}	R _{13,14,15}	10 KΩ
Res:	R ₁₁	R ₁₆	390 Ω
Res:	R ₁₂	R ₁₇	5,6 KΩ
Cap:	C _{2,3}	C _{4,5}	1 nF

2.2. Cut-off frequency

The **cut-off** frequency chosen is **15 kHz** and the constant **K [31 p.274] is 1,586**, while our LP-filter is taken from [31 p.274 fig.5.16]. 15 kHz is high, given that speech will be sent to the trainer. However since this is a prototype music samples also should be sent to see how the system handled this. If speech was all that was required to be sent, a typically choice of the cut-off frequency would be 4 kHz.

A 12 dB / Octave LP-filter cuts the frequency response with 12dB each time the frequency doubles. This means that in a filter-circuit with a cut-off of 500 Hz and an in-signal of 90 dB will be about 78 dB at 1 kHz and 66 db at 2 kHz.



$$C_{2,3} = \frac{1}{2\pi R f_c} = \frac{1}{2\pi * (10 * 10^3) * (15 * 10^3)} = 1,06 * 10^{-9} \approx \underline{\underline{1nF}}$$

Equation 4 – Capacitors value

<p>Choose $R = 10k\Omega$ $K = 1.586$</p> <p>$C_{2,3} = 1 / (2\pi R f_c)$ $C_{2,3} = 1 / (2\pi * (10 \times 10^3) * (15 \times 10^3))$ $C_{2,3} = 1.06 * 10^{-9}$ $C_{2,3} = \underline{1 \text{ nF}}$</p> <p>$R_{11} + R_{12} = R * (K-1)$ $R_{11} + R_{12} = 10 * 10^3 * (1.586 - 1)$ $R_{11} + R_{12} = \underline{5.86 \times 10^3 \Omega}$</p>	<p>$F_c = \text{cut-off at } -3\text{dB} = 15 \text{ KHz}$</p> <p>$RC = \frac{1}{2}\pi f_c$</p> <p>$R_6 = R_7 = R_8 = R$</p> <p>$(R_9 + R_{10}) = R * (K-1)$</p> <p>$C_2 = C_3$</p>
--	--

R_{11} and R_{16} is set to 390Ω (closest standard value) and R_{12} and R_{13} is set to $5,6 \text{ K}\Omega$, which make $R_{11+12} = \underline{5990 \text{ K}\Omega}$.

The cut-off frequency can easily be changed by replacing the respective capacitors and resistors.

2.3. Current amplifier

The speaker circuitry includes a unity gain buffer built in a common collector amplifier; see U2A on *figure 1.10:2 – Speaker circuitry*. This is to amplify the current before the LP-filter.



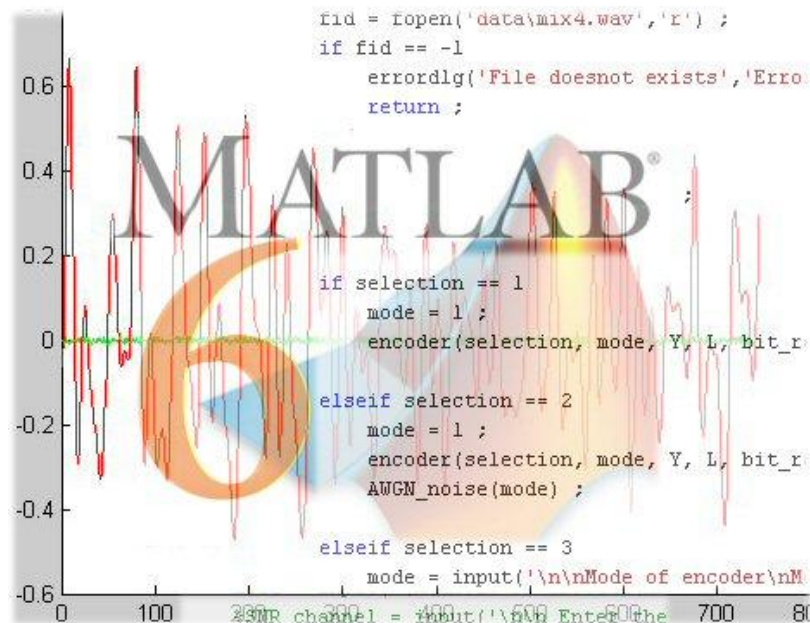
AGDER UNIVERSITY
COLLEGE

Faculty of Engineering and
Science



THE UNIVERSITY OF QUEENSLAND

School of Information Technology and
Electrical Engineering



Appendix B – MatLab Code

–
G.711 PCM
G.726 ADPCM
DPCM
Noise module

–
 For
 Agder University College
 &
 The University of Queensland

–
 by
 Knut Ole Hauge
 Svein Birger Skogly



Table of content

APPENDIX B – MATLAB CODE -----	1
TABLE OF CONTENT -----	2
CHAPTER 1 - MAIN -----	3
1.1. CODEC -----	3
1.2. ENCODER -----	4
1.3. DECODER -----	6
CHAPTER 2 - G.711 PCM -----	8
2.1. UNIFORM PCM -----	8
2.1.1. <i>Encoder</i> -----	8
2.1.2. <i>Decoder</i> -----	8
2.2. A-LAW -----	8
2.2.1. <i>Encoder</i> -----	8
2.2.2. <i>Decoder</i> -----	9
2.3. M-LAW -----	9
2.3.1. <i>Encoder</i> -----	9
2.3.2. <i>Decoder</i> -----	10
CHAPTER 3 - DPCM -----	11
3.1.1. <i>Encoder</i> -----	11
3.1.2. <i>Decoder</i> -----	11
CHAPTER 4 - G.726 ADPCM -----	12
4.1. ENCODER -----	12
4.2. DECODER -----	16
4.3. AI UPDATE -----	18
4.4. AP UPDATE-----	19
4.5. BI UPDATE -----	19
CHAPTER 5 - AWGN MODULE -----	21
5.1. AWGN CODE -----	21
5.2. RANDOM NOISE -----	22

Chapter 1 - Main

1.1. Codec

```

%=====
%
%=====
clear all ;
close all ;
%=====
% Input values
%=====

selection = input('Menu Selection guide \n 1\) View the quantization noise for
different PCM modes\n 2\) View the effect of SNR on BER \n 3\) Run the codec \n
Enter your choice : ');

L = input('\n\nEnter the number of steps for quantization : ');

bit_rate = 32 ;

%=====
fid = fopen('data\PCMCDQuality.wav','r');
if fid == -1
    error('File doesnot exists','Error');
    return ;
end

Y = fread(fid, 4000, 'int16');
fclose(fid);

if selection == 1
    mode = 1 ;
    encoder(selection, mode, Y, L, bit_rate);

elseif selection == 2
    mode = 1 ;
    encoder(selection, mode, Y, L, bit_rate);
    AWGN_noise(mode);

elseif selection == 3
    mode = input('\n\nMode of encoder\nMode 1 : Uniform PCM \nMode 2 : A Law
PCM \nMode 3 : Mu Law PCM \nMode 4 : DPCM \nMode 5 : ADPCM \nEnter the
mode of operation : ');
    %SNR_channel = input('\n\n Enter the desired SNR for channel (in dB) : ');

    encoder(selection, mode, Y, L, bit_rate);
    decoder(mode, L, bit_rate);

```



```
else
    error('Incorrect mode selected','Error') ;
    return ;
end
```

1.2. Encoder

```
function encoder(selection, mode, Y, L, bit_rate)

%=====
headlength = 256;
header = Y(1:headlength);
Y = Y(headlength+1: length(Y));
max_sample_value = power(2,16)/2 ; % range is from -32768 to + 32767
Y_normalized = Y/max_sample_value ;
delta = 2 / L ;
figure ;
iteration = 1 ;
binary = [] ;

while (iteration < 5) ;

    switch mode
    case 1
        [quantized binary quantisation_noise] = PCMEncoder(Y_normalized, delta, L) ;
        SNR_case1 =
        10*log10(power((sum(Y_normalized)/sum(quantisation_noise)),2));

    case 2
        Alaw_PCM = PCM_A_law_encoder(Y_normalized) ;
        [quantized binary quantisation_noise] = PCMEncoder(Alaw_PCM, delta, L) ;
        SNR_case2 =
        10*log10(power((sum(Y_normalized)/sum(quantisation_noise)),2));

    case 3
        mulaw_PCM = PCMMu_law_encoder(Y_normalized) ;
        [quantized binary quantisation_noise] = PCMEncoder(mulaw_PCM, delta, L) ;
        SNR_case3 =
        10*log10(power((sum(Y_normalized)/sum(quantisation_noise)),2));

    case 4
        DPCM = DPCMEncoder(Y_normalized) ;
        [quantized binary quantisation_noise] = PCMEncoder(DPCM, delta, L) ;
        SNR_case4 =
        10*log10(power((sum(Y_normalized)/sum(quantisation_noise)),2));

    case 5
        % for ADPCM
        ADPCM_quantized = ADPCMEncoder( Y*8191/32767, bit_rate, 'lin');
        %max_sample_value = power(2,((bit_rate/8)-1))/2 ;
```




```
max_sample_value = 1 ;
ADPCM_normalized = (ADPCM_quantized/max_sample_value)' ;
[quantized binary quantisation_noise] = PCMEncoder(ADPCM_normalized,
delta, L) ;

    otherwise
    error('Incorrect mode of encoding','Error') ;
    return ;
end

if selection == 1
    hold on ;
    plot(quantisation_noise(1:100),color(iteration)) ;
    iteration = iteration + 1 ;
    mode = iteration ;

elseif selection == 2

    switch iteration
    case 1
        save 'data\BinarySamples1.mat' binary ;
    case 2
        save 'data\BinarySamples2.mat' binary ;
    case 3
        save 'data\BinarySamples3.mat' binary ;
    otherwise
        save 'data\BinarySamples4.mat' binary ;
    end

    iteration = iteration + 1 ;
    mode = iteration ;

else
    %binary = [header' binary] ;
    encoded_samples = [header ; quantized] ;
    save 'data\samples.mat' encoded_samples ;
    save 'data\BinarySamples.mat' binary ;
    hold on ;
    plot(Y_normalized(1:500),'r') ;
    if mode ~= 5
        plot(quantisation_noise(1:500),'g') ;
        legend('samples','Quantization noise') ;
    end

    hold off ;
    break ;
end

end
if selection == 1
```



```
    legend('Uniform PCM', 'A-law PCM', 'Mu-law PCM', 'DPCM') ;  
end  
hold off ;  
figure ;  
  
SNR=[SNR_case1,SNR_case2,SNR_case3,SNR_case4] ;  
t=[1:4] ;  
stem(t,SNR) ;  
title('plot of SNR for different PCM modes');  
ylabel('SNR in dB');  
  
function index = color(iteration)  
switch iteration  
    case 1  
        index = 'r' ;  
    case 2  
        index = 'b' ;  
    case 3  
        index = 'g' ;  
    otherwise  
        index = 'k' ;  
end
```

1.3. Decoder

```
function decoder(mode, L, bit_rate)  
%=====   
% Input values  
%=====   
% mode = 1 ;  
% L = 32 ;  
% bit_rate = 32 ;  
  
load 'data\samples.mat' encoded_samples ;  
data = encoded_samples ;  
  
%=====   
headlength = 256;          % assumes SPPACK headers  
header = data(1:headlength) ;  
data = data(headlength+1: length(data));  
  
delta = (2 / L) ;  
  
switch mode  
    case 1  
        % PCM  
        PCM_decoded = (PCMdecoder(data, delta))*(power(2,16)/2) ;  
        wav_data = [header ; PCM_decoded'] ;  
  
    case 2  
        % A law
```



```
PCM_decoded = PCMdecoder(data, delta) ;
Alaw_PCM_decoded = PCMA_law_decoder(PCM_decoded)*(power(2,16)/2)
;
wav_data = [header ; Alaw_PCM_decoded] ;

case 3
% Mu law
PCM_decoded = PCMdecoder(data, delta) ;
PCMMu_decoded = PCMMu_law_decoder(PCM_decoded)*(power(2,16)/2) ;
wav_data = [header ; PCMMu_decoded] ;

case 4
% DPCM
PCM_decoded = PCMdecoder(data, delta) ;
DPCM_decoded = DPCMdecoder(PCM_decoded)*(power(2,16)/2) ;
wav_data = [header ; DPCM_decoded] ;

case 5
% for ADPCM
ADPCM_decoded = ADPCMdecoder(data, bit_rate);
plot(ADPCM_decoded*32767/8191) ;
decoded_samples = ADPCM_decoded*32767/8191 ;
wav_data = [header ; decoded_samples] ;

otherwise
%for LCELP
end

fid = fopen('data\mix4out.wav','w') ;
fwrite(fid, wav_data , 'int16') ;
fclose(fid) ;

[fid, message] = fopen('data\PCMCDQuality.wav','r');    % open given raw data
file
if fid == -1
    error(message) ;
    return
end

Y = fread(fid, 4000, 'int16');    % put the datafile into a vector %uint8
fclose(fid) ;

figure ;
t= 1:length(Y) ;
subplot(2,2,1)
plot ( Y , 'k') ;
%hold on
subplot(2,2,2)
plot (wav_data, 'b')
%hold off
```

Chapter 2 - G.711 PCM

2.1. Uniform PCM

2.1.1. Encoder

```
function [quantized, binary_PCM, quantisation_noise] = PCMencoder(Y, delta, L)
```

```
quantized = [] ;
```

```
temp =[] ;
```

```
for i = 1 : length(Y) ,
```

```
    level = 0 ;
```

```
    for j = 1:L,
```

```
        if Y(i) >= (-1+delta*(j-1)) & Y(i) < (-1+delta*(j))
```

```
            level = j-1 ;
```

```
            break ;
```

```
        end
```

```
    end
```

```
    quantized(1,i) = level ;
```

```
    temp = [temp dec2bin(level, log2(L))] ;
```

```
    binary_PCM = rem(double(temp),48) ;
```

```
    % calculation of quantization noise
```

```
    quantized_samples = round(Y(i)/delta)*delta ;
```

```
    quantisation_noise(1,i) = Y(i) - quantized_samples ;
```

```
end
```

2.1.2. Decoder

```
function PCM_decoded = PCMdecoder(PCM_encoded, delta)
```

```
%=====
```

```
% PCM decoder
```

```
%=====
```

```
for i = 1 : length(PCM_encoded) ,
```

```
    temp = PCM_encoded(i) * delta ;
```

```
    PCM_decoded(1,i) = -1 + temp ;
```

```
end
```

2.2. A – Law

2.2.1. Encoder

```
function Alaw_PCM = PCM_A_low_encoder(Y) ;
```

```
%=====
```

```
% PCM A-law encoder
```

```
%=====
```



```
A = 86.7;          % A-law value
Y_max = max(Y(:)); % Finding Y_max
denom = 1 + log(A);
Alaw_PCM = [];

for i = 1 : length(Y) ,

    if ((abs(Y(i))/Y_max) >= 0 & (abs(Y(i))/Y_max) <= 1/A) %A-law compressor
        Alaw_PCM(1,i) = (A*abs(Y(i))/denom)*sign(Y(i));

    elseif ((abs(Y(i))/Y_max) >= 1/A & (abs(Y(i))/Y_max) <= 1)
        Alaw_PCM(1,i) = Y_max * (1 + log(A*abs(Y(i))/Y_max)/denom);

    end
end
```

2.2.2. Decoder

```
function Alaw_PCM_decoded = PCMA_law_decoder(Alaw_PCM) ;

%=====
% PCM A-law decoder
%=====

A = 86.7;          % A-law value
Alaw_PCM_max = max(Alaw_PCM(:)); % Finding Y_max
denom = 1 + exp(A);
Alaw_PCM_decoded = [];

Signy = sign(Alaw_PCM);
for i = 1 : length(Alaw_PCM) ,

    if ((abs(exp(Alaw_PCM(i)))/exp(Alaw_PCM_max)) >= 0 &
(abs(exp(Alaw_PCM(i)))/exp(Alaw_PCM_max)) <= 1/A) %A-law compressor
        Alaw_PCM_decoded(i) =
(A*abs(Alaw_PCM(i))/denom)*sign(Alaw_PCM(i));

    elseif ((abs(exp(Alaw_PCM(i)))/exp(Alaw_PCM_max)) >= 1/A &
(abs(exp(Alaw_PCM(i)))/exp(Alaw_PCM_max)) <= 1)
        Alaw_PCM_decoded(i) = Alaw_PCM_max * (1 +
exp(A*abs(Alaw_PCM(i))/Alaw_PCM_max)/denom);
    end
end
```

2.3. μ – Law

2.3.1. Encoder

```
function mulaw_PCM = PCMMu_law_encoder(Y) ;
mu = 255;          % Mu-law value
Y_max = max(Y(:)); % Finding Y_max
```



```
denom2 = log(1 + mu);  
  
for i = 1:length(Y),  
    mulaw_PCM(1,i) = Y_max * sign(Y(i))*log(1+(mu*abs(Y(i))/Y_max))/denom2;  
end
```

2.3.2. Decoder

```
function PCMMu_decoded = PCMMu_law_decoder(mulaw_PCM) ;  
  
mu = 255;          % Mu-law value  
Y_max = max(mulaw_PCM(:)); % Finding Y_max  
denom2 = log(1 + mu);  
  
PCMMu_decoded=[];  
  
% decompress the output signal  
Signy = sign(mulaw_PCM);  
for i = 1:length(mulaw_PCM),  
    PCMMu_decoded(i) = Signy(i)* ((exp(mulaw_PCM(i)*denom2 *  
    Signy(i)/Y_max) - 1))*(Y_max/mu);  
end
```



Chapter 3 - DPCM

3.1.1. Encoder

```
function DPCM = DPCM_encoder(Y) ;  
  
%=====   
% DPCM encoder   
%=====   
  
DPCM = [];  
  
% find the difference signal with alpha=0.45  
alpha = 0.45;  
DPCM(1) = Y(1);  
for k = 2:length(Y),  
    DPCM(1,k) = Y(k) - alpha*Y(k-1);  
end
```

3.1.2. Decoder

```
function DPCM_decoded = DPCMdecoder(DPCM_encoded) ;  
  
%=====   
% DPCM decoder   
%=====   
  
DPCM_decoded = [];  
  
% find the difference signal with alpha=0.45  
alpha = 0.45;  
DPCM_decoded(1) = DPCM_encoded(1);  
for k = 2:length(DPCM_encoded),  
    DPCM_decoded(k) = DPCM_encoded(k) - alpha*DPCM_decoded(k-1);  
end
```

Chapter 4 - G.726 ADPCM

4.1. Encoder

```
function [I,sr,dout,dqout,yout,aiout,biout,tdout,trout,alout] = ADPCMencoder(sl, N,
format);
```

```
%ADPCM Adaptive differential pulse code modulation.
% [I, SR] = ADPCM( SL, N) encodes the PCM coded input signal SL
% with N kbit/s adaptive differential pulse code modulation (ADPCM)
% where N is selected from 40, 32, 24, or 16 kbit/s.
% If N is not specified, the bit rate is set to N = 32 kbit/s.
```

```
%
% By default, the input signal SL is assumed to be represented with
% 14-bit signed-magnitude uniform PCM, i.e., its amplitude range is
%  $-8191 \leq sl \leq 8191$ . This input format is also selected by
% ADPCM(SL, N, 'lin'). If the input signal is 8-bit logarithmic PCM
% (mu-law in accordance with LIN2MU), ADPCM( SL, N, 'mu') has to
% be used.
```

```
%
% The first output is I, the encoder output signal which represents
% the bit stream over the channel. The samples of I are from the
% discrete alphabet  $-2^{(N/8)-1}, \dots, +2^{(N/8)-1}$ . The second (optional)
% output is the reconstructed speech signal SR (scaled approximately
% between -8191 and +8191) which would be the output of the corres-
% ponding ADPCM decoder in the case of error-free transmission of I.
```

```
%
% If more than two outputs are specified, a whole list of
% diagnostic outputs (internal variables of the algorithm) becomes
% available:
```

```
%
%  $[I,SR,D,DQ,Y,AI,BI,TD,TR,AL] = ADPCM( SL, N);$ 
```

```
%
% with the following interpretation:
```

```
%
% D prediction difference signal
% DQ quatized prediction difference signal
% Y scale factor for adaptive quantizer
% AI recursive predictor coefficients (2-column matrix)
% BI non-recursive predictor coefficients (6-column matrix)
% TD tone detector signal
% TR transient detector signal
% AL speed control for scale factor adaptation
```

```
%
% Note that this latter option is VERY memory intensive as all
% variables are vectors/matrices of the size of the input signal.
```

```
%
% This function uses also APUPDATE, AIUPDATE, BIUPDATE.
% The decoder is provided by I_ADPCM.
```

```
%
```




```
% References:
% The algorithm follows closely the ITU-T (former CCITT)
% recommendation G.726 "40, 32, 24, 16 kbit/s Adaptive Differential
% Pulse Code Modulation (ADPCM)". The recommendation specifies
% various fixed-point formats for the internal variables which are
% not simulated in this MATLAB implementation. However, most of the
% variable names etc. are consistent with the recommendation.

% input argument checks, default bit-rate and format setting
if (nargin < 1) | (nargin > 3)
    disp('ADPCM: 1 to 3 input variables required!')
    return
end
if nargin == 1
    N = 32;
    disp('***** N = 32 kbit/s *****');
end
a = N==[40 32 24 16];
if (sum(a) ~= 1)
    N = 32;
    disp('***** N = 32 kbit/s *****');
end;
if nargin <= 2
    format = 'lin';
end;
if isstr(format) ~= 1
    error('ADPCM: Input format must be string variable!')
elseif strcmp(format,'mu')
    sl = 8191*mu2lin(sl);
elseif ~strcmp(format, 'lin')
    error('ADPCM: wrong input format')
end
if min(size(sl)) ~= 1
    error('ADPCM: Input signal must be a row or column vector only');
end;
sl = sl(:); % force input signal to be a column vector
if max(abs(sl)) > 8191
    disp('ADPCM warning: there can be clipping');
    disp(['You should use the amplitude range of ',...
        '-8191 <= SL <= 8191 for the input signal SL!']);
end;
if max(sl) <= 1
    disp('ADPCM warning: there can be underflow');
    disp(['You should use the amplitude range of ',...
        '-8191 <= SL <= 8191 for the input signal SL!']);
end;

I = zeros(size(sl)); % allocate output memory (channel symbols)
sr = zeros(size(sl)); % allocate output memory (reconstructed speech)
if (nargout > 2) % allocate memory for diagnostic outputs
```



```
dout = zeros(size(sl));
dqout = zeros(size(sl));
yout = zeros(size(sl));
aiout = zeros(length(sl),2);
biout = zeros(length(sl),6);
tdout = zeros(size(sl));
trout = zeros(size(sl));
alout = zeros(size(sl));

end

% define quantizer tables and scale-factor adaptation tables
if N == 40
    h_fi = fliplr([6 6 5 4 3 2 1 1 1 1 1 0 0 0 0]);
    h_wi = fliplr([43.50 33.06 27.50 22.38 17.50 13.69, ...
        11.19 8.81 6.25 3.63 2.56 2.50 2.44 1.50 0.88 0.88]);
    h_iadq = fliplr([4.42 4.21 4.02 3.81 3.59 3.35 3.09 2.80 2.48, ...
        2.14 1.75 1.32 0.81 0.22 -0.52 -inf]);
    h_qan = fliplr([4.31 4.12 3.91 3.70 3.47 3.22 2.95 2.64 2.32, ...
        1.95 1.54 1.08 0.52 -0.13 -0.96 -inf]);
end;
if N == 32
    h_fi = fliplr([7 3 1 1 1 0 0 0]);
    h_wi = fliplr([70.13 22.19 12.38 7.00 4.00 2.56 1.13 -0.75]);
    h_iadq = fliplr([3.32 2.91 2.52 2.13 1.66 1.05 0.031 -inf]);
    h_qan = fliplr([3.12 2.72 2.34 1.91 1.38 0.62 -0.98 -inf]);
end;
if N == 24
    h_fi = fliplr([7 2 1 0]);
    h_wi = fliplr([36.38 8.56 1.88 -0.25]);
    h_iadq = fliplr([2.91 2.13 1.05 -inf]);
    h_qan = fliplr([2.58 1.70 0.06 -inf]);
end;
if N == 16
    h_fi = fliplr([7 0]);
    h_wi = fliplr([27.44 -1.38]);
    h_iadq = fliplr([2.85 0.91]);
    h_qan = fliplr([2.04 -inf]);
end;

% numerical constants (leakage, step sizes etc.)
c1 = 0.125;           % 2^-3
c2 = 0.9375;         % 1-2^-4
c3 = 0.03125;        % 2^-5
c4 = 0.96875;        % 1-2^-5
c5 = 0.015625;       % 2^-6
c6 = 0.984375;       % 1-2^-6
c7 = 0.0078125;      % 2^-7
c8 = 0.9921875;      % 1-2^-7
c9 = 0.01171875;     % 3*2^-8
c10 = 0.99609375;    % 1-2^-8
```



```
c11 = 0.998046875; % 1-2^-9

% initialize internal variables
yu=1.06; yl=0; dms=0; dml=0; ap=0;
dq = zeros(1,7); % difference signal vector [dq(k),dq(k-1),...,dq(k-6)]
ai = zeros(1,2); % recursive predictor coefficients [a1(k),a2(k)]
bi = zeros(1,6); % non-recursive predictor coefficients
                    % [b1(k),b2(k),...,b6(k)]
p = zeros(1,3); % surrogate vector for recursive predictor
                    % adaptation [p(k),p(k-1),p(k-2)]
srv = zeros(1,2); % reconstructed-speech vector [sr(k),sr(k-1)]

% loop through all signal samples
for k = 1:length(sl)

    % difference signal computation and quantization
    sez = bi * dq(1:6)'; % signal prediction, non-recursive part
    se = ai * srv' + sez; % signal prediction, recursive part
    d = sl(k) - se; % difference signal computation
    al = min(ap, 1); % hard-limit speed control parameter
    y = al*yu + (1-al)*yl; % update quantizer scale factor
    x = log2(abs(d)) - y; % scale difference signal in log domain
    I(k) = sign(d) * (sum(h_qan <= x)-1); % flash quantizer

    % inverse quantization and signal reconstruction
    dqh = sign(I(k)) * 2^(h_iadq(abs(I(k))+1)+y); % inverse quant./log
    dq = [dqh dq(1:6)]; % shift difference signal vector
    p = [dq(1)+sez, p(1:2)]; % shift surrogate vector
    sr(k)= se + dq(1); % reconstruct speech sample
    srv = [sr(k), srv(1)]; % shift reconstructed-speech vector

    % adaptation of predictor coefficients
    ai = aiupdate(ai,p,c9,c10,c7,c8,c2);
    bi = biupdate(bi,dq,N,c7,c10,c11);

    % quantizer scale factor adaptation
    WI = h_wi(abs(I(k))+1);
    yu = c4*y + c3*WI; % fast (unlocked) scale factor
    yu = max(min(yu,10.00),1.06);
    yl = c6*yl + c5*yu; % slow (locked) scale factor

    % tone and transition detection
    td = ai(2) < -0.71875; % partial band signal detection
    tr = td & (abs(dq(1)) > 24*2^yl);
    if tr==1
        ai = zeros(1,2);
        bi = zeros(1,6);
    end;

    % quantizer adaptation speed control
```



```
FI = h_fi(abs(I(k))+1);
dml = c8*dml + c7*FI;           % long term average of F[I(k)]
dms = c4*dms + c3*FI;           % short term average of F[I(k)]
ap = apupdate(ap,dms,dml,y,td,tr,c1,c2); % speed control parameter

% write diagnostic outputs
if (nargout > 2)
    dout(k) = d;
    dqout(k) = dqh;
    yout(k) = y;
    aiout(k,:) = ai;
    biout(k,:) = bi;
    tdout(k) = td;
    trout(k) = tr;
    alout(k) = al;
end
end;
```

4.2. Decoder

```
function reconstructed_signal = ADPCMdecoder(ADPCMsignal, N);

% input argument checks, default bit-rate and format setting
if nargin == 1
    N = 32;
    disp('***** N = 32 kbit/s *****');
end
a = N==[40 32 24 16];
if (sum(a) ~= 1)
    N = 32;
    disp('***** N = 32 kbit/s *****');
end;
if nargin <= 2
    format = 'lin';
end;

reconstructed_signal = zeros(size(ADPCMsignal)); % allocate output memory
(reconstructed_speech)
% define quantizer tables and scale-factor adaptation tables
if N == 40
    h_fi = fliplr([6 6 5 4 3 2 1 1 1 1 1 0 0 0 0]);
    h_wi = fliplr([43.50 33.06 27.50 22.38 17.50 13.69, ...
        11.19 8.81 6.25 3.63 2.56 2.50 2.44 1.50 0.88 0.88]);
    h_iadq = fliplr([4.42 4.21 4.02 3.81 3.59 3.35 3.09 2.80 2.48, ...
        2.14 1.75 1.32 0.81 0.22 -0.52 -inf]);
    h_qan = fliplr([4.31 4.12 3.91 3.70 3.47 3.22 2.95 2.64 2.32, ...
        1.95 1.54 1.08 0.52 -0.13 -0.96 -inf]);
end;
if N == 32
    h_fi = fliplr([7 3 1 1 1 0 0 0]);
```

```

h_wi = fliplr([70.13 22.19 12.38 7.00 4.00 2.56 1.13 -0.75]);
h_iadq = fliplr([3.32 2.91 2.52 2.13 1.66 1.05 0.031 -inf]);
h_qan = fliplr([3.12 2.72 2.34 1.91 1.38 0.62 -0.98 -inf]);
end;
if N == 24
    h_fi = fliplr([7 2 1 0]);
    h_wi = fliplr([36.38 8.56 1.88 -0.25]);
    h_iadq = fliplr([2.91 2.13 1.05 -inf]);
    h_qan = fliplr([2.58 1.70 0.06 -inf]);
end;
if N == 16
    h_fi = fliplr([7 0]);
    h_wi = fliplr([27.44 -1.38]);
    h_iadq = fliplr([2.85 0.91]);
    h_qan = fliplr([2.04 -inf]);
end;

% numerical constants (leakage, step sizes etc.)
c1 = 0.125;           % 2^-3
c2 = 0.9375;         % 1-2^-4
c3 = 0.03125;        % 2^-5
c4 = 0.96875;        % 1-2^-5
c5 = 0.015625;       % 2^-6
c6 = 0.984375;       % 1-2^-6
c7 = 0.0078125;     % 2^-7
c8 = 0.9921875;     % 1-2^-7
c9 = 0.01171875;    % 3*2^-8
c10 = 0.99609375;   % 1-2^-8
c11 = 0.998046875;  % 1-2^-9

% initialize internal variables
yu=1.06; yl=0; dms=0; dml=0; ap=0;
dq = zeros(1,7);     % difference signal vector [dq(k),dq(k-1),...,dq(k-6)]
ai = zeros(1,2);     % recursive predictor coefficients [a1(k),a2(k)]
bi = zeros(1,6);     % non-recursive predictor coefficients
                    % [b1(k),b2(k),...,b6(k)]
p = zeros(1,3);      % surrogate vector for recursive predictor
                    % adaptation [p(k),p(k-1),p(k-2)]
srv = zeros(1,2);    % reconstructed-speech vector
[reconstructed_signal(k),reconstructed_signal(k-1)]

% loop through all signal samples
for k = 1:length(ADPCMsignal)

    % difference signal computation and quantization
    sez = bi * dq(1:6)'; % signal prediction, non-recursive part
    se = ai * srv' + sez; % signal prediction, recursive part
    al = min(ap, 1);      % hard-limit speed control parameter
    y = al*yu + (1-al)*yl; % update quantizer scale factor

```

```

    % inverse quantization and signal reconstruction
    dqh = sign(ADPCMsignal(k)) * 2^(h_iadq(abs(ADPCMsignal(k))+1)+y);    %
inverse quant./log
    dq = [dqh dq(1:6)];          % shift difference signal vector
    p = [dq(1)+sez, p(1:2)]; % shift surrogate vector
    reconstructed_signal(k)= se + dq(1);          % reconstruct speech
sample
    srv = [reconstructed_signal(k), srv(1)];    % shift reconstructed-speech vector

    % adaptation of predictor coefficients
    ai = aiupdate(ai,p,c9,c10,c7,c8,c2);
    bi = biupdate(bi,dq,N,c7,c10,c11);

    % quantizer scale factor adaptation
    WI = h_wi(abs(ADPCMsignal(k))+1);
    yu = c4*y + c3*WI;          % fast (unlocked) scale factor
    yu = max(min(yu,10.00),1.06);
    yl = c6*yl + c5*yu;          % slow (locked) scale factor

    % tone and transition detection
    td = ai(2) < -0.71875;          % partial band signal detection
    tr = td & (abs(dq(1)) > 24*2^yl);
    if tr==1
        ai = zeros(1,2);
        bi = zeros(1,6);
    end;

    % quantizer adaptation speed control
    FI = h_fi(abs(ADPCMsignal(k))+1);
    dml = c8*dml + c7*FI;          % long term average of F[ADPCMsignal(k)]
    dms = c4*dms + c3*FI;          % short term average of F[ADPCMsignal(k)]
    ap = apupdate(ap,dms,dml,y,td,tr,c1,c2); % speed control parameter

end

```

4.3. AI Update

```

function ai = aiupdate(aio,p,c1,c2,c3,c4,c5);
%AIUPDATE      Update of recursive predictor coefficients ai.
% ai = aiupdate(aio,p,c1,c2,c3,c4,c5)
%
% function:    update a1 and a2 coefficient of second-order predictor
%
% input:aio    old coefficient of second-order predictor
%             p        surrogate vector
%             c1      = 3*2^-8
%             c2      = 1-2^-8
%             c3      = 2^-7
%             c4      = 1-2^-7
%             c5      = 1-2^-4
%

```

```
% output:    ai    coefficients of second-order recursive predictor
```

```
% 1995-05-16, Martin Kummernecker
% 1995-08-30, Gernot Kubin (g.kubin@ieee.org)
% Vienna University of Technology, Vienna, Austria
```

```
sgnp = sign(p);
sgnp(2:3) = sign(sgnp(2:3) + 0.5); % modify 0's to 1's
```

```
ai(1) = c2*aio(1) + c1*sgnp(1)*sgnp(2);
if abs(aio(1)) > 0.5
    fa1 = 2*sign(aio(1));
else
    fa1 = 4*aio(1);
end;
ai(2) = c4*aio(2) + c3*(sgnp(1)*sgnp(3) - fa1*sgnp(1)*sgnp(2));
ai(2) = max(min(ai(2),0.75),-0.75);
b = c5-ai(2);
ai(1) = max(min(ai(1),b),-b);
```

4.4. AP Update

```
function ap = apupdate(apo,dms,dml,y,td,tr,c1,c2);
% ap=apupdate(apo,dms,dml,y,td,tr,c1,c2)
%
% function:    update speed control parameter ap(k)
%              implements eq. (7) of recommendation G.726
%
% input:apo    old speed control parameter ap(k-1)
%            dml    long term average of F[I(k)]
%            dms    short term average of F[I(k)]
%            y      quantizer scale factor y(k)
%            td     tone detector
%            tr     transition detector
%            c1     = 2^-3
%            c2     = 1-2^-4
%
% output:     ap    speed control parameter
```

```
if tr==1
    ap = 1;
elseif ( abs(dms-dml) >= c1*dml | y < 3 | td==1 )
    ap = c2*apo + c1;
else
    ap = c2*apo;
end;
```

4.5. BI Update

```
function bi = biupdate(bio,dq,N,c1,c2,c3);
% BIUPDATE    Update of non-recursive predictor coefficients
```

```

% bi=biupdate(bio,dq,N,c1,c2,c3)
%
% function:   update coefficients of sixth-order predictor
%
% input:     bio    old coefficient of sixth-order predictor
%            dq     difference signal vector
%            c1     = 2^-7
%            c2     = 1-2^-8
%            c3     = 1-2^-9
%
% output:    bi     coefficients of sixth-order predictor

% 1995-05-16, Martin Kummernecker
% 1995-08-30, Gernot Kubin (g.kubin@ieee.org)
% Vienna University of Technology, Vienna, Austria

sgndq = sign(dq);
sgndq(2:6) = sign(sgndq(2:6) + 0.5); % modify 0's to 1's
if N == 40
    bi=c3*bio+c1*sgndq(1)*sgndq(2:7);
else
    bi=c2*bio+c1*sgndq(1)*sgndq(2:7);
end;

```


Chapter 5 - AWGN Module

5.1. AWGN code

```

function AWGN_noise(mode)
%=====
% Random noise
%=====
SNR_range = 0:25 ;

figure ;
for i = 1:4,
    if i == 1
        load data\BinarySamples1.mat binary ;
    elseif i == 2
        load data\BinarySamples2.mat binary ;
    elseif i == 3
        load data\BinarySamples3.mat binary ;
    else
        load data\BinarySamples4.mat binary ;
    end

    BER = calculate_BER(binary) ;
    hold on ;
    plot(SNR_range, BER, color(i)) ;
end
title('Plot of SNR(dB) versus BER(%)') ;
legend('Uniform PCM', 'A-law PCM', 'Mu-law PCM', 'DPCM') ;

hold off ;

function BER = calculate_BER(binary)

for i = 0:25, % SNR range
    awgn_noised_signal = awgn(binary,i,'measured',0,'db') ;

        for j = 1:length(awgn_noised_signal),
            if awgn_noised_signal(j) >= 0.5
                noised_signal(i+1,j) = 1 ;
            else
                noised_signal(i+1,j) = 0 ;
            end

            if noised_signal(i+1,j) ~= binary(j)
                error_signal(i+1,j) = 1 ;
            else
                error_signal(i+1,j) = 0 ;
            end
        end
    end
end

```



```
end

Bits_in_Error = sum(error_signal,2) ;
BER = (Bits_in_Error / (length(binary)))*100 ;
function index = color(iteration)
switch iteration
case 1
index = 'r' ;
case 2
index = 'b' ;
case 3
index = 'g' ;
otherwise
index = 'k' ;
end
```

5.2. Random noise

```
%function random_noise = PCMencoder(encoded_samples);
%=====
% Random noise error = input('Enter the number of error in % : '); % no of levels
%=====

load binPCM.mat binary_samples ;
headlength = 256; % assumes SPPACK headers
header = encoded_samples(1:headlength);
encoded_sig = binary_samples(headlength+1: length(binary_samples));

error_p = error/100;
no_of_bits_in_error = floor(error_p * length(encoded_sig)) ;

no_of_groups = floor(length(encoded_sig) / no_of_bits_in_error) ;
err_pattern = [] ;
for i = 1:no_of_bits_in_error,
error_pattern = [err_pattern randerr(1, no_of_groups)] ;
end
err_pattern = [error_pattern zeros(1,length(encoded_sig)-length(error_pattern))] ;

input_bin = [] ;
for i = 1:length(encoded_sig),
input_bin(i) = numeric(encoded_sig(i)) ;
if err_pattern(i) == 1
if encoded_sig(i) == 1
errored_sig(i) = 0 ;
else
errored_sig(i) = 1 ;
end
else
errored_sig(i) = numeric(encoded_sig(i)) ;
end
end
```

```
%Noise_power = power(random_noise,2);
%Signal_power = power(encoded_sig,2);

%SNR = Signal_power' / Noise_power ;
%SNR_db = 10*log(SNR)

channel_noise = errored_sig' + encoded_sig ;
channel_sig = [header ; channel_noise] ;
save noise.mat channel_sig;

t= 1:length(encoded_sig) ;
subplot(2,2,1)
plot (t, channel_noise, 'k') ;
%hold on
subplot(2,2,2)
plot (t, encoded_sig, 'b') ;
```



AGDER UNIVERSITY
COLLEGE

Faculty of Engineering and
Science



THE UNIVERSITY OF QUEENSLAND

School of Information Technology and
Electrical Engineering



Appendix C – Visual Basic V.6.0 Code

—
Visual Basic V.6.0

—
For
Agder University College
&
The University of Queensland

—
by
Knut Ole Hauge
Svein Birger Skogly

Chapter 1 - Visual basic V.6.0 Code

1.1. form1.frm

Option Explicit

```
Private Declare Function Playsound Lib "winmm.dll" Alias "PlaySoundA" (ByVal
IpszName As String, ByVal hModule As Long, ByVal dwFlags As Long) As Long
Private Declare Function sndPlaySound Lib "winmm.dll" Alias "sndPlaySoundA"
(ByVal IpszSoundName As String, ByVal uFlags As Long) As Long
```

```
Const SND_SYNC = &H0      ' play synchronously (default)
Const MWAVE = &H1        ' The program will continue operation with the
sound playing at the same time
Const SND_NODEFAULT = &H2 ' silence not default, if sound not found
Const SNDSTOP = &H16     ' Stop playing song
Dim i As String
```

```
Private Sub cmdChooseSendFile_Click()
    CommonDialog1.ShowOpen
    lblfiletosend.Text = CommonDialog1.FileName
    cmdSendFile.Enabled = True ' Enables the Send File command
End Sub
```

```
Private Sub cmdExit_Click()
    End
End Sub
```

```
Private Sub cmdSendFile_Click()
    Dim b(0 To 8000) As Byte
    On Error GoTo ErrHandler3

    Open lblfiletosend.Text For Binary Access Read As #1
    Get #1, , b

    UDP_socket.SendData b
    'Close lblfiletosend.Text

    If EOF(1) Then
        MsgBox "SENT"
    Else
        MsgBox "ERROR"
    End If
ErrHandler3:
End Sub
```

```
Private Sub Command1_Click()
    CommonDialog1.CancelError = True
    On Error GoTo ErrHandler1
    CommonDialog1.Filter = "WAV file (*.wav*)|*.wav"
```



```
CommonDialog1.Flags = &H2 Or &H400  
CommonDialog1.ShowSave  
cmdSendFile.Enabled = True ' Enables the Send File command
```

```
'If file already exists then remove it  
FileFound CommonDialog1.FileName  
If ValidFile = True Then  
Kill CommonDialog1.FileName  
End If
```

```
'MCI command to save the WAV file  
i = mciSendString("save capture " & CommonDialog1.FileName, 0&, 0, 0)
```

```
ErrorHandler1:  
End Sub
```

```
Private Sub Command2_Click()  
'Samples Per Second that are supported:  
'11025 low quality  
'22050 medium quality  
'44100 high quality (CD music quality)  
'Bits per sample is 16 or 8  
'Channels are 1 (mono) or 2 (stereo)  
  
i = mciSendString("seek capture to start", 0&, 0, 0) 'Always start at the beginning  
i = mciSendString("set capture samplespersec 11025", 0&, 0, 0) 'CD Quality  
i = mciSendString("set capture bitspersample 8", 0&, 0, 0) '16 bits for better  
sound  
i = mciSendString("set capture channels 1", 0&, 0, 0) ' 1 channel(s) for mono  
i = mciSendString("record capture", 0&, 0, 0) 'Start the recording  
  
Command3.Enabled = True 'Enable the STOP BUTTON  
Command4.Enabled = False 'Disable the "PLAY" button  
Command1.Enabled = False 'Disable the "SAVE AS" button  
End Sub
```

```
Private Sub Command3_Click()  
i = mciSendString("stop capture", 0&, 0, 0)  
  
Command1.Enabled = True 'Enable the "SAVE AS" button  
Command4.Enabled = True 'Enable the "PLAY" button  
End Sub
```

```
Private Sub Command4_Click()  
i = mciSendString("play capture from 0", 0&, 0, 0)  
End Sub
```

```
Private Sub Command5_Click()  
Dim msg As String  
Dim mssg As String * 255
```



```
i = mciSendString("set capture time format ms", 0&, 0, 0)
i = mciSendString("status capture length", mssg, 255, 0)
  msg = "Milliseconds = " & Str(mssg) & vbCrLf
i = mciSendString("set capture time format bytes", 0&, 0, 0)
i = mciSendString("status capture length", mssg, 255, 0)
  msg = msg & "Bytes = " & Str(mssg) & vbCrLf
i = mciSendString("status capture channels", mssg, 255, 0)
If Str(mssg) = 1 Then
  msg = msg & "Channels = 1 (mono)" & vbCrLf
ElseIf Str(mssg) = 2 Then
  msg = msg & "Channels = 2 (stereo)" & vbCrLf
End If

i = mciSendString("status capture bitspersample", mssg, 255, 0)
  msg = msg & "Bits per sample = " & Str(mssg) & vbCrLf
i = mciSendString("status capture bytespersec", mssg, 255, 0)
  msg = msg & "Bytes per second = " & Str(mssg) & vbCrLf

Label3.Caption = msg
End Sub

Private Sub Form_Load()

  ' The control's name is UDP_socket.
  With UDP_socket
    .RemoteHost = "localhost" ' Remote IP (rabbit)
    .RemotePort = 1002      ' Port to connect to.
    .Bind 1001              ' Bind to the local port.
  End With

  'Close any MCI operations from previous VB programs
  i = mciSendString("close all", 0&, 0, 0)

  'Open a new WAV with MCI Command...
  i = mciSendString("open new type waveaudio alias capture", 0&, 0, 0)
End Sub

Private Sub Form_Unload(Cancel As Integer)
  i = mciSendString("close capture", 0&, 0, 0)
End Sub

Private Sub Timer1_Timer()
  Dim mssg As String * 255

  i = mciSendString("status capture mode", mssg, 255, 0)
  Label1.Caption = " " & mssg
End Sub
```



1.2. *Module1.bas*

```
*****
'Windows API/Global Declarations for :FileFound()
*****
Public Declare Function FindFirstFile& Lib "kernel32" Alias "FindFirstFileA"
(ByVal lpFileName As String, lpFindFileData As WIN32_FIND_DATA)

Public Declare Function FindClose Lib "kernel32" (ByVal hFindFile As Long) As
Long

Public Const MAX_PATH = 260

Type FILETIME ' 8 Bytes
    dwLowDateTime As Long
    dwHighDateTime As Long
End Type

Type WIN32_FIND_DATA ' 318 Bytes
    dwFileAttributes As Long
    ftCreationTime As FILETIME
    ftLastAccessTime As FILETIME
    ftLastWriteTime As FILETIME
    nFileSizeHigh As Long
    nFileSizeLow As Long
    dwReserved As Long
    dwReserved1 As Long
    cFileName As String * MAX_PATH
    cAlternate As String * 14
End Type
Declare Function mciSendString Lib "winmm.dll" Alias "mciSendStringA" (ByVal
lpstrCommand As String, ByVal lpstrReturnString As Any, ByVal uReturnLength
As Long, ByVal hwndCallback As Long) As Long

Global ValidFile As Boolean

Function FileFound(strFileName As String) As Boolean

Dim lpFindFileData As WIN32_FIND_DATA
Dim hFindFirst As Long

    hFindFirst = FindFirstFile(strFileName, lpFindFileData)

    If hFindFirst > 0 Then
        FindClose hFindFirst
        ValidFile = True
    Else
        ValidFile = False
    End If
End Function
```




AGDER UNIVERSITY
COLLEGE


Faculty of Engineering and
Science



THE UNIVERSITY OF QUEENSLAND

School of Information Technology and
Electrical Engineering

```

while(1)
(

tcp_tick(NULL); //Process all s
pass=udp_recvfrom(&udpSocket, //ci
udpBuffer,
UDP_BUFFER_LEN,
&remoteIP,
&remotePort);

tcp_tick(NULL);
pass_cmd=udp_recvfrom(&cmdSocket,
udpBuffer,
UDP_BUFFER_LEN,
&remoteIP,
&remoteCmdPort);

```

Appendix D – Dynamic C Code

–
Dynamic C code

–
For
Agder University College
&
The University of Queensland

–
by
Knut Ole Hauge
Svein Birger Skogly

Chapter 1 - Dynamic C Code

1.1. UDP_srv.c

```

#class auto
#define TCPCONFIG 1
#define PORTA_AUX_IO
#define MAX_UDP_SOCKET_BUFFERS 1
/* what local UDP port to use - we receive packets only sent to this port */
#define LOCAL_PORT    1234

/* If this is set to "0", we will accept a connection from anybody.
 * If it is set to all "255"s, we will receive all broadcast packets instead.
 */
#define REMOTE_IP          "0"
/*#define REMOTE_IP      "255.255.255.255" broadcast*/
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/timeb.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <string.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/timeb.h>

udp_Socket sock;

/* receive one packet (heartbeat) */
int receive_packet(void)
{
    static char buf[128];

    #GLOBAL_INIT
    {
        memset(buf, 0, sizeof(buf));
    }

    /* receive the packet */
    if (-1 == udp_rcv(&sock, buf, sizeof(buf))) {
        /* no packet read. return */
        return 0;
    }

    printf("Received-> %s\n",buf);
    return 1;
}

void main()
{
    sock_init();
    /*printf("Opening UDP socket\n");*/

    if(!udp_open(&sock, LOCAL_PORT, resolve(REMOTE_IP), 0, NULL)) {
        printf("udp_open failed!\n");
        exit(0);
    }
}

```



```
    }  
  
    /* receive heartbeats */  
    for(;;) {  
        tcp_tick(NULL);  
        receive_packet();  
    }  
}
```



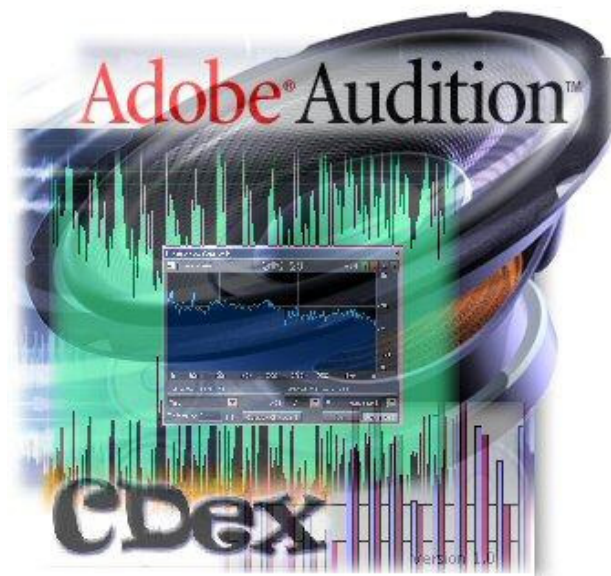
AGDER UNIVERSITY
COLLEGE

Faculty of Engineering and
Science



THE UNIVERSITY OF QUEENSLAND

School of Information Technology and
Electrical Engineering



Appendix E – Subjective assessment of audio quality

–

Music samples

Speech samples

Speech with introduced errors

–

For

Agder University College

&

The University of Queensland

–

by

Knut Ole Hauge

Svein Birger Skogly



Table of contents

TABLE OF CONTENTS	2
LIST OF TABLES	2
CHAPTER 1 - RATING OF AUDIO SAMPLES	3
1.1. RATING OF MUSIC SAMPLES	3
1.2. RATING OF SPEECH SAMPLES	4
CHAPTER 2 - RESULTS	5
2.1. TRANCE	5
2.2. POP	6
2.3. CLASSICAL	8
2.4. SPEECH	9

List of tables

TABLE 1.1:1 – QUALITY SCALE	3
TABLE 1.1:1 - MUSIC SAMPLES	3
TABLE 2.1:1 - TRANCE SAMPLES, RESULTS PART I	5
TABLE 2.1:2 - TRANCE SAMPLES, RESULTS PART II	5
TABLE 2.1:3 - TRANCE SAMPLES, RESULTS PART III	6
TABLE 2.2:1 - POP SAMPLES, RESULTS PART I	6
TABLE 2.2:2 - POP SAMPLES, RESULTS PART II	7
TABLE 2.2:3 - POP SAMPLES, RESULTS PART III	7
TABLE 2.3:1 - CLASSICAL SAMPLES, RESULTS PART I	8
TABLE 2.3:2 - CLASSICAL SAMPLES, RESULTS PART II	8
TABLE 2.3:3 - CLASSICAL SAMPLES, RESULTS PART III	9
TABLE 2.4:1 - SPEECH SAMPLES, RESULTS PART I	9
TABLE 2.4:2 - SPEECH SAMPLES, RESULTS PART II	10
TABLE 2.4:3 - SPEECH SAMPLES, RESULTS PART III	10
TABLE 2.4:4 - SPEECH SAMPLES, RESULTS PART IV	11

Chapter 1 - Rating of audio samples

1. Some samples will be played to raise awareness regarding the quality of the sound (best and worst) of the various samples.
2. The samples will then be played in random order and rated from 1 to 5, where 1 is bad and 5 is excellent. See *Table 1:1 - Quality scale*.
3. If one sample sounds worse, equal or better to the previous sample, indicate this by means of a W for WORSE, E for EQUAL or B for BETTER in the WEB column.
4. (SPEECH ONLY) Indicate by means of an E in the Error column if you think there have been added errors or an error burst to the sample.

Table 1.1:1 – Quality Scale

Rating:	Explanation:	
1	Bad	Very annoying distortion which is objectionable
2	Poor	Annoying distortion but not objectionable
3	Adequate	Perceptible distortion that is slightly annoying
4	Good	Slight perceptible level of distortion but not annoying
5	Excellent	Imperceptible level of distortion

1.1. Rating of Music samples

Table 1.1:1 - Music samples

No:	Trance: Paul Van Dyk		Pop: Smashing Pumpkins		Classical: J.S Bach	
	Quality: (1 to 5)	WEB:	Quality: (1 to 5)	WEB:	Quality: (1 to 5)	WEB:
1						
2						
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						
15						
16						
17						
18						



1.2. Rating of Speech samples

Table 1 - Speech samples

Sample:	Quality: (1 to 5)	WEB:	Error: (E)
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			



Chapter 2 - Results

Eight participants took the audio-quality test. The results of these tests are presented in the subchapter under the respective genres. The performance of the test is explained in the *main report chapter 4.2*

2.1. Trance

Table 2.1:1 - Trance samples, results part I

Played:	No:	1		2		3		4	
		Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB
PCM μ -law	1	3		3		3		3	
PCM μ -law	2	3	E	3	E	2	W	3	E
ADPCM 16	3	2	W	1	W	1	W	1	W
ADPCM 32	4	5	W	4	B	4	B	2	B
WMA 320	5	4	W	4	W	5	B	5	W
MP3 320	6	3	W	2	W	3	W	5	W
ADPCM 24	7	2	W	2	W	3	E	1	W
PCM A-law	8	2	B	2	B	2	W	2	B
WMA 128	9	5	B	5	B	5	B	5	B
ADPCM 40	10	2	W	3	W	4	W	1	W
ADPCM 40	11	2	E	2	W	3	W	1	E
MP3 80	12	4	B	4	B	5	B	4	B
WMA 80	13	5	B	5	B	5	E	4	W
MP3 30	14	2	W	1	W	2	W	2	W
WMA 30	15	4	B	3	B	4	B	3	B
WMA 30	16	4	E	3	E	5	B	3	E
Ogg-V 160	17	5	B	5	B	5	B	5	B
Mp3 128	18	5	E	4	W	5	E	4	W

Table 2.1:2 - Trance samples, results part II

Played:	No:	5		6		7		8	
		Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB
PCM μ -law	1	3		2		1		4	
PCM μ -law	2	2	E	2	E	2	B	4	E
ADPCM 16	3	1	W	1	W	1	W	2	W
ADPCM 32	4	3	B	2	B	1	W	4	B
WMA 320	5	5	B	5	B	5	B	5	B
MP3 320	6	5	E	4	W	4	W	4	W
ADPCM 24	7	2	W	1	W	2	W	2	W
PCM A-law	8	3	E	1	E	3	B	3	B
WMA 128	9	5	B	4	B	4	B	4	B
ADPCM 40	10	3	W	3	W	1	W	3	W
ADPCM 40	11	3	E	3	E	3	B	3	E
MP3 80	12	4	B	4	B	3	E	5	B
WMA 80	13	5	B	4	E	4	B	5	E
MP3 30	14	3	W	2	W	1	W	4	W
WMA 30	15	3	E	4	B	2	B	4	E
WMA 30	16	3	E	4	E	2	E	3	W
Ogg-V 160	17	5	B	3	W	2	W	2	W
Mp3 128	18	5	E	4	B	4	B	4	B



Table 2.1:3 - Trance samples, results part III

Played:	No:	Average:	Expected rate:
		Quality: (1 to 5)	Quality: (1 to 5)
PCM μ -law	1	2,4	3
PCM μ -law	2	2,3	3
ADPCM 16	3	1,1	1
ADPCM 32	4	2,8	1
WMA 320	5	4,2	5
MP3 320	6	3,3	5
ADPCM 24	7	1,7	1
PCM A-law	8	2,0	3
WMA 128	9	4,1	5
ADPCM 40	10	2,2	2
ADPCM 40	11	2,2	2
MP3 80	12	3,7	4
WMA 80	13	4,1	4
MP3 30	14	1,9	3
WMA 30	15	3,0	3
WMA 30	16	3,0	3
Ogg-V 160	17	3,6	5
Mp3 128	18	3,9	5

2.2. Pop

Table 2.2:1 - Pop samples, results part I

Played:	No:	1		2		3		4	
		Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB
WMA 320	1	5		5		4		5	
WMA 128	2	5	E	5	B	4	E	5	E
WMA 80	3	5	E	5	E	3	W	5	E
WMA 30	4	4	W	4	W	4	B	3	W
MP3 320	5	5	B	5	E	4	E	5	B
MP3 128	6	5	B	5	E	4	E	5	E
MP3 80	7	5	W	4	W	3	W	4	W
MP3 30	8	3	W	2	W	1	W	2	W
MP3 30	9	2	W	2	E	2	B	2	E
Ogg-V 160	10	5	B	4	B	5	B	5	B
PCM μ -law	11	1	W	1	W	3	W	3	W
PCM A-law	12	1	E	1	B	3	E	3	E
ADPCM 40	13	1	W	1	W	2	W	2	W
ADPCM 32	14	1	W	1	E	2	W	1	W
ADPCM 24	15	1	E	1	E	1	W	1	W
ADPCM 16	16	1	W	1	W	1	E	1	W
ADPCM 16	17	1	W	1	E	1	E	1	W



Table 2.2:2 - Pop samples, results part II

Played:	No:	5		6		7		8	
		Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB
WMA 320	1	5		5		4		5	
WMA 128	2	4	W	5	E	4	E	5	E
WMA 80	3	5	B	4	W	5	B	4	W
WMA 30	4	3	W	3	W	3	W	2	W
MP3 320	5	4	B	4	B	5	B	3	B
MP3 128	6	4	E	4	E	5	E	5	B
MP3 80	7	3	W	3	W	4	W	2	W
MP3 30	8	1	W	1	W	1	W	1	W
MP3 30	9	2	W	1	E	1	B	2	B
Ogg-V 160	10	5	B	5	B	5	B	3	B
PCM μ -law	11	1	W	1	W	1	W	1	W
PCM A-law	12	1	E	1	E	2	B	2	B
ADPCM 40	13	2	B	1	E	1	W	2	E
ADPCM 32	14	2	E	1	E	1	E	3	B
ADPCM 24	15	2	E	2	B	1	W	2	W
ADPCM 16	16	1	W	1	W	1	E	1	W
ADPCM 16	17	1	E	2	B	1	W	1	E

Table 2.2:3 - Pop samples, results part III

Played:	No:	Average:	Expected rate:
		Quality: (1 to 5)	Quality: (1 to 5)
WMA 320	1	4,8	5
WMA 128	2	4,6	5
WMA 80	3	4,5	4
WMA 30	4	3,3	3
MP3 320	5	4,4	5
MP3 128	6	4,6	5
MP3 80	7	3,5	4
MP3 30	8	1,5	2
MP3 30	9	1,8	2
Ogg-V 160	10	4,6	5
PCM μ -law	11	1,5	2
PCM A-law	12	1,8	2
ADPCM 40	13	1,5	1
ADPCM 32	14	1,5	1
ADPCM 24	15	1,4	1
ADPCM 16	16	1,0	1
ADPCM 16	17	1,1	1



2.3. Classical

Table 2.3:1 - Classical samples, results part I

Played:	No:	1		2		3		4	
		Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB
ADPCM 40	1	3		3		3		2	
ADPCM 32	2	3	W	3	E	4	B	2	E
ADPCM 24	3	1	W	2	W	2	W	2	W
ADPCM 16	4	1	W	1	W	1	W	1	W
PCM A-law	5	3	B	3	B	4	B	3	W
PCM A-law	6	2	W	2	W	4	E	3	B
PCM μ -law	7	2	B	3	B	3	W	3	E
PCM μ -law	8	2	B	3	E	4	B	3	E
WMA 320	9	5	B	4	B	5	B	5	E
MP3 320	10	4	W	4	B	5	E	5	B
WMA 30	11	4	E	3	W	5	E	4	E
MP3 30	12	3	W	3	W	4	W	3	W
WMA 80	13	5	B	5	B	5	B	2	W
MP3 80	14	5	W	4	W	5	E	4	B
WMA 128	15	5	B	5	B	4	W	4	E
MP3 128	16	5	E	5	E	5	B	5	B
Ogg-V 160	17	4	W	5	B	4	W	5	E
Ogg-V 160	18	5	B	5	E	4	E	5	E

Table 2.3:2 - Classical samples, results part II

Played:	No:	5		6		7		8	
		Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB	Quality: (1 to 5)	WEB
ADPCM 40	1	2		3		2		3	
ADPCM 32	2	2	E	4	B	2	W	2	W
ADPCM 24	3	2	E	3	W	1	W	3	B
ADPCM 16	4	1	W	2	W	1	W	2	W
PCM A-law	5	2	B	4	B	3	B	4	B
PCM A-law	6	2	E	2	W	2	W	3	W
PCM μ -law	7	2	E	3	E	2	W	4	B
PCM μ -law	8	2	E	4	B	1	W	4	E
WMA 320	9	4	B	5	B	4	B	5	B
MP3 320	10	4	E	5	E	5	B	5	E
WMA 30	11	3	W	4	W	3	W	4	W
MP3 30	12	2	W	1	W	3	W	2	W
WMA 80	13	4	B	3	B	4	B	4	B
MP3 80	14	3	W	4	B	5	B	4	E
WMA 128	15	4	B	4	E	5	B	5	B
MP3 128	16	4	E	3	W	5	E	5	E
Ogg-V 160	17	5	B	2	W	5	E	4	W
Ogg-V 160	18	5	E	3	B	5	B	4	E



Table 2.3:3 - Classical samples, results part III

Played:	No:	Average:	Expected rate:
		Quality: (1 to 5)	Quality: (1 to 5)
ADPCM 40	1	2,6	1
ADPCM 32	2	2,8	1
ADPCM 24	3	2,0	1
ADPCM 16	4	1,3	1
PCM A-law	5	3,3	3
PCM A-law	6	2,5	3
PCM μ -law	7	2,8	3
PCM μ -law	8	2,9	3
WMA 320	9	4,6	5
MP3 320	10	4,6	5
WMA 30	11	3,8	3
MP3 30	12	2,6	3
WMA 80	13	4,0	4
MP3 80	14	4,3	4
WMA 128	15	4,5	5
MP3 128	16	4,6	5
Ogg-V 160	17	4,3	5
Ogg-V 160	18	4,5	5

2.4. Speech

Table 2.4:1 - Speech samples, results part I

Played:	No:	1			2			3		
		Quality (1 to 5)	W E B	Error	Quality (1 to 5)	W E B	Error	Quality (1 to 5)	W E B	Error
PCM μ -law	1	4			5			5		
ADPCM 40	2	4	W		4	W		4	W	
Mp3 48	3	4	E		4	E		3	W	
WMA 48	4	5	B		5	B		4	B	
PCM A-law	5	5	W		5	E		4	E	
PCM A-law	6	4	W		5	E		5	B	
ADPCM 32	7	3	W		4	W		5	B	
Mp3 48	8	3	W		3	W		4	W	
ADPCM 24E	9	2	W	1	2	W	1	2	W	
WMA 48	10	5	B		4	B		4	B	
ADPCM 24EB	11	2	W	1	2	W	1	2	W	
PCM A-law 64E	12	4	B		4	B	1	2	E	1
ADPCM 16	13	1	W		1	W		1	W	
PCM A-law 64EB	14	5	B	1	4	B	1	3	B	1
ADPCM 24	15	2	W		2	W		2	W	
LD-CELP 16	16	1	W		1	W		1	W	1



Table 2.4:2 - Speech samples, results part II

		4			5			6		
		Quality (1 to 5)	W E B	Error	Quality (1 to 5)	W E B	Error	Quality (1 to 5)	W E B	Error
Played:	No:									
PCM μ -law	1	5			5			5		
ADPCM 40	2	3	W		4	W		4	W	
Mp3 48	3	4	B		5	B		4	E	
WMA 48	4	4	E		5	E	1	4	W	1
PCM A-law	5	5	B		5	E		5	B	
PCM A-law	6	5	E		4	W		3	W	
ADPCM 32	7	2	W		3	W		2	W	
Mp3 48	8	5	B		4	B		2	E	
ADPCM 24E	9	2	W	1	3	W	1	1	W	1
WMA 48	10	5	B		5	B		5	B	
ADPCM 24EB	11	1	W	1	3	W	1	3	W	1
PCM A-law 64E	12	2	B	1	3	E		4	B	1
ADPCM 16	13	1	W		2	W		1	W	
PCM A-law 64EB	14	4	B	1	4	B	1	5	B	1
ADPCM 24	15	2	W		3	W		3	W	
LD-CELP 16	16	1	W		2	W		2	W	1

Table 2.4:3 - Speech samples, results part III

		7			8		
		Quality (1 to 5)	W E B	Error	Quality (1 to 5)	W E B	Error
Played:	No:						
PCM μ -law	1	4			4		
ADPCM 40	2	3	W		3	W	
Mp3 48	3	4	B		4	B	
WMA 48	4	3	W	1	5	B	1
PCM A-law	5	3	E		4	W	
PCM A-law	6	3	E		4	E	
ADPCM 32	7	2	W		3	W	
Mp3 48	8	3	B		3	E	
ADPCM 24E	9	1	W		1	W	
WMA 48	10	5	B		4	B	1
ADPCM 24EB	11	2	W	1	3	W	
PCM A-law 64E	12	3	B	1	2	W	1
ADPCM 16	13	1	W		1	W	
PCM A-law 64EB	14	4	B	1	4	B	1
ADPCM 24	15	2	W		2	W	
LD-CELP 16	16	1	W		1	W	



Table 2.4:4 - Speech samples, results part IV¹

Played:	No:	Average:		Expected:	
		Quality: (1 to 5)	Error:	Error: (E)	Quality: (1 to 5)
PCM μ -law	1	4,7	0	1	5
ADPCM 40	2	3,7	0	2	4
Mp3 48	3	4,0	0	3	4
WMA 48	4	4,4	4	4	4
PCM A-law	5	4,6	0	5	5
PCM A-law	6	4,2	0	6	5
ADPCM 32	7	3,1	0	7	3
Mp3 48	8	3,3	0	8	4
ADPCM 24E	9	1,8	5	9E	2
WMA 48	10	4,6	1	10	4
ADPCM 24EB	11	2,2	6	11EB	2
PCM A-law 64E	12	3,1	6	12E	4
ADPCM 16	13	1,1	0	13	1
PCM A-law 64EB	14	4,1	8	14EB	4
ADPCM 24	15	2,2	0	15	2
LD-CELP 16	16	1,2	2	16	1

E = Error, EB = Error Burst

The participants were to put an E in the Error column when the thought they heard an error in the sample. Test examples of error samples were played before the rating test.

¹ E indicates an Error, EB indicates an Error Burst