MEE04:26

G.722 WIDEBAND SPEECH CODEC IMPLEMENTATION ON BF-533 DSP

By

Avi Peretz Cagdas Gumusoglu

Master's Thesis
Department of Telecommunications and Signal Processing
Blekinge Institute of Technology, Sweden

Abstract

The technological developments in digital communication systems increase the usable bandwidth of sound signals which results in increased intelligibility and naturalness of the signal. The emerging digital communication systems enable the use of wideband speech codec in a wide area of applications. Recognizing the need of high quality wide band speech codec, several standardization activities have been recently conducted. G.722 is one of the first wideband speech codec standards implemented in the telecommunication systems. In this thesis, after examining the G.722 wideband speech codec theoretically, a real time implementation of G.722 wideband speech codec has been developed on Blackfin BF533 DSP development kit from Analog Devices. Also, G.722 Codec is optimized by using assembler for BF533. As a result to the thesis, very good speech performance is obtained in real time implementation of G.722 and optimization increases the efficiency of Blackfin BF533 DSP drastically.

Contents

Chapter 1

Introduction

The topic for this thesis is "Implementation of G.722 Wideband Speech Codec on a BF533 DSP".In most codecs used today, the bandwidth is limited to approximately 200-3,400 HZ because of the Public Service Telephone Networks(PSTN)standards. Wideband Speech Codecs extend the useable pass band to 50-7000 Hz and are targeted at applications that do not directly interoperate with the legacy digital PSTN. This gives a much richer sound to the voice conversation, making the speech more intelligible and better speaker recognition possible. Also the transmission of better quality audio signals (music, etc.) becomes possible.

Application areas of wideband speech coding include:

- Audio and video conferencing.
- Digital AM radio broadcasting.
- High-fidelity telephony transmitted over cables or fiber optic cables.
- Dual language programming in audio/video broadcasts of news, TV programs, etc.

This thesis will focus on working principles of G.722 Wideband Speech Codec. The goals of this thesis are as follows :

- To examine principles of G.722 Wideband Speech Codec
- To develop a real time implementation of G.722 codec on BF533 Blackfin DSP Development kit by using C programming language
- To Optimize the implementation code
- To test the performance of G.722 implementation

This thesis is divided into a number of sections :

Chapter 2 gives some basic information about speech coding theory and describes some different types of codecs.

Chapter 3 studies the G.722 Wideband Speech Codec as well as the development and derivation of algorithms used within the thesis.

Chapter 4 gives some information about the Anolog Devices Blackfin BF533 DSP Development kit.

Chapter 5 details the source code used in the real time implementation of G.722 Wideband Speech Coding system, further it provides the optimization and test results.

Chapter 6 concludes the thesis and provides recommendations for future research and development.

Appendix lists the source code used in real time implementation.

Chapter 2

Speech Coding

2.1 Basic Properties Of Speech

Speech is produced by the excitation of anacoustic tube, the vocal tract, which is in average 16 cm long and located between the lips and glottis. Speech sounds can be put into three basic classes, [1] :

Voice Sounds are produced by exciting the vocal tract with quasi-periodic pulses. Voice sounds show a high degree of periodicity at the pitch period, between 2ms and 20ms.

Fricative Sounds are produced by forcing air at high velocities through the constriction in the vocal tract, so that a noise like excitation is created. Such sounds show little long-term periodicity.

Plosive Sounds are produced when a complete closure is made in the vocal tract, building up pressure behind the closure, then released suddenly.

The shape of the vocal tract and its mode of excitation change relatively slowly, thus speech can be considered to be quasi-stationary over short periods of time (of the order of 20 ms). Speech signals show a high degree of predictability, due to the quasi-periodic vibrations of the vocal cords and also due to the resonances of the vocal tract. Most of the Speech codecs attempt to exploit this predictability in order to reduce the data rate necessary for good quality voice transmission. We will focus on waveform codecs in the following section.

2.2 Waveform Codecs

Waveform codecs are used at high bit rates and gives very good quality of speech. Waveform codecs attempt, without any prior knowledge of how the signal to be coded was generated, to produce a reconstructed signal whose waveform is as close as possible to the original. Some of the waveform codecs are provided by International Telecommunication Union Standardization Sector (ITU), e.g. G.711, G.721, G.722, G.726 and G.727.

2.2.1 PCM Waveform Codecs

Pulse Code Modulation (PCM) is a digital scheme for transmitting analog data and it is the simplest form of Waveform Coding. PCM is composed of two main parts : Sampling and Quantization. Narrowband signal is sampled by 8 kHz and and then each sample is quantized. The quantizer is memoryless and non-linear in PCM codecs. Non-linear quantizers are used to have lower bit rates. G.711 codec, which operates at 64 kbits/s, is a well-known example of a PCM codec.

2.2.2 DPCM Waveform Codecs

In speech codecs, one of the most commonly used techniques is to predict the value of the next sample from previous values by using the present correlation between speech signals. (Main reasons for the correlation of speech signals are discussed in section 2.1) If the predictions are effective then the error signal between the predicted samples and original speech samples will have a lower variance than the original signal, [4]. Therefore, it is possible to quantize the difference signal with fewer bits than the original speech signal. In short, to quantize the difference signal between the original and predicted samples is the basis of Differential Pulse Code Modulation (DPCM) codecs. DPCM codecs can be improved by using adaptive predictor and quantizers. In the following section, we will explain Adaptive Differential Pulse Code Modulation (ADPCM).

2.2.3 ADPCM Waveform Codecs

Adaptive DPCM is a variant of DPCM that varies the size of the quantization step, to allow further reduction of the required bandwidth. The performance of the codec is aided by using adaptive prediction and quantization, so that the predictor and difference quantizer adapt to the changing characteristics of the speech being coded. Some of the ADPCM codecs which are provided by ITU are: G.721, G.722, G.726 and G.727.

In the mid 1980s, the 32 kbits/s ADPCM codec, G.721, was standardized which gave reconstructed speech almost as good as the 64 kbits/s PCM codecs. Later, G.726 and G.727 codecs operating at 40, 32, 24 and 16 kbits/s were standardized.

2.2.4 Subband ADPCM Waveform Codecs

The waveform codecs described above all code signals with an entirely time domain approach. Frequency domain approaches are also possible, and have certain advantages. In Sub-Band Coding (SBC) the input signal is split into a number of frequency bands, or sub-bands, and each is coded independently using for example an ADPCM like coder, [4]. At the receiver the sub-band signals are decoded and recombined to give the reconstructed speech signal. The advantages of doing this come from the fact that the noise in each sub-band is dependent only on the coding used in that sub-band. Therefore we can allocate more bits to perceptually important sub-bands so that the noise in these frequency regions is low, while in other sub-bands we may be content to allow a high

coding noise because noise at these frequencies is less perceptually important. Due to the filtering necessary to split the signal into sub-bands they are more complex than simple DPCM coders, and introduce more coding delay.

G.722 is one of the most common SubBand ADPCM Waveform Codecs. The details of the G.722 codec will be explained in Chapter 3 and the implementation details of the G.722 codec are given at Chapter 5.

Chapter 3

G.722 Standard

3.1 Introduction to G.722

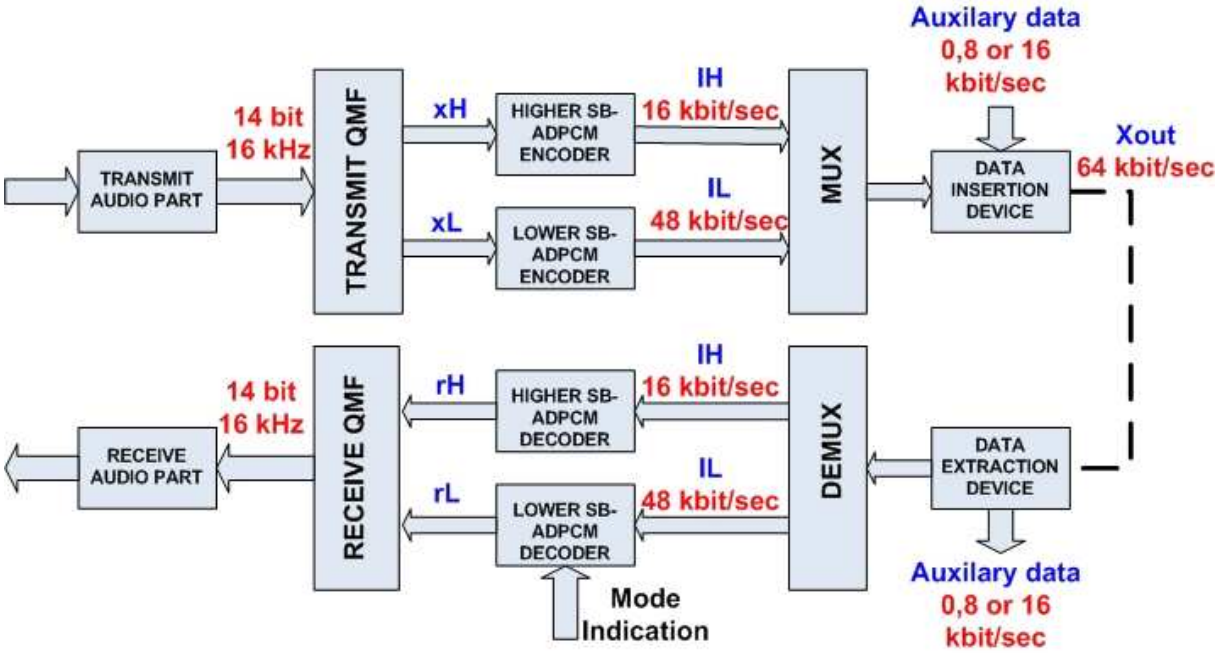

Figure 3.1: The Block Diagram of G.722 WideBand Speech Codec.

Standardized in 1988, the ITU G.722 was the first standardized wideband speech coding algorithm to be used at a 16 kHz sampling rate. The G.722 codec takes 14-bit data at 16 kHz (bandwidth from 50 Hz to 7 kHz) and compresses from 224 kbit/s to 64, 56, and 48 kbit/s according to the mode selected. The coding system uses Sub-Band Adaptive Differential Pulse code modulation (SB-ADPCM) within a bit rate of 64 kbit/s. G.722 encoder uses 64 kbit/s at all times irrespective of mode of operation but the G.722 decoder uses 64, 56, or 48 kbit/s depending on the mode of operation.

Fig. 3.1 illustrates the block diagram of G.722 Codec. The main functional parts of the 64 kbit/s (7 kHz) audio codec are as follows:

64 kbit/s audio encoder

- Transmit audio part which converts an audio signal to a uniform digital signal which is coded using 14 bits with 16 kHz sampling (224 kbit/s)
- SB-ADPCM encoder which reduces the bit rate to 64 kbit/s (G.722 encoder uses 64 kbit/s at all times irrespective of mode of operation)
- Data Insertion Device which uses the least significant bit or two least significant bit of the encoded signal as auxiliary data channel

64, 56, or 48 kbits audio decoder

- Data Extraction Device which extracts the auxiliary data from the encoded audio data at the receiver side
- SB-ADPCM decoder which performs the reverse operation to the encoder (The effective audio decoding bit rate of the decoder can be 64, 56 or 48 kbit/s depending on the mode of operation)
- Receive audio part which reconstructs the audio signal from the uniform digital signal which is encoded using 14 bits with 16 kHz sampling

3.2 Transmit Audio Part

Transmit Audio Part is composed of

- Input level adjustment device
- Input anti-aliasing filter
- Sampling device operating at 16 kHz
- Analog-to-uniform digital converter with 14 bits and with 16 kHz sampling

3.3 SB-ADPCM Encoder

Fig. 3.2 is a block diagram of SB-ADPCM Encoder. The components of SB-ADPCM encoder are as follows:

- Transmit Quadrature Mirror Filters
- Lower Subband ADPCM Encoder
- Higher Subband ADPCM Encoder
- Multiplexers

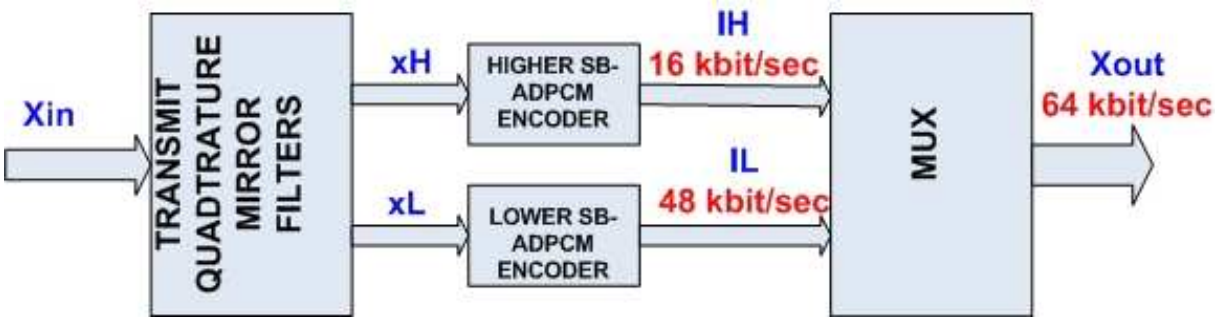

Figure 3.2: The Block Diagram of G.722 SB-ADPCM Encoder.

3.3.1 Transmit Quadrature Mirror Filters

The transmit QMFs comprise two linear-phase finite impulse response digital filters which split the frequency band 0 to 8000 Hz into two sub-bands: the lower sub-band (0 to 4000 Hz) and the higher sub-band (4000 to 8000 Hz). Fig. 3.2 shows the input and outputs to the Transmit QMF. The input to the transmit QMFs, xin , is the output from the transmit audio part which was sampled at 16 kHz. The outputs, xL and xH, for the lower and higher sub-bands respectively, are decimated to 8 kHz.

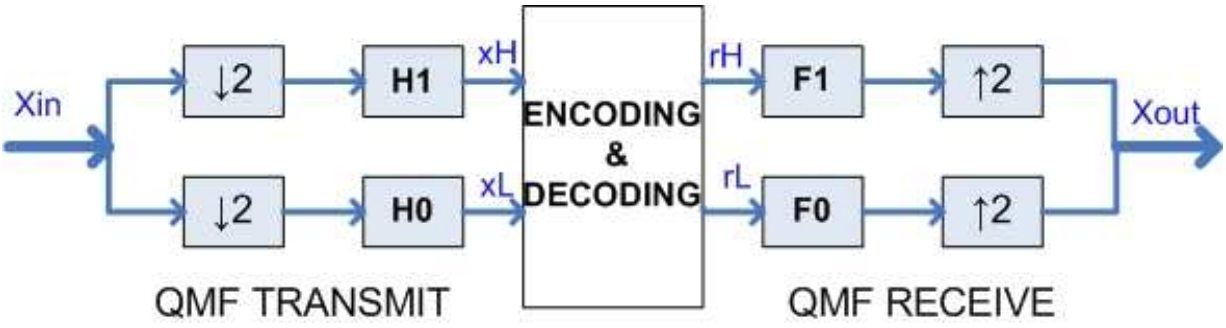

Figure 3.3: The Block Diagram of Quadrature Mirror Filters (QMF)

Fig. 3.3 illustrates a block diagram of QMF. H0 is the low PASS filter and H1 is the highpass filter with cutoff frequency 4 kHz at the transmitter side. H0=h0(n) is a 24 tap filter described in time domain. All the other filters (H1,F1,F0) are derived from H0. H1 is the shifted version of H0 in the frequency domain by $\Pi/2$.

$$H1(\omega) = H0(\omega - \Pi/2) \tag{3.1}$$

which is in time domain

$$h1(n) = (-1)^n \times h0(n) \tag{3.2}$$

The low pass reconstruction filter, F0 , at the receiver side is equal to H0.

$$f0(n) = h0(n) \tag{3.3}$$

The high pass reconstruction filter, F1 , is equal to the negative of H1.

$$f1(n) = -h1(n) \tag{3.4}$$

The coefficients of the filter, H0 , are stored as an array in the program. The scaled values of filter, h0(n) , are given at Table. 3.1. These coefficients will be used in the C code for the implementation.

Filter Coefficients	Scaled Values
h0(0), h0(23)	3
h0(1), h0(22)	-11
h0(2), h0(21)	-11
h0(3), h0(20)	53
h0(4), h0(19)	12
h0(5), h0(18)	-156
h0(6), h0(17)	32
h0(7), h0(16)	362
h0(8), h0(15)	-210
h0(9), h0(14)	805
h0(10), h0(13)	951
h0(11), h0(12)	3876

Table 3.1: Quadrature Mirror Filter Coefficients

3.3.2 Lower Subband ADPCM Encoder

Fig. 3.4 is a block diagram of Lower Subband ADPCM Encoder. Three main blocks of Lower Subband ADPCM Encoder are as follows:

- Adaptive Quantizer
- Inverse Adaptive Quantizer
- Adaptive Predictor

An estimate, sL, of the the lower sub-band input signal, xL, is subtracted from the input signal, xL, and the difference signal, eL, is produced. An adaptive 60-level nonlinear quantizer is used to assign six binary digits to the value of the difference signal to produce a 48 kbit/s signal, IL.

In the feedback loop, the two least significant bits of IL are deleted to produce a 4-bit signal ILt. ILt is used for the quantizer adaptation and applied to a 15-level inverse adaptive quantizer to produce a quantized difference signal, dLt. The signal estimate, sL is added to this quantized difference signal to produce a reconstructed version, rLt , of the lower sub-band input signal. Both the reconstructed signal and the quantized difference signal are operated upon by an adaptive predictor which produce the estimate, sL , of the input signal, thereby completing the feedback loop.

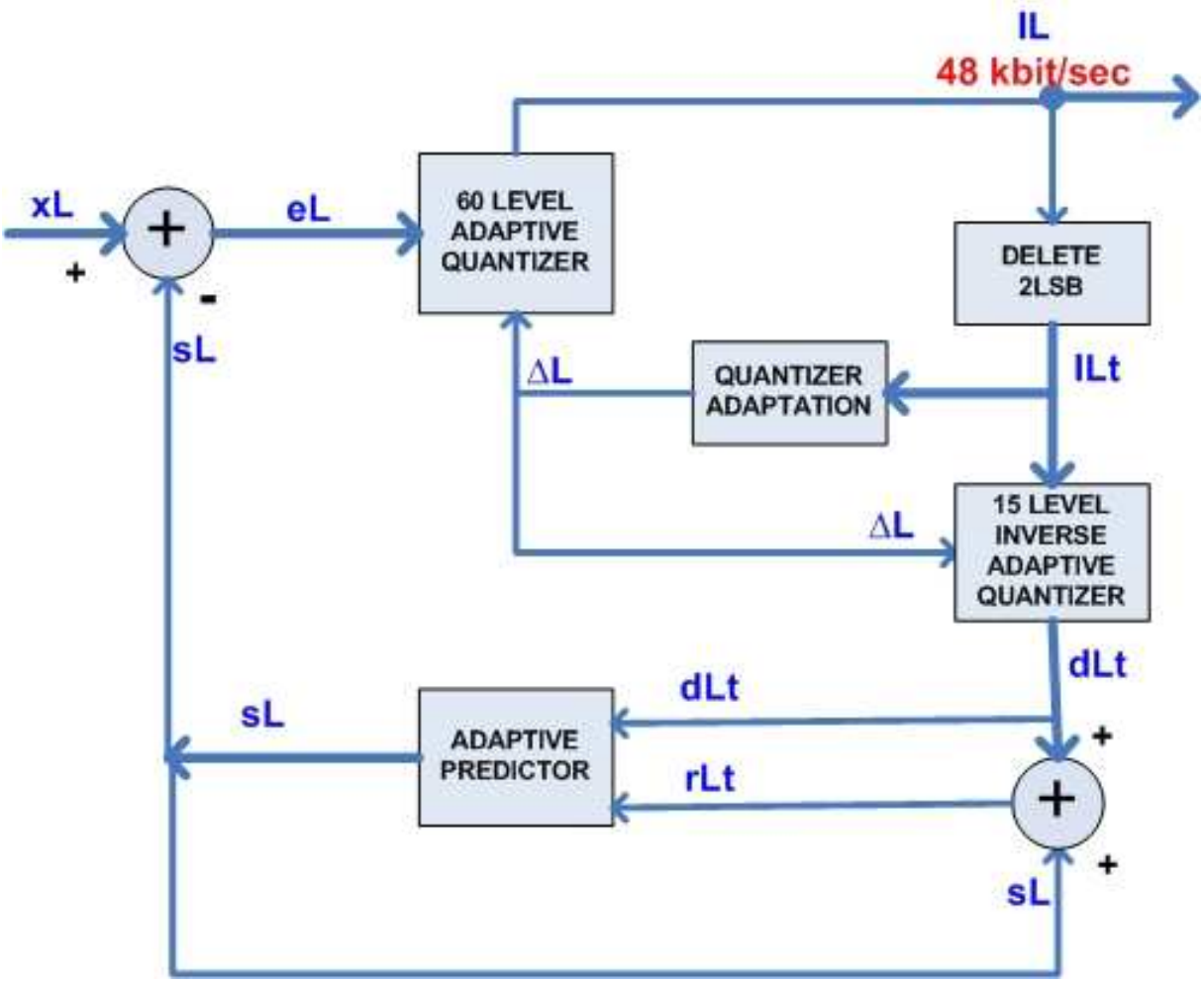

Figure 3.4: The Block Diagram of Lower Subband ADPCM Encoder

3.3.3 Higher Subband ADPCM Encoder

Fig. 3.5 is a block diagram of the higher sub-band ADPCM encoder. Three main blocks of Higher Subband ADPCM Encoder are same as the Lower Subband ADPCM Encoder and can be summarized as follows:

- Adaptive Quantizer
- Inverse Adaptive Quantizer
- Adaptive Predictor

An estimate, sH, of the input signal is subtracted from the higher sub-band input signal, xH and the difference signal, eH, is produced. An adaptive 4-level nonlinear quantizer is used to assign two binary digits to the value of the difference signal to produce a 16 kbit/s signal, IH.

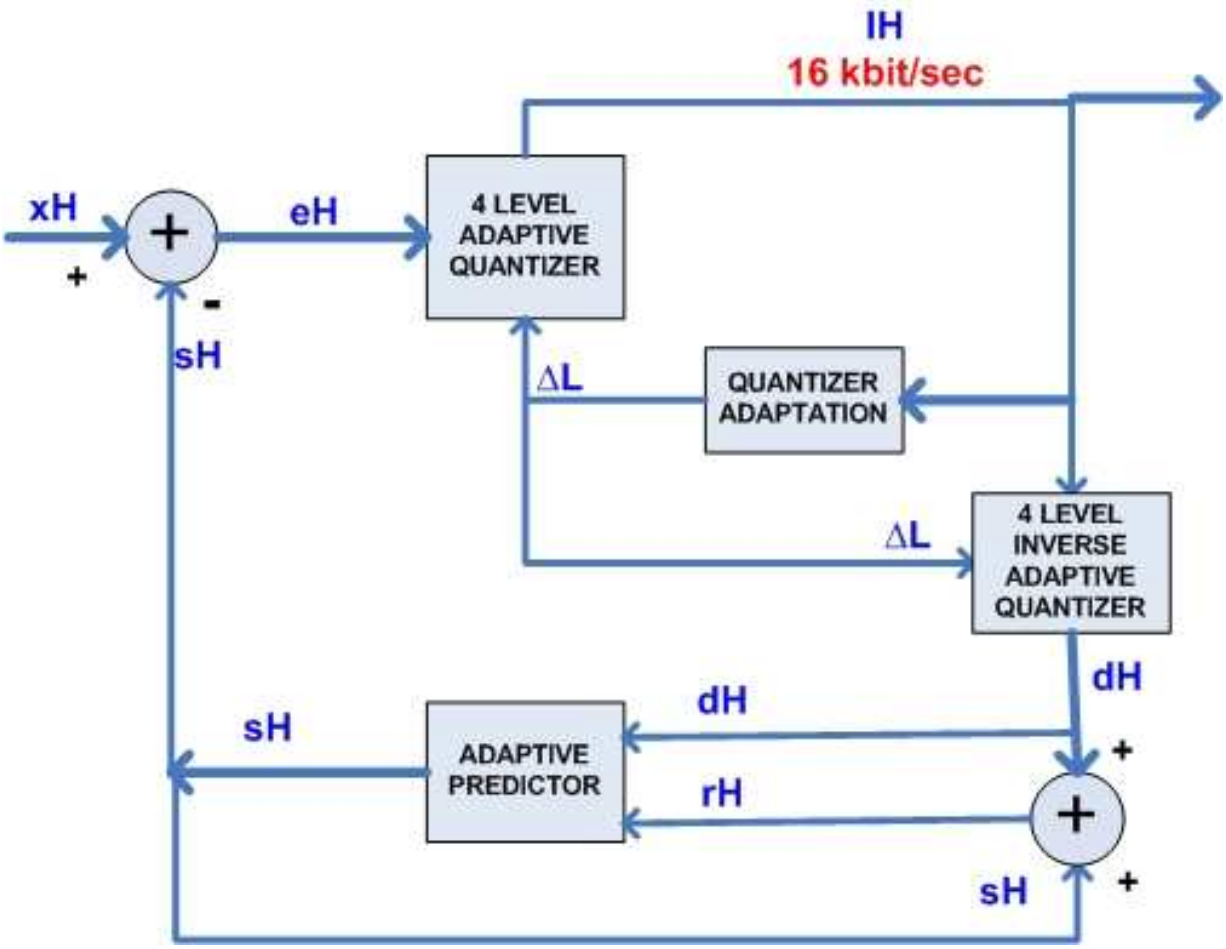

Figure 3.5: The Block Diagram of Higher Subband ADPCM Encoder

An inverse adaptive quantizer produces a quantized difference signal, dH, from these same two binary digits. The signal estimate, sH, is added to this quantized difference signal to produce a reconstructed version, rH, of the higher sub-band input signal. Both the reconstructed signal and the quantized difference signal are operated upon by an adaptive predictor which produces the estimate, sH, of the input signal, thereby completing the feedback loop.

3.3.4 Multiplexer

The Multiplexer (MUX) shown in Fig. 3.2 is used to combine the signals, IL and IH, from the lower and higher sub-band ADPCM encoders respectively. IL is a 6 bit ADPCM Codeword and IH is a 2 bit ADPCM Codeword. The output octet format, after multiplexing, is as follows:

IH1 IH2 IL1 IL2 IL3 IL4 IL5 IL6

where IH1 is the first bit transmitted, and where IH1 and IL1 are the most significant bits of IH and IL respectively, while IH2 and IL6 are the least significant bits of IH and IL respectively.

3.4 Data Insertion Device

The Data Insertion Device is only needed for the applications which requires an auxiliary data channel within the 64 kbits/sec. The three possible modes of operations and the respective audio decoding rates of decoder are shown in Table. 3.2.

Mode	Audio Decoding Bit Rate Kbps	Auxiliary Data Channel Bit Rate Kbps
1	64	0
2	56	8
3	48	16

Table 3.2: Possible Mode of Operations

3.5 Data Extraction Device

Data Extraction Device, like the Data Insertion Device, is only needed for applications requiring an auxiliary data channel within the 64 kbits/sec.

Data Extraction Device at the receiver side determines the mode of operation and extracts the appropriate data bits. If auxiliary data channel is used at the transmitter side, Data Extraction device separates the audio data and auxiliary data at the receiver side.

3.6 SB-ADPCM Decoder

Fig. 3.6 is a block diagram of SB-ADPCM Decoder. The components of SB-ADPCM decoder are as follows:

- Demultiplexer
- Lower Subband ADPCM Decoder
- Higher Subband ADPCM Decoder
- Receive Quadrature Mirror Filters

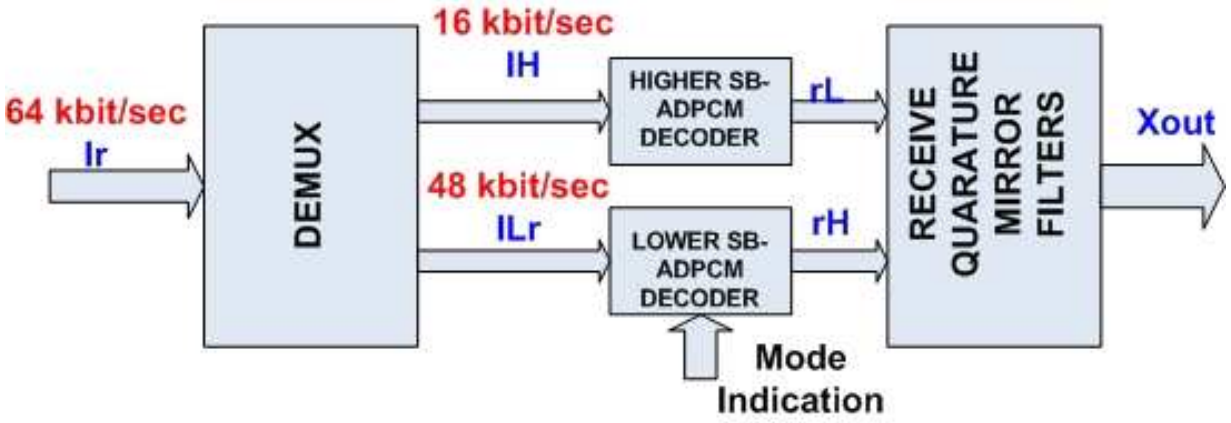

Figure 3.6: The Block Diagram of G.722 SB-ADPCM Decoder.

3.6.1 Demultiplexer

The Demultiplexer (DEMUX) shown in Fig. 3.6 is used to decompose the signal, Ir, received by the decoder at 64 kbit/sec into two signals such as IH and ILr. One of the output signals of DEMUX, IH, is the input for the Lower Subband ADPCM decoder, and the other output signal, ILr, is the input for the Higher Subband ADPCM decoder. The signal IH is a 2 bit ADPCM Codeword and the signal ILr is 6 bit ADPCM Codeword. All the input and output signals of DEMUX are sampled at 8 kHz.

3.6.2 Lower Subband ADPCM Decoder

Fig. 3.7 is a block diagram of Lower Subband ADPCM Decoder. This decoder can operate in three different modes of operations according to the desired audio decoding bit rate. These three different modes are described in Section 3.4.

The output of the Lower Subband ADPCM Decoder, the reconstructed signal, rL, is produced by adding the estimate, sL , of the input signal to the one of the quantized difference signals dLt (dL6 ,dL5 or dL4) according to the indicated mode of operation. For each mode of operation, different quantized difference signals are selected, different inverse adaptive quantizers are used and different number of least significant bits are deleted from the input codeword. Table. 3.3 shows the differences of decoder for each mode of operation.

By deleting two least significant bits of the input signal, ILr, to the Decoder, the signal, ILt, which is the input signal to the Quantizer Adaptation and to the 15-bit inverse Adaptive quantizer is obtained. The derivation of the estimate signal, sL, from the signal, ILt,is identical to the feedback loop of the Lower Subband ADPCM Encoder described in subsection 3.3.2.

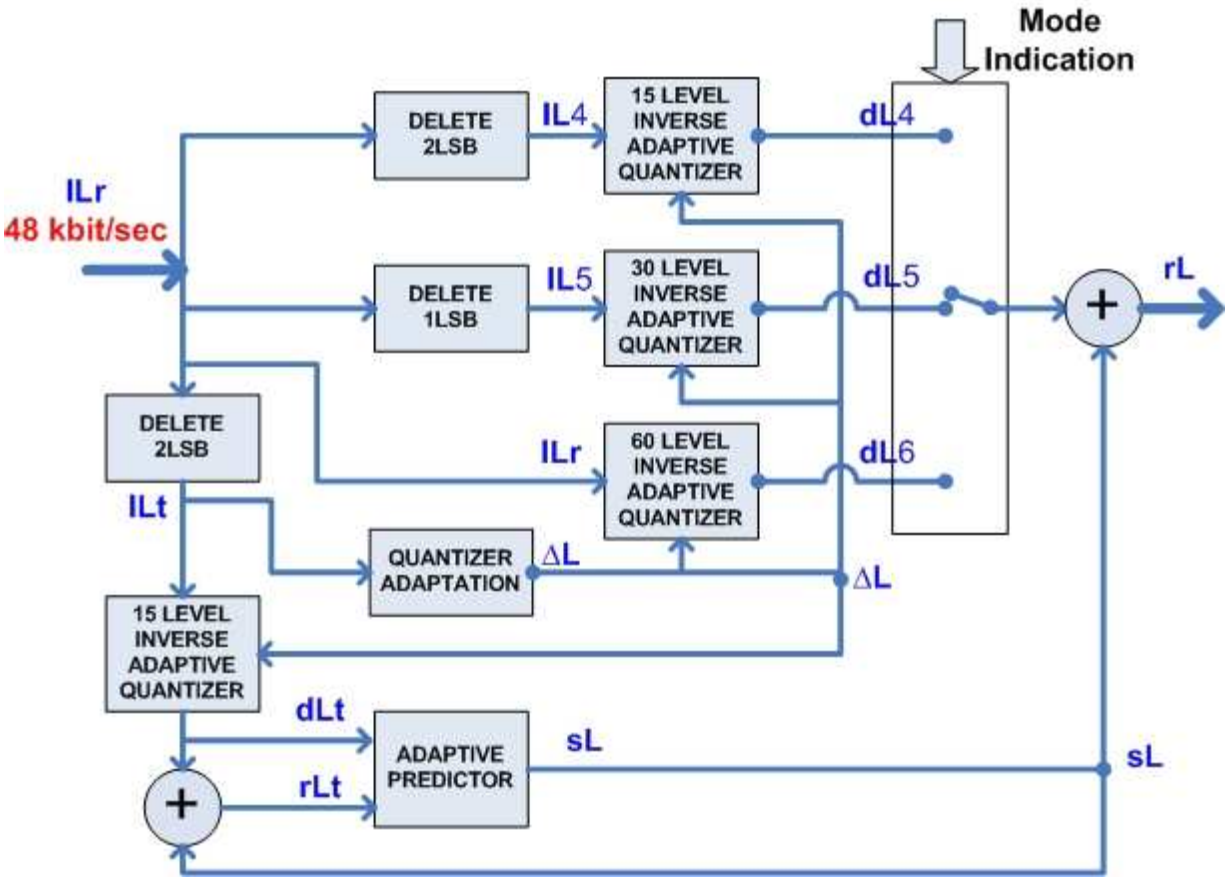

Figure 3.7: The Block Diagram of G.722 Lower Subband ADPCM Decoder.

3.6.3 Higher Subband ADPCM Decoder

Fig. 3.8 is a block diagram of Higher Subband ADPCM Decoder. The input signal, IH, to the decoder, is operated upon the inverse adaptive quantizer and the difference signal, dH, is produced. The output of the decoder, the reconstructed signal, rH, is produced by the addition of difference signal, dH, to the estimate signal, sH. The adaptive predictor produces the estimate signal, sH.

3.6.4 Receive Quadrature Mirror Filters

Fig. 3.6 shows the inputs and output to the Receive QMF. The Receive Quadrature Mirror Filters (Receive QMFs) comprise two linear-phase finite impulse response digital filters which is used to interpolate the signals from 8 kHz to 16 kHz. Receive QMFs transforms two output signals of the decoders, rL and rH, which are sampled at 8kHz, into an output signal, Xout, which is sampled at 16KHz.

Mode Of Operation	Selected Quantized Difference Signal	Inverse Adaptive Quantizer Used	Number of Least Significant Bit Deleted
Mode 1	dL6	60 Level	0
Mode 2	dL5	30 Level	1
Mode 3	dL4	15 Level	2

Table 3.3: Differences of Decoder for each Mode of Operation

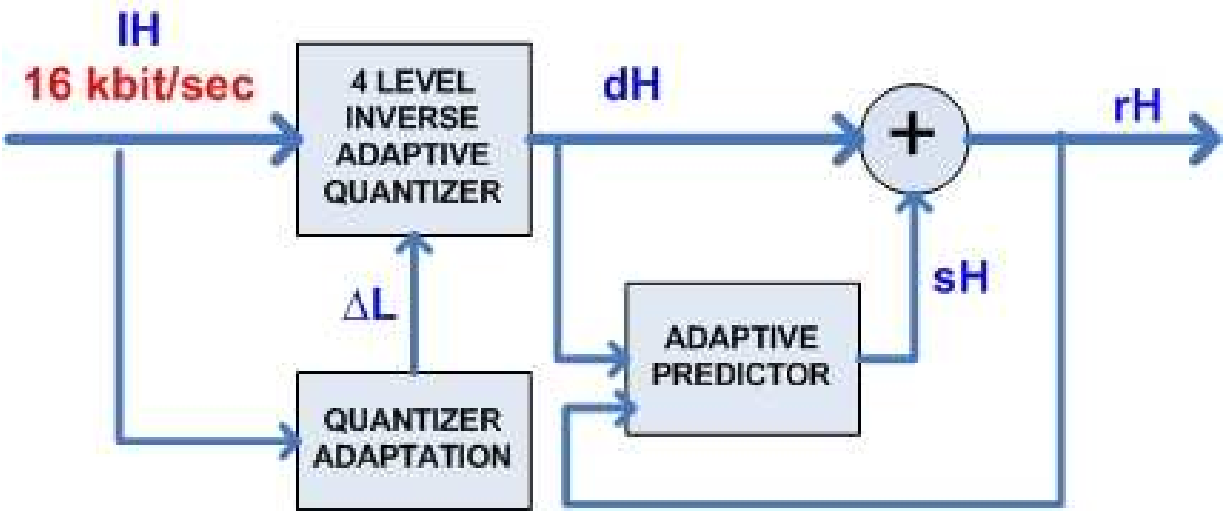

Figure 3.8: The Block Diagram of G.722 Higher Subband ADPCM Decoder.

3.7 Receive Audio Part

Receive Audio Part is composed of

- A uniform digital to anolog converter with 14bits at 16kHz sampling
- A reconstruction filter which includes x/sinx correction
- An output level adjustment device

Chapter 4

The BF-533 DSP

The BF-533 is a fixed-point digital signal processor which was developed by Analog Devices. The BF-533 is a member of the Blackfin family but offers much better performance with a low power consumption compare with the previous Blackfin processors.

Main features

- Clock Speed of 750MHz
- Dual 16-Bit multiplication-accumulation (MAC)
- Two 40-Bit Arithmetic Logic units(ALU)
- Four 8-Bit Video ALUs
- 40-Bit shifter
- Risc like Register and Instruction Model

In addition it offers a friendly and easy to use compiler support such as: Advanced Debug,trace and Performance-Monitoring.

The BF-533 processor system includes also peripherals as shown in figure 4.1. These peripherals are connected to the core of the processor via several high bandwidth buses.

Some of these important peripherals are described below:

Watchdog Timer:

This is a 32-bit timer that can be used to implement a software watchdog function. A watchdog timer is a piece of hardware built into the processor that causes the processor i.e. reset or interrupt when it judges that the system has hung, or is no longer executing the correct sequence of code. The watchdog timer is initialized and enabled by the programmer.

Real-time clock:

The Real-Time Clock (RTC) provides a robust set of digital watch features such as: stopwatch, current time and alarm and is clocked by a 32.768 kHz crystal external to the processor. The RTC has a separate power supply that is isolated from the rest of the DSP. This means that even when the DSP is in low power state, the RTC remain powered to preserve the current time and calendar information. The RTC can wake up the processor from Sleep mode or Deep Sleep mode by generating an RTC wake-up event.

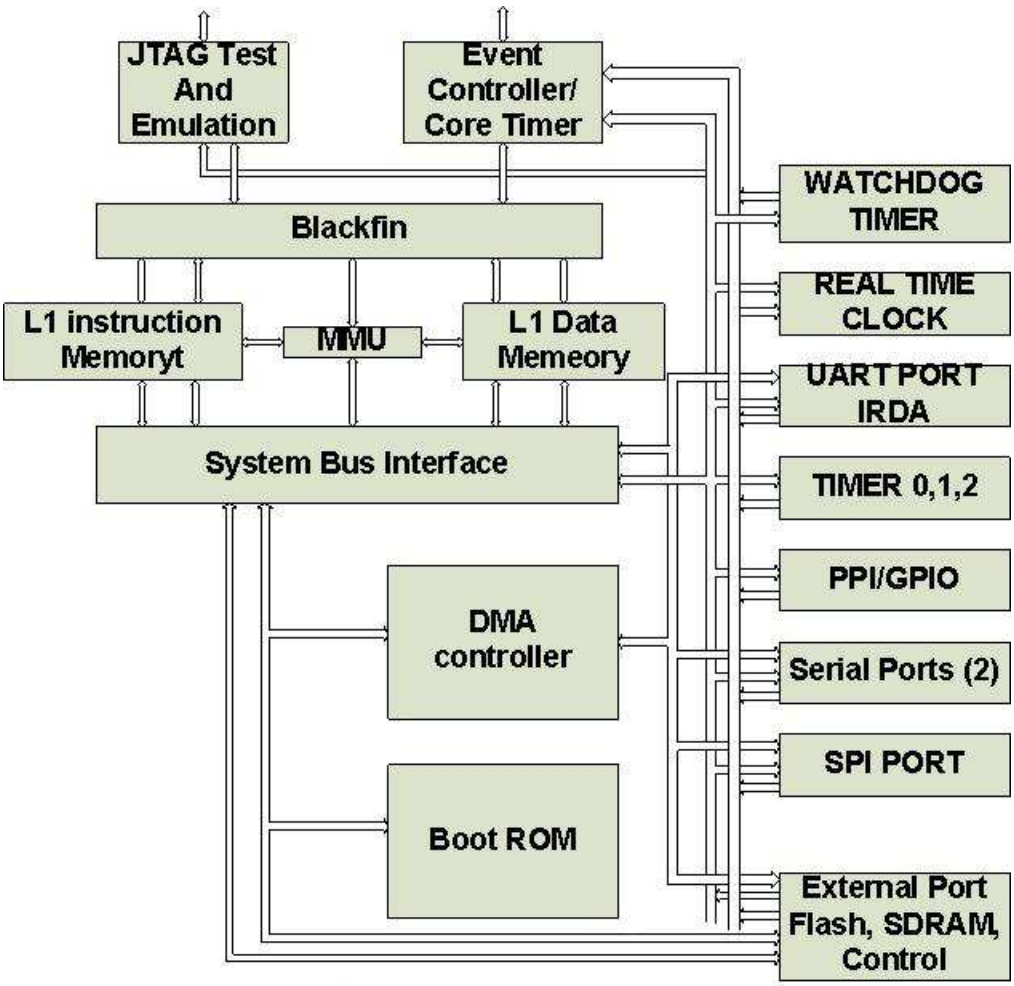

Figure 4.1: BF-533 Block Diagram

Serial Ports:

The processor has two dual-channel synchronous serial ports (SPORT0 and SPORT1) for serial and multiprocessor communications. Some main features of these SPORTs are:

- Capable to operate as a bidirectional I2S serial bus. Each SPORT has two sets of independent transmit and receive pins, enabling eight channels of I2S stereo audio.
- Each transmit and receive port can used either as an external serial clock or can generate its own clock in a wide range of frequencies.
- Each SPORT can be set to data word lengths from 3 to 32 bits. And can be set to transfer the most significant bit(MSB) first or the Least significant bit (LSB) first transferred in most significant bit first or least significant bit first format.
- The SPORTs can generates an interrupt once completing the transfer of a data word or after transferring an entire data buffer or buffers through DMA.
- Each SPORT can operate according to ITU recommendation G.711.

Serial Peripheral Interface (SPI) Port:

Serial peripheral interface (SPI) is an interface that enables the serial (one bit at a time) exchange of data between two devices, one device called a master and the other device called a slave. The SPI operates in full duplex mode which means that data can be transferred in both directions at the same time. The select input pin (SPISS) lets other SPI devices select the processor, and seven SPI (SPISEL 1-7) select output pins let the processor select other SPI devices.

Direct Memory Access Controller (DMA):

Direct memory access (DMA) is a means of having a peripheral device control a processor's memory bus directly. DMA permits the peripheral, such as a UART, to transfer data directly to or from memory without having each byte (or word) handled by the processor. DMA transfers can occur between the internal memories and any of processor's DMA-capable peripherals. In addition DMA transfer can accurse between any DMA-capable peripheral and external devices which are connected to the external memory interfaces such as SDRAM controller and the asynchronous memory controller. DMA-capable peripherals include the SPORTs, SPI port, UART, and PPI. Each DMA-capable peripheral has at least one dedicated DMA channel.

The DMA controller is a device that control these DMA transfers. Upon a request for DMA transfer from any DMA-capable peripheral, it initiates a DMA request signal to the processor asking its permission to use its bus, once a permission from the processor was granted, the DMA controller reads and writes one or more memory bytes, driving

the address, data, and control signals as if it were itself the processor. When the transfer is complete, the DMA controller deasserts the DMA request signal and the processor resumes its control of the bus.

L1 Instruction and Data Memory:

L1 instruction Memory and L1 data memory are both part of the internal memory and are accessed at full processor speed. The instruction memory consist of up to 80K bytes SRAM, of which 16K bytes can be configured as a four-way set-associative cache. The data memory consist of up to two banks of up to 32K bytes each. Each memory bank is configurable, offering both Cache and SRAM functionality.

All of the peripherals, except for general-purpose I/O, Real-Time Clock, and Timers, are supported by the DMA. In addition, the processor core can operate in full speed even when all peripherals are used due to the multi on-chip busses.

Chapter 5

Implementation

5.1 G.722 Codec Simulation

G.722 codec is first implemented on the VisualDSP++ Simulator. By using VisualDSP++ Simulator, the implemented codec works with files, which means that the data is read from an input file, processed by the simulator and sent back to an output file. Working with files is an easy and accurate way to verify that codec works properly before it is implemented in real-time. The test files used for the verification of codec are downloaded from ITU web page. Appendix C shows the names of the files and where they are used in the test configuration.

The downloaded input test files should be converted into raw data to satisfy the input requirements of G.722 test program.

5.1.1 Test configuration

Two configurations (figure 5.1 and figure 5.2) are appropriate for use with the test files. In both configurations the QMFs are by-passed and the test sequences are applied directly to the ADPCM encoders or decoders. Reset signal, rS, is extracted from the input test sequences and is used to initialize state variables of both encoder and decoder.

Configuration 1

In configuration 1, the input sequence signal X# is fed to function INFA which extract reset signal and input signals to lower (xL) and higher (xH) sub-band ADPCM encoder. The encoder input signals, xL and xH, are directly fed to the respective lower and higher sub-band ADPCM encoders, by-passing the QMF. The encoder output signals, IL and IH, are than fed into INFB which create an output test sequence by combining lower and higher sub-band ADPCM encoder output signals and the reset signal.

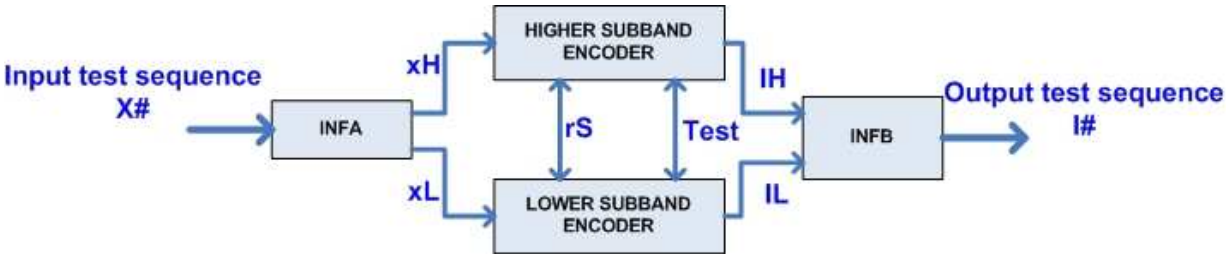

Figure 5.1: Test Configuration 1. Tests the encoder only.

Configuration 2

In Configuration 2 the input test sequence I# is fed to function INFC which extract reset signal and input signals to lower (ILr) and higher (IH) sub-band ADPCM decoder. The test signals, ILR and IH, and the MODE signal are than fed to both higher and lower sub-band of the decoder. The corresponding decoder output signals, RL and RH are than used in function INFD which create output test sequence by combining lower or higher sub-band ADPCM decoder output signal and the reset signal.

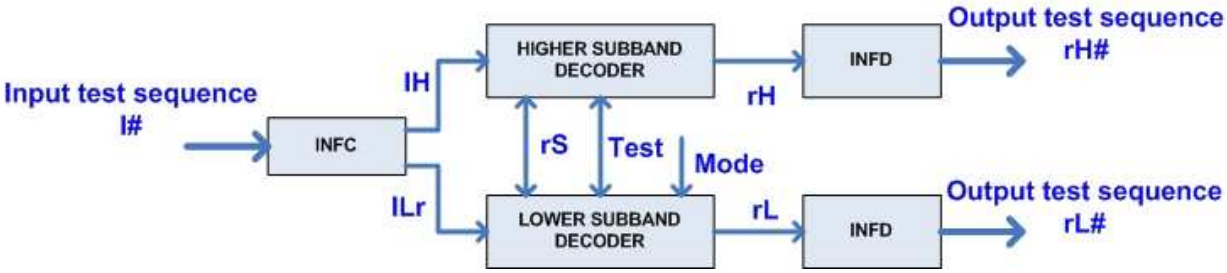

Figure 5.2: Test Configuration 2. Tests the decoder only.

5.1.2 ITU Test File Conversion

All test files supplied by ITU are written in ASCII with 64 characters per line of data followed by a two characters of checksum. The first two lines give us information about the file and at the end of each file a line of comment closes the file. In order to use these test files, they first need to be converted so that every 4 characters, which make one sample, are separated from each other by space in order to be read correctly by the codec test program. Comments and checksum in the original test file will be ignored. Each line in the converted file contains 16 sample of 16-bit each. The program used to convert the test files is shown in Appendix A.1.

5.1.3 Test files input

Once all test files were converted they are applied to the test program (Appendix A.2). The program accepts an input file and output three files. The first file is the output of the encoder as described in INFB. The second and third files are the output of the decoder rl and rh as described in INFD.

All input files provided by ITU were tested and the corresponding output matched the ones from ITU.

5.1.4 Real Signal Input

The codec is also tested using real speech and music signals extracted from WAV files using MATLAB program. The program stores the raw data from the WAV file in another file as raw data (Appendix A.3). The file is than applied to the simulation program (Appendix A.4) with both QMF filters enabled. The output of the codec is stored in a file as raw data.

The output file which contains raw data is processed in MATLAB using a program that creates a WAV file from the raw data stored in the output file (Appendix A.5).

Subjective tests in which the input and output WAV files were acoustically compared showed good results.

5.2 Blackfin Real Time Implementation

The G.722 codec was implemented in real time on the EZ-KIT lite develpment kit from Analog Devices.

Key features of the EZ-KIT lite are:

- The BF-533 processor
- SDRAM and Flash Memory
- AD1836 (D/A, A/D) operating at either 96 or 48 kHz
- LEDs, Push buttons and DIP Switches
- Background Telemetry Channel
- Evaluation suite of VisualDSP++

The real time implementation will be explained in this chapter. Fig. 5.3 illustrates the real time implementation of G.722 Codec on the EZ-KIT lite.
In this chapter several names and functions are used that are unique to the BF-533 processor. For a definitions of these see [5].

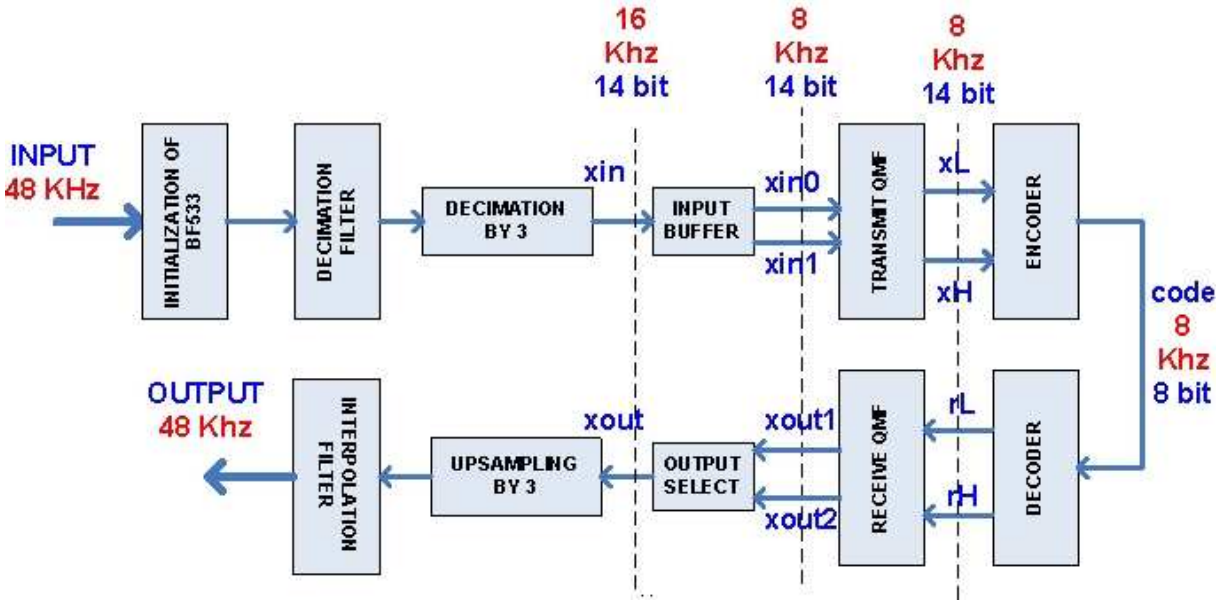

Figure 5.3: Block Diagram of the Blackfin Real Time Implementation

5.2.1 Initialization for Audio Configuration

In order to implement the G.722 codec on the BF533 our program calls a few initialization routines to set up the EZ-KIT Lite in a configuration that is needed for audio signal processing according to figure. 5.4

EBIU Initialization

The EBIU (External Bus Interface Unit) provides interfaces to external memories. EBIU_AMBCTL0 and EBIU_AMBCTL1 are registers used to control the memory timing for memory banks, i.e. Banks 1 and 0 (EBIU_AMBCTL0) and Banks 2 and 3 (EBIU_AMBCTL1). Correct configuration of these registers together with EBIU_AMGCTL enables all four memory banks access to Flash A.

Flash Initialization

The AD1836 reset is controlled via the flash A port A pin 0 and is there for need to be set as an output by the direction register.

A/D1836 Codec Initialization

We start by setting the data out register bit 0 in the flash memory to '0' (Port A.0) to reset the codec. and then set it back to '1' to enable it again. Once the codec is enabled we wait in a loop for the codec to recover from reset. The Serial Port Interface (SPI) is set to Direct Memory Access (DMA) transmit mode, 16 bit single data transfer and as a master. DMA5 is than mapped to SPI and is filled with all the data needed to initialize the codec. Once the data is organized in the DMA, we enable both the DMA and the SPI. At this point the data is transmitted from the

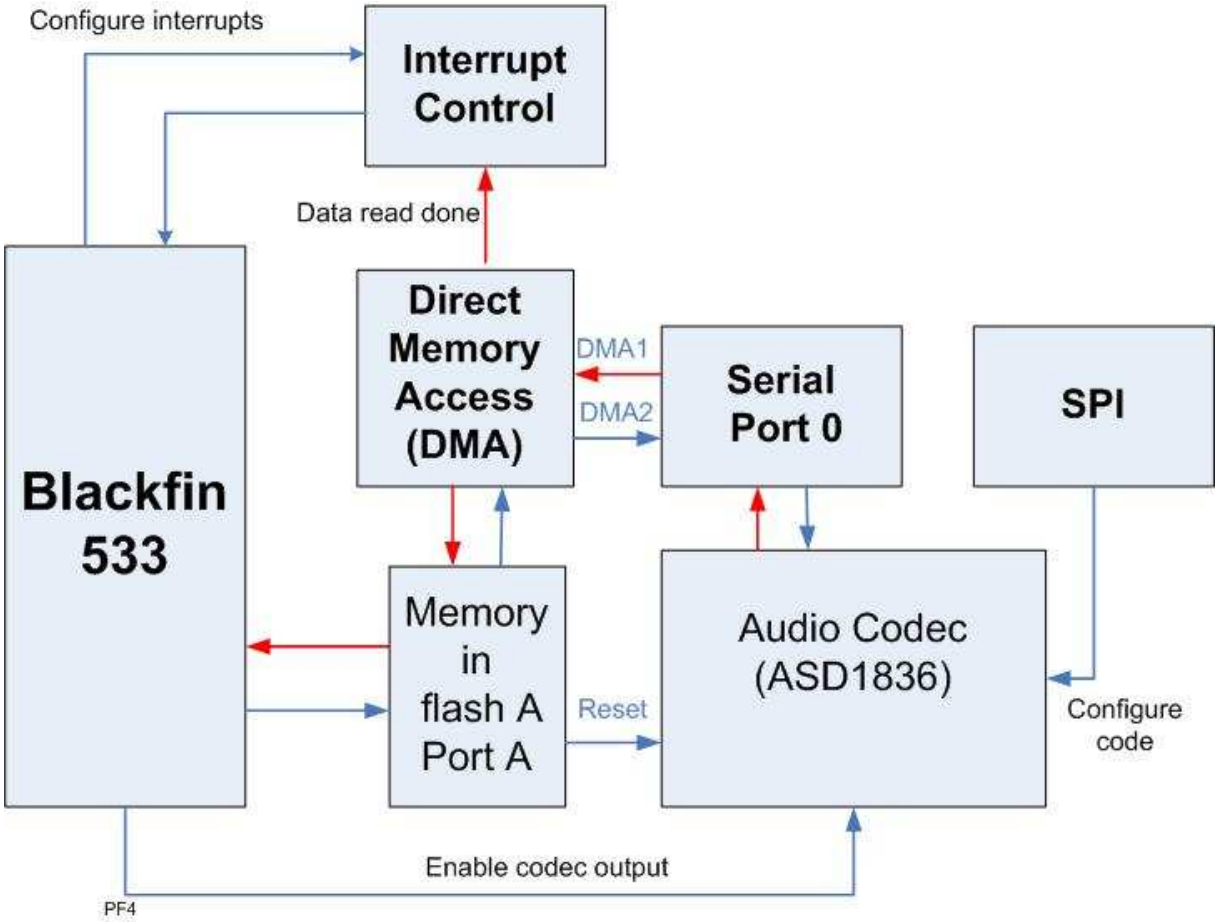

Figure 5.4: Setup of the Blackfin and the Audio Codec on the EZ-KIT Lite

DMA to the AD1836 while we wait in a loop. Once the data has been transferred the SPI is disabled in the SPI_CTL register.

Sport0 Initialization

Serial Port 0 (Sport0) has two registers that needs to be initialized for receive(SPORT0_RCR 1 and 2) and two registers for transmit(SPORT0_TCR 1 and 2). The registers are configured to external clock(IRCLK=ITCLK='0'), external frame sync(RFSR=TFSR='1'), MSB first(RLSBIT=TLSBIT='0'), drive data with falling edge of TSCLK and RSCLK(TCKFE=RCKFE='1').

DMA Initialization

The DMA is a device for transferring data to or from other memory locations or peripherals without the attention of the CPU and is used in our configuration to buffer the data of the four audio channels from Sport0 and from the Blackfin. DMA1 and DMA2 are mapped to SPORT0_RX and SPORT0_TX respectively and are

configured using DMA_CONFIG register to 16-bit transfers, interrupt on completion of receiving or transmitting data, autobuffer mode which means that the program is interrupted at the conclusion of each buffer and in that way we know that data is ready to be fetched. DMAx_START_ADDR is set to the address of the transmit and receive array, DMAx_X_COUNT is set to the number of the elements in the array which correspond to our four input channels and DMAx_X_MODIFY is set to 2 which correspond to 16-bit word (2 bytes from the memory).

Enable_DMA_Sport0 Initialization

Once Sport0 and the DMA are configured they need to be enabled. This is done by setting DMA_EN bit to '1' in the DMAx_CONFIG registers and by setting TSPEN and RSPEN to '1' in SPORT0_TCR1 and SPORT0_RCR1 respectively.

Interrupt Initialization

Sport0, which is serial port 0, is used to transfer the data to and from the codec to the system and need to be initialized. Sport0 has two control registers for transmit and two control registers for receive that are set with the following values for correct operation: External CLK, external frame sync, MSB first, active low, 16-bit data and stereo frame sync enable for both transmit and receive. Sets Sport0 RX (DMA1) interrupt priority to 2, i.e IVG9=2, in the SIC_IAR1 register to allow an interrupt each time the buffer is full. The setting of the rest of the interrupts is irrelevant as they are masked in the SIC_IMASK register, which is a System Interrupt Mask Register, and which is set to only enable DMA1 Interrupt (SPORT0 RX). The Register_handler is used to assign interrupt handlers to different interrupt requests. The first argument is the name of the interrupt (IVG9) and the second argument is the desired name of the interrupt handler(Sport0_RX_ISR).

5.2.2 Audio Program

After completing all initialization routines, the next step is to run an Audio program which basically gets one audio input sample from the input buffer and send this input sample, without any processing, to the output buffer. The Audio program is executed to verify that input-output chain works accurately.

5.2.3 Changing Bit Rate

As it is explained in Chapter 3.2, transmit audio part of G.722 should be composed of an Analog to Digital (A/D) converter which outputs 14 bit data to satisfy the input specifications. However, A/D 1836 of BF533 EZ-KIT Lite can output data with three different bit rates such as 16, 20 and 24. Hence, DMA and A/D 1836 Codec registers are modified to output 16 bit data as explained in 5.2.1. Then, 16 bit output data is right-shifted by two bits to have 14 bit data as an input to G.722 encoder.

5.2.4 48 kHz to 16 kHz Decimation

As explained in Chapter 4, the sampling frequency of Analog to Digital (A/D) Converter on BF533 EZ-KIT Lite can be set to 48 kHz or 96 kHz. However, the sampling frequency of the input signal specified for the G.722 codec is 16KHz. Thus, the sampling frequency of the A/D converter is decimated to 16 kHz to satisfy the input requirements of G.722 codec. The code which performs decimation can be seen at Appendix B.5, line 28.

5.2.5 Decimation Filter Implementation

Decimation can be done without alliasing if we reduce the bandwidth of the signal before downsampling, [2] . Thus, input signal is filtered by an lowpass filter with a cutoff frequency, 16 kHz, which is equal to the sampling frequency divided by decimation factor. Decimation filter used in the implementation is a built-in filter written in assembly. Usage of Decimation filter can be seen at at Appendix B.6, lines 23-27.

5.2.6 Input Buffer

The input to the input buffer, xin ,is the output from the decimation filter and sampled by 16 kHz as shown in figure. 5.3. The G.722 codec which runs at 8Khz requires two samples as an input: xin0, xin1. xin1 is the previous sample and xin0 is the current sample of the 16Khz input. These two samples are stored in the input buffer and are used each time the codec is executed. The code which performs input buffer can be seen at Appendix B.6, lines 32-43.

5.2.7 Output Select

The inputs to the output select function, xout1 and xout2, are the outputs from the decoder and sampled by 8 kHz as shown in figure. 5.3. This function selects one of the 8 kHz input signals alternately to produce 16 kHz sampled output signal, xout.

The code which performs output select can be seen at Appendix B.6, line 35 and 41.

5.2.8 16KHz to 48Khz Upsampling

After processing the input signal with Blackfin DSP, the processed signal is interpolated before the signal is sent to the Digital to Analog (D/A) converter . As it is explained in Chapter 4, D/A converter works with the signals sampled by 48 kHz. Hence, in our case, the sampling frequency of the decimated signal is 16 kHz. It should be upsampled to 48K for being able to processed by D/A converter properly. The code for upsampling is given at Appendix B.6, line 44 and 48.

5.2.9 Interpolation Filter Implementation

After increasing the sampling rate to 48 kHz, an interpolation filter is needed to remove all frequency-shifted images of the signal, [2]. The interpolation filter has also an amplitude gain, which is equal to the interpolation factor, compensating for the amplitude loss due to upsampling. Interpolation filter used in the implementation is a built-in filter written in assembly. Usage of Interpolation filter can be seen at at Appendix B.6, line 50-52.

5.3 Optimization

Code optimization is done in two steps. The first step is done by using VisualDSP++ tool as described in section 5.3.2. The second step is Assembly Optimization as described in section 5.3.3. In order to determine what functions need to be optimized we use the Statistical Profiling to determine the most frequently executed functions as described in section 5.3.1

5.3.1 Statistical Profiling of functions

VisualDSP++ Statistical Profiling measures the performance of the DSP's program by sampling the Program Counter (PC) register and the memory accesses at random intervals while the program is running. The output of is a graphical display that shows where the processor spends its time.

5.3.2 VisualDSP++ Optimization

The VisualDSP++ Optimization tool is the first step used to optimize our program. The tool provides three ways to optimize our code:

- Automatic Inlining, in which the compiler automatically inlines all the functions which are not necessarily declared as inline in the our code. Automatic Inlining is done by the compiler when it has determined that doing so will reduce execution time.

- Procedural optimizations, in which the compiler performs advanced, aggressive optimization on each procedure in the file being compiled. The optimizations can be directed to favor optimizations for speed or size. In our case speed was chosen.

- Interprocedural optimizations (IPA), in which compiler performs advanced, aggressive optimization over the whole code in addition to the per-file optimizations in procedural optimization.

  Table. 5.1 shows the functions in which most of the time is spent during execution before the optimization.

Function	Execution unit in %
main	53.11
add	9.05
mult	7.14
shr	6.70
Sport0_RX_ISR	3.99
_firfr_16	3.61
lsbcod	3.33
hsbcod	2.79
lsbdec	2.66
hsbdec	2.65
sub	1.39
L_add	1.12
qmf_rx	0.54
qmf_tx	0.51
shl	0.43
G722	0.42
invqbl	0.25
L_shr	0.23
negate	0.08

Table 5.1: Execution unit of each function After VisualDSP++ optimization. Add, mult and shr are the most frequently used functions.

5.3.3 Assembler Optimization

Once maximum VisualDSP++ optimization was achieved, it is needed to use assembly language in order to further optimize the most frequently used functions. Programming in assembly gives us access to the Blackfin hardware: registers, I/O memory etc, and in that way we can produce a more efficient code with lower number of cycle consumption. The most frequent functions as seen in table 5.1 are: add, mult and shr. These functions were implemented in assembly (Appendix B.12). This resulted in significant reduction of execution time of these functions as seen in table 5.2.

Function	Execution unit in %
main	72.34
Sport0_RX_ISR	3.99
firfr_16	3.61
lsbcod	3.29
_add	2.88 (9.05)
hsbcod	2.77
lsbdec	2.65
hsbdec	2.62
_mult	1.72 (7.14)
_shr	1.40 (6.70)
qmf_rx	0.53
qmf_tx	0.49
_sub	0.44 (1.39)
G722	0.42
invqbl	0.25
L_shr	0.23
_L_add	0.21
_shl	0.12 (0.43)
negate	0.03

Table 5.2: Execution unit of each function after assembly optimization. Execution units of add, mult and shr functions decreased drastically.

5.3.4 Cycle Count Comparison

After the optimization of the program, cycle count of whole program is compared with the cycle count of whole non-optimized program. Table. 5.3 shows the cycle count of optimized and non-optimized programs. Optimization with only assembly functions decreases cycle count by 47.72 percent and optimization with only VisualDSP++ tool decreases the cycle count by 69.5 percent. When both optimization methods are used, the cycle count is decreased by 83.82 percent.

Type of optimization	Number of Cycles
Non optimized program	42758
Optimized with only assembly functions	18508
only VisualDSP++ Optimization	13486
Both assembly and VisualDSP++ Optimization	6922

Table 5.3: Comparison of cycle counts between optimized and non-optimized programs

5.4 Test Results

5.4.1 Subjective Methods

The output of the codec is acoustically compared with the original signal. We use DIP switch 9 on the EZ-KIT Lite to bypass the codec in the program. Thus, the audience can easily compare the output of the codec with the original speech signal. Table. 5.4 shows the performance grade of each person for the real time implementation of codec on BF533.

Name	**Performance Grade over 5**
Jenny Hogberg	5
Ferhat Erdogan	4
Mahaboob ALI Basha	5
Ali Asad	5
Abdul Haseeb	5
Johan Larsson	5
Ram Chandra	5

Table 5.4: Subjective test results of G.722 codec performance

This performance test shows that the quality of coded speech signal is excellent.

Chapter 6

Conclusions

Today, G.722 Wideband Speech Codecs are widely used in some areas such as Audio and video conferencing, Digital AM radio broadcasting, dual language programming in audio/video broadcasts of news, TV programs. A real time implementation of G.722 WideBand Speech Codec was successfully developed on Blackfin 533 DSP in this thesis. The code was developed in C and functions which are used frequently were optimized by assembler. Significant reduction of execution time was achieved

The G.722 codec was verified by using different types of test files from the ITU organization. On VisualDSP++ Blackfin Simulator, the same outputs as ITU were obtained for test files.

G.722 Speech codec requires that the input signal is sampled at 16KHz and quantized with 14 bits. It can be seen that very good sound quality results were obtained in real time when the input requirements were satisfied. Optimization results were also shown in this thesis. VisualDSP++ optimization tool and assembly programming decreased the number of cycles more than 80 percent.

There are many possibilities for further development in this discipline, some of these are as follows.

- The G.722 Codec system was developed on the EZ-KIT Lite board of BF533 DSP, this has certain parameters such as sampling rate and quantization rate that are unalterable by the developer. Another possible way to increase the performance of the G.722 Codec is to use the BF533 DSP in a custom made circuit which could properly utilize its full potential.
- The codec system can be optimized in terms of cycle counts by modification of software code of G.722 QMF filters, Adaptive Quantizers, Inverse Adaptive Quantizers or Adaptive predictors.

The goals of this thesis project have been successfully accomplished.

Sincerest thanks go to our thesis supervisors, Benny Sallberg and Software Development Manager Fredric Lindstrom, whose assistances were invaluable throughout this project.

Bibliography

[1] Alan V. Oppenheim and Ronald W. Schafer. Discrete Time Signal Processing . *Prentice Hall Signal Processing Series*, ISBN: 0-13-754920-2, pages:724-725,1998.

[2] Alan V. Oppenheim and Ronald W. Schafer. Discrete Time Signal Processing . *Prentice Hall Signal Processing Series*, ISBN: 0-13-754920-2, pages:167-176,1998.

[3] 7 kHz Audio-Coding Within 64 kbit/s. *ITU-T Recommondation G.722*, 1988

[4] Snow. Lam, James. Kagie, Anad. Xavier, Warren. Machen, Mahbubur. Khan, Mahmudur. Rahman.Final Presentation of SBC/ADPCM Project, 2004, 01 Nov. `http://www.eas.asu.edu/~speech/research/sbc/`

[5] ADSP-BF533 Blackfin Processor Hardware Reference. *Analog Devices.*

Appendix A

SIMULATION AND TEST PROGRAMS

To run the simulation program on VisualDSP++ Simulator Session, the files in Appendix B.1, B.2, B.4, B.8, B.9, B.10, B.11 should be included with the main file in Appendix A.4.

A.1 main_conversion.c

A.2 main_test.c

A.3 wav2array.m

```
y = wavread('Input_wav_file');% raw data into variable y
fid = fopen('input_file_name','w');
[e,d]=size(y);
for i=0:e-1
    fprintf(fid,'%d \n',round(y(i+1)*32000));% saves samples in a
       file
end
fclose(fid)
end
```

A.4 main_simulation.c

A.5 array2wav.m

```
load output_file_name_with_extension;% load the raw data to variable
                                     %
                                        output_file_name_without_extension

% Create a WAV file
wavwrite(output_file_name_without_extension/16000,16000,'
    output_wav_file');
```

Appendix B

REAL TIME IMPLEMENTATION

B.1 funcG722.h

B.2 G722.h

B.3 AudioConf.h

B.4 operg722.h

B.5 main.c

B.6 ISR.c

B.7 Initialize.c

B.8 g722reset.c

B.9 g722.c

B.10 operg722.c

B.11 funcG722.c

B.12 operg.asm

Appendix C

Test files

File name	Input/Output
T1C1.XMT	Encoder Input
T1C2.XMT	Encoder Input
T1D3.COD	Decoder Input
T2R1.COD	Encoder Output Decoder Input
T2R2.COD	Encoder Output Decoder Input
T3L1.RC1	Decoder (Mode 1) Output (rl)
T3L1.RC2	Decoder (Mode 2) Output (rl)
T3L1.RC3	Decoder (Mode 3) Output (rl)
T3H1.RC0	Decoder Output (rh)
T3L2.RC1	Decoder (Mode 1) Output (rl)
T3L2.RC2	Decoder (Mode 2) Output (rl)
T3L2.RC3	Decoder (Mode 3) Output (rl)
T3H2.RC0	Decoder Output (rh)
T3L3.RC1	Decoder (Mode 1) Output (rl)
T3L3.RC2	Decoder (Mode 2) Output (rl)
T3L3.RC3	Decoder (Mode 3) Output (rl)
T3H3.RC0	Decoder Output (rh)

The two Encoder input files can be, after conversion (Appendix A.1), applied to the test program (Appendix A.2). The program accepts two arguments: the first argument is the input file name and the second argument is the mode.