Mika Sundvall

Opus Audio Codec in Mobile Networks

School of Electrical Engineering

Thesis submitted for examination for the degree of Master of Science in Technology.

Espoo 21.11.2014

Thesis supervisor:

Prof. Vesa Välimäki

Thesis advisor:

Lic.Sc. (Tech.) Jyri Suvanen

Aalto University
School of Electrical Engineering

AALTO UNIVERSITY SCHOOL OF ELECTRICAL ENGINEERING	ABSTRACT OF THE MASTER'S THESIS

Author: Mika Sundvall

Title: Opus Audio Codec in Mobile Networks

Date: 21.11.2014 Language: English Number of pages: 8+74

Department of Signal Processing and Acoustics

Professorship: Acoustics and Audio Signal Processing Code: S-89

Supervisor: Prof. Vesa Välimäki

Advisor: Lic.Sc. (Tech.) Jyri Suvanen

The latest generations in mobile networks have enabled a possibility to include high quality audio coding in data transmission. On the other hand, an on-going effort to move the audio signal processing from dedicated hardware to data centers with generalized hardware introduces a challenge of providing enough computational power needed by the virtualized network elements.

This thesis evaluates the usage of a modern hybrid audio codec called Opus in a virtualized network element. It is performed by integrating the codec, testing it for functionality and performance on a general purpose processor, as well as evaluating the performance in comparison to the digital signal processor's performance. Functional testing showed that the codec was integrated successfully and bit compliance with the Opus standard was met.

The performance results showed that although the digital signal processor computes the encoder's algorithms with less clock cycles, related to the processor's whole capacity the general purpose processor performs more efficiently due to higher clock frequency. For the decoder this was even clearer, when the generic hardware spends on average less clock cycles for performing the algorithms.

Keywords: speech codecs, transcoding, digital signal processing, software testing, digital signal processors, microprocessors

AALTO-YLIOPISTO
SÄHKÖTEKNIIKAN KORKEAKOULU

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä: Mika Sundvall

Työn nimi: Opus audiokoodekki matkapuhelinverkoissa

Päivämäärä: 21.11.2014 Kieli: Englanti Sivumäärä: 8+74

Signaalinkäsittelyn ja akustiikan laitos

Professuuri: Akustiikka ja äänenkäsittely Koodi: S-89

Valvoja: Prof. Vesa Välimäki

Ohjaaja: TkL Jyri Suvanen

Uusimmat sukupolvet matkapuhelinverkoissa mahdollistavat korkealaatuisen audiokoodauksen tiedonsiirrossa. Toisaalta audiosignaalinkäsittelyn siirtäminen sovelluskohtaisesta laitteistosta keskitettyjen palvelinkeskusten yleiskäyttöiseen laitteistoon on käynnissä, mikä aiheuttaa haasteita tarjota riittävästi laskennallista tehoa virtualisoituja verkkoelementtejä varten.

Tämä diplomityö arvioi modernin hybridikoodekin, Opuksen, käyttöä virtualisoidussa verkkoelementissä. Se on toteutettu integroimalla koodekki, testaamalla funktionaalisuutta ja suorituskykyä yleiskäyttöisellä prosessorilla sekä arvioimalla suorituskykyä verrattuna digitaalisen signaaliprosessorin suorituskykyyn. Funktionaalinen testaus osoitti että koodekki oli integroitu onnistuneesti ja että bittitason yhdenmukaisuus Opuksen standardin kanssa saavutettiin.

Suorituskyvyn testitulokset osoittivat, että vaikka enkoodaus tuotti vähemmän kellojaksoja digitaalisella signaaliprosessorilla, yleiskäyttöinen prosessori suoriutuu tehokkaammin suhteutettuna prosessorin kokonaiskapasiteettiin korkeamman kellotaajuuden ansiosta. Dekooderilla tämä näkyi vielä selkeämmin, sillä yleiskäyttöinen prosessori kulutti keskimäärin vähemmän kellojaksoja algoritmien suorittamiseen.

Avainsanat: puhekoodekit, transkoodaus, digitaalinen signaalinkäsittely, ohjelmistotestaus, digitaaliset signaaliprosessorit, mikroprosessorit

Preface

I would like to thank my instructor Jyri Suvanen for sharing the expertise and guiding me through the project. I would also like to thank my supervisor Vesa Välimäki for giving valuable advice during my work and for inspirational education during my studies.

In addition, I would like to thank Nokia Networks and Matti Lehtimäki for giving me the opportunity to work on this project. Furthermore, I want to show my gratitude to the whole "SPS" crew for providing the right answers when needed.

I want to thank my family for supporting me through my studies. Furthermore, I want to say a special thank you to my wife, Elżbieta, for being always there.

Espoo, 21.11.2014

Mika Sundvall

Contents

Abbreviations

A/D	Analog-to-Digital
ACR	Absolute Category Rating
AGW	Access Media Gateway
ALU	Arithmetic Logic Unit
AMR	Adaptive Multi Rate
ARM	Advanced RISC Machine
ATCA	Advanced Telecommunications Computing Architecture
ATCF	Access Transfer Control Function
ATGW	Access Transfer Gateway
AVX	Advanced Vector Extensions
BDTi	Berkeley Design Technology, Inc.
CBR	Constant Bitrate
CELT	Constrained Energy Linear Transform
D/A	Digital-to-Analog
DSP	Digital Signal Processor
DTX	Discontinuous Transmission
EPC	Evolved Packet Core
FB	Fullband
FEC	Forward Error Correlation
FIR	Finite Impulse Response
FMA	Fused Multiply-Add
GCC	Gnu Compiler Collection
GMAC	Giga Multiply-and-Accumulate Operations per Second
GPP	General Purpose Processor
GSM	Global System for Mobile Telecommunications
HF	High Frequency
HP	High-Pass
HR	Half Rate
IIR	Infinite Impulse Response
IMS	IP Multimedia Subsystem
IP	Internet Protocol
I/O	Input/Output
iLBC	Internet Low Bitrate Codec
IMDCT	Inverse Modified Discrete Cosine Transform
ITU	International Telecommunication Union
LSF	Line Spectral Frequency
LTE	Long Term Evolution
MAC	Multiply-And-Accumulate
MB	Medium-band
MGC	Media Gateway Controller
MGW	Media Gateway
MDCT	Modified Discrete Cosine Transform
MIMD	Multiple Instruction, Multiple Data

MIPS	Million Instructions Per Second
MMX	Multimedia Extension
MOS	Mean Opinion Square
MSC	Multiple Adaptive Spectral Audio Coding
MTU	Maximum Transmission Unit
NB	Narrowband
NSQ	Noise Shaping Quantization
LP	Linear Prediction
LPC	Linear Predictive Coding
LTP	Long-Term Prediction
OS	Operating System
PCM	Pulse Code Modulation
POLQA	Perceived Objective Listening Quality Assessment
QoS	Quality of Service
RCF	Request for Comments
RISC	Reduced Instruction Set Computer
RTP	Real-time Transport Protocol
SDP	Session Description Protocol
SIMD	Single Instruction, Multiple Data
SIP	Session Initiation Protocol
SNR	Signal-To-Noise Ratio
SWB	Super-wideband
TCP	Transmission Control Protocol
TI	Texas Instruments
TOC	Table-of-Contents
UDP	User Datagram Protocol
VAD	Voice Activity Detector
VBR	Variable Bitrate
VM	Virtual Machine
VMM	Virtual Machine Monitor
VoIP	Voice over Internet Protocol
WB	Wideband
WebRTC	Web Real-Time Communication
WOLA	Weighted Overlap-and-Add

1 Introduction

Mobile networks have evolved rapidly from circuit switched networks of the first generation to the latest deployments of all packet switched networks [1]. Although the main use case of transmitting speech has remained through the different generations, the new technologies and data processing techniques have resulted in increasing amount of mobile applications.

Audio coding is an integral part of mobile networks. Traditionally, speech coding has been developed in parallel to high quality audio coding, as the targeted usage of the latter one has been generally more limited in terms of transmission and storing capacity, as well as computational power [2]. However, as the mobile networks have evolved, more sophisticated coding methods can be utilized to maximize the transmitted information's quality. On the other hand, as the coding methods improve, less capacity is needed for high quality. Consequently, there has been an effort to include high quality audio coding in mobile networks.

Another increasing trend in the mobile networks include moving the data processing to centralized data centers, so that the operators do not need dedicated hardware for network element functions [3]. An essential part of this cloudification is virtualization of the network elements, which essentially moves the hardware assisted processing to software built processing with generic hardware.

In this thesis, usage of a modern audio codec called Opus [4] is evaluated when it is integrated on a virtualized mobile network real-time application. It is performed by conducting a case study where the codec's integration requirements are first specified, after which it is integrated to the application. Subsequently, it is tested for functionality, as well as performance. The study aims to evaluate, how a modern high-quality audio codec performs in a virtualized network element, and how well the performance supports the cloudification when the results are reflected to traditional hardware performance.

The thesis is divided into seven chapters. The first three provide the theoretical background to the topic. First, mobile networks and general audio coding theories are introduced in Chapter 2. Second, the Opus audio codec is reviewed in Chapter 3. Third, the used hardware platforms are introduced in Chapter 4. The theory is followed by three implementation related chapters. First, the realization of the codec is introduced in Chapter 5. Second, testing of the integrated codec is discussed in Chapter 6. Third, the results of these tests are presented and compared between the virtual and traditional hardware implementations in Chapter 7. The implementation is followed by discussion and conclusions (Chapter 8).

2 Background

Mobile networks are the part of communication networks that provide mobility to communications, i.e., the users do not need to be located at any fixed location in order to communicate [2]. As the communication typically includes speech signal transmission, there is a need to provide as light and agile representation of speech as possible. Consequently, different audio compression methods (also known as audio coding methods) have been developed to serve this purpose. When these methods are combined to a framework, an entity called codec is formed.

In this chapter, a broad overview of the mobile networks is introduced with a discussion of media processing in these networks. Moreover, general aspects of audio coding are presented with an emphasis on audio coding in mobile networks.

2.1 Mobile Networks

Communication networks include the services whose main purpose is to transmit information between users at different locations [2]. The part of communication networks that provides the ability to access the networks while not connected to the base or home network is called mobile networks. In the context of telephone networks, mobile networks are implemented as cellular telephone services [2]. They divide geographical areas into smaller cells with a radio transmission system which allow the users to communicate with each other without being physically connected via cables or wires [2]. The use of radio transmission systems introduces certain design related compromises which may result in lower quality of the transmitted signal, lower availability, and higher information security risks [2]. In the scope of this thesis, the first one of these drawbacks is of high interest.

Communication networks can be divided into different types depending on how information is organized for transmission, multiplexed, routed, and switched in a network [2]. This type of division yields three main types of networks. The first, message switched network, provides services to address, route, and forward messages [2]. The second one, circuit switched network, which is used e.g. in the traditional telephone networks, uses a dedicated end-to-end connection that is set up between the users [2]. Moreover, signaling network, which is built on top of circuit switched network, can be used to carry messages between the two ends. The third, packet switched network, divides the transmitted (digital) data into variable-length blocks, called packets, so that the data that is bigger than the packet size is segmented and transmitted in multiple packets [2].

By providing an end-to-end connection between the end-users, circuit switched networks allow an information flow with a very low delay [2]. However, there are certain characteristics that make them unfit for data transmission [2]. First, they perform optimally only with a steady information flow of voice signals, but do not support the transfer of many types of data. This is mainly due to the heavy overhead caused by setting up a connection [2]. This overhead introduces a delay that is too long for data transmission. Second, circuit switched networks need to maintain state information on all their connections and thus an extensive list of a potentially large number of connection is needed [2]. Finally, a failure in switching a node or transmission line requires a set up of a totally new connection and thus causes potentially a long break in the transmission [2].

Packet switched networks are very robust against failures that may occur on the data's path [2]. This is due to their flexibility to reroute the connection efficiently. In addition, packet switching does not involve the need of state information, because it utilizes so called connectionless type of switching, i.e., no end-to-end connection is needed because the packets are sent separately [2]. However, connectionless transmission includes a requirement that every transmitted packet must contain information about itself included in the packet's header [2]. This increases the overhead which reduces the transmission capacity. Furthermore, because the packet transfer is a unreliable transfer method, mechanisms for reliable packet ordering and handling is needed (such as Transmission Control Protocol (TCP)) [2].

The development of mobile networks has been divided into generations each having their own specification [1]. Simply abbreviated as 1G, 2G, 3G, and 4G, each generation has provided a new approach to the mobile networks reflecting the newest innovations in technology at the time of the specification's deployment. 1G was the first generation of the mobile network specifications and was purely an analog circuit switched network [1]. Therefore, there is no possibility to transmit data via 1G networks. 2G started to provide data transmission by allowing digital systems as overlays or parallel to the analog ones [1]. With 2G the typical data rates vary from 100 to 400 Kbit/s. 3G started to already include dedicated digital networks in parallel to the diminishing analog ones, allowing data rates from 0.5 to 5 Mbit/s [1].

4G is the latest generation that has been already deployed [5]. Unlike the previous generations, which were explicit specifications for the technology, 4G is rather a set of requirements [1]. Hence, any technology meeting these requirements can be categorized as 4G. An example of a 4G technology is Long Term Evolution (LTE) Advanced [1]. It is purely a digital and packet switched network and is based on IP (Internet Protocol). Therefore, the technological requirements are only designed for packet switching and data rates from 1 Mbit/s even up to 50 Mbit/s can be obtained [1].

As the development of these different generations shows, the mobile network evolution is heading from the circuit switched networks towards purely packet switched ones. This allows more flexibility to the applications, since all sorts of data can be transmitted via the packets and even internet applications can be developed as part of the mobile network systems.

2.1.1 Media Processing in Mobile Networks

Mobile networks divide the geological areas into cells, each of which include a radio access network (RAN) [1]. This is illustrated in Figure 1. When a connection is established, the user's mobile device connects to the RAN of the cell where they are currently located. This RAN is connected to a core network where data routing, accounting, and policy management is performed [1]. Subsequently, a connection to the destination network is established via an external carrier network, e.g., public Internet, [1] if the receiver is not within the range of the same core network [2]. The destination core network processes the connection and transmits it further to the end user via the RAN of the end user's location [1].

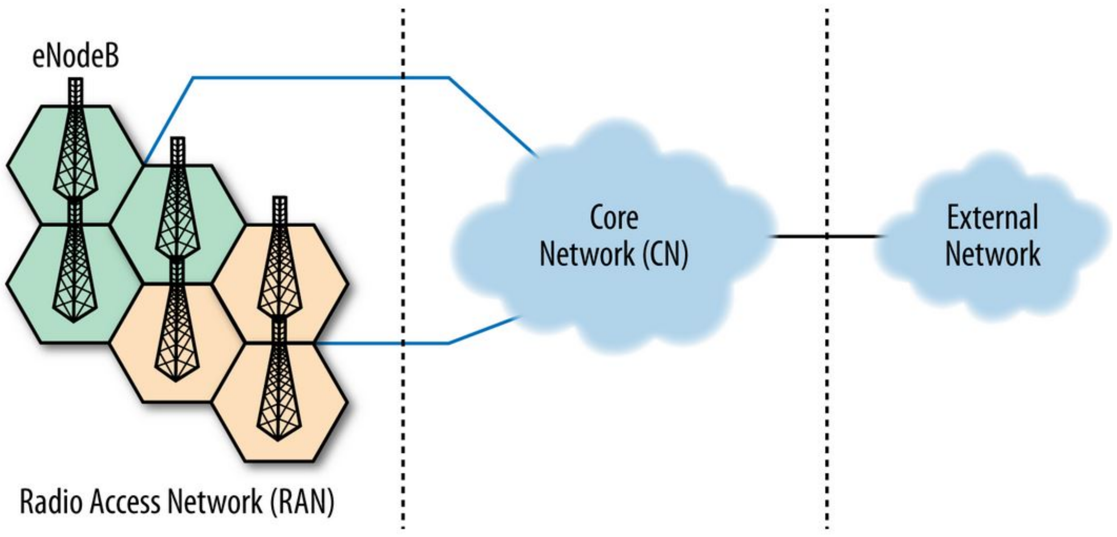


Figure 1: Schema of a mobile network, adopted from [1].

The packet switched core network (e.g., Evolved Packet Core (EPC) in LTE) includes a specification called IP multimedia subsystem (IMS), which provides certain functions and services for processing and streaming multimedia, such as audio or video signal [6]. It was specified by the 3GPP (The 3rd Generation Partnership Project), which is a union of different telecommunications standard organizations [7].

IMS includes all the core network elements that are related to multimedia services. Its main purpose is to provide the operators a possibility to offer multimedia services that are based on and built upon Internet applications, services, and protocols [6]. Hence, IMS includes the functions needed in providing applications for Voice over Internet Protocol (VoIP), Voice over LTE, and other IP based applications.

When there are more than one network involved, different types of interworking are needed, e.g., matching their media resources [8]. An essential instance where media resource matching is needed, is the case of differing codecs, i.e., when the source network has encoded the speech or audio stream with a codec that the destination network does not support [8]. In this case, the speech or audio stream must be first decoded using the original codec's decoder and then encoded with a codec that is mutual to both networks so that the destination network is able to process and decode the data. This, very essential part of media resource matching, is called transcoding. E.g., when two networks' voice bearers are communicating, the media matching is provided with a function called Media Gateway (MGW) [9]. Moreover, Access Transfer Gateway (ATGW) and Access Transfer Control Function (ATCF) functions, both implemented in an element called Access Media Gateway (AGW), need to provide transcoding in certain situations [6].

In a packet switched network, data is transmitted in packets with information of themselves [2]. There are different ways and methods to pack data into packets, which each have their use cases. However, it is essential that the sending and receiving end have the same set of rules to pack and unpack these packets. Therefore, different protocols have been developed to ensure that both sides are using the same methods to handle the packets. According to Leon-Garcia and Widjaja "a protocol is a set of rules that governs how two communicating parties are to interact" [2]. Next, two common protocols in real-time speech transmission, Real-time Transport Protocol (RTP), and Session Description Protocol (SDP) are introduced. They are used for transmitting real-time audio in mobile networks, as well as setting up a voice call.

2.1.2 Real-time Transport Protocol

Real-time transport protocol provides end-to-end transport functions for real-time data transmission including, e.g., audio or video [10]. These functions include payload type identification, sequence numbering, timestamping, and delivery monitoring. Furthermore, RTP is often run on top of Unified Datagram Protocol (UDP) so that both protocols contribute to the protocol functionality, the latter one providing multiplexing and checksum services [10].

Generally, RTP data can be divided into two parts. First, "a profile defines a set of payload type codes and their mapping to payload formats (e.g., media encodings)" [10]. Furthermore, profile may also include additional information about possible modifications to RTP that are specific to a certain application. Second, payload format defines how the actual data, e.g. audio or video encoding, is carried in RTP. [10]

A basic RTP packet consists of a 10-field header, and a payload [10]. These fields include information about the version used, payload type, sequence number, timestamp (which denotes the sampling instant of the first payload octect), synchronization source identification, and contributing source list.

2.1.3 Session Description Protocol

When a connection is established in a mobile network, certain initial information is needed to set up the connection with the right parameters. For this purpose, there are protocols to provide such information. Traditionally, Session Initiation Protocol (SIP) has been used in application-layer as a control protocol for creating, modifying, and terminating sessions [11]. However, sessions like VoIP calls, video streaming, and multimedia teleconferencing require media details, transport addresses, and other description metadata to the participants, which are not included in the SIP protocol [11]. A protocol called Session Description Protocol (SDP) provides a standard representation for such information.

SDP session description includes information about session and name purpose, time(s) the session is active, the media comprising the session, and information needed to receive those media [11]. The last one of these may include information about addresses, ports, formats, etc. Thus, information about the used codec's details may be passed as part of SDP.

2.2 Audio Coding

When transmitting voice over a communication network, the voice must be first transformed into an electrical form after which it can be processed and transmitted further. This electrical signal, which can be either a digital or an analog audio signal, represents the acoustic waves that we hear, only presented in a different domain [12].

The transform procedure begins from transforming the acoustic wave first from the acoustic domain to the electric one by using a transducer (e.g., a microphone), after which the obtained electric signal denotes the acoustic pressure wave captured at the point of the transducer [12]. The electric signal can be digitized so that it is sampled with a certain sample rate (given in Hz), i.e., the signal's value is measured every sample period [12]. Therefore, a higher sample rate produces more samples under a certain period of time compared to a lower one. The sampled values can be presented in the digital domain by quantizing to certain quantization levels that can be presented in binary form [12, p. 482]. This, the most basic form of acoustic wave's digital representation, is called pulse code modulation (PCM) [12].

For speech and audio signals the sampling frequency of the signal needs to be sufficient to contain enough information for the application's needs. Moreover, according to Nyquist's sampling theorem, the sample rate must be at least twice the highest frequency to be transmitted [13, p. 39]. As a result, the digital signal needs lots of capacity for storage or transmission. Therefore, audio compression plays a significant role when transmitting voice signals over mobile networks. This means that the audio signal is encoded (compressed), transmitted, and reproduced at the receiving end [13].

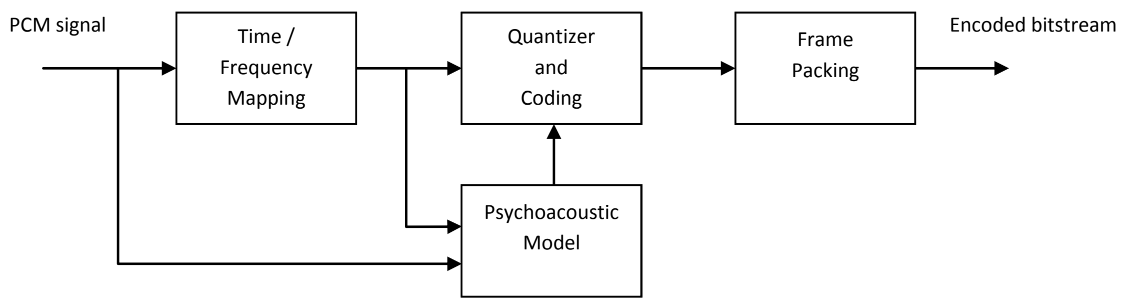
There are two types of audio coding, lossless and lossy coding (also known as perceptual coding). The former one is defined as "an algorithm that can compress the digital audio data without any loss in quality due to a perfect reconstruction of the original signal" [14]. However, there are limitations to the compression ratio, because all the information needed for perfect reconstruction must be transmitted. Lossy coding introduces some information loss, and thus does not provide a perfect reconstruction of the signal [15]. However, higher compression ratios can be obtained and therefore, perceptual coding methods are more extensively used in the mobile networks. The essential goal in coding audio in the mobile networks is to find a good ratio between the compression ratio and the loss of quality.

Audio coding combines physical models of the audio and voice signals, e.g., source-filter-modeling in linear predictive coding [13, p. 59], to traditional information coding methods. The physical representation of the signal, however, is only an approximation of the perceived voice. Furthermore, the transformation from acoustic to first electric and then to digital domain introduces distortion to the system, which cannot be necessarily modeled [12]. As a result, there is always an error between the physically modeled signal and the actual incoming signal. This has resulted in a method of transmitting the error signal, in its entirety or an approximation, along with the calculated model [16]. Moreover, better compression ratios can be obtained by lowering the resolution or the accuracy of the used physical formula, and thus lowering the correspondence of the modeled and original signals [13].

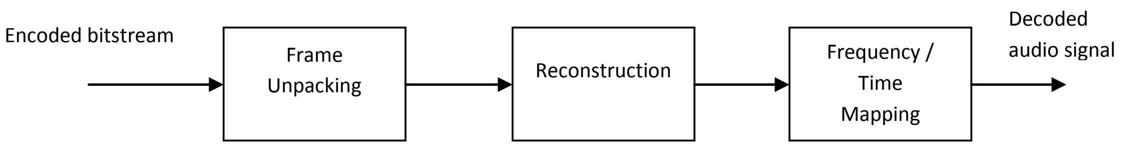
Generally, when coding information, a sequence of symbols is coded to codewords which can be presented with lower storage capacity than the original word [2, p. 753]. When the previously described physically modeled audio signal parameters are coded, as well as the potential error signal, it is possible to obtain even better compression ratios compared to purely coding the PCM signal with the same coding algorithms, because some of the irrelevant information can be discarded from the model [15].

A very common approach in audio coding is to form the modeled signal with time-frequency mapping, in which the incoming (digital) data is segmented into smaller blocks called frames after which the temporal and spectral components are analyzed on each frame [15]. In Figure 2, a general overview of an audio codec is illustrated with a block diagram.

Figure 2a presents an encoder of a generic codec that utilizes time/frequency mapping. The input PCM signal is first mapped in the *Time/Frequency Mapping* block, after which it is fed to the *Quantizer and Coding* block. Moreover, the input PCM signal and the output of the Time/Frequency mapping is fed to the *Psychoacoustic Model* block, which performs e.g., the psychoacoustic masking threshold, depending on the codec, are calculated [17]. Also this block's output is fed through the quantizer and coding block so that all the information can be packed into one encoded bitstream. The *Frame Packing* block performs this packing.



(a)



(b)

Figure 2: General block diagrams of a codec's encoder (a), and decoder (b), adapted from [17].

When the receiving end has received the encoded bitstream, it can be decoded back to an audio signal with a decoder illustrated in Figure 2b. First, the incoming bitstream is unpacked in the *Frame Unpacking* block. Second, the signal is reconstructed in the *Reconstruction* and *Frequency/Time Mapping* blocks using both, the time/frequency mapped data as well as the data obtained from the encoder's psychoacoustic model block [17]. The output of the decoder is an audio stream, which in lossless case corresponds exactly to the input of the encoder. Next, a very common audio coding method called Linear Predictive Coding is reviewed.

2.2.1 Linear Predictive Coding

A very classic speech coding method called Linear Predictive Coding (LPC) utilizes Linear Prediction (LP), which is an autoregressive modeling method [13, p. 43]. It is fundamentally based on an assumption that the signal under processing is formed as a response of an IIR filter (Infinite Impulse Response). One method of finding optimal coefficients for this filter is to use autocorrelation analysis as specified below

$$\begin{bmatrix} r(0) & r(1) & r(2) & \cdots & r(p-1) \\ r(1) & r(0) & r(1) & \cdots & r(p-2) \\ r(2) & r(1) & r(0) & \cdots & r(p-3) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ r(p-1) & r(p-2) & r(p-3) & \cdots & r(0) \end{bmatrix} \begin{bmatrix} a(1) \\ a(2) \\ a(3) \\ \vdots \\ a(p) \end{bmatrix} = \begin{bmatrix} r(1) \\ r(2) \\ r(3) \\ \vdots \\ r(p) \end{bmatrix}, \qquad (1)$$

in which *r(i)* denotes the autocorrelation coefficients calculated from the input signal, and *a(i)* denotes the filter coefficients [13, p. 43]. There are also other methods to obtain the optimal filter coefficients such as covariance method [18]. By inverse filtering the input signal with these coefficients, a residual signal can be obtained.

Speech production of a human can be presented as a simple source-filter model. The vocal tract can be approximated with a filter, through which the glottis excitation signal flows [13, p. 59]. Consequently, when denoting the vocal tract as the filter calculated in LP, LPC can be utilized to reproduce the speech signal. The excitation signal of the source-filter -model is the residual signal obtained in the LP analysis [13, p. 43].

In order to reproduce a speech signal intelligibly, all the fine harmonic structure of the vocal tract's spectrum does not need be reproduced [13, p. 44]. This can improve the computational efficiency significantly, as well as the compression ratio. The order needed for intelligible speech transmission with LPC can be reduced to 10-12 [13, p. 44].

LPC has played a very important role in audio coding from its invention. It is the core technology of several lossless codecs such as MPEG4-ALS, FLAC, and WavPack [14], and lossy codecs such as AMR (Adaptive Multi Rate) [19], GSM-HR (Half Rate) [20], and GSM-FR (Full Rate) [21]. With lossless codecs LPC can provide compression ratios up to 70 % [14], and with lossy codecs even greater.

2.2.2 Transform Coding

In transform coding, the time-frequency mapping is performed by using a time-to-frequency-domain transform. Generally, this approach divides the signal into spectral components [17]. Depending on the algorithm used, the resolution of this division as well as the computational complexity varies. The most common transforms in this coding method are the Fourier transform and the cosine transform, as well as their derivatives [17]. When a signal is represented in the frequency domain, it is convenient to evaluate if there are frequencies that are not needed in the particular application and thus can be omitted. Even though some information is lost, the compression ratio is increased. When the transmission capacity is limited this is a more desired result than maintaining all the information, also the irrelevant, at the expense of compression ratio.

With transform coding, it is possible to obtain a very high-resolution frequency domain estimate of the signal [15]. However, this is at the expense of the temporal characteristics. A more sophisticated method to obtain the frequency domain characteristics of the signal is to divide the signal into frequency bands (also known as subbands). This, so called subband coding or multiple adaptive spectral audio coding (MSC), is often designed to match psychoacoustic characters of a human auditory system and is also calculated in the psychoacoustic model block of the encoder (Figure 2a) [15]. A common subband division divides the spectral information with a resolution of critical bands which yields the Bark scale [13]. It is closely related to human hearing mechanism, i.e., a constant change in a Bark scale corresponds to a constant change in the basilar membrane's resonance position [13, p. 111].

2.2.3 Entropy Coding

When the physical models of a signal are calculated, the obtained coefficients and other parameters are coded using traditional information coding algorithms and methods. Typically, in information coding, a block of data is seen as a sequence of symbols [2, p. 753]. The main goal is to map these symbols into binary representation so that the average number of bits is as low as possible. Furthermore, in the case of lossless coding, the original symbols must be able to be recovered from the encoded bitstream [2, p. 753]. The best performance in coding, in terms of how many bits were needed to represent the original symbol, can be obtained by using an entropy function [2, p. 757]. A coding method called entropy coding utilizes this function, which depends on the probabilities of the sequences of the symbols. If a sequence of symbols is given as $1, 2, 3, ..., K$ and the respective probabilities to their occurrence as $P[1], P[2], P[3], ..., P[K]$, and these probabilities are statistically independent, the entropy function is defined as follows

$$\begin{aligned} H &= -\sum_{k=1}^{K} P[k]\log_2(P[k]) \\ &= -P[1]\log_2(P[1]) - P[2]\log_2(P[2]) - ... - P[K]\log_2(P[K]), \end{aligned} \tag{2}$$

in which H stands for the entropy [2, p. 757].

There are two approaches in entropy coding depending on how the coded information is arranged into codewords. One of them utilizes Huffman coding and the other arithmetic coding [22]. The former one of these is a non-adaptive method, i.e., the values can be read from a table and no other values need to be coded in order to obtain a certain value at an arbitrary index of the table. However, Huffman coding is rather limited in the amount of compression it can provide [23]. In addition, it can only provide an integer number of bits and thus in the case where the given bits for coding is fractional, there is a redundancy of difference between the rounded up number of bits and the fractional value of the bits [22].

Arithmetic coding is based on a presumption that the data is organized in vectors [22]. As a result, in order to be able to decode one symbol from the vector, all the previous symbols must be decoded first. This results in a highly serial operation and increases computational complexity. However, with arithmetic coding, it is possible to obtain fractional allocation of bits per sample and thus no redundancy due to rounding is introduced [22].

2.3 Audio Coding in Mobile Networks

When the whole auditory bandwidth is included in the coding, such as in MP3 and Vorbis codecs, high qualities can be obtained [24]. However, they typically introduce a long algorithmic delay and thus cannot be used in real time applications. The mobile networks have always been limited by the transmission capacity [25]. Therefore, the audio coding technologies in mobile networks have traditionally limited the transmitted information, e.g., in terms of bandwidth, to gain better compression rations. As a result, these technologies introduce a low algorithmic delay with high compression ratios. The bandwidth needed for intelligible speech transmission can be limited to 4 kHz, which yields a commonly used sample rate in speech applications of 8 kHz [2, 26]. However, through the evolution higher bandwidths, such as 8 kHz (with 16 kHz sample rate) [27], have been also introduced in this context.

The speech coding technologies have been developed by different groups and organizations to provide codecs that utilize the constantly developing computation, storing, and transmission technologies. Organizations such as 3GPP, ITU (International Telecommunication Union), and GSM (Global System for Mobile Communications) have developed speech codecs solely for mobile networks. These codecs include for instance AMR by 3GPP [19], GSM-HR and GSM-FR by GSM [20, 21], and G-series codecs by ITU [28, 29].

In parallel, as the VoIP technology has become more common, internet speech codecs have been developed by organizations such as Xiph.Org Foundation [30], Skype Technologies [31], and Global IP Solutions (currently owned by Google Inc.) [32]. The first one of these has developed internet codecs such as Speex [33], CELT (Constrained Energy Lapped Transform) [34], and Opus [4]. The second one has developed the SILK codec mainly targeted for the web call application Skype [35]. The last one of these is the developer of iLBC (Internet Low Bitrate Codec), which is also a VoIP codec [36].

As was discussed earlier, the transmission capacity of the mobile networks has improved significantly through the evolution of the newer mobile network generations [1]. Furthermore, the newest audio coding technologies, such as CELT [4, 24], have enabled the use high-quality full auditory bandwidth coding in real-time communication applications by providing low enough algorithmic delay. As a result, the distinction between the two different coding approaches is diminishing. On the other hand, as the mobile networks are increasingly becoming all packet switched networks, the VoIP applications are also included in mobile communications in addition to the native audio transmission. Hence, also the codec development branches, for internet and mobile applications separately, are gradually converging.

The distinction that remains between the VoIP and native voice communication within the mobile networks is the way the data is transmitted, i.e., VoIP utilizes the IP as is, whereas the native audio transmission in mobile networks is based on the protocols specified for mobile networks [37]. Furthermore, VoIP does not include quality of service (QoS), e.g., packet loss handling, delay handling, and delay variations handling, which introduces a need of additional processing when transmitting VoIP in mobile networks [38]. These distinctions do not affect on the audio coding algorithms as the encoded bitstream is transmitted within the payload regardless of the means of transmission.

The latest mobile audio coding development efforts incorporate the full bandwidth coding and low algorithmic delay needed in the mobile applications [24, 4]. The state-of-art real-time codec is currently a hybrid VoIP codec called Opus, providing versatility and a wide set of parameter settings. In the following chapter, Opus codec is introduced with an in-depth review on the technology behind the codec.

3 Opus Audio Codec

In this chapter a modern hybrid codec called Opus is reviewed. First, an overview of the codec is introduced with a high-level description of the features provided. Second, the different components, of which Opus consists of, are discussed more in detail. Finally, at the end of this chapter computational complexity is evaluated, and possible applications of the Opus are reviewed.

The Opus codec combines two existing coding technologies, SILK and CELT, and was originally developed to fulfil a wide range of requirements from low bitrate telephony to stereo full bandwidth real-time teleconferencing [39]. To be able to provide such a wide range of features, the codec needs to be very versatile and configurable. This yields a large set of parameters that can be configured depending on the application used. The available parameters in the Opus codec are presented in Table 2.

Table 2: List of available configuration parameters of Opus audio codec, adapted from [4].

Parameter	**Options**	**Description**
Sample rate	8 kHz ... 48 kHz	Sample rate selection
Bitrate	6 kbit/s ... 510 kbit/s	Bitrate selection
Number Of Channels	Mono/Stereo	Mono or Stereo selections
Frame Duration	2.5 ms ... 60 ms	Frame duration selection
Complexity	0 ... 10	Complexity selection (affects mainly on computational complexity, e.g. order of filters)
Constant/Variable Bitrate	Constant/Variable	Selection of constant or variable bitrate
Packet Loss Resilience	On/Off	Inter-frame dependency selection (for better robustness against packet loss)
Forward Error Correction	On/Off	Another method to provide better robustness against packet loss
Discontinuous Transmission	On/Off	Reduces the bitrate during silence or background noise

The first parameter in the table is sample rate. Opus codec provides a selection of sample rates from 8 kHz to 48 kHz, with several sample rates in between. In Table 3 these sample rates are presented, and their corresponding audio bandwidths are introduced, as they are used later in this thesis. Super-wideband used in the

Opus is defined as 24 kHz, even though the same name is used in other audio coding standards for 32 kHz sample rate. This is mainly because it is more convenient for Opus' internal processing [4].

Table 3: Supported samplerates in the Opus codec, adapted from [4].

Sample rate	**Name**	**Abbreviation**
8 kHz	Narrowband	NB
12 kHz	Medium-band	MB
16 kHz	Wideband	WB
24 kHz	Super-wideband	SWB
48 kHz	Fullband	FB

The second parameter, bitrate, denotes how many bits are used to represent one second of data in the coded domain [40]. Thus, the storage and transmission capacity is directly proportional to it. The Opus codec supports any bitrate in the range of 6 kbit/s to 510 kbit/s. Furthermore, there are recommendations of the optimal bitrates depending on the used sample rate [4].

Frame duration selection in the Opus offers several alternatives for the frame length. The available frame lengths are 2.5 ms, 5 ms, 10 ms, 20 ms, 40 ms, and 60 ms [4]. Complexity selection affects the computational complexity of certain parts of the coding. E.g., order of certain filters can be reduced with this parameter [4]. However, lower complexity results in lower quality because the complexity is reduced by lowering the accuracy.

Constant/variable bitrate selection provides an option to choose between constant bitrate (CBR), and variable bitrate (VBR). The former one of these uses the same bitrate throughout the audio data, whereas the latter one varies the bitrate to find the optimal value for the data processed at all times [4]. VBR also results in better coding efficiency [40].

Audio codecs are often designed to use the inter-frame correlations in order to reduce the bitrate [4]. Therefore, when a packet is lost, the decoder needs to receive several packets before it is able to effectively reconstruct the signal. Forward Error Correction (FEC) can be used to improve the robustness against packet loss caused by this problem. In addition, Packet Loss Concealment (PLC), can be utilized using the algorithms provided within the codec for improved robustness against packet loss. It attenuates the signal slowly based on the previous frames, and additionally generates conform noise [41]. Consequently, a sudden gap in the audio stream is avoided when packet is occasionally lost.

Discontinuous Transmisson (DTX) is part of the SILK's functionality and it reduces the transmission bitrate when the signal is silent or there is only background noise present [41]. In fact, when DTX is used and silence is detected, only one frame every 400 ms is transmitted. The packet loss concealment functionality is invoked when no data is transmitted due to DTX, so that the indicated silence does not result in total silence at the receiving side [41].

In addition to the previously introduced parameters, there are three different application selections that result in different coding paths within the codec [42]. First, VoIP application aims at best quality for speech signals by emphasizing formants and harmonics. Second, Audio selection results in best quality for non-speech signals such as music or combined speech and music signals. Third, Restricted Low-Delay mode provides slightly reduced algorithmic delay, but also disables the speech enhancements.

Being a hybrid codec, Opus provides three different modes of coding. Purely either CELT or SILK codec, or a hybrid mode which combines these two [39]. The usage of these layers is automatically toggled depending on the parameter set. Next, the SILK codec is reviewed more in detailed level. It is followed by a detailed description of the CELT codec, after which the way these two codecs are combined into the Opus is discussed.

3.1 SILK

SILK codec is a speech codec developed by Skype [35] and is based on traditional speech coding techniques like LPC and LTP (Long-Term Prediction, also derived from LP) [43]. Similarly to Opus codec's development, SILK was developed to meet the needs of VoIP applications. With a support for four different samplerates 8 kHz, 12 kHz, 16 kHz, and 24 kHz [44], it is a highly scalable codec. Furthermore, changeable bitrate and complexity parameters provide flexibility to meet the used application the best [35]. Also features such as DTX and VBR are supported [44].

The two linear prediction based coding methods (LPC and LTP) used in SILK are divided so that LPC is used every 20 ms to model the vocal tract transfer function, and LTP every 5 ms to model the long-term correlation of pitch harmonics in the voiced speech signal [43]. The block diagram of a SILK encoder is illustrated in Figure 3.

As the block diagram shows, encoding begins with feeding the input signal through a high-pass filter (HP), as well as voice activity detector (VAD). The former is a 2nd order adaptive IIR filter with a cutoff frequency between f_c = 60 Hz ... 100 Hz, and its main function is to filter the low-frequency background and breathing noises [45]. VAD measures the speech activity level by combining the SNRs (signal-to-noise ratios) of four separate frequency bands [45]. The HP filter is dependent on the detected voice activity level so that the SNR of the lowest frequency band of the VAD, and the smoothed pitch frequencies found in the pitch analysis, affect on its cutoff frequency. As a result, high pitched voices have a higher cutoff frequency.

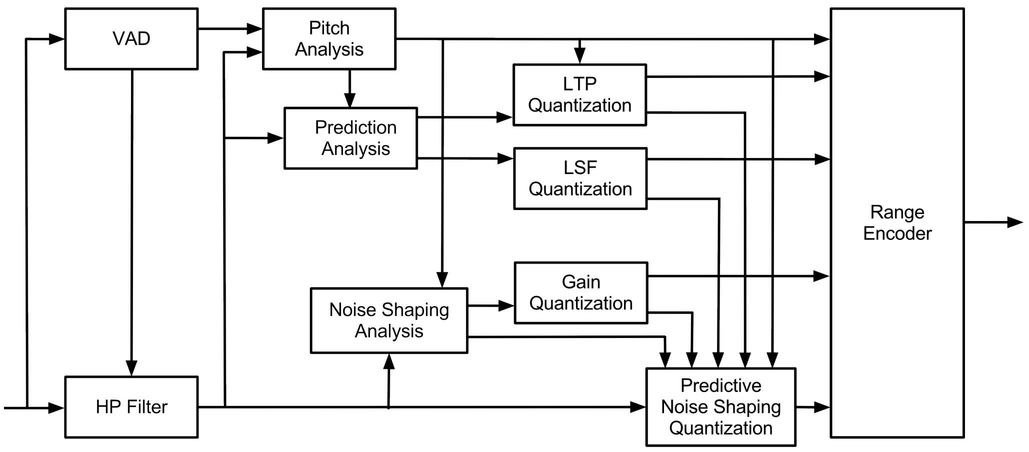


Figure 3: Block diagram of a SILK encoder, adopted from [45].

After determining the voice activity level and filtering the input signal, the pitch of the voice is analyzed in the *Pitch Analysis* block. It is performed to a pre-whitened signal (pre-whitening is performed within the block) which is decimated into two different signals with 8 kHz and 4 kHz sample rates [45]. The fundamental frequency is found through autocorrelation analysis so that it is performed first to the signal with 4 kHz sample rate for less accurate estimation, and then to the signal with 8 kHz signal for more precise estimation [45]. The final pitch is determined by analyzing the original, not decimated signal with an integer valued pitch lag search around the previously obtained estimate. Subsequently, each lag is evaluated with a pitch contour from a codebook [45]. The found pitch is fed directly to the range encoder, and encoded as part of the bitstream.

In addition to pitch calculation, also the signal classification for the frame under analysis is determined in the pitch analysis block. The result of the autocorrelation analysis is compared to a certain threshold, so that if the autocorrelation result is lower than the threshold, the signal is classified as unvoiced speech [45]. The threshold is calculated from a weighted combination of the signal classification of the previous frame, speech activity level, and the slope of the SNR obtained in the VAD with the corresponding frequency [45].

The actual linear prediction is performed in the Prediction Analysis Block with the previously analyzed signal classification taken into account. For voiced speech, LTP is used with the pre-whitened input signal incoming from the pitch analysis block [45]. The LTP-coefficients are estimated with covariance method [18] from every 5 ms subframe. The residual signal is determined by filtering the non-whitened input signal with the LTP-filter.

The short-term prediction coefficients are calculated from the LTP-analysis' residual signal with the traditional LPC-analysis [45]. However, instead of determining the LP-coefficients with autocorrelation or covariance method, they are computed by using the novel implementation of Burg's method [46]. Therefore, stable filter coefficients are obtained efficiently and the computational complexity is reduced remarkably [45, 46]. Calculating the LPC-coefficients from the LTP-signal results in more prediction gain and consequently, lower bitrate can be used. For unvoiced signal, LPC is performed directly to the non-whitened input signal.

All the LPC-coefficients are transformed into Line Spectral Frequencies (LSFs), which provide an alternate and more efficient representation of these coefficients for speech coding and transmission [47]. Subsequently, these frequencies are quantized, and used to re-calculate the LPC residual signal so that also the quantization error of LSFs is taken into account when reproducing the signal [45].

SILK uses noise shaping to make the quantization noise less audible to human auditory system by performing weighting filtering when encoding [45]. Therefore, no weighting filtering is needed when decoding and thus, no side information about the filter needs to be sent along with the encoded bitstream. The noise shaping is performed in the encoder's noise shaping analysis, gain quantization, and predictive noise shaping quantization blocks (in Figure 3).

Predictive Noise Shaping Quantization (NSQ) block also generates the residual signal from the previously determined LTP- and LPC-coefficients [45]. This excitation signal is subsequently quantized into an integer-valued signal, which is entropy coded in blocks of 16 samples in the range encoder. The remaining LSF Quantization block quantizes the line spectral frequencies with vector quantization using codebook vectors and scalar inter-LSF prediction [45]. The quantized signal is entropy coded with the range coder which outputs data with variable bitrate [45].

The encoded bitstream can be decoded back to an audio signal with a decoder. The decoder side LP functionality is illustrated in Figure 4. It performs the inverse operation to the encoder's essential blocks and consists of excitation generator and the predictive filtering of the LTP and LPC synthesis filters. The first one of these generates the excitation signal for the LTP synthesis filter from the quantization indices transmitted from the encoder. The LTP synthesis is performed as was described in Section 2.2 for linear predictive coding. The LPC synthesis filter is in series with the LTP and thus the output of the LTP denotes the LPC excitation signal.

3.2 Constrained Energy Lapped Transform

Constrained Energy Lapped Transform is a transform coding technology developed by Xiph.Org, and is based on the Modified Discrete Cosine Transform (MDCT) [48]. The fundamental idea behind the codec is to preserve the spectral envelope of

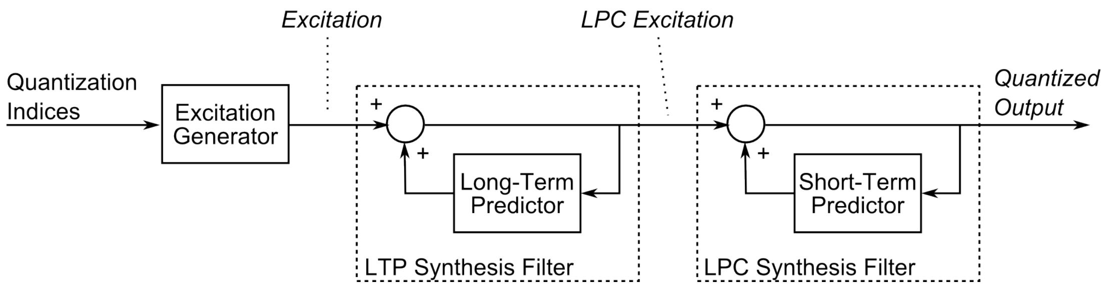


Figure 4: Block diagram of a SILK decoder's LP functionality, adopted from [45].

the input [24]. Moreover, CELT was designed to support also higher sample rate applications up to 48 kHz, and bitrates from 32 kbit/s to 128 kbit/s [49]. One of the key features in CELT is to use the transform with very short windows so that the algorithmic delay is minimized [48]. The development with CELT started from same principles as with Opus codec, to have a framework that is highly scalable and versatile. Subsequently, the development of CELT was merged to Opus [49] and hence the technology behind is being developed with the Opus instead of a separate codec.

There are four principle elements behind the CELT algorithm [24]. First, the MDCT output is divided into bands, referred as energy bands, that approximately correspond to the Bark scale. This improves the perceived quality as the band division corresponds to the resolution of the auditory system. The comparison between the CELT energy band division and bark-scale is illustrated in Figure 5. As can be seen in the figure, at the higher frequencies the codec's energy bands correlate highly with the Bark scale. However, at the spectrum's lower end this correspondence is not as high due to MDCT's lower resolution at the low frequencies [24].

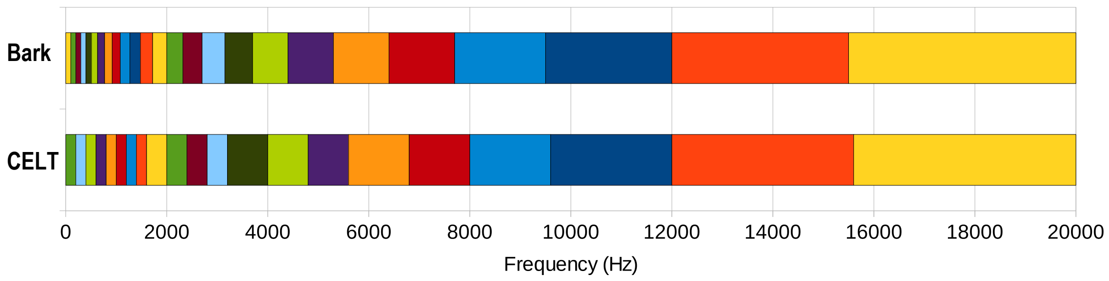


Figure 5: Comparison of CELT energy bands and Bark scale, adapted from [48].

Second, the encoder codes each band's energy separately with the decoder ensuring that the output's energy corresponds these energies [24]. Third, the normalized spectrum of each band is constrained so that they have unit norm through the whole algorithm [24]. Fourth, a long-term predictor is used for coding pitches and is encoded as a time offset, yet encoding the gain of the pitch in the frequency domain [24].

Figure 6 represents the block diagram of the CELT codec, with both encoder and decoder presented. The first block, *Window*, performs windowing to each frame [48] using flat-top MDCT windows with 2.5 ms overlap between the frames. For the overlapping parts, Vorbis power-complementary window (as is specified in [50]) is used, which is also used in the decoder side when performing the weighted overlap-and-add (WOLA) synthesis. The MDCT block performs the modified discrete cosine transform so that the computed MDCT spectrum is divided into the energy bands as was presented in Figure 5 [24]. Each band is normalized separately and transmitted to the decoder through a quantization block, Q_1.

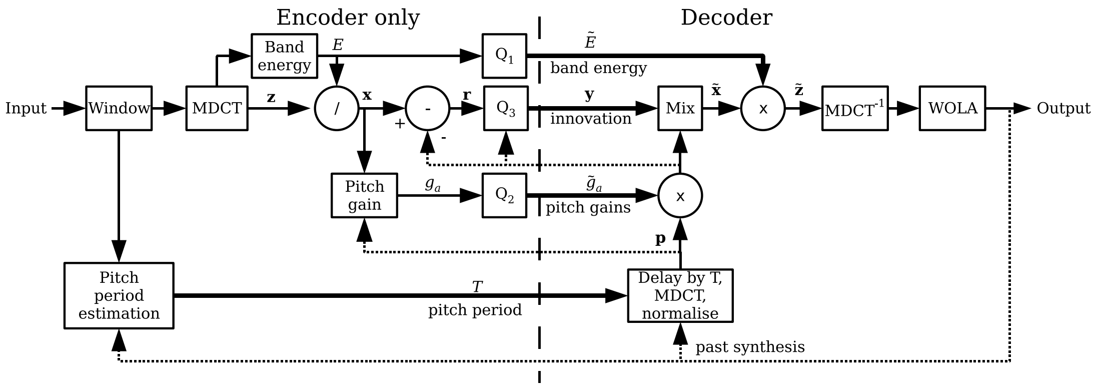


Figure 6: Block diagram of a CELT codec, adopted from [24].

Pitch prediction is used to model high-frequency information such as closely-spaced harmonics of speech or solo instruments [24]. The determined pitch's gain is computed and normalized to unity. As a result, the pitch can be applied in the frequency domain to avoid the weakening of the pitch harmonics at high frequencies. In addition, fixed and adaptive codebooks are combined in the encoder's quantization blocks where entropy coding is applied [24]. However, the fixed codebook gains do not need to be transmitted because the computation can be deduced to a rather simple form and it can be calculated when decoding. The codebook contribution is regarded as *innovation* in Figure 6.

There are altogether three different quantizers in the CELT encoder (Q_1, Q_2, and Q_3 in Figure 6). The first, Q_1, quantizes the band energies, the second, Q_2, the pitch gains, and the third, Q_3, the innovation. For all quantizers, entropy coding is used to allocate the fractional number of bits to integers. For bit allocation, two of the transmitted parameters are encoded with a variable bitrate [24]: the entropy-coded energy of each band and the pitch period. However, when constant bitrate is needed the innovation quantization can be adapted to compensate the variability of the previous two parameter sets [24]. Additionally, to minimize the amount of used bits, it is assumed that certain parameters are known for both encoder and decoder, and do not need to be transmitted [24]. E.g., the number of octets (8-bit bytes) used to encode the frame is assumed to be shared information.

Occasionally, transform-based codecs introduce pre-echo to the signal due to the quantization error that is spread through the window, including the transient events [51]. Generally, this does not affect CELT because the frame sizes are small. However, in certain extreme cases it may occur, and therefore CELT uses a detector to detect the transients [51]. In the case a of transient, the frame is split into two shorter half-frames, to which the MDCTs are applied. The outputs are then interleaved yielding a situation where the rest of the codec is not affected.

3.3 Opus Codec

The Opus codec combines the two previously presented coding technologies, SILK and CELT, into a single codec. It can be used either in a hybrid mode where it uses both of them, or only with either of the two codecs [4]. However, in addition to simply combining these two codecs into one framework, certain changes have been made in order to improve the quality, to fit them better together, and to reduce the psychoacoustic artifacts of the two codecs [48].

3.3.1 Combination of the Two Coding Technologies

As the Opus combines the SILK and CELT technologies, they are separated to serve different purposes. When using either of the original codec modes alone, SILK can be operated with up to 16 kHz samplerate, and CELT up to 48 kHz [4]. Thus, according to the naming convention introduced in Table 3, SILK supports NB, MB, and WB, whereas CELT supports all the sample rates from NB to FB. For the frame lengths, SILK supports 10 ms ... 60 ms, and CELT supports 2.5 ms ... 20 ms. As a result, SWB and FB data cannot be coded with higher frame lengths than 20 ms and therefore data containing longer frames is split into more appropriate frame lengths for the CELT.

In the hybrid mode of the Opus codec, the operation is divided so that the crossover sample rate is 16 kHz [4]. In other words, the input signal is divided into the two coding paths, in which it is decimated to 16 kHz sample rate for SILK, and for CELT the frequencies below WB are discarded. With this configuration, Opus is able to provide hybrid mode operation at SWB and FB. Due to the different look-ahead times between these two codecs (CELT having shorter) a delay of 4 ms is introduced to the CELT coding path. However, when using only CELT the additional delay is omitted. An overview block diagram of the Opus codec is illustrated in Figure 7.

As can be seen in Figure 7, the coding is divided into two branches. In the encoder, the higher branch represents the input of the CELT-encoder, and is delayed in the D block to match the different look-ahead times. The lower one represents the input of the SILK-signal, which is decimated so that only lower frequencies are encoded with the SILK layer. After encoding them in their respective encoders, the two

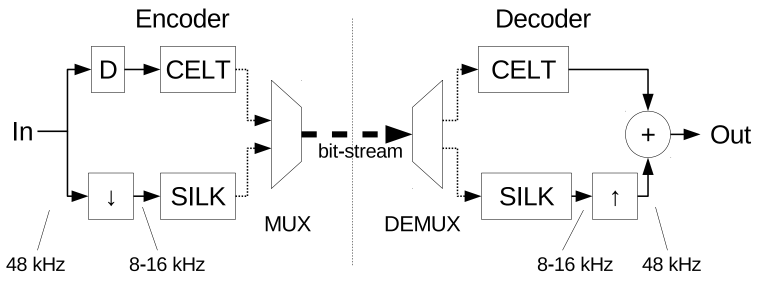


Figure 7: Block diagram of Opus codec, adopted from [48].

encoded signals are multiplexed, i.e., multiple signals combined into one signal or bitstream in the digital domain [2], and transmitted. In the decoder, the bitstream is demultiplexed and the two separate, CELT and SILK, signals are then decoded in their respective decoders. A sample rate conversion is performed to the output of the SILK-decoder to match the desired output sample rate. Finally, the two coding branches are summed together so that the output of the Opus decoder is obtained.

Opus uses also additional internal framing to allow the packing of multiple frames into a single packet [48]. Moreover, the mode, frame size, audio bandwidth, and channel count are signaled in-band, and are encoded in a table-of-contents (TOC) byte. These are not entropy coded, an therefore it is easy for the external applications to "access the configuration, split packets into individual frames, and recombine them" [48].

When changing the configuration so that the CELT mode is used before and after the change, the overlap of the transform window is used to avoid discontinuities [48]. However, when a change between CELT and SILK or the hybrid mode occurs, there is a mismatch between the domains where the computation is performed, the former one utilizing frequency- and the latter one time-domain algorithms. Therefore, in this type of change in modes, an additional 5 ms redundant CELT frame is included to the bitstream. Furthermore, in this case decoder uses overlap-add to bridge and gap between the discontinuous data, which smooths the changes between the different modes [48].

3.3.2 Modifications to the Two Coding Technologies

The SILK codec has been an individual project and thus is used in the Opus codec almost as is. The originally supported sample rate of 24 kHz was not included when SILK was integrated into the Opus codec [4]. CELT, however, was merged to the Opus project [49] and therefore modifications to the originally proposed version [24, 51] have emerged through the development of Opus.

CELT introduces a drawback when using the low overlap window by increasing spectral leakage which is especially problematic for highly tonal signals [48]. Therefore, in the Opus a pre-emphasis first-order low-pass FIR filter (Finite Impulse Response) is introduced in the encoder side. Respectively, the corresponding inverse operation with a de-emphasis filter is applied when decoding. As a result, lower frequencies are attenuated, and therefore do not cause leakage at the higher frequencies. In addition, a perceptual pitch enhancing prefilter and postfilter pair is used to address the same problem [48].

Depending on the signal, some frames may contain both transients and tonal information. Consequently, these frames require good frequency resolution in addition to good time resolution [48]. Therefore, the Opus utilizes a selective modification of the time-frequency resolution so that it includes a good resolution for low-frequency tonality, but also a good time resolution for transients' high frequencies. The time-frequency resolution is improved by computing multiple short MDCTs [48]. This, however, decreases the frequency resolution, and therefore the Hadamard transform is used to the transform coefficients across the computed multiple short MDCTs [48].

With transform based codecs, such as CELT, tonal-noise may be introduced [48]. This is due to the quantization in which a large number of HF coefficients get rounded to zero, and thus the remaining non-zero coefficients may sound tonal. To address this problem Opus applies spreading rotations that spread the transform coefficients dynamically over multiple frequencies [48, 52]. When decoding, an inverse operation can be applied to obtain the original transform coefficients [48].

When transients are encoded with low bitrates, it may happen that all the coefficients of a short MDCT are quantized to zero [48]. This causes audible drop outs, even when the energy of the entire band would be preserved. The Opus codec detects such holes and fills them with pseudo-random noise at the same level as the minimum of the two previous bands' energy levels [48].

Entropy coding is used extensively in the SILK- and CELT-codec, and consequently also in the Opus codec [45, 48]. In the context of Opus, all the entropy coding is performed with a range coder which is an arithmetic coder and which outputs 8 bits at a time [24]. The main reason for using arithmetic coding is its ability to accept fractional bits and provide better compression ratios [22].

3.4 Computational Complexity

Both SILK and CELT were designed for real-time applications and thus need be computationally simple enough to reduce the algorithmic delay to sufficient limits [31, 34]. The former one of these uses the time-domain techniques for coding (LPC and LTP) and its original target was speech coding. At the frequency spectrum's low end, where most of the essential speech information is, these techniques are computationally efficient [4]. At the high end of the spectrum, however, the transform based techniques are more efficient and thus combining these two into one hybrid codec is an effort to maintain the maximum efficiency [4]. Furthermore, when combining the most efficient parts of the two codecs, the combination may operate even more efficiently than the original codecs individually.

Generally, transform and subband coding techniques, such as MDCT in CELT, are limited by their coding efficiency [53], because the transform itself is computationally demanding operation [54]. MDCT transform is defined as follows

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot \cos \frac{(2n + 1 + \frac{N}{2})(2k + 1)\pi}{2N}, \tag{3}$$

where $X[k]$ is the transformed signal, $x[n]$ is the input signal, $n = 0, ..., N - 1$ and $k = 0, ..., N/2 - 1$, N is the window length [54]. As Equation 3 shows, there is a sum of products between the input sample and a cosine term over the entire window. Therefore, every time MDCT is computed the whole window needs to be iterated through. As a result, MDCT introduces always an algorithmic delay that corresponds to the number of iterations [55]. The inverse MDCT (IMDCT) is applied to the half of the window length and thus the delay caused by IMDCT is half of the corresponding MDCT [54]. IMDCT is defined as

$$x[n] = \sum_{k=0}^{N/2-1} X[k] \cdot \cos \frac{(2n + 1 + \frac{N}{2})(2k + 1)\pi}{2N}, \tag{4}$$

in which $x[n]$ is the reconstructed signal, $X[k]$ is the MDCT transformed signal, $k = 0...N/2 - 1$, and $n = 0, ..., N - 1$ [54]. Similarly to the MDCT, also IMDCT introduces an algorithmic delay, as it includes iteration through the coefficients.

The algorithmic delay caused by the iterations, shown in Equations 3 and 4, can be only reduced by minimizing the number of iterations. This number depends on the frame size, as the processing is done for one frame at the time. To address this issue, CELT uses a very short frame length to minimize the delay [51].

The range coder that performs the entropy coding is an arithmetic coder, which minimizes the used bits for coding. This may introduce computational complexity to the decoder side, as the coded vector needs to be decoded and iterated through $N - 1$ times, if a specific value with an index N needs to be decoded. However, as could be seen in the SILK and CELT codecs' block diagrams (Figures 4 and 6), the arithmetic decoder is applied only in the beginning of the decoding procedure. Therefore, when the actual audio decoding is performed, all the data is already accessible and no range decoding is needed to obtain single symbols from the bitstream.

Opus' computational complexity can be modified with the complexity parameter, as introduced in Table 2. It adjusts the order of certain filters, the number of states in the residual signal quantization, and the use of certain bitstream features including variable time-frequency resolution and the pitch post-filter [4]. The first one of these, filtering (all the IIR- and FIR-filters), is performed by convolving the input signal with the impulse response of the filter [56]. Convolution integral in discrete domain is defined as a sum of products of the filter coefficients, and current and old samples. This is shown below in Equation 5

$$y[n] = x[n] * h[n] = \sum_{k=0}^{n} x[n] \cdot h[n-k], \tag{5}$$

in which $y[n]$ is the convolution result, $x[n]$ is the input signal, and $h[n]$ is the filter's impulse response [56]. As can be seen in the equation, also filtering is an iterative process. The number of these iterations is directly proportional to the filter's order. Therefore, adjusting the order of the filter can result in lower complexity of the algorithm.

When the complexity of the algorithm is reduced, the quality is also reduced [4]. Figure 8 illustrates how the complexity parameter affects the voice quality of the codec with different bitrates. In the figure, objective voice quality results for WB signal is presented. X-axis denotes the bitrate in kbit/s, and y-axis denotes the MOS (Mean Opinion Square) score measured with POLQA (Perceptual Objective Listening Quality Assessment) measurement methods specified by ITU-T [57]. The voice quality measured with this method compares the original input PCM and the decoded output PCM together in a psychoacoustic model [57]. The comparison results are subsequently mapped to the MOS values that are common to the ACR (Absolute Category Rating) specified in the ITU-T recommendation P.800 [58].

As Figure 8 shows, the calculated MOS scores vary more with the lower bitrates compared to higher. However, they do not differ remarkably with the complexity values above 5. In other words, the complexity does not affect the sound quality significantly above this limit. Also other bandwidths show similar results, as presented in the [59].

3.5 Applications

The Opus codec was originally developed for interactive Internet applications [48]. Applications such as Skype and WebRTC (Web Real-Time Communication) are both based on VoIP, i.e., the encoded speech is digitized in voice packets and transmitted via Internet Protocol (IP) [60]. The former one of these is a peer-to-peer application, in which the connection is established between the two end users (peers) directly, and thus the application is processed on the peer's own hardware [61]. The latter one, provides an Application Programming Interface (API) so that developers can write rich, real-time multimedia applications on the web, i.e., the application is run within a browser and no external plug-ins or installable software is needed [62]. However, the actual processing is still performed on the user's hardware.

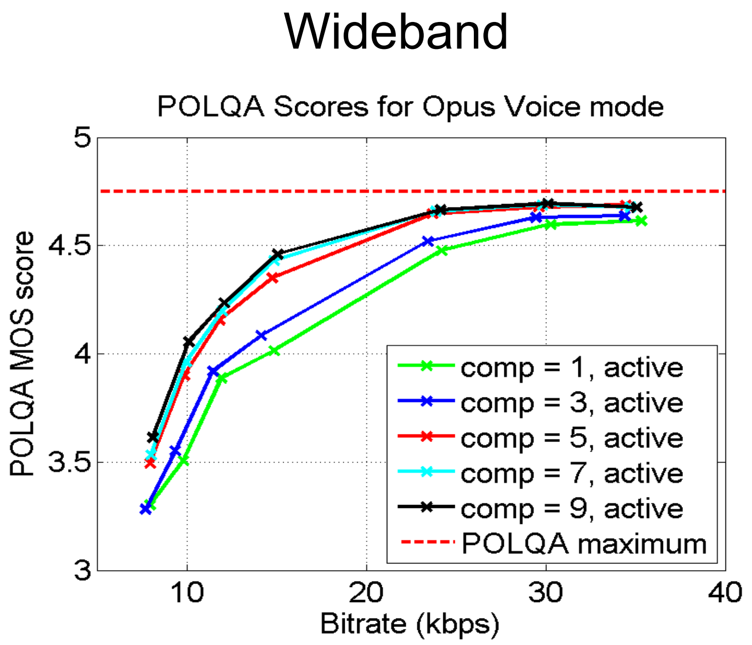


Figure 8: Speech quality with different complexities and bitrates measured for WB data with POLQA, adopted from [59].

The application to which the codec was integrated in this thesis was an IMS network element. This provides functionality for matching different codecs of two communicating networks. E.g., when a VoIP call is transcoded to use network's native codecs, the network element's hardware is used for matching the differing codecs.

The basic VoIP applications presented in this section are based on performing the processing on the end user's hardware. Therefore, they are not completely comparable to the processing performed in the IMS functions such as MGW, in which one piece of hardware is designed to perform the data processing for as many instances as there is capacity. This sets certain requirements to the computational efficiency of the application, and also to the hardware used for the processing. The latter one is discussed more in detail in Chapter 4.

4 Signal Processing Platforms

The network elements that provide signal processing functions, such as MGW, have been traditionally implemented on dedicated hardware (e.g. on an ATCA platform (Advanced Telecommunications Computing Architecture, [63])) to provide the required computational performance [64]. The piece of hardware that performs the actual calculative signal processing algorithm processes is called a processor.

For digital signal processing, a customized processor called Digital Signal Processor (DSP) has been generally used due to its ability to perform signal processing tasks effectively [65]. However, as the general purpose processors (GPP) have developed, their usage has become more reasonable also in digital signal processing. As a result, there has been an effort to move digital signal processing applications to more generic hardware, such as servers, so that no dedicated hardware is needed [3, 66]. This way, the signal processing tasks can be performed natively on the hardware, or within a virtual machine (VM) that is run on the server. The latter option enables one server to provide multiple VM instances and thus provide an effective way to share resources of one piece of hardware [67, p. 525]. Moreover, because the hardware is generic, there is no need for designing dedicated hardware and thus the product's development costs are decreased [3].

In this chapter, the signal processing platforms used in the network elements are reviewed. First, a generic overview of a processor is introduced by observing the common features that all the processors share. Second, both the DSP and GPP are introduced more in detail, after which they are compared in terms of performance. Third, the compilation of the source code to machine language is discussed, as it affects the performance of an application. Finally, the concept of a virtual machine is reviewed as it introduces certain differences compared to the native processing on the same hardware.

4.1 Processor Architecture and Performance

When choosing a processor for an application several performance related aspects need to be taken into consideration. On one hand, the computational speed may affect the choice when the processing times need to be reduced. On the other hand, if the application is part of mobile hardware with an external energy source (e.g., battery), the power related performance needs to be sufficient. In the scope of this thesis, mainly the execution speed related performance is discussed. Architecture is one key factor in processor's performance. There are many different types of processors that each serve different purposes. In the following section, general aspects of a processor architecture are discussed.

4.1.1 Architecture

When processing is performed on a processor, it flows through a certain path. This is called a data path, and along it the arithmetic operations of the processor are performed [67]. Another main function of a processor is control, which is responsible for commanding this data path, memory, and input/output (I/O). These two components depend highly on the architectural design of the processor. Instruction is a command that computer hardware understands and obeys, and the data path together with the control, affect on the manner, how these instructions are handled and computed [67]. As a result, the data path and the control, and thus the architecture of the processor, have a high impact on how efficiently the instructions get executed in a processor.

All the operations involving a processor are synchronized by an internal clock, which is oscillating with a constant rate [68]. This, the most basic unit for machine instructions, is called a clock cycle and is defined as the inverse of the processor's clock frequency. Furthermore, data path can be divided into different stages with a resolution of one clock cycle [67]. As a result, one instruction execution takes at least as many clock cycles as there are stages.

During instruction execution, the processing flows through the different stages of the data path such as instruction fetch, instruction decode and register file read, execution etc. [67]. In the most simplified case of instruction execution, so called single-cycle design, one instruction is executed at the time. This, however, is not very efficient because the previous instruction must be completed before the next one can be started [67, pp. 328-330].

Pipelining is a technique used in processor design, which utilizes parallelism "among the instructions in a sequential instruction stream" [67]. This means that after one stage of the data path is executed for a certain instruction, the same stage can be executed for the next instruction. For example, after the first instruction is fetched (instruction fetch stage), the execution for that particular instruction moves to the second stage of the data path. Simultaneously, the second instruction can be fetched so that during the same clock cycle, two different instructions are being executed only in the different stage of the data path. Consequently, pipelining can theoretically process as many instructions in parallel, as there are stages in the data path. In Figure 9 an example of a pipeline with a five-stage data path is illustrated. The five stages of the example data path are listed below [1] [67].

1. IF: Instruction fetch
2. ID: Instruction decode and register file read
3. EX: Execution or address calculation
4. MEM: Data memory access
5. WB: Write back

In the example presented in Figure 9, *IM* denotes the *Instruction Memory and the Program Counter* executed in the IF stage, *Reg* denotes the *Register File and Sign*

[1]In this thesis, abbreviation WB is stands for *Wideband*.

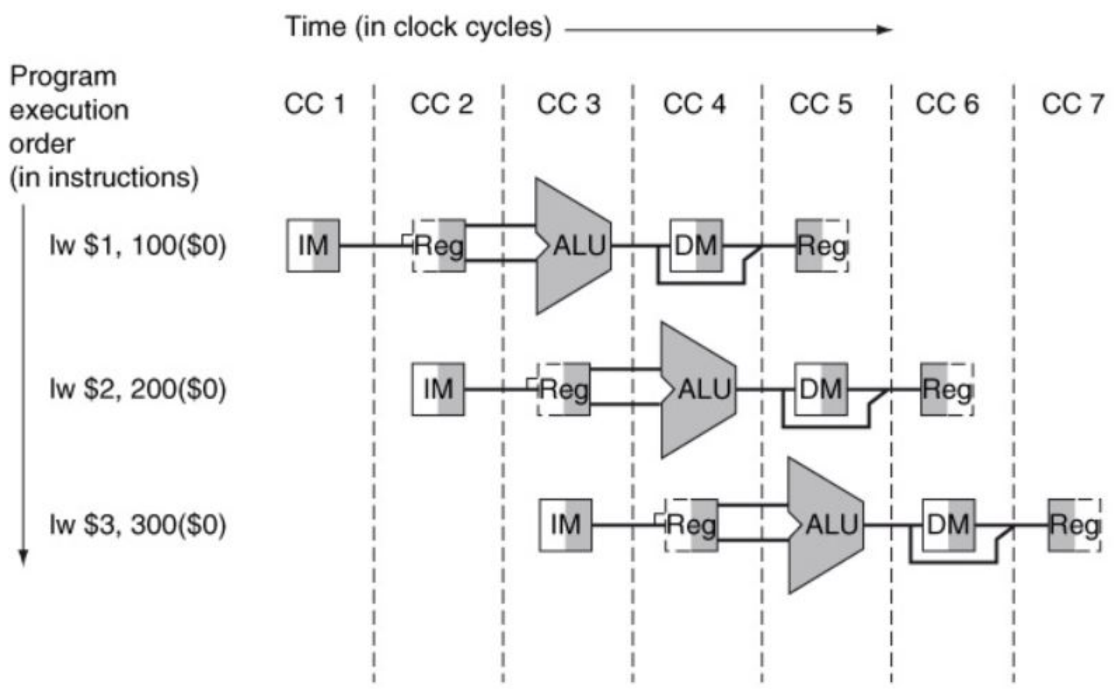


Figure 9: Example of a pipeline, adopted from [67].

Extender executed in the ID stage, *ALU* denotes the *Arithmetic Logic Unit* executed in the EX stage, and *DM* denotes the *Data Memory Access* executed in the MEM stage [67]. In addition, there is an additional *Reg* in the end, where the data is written back to the data register during the WB stage. Moreover, the scale on the x-axis, *Time (in clock cycles)*, denotes the elapsed clock cycles. The y-axis, *Program execution order (in instructions)*, represents the instruction to be executed. In this example, one instruction execution lasts five clock cycles.

In an ideal situation, when all the stages take one clock cycle to execute, pipelining allows one instruction to be completed every clock cycle [65, pp. 100-101]. This was the case in Figure 9, as instruction 1 is finished at CC5, instruction 2 at CC6, and instruction 3 at CC7. However, the data path's execution depends on the instruction to be executed and certain instructions may involve operations that require more clock cycles in some of the data path's stages [69]. Therefore, this ideal situation, cannot be always achieved. Naturally, some instructions need to be finished before others depending on the program's execution. E.g., when a result of a calculation is needed in the following operations, the instructions of that particular calculation need to be finished before the following operation.

Pipelining increases the computational speed because it is executing instructions in parallel. In fact, there has been an effort to increase the level of parallelism, rather than purely the clock rate, in order to gain computational efficiency [67, p. 632]. This is mainly because power consumption, which is proportional to the clock rate, has reached its limit in terms of cooling commodity [67, p. 39]. Using multiple processors to perform the computation increases the level of parallelism, because different processes can be run independently and simultaneously in these different processors [67, p. 632]. In practice, multiple processors are usually manufactured within one microchip and thus, one processor is actually a core of a microchip that consists of multiple cores. This design is called a multicore microprocessor [67, p. 632].

Another method for increasing parallelism is to use special SIMD (single instruction, multiple data) instructions (sometimes referred as Vector instructions) [68]. They allow the hardware to have many ALUs that operate simultaneously, or they divide one wider ALU into smaller ALUs that can operate simultaneously [67, p. 649]. In addition, when observing Figure 9, it can be seen that one instruction causes an overhead, e.g., when an instruction needs to be fetched, that is additional to the actual execution of the instruction. Thus, when multiple data operations can be executed with only one instruction, it decreases the execution time. There are several advantages in using vector instructions. These are listed in the below (adopted from [67, p. 652]).

- Vector instruction is equivalent to executing an entire loop
- Hardware does not have to check for data hazards within a vector instruction, but only between two vector instructions once per operand
- Data-level parallelism in SIMD is regarded as easier to implement when comparing to MIMD (multiple instruction, multiple data) multiprocessors
- The cost of latency to main memory is smaller because the vector is fetched entirely at once
- Control hazards caused by the loop branch are non-existent because vector instructions are well predetermined
- The savings in instruction bandwidth and hazard checking as well as the efficient memory handling reduce the power and energy costs

Even though parallelism can increase the performance, not all the operations can be run in parallel. Therefore, parallelism is often implemented as job-level parallelism or process-level parallelism in which complete bigger entities or processes are divided into parallel execution, instead of single operations of one process [67, p. 632].

In addition to instruction execution, also memory handling affects on how efficiently the processor can function [67, p. 453]. When an instruction is executed through the data path, the data needed for processing is fetched from the memory to a register, which is an integral part of the data path and which stores the value during the execution [65, p. 84]. Furthermore, the compiled and assembled program is stored onto the memory from which the instructions are fetched [65, p. 50].

Memory is often structured with a hierarchy that denotes how distant it is from the processor core in terms of efficiency [67, p. 453]. The faster the memory is, the more expensive it is to manufacture and thus, the fast memories are generally limited and used only for the most intensive operations [67, p. 453]. Therefore, slower memory technologies are also included so that more space can be obtained for the less demanding operations [67, p. 453]. To optimize the memory operations, processors may utilize the use of limited fast memory by temporarily storing essential data to it [67, p. 457]. This, so called caching, allows different stages of the program to use the fast memory when they are executed. Processors often include certain sized caches on their design by default (e.g., 6 Mbytes on Intel i7 4700MQ [70]).

4.1.2 Performance

Even though the clock rate of a processor denotes how many clock cycles are finished per second, it does not explicitly inform how fast the processor can execute in different applications. As was discussed earlier, instructions that the processor executes do not consume a constant number of clock cycles. Therefore, more descriptive units to measure performance have been developed. In this section, these units are discussed to choose a proper measure when evaluating the performance of the implemented codec.

As the data path is highly dependent on the processor's architecture, it is informative to measure how many instructions can be executed per second. This, often given as Million Instructions Per Second (MIPS), is a commonly used measure when talking about processor performance without presumptions of the application [67]. However, different architectures provide more suitable instruction execution than others for different purposes [65, p. 9]. Therefore, MIPS cannot be used alone when evaluating the processor performance.

For digital signal processing, there are common calculative operations that many algorithms share. These include an essential task, digital filtering, which is computed with an iterative operation called convolution, as was presented in Equation 5. In addition, many transform functions, such as the Fourier transform [56] or the cosine transform (Equations 3 and 4), are extensively used in signal processing, and are based on an integral function yielding an iterative sum function in the discrete domain. Thus, similarly to the filter functions, they introduce algorithmic delay to the program execution. In addition to involving a discrete sum function, each summand consists of a product of two factors. In other words, each iteration of the sum function includes a multiply-and-accumulate (MAC) operation, in which a product is calculated and accumulated to an existing value. This yields a formula as follows

$$d = a + (b \cdot c), \tag{6}$$

where a is the input parameter which the product is accumulated with, and b and c

are the input factors of the product. Subsequently, the calculated result is stored to the d variable. This operation has resulted in a very commonly used performance measure in the context of digital signal processing, which is given as how many MAC operations a processor can calculate per second [71].

As the digital signal processing algorithms are always part of a larger entity, the previously introduced generic measures do not apply for accurate performance calculations [72]. A more reliable method for measuring the performance is to measure the execution time of real algorithms, such as computationally intensive digital signal processing algorithms in the context of DSP [72]. Different organizations, such as BDTi (Berkeley Design Technology, Inc.), conduct benchmarks on different DSPs and other microprocessors by measuring the performance for algorithms related to the application to be benchmarked. For example, BDTi measures the DSP performance scores by measuring the execution speed of common and demanding digital signal processing algorithms, such as different filtering functions, vector operations, transform functions etc. [72].

Benchmarking with generic DSP algorithms have also limitations. They do not necessarily give reliable measures for application specific algorithms, and thus the actual execution speed may differ from the benchmark results [72]. In addition, processors that have customized units to perform certain specific tasks may score better in the benchmark because they perform the particular task more efficiently [72]. Finally, the DSP benchmarks often measure only the algorithm execution speed and the processor's memory usage, whereas other performance related factors are not included in the measurement [72].

The performance measures described in this chapter are descriptive, but cannot be used as the only input for application performance evaluation. The only method to provide reliable measures is to benchmark the performance with the actual application instead of generic algorithms. Only the benchmark results obtained with this type of measurement can give reliable suggestions of how good the performance is with the given application, as well as how big part of the whole processing capacity certain part of the application consumes.

4.2 Digital Signal Processor

A processor dedicated for signal processing purposes is called a digital signal processor. Its hardware is shaped by the digital signal processing algorithms and thus it is specialized to perform them efficiently and inexpensively [73]. Although, there are different architectural designs among DSP vendors, certain characteristics are shared with most DSP microprocessors. In this section, these common characteristics are reviewed.

Most DSPs have specialized arithmetic units integrated in the hardware that perform different digital signal processing tasks. These tasks can be characterized as repetitive and numerically intensive [65, 73]. For example, a very common unit in DSPs is the MAC-unit, which performs a MAC operation presented in Equation 6. This makes the computation of convolution and transform functions efficient, because the sum of products can be calculated by iterating only one MAC-instruction instead of separate instructions for multiply and addition. Furthermore, for additional precision DSPs generally provide extra bits for calculation results so that the result is rounded only after the operation is completed [65]. As a result, the rounding error is smaller compared to a situation where the result is rounded after each multiplication and accumulation separately.

In addition to the hardware units that support the algorithmic and arithmetic computation, DSPs include also other common features that support efficient digital signal processing tasks. Multiple-access memory architecture provides a support for several memory accesses during one instruction cycle [65]. Consequently, the processor may fetch an instruction while simultaneously fetching operands for the instruction or storing results. Moreover, specialized addressing modes provide efficient ways to perform digital signal processing algorithms. E.g., modulo addressing is an essential feature when calculating first-in, first-out (FIFO) type of operations, such as IIR or FIR filter computation where the newest values are saved on top of the oldest ones [65].

DSPs also provide commonly efficient execution control. E.g., hardware assisted loops allow the processor to repeat an instruction multiple times without an instruction fetch [65]. As a result, the overhead for the instruction fetch is decreased. Moreover, DSPs incorporate often complete peripheral devices (such as Analog-to-Digital (A/D) and Digital-to-Analog (D/A) converters) within the chip in order to allow low-cost, high-performance I/O [65].

The latest developments in DSPs have increased the level of parallelism with extensive vector processing improvements [71]. Moreover, data and program cache is currently a feature present in many DSP processors (e.g., TI TMS320C66x and TI TMS320C64x families [71, 74]), which has also improved the memory handling efficiency. For instance, a state-of-art DSP, TI TMS320C6672, is a multicore processor with extended SIMD support and cache handling, and can perform 48 GMACS (Giga Multiply-and-Accumulate Operations per Second) per core [71], whereas one generation older TI TMS320C6457 can perform 9.6 GMACS [74].

4.3 General Purpose Processor

As the name implies, general purpose processor is a microprocessor that is designed to meet various needs instead of specific type of processing as was the case with DSP. Typically, GPP is the first choice when designing control, user interface, or communication functions [65]. Moreover, the development kits for GPPs are typically sophisticated and developed, thus reducing the costs by simplifying the development process. However, for computationally intensive algorithms, such as the ones used in digital signal processing, GPPs have traditionally been less efficient [65]. As a result, also GPP vendors have started to include different functionalities to extend their processors so that they can perform the intensive digital signal processing at a decent efficiency.

There are different GPPs available for various purposes. In embedded systems, commonly used GPPs are ARM (Advanced RISC Machine) and MIPS (Microprocessor without Interlocked Pipeline Stages)[2], both of which are based on RISC (Reduced Instruction Set Computer) instruction set [67, p. 161]. In personal computers, the most widely used GPP is Intel's x86 architecture [67, p. 77]. Furthermore, servers where the virtualized network elements are implemented, are also commonly equipped with an x86 processor.

The most commonly used GPP, x86, was developed by Intel. Its register structure varies depending on the particular processor [68, 75]. However, generally an x86 processor includes basic program execution registers which are used for basic arithmetic and data movement as well as memory address handling [68]. These registers include general-purpose registers, segment registers, flag register, and instruction pointer register.

Through the years, an effort to provide better performance has resulted in various extensions in the processors, many of them based on the SIMD architecture and better floating point support [67, 68]. The first SIMD-extensions date back to the late 90's when MMX (Multimedia Extensions) were introduced. The latest x86 extensions, e.g., AVX (Advanced Vector Extensions) and AVX2, provide SIMD registers that are up to 256 bits wide [76].

Also other extensions in newer generation Intel x86 processors, e.g. FMA3 (Fused Multiply-Add) in the Haswell processor, support digital signal processing [77]. FMA provides a support for high-performance multiply and add operation, i.e. $d = \pm(a \cdot b) \pm c$ [78]. This corresponds to the MAC-formula presented earlier in Equation 6. FMA is also a SIMD based instruction set extension, i.e., the it can be performed to multiple values simultaneously [78].

In addition to an extensive offering of SIMD extensions, x86 processors include high level of parallelism with an advanced use of multicore design. E.g., Intel Core i7-4690X processor incorporates physical 6 processor cores, which are extended to 12 logical cores through hyperthreading [79]. Thus, very high process-level parallelism can be achieved with the particular processor.

[2]In this thesis, abbreviation MIPS stands for *Million Instructions Per Second.*

The clock rate of an x86 processor is typically higher than the one used in DSPs or in other GPPs. E.g., a state-of-art DSP processor, Texas Instruments (TI) TMS320C6672 has a clock rate of 1,5 GHz [71], whereas Intel's x86 clock rates may be up to 4 GHz [79]. This can be understood by observing the purposes that these processor are used for. Because the processor's power consumption is proportional to clock rate, the processors need to have low enough clock rate when low power consumption is needed. Therefore, processors such as MIPS, ARM, or DSP are typically used in the embedded systems. The x86 processors, however, are mostly used in the desktops and servers where the overall power consumption is less important, and thus higher clock rates can be used.

4.4 Platform Comparison

As was discussed in this chapter, there are certain essential differences between the DSP and GPP processors. DSP is more dedicated to digital signal processing tasks, whereas GPP offers more general processing compatibility. Generally, GPP is used in applications that involve rather general tasks such as basic user interface, control, and communications functions [65, p. 18]. In addition, when the application is undemanding in terms of performance, GPP is often the preferred choice due to its simplicity in development. This, however, applies only in the most simplified uses of GPPs, especially now, when there is a large variety of different extensions to the basic functionality. E.g., the SIMD extensions to x86 architecture have improved the performance of complex algorithm calculation remarkably [67], but also complicated the development with high amount of parallelism involved.

As the x86 extensions have introduced hardware improvements to the processors, the distinction to the DSP has diminished. E.g., the specialized MAC unit of the DSP has a counterpart in the x86 as the FMA performs essentially the same operation. On the other hand, the DSPs have also included more parallelism through multicore design and SIMD instruction extensions. Thus, the means of how the data is processed begins to resemble the x86.

For digital signal processing tasks, DSP has traditionally outperformed GPPs. In Figure 10, BDTi floating point single core processor benchmark results for the year 2013 are presented [80]. In the figure, the higher the score is, the better the performance is. The results are based on the measurements for typical digital signal processing algorithms. As can be seen in the figure, an older generation x86, Intel Pentium III [81], performs decently in the DSP comparison. However, the DSP TI TMS320C66x has flawlessly the highest score in the benchmark. When the processor's clock rate is taken into account, the actual efficiency reflected to the processor's full capacity can be measured. This is illustrated in Figure 11, where the BDTImark results are divided by the processor clock rates.

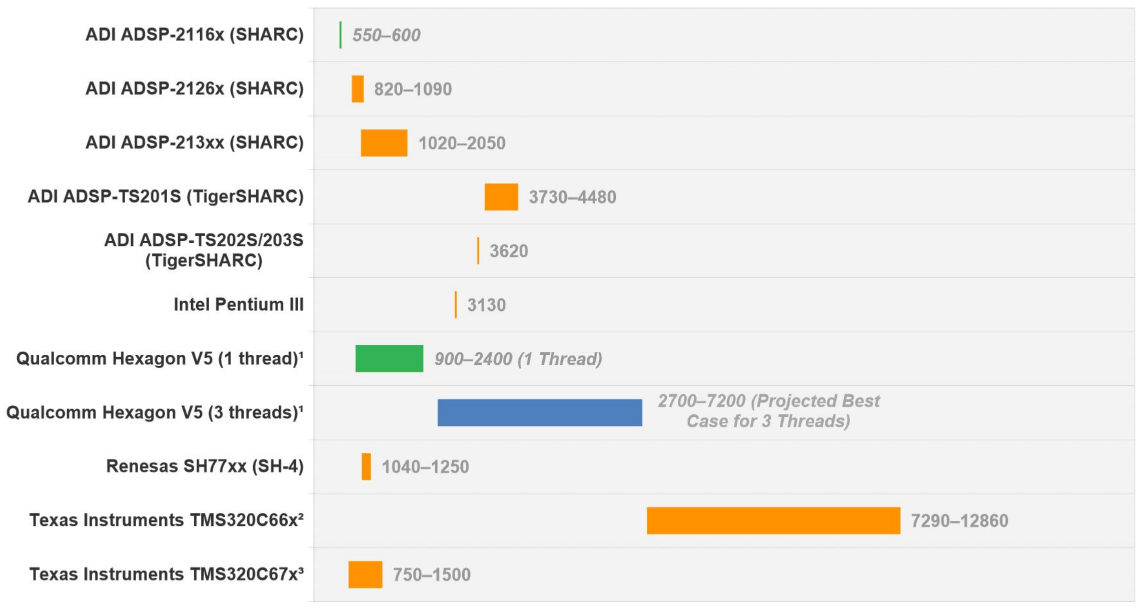


Figure 10: BDTI floating point single core processor benchmark results for the year 2013 (higher is better), adopted from [80].

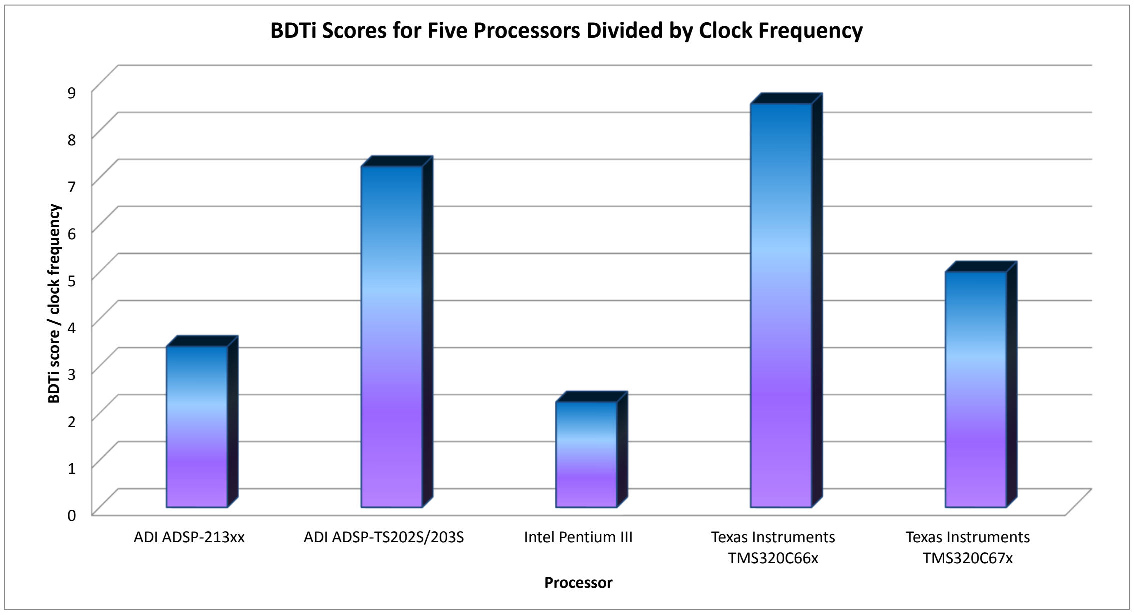


Figure 11: Selection of benchmark results proportioned to the clock rate, calculated from the values of the BDTi benchmark 2013 (higher is better), adapted from [80].

As Figure 11 shows, the BDTImark results proportioned to the clock rate are higher for dedicated DSPs compared to the GPP Intel Pentium III. In other words, signal processing tasks can be computed with less computational costs using DSP as less of the full processor capacity is needed during the signal processing.

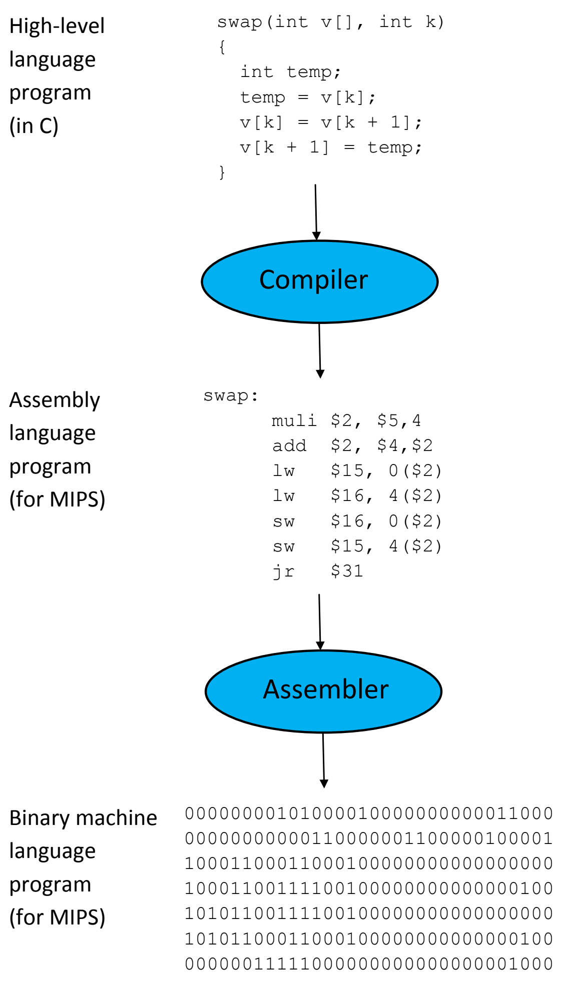
There was no corresponding data where modern x86 processors are compared against the DSP processors. However, the implementation conducted for this thesis aims partly to measure the difference in the signal processing performance, when a modern codec is run on modern x86- and DSP-processors.

4.5 Source Code Compilation

The instructions that the processor executes depend on how the software is developed. When low-level programming language such as Assembly is used, the instructions are listed directly in the source code. However, for larger programming entities Assembly is regarded as rather complex and therefore languages with higher abstraction levels are used for this purpose [67, p. 13]. For instance, the ANSI C-language is a high-level programming language, which is not tied to any specific platform [67, p. 13]. In the compilation phase, it is then compiled into the assembly language so that it can be assembled to the machine language [67]. Consequently, it is the compiler that generates the instructions and therefore is highly related on the instruction execution of a program. A general overview of compilation to lower level instructions is illustrated in Figure 12. In the figure, a short C-language function, *swap*, is compiled first to an assembly language program, which is platform specific, after which it is assembled into binary machine language program.

As the program execution depends highly on how the instructions are executed within a processor, the compiler may be a significant factor regarding the performance of the application. Consequently, many processor manufacturers provide their own compilers that are optimized to their processors (e.g. Intel C++ compiler for Intel processors, and Texas Instruments C6000 compiler for Texas Instruments TMS320C6x DSP processors) [82, 83]. In addition, there are also compilers that provide support for multiple different target platforms [84]. For example, GCC (Gnu Compiler Collection) is a widely used compiler and supports many different targets and programming languages [84]. The compilers provided by processor vendors offer support for only their own architecture and instruction set. Therefore, if portability between different platforms is desired, more generic compilers need to be used.

The compiler used for the implementation conducted for this thesis was GCC. As default, GCC aims at reduced cost of compilation time and making the debugging produce expected results [84]. However, there are different optimization options that are mainly toggled with compiler flags, and which enable the compilation with the goal of optimal performance. Three different levels of optimization are provided in GCC if the level of no optimization is excluded. With the basic level (flag -O or -O1), GCC performs optimization that does not increase the compilation time significantly. With the next optimization level, flag -O2, more optimization is performed with enabling all the other optimizations than those that involve a space-speed tradeoff [84]. With the highest optimization level, -O3, the best performance can be achieved. This attempts to, e.g., vectorize loops that can be vectorized.


High-level language program (in C)
Compiler
Assembly language program (for MIPS)
Assembler
Binary machine language program (for MIPS)

```
swap(int v[], int k)
{
  int temp;
  temp = v[k];
  v[k] = v[k + 1];
  v[k + 1] = temp;
}
```

```
swap:
      muli $2, $5,4
      add  $2, $4,$2
      lw   $15, 0($2)
      lw   $16, 4($2)
      sw   $16, 0($2)
      sw   $15, 4($2)
      jr   $31
```

```
00000000101000010000000000011000
00000000000110000001100000100001
10001100011000100000000000000000
10001100111100100000000000000100
10101100111100100000000000000000
10101100011000100000000000000100
00000011111000000000000000001000
```

Figure 12: Compilation and assembling of a high-level programming language, adapted from [67, p. 12].

4.6 Virtual Machine

The increasing trend of moving the network element's processing to a virtualized environment introduces a need of dividing the hardware's capacity efficiently. This can be done with an extensive use of so called virtual machines (VM). They are run as guests on top of the host operating system (OS) of the hardware. Although the hardware is used by the VM, there are certain differences comparing to the situation where applications are run natively.

Virtual machine (VM), by broadest definition, includes "all emulation methods that provide a standard software interface" [67, p. 525]. However, the VMs used in the cloud applications need to provide a complete system-level environment, i.e., at the binary instruction set architecture level. In this type of VM, the hardware resources are shared by all the virtual machines on the particular hardware [67, p. 525]. Virtual Machine Monitor (VMM), also called the hypervisor, is a software that is responsible for mapping the virtual resources to the physical ones [67, p. 526]. This provides a possibility for abstraction so that a VM instance (guest) can run the complete software stack independently, and consequently a complete OS can be run on top of VM with VMM. Hypervisor also provides a possibility to run multiple VM instances on one piece of hardware (host) [67]. This is a significant benefit as the used hardware can be divided efficiently.

The overhead caused by the virtualization is determined as the instructions that need to be emulated by the VMM, i.e., the instruction is emulated by the software, and the execution time of this emulation [67, p. 526]. It varies and depends on the workload and application [67, p. 526]. Generally, the I/O intensive workloads increase the overhead because they often execute many system calls and privileged instructions. However, certain types of operations, such as user-level processor-bound programs, result in almost no overhead at all, because the OS is rarely invoked and thus everything runs at native speed [67, p. 526].

VMM behaves on a VM highly like on a native hardware apart from the performance related virtualization. In addition, it isolates and protects the VMM from other guests running on the same host [67, p. 527]. Moreover, the "guest software should not be able to change the allocation of real system resources directly" [67, p. 527]. VMM also has a responsibility to control different system related operations, e.g., access to privileged state, address translation, I/O, exceptions, and interrupts. For example, when a timer interrupt occurs, VMM handles it by saving the guest VM's state, handling the interrupt, determining which guest VM to run next, and load its state [67, p. 527]. Due to these reasons, VMs provide a safe environment to run multiple instances as the error situations in one VM can be handled so that other instances are untouched.

The network element applications introduced in Section 2.1.1 include various processes, one of which audio coding is. The virtualization overhead caused by the high I/O nature of a process is not really relevant in the codec processing, as the codec is a user-level and processor-bound program. As a result, the performance related effects caused by the VM are not studied in this thesis.

5 Realization of the Opus Codec on x86 Platform

The implementation conducted for this thesis included integration of the Opus audio codec into transcoding units of a MGW run on a virtual platform equipped with an x86 processor. Furthermore, the performance of the implementation was evaluated and compared to the performance of a DSP implementation. The DSP version was a third-party implementation optimized for a TI TMS320TCI6486 DSP processor.

In this chapter, the detailed specification of the implementation is reviewed. First, the integration of the codec is introduced by reviewing an Opus Application Programming Interface (API). Furthermore, an implemented framework, which is a matching layer between the external application and the Opus API is introduced. Finally, the compilation of the source code is reviewed including the used compilation optimization details.

5.1 Integration

The integration of the Opus codec was performed with the API provided by the developers of the codec (IETF). It provides all the functionality needed for coding with Opus, as well as packing the coded packets [4]. The version of the integrated codec was 1.0.2, because it was the version used in the third-party DSP optimized implementation and the bit-compliance between the two architectures was needed for testing.

The API is divided by the functionality so that the high-level interface (called Opus encoder and Opus decoder) calls the lower level algorithm functions, which are divided into separate source files and grouped into separate SILK and CELT groups [42]. In addition, separate initialization and release functions are provided for allocating, initializing, and releasing the encoders and decoders used for the coding. Furthermore, the API also packs the processed frames into Opus packets, which include the information about the parameter configuration and the packet itself [4].

There is a support for both fixed-point and floating-point implementations included in the API. The format could be chosen with a compiler flag, and for this project, fixed point implementation was chosen as the DSP implementation used this arithmetic format.

5.1.1 Internal Opus Framework

The codec's integration required a layer that performed the matching between the external network element application and the Opus API. Therefore, a framework responsible for this matching was developed. In Figure 13, an overview of the implemented framework is illustrated. All the blocks inside the *Opus framework* entity are introduced in this section.

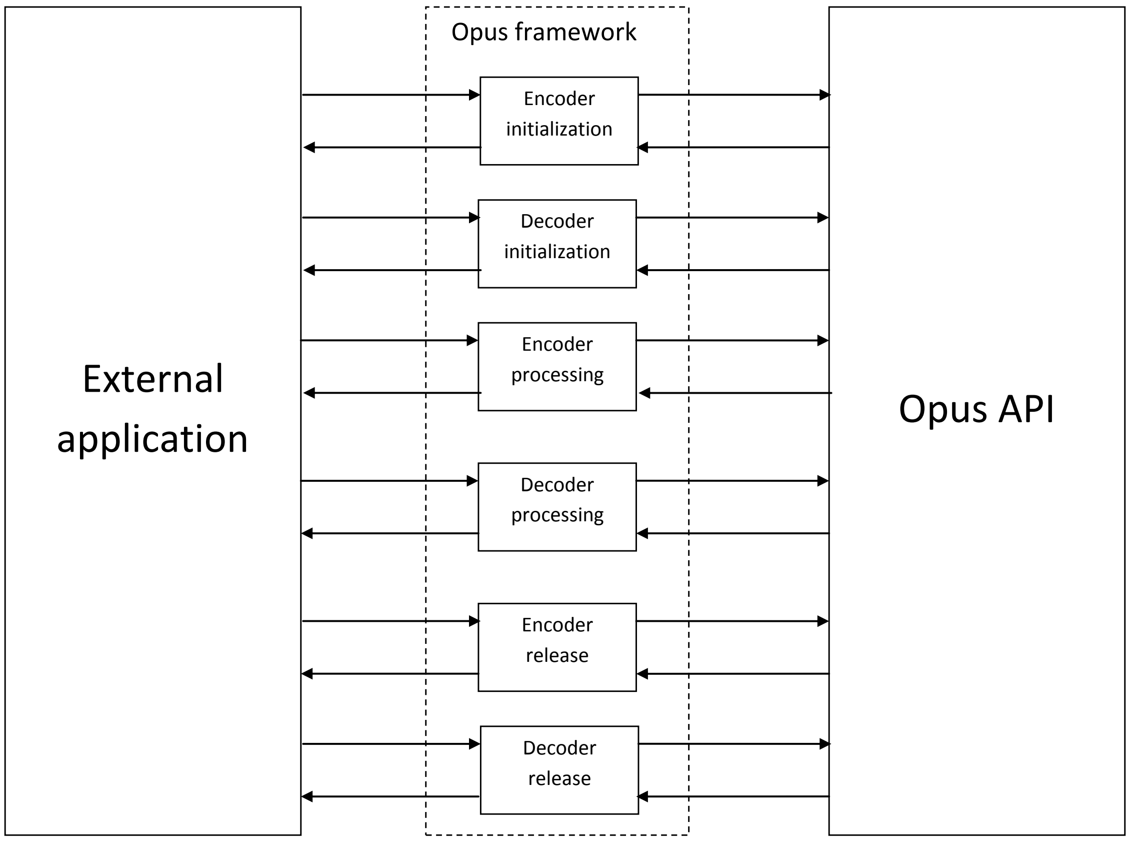


Figure 13: Overview of the implemented Opus codec framework.

The API provides by default an automatic memory allocation needed for the codec's internal functionality. However, there is also a manual memory allocation support where the developer can control explicitly how the memory is allocated. The latter one was chosen to ensure a controlled memory allocation behavior. The initialization of the codec includes the allocation of the memory needed for the encoder or decoder instance, and configuring it with the right set of initial parameters. After the initialization, this instance can be used for the data processing.

The implemented framework performs the needed memory allocations by calling the API's initialization functions, and allocating the needed memory. Furthermore, it initializes the instance with the initial parameters received from an external control I/O. After initialization, the newly created encoder or decoder instance is passed to the external application so that it can be used when the data is ready to be processed. When the coding is finished, the implemented framework destroys the used codec instance by calling the API release functions. In addition, as the internal memory was allocated manually, the freeing was also implemented within the framework.

For the actual data processing, the dynamic coding parameters for each processing instance are partly received from the external application and partly fixed (discussed more in Section 5.1.2). The implemented framework modifies the encoder or decoder instance's state with these parameters, after which it performs the API function calls for data processing. The data is received from the external application with a resolution of one frame packed with internal packing containing information about the input data in the header and the actual input data in the payload. At this stage, the data containers for the input and output data have been already allocated by the external application and thus the input data is simply passed to the API for the processing.

In addition to the basic functionality of calling the codec API, the framework included support for features like DTX and PLC, as well as error handling in order provide a controlled behavior in unexpected situations. Error handling included control and data I/O check to determine whether the input data is intact. In addition, the internal status of the encoder or decoder instance was checked, as the Opus API reports errors by returning an error code in the error situations, and indication about the status being normal when no error is detected. The implemented framework also included error reporting to the external application by an extensive use of error codes.

5.1.2 Control I/O

As the encoder supports a large set of control parameters, the framework was implemented to support calling the Opus API with changeable control information. The dynamic control I/O is received from outside of the coding unit depending on the parameter set that is settled when initiating a connection. Only a subset of all the parameters supported in Opus by default was supported in the integrated implementation. These are introduced in this chapter.

First, the application selection (VoIP, Audio, or Restricted Low-Delay mode [42]) was set to VoIP due to the likelihood of the information being speech is higher compared to general audio. Second, the complexity parameter was set to a constant value after its effect on the performance was evaluated in the performance testing phase. This will be discussed more in Section 6.3. Third, the supported audio bandwidths for the encoder were limited to NB and WB, because the incoming PCM stream from the external application is currently limited to these two bandwidths. Fourth, the frame length was set to a constant of 20 ms, as the standard, RFC6716 (Request for Comments) [4], recommends it as the optimal value for most cases. As a result, also the used bitrates were set to the recommended values for this frame length [4]. For NB, the recommended bitrate is 8 ... 12 kbit/s and for WB 16 ... 20 kbit/s [4]. The higher limits of these were chosen to maximize the quality. Fifth, the variable bitrate mode was chosen to be used as the Opus codec operates more efficiently with this selection [4]. In Table 4, the supported parameter settings for encoder are summarized.

Table 4: Opus parameter settings and constant values of the encoder supported in the implementation.

Supported Encoder Parameters	
Parameter	***Defined value***
Application selection	VoIP
Complexity	Fixed after performance tests, initially set to 5
Bandwidth	NB and WB
Frame length	20 ms
Bitrate	12 kbit/s for NB, 20 kbit/s for WB
VBR or CBR selection	VBR

The parameters that are controllable from the external application are transmitted to the framework with a control I/O data structure. The framework sets the correct runtime parameters to the encoder state via the API functions with the use of the the parameters received. The control I/O data structure is an external structure defined in the external application and did not include all the parameters needed by the Opus codec. Therefore, the control I/O was appended with the parameters needed.

For the decoder, most of the essential information is transmitted within the Opus packet (described more in detail in Section 5.1.3). However, there are certain control parameters that the decoder needs for decoding. These are received from the external application similarly to encoder's control I/O, and are dependent on the external parameter requirements, such as the sample rate of the other codec in the transcoding procedure.

First, the output sample rate can be chosen with a control parameter. The Opus decoder supports all the sample rates presented in Table 3, and regardless of the original sample rate of the encoded bitstream, the decoder performs a sample rate conversion internally when needed [4]. Similarly to the encoder, the supported sample rates for the output were chosen to correspond the network element's requirements, thus 8 kHz and 16 kHz.

Second, the output channel format, i.e., mono/stereo selection, could be chosen with a decoder control parameter. As the output of the decoder needs to match the input of the other codecs of the transcoder unit, the channel mode selection was fixed to mono. Third, FEC selection needs to be signaled to the decoder, so that it can make use of the redundant data in the encoded bitstream [4]. Fourth, the support for DTX was implemented along with the PLC functionality, as these are performed with the same algorithm within the codec. When no data is received, a one-byte Opus packet containing the previously decoded frame's header byte is passed to the API. Table 5 summarizes the supported control I/O parameters for the decoder.

Table 5: Implementation's supported Opus parameter settings and constant values for the decoder.

Supported Decoder Parameters	
Parameter	***Defined value***
Output sample rate	8 kHz or 16 kHz
Channel mode selection	Mono
FEC	Controllable parameter
PLC/DTX	Invoked when no data is received and no FEC is available

Previously, the external application did not transmit runtime control information to the decoders of the other codecs. Therefore, as a part of the implementation a decoder control I/O data structure was created with the runtime control parameters introduced in this chapter.

As the VBR produces arbitrary sized packets, the size of the Opus packet needs to be transmitted with the input control data. This is needed for the framework to be able to pass the correct amount of data to the decoder from the buffer. Opus packet size is relevant information throughout the external network element and therefore internal packing used in the data I/O of the external application was appended with the size information of the Opus packet. The size information is originally derived from the RTP/UDP packing received from the network.

5.1.3 Opus Packing

Internal packing is used for the Opus codec to be able to transmit the needed control information of the encoded bitstream. The encoded frame or frames are packed into a single packet according to the specification presented in the RFC6716 [4]. The packing is performed within the codec and thus no external functionality for Opus packing is needed.

In the simplest form, Opus packet contains only a single frame or all the encoded frames in the packet share the same set of configuration parameters that are defined in the packet's header [4]. The package itself does not contain the length information, but instead it assumes that the lower level packing (such as UDP or RTP packet) includes the information about the packet length [4]. This reduces the framing overhead because the length information can be omitted from the Opus packet.

The first byte of the packet, the TOC-byte (table-of-contents), contains the information about the configuration that was used when encoding the frame [4]. The first five bits of it contain the actual configuration information, which is one of the 32 possible combination of operating mode, audio bandwidth, and frame length. The sixth bit of the TOC-byte contains the information about the number of encoded audio channels, i.e., 0 corresponds mono and 1 stereo. The remaining two bits include the information of the number of frames within the packet. The number of frames is informed so that 0 corresponds to 1 frame, 1 corresponds to 2 frames (with equal compressed sizes), 2 corresponds to 2 frames (with different compressed sizes), and 3 corresponds to an arbitrary number of frames in the packet [4].

In addition to the TOC-byte, there may be a need for transmitting additional configuration information within the packet. Information about the configuration used for encoding, presented also in Table 2, is included in the packet by expanding the header with the needed parameters. The expanded header is not arbitrary length, but predefined depending on the two last bits of the TOC-byte (the ones that contain the frame number information) [4].

In the case of Code 2 packet, the header is extended with one or two bytes, which contain the information about the first frame's length. In the case of Code 3 packet, which is the most complex case, the header is expanded with the information on number of frames, as well as optional padding information. Additionally, the frame count byte in the expanded header contains the information whether variable bitrate and padding is used. In the case of VBR, an additional subheader is included where the number of bytes for each frame except the last one is signaled [4]. The last frame's length equals the remaining length of the packet excluding the known padding length. In Figure 14 the most complex Opus packet type of Code 3 with VBR enabled is illustrated.

In Figure 14, the two bits after TOC-byte denote the VBR option and padding selection (on/off). M denotes the packet count and indicates the number of frames in the packet. It is followed by the padding length, which stands for the length of the padding in the end of the packet if used. This is followed by the frame length information when VBR is used.

The packing results only in certain types and sizes of packets and therefore, all the packets not meeting these specifications need to be discarded. The Opus codec has a built-in handling for malformed packets, which checks whether the packet fulfils the requirements as specified in the Opus standard [4]. Therefore, no external check for packet's integrity was needed. When a malformed packet is received, an error code is returned, which the implemented framework passes to the external application for further error reporting.

5.1.4 Memory Allocation Requirements

The memory allocation requirements that the codec needs for internal computation were predefined as the internal buffer lengths are fixed. However, the data buffers used for passing the PCM data and encoded bitstream data to the codec's API need to be allocated to contain enough space for the maximum possible data length. Although these buffers are received from the external application, they must contain

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| config  |s|1|1|1|p|     M     | Padding length (Optional)     :
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
: N1 (1-2 bytes): N2 (1-2 bytes):     ...       :     N[M-1]    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
:               Compressed frame 1 (N1 bytes)...                :
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
:               Compressed frame 2 (N2 bytes)...                :
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
:                              ...                              :
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
:                     Compressed frame M...                     :
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
:                  Opus Padding (Optional)...                   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 14: Opus packet configuration with the Code 3 type of packing, adopted from [4].

enough space for the data needed in the Opus codec. In this section, these space requirements are presented. First, the maximum possible PCM audio length is computed, after which the same measure for the Opus packet size is computed.

For PCM, i.e., encoder's input and decoder's output, the maximum number of samples can be calculated from the maximum frame length and maximum sample rate supported in implementation (presented in Table 4). The bit depth in the case of PCM used by Opus is 16 per sample, and thus the number of bytes (octets) is twice the number of samples [42].

The Opus codec's standard [4] states, that the decoder must be able to receive Opus packets that are encoded with arbitrary control parameter settings, i.e., any kind of Opus packet. Therefore, the maximum buffer size needed for the PCM data is defined as the maximum size that the decoder will output. As the decoder's output could be limited to 16 kHz mono, the dominating factor in the maximum buffer size allocation is the audio length that the Opus packet contains. With multiple frames per packet, an Opus packet may contain up to 120 ms of audio data [4]. Therefore, the maximum number PCM samples that the decoder may output is for 120 ms audio length of 16 kHz sample rate mono audio. Thus, the maximum number of samples is

$$n_{max} = N_{120ms} = \frac{T_{120ms}}{T_{fs}} = T_{120ms} \cdot f_s = 0.120 \text{ s} \cdot 16000 \ \frac{1}{\text{s}} = 1920, \quad (7)$$

where n_{max} is the maximum number of samples, N_{120ms} is the number of samples in 120 ms of mono audio with 16 kHz sample rate, T_{fs} is the sampling period, and f_s is the sample rate. Thus, byte being 8 bits, the maximum number of bytes is 3840. Although this is a large buffer, it does not introduce transmission capacity issues, as it is only used for internal data handling.

Buffer sizes for the encoded bitstream, i.e., the output of the encoder and input of the decoder, require generally less space as the encoded bitstream represents the compressed audio. However, they could not be calculated with a formula similar to PCM sample calculation presented in Equation 7 due to the highly varying parameter settings that affect the packet's payload size. Opus codec has internally limited the maximum encoded frame size to 1275 bytes, which corresponds to the size of encoded FB 20 ms frame with 510 kbit/s bitrate [4]. Furthermore, when multiple frames can be packed into one packet up to 120 ms of audio data, one Opus packet may contain six maximum sized encoded Opus frames. Hence, the maximum size of one Opus packet, i.e., maximum buffer size needed for containing the maximum possible encoded bitstream, is

$$n_{max,encoded} = N_{maxframe} \cdot \frac{T_{120ms}}{T_{20ms}} = 1275 \cdot \frac{120 \text{ ms}}{20 \text{ ms}} = 1275 \cdot 6 = 7650, \quad (8)$$

where $n_{max,encoded}$ is the maximum size of one encoded bitstream packet in bytes, $N_{maxframe}$ is the maximum size of one encoded frame, T_{120ms} is the length of 120 ms of audio, and T_{20ms} is the length of 20 ms of audio.

The maximum number of bytes of the encoded bitstream is large when it is compared to the capacity of an ethernet link whose Maximum Transmission Unit (MTU) is defined as 1500 bytes [85]. Thus, the maximum Opus packet does not fit into the MTU without fragmenting it into separate packets. This is not currently supported in the external application and thus the implementation was limited to support only one maximum Opus frame at a time. Therefore, the maximum supported Opus size was set to a limit of 1275. The buffer limit calculated for the encoder's PCM (Equation 7) is sufficient to contain the maximum Opus packet and therefore, the buffers needed for internal calculations are allocated with the PCM buffer limit of 1920 16-bit samples, i.e., 3840 bytes.

5.1.5 Opus RTP Packing

When transmitting data between network elements, RTP is used on top of UDP. IETF has standardized, how RTP packet should be packed when transmitting Opus bitstream [41]. Opus' RTP packing follows the RTP standard specified in the RFC3550 [10]. However, certain RTP packing characteristics are also specified in the Opus RTP standard [41].

As there are multiple different frame sizes and sample rates available, the timestamp increment of the RTP packet has been set to the maximum possible, which is for 48000 Hz sample rate with 2 channels [41]. For the control information needed, the IETF has standardized an SDP mapping specification so that a session with the Opus codec can use the right set of control parameters. However, the control information does not include the payload size or length information, and thus it must be included in the transport layer packet headers. Furthermore, the standard states that only one Opus packet must be transmitted with one RTP packet [41]. As a result, the RTP payload size corresponds to the contained Opus packet's size. This size can be obtained from UDP header, on top of which the RTP packet is transmitted.

The protocol level handling was not implemented in the scope of this thesis. Therefore, the control I/O of the implemented codec framework may experience changes if the parameter settings need to be modified for the protocol handling layer.

5.1.6 Packet Loss Handling and Discontinuous Transmission

When the encoded Opus bitstream is transmitted over an unreliable network packets may be lost. Therefore, Opus has included methods to improve the packet loss robustness. In these situations, normal decoding procedure cannot be performed, but special handling for lost packets is needed. These methods include FEC and PLC.

FEC is controlled with a flag through control I/O, which informs the decoder that the previous packet was lost and that the decoder should use the redundant information encoded to the bitstream from this lost packet. This information comes from the external application and thus no additional check for lost packets was needed in the implemented framework. The FEC flag is transmitted via control I/O, which the implemented framework passes directly to the Opus API.

PLC is another feature for improving additional packet loss robustness. Originally, it was an optional feature that was not included in the Opus standard [4]. However, in the version used for this implementation the PLC functionality was already built-in. It is invoked by passing a packet of one-byte to the decoder containing the TOC-byte of the previously decoded frame [42]. When a packet is lost the external application generates an empty packet with a header with size information of 0. When this type of packet is transmitted to the implemented framework, it invokes the PLC functionality.

Discontinuous transmission is supported in Opus for reducing the transmission load even further. However, the standard [4] recommends to use only VBR as it already includes the bitrate reduction when no voice activity is detected. Therefore, in the encoder the DTX support was not implemented. However, as the decoder must support all configurations, DTX handling was included on the implemented framework's decoder functionality. The Opus codec's DTX uses the same algorithms as are used in PLC. The API invokes the DTX similarly to the PLC and therefore the framework was implemented to pass a one-byte packet to the decoder with the TOC byte of the previous frame, whenever a packet was loss and no FEC data was available [42].

5.2 Compilation of the Codec

As the implementation was integrated with with a C-language API, it needed to be compiled. As was discussed in Section 4.5 the compilation plays a significant role in the program execution. Therefore, as part of the implementation conducted for this thesis, also performance related to the compilation of the API was benchmarked.

In the implementation, the API functions were compiled into libraries in order to reduce the compilation time of the whole application. Consequently, the functions could be linked to the application from the libraries. Therefore, the prolonged compilation time caused by the optimization flag was not taken into consideration when benchmarking the compilation.

The used compiler was GCC version 4.3.2 and a 32-bit compilation was used. The Opus encoder and decoder libraries were compiled with and without optimization. For the optimization, level -O3 was used for the maximal improvements in the execution. After the performance of these two levels was evaluated, the compilation optimization was fixed depending on the gained performance.

6 Testing

The implemented Opus codec was tested for two aspects: functionality and performance. The former one was required to determine whether the implemented codec was working as specified and was compared to the reference Opus application provided by the codec's developers. The reference application used the Opus API for coding and had a user interface, through which raw PCM data files could be fed to the API functions. Performance was measured to evaluate how efficient the codec's algorithms are and consequently how much of the processor's full capacity one coding instance consumes. Furthermore, the performance measurement results were used for benchmarking different parameter sets and to fix certain parameters to a constant value, e.g., the complexity.

6.1 Functional Testing

The functionality was tested via module testing. Thus, the codec module was tested individually with a given input and the output was compared to a reference. The module tests were built on an existing codec test framework. The codec to be tested had a separate configuration source file, as well as input, output, and reference data binary files which were used as the data I/O for the tests. In addition, the test cases were listed in additional configuration files, independently to encoder and decoder, providing a possibility to choose which test cases to be tested without recompiling the source code.

The module test procedure is illustrated in Figure 15. As the figure shows, the actual tests were run in two phases. First, the input files were opened and loaded to the memory after which they were encoded, for the encoder test cases, or decoded, for the decoder test cases. Subsequently, the encoded/decoded data was written to output files. Second, the output files were compared to the reference files so that if the output was bit-exactly identical to the reference file, the test passed, otherwise failed.

Encoder input contained PCM data which had a format that the Opus codec can handle, thus 16-bit PCM [42]. The only controllable parameter of the encoder, the sample rate selection, was transmitted from the test environment and originally read from the configuration file. As there were no test vectors for the encoder provided along the codec, they were generated with the given reference application using the same input files as in the tests. This can be seen in Figure 15 where the lower branch denotes the reference file generation. The generation of the used test data is explained more in detail later in this chapter.

Test data of the decoder contained Opus-packets. The packet's header included all the configuration data excluding the output sample rate, channel selection, and FEC usage. Therefore, these were transmitted from the test framework to the implemented codec framework with the same control I/O structure, as was specified

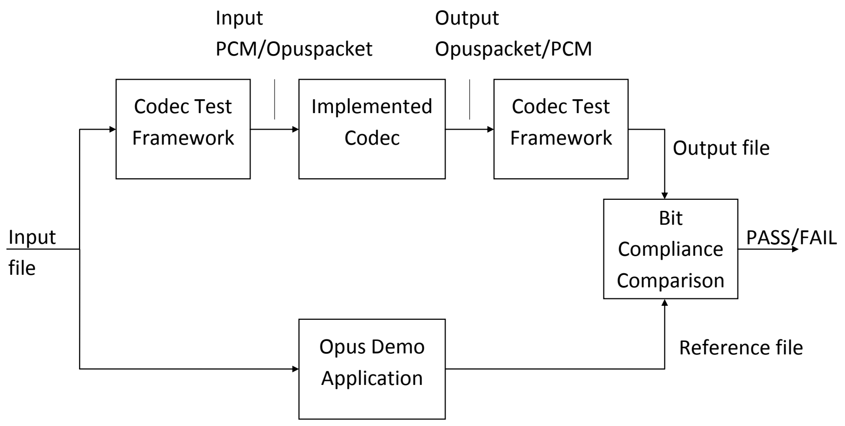


Figure 15: Module testing block diagram.

in Section 5.1.2. For the decoder there were test vectors provided along the codec. However, they were not bit exact vectors, but general conformance test vectors, which only tested if the implementation matches the reference within certain margins. To ensure that the implemented codec is fully bit compliant with the reference implementation, bit-exact test cases were generated with the use of the reference application also for the decoder.

In addition to tests with valid test data, the module tests were run with invalid input data. These included tests with invalid or corrupted Opus-packets, tests with NULL pointers, as well as tests with invalid internal framework data. These tests tested the error handling so that even in the error situation, the integrated codec did not abort the system unexpectedly. With internal error handling and valid use of error codes, the error situations could be handled to avoid uncontrolled behavior. Consequently, also the error reporting was tested in the invalid input data tests.

6.2 Test Data Generation

Because there were no bit-exact test vectors provided with the codec, they needed be generated. The essential aspect was to cover all the possible parameter configurations thoroughly.

The audio data was chosen from an existing audio bank. It consisted of raw 16-bit PCM files with a sample rate of 48 kHz. Furthermore, it included mainly various speech samples of different lengths, both with male and female speakers. Moreover, there were also music samples with the 48 kHz sample rate available. In addition to the continuous tests, the decoder DTX tests required data with silence or background noise periods. As a result, some test files were connected together with a several second gap in between. Furthermore, white noise was added with low level to the DTX files, as the input data in the mobile networks is very likely to contain background noises. From the FB PCM audio data, the lower sample rates could be obtained through sampling rate conversion [56]. For this purpose, ITU's toolset G.191 [86] was used.

After suitable raw PCM data was generated, it was driven through a reference application provided with the Opus codec [87]. For the encoder, the test cases consisted of the raw PCM data as an input, and the output of the decoder as the reference file for the actual test execution. The decoder test cases were generated so that the audio to be tested was first encoded with the reference application, after which the output of the encoder was saved as the input of the test case. Subsequently, it was driven through the reference application's decoder which output the reference to the test case. Moreover, the basic speech and music test cases with VoIP application selection were run with both of the supported output sample rates, 8 and 16 kHz.

The bitrate selection was set to the higher limits of the bitrate ranges recommended in the Opus standard [4] for all the audio bandwidths. However, an additional high-bitrate test for the decoder was included, in which the bitrate was set to maximum, 510 kbit/s. Furthermore, part of the tests was repeated with audio, and restricted low-delay application selections. As a result, with 20 speech, 2 music and 5 DTX PCM samples, almost 800 different test cases were generated.

6.3 Performance Testing

The performance of the implemented codec was tested by computing the execution speed for the encoder and decoder separately. For this purpose, the number of clock cycles elapsed for the coding was measured by reading the time stamp counter of the processor. The tests were run in the module test environment with a PC equipped with an Intel Haswell, Core i7-4700MQ, processor with a clock rate of 2.4 GHz [70]. Furthermore, the test application was run natively on Linux Mint with the application's process priority set to the highest and the scheduling class set to real-time. Furthermore, the test application's core affinity was set to only one core, so that no overhead of switching from a core to another during processing was introduced.

The actual execution speed integrated to the whole surrounding application could not be obtained with this test method, but the approximate execution time for purely codec processing could be evaluated. In addition, when the corresponding tests were run on a traditional DSP implementation, the performance could be compared between the two platforms.

For the DSP version, there was a third-party optimized implementation of the Opus codec for TI's TMS320TCI6486 processor. In the DSP implementation, the most computationally intensive parts of the code were implemented with Assembly language, and many general C-language functions were substituted with built-in functions, also known as intrinsics, which provide a possibility to call Assembly instructions from the C-code. The performance tests for the DSP were run with a TI's cycle accurate simulator for the particular processor.

The number of clock cycles does not indicate the execution speed totally because it is dependent on the clock rate of the processor. Furthermore, as the execution time depends on the input data's frame length, the performance was measured by computing how many clock cycles are consumed with a certain parameter set when processing one second of data. Consequently, when the data is processed so that it is segmented into frames, the measured clock cycle count needs to be multiplied with a factor of $\frac{1\,s}{T_{frame}}$, where T_{frame} is the frame length. Moreover, the number of cycles that the codec consumes when processing one second of data is a large number (when the used x86 processes 2.4 gigacycles per second). Therefore, the measure is given in megacycles per second. The number of megacycles in second can be computed with the following formula

$$N_{megacyclespersec} = \frac{N_{frame} \cdot 1s}{T_{frame} \cdot 10^6}, \tag{9}$$

where the $N_{megacyclespersec}$ is the number of megacycles consumed in processing one second of data, and N_{frame} is the measured cycles when processing one frame.

The clock cycles were measured simply by reading the Time Stamp Counter (TSC) register of the x86 and TMS320TCI6486 DSP processors, before and after the encode and decode function calls. This register stores the value of how many clock cycles have passed after the previous reset [88]. Thus, the difference in the value, read from this register before and after the encoder or decoder function call, results in the consumed clock cycles for encoding and decoding. However, this register is core specific and thus all processes run with the particular core affect the value within this register. For the x86 this yields a situation where system interrupts may occur during the execution resulting potentially in a bias. The DSP is not affected by this because the simulator calculates only the cycles spent for processing the algorithms.

Before executing the general performance tests, the compilation optimization levels were tested with the method introduced in this chapter. The tested levels were none, and the level -O3. After obtaining the results, the compilation optimization was fixed to the level with the best performance, after which other performance tests could be run on that level.

To determine the performance thoroughly, the number of clock cycles needs to be measured for all the parameter configurations supported in the implementation. Therefore, all the module test cases were run with the cycle measurements to collect data from all the different combinations. In addition, this type of performance analysis provides information about the most demanding parameter settings, as well as the maximum capacity needed for running one encoding or decoding instance. With this information, approximations of how many coding instances can be run on one processor core simultaneously can be derived.

The only parameter with separate performance tests was the implementation specific complexity parameter. It was tested by altering the value from 0 to 10 with an increment of 1, after which it could be fixed to the most suitable value. All the other parameter configuration performance tests used the predefined constant of 5 for the complexity.

7 Test Results

The implementation of the Opus codec was tested for functionality, as well as for performance on an x86 processor. In this chapter, the results of these tests are presented and evaluated with a comparison to the performance of a DSP. The processors that were used in the tests were Intel Core i7-4700MQ for x86 and Texas Instruments TMS320TCI6486 for DSP.

7.1 Functional Test Results

The functional tests compared the bit compliance between the output of the implemented codec and the output of the reference codec application. After the data was generated for all the tests, they were run through the implemented codec within the module test environment. All the test cases matched bit-exactly and thus, the requirement for bit compliance was met and the valid input data test cases passed the functional tests.

The error handling was tested with invalid input test set. The tests included tests with corrupt data including NULL tests, and conflicting parameter settings. The implemented framework reported error using predefined error codes in the error situations. The error reports of the tests were compared to previously generated reference of the report. All the error reports matched the predefined error codes and thus no unexpected behavior was detected during the tests. Thus, also the invalid input data tests passed the tests.

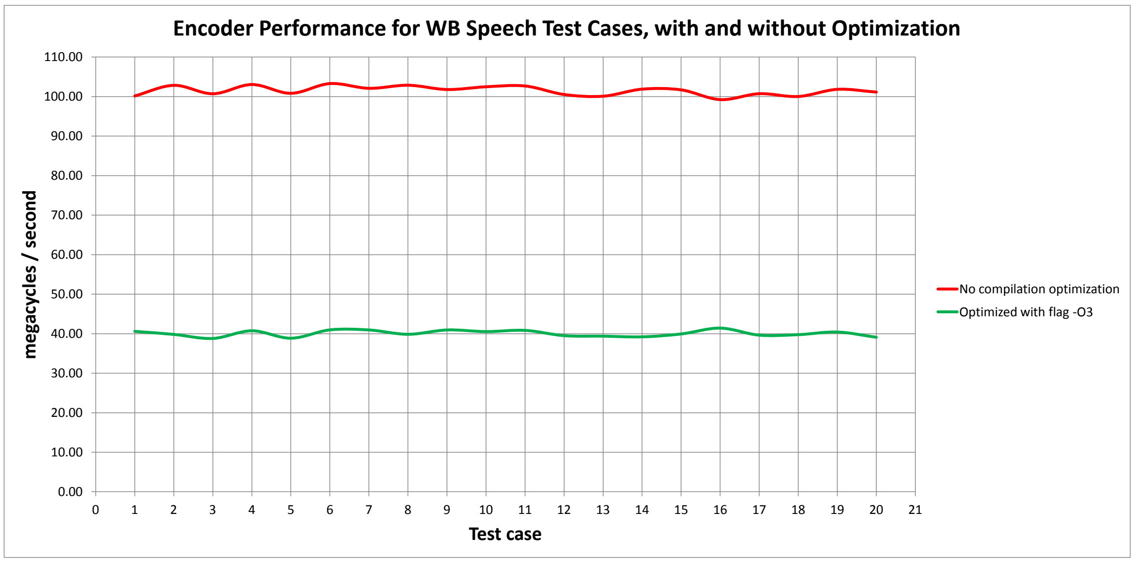
7.2 Performance Test Results

The performance tests provided information about the execution speed of the implemented codec as well as the capacity needed for one coding instance. In the following sections the results of the performance tests are presented. First, the compilation optimization test result is reviewed as its results were used in the subsequent performance tests. Second, the performance measured for each parameter configuration is reviewed, after which the complexity parameter test results are introduced and discussed. The results include also the DSP performance. Finally, at the end of this chapter the performances of these two architectures are evaluated and compared.

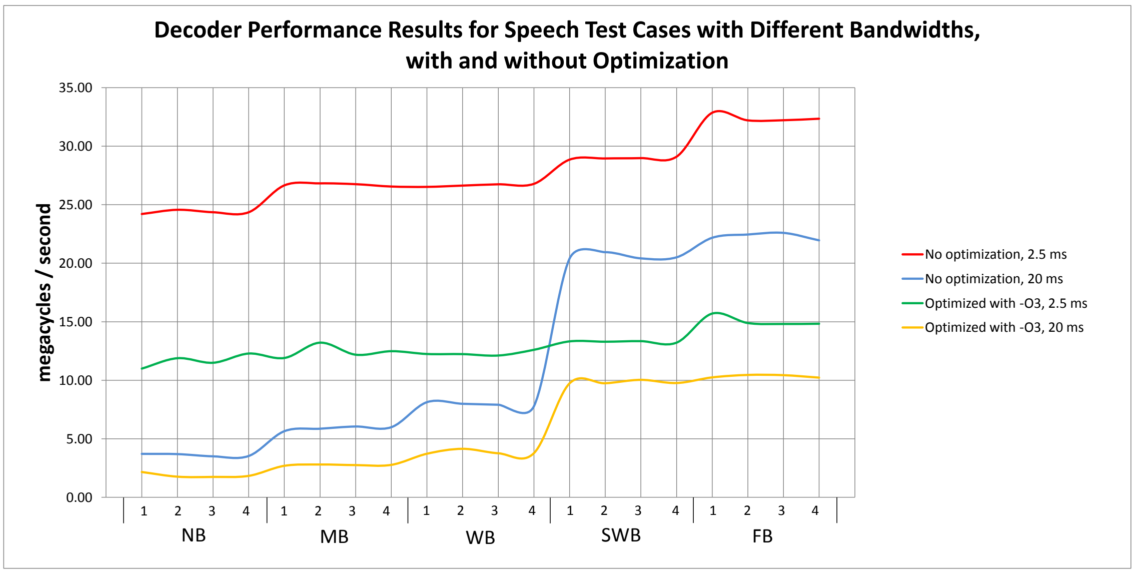
7.2.1 Compilation Optimization Test Results

The performance was measured with different optimization levels of the used compiler, GCC, with WB speech test cases for encoder and speech test cases with two different frame lengths for the decoder. The used optimization levels were none and optimization level -O3, which provides the highest optimization in compilation.

In Figure 16a the performance measurement results with two different optimization levels are shown for the encoder. The y-axis denotes the *megacycles / second* and the x-axis denotes the testcase under execution. The results denote the average performances of all the frames for each test case. Figure 16b presents the compilation performance comparison for the decoder with two different frame lengths, 2.5 ms and 20 ms. The x-axis denotes the test case under execution with four speech test cases presented for all the sample rates supported in the Opus codec.



(a)



(b)

Figure 16: Compilation optimization performance results for the encoder (a) and decoder (b).

As can be seen in Figure 16, the optimization in the compilation stage affects significantly on the performance of the application. For the encoder, the optimized compilation approximately halves the consumed *megacycles / second* compared to non-optimized version.

For the decoder, the performance shows similar behavior, as the version with optimized compilation of the codec seems to result in half of the *megacycles / second* compared to the non-optimized one. There is a significant increment in the results for SWB and FB with the frame length 20 ms, for both optimized and non-optimized compilations.

When the input Opus packets of these test cases were observed, it could be seen that the coding mode was hybrid for SWB, and CELT for FB, whereas all the other bandwidths were coded in the SILK mode. Thus, including the CELT layer decreases the execution speed. Furthermore, the magnitude of the 2.5 ms test cases' execution time is higher with all the sample rates. The coding mode used for all of these test cases was CELT as well, because the frame length is not supported in the SILK layer.

The optimized compilation results in significant improvement in the execution speed compared to the non-optimized compilation. Therefore, the compilation optimization level -O3 was fixed for the implementation. Furthermore, the codec libraries compiled with this optimization flag were used in the further performance tests.

7.2.2 Encoder Performance with Varied Parameter Configuration

The performance tests with the varying parameter configurations were performed by measuring the execution speed for all the test cases generated for the functional tests. For the encoder the parameters were mostly fixed to the external application's needs and therefore the only varied parameter was the sample rate. Furthermore, the used parameters resulted in a situation where all the test cases were encoded with the SILK layer. Therefore, all the encoder performance results gathered for this thesis denote the performance of the this layer of the encoder.

In Figure 17 the average performance measurement results for the 20 speech test cases are illustrated for both, NB (blue) and WB (black) bandwidths. The measures were obtained by computing arithmetic means from all the frames' individual performance results of each test case. In the figure, x-axis denotes the test case under execution and y-axis the average number of *megacycles / second* of each test case.

As can be seen in Figure 17, the performance results for NB test cases are varying between 21 ... 24 *megacycles / second* with an average of 22.3625 *megacycles / second*. The performance results for WB test cases are varying between 38 ... 42 *megacycles / second* with an average of 40.0595 *megacycles / second*. Thus, a higher audio bandwidth results in a longer execution time.

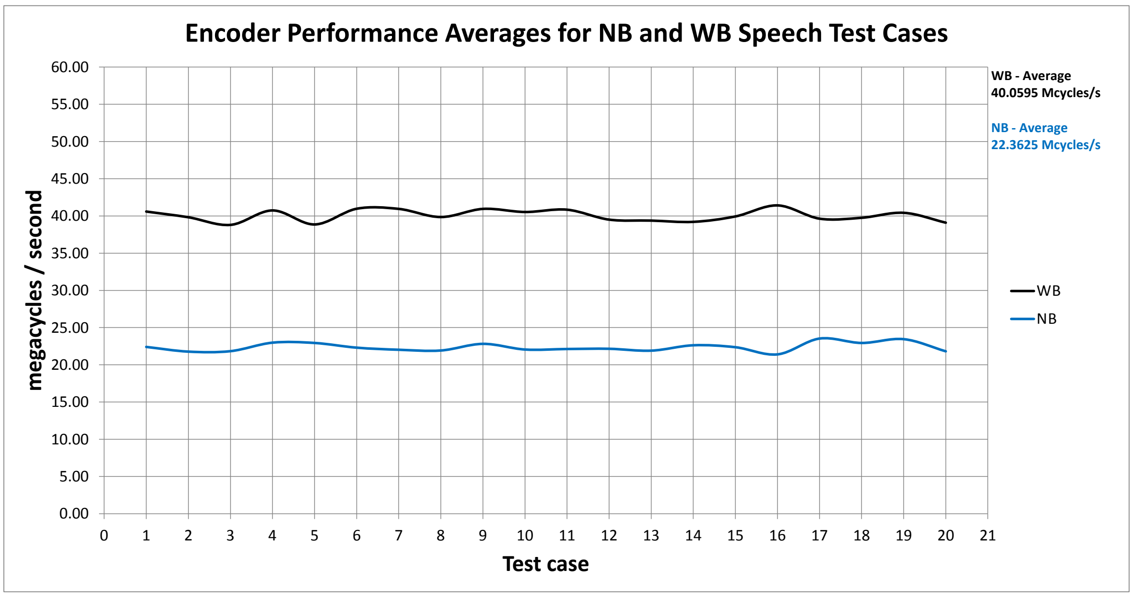


Figure 17: Encoder performance averages for the speech test cases.

The encoder performance was tested also with data containing music. The functional tests included one NB and one WB test containing music data. For the former one, average execution speed, calculated over all the frames tested, was 22.67 *megacycles / second*, and for the latter one 39.80 *megacycles / second*. Thus, the performance is approximately of the same magnitude as was measured for speech data.

As the actual execution speed depends on the clock rate of the processor, the relative performance is a more descriptive measure. It can be obtained by dividing the measured absolute values of the performance by the processor's clock rate. For NB it is approximately

$$\frac{22.3625 \;\; megacycles \;/\; second}{2400 \;\; megacycles \;/\; second} \cdot 100 \;\% \approx 0.93 \;\% \tag{10}$$

of the processor's full capacity. Respectively, for WB the relative performance is approximately

$$\frac{40.0595 \; megacycles \;/\; second}{2400 \; megacycles \;/\; second} \cdot 100 \;\% \approx 1.67 \;\% \tag{11}$$

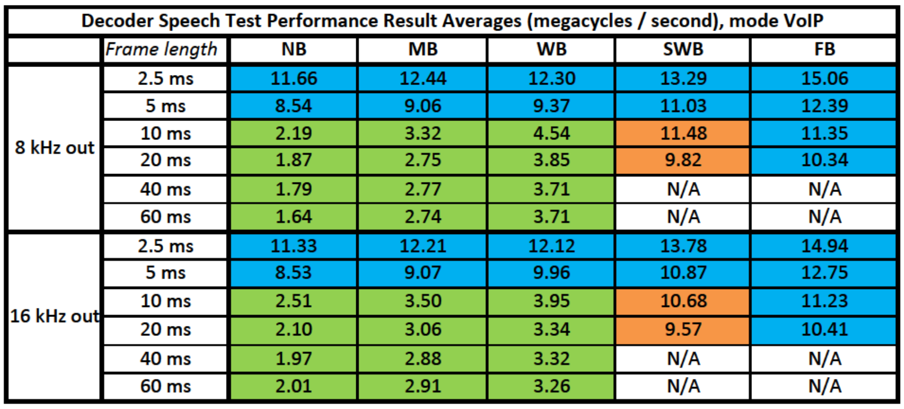
of the processor's full capacity.

7.2.3 Decoder Performance with Varied Parameter Configuration

Decoder test cases included a larger set of test cases as all the possible parameter combinations needed to be supported in the implementation. The performance results of the speech test cases with the VoIP application selection are presented in

Table 6. The results were obtained by calculating arithmetic means of all the test cases' average performance results that share the same set of parameters. The input Opus packet of each test case was reviewed to determine, whether it was encoded with the SILK layer, CELT layer, or both. In the table, green color denotes the SILK layer cases, blue the CELT layer cases, and orange the hybrid cases.

Table 6: Decoder performance for speech test cases. Green color denotes the SILK layer, blue the CELT layer, and orange the Hybrid layer.

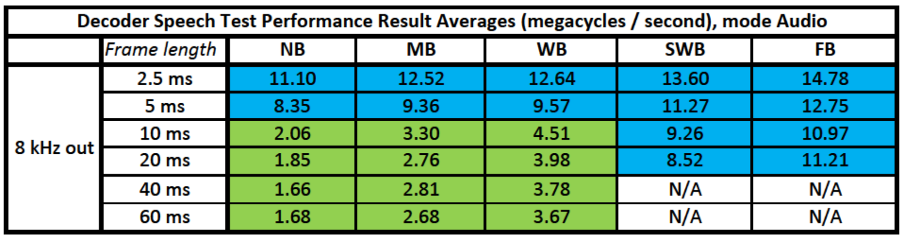


Decoder Speech Test Performance Result Averages (megacycles / second), mode VoIP						
	Frame length	NB	MB	WB	SWB	FB
8 kHz out	2.5 ms	11.66	12.44	12.30	13.29	15.06
	5 ms	8.54	9.06	9.37	11.03	12.39
	10 ms	2.19	3.32	4.54	11.48	11.35
	20 ms	1.87	2.75	3.85	9.82	10.34
	40 ms	1.79	2.77	3.71	N/A	N/A
	60 ms	1.64	2.74	3.71	N/A	N/A
16 kHz out	2.5 ms	11.33	12.21	12.12	13.78	14.94
	5 ms	8.53	9.07	9.96	10.87	12.75
	10 ms	2.51	3.50	3.95	10.68	11.23
	20 ms	2.10	3.06	3.34	9.57	10.41
	40 ms	1.97	2.88	3.32	N/A	N/A
	60 ms	2.01	2.91	3.26	N/A	N/A

As Table 6 shows, the differences in the performance between the varying output sample rates are quite minimal. Moreover, the test data containing CELT and hybrid layers results in longer execution times than the SILK layer cases. There is also a slight increment in the measured *megacycles / second* when the audio bandwidth is increased.

When the speech test cases were measured with the application selection of audio, the performance was approximately at the same magnitude as was measured for VoIP mode. However, the SWB cases with frame lengths of 20 ms and 40 ms were decoded with the CELT layer, not the hybrid layer as was the case with VoIP application selection. This reduced the execution time slightly. This is presented in Table 7.

Table 7: Decoder performance for speech data, application selection Audio. Green color denotes the SILK layer and blue the CELT layer.



Decoder Speech Test Performance Result Averages (megacycles / second), mode Audio						
	Frame length	NB	MB	WB	SWB	FB
8 kHz out	2.5 ms	11.10	12.52	12.64	13.60	14.78
	5 ms	8.35	9.36	9.57	11.27	12.75
	10 ms	2.06	3.30	4.51	9.26	10.97
	20 ms	1.85	2.76	3.98	8.52	11.21
	40 ms	1.66	2.81	3.78	N/A	N/A
	60 ms	1.68	2.68	3.67	N/A	N/A

To determine the worst case performance, the decoder was tested with an input of encoded FB stereo music with a bitrate of 510 kbit/s. The output was forced to mono 8 kHz, as the external application can operate only with a subset of the parameters supported in the Opus codec by default. In Table 8, the measured *megacycles / second* are presented for the high bitrate stereo music test case. Blue color indicates that the cases were encoded with CELT layer.

Table 8: Decoder performance for FB stereo music, bitrate 510 kbit/s. Blue color denotes the CELT layer.

Decoder High Bitrate FB Stereo Music Performance Test Results (megacycles / second), mode Audio		
	Frame length	FB
8 kHz out	2.5 ms	29.83
	5 ms	29.33
	10 ms	27.69
	20 ms	26.01

The number of the consumed cycles for high bitrate FB stereo music is significantly higher compared to the FB mono speech with recommended bitrate as was presented in Table 6. Furthermore, the shorter the frame length is, the more *megacycles / second* are consumed for decoding. The maximum value of 29.83 *megacycles / second* was measured for the shortest frame length, 2.5 ms. This yields a relative performance of approximately

$$\frac{29.83\ megacycles\ /\ second}{2400\ megacycles\ /\ second} \cdot 100\ \% \approx 1.24\ \% \tag{12}$$

of the processor's full capacity.

Restricted Low-Delay application selection was also tested in terms of performance. The speech data test case performance results are gathered in Table 9. Blue color indicates that the test cases were decoded with CELT layer. As the results show, the Restricted Low-Delay mode for short frame lengths seems to produce better performance than VoIP application selection. However, the SILK layer cases produce better performance as could be seen in the VoIP application selection performance results with the speech data (Table 6).

The remaining parameter configurations resulted approximately in the same magnitude of performance as was measured for speech test cases with the application selection VoIP (Table 6). All the remaining performance measurement results for the decoder are presented in Appendix A.

7.2.4 Complexity Parameter Performance Test Results

The complexity parameter of the encoder was tested in terms of performance. For this purpose, the WB speech test case number 20 was run with all the different

Table 9: Decoder performance for speech data, application selection Restricted Low Delay. Blue color denotes the CELT layer.

Decoder Speech Test Performance Result Averages (megacycles / second), mode Restricted Low Delay						
	Frame length	NB	MB	WB	SWB	FB
8 kHz out	2.5 ms	11.02	12.42	11.83	12.31	15.13
	5 ms	8.43	9.38	9.64	10.45	12.45
	10 ms	6.71	7.23	7.62	10.74	11.41
	20 ms	5.94	6.60	6.81	9.41	10.84
	40 ms	5.96	6.91	7.02	N/A	N/A
	60 ms	6.45	6.81	6.87	N/A	N/A

complexity values, from 0 to 10, with an increment of 1. In Figure 18 the performance results with varied complexity are illustrated for this test case. On the x-axis the frame under processing is presented, whereas the y-axis denotes the average *megacycles / second* over the whole test case.

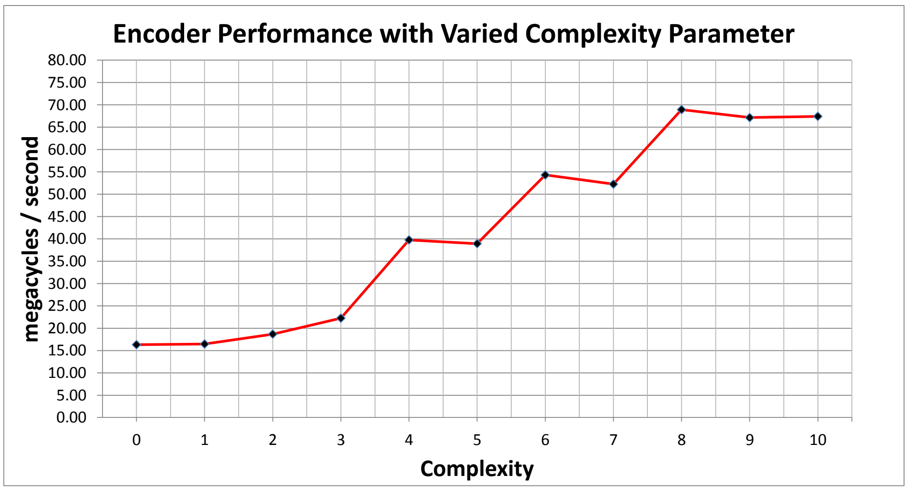


Figure 18: Encoder performance for speech test case 20 with varied complexity parameter.

As can be seen in the figure, the execution time increases nearly directly proportionally to the complexity parameter. The initially set complexity level of 5 does not seem to demand more execution time than the complexity level 4, whereas complexity level 3 is significantly more efficient than the initially set value. However, as could be seen in Figure 8, the complexity values that are less than 5 result in lower audio quality. On the other hand, there is no significant improvement in the quality with the complexity levels higher than 5. Therefore, the initially set complexity value of 5 was fixed for the implementation.

7.2.5 Performance Results of DSP

The same set of tests was run with an optimized DSP version of the codec with the TI TMS320TCI6486 processor. In Figure 19 the encoder speech test performance averages are presented and in Table 10 the decoder performance averages for speech test cases are presented. Green color in the decoder results denotes the SILK layer cases, blue color the CELT layer cases, and orange color the hybrid cases.

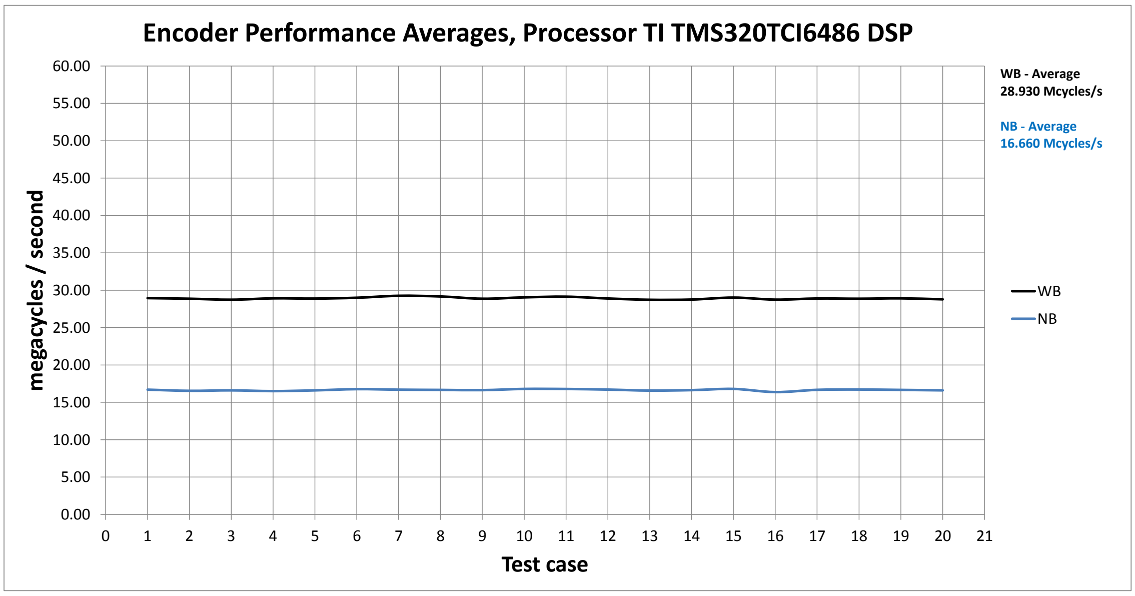


Figure 19: Encoder performance averages for speech test cases with TI TMS320TCI6486 DSP.

Table 10: Decoder performance speech test cases, with TI TMS320TCI6486 DSP. Green color denotes the SILK layer, blue the CELT layer, and orange the Hybrid layer.

Decoder Speech Test Performance Result Averages (megacycles / second), TI TMS320TCI6486 DSP

	Frame length	NB	MB	WB	SWB	FB
8 kHz out	2.5 ms	13.15	15.53	15.52	17.82	21.00
	5 ms	9.03	10.35	11.05	13.57	15.80
	10 ms	2.26	3.28	4.73	10.22	12.41
	20 ms	1.85	2.76	4.08	9.21	11.40
	40 ms	1.82	2.74	4.06	N/A	N/A
	60 ms	1.79	2.72	4.02	N/A	N/A
16 kHz out	2.5 ms	13.16	15.53	15.52	17.82	21.00
	5 ms	9.04	10.35	11.05	13.57	15.80
	10 ms	2.36	3.28	4.07	9.54	12.40
	20 ms	1.95	2.77	3.43	8.55	11.40
	40 ms	1.93	2.74	3.41	N/A	N/A
	60 ms	1.90	2.72	3.37	N/A	N/A

As the results of the encoder performance show, the speech test cases result in an average of 16.660 *megacycles / second* for NB and 28.930 *megacycles / second* for WB. The decoder results show, similarly to x86 results, no difference in performance when comparing the two different output sample rates. Furthermore, the SILK test cases result in a better performance than the CELT cases or hybrid cases.

In addition to speech tests, also the high bitrate FB stereo music test case was tested with the DSP version of the implemented codec. In Table 11 results for this test case are presented. Blue color denotes the CELT layer. The results are higher than the decoder's speech test case results encoded with the recommended bitrates. The average number of *megacycles / second* that the decoding procedure consumes with the DSP for the high bitrate FB stereo data, is 37.85 *megacycles / second*.

Table 11: Decoder performance for high bitrate FB music data, with TI TMS320TCI6486 DSP. Blue color denotes the CELT layer.

High Bitrate FB Stereo Music Performance Test Results (megacycles / second), TI TMS320TCI6486 DSP		
	Frame length	**FB**
8 kHz out	2.5 ms	41.30
	5 ms	39.13
	10 ms	36.66
	20 ms	34.32

The relative performance of the DSP can be calculated when the clock rate, 625 MHz, is known [89]. Encoding of NB speech could be performed with

$$\frac{16.660 \; megacycles \; / \; second}{625 \; megacycles \; / \; second} \cdot 100 \;\% \approx 2.67 \;\% \tag{13}$$

of the processor's full capacity. Respectively, WB speech could consumed the processor with

$$\frac{28.930 \; megacycles \; / \; second}{625 \; megacycles \; / \; second} \cdot 100 \;\% \approx 4.63 \;\% \tag{14}$$

load. For the decoder, the most demanding test case of FB stereo music took

$$\frac{37.85 \; megacycles \; / \; second}{625 \; megacycles \; / \; second} \cdot 100 \;\% = 6.06 \;\% \tag{15}$$

of the processor's full capacity.

7.2.6 Performance Evaluation and Comparison

The encoder resulted in a different magnitude of performance depending on the input sample rate of the audio. The reason for this is related to the higher number of samples when higher sample rate is used. This yields longer algorithmic delays when sample-based processing, e.g., iteration over a whole frame, is performed within the codec. Similar phenomenon was present in the decoder, only with smaller relative increment between the different sample rates. On the other hand, the higher sample rates were also encoded with higher bitrate, which may partly also increase the processing load.

The decoder's performance showed significant differences between the SILK and CELT layers with SILK performing faster in all the test cases. This is due to different sample rates. The SILK operates internally always with the sample rate of the incoming signal, whereas the CELT layer is handled internally with the sample rate of 48 kHz regardless of the input audio's bandwidth [4]. Furthermore, the decoder tests that were coded with both of the layers, thus with the hybrid mode, were executed slower than the corresponding test cases that were coded with the CELT only. This mode includes both of the layers and thus the procedure is computationally more demanding.

Varying frame length affects on the performance as could be seen in the decoder performance tests. Shorter frame lengths are computationally more demanding than the long ones. According to Opus standard [4] the coding is more efficient on longer frame lengths up to 20 ms. Thus, this behavior is an internal feature of the codec.

The most demanding test case of the decoder contained FB stereo music with maximum possible bitrate. The bitrate affects on the quality and thus demands more computation. On the other hand, stereo data contains the double amount of samples compared to mono and thus, the two separate channels introduce the same phenomenon as was present in the increasing sample rate.

When comparing the absolute values of the performances between the two tested processors, it can be seen that the DSP consumes less *megacycles / second* when encoding. Furthermore, the x86 performance results include slightly more fluctuation compared to the DSP ones. This is due to the timestamp counter, which is system wide in the x86 test environment. As a result, an occasional system interrupt that was run on the same core of the x86 as the test process, increased elapsed cycles of the counter. However, these interrupts were rare and did not include significant bias to the measured results. The cycles needed for decoding were approximately at the same magnitude with of the both processors apart from the short frame length test cases. With the 2.5 ms and 5 ms frame lengths the x86 resulted in less cycles than the DSP.

Comparison of the relative performances between the two processors provides a better picture of the processor's capability. Theoretically, in the real application, there might be one encoding and one decoding instance processed simultaneously. This, so called full-duplex case consumes for the most demanding parameter set $1.67\ \% + 1.24\ \% = 2.91\ \%$ of the x86 processor's full capacity. Respectively, the DSP uses $4.63\ \% + 6.06\ \% = 10.69\ \%$ of its full capacity when processing full-duplex coding with the Opus codec. Thus, when comparing the relative performance between the two tested processors, the x86 outperforms the DSP.

Although there is always an external application that consumes the processor capacity, a rough estimation on how many simultaneous full-duplex coding instances can be performed on a single processor core can be computed with the relative processor load measurements. With x86 processor, approximately

$$\frac{100\ \%}{2.91\ \%} \approx 34 \tag{16}$$

simultaneous full-duplex coding instances can be performed per core. Respectively, with one TI TMS320TCI6486 DSP core approximately

$$\frac{100\ \%}{10.69\ \%} \approx 9 \tag{17}$$

simultaneous full-duplex coding instances can be performed.

8 Discussion and Conclusions

The study performed for this thesis aimed to evaluate a modern hybrid codec called Opus when implemented on a virtualized core network function with an evaluation of the computational performance. The evaluation consisted of specifying the requirements, integrating the codec, test the functionality and performance, as well as comparing the latter one to an optimized DSP version of the codec.

The results of performance measurements showed that the standard C-implementation of the Opus codec performed sufficiently for the purpose used. In addition, when comparing to an optimized implementation of a DSP, the performance of the x86 showed very promising results. Although, the absolute number of *megacycles / second* was slightly higher for the x86 version when encoding, the relative performance was better on the used GPP. This was a remarkable result since when the similar comparison was performed to another codec, AMR-WB, there was a difference of a decade between the number of *megacycles / second* with x86 performing slower. This can be understood by taking the history of the codec's development into consideration, as the Opus has been from the beginning targeted to the GPPs of the desktops. Comprehensive code profiling was not performed as a part of this thesis. However, the preliminary profiling showed that the AMR-WB codec consumed majority of the execution time performing the MAC operation on the x86, whereas Opus' algorithms included this operation more efficiently.

The performance results obtained in this thesis denote purely the execution speed, whereas the power consumption or price of the processor are not taken into consideration. E.g., compared to the power rating of TI TMS320C64x processor family, the used x86 processor consumes power of almost a decade more [70, 90]. Furthermore, the DSP of the same family can be purchased with 1/4 ... 1/2 the price of the used x86 [70, 91]. To conclude the performance comparison in this thesis, for Opus codec the tested x86 provides sufficient performance in terms of execution speed. Thus, the execution time will not be the limiting factor when implementing the network functions for generalized hardware.

There have been newer releases of the codec since the standardization, which have included improvements in the performance as well as additional features. For instance, as of version 1.1 a built-in application selection can be used so that the codec detects the information automatically and determines whether to code in VoIP, Audio or Restricted Low-Delay mode [87]. This would be especially useful in the application to which the codec was implemented, as there is no way to detect whether the input data is audio or speech. In addition, newer releases have also included further optimization of the codec for different GPPs, e.g., x86, and ARM [87].

In addition to the Opus codec, other efforts to provide high-quality and versatile codecs are on-going. 3GPP's EVS combines their previous Adaptive Multi Rate (AMR) coding technologies, and a transform coding technology into one framework to provide versatility as well as enhanced quality and coding efficiency [92]. The supported sample rates start from commonly used 8 kHz and span up to 48 kHz, which allows better quality for information containing both speech and music. Also, the codec utilizes different methods to provide robustness to packet loss and delay jitter and thus leads to optimized IP behavior. The difference compared to Opus codec is that EVS is designed to be run natively on the mobile networks whereas Opus was targeted to VoIP communication [4, 92].

For further study, a comparative test between this new codec and the implemented Opus codec would provide valuable information on how another state-of-art high quality codec fits into the virtualized network. The performance study introduced in this thesis could be also conducted to other GPPs, such as ARM, to provide information on how the other generic processor architectures perform the signal processing. Furthermore, comparison to a newer DSP, such as TI TMS320C66x family, could be performed so that the DSP would also be state-of-art technology. In addition, it would be interesting to measure how the newer releases of the Opus codec perform in the general hardware, as there have been efforts to optimize the algorithms for GPPs. Furthermore, the use of processor vendors' own compilers, as well as tuning the compilation to take advantage of the newest instruction sets of the used processor, could improve the execution speed even further.

References

[1] I. Grigorik, *High Performance Browser Networking.* O'Reilly Media, Inc., 2013.

[2] A. Leon-Garcia and I. Widjaja, *Communication Networks.* McGraw-Hill, Inc., 2003.

[3] P. Bosch, A. Duminuco, F. Pianese, and T. Wood, "Telco Clouds and Virtual Telco: Consolidation, Convergence, and Beyond," in *IEEE International Workshop on Broadband Convergence Networks*, May 2011, pp. 982–988.

[4] IETF, "Definition of the Opus Audio Codec," Internet Engineering Task Force, RFC 6716, 2012.

[5] S. Akhtar, "2G-4G Networks: Evolution of Technologies, Standards, and Deployment," *Encyclopedia of Multimedia Technology and Networking*, 2009, http://faculty.uaeu.ac.ae/s.akhtar/EncyPaper04.pdf, accessed: 29.10.2014.

[6] 3GPP, "Technical Specification Group Core Network and Terminals; IP Multimedia Subsystem (IMS) Application Level Gateway (IMS-ALG) - IMS Access Gateway (IMS-AGW) Interface: Procedures Descriptions," ETSI, TS 23.334 V12.4.0, 2014.

[7] 3GPP, "About 3GPP," http://www.3gpp.org/about-3gpp/about-3gpp, accessed: 12.08.2014.

[8] IETF, "Media Gateway Control Protocol Architecture and Requirements," Network Working Group, RFC 2805, 2000.

[9] 3GPP, "Technical Specification Group Core Network and Terminals; Media Gateway Controller (MGC) - Media Gateway (MGW) Interface; Stage 3," ETSI, TS 29.232 V12.0.0, 06 2014.

[10] IETF, "RTP: A Transport Protocol for Real-Time Applications," Internet Engineering Task Force, RFC 3550, 2003.

[11] IETF, "SDP: Session Description Protocol," Internet Engineering Task Force, RFC 4556, 2006.

[12] T. D. Rossing, F. R. Moore, and P. A. Wheeler, *The Science of Sound.* Addison-Wesley, CA, 2002, vol. 3.

[13] M. Karjalainen, *Kommunikaatioakustiikka*, 2nd ed. Espoo: Helsinki University of Technology, 2008.

[14] W. He and T. Qu, "Audio Lossless Coding/Decoding Method Using Basis Pursuit Algorithm," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, CA: Vancouver, May 2013, pp. 552–555.

[15] T. Painter and A. Spanias, "Perceptual Coding of Digital Audio," *Proceedings of the IEEE*, vol. 88, no. 4, pp. 451–515, 2000.

[16] T. Liebchen, M. Purat, and P. Noll, "Improved Lossless Transform Coding of Audio Signals," *Impulse und Antworten*, pp. 159–170, 1999.

[17] K. Brandenburg and G. Stoll, "ISO/MPEG-1 Audio: A Generic Standard for Coding of High-quality Digital Audio," *Journal of the Audio Engineering Society*, vol. 42, no. 10, pp. 780–792, 1994.

[18] M. Morf, B. Dickinson, T. Kailath, and A. Vieira, "Efficient Solution of Covariance Equations for Linear Prediction," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 25, no. 5, pp. 429–433, Oct 1977.

[19] N. Bhatt and Y. Kosta, "Overall Performance Evaluation of Adaptive Multi Rate 06.90 Speech Codec Based on Code Excited Linear Prediction Algorithm Using MATLAB," *International Journal of Speech Technology*, vol. 15, no. 2, pp. 119–129, 2012.

[20] 3GPP, "European Digital Cellular Telecommunications System; Half Rate Speech Part 2: Half Rate Speech Transcoding (GSM 06.20)," ETSI, TS 300 581-2, 1995.

[21] 3GPP, "Digital Cellular Telecommunications System (Phase 2+); Full Rate Speech; Transcoding (GSM 06.10 Version 5.1.1)," ETSI, TS 300 961 V5.1.1, 1998.

[22] C. Cellier, P. Chenes, and M. Rossi, "Lossless Audio Bit Rate Reduction," in *Audio Engineering Society Conference: UK 9th Conference: Managing the Bit Budget (MBB)*. Audio Engineering Society, UK, 1994.

[23] IETF, "Coding Tools for a Next Generation Video Codec," Internet Engineering Task Force, draft-terriberry-codingtools-01, 2014.

[24] J.-M. Valin, T. B. Terriberry, C. Montgomery, and G. Maxwell, "A High-quality Speech and Audio Codec With Less Than 10 ms Delay," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 18, no. 1, pp. 58–67, 2010.

[25] J. Popp, M. Neuendorf, H. Fuchs, C. Forster, and A. Heuberger, "Recent Advances In Broadcast Audio Coding," in *IEEE International Symposium on Broadband Multimedia Systems and Broadcasting (BMSB)*, June 2013, pp. 1–5.

[26] 3GPP, "Digital Cellular Telecommunications System (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Mandatory speech CODEC speech processing functions; AMR speech Codec; General description (3GPP TS 26.071 version 12.0.0 Release 12)," ETSI, TS 126 071 V12.0.0, 2014.

[27] 3GPP, "Digital Cellular Telecommunications System (Phase 2+); Universal Mobile Telecommunications System (UMTS); LTE; Speech codec speech processing functions; Adaptive Multi-Rate - Wideband (AMR-WB) speech codec; General description (3GPP TS 26.171 version 12.0.0 Release 12)," ETSI, TS 126 171 V12.0.0, 2014.

[28] ITU-T, "Pulse Code Modulation of Voice Frequencies," ITU, ITU-T Recommendation G.711, 1993.

[29] ITU-T, "Low-complexity coding at 24 and 32 kbit/s for hands-free operation in systems with low frame loss," ITU, ITU-T Recommendation G.722.1, 2005.

[30] Xiph.org, "The Xiph Open Source Community," http://www.xiph.org/, accessed: 27.10.2014.

[31] IETF, "SILK Speech Codec," Internet Engineering Task Force, draft-vos-silk-02, 2010.

[32] G. I. Solutions, "Global IP Solutions," http://www.gipscorp.com/, accessed: 27.10.2014.

[33] IETF, "RTP Payload Format for the Speex Codec," Internet Engineering Task Force, RFC 5574, 2009.

[34] IETF, "Constrained-Energy Lapped Transform (CELT) Codec," Internet Engineering Task Force, draft-valin-celt-codec-02, 2010.

[35] M. Goudarzi, L. Sun, and E. Ifeachor, "Modelling Speech Quality for NB and WB SILK Codec for VoIP Applications," in *Proc. 5th International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST).* IEEE, UK: Cardiff, 2011, pp. 42–47.

[36] IETF, "Internet Low Bit Rate Codec (iLBC)," The Internet Society, RFC 3951, 2004.

[37] H.-H. Rao, Y.-B. Lin, and S.-L. Cho, "iGSM: VoIP Service for Mobile Networks," *Communications Magazine, IEEE*, vol. 38, no. 4, pp. 62–69, Apr 2000.

[38] R. Cuny and A. Lakaniemi, "VoIP in 3G Networks: an End-to-End Quality of Service Analysis," in *Vehicular Technology Conference, 2003. VTC 2003-Spring. The 57th IEEE Semiannual*, vol. 2, April 2003, pp. 930–934 vol.2.

[39] A. Rämö and H. Toukomaa, "Voice Quality Characterization of IETF Opus Codec," in *Proc. Interspeech*, IT: Florence, 2011, pp. 2541–2544.

[40] V. Gupta, M. Onufry, H. Suyderhoud, and K. Virupaksha, "Variable Bit Rate Speech Codec With Backward-type Prediction and Quantization," Jun. 14 1988, US Patent 4,751,736.

[41] IETF, "RTP Payload Format for Opus Speech and Audio Codec," Internet Engineering Task Force, draft-ietf-payload-rtp-opus-02, 2014.

[42] Xiph.Org, "Opus Audio Codec (RFC 6716): API and Operations Manual," http://www.opus-codec.org/docs/html_api-1.1.0/index.html, accessed: 29.10.2014.

[43] M.-K. Lee and H.-G. Kang, "Speech Quality Estimation of Voice over Internet Protocol Codec Using a Packet Loss Impairment Model," *Journal of the Acoustical Society of America*, vol. 134, no. 5, pp. EL438–EL444, 2013.

[44] A. Rämö and H. Toukomaa, "Voice Quality Evaluation of Recent Open Source Codecs," in *Proc. Interspeech*, JP: Makuhari, 2010, pp. 2390–2393.

[45] K. Vos, K. V. Sørensen, S. S. Jensen, and J.-M. Valin, "Voice Coding with Opus," in *Audio Engineering Society Convention 135*. Audio Engineering Society, US: New York, 2013.

[46] K. Vos, "A Fast Implementation of Burg's Method," http://www.3gpp.org/about-3gpp/about-3gpp, 2013, accessed: 15.11.2014.

[47] P. Kabal and R. Ramachandran, "The Computation of Line Spectral Frequencies Using Chebyshev Polynomials," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 34, no. 6, pp. 1419–1426, Dec 1986.

[48] J.-M. Valin, G. Maxwell, T. B. Terriberry, and K. Vos, "High-Quality, Low-Delay Music Coding in the Opus Codec," in *Audio Engineering Society Convention 135*. Audio Engineering Society, US: New York, 2013.

[49] Xiph.Org, "The CELT Ultra-low Delay Audio Codec," http://celt-codec.org/, accessed: 12.08.2014.

[50] Xiph.Org Foundation, "Vorbis I Specification," http://www.xiph.org/vorbis/doc/Vorbis_I_spec.html, accessed: 12.08.2014.

[51] J.-M. Valin, T. B. Terriberry, and G. Maxwell, "A Full-bandwidth Audio Codec With Low Complexity and Very Low Delay," in *Proc. European Signal Processing Conference (EUSIPCO)*, UK: Glasgow, 2009.

[52] T. B. Terriberry and J.-M. Valin, "Method and System for Two-step Spreading for Tonal Artifact Avoidance in Audio Coding," 2012, US Patent App. 13/414,418.

[53] H. Malvar, "Lapped Transforms for Efficient Transform/Subband Coding," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 38, no. 6, pp. 969–978, Jun 1990.

[54] C.-H. Chen, B.-D. Liu, and J.-F. Yang, "Recursive Architectures for Realizing Modified Discrete Cosine Transform and Its Inverse," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 50, no. 1, pp. 38–45, Jan 2003.

[55] S. Lee and I. Lee, "A Low-Delay MDCT/IMDCT," *ETRI Journal*, vol. 35, no. 5, 2013.

[56] S. K. Mitra and Y. Kuo, *Digital Signal Processing: a Computer-based Approach.* McGraw-Hill New York, 2006, vol. 2.

[57] ITU-T, "Perceptual Objective Listening Quality Assessment," ITU, ITU-T Recommendation P.863, 2014.

[58] ITU-T, "Subjective Video Quality Assessment Methods for Multimedia Applications," ITU, ITU-T Recommendation P.910, 2008.

[59] R. Chen, T. Terriberry, J. Skoglund, G. Maxwell, and H. T. M. Nguyet, "Opus Testing," https://www.ietf.org/proceedings/80/slides/codec-4.pdf, 2011, accessed: 29.10.2014.

[60] B. Goode, "Voice over Internet Protocol (VoIP)," *Proceedings of the IEEE*, vol. 90, no. 9, pp. 1495–1517, 2002.

[61] S. Guha and N. Daswani, "An Experimental Study of the Skype Peer-to-Peer VoIP System," Cornell University, Tech. Rep., 2005.

[62] Google Inc., "WebRTC," http://www.webrtc.org/, accessed: 29.10.2014.

[63] PCI Industrial Computers Manufacturers Group, "AdvancedTCA Short Form Specification," PICMG, PICMG 3.0, 2003.

[64] A. Karlsson and B. Martin, "ATCA: Its Performance and Application for Real Time Systems," *IEEE Transactions on Nuclear Science*, vol. 53, no. 3, pp. 688–693, June 2006.

[65] P. Lapsley, J. Bier, A. Shoham, and E. A. Lee, *DSP Processor Fundamentals: Architectures and Features. Berkeley Design Technology.* Inc, Fremont, CA: IEEE Press, 1997.

[66] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper, "Virtual Network Computing," *IEEE Internet Computing*, vol. 2, no. 1, pp. 33–38, Jan 1998.

[67] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: the Hardware/Software Interface*, 4th ed. Newnes, 2009.

[68] K. R. Irvine, *Assembly Language for x86 Processors.* Prentice Hall, 2011.

[69] Texas Instruments, "TMS320C66x DSP CPU and Instruction Set Reference Guide," http://www.ti.com/lit/ug/sprugh7/sprugh7.pdf, accessed: 18.08.2014.

[70] Intel, "Intel® Core^{TM} i7-4700MQ Processor (6M Cache, up to 3.40 GHz)," http://ark.intel.com/products/75117/Intel-Core-i7-4700MQ-Processor-6M-Cache-up-to-3_40-GHz, accessed: 08.11.2014.

[71] Texas Instruments, "TMS320C6672 Multicore Fixed and Floating-Point Digital Signal Processor," http://www.ti.com.cn/cn/lit/ds/symlink/tms320c6672.pdf, accessed: 18.08.2014.

[72] Berkeley Design Technology, Inc., "The BDTImark2000: A Summary Measure of Signal Processing Speed White Paper," http://www.bdti.com/MyBDTI/pubs/BDTImark2000.pdf, accessed: 18.08.2014.

[73] K. Williston, "Microprocessors vs. DSPs: Fundamentals and Distinctions," in *Conference on Embedded Systems*, 2005, accessed: 18.08.2014. [Online]. Available: http://www.bdti.com/MyBDTI/pubs/050307ESC_MPUs_vs_DSPs.pdf

[74] Texas Instruments, "TMS320C6457 Communications Infrastructure Digital Signal Processor," http://www.ti.com/lit/ds/symlink/tms320c6457.pdf, accessed: 29.10.2014.

[75] Intel, "Intel Xeon Phi Coprocessor Instruction Set Architecture Reference Manual," https://software.intel.com/sites/default/files/forum/278102/327364001en.pdf, accessed: 18.08.2014.

[76] Intel, "AVX2 Migrating from SSE2 Vector Operations to AVX2 Vector Operations," https://software.intel.com/en-us/tags/18026, accessed: 08.11.2014.

[77] J. Hofmann, J. Treibig, G. Hager, and G. Wellein, "Comparing the Performance of Different x86 SIMD Instruction Sets for a Medical Imaging Application On Modern Multi-and Manycore Chips," in *Proc. Workshop on Programming Models for SIMD/Vector Processing*. ACM, US: Orlando, 2014, pp. 57–64.

[78] B. Toll, E. Ould-Ahmed-Vall, and I. Albrekth, "Intel Advanced Vector Extensions 2 and Software Optimization," in *Intel Developers Forum - IDF13*. Intel Corporation, 2013.

[79] Intel, "Intel®Core^{TM}i7-4960X Processor Extreme Edition (15M Cache, up to 4.00 GHz)," http://ark.intel.com/products/77779/Intel-Core-i7-4960X-Processor-Extreme-Edition-15M-Cache-up-to-4_00-GHz, accessed: 12.08.2014.

[80] "Speed Scores for Floating-Point Packaged Processors," http://www.bdti.com/MyBDTI/bdtimark/chip_float_scores.pdf, Berkeley Design Technology, Inc., accessed: 12.08.2014.

[81] Intel, "Intel® Pentium® III Processor 1.00 GHz, 256K Cache, 133 MHz FSB," http://ark.intel.com/products/27529/Intel-Pentium-III-Processor-1_00-GHz-256K-Cache-133-MHz-FSB, accessed: 29.10.2014.

[82] Intel, "Intel® C++ Compiler XE 13.1 User and Reference Guide," https://software.intel.com/sites/products/documentation/doclib/iss/2013/compiler/cpp-lin/, accessed: 29.10.2014.

[83] Texas Instruments, "TMS320C6000 Optimizing Compiler v7.6," http://www.ti.com/lit/ug/spru187v/spru187v.pdf, accessed: 29.10.2014.

[84] Free Software Foundation, Inc., "Using the GNU Compiler Collection, For GCC Version 4.9.1," https://gcc.gnu.org/onlinedocs/gcc-4.9.1/gcc.pdf, accessed: 29.10.2014.

[85] IETF, "A Standard for the Transmission of IP Datagrams over Ethernet Networks," Internet Engineering Task Force, RFC 894, 1984.

[86] ITU-T, "Software Tools for Speech and Audio Coding Standardization," ITU, ITU-T Recommendation G.191, 2010.

[87] Xiph.Org, "Opus Interactive Audio Codec," http://www.opus-codec.org/, accessed: 30.10.2014.

[88] Intel, "How to Benchmark Code Execution Times on Intel® IA-32 and IA-64 Instruction Set Architectures," http://www.intel.com/content/dam/www/public/us/en/documents/white-papers/ia-32-ia-64-benchmark-code-execution-paper.pdf, accessed: 29.10.2014.

[89] Texas Instruments, "TMS320TCI6486 Communications Infrastructure Digital Signal Processor," http://www.ti.com/lit/ds/symlink/tms320tci6486.pdf, accessed: 08.11.2014.

[90] Berkeley Design Technology, Inc., "Speed per Milliwatt Ratios for Fixed-Point Packaged Processors," http://www.bdti.com/MyBDTI/bdtimark/chip_fixed_power_scores.pdf, accessed: 08.11.2014.

[91] Texas Instruments, "TMS320C6472 Fixed-Point Digital Signal Processor," http://www.ti.com/product/TMS320C6472/samplebuy, accessed: 08.11.2014.

[92] 3GPP, "Codec for Enhanced Voice Services," 3GPP, SP 140151, 2014.

A Decoder Parameter Configuration Performance Results

In this section all the remaining performance measurement results that were not presented in Section 7.2.3, are gathered. The SILK layer is denoted with green color, the CELT layer with blue color, and hybrid (SILK and CELT) with orange color.

Table A1: Decoder performance for music data, application selection VoIP.

Decoder Music Test Performance Results (megacycles / second), mode VoIP						
	Frame length	NB	MB	WB	SWB	FB
8 kHz out	2.5 ms	10.94	12.04	12.03	13.16	14.75
	5 ms	8.21	8.95	9.40	10.90	11.99
	10 ms	2.08	3.32	4.49	10.90	10.04
	20 ms	1.64	2.79	3.79	9.84	9.54
	40 ms	1.69	2.73	3.69	N/A	N/A
	60 ms	1.59	2.70	3.71	N/A	N/A
16 kHz out	2.5 ms	10.91	12.14	11.94	13.07	14.63
	5 ms	8.25	8.88	9.35	10.87	12.11
	10 ms	2.41	3.35	3.85	10.38	10.06
	20 ms	1.99	2.92	3.24	9.36	9.59
	40 ms	1.92	2.90	3.15	N/A	N/A
	60 ms	1.93	2.85	3.22	N/A	N/A

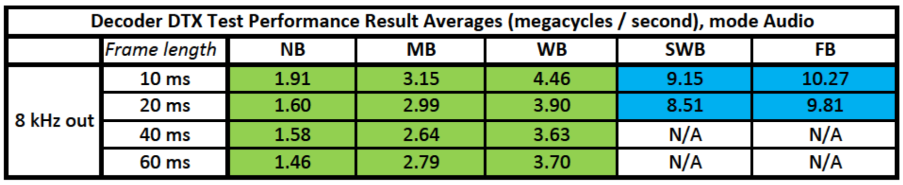
Table A2: Decoder performance for speech data with DTX, application selection VoIP.

Decoder DTX Test Performance Result Averages (megacycles / second), mode VoIP						
	Frame length	NB	MB	WB	SWB	FB
8 kHz out	10 ms	1.92	2.92	4.45	10.51	11.64
	20 ms	1.59	2.38	3.79	9.16	10.57
	40 ms	1.49	2.32	3.69	N/A	N/A
	60 ms	1.47	2.37	3.74	N/A	N/A

Table A3: Decoder performance for music data, application selection Audio.

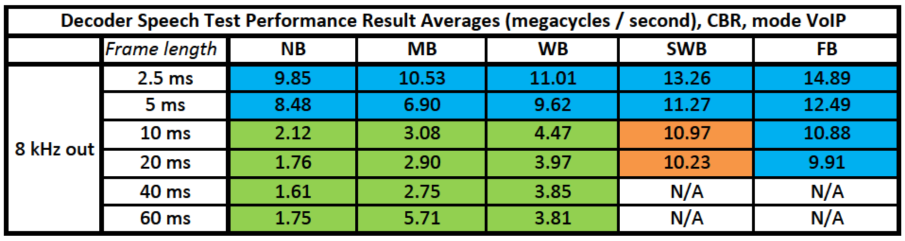
Decoder Music Test Performance Results (megacycles / second), mode Audio						
	Frame length	NB	MB	WB	SWB	FB
8 kHz out	2.5 ms	10.97	12.17	12.03	13.49	14.82
	5 ms	8.42	8.92	9.41	11.04	12.11
	10 ms	2.07	3.23	4.56	8.81	10.06
	20 ms	1.65	2.82	3.83	8.06	9.50
	40 ms	1.69	2.70	3.77	N/A	N/A
	60 ms	1.60	2.66	3.64	N/A	N/A

Table A4: Decoder performance for speech data with DTX, application selection Audio.



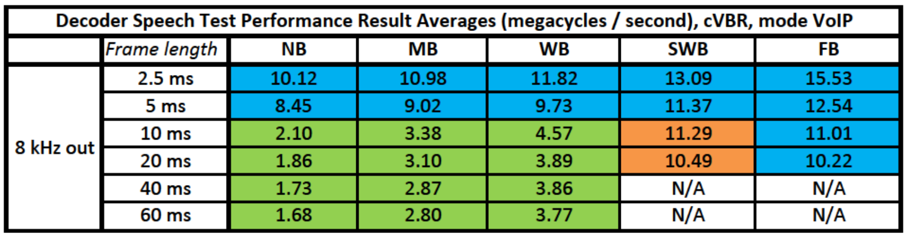
Decoder DTX Test Performance Result Averages (megacycles / second), mode Audio						
	Frame length	NB	MB	WB	SWB	FB
8 kHz out	10 ms	1.91	3.15	4.46	9.15	10.27
	20 ms	1.60	2.99	3.90	8.51	9.81
	40 ms	1.58	2.64	3.63	N/A	N/A
	60 ms	1.46	2.79	3.70	N/A	N/A

Table A5: Decoder performance for speech data with CBR enabled, application selection VoIP.



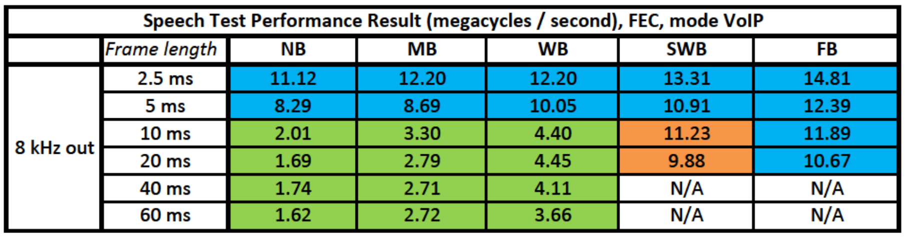
Decoder Speech Test Performance Result Averages (megacycles / second), CBR, mode VoIP						
	Frame length	NB	MB	WB	SWB	FB
8 kHz out	2.5 ms	9.85	10.53	11.01	13.26	14.89
	5 ms	8.48	6.90	9.62	11.27	12.49
	10 ms	2.12	3.08	4.47	10.97	10.88
	20 ms	1.76	2.90	3.97	10.23	9.91
	40 ms	1.61	2.75	3.85	N/A	N/A
	60 ms	1.75	5.71	3.81	N/A	N/A

Table A6: Decoder performance for speech data with cVBR enabled, application selection VoIP.



Decoder Speech Test Performance Result Averages (megacycles / second), cVBR, mode VoIP						
	Frame length	NB	MB	WB	SWB	FB
8 kHz out	2.5 ms	10.12	10.98	11.82	13.09	15.53
	5 ms	8.45	9.02	9.73	11.37	12.54
	10 ms	2.10	3.38	4.57	11.29	11.01
	20 ms	1.86	3.10	3.89	10.49	10.22
	40 ms	1.73	2.87	3.86	N/A	N/A
	60 ms	1.68	2.80	3.77	N/A	N/A

Table A7: Decoder performance for speech data with FEC enabled, application selection VoIP.



Speech Test Performance Result (megacycles / second), FEC, mode VoIP						
	Frame length	NB	MB	WB	SWB	FB
8 kHz out	2.5 ms	11.12	12.20	12.20	13.31	14.81
	5 ms	8.29	8.69	10.05	10.91	12.39
	10 ms	2.01	3.30	4.40	11.23	11.89
	20 ms	1.69	2.79	4.45	9.88	10.67
	40 ms	1.74	2.71	4.11	N/A	N/A
	60 ms	1.62	2.72	3.66	N/A	N/A

Table A8: Decoder performance for music data, application selection Restricted Low Delay.

Decoder Music Test Performance Result (megacycles / second), mode Restricted Low Delay						
	Frame length	**NB**	**MB**	**WB**	**SWB**	**FB**
8 kHz out	**2.5 ms**	10.78	12.05	11.80	13.20	14.93
	5 ms	8.30	8.95	9.36	10.84	12.08
	10 ms	6.70	7.09	7.38	8.64	10.15
	20 ms	6.03	6.40	6.69	8.00	9.60
	40 ms	5.77	6.26	6.51	N/A	N/A
	60 ms	5.71	6.15	6.53	N/A	N/A

Table A9: Decoder performance for speech data with six 20 ms frames in a packet, mode VoIP.

Decoder Performance, Six 20 ms Speech Frames in Packet (megacycles / second), mode VoIP		
	Test case	**FB**
8 kHz out	1	2.14
	2	2.01
	3	2.31
	4	2.23