Voice Codec for Floating Point Processor

Hans Engström & Johan Ross

LiTH-ISY-EX--08/3782--SE
Linköping 2008
Voice Codec for Floating Point Processor

Master Thesis

In Electronics Design,
Dept. Of Electrical Engineering
At Linköping University

By Hans Engström & Johan Ross
Reg no: LiTH-ISY-EX--08/3782--SE

Supervisor: Johan Eilert
Examiner: Dake Liu
Linköping, 2008
Publikationens titel
Voice Codec for Floating Point Processor

Författare
Hans Engström & Johan Ross

Sammanfattning
As part of an ongoing project at the department of electrical engineering, ISY, at Linköping University, a voice decoder using floating point formats has been the focus of this master thesis. Previous work has been done developing an mp3-decoder using the floating point formats. All is expected to be implemented on a single DSP.

The ever present desire to make things smaller, more efficient and less power consuming are the main reasons for this master thesis regarding the use of a floating point format instead of the traditional integer format in a GSM codec. The idea with the low precision floating point format is to be able to reduce the size of the memory. This in turn reduces the size of the total chip area needed and also decreases the power consumption.

One main question is if this can be done with the floating point format without losing too much sound quality of the speech. When using the integer format, one can represent every value in the range depending on how many bits are being used. When using a floating point format you can represent larger values using fewer bits compared to the integer format but you lose representation of some values and have to round the values off.

From the tests that have been made with the decoder during this thesis, it has been found that the audible difference between the two formats is very small and can hardly be heard, if at all. The rounding seems to have very little effect on the quality of the sound and the implementation of the codec has succeeded in reproducing similar sound quality to the GSM standard decoder.

Nykkelord
Voice codec, floating point, GSM decoder, low precision codec, speech coding
Abstract

As part of an ongoing project at the department of electrical engineering, ISY, at Linköping University, a voice decoder using floating point formats has been the focus of this master thesis. Previous work has been done developing an mp3-decoder using the floating point formats. All is expected to be implemented on a single DSP.

The ever present desire to make things smaller, more efficient and less power consuming are the main reasons for this master thesis regarding the use of a floating point format instead of the traditional integer format in a GSM codec. The idea with the low precision floating point format is to be able to reduce the size of the memory. This in turn reduces the size of the total chip area needed and also decreases the power consumption.

One main question is if this can be done with the floating point format without losing too much sound quality of the speech. When using the integer format, one can represent every value in the range depending on how many bits are being used. When using a floating point format you can represent larger values using fewer bits compared to the integer format but you lose representation of some values and have to round the values off.

From the tests that have been made with the decoder during this thesis, it has been found that the audible difference between the two formats is very small and can hardly be heard, if at all. The rounding seems to have very little effect on the quality of the sound and the implementation of the codec has succeeded in reproducing similar sound quality to the GSM standard decoder.
# Contents

1  INTRODUCTION
   1.1  BACKGROUND ................................................................................. 1
   1.2  PURPOSE AND OBJECTIVES OF THIS WORK ................................. 1
   1.3  METHOD ....................................................................................... 2
   1.4  LIMITATIONS AND PROBLEM PRESENTATION ......................... 2
   1.5  TECHNICAL AIDS ........................................................................ 3
   1.6  MOTIVATION ................................................................................ 3
   1.7  REPORT OUTLINE......................................................................... 4

2  THEORY
   2.1  GSM SPEECH CODING ................................................................. 5
   2.2  CODECS IN THE GSM SYSTEM .................................................. 7
   2.3  SPEECH CODECS ........................................................................ 8
   2.4  A MODEL OF THE HUMAN SPEECH ......................................... 11
   2.5  FREQUENCY RANGE AND CHARACTERISTICS OF SPEECH ....... 17

3  THE FLOATING POINT FORMATS
   3.1  FLOATING POINT FORMAT ......................................................... 19
   3.2  EMULATION OF THE DSP ON PC ............................................. 20
   3.3  PRECISION AND QUANTIZATION ............................................. 21
   3.4  CONVERSION AND SCALING.................................................... 22

4  GSM FULL RATE ENCODER
   4.1  FUNCTIONAL OVERVIEW ........................................................... 25
   4.2  PREPROCESSING ....................................................................... 26
   4.3  LPC ANALYSIS ........................................................................... 27
   4.4  SHORT TERM ANALYSIS FILTERING ....................................... 29
   4.5  LONG TERM PREDICTION ......................................................... 31
   4.6  RPE ENCODING ....................................................................... 33

5  GSM FULL RATE DECODER
   5.1  THE SPEECH FRAME ................................................................. 37
   5.2  FUNCTIONAL OVERVIEW ......................................................... 38
   5.3  RPE DECODING AND LONG TERM PREDICTION ...................... 39
   5.4  LAR DECODING AND SHORT TERM SYNTHESIS FILTERING .... 40
   5.5  POSTPROCESSING .................................................................... 41

6  TESTS AND CODEC BEHAVIOUR
   6.1  CODEC IMPLEMENTATIONS ........................................................ 43
   6.2  DIFFERENT CODEC IMPLEMENTATIONS ................................... 43
   6.3  CHANGING FROM INTEGER TO FLOAT .................................... 44
   6.4  CHANGING TO LOWER PRECISION ......................................... 45
   6.5  PERFORMANCE ......................................................................... 50

7  RESULTS ......................................................................................... 51

8  FUTURE WORK .................................................................................. 53

9  REFERENCES ....................................................................................... 55

10  ABBREVIATIONS AND EXPLANATIONS ........................................ 57
1 Introduction

1.1 Background

Most existing algorithms and applications with high data throughput are intended to run on DSP’s that use integers with both high and low precision or use standardized floating point format with high precision. At the Department of Electrical Engineering (ISY) at the Linköping University, a DSP with customized low precision floating point formats is used for research on sound compression algorithms and similar areas of application.

Previous projects have examined how well suited the floating point formats are for mp3 compression and an implementation of an mp3 decoder has successfully been created. As a step in finding other possible applications, the intention with this thesis has been to examine how well compression and decoding of speech works with the limitations in precision with the floating point formats.

1.2 Purpose and objectives of this work

The main purpose of the work is to implement a functional speech decoder adapted to the floating point DSP that is used at ISY. Even though the output from the decoder numerically should be as close as possible to the output from the original decoder, it is more important to produce as good perceived sound quality as possible. One objective is also to examine the impact on the speech compression that the floating point format and the limited precision have. This includes finding out how low precision that can be used before the sound quality starts to deteriorate and eventually become unintelligible.

Since the reason for using low precision and the floating point format is to keep the memory usage and power consumption down, it is reasonable to try to keep all resources needed for the speech codec as low as possible. This means that a fairly simple codec that do not have too computational intense iterative algorithms should be suitable for this project.
1.3 Method

First stage: Examine the speech codecs that exist today, what they are used for and the limitations they may have. Then based on this information choose a codec that fits this project.

Second stage: Create reference code for the chosen codec that generates bit exact results with the standard that is described for the original codec. This way results after each function in the code can be compared.

Third stage: Create and adapt code for the standardized IEEE 32-bit floating point format so that the effects of conversion to floating point can be examined without any impact from low precision.

Fourth stage: Adapt the code to the DSP floating point formats and use the available functions from earlier projects at ISY to emulate it on a regular computer.

Fifth stage: Testing. Compare sound quality of the codecs, test different kind of speech and sounds, introduce errors into the algorithms to test what parts are most vulnerable to errors and finally decrease the precision to see where the limit for intelligible speech goes.

1.4 Limitations and problem presentation

To get a GSM network system fully functional, literally hundreds of functions have to work together. This report however only briefly describes how the GSM speech coding works in general and then focuses on the speech codecs, especially the Full Rate codec. There are some assisting functions to the codecs that are of some interest and are shortly described since they may have an immediate effect on the sound. Transmission functions such as channel coding, error detection etc may also have an effect on the sound but is outside the scope of the report.

Both the encoder part and decoder part are included in this work, but the focus lies on the decoder and it is only this part that has been adapted to the floating point formats of the DSP. For emulation on personal computers, code libraries from previous projects at ISY have been used. How the DSP work and what operations it can perform will not be found in this report since it is already described in other projects. (See [7]).
1.5 Technical aids

For the work that has been done with Matlab, version 7.0.1 has been used. Trying to run code from this project on earlier versions will most likely give different and erroneous results.

For the work that has been done in C/C++, the editor Blodshed Dev-C++ version 4.9.9.2 has been used with GNU /mingw compiler. All code should follow ANSI-C.

For frequency spectrum analysis and images, Spectrum Player by Visualization Software LLC was used.

1.6 Motivation

1.6.1 Why use floating point format instead of integer?

The main reason to use a floating point format which is limited to lower precision, in this case 16 bits for external memory storage and 23 bits in the internal registers of the DSP, is that a much wider number range can be used in comparison to the range of 16 bit integers. This reduces or sometimes completely removes the need for scaling to keep the numbers within the valid range.

The downside is the low precision. Every possible number within a 16-bit integer can not be represented by the 16 bit floating point format, instead it has to be rounded. However, speech compression algorithms are built to make approximations and do not reproduce the sound perfectly. Thus there should be room for some rounding that are caused by the floating point formats without distorting the sound quality too much.

The advantage of having a low precision format is that less memory is needed when the values are stored. This makes it possible to cut down on the amount of memory that the DSP needs. This in turn reduces the size of the total chip area needed which makes the production costs lower and also decreases the power consumption which may save battery for portable devices.
1.6.2 Why choose the GSM Full Rate codec?

Cell phones and the telecommunications industry is the most important area where compression of speech is needed and thus the choice obviously had to be one of the GSM codecs. There are several different codecs available in the GSM standard and of these the Full Rate codec was chosen, partially because its wide use during the 1990’s and that it is still compatible with current networks. The main reasons however are that it has a constant compression rate which always generates the same bitrate, and that the algorithms are not so computational intense.

The other most interesting alternative would be the AMR codec which was the latest within GSM and is also used with 3G. It has better compression and can have better sound quality since it supports several bitrates. The downside is that it is more computational intense. Since it is within the objective of this work to keep transistor count and power consumption as low as possible, the AMR codec was rejected in favour of the Full Rate codec.

1.7 Report outline

In the beginning of chapter 2, the GSM speech coding is described in general and then moves on to the speech codecs that are used. The middle and end of the chapter describes how speech is created by humans, how the speech is perceived and how it can be modelled to be digitally reproduced.

Chapter 3 describes normal floating point formats and the special floating point formats the DSP at ISY uses. The chapter also explains how these formats can be emulated on an ordinary PC.

The speech encoder used in this project is described in chapter 4 along with the algorithms and functions it is made up of.

Chapter 5 describes the GSM Full Rate decoder and its algorithms, along with the solutions and adaptations made to convert the decoder to the floating point format.

Chapter 6 describes the work that has been done in this project, the effects that have been observed during the project and the limitations found when converting the codec for the floating point format.

The results of the project can be found in chapter 7, while chapter 8 suggests future work.
2 Theory

2.1 GSM speech coding

The GSM system consists of a large number of functions that handle different areas of the network traffic, but the focus is here on the speech coding parts that are used in the cell phones and in the base stations.

The functions that directly affect the speech coding and sound quality on the transmitting side are shown in figure 2.1. The conversion from A-law (see 2.3.1) to PCM is only necessary in the GSM network gateway when the samples are coming from another network than the GSM network. This function is never necessary in the cell phones.

The speech encoder receives its input either from the audio part of the cell phone (microphone) or from the network side. The input signal is a 13 bit uniform PCM signal. The encoder calculates speech frames which are then passed on to the so called TX DTX handler. DTX stands for discontinuous transmission, meaning that information will only be transmitted when necessary. The transmission will pause when there is no speech, which saves battery time on the cell phone and also saves bandwidth over the network. This is detected by the Voice Activity Detection function, VAD. The voice activity detection takes its input parameters from the speech encoder and uses this information to determine the noise levels and detect if there is any speech present in the frame. The result from the VAD is used by the DTX handler to determine if transmission should be shut off.

Figure 2.1: Functions on the transmitting side.

The speech encoder receives its input either from the audio part of the cell phone (microphone) or from the network side. The input signal is a 13 bit uniform PCM signal. The encoder calculates speech frames which are then passed on to the so called TX DTX handler. DTX stands for discontinuous transmission, meaning that information will only be transmitted when necessary. The transmission will pause when there is no speech, which saves battery time on the cell phone and also saves bandwidth over the network. This is detected by the Voice Activity Detection function, VAD. The voice activity detection takes its input parameters from the speech encoder and uses this information to determine the noise levels and detect if there is any speech present in the frame. The result from the VAD is used by the DTX handler to determine if transmission should be shut off.
There is another effect that discontinuous transmission brings. The perceived noise would, if no artificial noise was added, drop to a very low level. This is found to be very disturbing if presented to a listener without modification. Therefore the noise is kept at the same level by creating an artificial noise that is calculated by the comfort noise function. This noise information is sent in the last frame before the transmission is paused.

On the receiving side the functions are placed in the opposite order. The RX DTX handler determines what functions should be used for each frame. The info bits and SID flag (corresponds to the SP flag) comes from the transmit side, while the Bad Frame Indicator (BFI) and Time Alignment Flag (TAF) is information added by the radio subsystem.

![Figure 2.2: Functions on the receiving side.](image)

At the receiving side, frames may be lost due to transmission errors and frame stealing. To minimize the effects of the lost frames, a scheme is used to substitute a lost frame with a predicted one. The predicted frame is calculated based on the previous frames since simply inserting a silent frame would be more disturbing to the listener. However if there are several errors in a row the sound will eventually be muted, alerting the listener that there are problems with the transmission.

The comfort noise function is used on the receiving side when a frame with noise information is sent from the transmitting side just before it shuts off the transmission. The speech codec is then fed with artificial noise from the comfort noise function instead of real speech frames. [1, 2]
2.2 Codecs used in the GSM system

Since telephone networks are digital systems and speech is analogue, the speech has to be digitized. This is usually done using PCM (Pulse Coded Modulation) and gives a bit stream of 64kbit/s. But this rate is too high to be used in large scale over a radio link. Thus the GSM system needs speech coding algorithms to decrease the data traffic. There are currently four different speech coding standards used in GSM. They vary in sound quality, complexity and bitrate, but they are all so called hybrid codecs of different types.

The first codecs that was developed for GSM was the Full Rate speech codec, which has an average to fair sound quality and has a bitrate of 13kbit/s. Soon after the GSM was released, the Half Rate codec was developed which utilizes a more advanced technique called CELP. It has a similar sound quality to the FR codec, but only has a 5.6 kbit/s bitrate which allowed more cellphone users without having to change the network infrastructure [3, 4].

The later codecs that were developed for GSM were the Enhanced Full Rate and Adaptive Multi Rate codecs. The AMR codec uses several coding algorithms that allow the bitrate to vary between 4.75 kbit/s and 12.2 kbit/s. The one with the highest bitrate is the same as the EFR codec. The sound quality is also better than for the FR and HR codecs. The biggest advantage with variable bitrate for the codec is that the remaining bits can be used for error correction instead when there is a lot of interference on the network [5, 6].
2.3 Speech codecs

In general there are three different types of speech codecs, which have very different characteristics and areas of use. The first type is called waveform codecs and offer good sound quality but needs high bit rate. The second type is the source codecs which offer low bit rates, but have poor sound quality that is perceived as synthetic. The third type is a combination of the other two and is also called hybrid codecs. This makes good quality sound possible at fairly low bit rates.

![Figure 2.3: Sound quality of speech for the codec types. [8]](image)

2.3.1 Waveform codecs

Waveform codecs are quite simple algorithms and reconstructs the signal without using any information about how it originally was generated. An example of a waveform codec is the A-law compression algorithm used in the regular phone network (for example ISDN uses this) where 16-bit linear samples are compressed to 8-bit logarithmic samples. This means that every sample still exist in a compressed format where the precision is lowered. Trying to further decrease the number of bits used per sample with this type of coding would be difficult as sound quality will decrease very fast when less than 8 bits are used per sample. It is thus difficult to reach a bitrate lower than 64 kbit/s with this type of coding.

There is though another way for waveform codecs to decrease the bitrate and that is by using simple predictions. The coder uses the same algorithm as the
decoder to predict what the next sample will be. The coder will compare the results of the prediction with the real sample and then send the error information to the decoder instead of a full sample. The decoder can then add the error to its own prediction to recreate the original sample. This type of coding is called Differential PCM (DPCM) or Adaptive Differential PCM (ADPCM) and makes it possible to reach around 16 kbit/s bitrate. [8]

2.3.2 Source Codecs

Source codecs use a model for how the sound was generated and tries to calculate parameters that can be used to reconstruct the sound at the decoder side. Source codecs that are adapted for speech are called vocoders. These approximate the mouth and nose cavities as a row of cylinders that have different diameters. (See chapter 2.4). The information sent to the decoder is thus the parameters for the different sized cylinders which means that only a small amount of data is sent, in comparison to the inefficient method of sending full samples. Also information about the pitch is sent to the decoder. The pitch is needed to reconstruct the basic sound that is sent through the cylinders (This sound is called excitation, see chapter 2.4.2). The excitation is basically a pulse train which varies with the pitch of the speech. If the sound is not speech, white noise can be used instead of the pulse train to reconstruct the sound.

Since vocoders use this simplified cylinder model, the sound quality is suffering from the approximations. The speech is usually considered to sound synthetic and “robotic” with these codecs, even though it is possible to hear what is being said. The sound quality will only be insignificantly improved by using a greater number of parameters for the cylinder model and that is why most vocoders stay below 2-3 kbit/s in bitrate. [8]

2.3.3 Hybrid Codecs

In order to get a lower bitrate than the wave form codecs but better sound quality than the source codecs, a mixture of these two has been developed. Hybrid codecs use the cylindrical model just as the source codecs, but also use a sequence of samples as excitation instead of a pulse train or white noise. The decoder will try to find the sample sequence most suitable to make the reconstructed sound as similar to the wave form of the original sound as possible. The process to search for the pulse sequence and model parameters that gives the best result is called Analysis by Synthesis (AbS). This is however a computational intensive method and it would not be realistic to try all the possible combinations. The codecs instead uses different algorithms and approximations to find a result faster that is considered to be good enough.
The Full Rate codec uses something called Regular Pulse Excitation (RPE) to create the excitation. The RPE encoder sends information about the time position and amplitude for the first pulse. The pulses that follow only have information about the amplitude so the decoder will assume that they have a constant interval between each other. The predecessor, Multi Pulse Excitation (MPE), includes time position for all pulses which actually has proven to be less efficient since RPE can have more pulses instead of time position information. These two codecs work fairly well with a bitrate above 10 kbit/s. The GSM Full Rate uses a bitrate of 13 kbit/s with the RPE codec.

A more efficient codec is the Code Excited Linear Prediction (CELP) that heavily uses the AbS method to find the best pulse sequence for the excitation. The encoder compares the chosen sequence to those available in a codebook and then passes an index number to the decoder. The decoder can then use the index number to find the same sequence in its own codebook. The bitrate needed for transferring information about the excitation is greatly reduced this way. [8]
2.4 A model of the human speech

By knowing how human speech works and how it is created, it can be modelled and approximated with digital parameters. This allows for better compression of the speech data, as the speech then can be reconstructed with the help of a few parameters sent into the model. Most models use that creation of speech can be separated into two different parts. The first part is the basic sound that is created in the throat when air passes by the vocal cords. The second part is the reflections of the sound in the mouth and nose cavities.

2.4.1 The human speech

The basic sound when pronouncing vowels is created by the vibrations of the vocal cords. The pitch of the sound varies depending on how tense or relaxed the vocal cords are, which is controlled by muscles in the throat. The amplitude of the sound is regulated by the air volume that passes through the throat. When the sound passes through the mouth and nose, letters and words are created from the basic sound. The tongue, lips and teeth also help to alter the sound.

Vowels are created by letting the air flow freely from the throat and are thus very dependant on the sound created by the vocal cords in the throat. Consonants on the other hand that contain sharp or sudden sounds may not be affected by the basic sound at all. For example the letters “s” and “f” (fricatives) which are created in the front of the mouth, or “p” and “k” (plosives) which are created by a sudden burst of air when some part of the mouth has been completely closed and then rapidly opened again.

![Figure 2.4: Shape of mouth cavity when pronouncing certain vowels.](image)
Figure 2.4 and 2.5 shows how the throat and mouth are shaped when pronouncing certain voiced sounds. Notice that it is only the shape of the throat and mouth that differs, the basic sound and pitch created by the vocal cords can remain constant for all these sounds. The shape of the mouth can be thought of as a filter that the sound passes through that adds new characteristics to it. This is also how the speech model should be thought of; an input sound and a filter. [8, 9]
2.4.2 Excitation

The sound that is created by the vocal cords is usually called excitation when dealing with voice codecs. When the vocal cords open and close rapidly, sound is created in the form of periodic pulses. The time of the periodicity varies depending on the pitch, but is usually within 2 to 20 ms for vowels with voiced speech. This is called long term periodicity, even though it may seem like short periods. The same periodic behaviour can not be seen for unvoiced sounds, as the vocal cords then let the air pass unrestricted and no vibrations are caused.

![Figure 2.6: Voiced speech with visible long-term periodicity. [8]](image)

![Figure 2.7: Unvoiced speech lacks most of the long-term periodicity. [8]](image)

Both source codecs and hybrid codecs use the long term periodicity to reconstruct the excitation tone for voiced speech. Hybrid codecs also use samples for the excitation, which the source codecs do not. During periods of the speech that is unvoiced, the source codec decoder even replaces the excitation with generated white noise.
2.4.3 Formants

The second part of the model is the shape of the mouth cavity. The sound is reflected against the walls of the mouth and nose cavity on the way out. This distorts the original sound from the throat where the excitation was created and shapes the sound to something that can be understood as pronounced letters and words. When looking at the frequency spectrum for speech, it can be seen that each letter have characteristic peaks at certain frequencies where the energy is concentrated. (Swedish vowels have only one sound for the vowels which makes them suitable for showing in a spectrogram, as opposed to English vowels that have several letters when pronouncing them, for example “aye” for the letter “i”.)

These peaks are called formants and create a combination that is unique for each of the voiced speech letters. The peaks with the lowest frequencies are the most important for the understanding of the letters. Since the frequency range is limited for telephony, only the three lowest formants are considered to be of interest. These are called f1, f2 and f3. These formants are necessary for making the speech intelligible, but the higher formants may though add some better quality to the speech.

Figure 2.8: Vowels (Swedish) displayed in a spectrogram.

Figure 2.9: Formants shown in a smoothed frequency spectrum. [10]
The formants are created when the sound is reflected in the mouth cavity and standing waves arise at certain frequencies. When the shape of the mouth changes, the standing waves may be limited depending on where the restrictions are. If the standing wave is restricted at a point where it has the maximum pressure, the frequency will be lowered for that formant. If the restriction is close to a node where the pressure is low, the frequency will be higher for the formant.

The frequency for the lowest formant, $f_1$, depends mostly on how restricted the space is in the front part of the mouth. For example the sound [iː] (as in “me”) is created when the tongue almost touches the ceiling of the mouth, while [aː] (as in “father”) is pronounced with a more open mouth.

The frequency for the second formant, $f_2$, is more dependent on restrictions further back in the mouth, closer to the throat. The sound [æː] (as in “help”) and [uː] (as in “you”) shows this difference, where the throat is more restricted when pronouncing the [æː]. (See also the chart in figure 2.5). Furthermore the lips can be used to lower or raise the frequencies for all the formants. [11]
2.4.4 Reflection coefficients

When approximating the shape of the vocal tract, reflection coefficients or Log Area Ratios are used in the GSM full rate codec. These parameters describe how the sound is reflected and amplified when it passes through the vocal tract. The vocal tract can be thought of as a row of cylinders with different diameters that thus reflect the sound waves differently as they pass through.

![Simplified model of the vocal tract](image)

**Figure 2.11: Simplified model of the vocal tract**

Since the GSM full rate decoder needs the excitation and reflection coefficients to reconstruct the speech, the encoder has to separate the original speech into these two parts. The reflection coefficients are converted into Log Area Ratios before they are sent, since these are less sensitive to transmission errors.
2.5 Frequency range and characteristics of speech

The sound of a human voice always contains certain overtones, which are different for each person. This makes it possible for people to recognize each other by simply hearing that voice. Over telephone however it may be more difficult to immediately recognize a person, since the frequency range for analogue phone lines is only 300 – 3400 Hz. (Digital voice transmissions such as ISDN and GSM have a theoretical upper limit of 4000 Hz)

The lowest tone of a human voice during speech usually lies around 90-250 Hz and is thus outside the phone frequency range. But since speech have overtones to the lowest tones and the human brain tends to “fill in” the missing low frequency, this does not have much of an impact on the perceived speech. The highest frequencies that speech normally contains is up to around 6-8 kHz, which is also outside the range. However, the most important formants and overtones for speech lie within the range.

![Figure 2.12: Hearing curve and frequency ranges.](image)

As the picture shows, the frequency range for music is clearly greater compared to what the phone is capable of. The human ear is also not equally sensitive to sounds at different frequencies. Lower frequencies must have high amplitude to be heard, as well as the higher frequencies. The ear is on the other hand very sensitive to sounds with frequencies between 2 - 5 kHz.
3 The floating point formats

3.1 Floating point format

The floating point formats that the DSP uses are adapted for mp3 decoding and to have lower precision than more common hardware in order to cut down on the memory requirements. The most common formats that regular hardware uses are the 32-bit and 64-bit formats that are described by the IEEE-754 standard. Here 16-bit and 23-bit floating point formats are used instead, with quite different properties than the standard formats.

<table>
<thead>
<tr>
<th>Format</th>
<th>ISY 16-bit</th>
<th>ISY 23-bit</th>
<th>IEEE-754 32-bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponent</td>
<td>5 bits</td>
<td>6 bits</td>
<td>8 bits</td>
</tr>
<tr>
<td>Fraction</td>
<td>10 bits</td>
<td>16 bits</td>
<td>23 bits</td>
</tr>
<tr>
<td>Bias</td>
<td>2(^{-11})</td>
<td>2(^{-11})</td>
<td>1</td>
</tr>
<tr>
<td>Exponent format</td>
<td>2’s complement</td>
<td>2’s complement</td>
<td>Biased (127)</td>
</tr>
<tr>
<td>Max range</td>
<td>±3.198⋅10(^1)</td>
<td>±2.097⋅10(^6)</td>
<td>±3.403⋅10(^{38})</td>
</tr>
<tr>
<td>Min range</td>
<td>±1.490⋅10(^{-8})</td>
<td>±2.274⋅10(^{-13})</td>
<td>±1.76⋅10(^{-38})</td>
</tr>
</tbody>
</table>

*Table 3.1: Differences between the floating point formats*

For the 16-bit format, the floating point number is

\[ x = (-1)^{sign} \cdot 2^{exponent-11} \cdot (1 + \frac{mantissa}{1024}) \]

except for zero which is represented by an exponent of -16. Notice the bias of -11 for the exponent.

For the 32-bit format, the number is

\[ x = (-1)^{sign} \cdot 2^{exponent-11} \cdot (1 + \frac{mantissa}{65536}) \]

except for zero which is represented by an exponent of -32.
3.2 Emulation of the DSP on PC

The DSP uses 23-bits for internal arithmetic calculations, while the 16-bit format is used externally when storing the values to memory. A regular PC cannot handle the special floating point formats natively like the DSP can. To be able to emulate the programs for the DSP on a PC, a wrapper library with floating point functions is used. The wrapper uses integer formats and instructions towards the hardware, but behaves as if the special floating point operations were used.

![Diagram of floating point wrapper library](image1.png)

**Figure 3.1:** Floating point wrapper library.

The operations that are used from the wrapper library are the following:
- Op_fsub: 23-bit subtraction
- Op_fadd: 23-bit addition
- Op_fmul: 23-bit multiplication
- Op_fexpand: Convert a 16-bit float to 23-bit float
- Op_fround: Convert a 23-bit float to 16-bit float (with rounding)
- Op_fint: Convert a 23-bit float to an integer (with scaling $2^{15}$)

Trying to read the integer value when there is a floating point value stored within it would make no sense, until converted to an actual integer value with op_fint. The picture below demonstrates the floating point value 12.0 stored in a 32-bit integer. If this number would be read as an integer, the value would incorrectly be interpreted as 14848.

![Diagram of floating point number stored in an integer](image2.png)

**Figure 3.2:** Floating point number stored in an integer.
3.3 Precision and quantization

When converting numbers from 16-bit integer representation to 23-bit floating point representation, the precision is good enough to handle all the integer numbers possible. But when converting to 16-bit floating point representation, not all numbers can be represented since there is only a 10-bit mantissa available. Up to the value 2048 every integer number can be represented, between 2048 and 4096 only every second number, between 4096 and 8192 only every fourth number and so on.

Depending on how the conversion is implemented, the quantization error may vary. The quantization error is how much a converted value may differ from the actual value. Rounding gives a smaller error than if simple truncation is used. The functions in the wrapper library use rounding when converting from 23-bit float to 16-bit float. When integers are loaded from memory, they are always converted to 23-bit float immediately and do not suffer from the quantization effects for such small integers.

![Figure 3.3: Quantization error.](image)
3.4 Conversion and scaling

The input parameters are represented in a fixed point format. Therefore conversion to the floating point format is needed before any floating point arithmetic can be performed. However, there is no function available for conversion of the integers to floating point. Instead the integer number is treated as a 23-bit float and then the implicit one from the mantissa is subtracted by using float subtraction. This way the integer is converted to float, but with a scaling of $2^{-27}$ (or $\frac{1}{32768 \cdot 2048}$). The scaling depends on the number of bits in the mantissa and on the bias that the floating point format uses on the exponent. The mantissa is 16 bits wide which gives a scaling of $2^{-16}$, and the bias is responsible for another $2^{-11}$.

![Conversion from fixed point number to 23-bit float.](image1)

### Figure 3.4: Conversion from fixed point number to 23-bit float.

However, by setting the exponent bits to something else than 0 when loading the value and subtracting them again, the scaling can be adjusted as needed. Setting the exponent bits to 27 for example, would result in no scaling and the result would be 512.

![Conversion from fixed point number to 23-bit float.](image2)

### Figure 3.5: Conversion from fixed point number to 23-bit float.
There is one problem with the scaling however. If for example the integer number 1 is converted to 23-bit float with a scaling of \(2^{-27}\) and then converted to a 16-bit float when it is about to be stored in memory, the range of the 16-bit float is not large enough to hold such a small number. It will instead be rounded to zero. Also, the value must be less than 32, as the highest number the 16-bit float can hold is just below 32. Scaling of the input parameters is thus necessary if they are going to be stored in memory.

Unfortunately scaling can not be avoided since the range of the 16-bit float is too small to make up for the scaling effects when converting the integers to float. Most scaling is though possible to avoid by carefully preparing the different variables for each step in the algorithms. Constants could be upscaled to counter the downscaled variables but must stay below the upper limit of the 16-bit float format since they have to be loaded from memory. The upper limit for the exponent of the 23-bit floating point format is \(2^{20}\), which is far more than needed in this case.

The table below shows the maximum up and down scaling that is possible for some of the constants and variables while still fitting the 16-bit floating point format.

<table>
<thead>
<tr>
<th>Variable/constant</th>
<th>Max magnitude</th>
<th>Min magnitude</th>
<th>Max scaling</th>
<th>Min scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>-5</td>
<td>0.184</td>
<td>(2^2)</td>
<td>(2^{-24})</td>
</tr>
<tr>
<td>MIC</td>
<td>-32</td>
<td>-4</td>
<td>(2^{-1})</td>
<td>(2^{-28})</td>
</tr>
<tr>
<td>LAR</td>
<td>-32</td>
<td>-4</td>
<td>(2^{-1})</td>
<td>(2^{-28})</td>
</tr>
<tr>
<td>INVA</td>
<td>0.1199904</td>
<td>0.05</td>
<td>(2^{-1})</td>
<td>(2^{-28})</td>
</tr>
<tr>
<td>s</td>
<td>-65535 (*)</td>
<td>1</td>
<td>(2^{-12})</td>
<td>(2^{-26})</td>
</tr>
<tr>
<td>ep</td>
<td>-65535 (*)</td>
<td>1</td>
<td>(2^{-12})</td>
<td>(2^{-26})</td>
</tr>
<tr>
<td>dp</td>
<td>-65535 (*)</td>
<td>1</td>
<td>(2^{-12})</td>
<td>(2^{-26})</td>
</tr>
</tbody>
</table>

* The fixed point codec clips values higher than 32 767 to not overflow the signed 16-bit integers, but a floating point implementation does not need to do that with the proper scaling and thus the values may be larger.
4 GSM full rate encoder

4.1 Functional overview

The encoder contains more steps than the decoder. Actually the encoder contains some of the parts of the decoder also to make sure that the same values are used as in the decoder when determining the long term prediction parameters.

![Diagram](image)

**Figure 4.1: Overview of the Full Rate encoder. [3]**

First, low frequency and static signals are removed from the samples. The samples are then run though a filter to boost the higher frequencies before the samples are segmented into frames containing 160 samples.

The next step is to calculate the autocorrelation parameters, which are needed for the Schur recursion to calculate the reflection coefficients. The reflection coefficients are then transformed into Log Area Ratios which will be sent to the decoder. The LAR’s are also decoded again in the encoder. It may at first seem strange to decode what has just been coded, but is to ensure that the same values are used for both the encoder and decoder. The decoded LAR’s are then interpolated with the LAR’s from the previous frame to decrease the effects of any sudden changes. The interpolated LAR’s are transformed back to reflection coefficients before they are used in the short term analysis filtering.

To calculate the excitation samples, the reflection coefficients are used to do inverse filtering on the speech samples. For the excitation samples the long term prediction lag and gain is calculated by comparing with previous excitation samples. When the excitation samples have passed through a weighting filter to
decrease the noise, every third sample is picked out to form a new shrunken sample sequence. The samples are then quantized according to an APCM table. The samples are transmitted to the decoder, but are also decoded in the encoder again to be able to compare with the next frame and find the LTP lag and gain for that frame.

4.2 Preprocessing

In the preprocessing stage of the encoder, the samples are first adjusted to fit the encoder. The samples are downscaled since they come in a 16-bit format, but only 13 bits are used and the 3 least significant bits are ignored.

When the samples are downscaled, the offset compensation tries to remove any static parts of the input signal by running it through a high-pass filter. The offset free signal $s_{of}$ is calculated from the input signal $s_o$ according the formula

$$s_{of}(k) = s_o(k) - s_o(k-1) + \alpha \cdot s_{of}(k-1)$$  \hspace{1cm} (4.1)

where the constant $\alpha = \frac{32735}{32768} \approx 0,99899$.

It should be mentioned that the original integer encoder implementation uses 32-bit variables for this calculation.

The next step is to run the offset free signal through a pre-emphasis filter. Since formants with lower frequencies contain more energy, the pre-emphasis stage is used to enhance the higher frequencies. This makes the speech model work better and results in better transmission efficiency. The signal $s$ is calculated from $s_{of}$ like

$$s(k) = s_{of}(k) - \beta \cdot s_{of}(k-1)$$  \hspace{1cm} (4.2)

where the constant $\beta = \frac{28180}{32768} \approx 0,85999$. 

Figure 4.2: Downscaling

| $S$ | $VVVVVVVVVVVXX$ | $s$ = sign
| $V$ | $VVVVVVVVVVV$ | $v$ = valid
| $000S$ | $VVVVVVVVVVVVV$ | $x$ = dont care
4.3 LPC analysis

In the previous steps the samples can be considered as a signal or a continuous flow of samples. But from here on, the samples need to be treated in separate blocks. The samples are thus segmented into blocks of 160 samples, forming a speech frame. A linear prediction of order p=8 is then made for each frame. The goal of the linear prediction algorithms is to find parameters for a filter that predicts the signal in the current frame as a weighted sum of the previous ones.

The first step is to calculate $p+1 = 9$ values of the autocorrelation function $ACF$ for the samples. Since 160 values are summed in this calculation, the integer encoder uses 32-bit variables to accomplish this and hold the resulting $ACF$ values. The samples are also scaled first with regard to the maximum value so that no overflow occurs in the integer encoder.

$$ACF(k) = \sum_{i=k}^{159} s(i)s(i-k) \quad k = 0...8$$ \hspace{1cm} (4.3)

The $ACF$ values are then used as input to a Shur recursion where the eight reflection coefficients are calculated. The range is $-1 \leq r(i) \leq +1$ for all the coefficients.
The reflection coefficients are transformed to Log. Area Ratios since these are better to quantize and has better companding characteristics. The relation between the reflection coefficients and LAR’s is $\text{Log area}(i) = \log_{10}\left(\frac{1 + r(i)}{1 - r(i)}\right)$. This is approximated for the implementation of the GSM encoder by

If $|r(i)| < 0.675$ then $LAR(i) = r(i)$

If $0.675 \leq |r(i)| < 0.950$ then $LAR(i) = \frac{r(i)}{|r(i)|} (2 - |r(i)| - 0.675)$ (4.4)

If $0.950 \leq |r(i)| \leq 1.000$ then $LAR(i) = \frac{r(i)}{|r(i)|} (8 - |r(i)| - 6.375)$
To make the LAR’s as small as possible, they are also quantizated and coded before they are transmitted. For the first two LAR’s, there are 6 bits each reserved in the packed frame. The number of bits then decreases so that the last two LAR’s only get 3 bits each. The equation for coding the LAR’s is

\[ LAR_c(i) = \text{round}(A(i) \cdot LAR(i) + B(i)) \]  

(4.5)

The values that the constant arrays A and B have are shown in the table below, along with the allowed range for each LAR.

<table>
<thead>
<tr>
<th>i</th>
<th>A(i)</th>
<th>B(i)</th>
<th>LAR_c(i) range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>20</td>
<td>0</td>
<td>-32 to +31</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>0</td>
<td>-32 to +31</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>4</td>
<td>-16 to +15</td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td>-5</td>
<td>-16 to +15</td>
</tr>
<tr>
<td>5</td>
<td>13.637</td>
<td>0.184</td>
<td>-8 to +7</td>
</tr>
<tr>
<td>6</td>
<td>15</td>
<td>-3.5</td>
<td>-8 to +7</td>
</tr>
<tr>
<td>7</td>
<td>8.334</td>
<td>-0.666</td>
<td>-4 to +3</td>
</tr>
<tr>
<td>8</td>
<td>8.824</td>
<td>-2.235</td>
<td>-4 to +3</td>
</tr>
</tbody>
</table>

*Table 4.1: LAR coding constants and range of the resulting variable*

### 4.4 Short term analysis filtering

The task of the short term analysis filtering clause is to try to remove the effects of the mouth and nose cavity so that the pure excitation signal can be extracted. For this the filter parameters, i.e. the reflection coefficients, are needed. But these cannot be used as is from the schur recursion step in the LPC analysis clause. Instead the compressed and coded LAR’s must be transformed back to reflection coefficients again. The reason for this is that the same values as the ones that the decoder receives must be used, since the decoder has to revert the calculation later to get the original sound.

When decoding the LAR’s, the inverse of equation 4.5 is used:

\[ LAR(i) = \frac{LAR_c(i) - B(i)}{A(i)} \]  

(4.6)

If the filter coefficients change too fast, there may be strange effects. To avoid this, the decoded LAR’s for this frame are interpolated with the LAR’s from the previous frame frames so that no sudden changes occur. Table 4.2 shows how the interpolation is applied over the samples in the frame. (J= frame number, i = LAR number).
Table 4.2: Interpolation of the reconstructed LAR’s

<table>
<thead>
<tr>
<th>sample #</th>
<th>LAR&lt;sub&gt;i&lt;/sub&gt;(i) =</th>
</tr>
</thead>
<tbody>
<tr>
<td>0...12</td>
<td>0.75·LAR&lt;sub&gt;i-1&lt;/sub&gt;(i) + 0.25·LAR&lt;sub&gt;i&lt;/sub&gt;(i)</td>
</tr>
<tr>
<td>13...26</td>
<td>0.5·LAR&lt;sub&gt;i-1&lt;/sub&gt;(i) + 0.5·LAR&lt;sub&gt;i&lt;/sub&gt;(i)</td>
</tr>
<tr>
<td>27...39</td>
<td>0.25·LAR&lt;sub&gt;i-1&lt;/sub&gt;(i) + 0.75·LAR&lt;sub&gt;i&lt;/sub&gt;(i)</td>
</tr>
<tr>
<td>40...159</td>
<td>LAR&lt;sub&gt;i&lt;/sub&gt;(i)</td>
</tr>
</tbody>
</table>

After the interpolation, the LAR’s are transformed back into reflection coefficients according to

If \(|LAR(i)| < 0.675\) then \(r(i) = LAR(i)\)

If \(0.675 \leq |LAR(i)| < 1.225\) then \(r(i) = \frac{LAR(i)}{|LAR(i)|} \cdot \left(\frac{|LAR(i)|}{2} + 0.3375\right)\) \hspace{1cm} (4.7)

If \(1.225 \leq |r(i)| \leq 1.625\) then \(r(i) = \frac{LAR(i)}{|LAR(i)|} \cdot \left(\frac{|LAR(i)|}{8} + 0.796875\right)\)

which is the inverse of equation 4.4.

When the reconstructed reflection coefficients are calculated, the short term analysis filtering can be done. Each sample \(s(k)\) in the frame is run through the filter one at a time. The effects of the eight reflection coefficients are applied to the sample and the result is a short term residual signal sample, \(d(k)\).

The implementation of the filter uses two temporary arrays, \(d_i\) and \(u_i\) where \(i=0...8\). The following equations are needed to calculate \(d(k)\):

\[
d_0(k) = s(k)\] \hspace{1cm} (4.8a)

\[
u_0(k) = s(k)\] \hspace{1cm} (4.8b)

\[
d_i(k) = d_{i-1}(k) + r_i \cdot u_{i-1}(k - 1)\] \hspace{1cm} (4.8c)

\[
u_i(k) = u_{i-1}(k - 1) + r_i \cdot d_{i-1}(k)\] \hspace{1cm} (4.8d)

\[
d(k) = d_8(k)\] \hspace{1cm} (4.8e)
4.5 Long term prediction

For the long term prediction, the speech frame needs to be divided into smaller frames called sub frames. One frame contains four sub frames which correspond to 5 ms of speech. The sub frames are denoted by $j$.

As mentioned in chapter 2.4.2, voiced speech has a typical periodicity. This is what the encoder tries to find when calculating the LTP lag ($N_j$). The long term prediction parameters are calculated for each sub frame ($j$) from the short term residual samples $d(k_j + k)$. The current samples are compared to the previous reconstructed samples $d'(k_j + k - \lambda)$ by calculating the maximum value from the cross-correlation $R_j(\lambda)$.

$$R_j(\lambda) = \sum_{i=0}^{39} d(k_j + i) \cdot d'(k_j + i - \lambda) \quad j = 0...3 \quad (4.9a)$$

$$k_j = k_0 + j \cdot 40$$

$$R_j(N_j) = \max(R_j(\lambda)) \quad \lambda = 40...120 \quad (4.9b)$$

The lag parameter tells how many samples ago that the speech looked most similar, which is the same as the periodicity. The valid range for this parameter is from 40 to 120 samples, meaning that the lag must be at least from the previous sub frame and at most two sub frames back in time. The lag parameter has to be coded with 7 bits to fit the value range and this makes it the largest parameter in the coded speech frame.
There is also a LTP gain parameter ($b_j$) that is needed to adjust the amplitude so that the found matching sample sequence and the current sample sequence has the same amplitude scale. This is calculated by dividing the cross correlation $R_j(N_j)$ with the autocorrelation ($S_j(N_j)$) of the previous found sample sequence $d'$.

$$b_j = \frac{R_j(N_j)}{S_j(N_j)} \quad j = 0...3 \quad (4.10a)$$

$$S_j(N_j) = \sum_{i=0}^{39} d^{12} (k_j + i - N_j) \quad (4.10b)$$

When coding the gain parameter, it is approximated very roughly so that it can be fit in 2 bits, thus having a value range from 0 to 3. The decision levels are according to table 4.3. When the parameter is decoded, it corresponds to the average value in the decision range. ($b_c$ = coded gain parameter)

<table>
<thead>
<tr>
<th>Decision level</th>
<th>$b_c$</th>
<th>Quantizing level</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \leq b_j \leq 0.2$</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>$0.2 &lt; b_j \leq 0.5$</td>
<td>1</td>
<td>0.35</td>
</tr>
<tr>
<td>$0.5 &lt; b_j \leq 0.8$</td>
<td>2</td>
<td>0.65</td>
</tr>
<tr>
<td>$0.8 &lt; b_j$</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

**Table 4.3:** LTP gain coding and decoding.
The next step is to calculate the long term residual signal \((e)\). This is done by first calculating an estimate of the short term residual signal \((d'')\) based on the lag and gain parameters that were previously calculated. This signal is then subtracted from the current short term residual signal \((d)\) and thus gives the difference between the new signal and the previous signal.

\[
d''(k_j + k) = b'_j \cdot d'(k_j + k - N_j) \quad j = 0...3
\]
\[
k = 0...39
\]
\[
e(k_j + k) = d(k_j + k) - d''(k_j + k) \quad k_j = k_0 + j \cdot 40
\]

The reconstructed short term residual signal \(d'\) can be calculated from the reconstructed long term residual signal \(e'\) and the estimated short term residual signal \(d''\). \((e'\) is calculated after the RPE encoding section so it can be used for the next sub frame LTP calculation.)

\[
d'(k_j + k) = e'(k_j + k) + d''(k_j + k)
\]

### 4.6 RPE encoding

In the RPE encoding clause, the long term residual signal is first run through a weighting filter. This is generally a low pass filter that tunes down frequencies that are more likely to contain sound that is perceived as noise to humans while not interfering with frequencies that contains sound that is perceived as tones.

The weighting filter used in GSM-FR is a FIR block filter described as

\[
x(k) = \sum_{i=0}^{10} H(i) \cdot e(k + 5 - i) \quad k = 0...39
\]

The algorithm is applied for each sub-segment and merges the 40 samples \(e(k)\) with the impulse response \(H(i)\). The coefficients of the filter are listed in table 4.4. When \(\omega = 0\) then \(H(\omega) = 2.779\) for the filter.

<table>
<thead>
<tr>
<th>(i)</th>
<th>5</th>
<th>4 or 6</th>
<th>3 or 7</th>
<th>2 or 8</th>
<th>1 or 9</th>
<th>0 or 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>(H(i))</td>
<td>1</td>
<td>0.70081</td>
<td>0.25073</td>
<td>0</td>
<td>-0.045654</td>
<td>-0.016357</td>
</tr>
</tbody>
</table>

*Table 4.4: Weighting filter coefficients.*
The filtered signal $x(k)$ is then downsampled so that there only remain 13 samples out of the original 40. This is done by selecting every third sample, like 0, 3, 6, 9 … or 1, 4, 7, 10… or 2, 5, 8, 11… or 3, 6, 9, 12… The first sequence and fourth sequence use the same samples, except for the first and last sample. In the first sequence, sample 39 is left out, while in the last sequence sample 0 is left out instead.

$$x_m(i) = x(k_j + m + 3 \cdot i) \quad i = 0...12 \ , \ m = 0...3 \quad (4.13a)$$

The decision of which sample sequence to select ($m$) is based on which sequence that contains most energy ($E_M$). $M$ is the grid selection variable which is coded with 2 bits in the sub frame and sent to the decoder.

$$E_M = \max(\sum_{i=0}^{12} x_m^2(i)) \quad (4.13b)$$

When the appropriate sequence has been selected, the samples are coded using APCM. This means that there is a block amplitude parameter of 6 bits for the sequence and each sample is coded to fit into only 3 bits. The block amplitude is based on the maximum value of any sample ($x_{\max}$) and is then quantized according to table 4.5. The samples are divided with the block amplitude and quantized according to table 4.6.

$$x'(i) = \frac{x_M(i)}{x'_{\max}} \quad i = 0...12 \quad (4.14)$$

where $x'(i)$ are the normalized samples, based on the decoded block amplitude $x'_{\max}$.
Table 4.5: Quantization table for the block amplitude $x_{\text{max}}$.

<table>
<thead>
<tr>
<th>Decision range $x_{\text{max}}$</th>
<th>Coded $x_{\text{max}}$</th>
<th>Quantized $x_{\text{max}}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3 - 31</td>
<td>0</td>
<td>31</td>
</tr>
<tr>
<td>32.63</td>
<td>1</td>
<td>63</td>
</tr>
<tr>
<td>64.95</td>
<td>2</td>
<td>95</td>
</tr>
<tr>
<td>96 - 127</td>
<td>3</td>
<td>127</td>
</tr>
<tr>
<td>128 - 159</td>
<td>4</td>
<td>159</td>
</tr>
<tr>
<td>160 - 191</td>
<td>5</td>
<td>191</td>
</tr>
<tr>
<td>192 - 223</td>
<td>6</td>
<td>223</td>
</tr>
<tr>
<td>224 - 255</td>
<td>7</td>
<td>255</td>
</tr>
<tr>
<td>256 - 287</td>
<td>8</td>
<td>287</td>
</tr>
<tr>
<td>288 - 319</td>
<td>9</td>
<td>319</td>
</tr>
<tr>
<td>320 - 351</td>
<td>10</td>
<td>351</td>
</tr>
<tr>
<td>352 - 383</td>
<td>11</td>
<td>383</td>
</tr>
<tr>
<td>384 - 415</td>
<td>12</td>
<td>415</td>
</tr>
<tr>
<td>416 - 447</td>
<td>13</td>
<td>447</td>
</tr>
<tr>
<td>448 - 479</td>
<td>14</td>
<td>479</td>
</tr>
<tr>
<td>480 - 511</td>
<td>15</td>
<td>511</td>
</tr>
<tr>
<td>512 - 575</td>
<td>16</td>
<td>575</td>
</tr>
<tr>
<td>576 - 639</td>
<td>17</td>
<td>639</td>
</tr>
<tr>
<td>640 - 703</td>
<td>18</td>
<td>703</td>
</tr>
<tr>
<td>704 - 767</td>
<td>19</td>
<td>767</td>
</tr>
<tr>
<td>768 - 831</td>
<td>20</td>
<td>831</td>
</tr>
<tr>
<td>832 - 895</td>
<td>21</td>
<td>895</td>
</tr>
<tr>
<td>896 - 959</td>
<td>22</td>
<td>959</td>
</tr>
<tr>
<td>960 - 1023</td>
<td>23</td>
<td>1023</td>
</tr>
<tr>
<td>1024 - 1151</td>
<td>24</td>
<td>1151</td>
</tr>
<tr>
<td>1152 - 1279</td>
<td>25</td>
<td>1279</td>
</tr>
<tr>
<td>1280 - 1407</td>
<td>26</td>
<td>1407</td>
</tr>
<tr>
<td>1408 - 1535</td>
<td>27</td>
<td>1535</td>
</tr>
<tr>
<td>1536 - 1663</td>
<td>28</td>
<td>1663</td>
</tr>
<tr>
<td>1664 - 1791</td>
<td>29</td>
<td>1791</td>
</tr>
<tr>
<td>1792 - 1919</td>
<td>30</td>
<td>1919</td>
</tr>
<tr>
<td>1920 - 2047</td>
<td>31</td>
<td>2047</td>
</tr>
<tr>
<td>2048 - 2303</td>
<td>32</td>
<td>2303</td>
</tr>
<tr>
<td>2304 - 2559</td>
<td>33</td>
<td>2559</td>
</tr>
<tr>
<td>2560 - 2815</td>
<td>34</td>
<td>2815</td>
</tr>
<tr>
<td>2816 - 3071</td>
<td>35</td>
<td>3071</td>
</tr>
<tr>
<td>3072 - 3327</td>
<td>36</td>
<td>3327</td>
</tr>
<tr>
<td>3328 - 3583</td>
<td>37</td>
<td>3583</td>
</tr>
<tr>
<td>3584 - 3839</td>
<td>38</td>
<td>3839</td>
</tr>
<tr>
<td>3840 - 4095</td>
<td>39</td>
<td>4095</td>
</tr>
<tr>
<td>4096 - 4352</td>
<td>40</td>
<td>4352</td>
</tr>
<tr>
<td>4353 - 4607</td>
<td>41</td>
<td>4607</td>
</tr>
<tr>
<td>4608 - 4864</td>
<td>42</td>
<td>4864</td>
</tr>
<tr>
<td>4865 - 5121</td>
<td>43</td>
<td>5121</td>
</tr>
<tr>
<td>5122 - 5378</td>
<td>44</td>
<td>5378</td>
</tr>
<tr>
<td>5379 - 5636</td>
<td>45</td>
<td>5636</td>
</tr>
<tr>
<td>5637 - 5894</td>
<td>46</td>
<td>5894</td>
</tr>
<tr>
<td>5895 - 6151</td>
<td>47</td>
<td>6151</td>
</tr>
<tr>
<td>6152 - 6409</td>
<td>48</td>
<td>6409</td>
</tr>
<tr>
<td>6410 - 6666</td>
<td>49</td>
<td>6666</td>
</tr>
<tr>
<td>6667 - 6924</td>
<td>50</td>
<td>6924</td>
</tr>
<tr>
<td>6925 - 7181</td>
<td>51</td>
<td>7181</td>
</tr>
<tr>
<td>7182 - 7438</td>
<td>52</td>
<td>7438</td>
</tr>
<tr>
<td>7439 - 7695</td>
<td>53</td>
<td>7695</td>
</tr>
<tr>
<td>7696 - 7952</td>
<td>54</td>
<td>7952</td>
</tr>
<tr>
<td>7953 - 8209</td>
<td>55</td>
<td>8209</td>
</tr>
<tr>
<td>8210 - 8465</td>
<td>56</td>
<td>8465</td>
</tr>
<tr>
<td>8466 - 8722</td>
<td>57</td>
<td>8722</td>
</tr>
<tr>
<td>8723 - 8979</td>
<td>58</td>
<td>8979</td>
</tr>
<tr>
<td>8980 - 9236</td>
<td>59</td>
<td>9236</td>
</tr>
<tr>
<td>9237 - 9493</td>
<td>60</td>
<td>9493</td>
</tr>
<tr>
<td>9494 - 9750</td>
<td>61</td>
<td>9750</td>
</tr>
<tr>
<td>9751 - 10000</td>
<td>62</td>
<td>10000</td>
</tr>
</tbody>
</table>

Table 4.6: Quantization of the RPE samples
5 GSM full rate decoder

The full rate decoder is in the GSM system fed with speech frames from the DTX handler and puts out sound samples to the audio circuits in the mobile station, or to an A-law converter in the base station system. In this project, the input is a file containing encoded speech frames following the standardized format. The output is either a file containing raw samples, or a wave file with the proper header which allows it to be played by most sound programs. The samples are 16-bit mono PCM with 8 kHz sample rate.

5.1 The speech frame

The speech frames sent to the decoder contain 76 variables and have a size of 260 bits in total. The variables have different sizes, the largest having 7 bits. This is sufficient for storing a value of up to 120 which is needed for the LTP Lag variable. One speech frame is equivalent to 20ms of speech, but is also divided into four sub frames. The filter parameters (Log Area Ratios) are valid for the whole frame, while there are 13 excitation pulses (RPE samples) per sub frame. The sub frames also have a block amplitude and a RPE selection grid parameter, along with the LTP parameters for lag and gain.

<table>
<thead>
<tr>
<th>Type</th>
<th>Param. name</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>LAR 1</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>LAR 2</td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>LAR 3</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>LAR 4</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>LAR 5</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>LAR 6</td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>LAR 7</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>LAR 8</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Type</th>
<th>Param. name</th>
<th>Bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTP Lag</td>
<td>Nc</td>
<td>7</td>
</tr>
<tr>
<td>LTP Gain</td>
<td>b_c</td>
<td>2</td>
</tr>
<tr>
<td>RPE grid pos.</td>
<td>M0</td>
<td>2</td>
</tr>
<tr>
<td>Block ampl.</td>
<td>x_{max}</td>
<td>6</td>
</tr>
<tr>
<td>RPE pulse 1</td>
<td>x_{M0}(0)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 2</td>
<td>x_{M0}(1)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 3</td>
<td>x_{M0}(2)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 4</td>
<td>x_{M0}(3)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 5</td>
<td>x_{M0}(4)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 6</td>
<td>x_{M0}(5)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 7</td>
<td>x_{M0}(6)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 8</td>
<td>x_{M0}(7)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 9</td>
<td>x_{M0}(8)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 10</td>
<td>x_{M0}(9)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 11</td>
<td>x_{M0}(10)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 12</td>
<td>x_{M0}(11)</td>
<td>3</td>
</tr>
<tr>
<td>RPE pulse 13</td>
<td>x_{M0}(12)</td>
<td>3</td>
</tr>
</tbody>
</table>

Figure 5.1: Contents of a speech frame.
5.2 Functional overview

The task of the decoder is to run the excitation through the filter to create sound samples. But before this can be done both the excitation pulses and the filter parameters need some preparation.

The decoder starts with the excitation pulses, one subframe at a time. The first thing that happens is that the 13 excitation samples are upsampled to 40 samples, and then multiplied with the block amplitude. Samples from previous sub frames are then multiplied with the new excitation samples according to a pattern that is decided by the long term prediction lag and gain. The resulting samples from each subframe are merged into one single array for the whole frame and can now be used in the short term synthesis. But first the filter parameters must be decoded.

The Log Area Ratios in the speech frame are first interpolated with previous filter parameters so that any sudden changes from the previous frame are smoothed out. The parameters are then multiplied with static factors so that they turn into reflection coefficients that can be used in the short term synthesis filtering. In the short term synthesis filter, the excitation samples are multiplied with the reflection coefficients in several iterations and the result is finally fed through a de-emphasis filter that decreases the magnitude of high frequencies.
5.3 RPE Decoding and Long Term Prediction

There are 13 excitation pulses for every sub frame, along with a parameter for block amplitude and a RPE grid position parameter. The RPE grid position tells the position of the first pulse in the upsampling array. This value can be 0 to 3. (In the picture below for example, this value is 2). The rest of the pulses are inserted into every third position, as this is a regular pulse excited (RPE) codec.

The block amplitude is coded with 6 bits using APCM, see table 4.5. This means that every value corresponds to a larger value, on a non linear scale. The translation can either be made by calculating each value, or by simply using a table. In this case, a file containing the translation values is loaded into the memory to avoid the calculation.

The RPE samples are coded according to table 4.6 and are simply decoded by multiplying with 2 and subtracting 7. The additional scaling that results is eliminated by letting the block amplitude have an opposite scaling.

![Figure 5.3: Upsampling of the excitation samples](image)
The result of the RPE decoding and upscaling is the reconstructed long term residual signal $r_e'$, which represent the change since previous subframes. The short term residual signal $d_r'$ needed for the short term synthesis filtering is calculated according to equation 4.11c, where the long term residual signal is added to the short term residual signal from previous subframes ($d_r''$).

$d_r''$ is in turn calculated by applying the LTP gain and selecting previous sample sequence with respect to the LTP lag parameter (See equation 4.11a).

### 5.4 LAR decoding and short term synthesis filtering

Before the short term synthesis filtering can be done, the Log. Area Ratios have to be transformed back to reflection coefficients. This is done the same way as in the encoder; first the LAR’s are decoded according to equation 4.6, then interpolated according to table 4.2 to avoid any sudden changes, and then transformed into reflection coefficients according to equation 4.7.

The encoder separated the speech samples into reflection coefficients and excitation samples in the short term analysis filtering. The decoder does the opposite in the short term synthesis filtering; the effects of the reflection coefficients are reapplied to the excitation samples to reconstruct the speech samples.

For the implementation of the synthesis filter, two temporary arrays are needed, here called $s_{ri}$ and $v_i$. The following equations describe the filter:

\[
s_{r(0)}(k) = d_r'(k) \quad \text{(5.1a)}
\]

\[
s_{r(i)}(k) = s_{r(i-1)}(k) - r_{r'}(9-i) \cdot v_(8-i) (k-1) \quad i = 1...8 \quad \text{(5.1b)}
\]

\[
v_(9-i) (k) = v_(8-i) (k-1) + r_{r'}(9-i) \cdot s_{r(i)}(k) \quad \text{(5.1c)}
\]

\[
s_r'(k) = s_{r(8)}(k) \quad \text{(5.1d)}
\]

\[
v_0 (k) = s_{r(8)}(k) \quad \text{(5.1e)}
\]

$s_r(k)$ are the reconstructed speech samples, $d_r'(k)$ are the reconstructed excitation samples (reconstructed short term residual signal) and $r_{r'}i$ are the reconstructed reflection coefficients.
5.5 Postprocessing

In the encoder, the signal was run through a pre-emphasis filter that enhanced the higher frequencies to make the speech model more efficient. The effects of this now have to be removed and thus the signal is run through a de-emphasis filter that dampens the higher frequencies again.

\[ s_{ro}(k) = s_r(k) + \beta \cdot s_{ro}(k-1) \quad k = 0...160 \]

where \( \beta = \frac{28180}{32768} \approx 0.85999 \)

The last thing that has to be done in the decoder is to upscale the signal again since the output is supposed to only use the highest 13-bits and not all 16-bits that were used in the decoder.
6 Tests and codec behaviour

6.1 Codec implementations

The first idea was to create an integer encoder and decoder in Matlab. These were intended to be used as reference codec so that new samples could be produced that follow the GSM Full Rate standard bit-exact. The integer codec was implemented, but proved to be a bit tricky in Matlab since there is no good support for 16-bit integers. Matlab usually calculates everything with double or single floating point precision, which though was excellent for creating a floating point decoder with high precision.

The code library for the DSP floating point operations was available in C from the previous project (See [7]), and thus the decoder also had to be implemented in C to use these functions. An integer encoder was implemented, as well as the following decoders; 16/32-bit integer, IEEE 32-bit fp, 16/23-bit fp, 16/-17-bit fp, 12/17-bit fp, 12/13-bit fp, 10/13-bit fp and 8/13-bit fp.

6.2 Different codec implementations

There are a few test sequences that ETSI provide for the GSM Full Rate codec that can be used to test integer implementations of the codec. The test sequences do not contain any natural sound or speech; instead the samples are created to provoke the integer encoder and decoder implementation. Certain errors that shouldn’t occur with normal samples are exposed with these samples so they can be detected.

To get an integer codec implementation that produces bit exact results, saturation is needed in several parts of the codec as well as scaling to stay within the 16-bit signed integer range. The value 32768 must never occur and has to be handled separately so that it is saturated to 32767. In this project, this feature has been responsible for most of the errors in the integer implementation as the code often appears to be correct, except for this special case.

While the test sequences are valuable when testing the integer codec, they are not equally useful for a floating point implementation. Using 32-bit floating point for the codec, the peculiarities that saturation, truncation and integer adapted scaling brings can be avoided. This and the higher precision creates a different output that will not match the original test sequences, but may sound just as good or better to a listener. A floating point implementation where lower
precision is used needs scaling once again, along with rounding. This brings new effects on the output.

This is why three different implementations of the decoder has been created and used throughout this project. The integer codec is needed to create reference sound files, since the test sequences can not be listened to. The 32-bit floating point decoder is needed to be able to tell which effects that depends on the change from integer to float, and with the 16/23-bit decoder and lower, the effects of the limited precision can be heard.

### 6.3 Changing from integer to float

#### 6.3.1 Sound quality

It should first be said that the original sound samples does not sound particularly good to start with. They are coded in 16-bit PCM, 8kHz mono. Using a floating point implementation of the decoder does not seem to have much of an influence on the sound quality when using higher precision formats. The output is though not bit exact with the integer decoder output, but this can not be heard.

#### 6.3.2 Overflow and saturation

The integer implementation of the decoder has got overflow control and saturation in several places to avoid unwanted values. The floating point implementation is not as sensitive to this and does generally not need to check for overflow until the value is ready to be output and saved. This is possible due to the larger range of the floating point formats. There are though two occasions where overflow control and saturation are needed.

The first occasion is when the values are converted to integers. This is done when the decoder has completed calculating a frame and is ready to output the values, which has to be as 16-bit integers. The function that handles this, op_fint, luckily has got built in overflow control with saturation.

The other occasion is when a 23-bit fp value has to be downsized to a 16-bit fp value. The 23-bit value may be larger than the 16-bit format can hold, and if this occurs, the function op_fround that performs this operation will state that the operation was used on too large a value and then terminate the program. This can though be left unhandled in most of the decoder implementation as it does not occur. There is however in one very rare case that an extreme overflow can occur in the de-emphasis phase and there is even a special test sequence created.
by ETSI to invoke this. So to pass this test sequence, overflow has to be detected and saturated in the de-emphasis part of the decoder.

6.4 Changing to lower precision

Since the high precision floating point formats seem to handle the speech algorithms very well, the question is how low precision that can be used before it becomes obvious to a listener that the sound quality has changed for the worse.

The way that the precision has been changed is by keeping the same number of bits for the exponent so that the range is the same, but decrease the available bits for the mantissa. There is though limits for how few bits that can be used for the internal format and this depend on how large integer values that are taken as input to the decoder.

The largest value from the speech frames that have to be converted to floating point is 63. (The long term lag parameter (Nc) can be up to 120, but is never treated as a float.) To hold this number, 6 bits are needed. When loading the integers, they are scaled and converted to float (See figure 3.5) and the integer must then fit in the mantissa of the floating point format in order not to change the exponent. So the internal format can be 13 bits as lowest, since the mantissa must be 6 bits in this case. For the external format, there is no other limit than that there ought to be at least 1 bit left for the mantissa.

6.4.1 Errors for test sequences

The table below shows the average error in the results of different precision used with the five test sequences. The reference is the output from the 32-bit IEEE-floating point format decoder.

<table>
<thead>
<tr>
<th>Decoder</th>
<th>Average error</th>
<th>% of samples erroneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 / 23-bit</td>
<td>29,4</td>
<td>86,0</td>
</tr>
<tr>
<td>16 / 17-bit</td>
<td>275,3</td>
<td>94,3</td>
</tr>
<tr>
<td>12 / 17-bit</td>
<td>421,7</td>
<td>94,5</td>
</tr>
<tr>
<td>12 / 13-bit</td>
<td>2573,5</td>
<td>95,9</td>
</tr>
<tr>
<td>10 / 17-bit</td>
<td>2597,1</td>
<td>95,1</td>
</tr>
<tr>
<td>10 / 13-bit</td>
<td>2535,6</td>
<td>95,2</td>
</tr>
<tr>
<td>8 / 13-bit</td>
<td>6275,3</td>
<td>96,4</td>
</tr>
</tbody>
</table>

Table 6.1: Average bitwise error with different precision.
Since the test sequences contain samples that change very suddenly and do not follow any “normal” behaviour, it is difficult to make any judgements from these numbers except for that precision decrease for the internal format seems to increase the average error more than a decrease in precision for the external format.

6.4.2 Errors for short speech sample

One of the sound samples that have been used contains 11 seconds of artificial, computer created speech. For this short speech sample, the error levels can be seen in table 6.2. The reference is still the 32-bit IEEE decoder. The average error is smaller here than for the test sequences, which could imply that a “smoother” and voiced sample sequence works better with the lower precision.

<table>
<thead>
<tr>
<th>Decoder</th>
<th>Average error</th>
<th>% of samples erroneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 / 23-bit</td>
<td>4,1</td>
<td>63,1</td>
</tr>
<tr>
<td>16 / 17-bit</td>
<td>33,0</td>
<td>98,7</td>
</tr>
<tr>
<td>12 / 17-bit</td>
<td>51,7</td>
<td>98,8</td>
</tr>
<tr>
<td>12 / 13-bit</td>
<td>343,0</td>
<td>99,4</td>
</tr>
<tr>
<td>10 / 17-bit</td>
<td>342,7</td>
<td>99,2</td>
</tr>
<tr>
<td>10 / 13-bit</td>
<td>370,9</td>
<td>99,4</td>
</tr>
<tr>
<td>8 / 13-bit</td>
<td>1017,1</td>
<td>99,4</td>
</tr>
</tbody>
</table>

Table 6.2: Error levels for the short speech sample.

The error distribution for some of the decoders can be seen in the next three figures. The largest possible error is 65535.

There are a lot of small errors already when this fairly high precision is used, but the total error does not add up to much since no really large errors occur. Largest error is here 94.
When the precision has decreased to 12/17-bits, the errors are starting to become spread over larger errors as well. There are some spikes at 17, 20 and 23 in error which seem a bit odd. These could possibly come from some sort of decision level error that frequently occurs. It should also be taken into consideration that this speech sample is based on an artificial computer voice, which may have an influence if there are “unnatural” repeated parts in the sample sequence. The largest error is here 1625.

The errors are even more spread with 8/13-bit precision. The spikes remain at same position, though with slightly fewer occurrences of those particular errors than with the 12/17-bit decoder. Largest error is 37259.
6.4.3 Errors for long speech sample

The long sample is a natural speech sample with a real voice. There has though been no anti-aliasing filter in use when recording it, so the sample sequence has got a bit of noise and sharp “s” in it. The error levels for this sample sequence can be seen in table 6.3.

<table>
<thead>
<tr>
<th>Decoder</th>
<th>Average error</th>
<th>% of samples erroneous</th>
</tr>
</thead>
<tbody>
<tr>
<td>16 / 23-bit</td>
<td>4,9</td>
<td>91,7</td>
</tr>
<tr>
<td>16 / 17-bit</td>
<td>25,9</td>
<td>98,5</td>
</tr>
<tr>
<td>12 / 17-bit</td>
<td>38,7</td>
<td>98,6</td>
</tr>
<tr>
<td>12 / 13-bit</td>
<td>353,0</td>
<td>99,8</td>
</tr>
<tr>
<td>10 / 17-bit</td>
<td>217,9</td>
<td>99,2</td>
</tr>
<tr>
<td>10 / 13-bit</td>
<td>354,3</td>
<td>99,8</td>
</tr>
<tr>
<td>8 / 13-bit</td>
<td>794,5</td>
<td>99,8</td>
</tr>
</tbody>
</table>

*Table 6.3: Error levels for the long speech sample.*

The errors are restricted to small errors only, just as for the short sample sequence. The largest error is here 170, a bit more than for the short sample sequence.
The errors are starting to spread out a bit towards larger errors, but are still quite concentrated to the lower ones. There are no spikes here as there were for the short sample sequence. The largest error is 2357.

The errors are now spread all over the range and the largest error is 45121. The average error is though smaller for this long sample sequence than for the short sample sequence.
6.5 Performance

The most calculation intense part of the decoder is the short term synthesis. It is responsible for approximately 74-78% of execution time. The reason for this is that all eight reflection coefficients have to be applied to every sample, which results in that the calculations in this loop are done 1280 times per frame. A frame is 20 ms of speech, which means that the short term synthesis loop is run 64000 times per second. The rest of the program has the computation time distributed fairly even. The only part that accounts for slightly more computation time than the rest is when the excitation sample array (ep) has to be initialised and filled.

Looking at the individual floating point operations, addition takes the most time in total and is the most commonly performed operation. (The function for addition is also used by the function for subtracting). Table 6.4 lists the number of times the operations are carried out per speech frame, per second of speech and how many percent of the calculation time they approximately account for.

<table>
<thead>
<tr>
<th>Function</th>
<th>Calls/frame</th>
<th>Calls/sec of speech</th>
<th>Avg. % of total runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>op_fadd</td>
<td>3579</td>
<td>178950</td>
<td>69.5%</td>
</tr>
<tr>
<td>op_fmul</td>
<td>2990</td>
<td>149517</td>
<td>12.2%</td>
</tr>
<tr>
<td>op_fround</td>
<td>1837</td>
<td>91850</td>
<td>3.7%</td>
</tr>
<tr>
<td>op_fexpand</td>
<td>2009</td>
<td>100450</td>
<td>2.8%</td>
</tr>
<tr>
<td>op_fsub</td>
<td>1465</td>
<td>73233</td>
<td>0.7%</td>
</tr>
<tr>
<td>op_fint</td>
<td>160</td>
<td>8000</td>
<td>0.2%</td>
</tr>
<tr>
<td>signextend</td>
<td>14976</td>
<td>748780</td>
<td>5.3%</td>
</tr>
<tr>
<td>other</td>
<td>-</td>
<td>-</td>
<td>5.5%</td>
</tr>
</tbody>
</table>

Table 6.4: Performance properties for different operations.

The number of times the functions are called is an average from the five test sequences and a shorter speech sample of 11 seconds. The average percentage of the total runtime was instead measured several times with the longer sample to get a more accurate profile analysis than the shorter samples gives. The long sample contains almost 2 minutes of speech. Note that the function “signextend” is included in the table, which is used by op_fround, op_fmul and op_fadd.
7 Results

The GSM full rate decoder has successfully been implemented in C code, with the restrictions that comes with the floating point formats. All variables and constants use only the 16-bit format when stored in memory and are expanded to the 23-bit format when they are loaded into the register.

Scaling has been avoided as far as possible, but with the limited range of the 16-bit format some scaling is always needed. The bias of $2^{-11}$ increases the number of scaling operations since values otherwise gets infinitely small when multiplied with each other.

The sound quality does not seem to suffer much from the limited precision. Since there is less clipping than the integer codec needs to avoid too large values, the sound can at times be perceived as softer when high precision formats are used. The GSM full rate codec does however not produce great sound to start with, so it can be difficult to hear small differences between the original implementation and the floating point implementation when the precision is fairly high.

When lowering the precision of the floating point format, the sound degradation is obvious when the external precision is down to 8-bits. The mantissa has only two bits at that stage, but the speech is still intelligible even though it sounds bad. The internal precision was lowered to 13-bits which increased the average error on the output samples, but this did not impact the sound quality much.
8 Future work

The decoder has so far only been simulated on a PC. It would be interesting to examine how the implementation works on the real DSP. To get a complete decoder for GSM packets the scope could be expanded to also include the assisting functions, such as muting, comfort noise, substitution of lost frames etc. To get a complete codec, the Full Rate encoder could also be implemented.

Another interesting follow up would be to adapt other codecs for the DSP and compare how they perform. Especially an implementation of the EFR codec as it is supposed to give better sound quality and is the most common codec in the applications today. Also the wideband AMR codec which uses higher sample rates could be interesting as it may suffer more from the limitations of the low precision format.
9 References


10 Abbreviations and explanations

3G – Third Generation
A-law – Logarithmic wave form codec used in the regular phone network where the data is digital (such as ISDN). In the USA, μ-law is used instead of A-law, but the differences are small between these codecs.
AbS – Analysis by Synthesis
ACELP – Algebraic Code Excited Linear Prediction
ADPCM – Adaptive Differential Pulse Code Modulation
AMR – Adaptive Multi Rate, a newer speech codec in the GSM and 3G networks.
APCM – Adaptive Pulse Code Modulation.
CELP – Code Excited Linear Prediction
DPCM – Differential Pulse Code Modulation
DSP – Digital Signal Processor
DTX – Discontinuous Transmission
EFR – Enhanced Full Rate, a newer speech codec in the GSM network.
ETSI – European Telecommunications Standard Institute, an organisation that maintains the standards and documentation for the GSM and 3G networks.
FIR – Finite Impulse Response
FR – Full Rate
GSM – Global System for Mobile Communication
HR – Half Rate
IEEE – Institute of Electrical and Electronics Engineers
IP – Internet Protocol
ISDN – Integrated Services Digital Network, a digital data transmission standard used in the regular phone network.
ITU – Internation Telecommunication Union
LAR – Log Area Ratios
LPC – Linear Predictive Coding
LSB – Least Significant Bit
LTP – Long Term Prediction
MPE – Multi Pulse Excited, the excitation samples have variable time intervals between them.
MR-ACELP – Multi Rate - Algebraic Code Excited Linear Prediction
PCM – Pulse Code Modulation
RPE – Regular Pulse Excited, the excitation samples have same time interval between them.
RPE-LTP – Regular Pulse Excited - Long Term Prediction
VAD – Voice Activity Detection
VSELP – Vector Sum Excited Linear Prediction

Pharynx – This is the area far back in the mouth, where the throat separates into the mouth cavity and the nose cavity. (Laryngopharynx – lower throat, Oropharynx – middle part of the throat leading into the mouth, Nasopharynx – the upper part of the throat leading into the nose cavity)
Postalveolar – This is generally the area in the mouth that can be touched with the tongue.