Implementation and testing of an improved echo canceller and an ADPCM speech codec

Besselink, R.J.C.

Award date:
1998

Eindhoven University of Technology

Faculty of Electrical Engineering
Telecommunication Technology and Electromagnetics (TTE)
Radiocommunications Group (ECR)

Implementation and testing of an improved Echo Canceller and an ADPCM Speech codec

Master's Thesis of R.J.C. Besselink

Supervisor	Prof. dr. ir. G. Brussaard
Coaches:	Ing. A. Tognoni
	K. Elmalki, PhD
Period:	May 1997 until July 1998
Place:	Ericsson Telecommunicazioni S.p.A.
	Rome, Italy
	Systems department

ABSTRACT

In the field of telecommunications, network operators attempt to increase revenue by augmenting the speech quality of the network and reducing costs. The use of echo cancellers for long-distance telephoning yields a drastic increase in speech quality of the network. Whereas the ADPCM speech codec, that is standardised by the ITU, provides a similar performance when compared to log-PCM, but operates at a bit rate that is only half of log-PCM. Thus, with ADPCM the capacity of existing links can be doubled.

The reasons mentioned before clarify why a company like Ericsson is very interested in these types of algorithms, especially because they are both used in DECT handsets. Ericsson in Rome, where this Master's Thesis was carried out, wanted to perform practical tests with these algorithms in one of their laboratories. For this, a test environment had to be set up that resembles the situation found in a modern PSTN. The first part of this Thesis describes this test environment as well as the test requirements.

The second part of the activities involved the testing of an improved time-domain echo canceller that was implemented by Ericsson. Several novelties were used to improve the performance and reduce the computational load of a DSP, like Voice Activity Detection (VAD) and 'echo path optimisation'. The low-level code had to be downloaded on a DSP platform, debugged and tested. A simulated delay had to be implemented within pre-written Firmware to account for the high transmission times with long-distance telephoning. Several LMS-based algorithms were evaluated like: NLMS, DLMS and FNLMS.

After many hardware and software problems were solved in the lab, tests were conducted revealing that an excellent quality was achievable with the echo canceller. Both novel features proved working, after minor modifications were made. With speech the algorithm converged perfectly. On the other hand, with tones as input signal, the algorithm often caused the DSP to block completely. The problem was left unsolved. The Thesis reveals how this error may be solved, and additional recommendations were written down to further improve the implemented algorithm. Up to mean one-way transmission times of about 80 ms no NLP is required. Of the various LMS-based algorithms the NLMS adaptive filter worked best.

One of the speech compression algorithms Ericsson was eager to test is ADPCM. To be able to test ADPCM, an efficient low-level had to be developed for the TMS320C542 DSP, that uses 16-bit fixed-point arithmetic. Therefore, a floating-point and a fixed-point program in C language was developed from which an efficient program in assembler could be derived. This microcode was tested using the 'C54x Device Simulator from Texas Instruments, before the code was downloaded on the real DSP. After this proved working some minor tests were carried out in the laboratory with tones and speech as input signal. No problems were encountered. From this point, Ericsson will take care of the desired tests.

Additionally, simulations were done using the Encoder and Decoder programs written in C language. A number of five different tests were carried out with fixed-point and floating-point variables, a comparison between the ITU Recommendations G.721 and G.726 of ADPCM, and allowing +0 to be transmitted by the Encoder. From these simulations, using 6 different speech files and up to 10 synchronous tandem codings, it was found that the codec using floating-point variables yields an increased *subjective* speech quality over a codec using fixed-point variables, though not spectacularly. The older ITU standard G.721 provided for a higher speech quality than the newer G.726 standard. The largest effect was obtained by allowing the +0 to be transmitted by the Encoder. The speech material still was highly intelligible after even 10 synchronous tandem codings.

Benchmark values were obtained from the microcode showing that about 8 channels could be either encoded or decoded simultaneously. The ADPCM speech codec will consume 1216 words of program memory. For each channel 48 words of data memory should be reserved.

CONTENTS

CHAPTER 1

INTRODUCTION

What started as an intended substitution of the numerous incompatible systems for cordless telephones evolved in something much bigger. Digital European Cordless Telecommunications (DECT) became an international standard that is estimated to reach 30 milion units to be sold in the year 2000. Like GSM, also DECT is a European initiative that became a global standard. For this reason the 'E' in the abbreveation for DECT now stands for 'Enhanced'.

The residential use for DECT is the most successful part of all possible application. Other applications where DECT is implemented are:

1. Radio in the Local Loop (RLL)
2. Business cordless
3. Public Cordless Terminal Mobility (CTM)

where,

1. The first substitutes the twisted-pair local loop that exists today to connect the Public network to the user premises. In case a fixed network does not exist, like in many third-world countries, it may be cheaper to implement RLL.
2. Due to a high flexibility, small businesses are able to implement DECT for their Wireless LAN and/or digital cordless in-door communication at low cost. The flexibility enables the DECT system to grow with the size of the company.
3. Also, in the coming years CTM will be introduced next to the GSM network. The idea is to provide for greater competition with the existing GSM network. In a sense it can be seen as a fixed network and therefore be operated by fixed network operators. The service will be offered at a reduced price, the same as for fixed telephony, at the cost of reduced mobility (inter-city, or highly populated zones), compared to GSM. The Ericsson site in Rome, where this master's thesis was carried out, is involved with the implementation in Italy.

The base station of DECT is able to handle 120 users simultaneously. For this, it uses 10 different frequencies and 24 different time slots (duplex), which are assigned in a dynamic way. For example, the handset determines the frequency and the timeslot that has the smallest disturbance. The cells are small, which is the main key to high density. Since the distance to the mobile units has a maximum of a few hundred meters, both antennas and mobile unit consume little energy. This is a great advantage when considering the price and weight of batteries.

To more efficiently use the bandwidth, 32 kbit/s ADPCM coding is applied for speech instead of 64 kbit/s log-PCM. Both in the handset and the base station the ADPCM codec can be found. For this reduced bit rate, speech will only be subject to a minor degradation of quality. This performance is significantly better than for GSM, since the latter operates at a much lower bit rate (13 kbit/s).

Speech transmitted over DECT introduces a round-trip delay of approximately 20ms. This will not affect the speech quality for the far-end user, since DECT is situated outside of the PSTN. However the DECT user will, under certain circumstances, experience degradation of quality due to a delayed echo. For this reason, an Echo Canceller is required within the Fixed Part (FP) of DECT.

More detailed information about DECT can be found in [ETSI_1] and in [ETSI_2].

1.1 ACTIVITIES

The introduction of DECT shows a new application for both echo cancellation algorithms and ADPCM transcoders. Regarding echo cancellation algorithms, an Ericsson employee was involved in the development and implementation of an Echo Canceller with improvements compared to existing algorithms. The prototype had to be tested in a practical environment to verify the simulated performance. This work is important since simulated quality of any product is often quite different from reality. False, considerations, or badly estimated parameters may well lead to poor behaviour.

The company site where this work was carried out has very little knowledge on the working and behaviour of speech codecs. In order to gain more experience in this field, they were anxious to practically test an important transcoding algorithm. Therefore, a low-level code had to be developed that could operate on a specific DSP. With this code they could carry out required tests, use it for demonstrations and retrieve benchmarks for the particular DSP. Benchmarks are important for calculations of computational requirements in future systems that will use this transcoder. A possible test could be the interoperability with Echo Cancellers. It is widely known that non-linearities caused by e.g. transcoders may lead to a troublesome performance of echo cancellers.

In summary, the activities that were carried out for the project consisted of three major parts:

- setting up a test environment that was able to create a real-world situation in which both speech codecs and the echo canceller algorithm could be tested;
- testing of the improved Echo Canceller algorithm in the laboratory. This work includes the implementation of a simulated long telephone line (delay), since echo is mainly observed in long-distance calls;
- developing the low-level code of an ADPCM transcoder that operates at a bit rate of 32 kbit/s. And, validate the proper working in the laboratory.

1.2 LAYOUT OF THIS THESIS

Since the tasks for this thesis are so distinct I separated them in three different parts. Chapter 2 explains the test-bed and everything it involves with. This also means that everything that was required to be able to test the Echo Canceller, as well as the ADPCM transcoder is depicted here. It also clarifies why certain tests were not performed due to the limited equipment of the laboratory.

The second part, Chapter 3, handles the echo canceller. It starts with a fairly detailed description of the aspects of echo cancellation, including the theoretical aspects. This initial part is followed by a description of the improvements that were made by Ericsson, together with the simulation and laboratory test results. Conclusions and recommendations can be found in Chapter 4.

The final part, being Chapter 5, gives an introduction to speech compression, with the emphasis on ADPCM. A general description of the algorithm is provided, as well as a more in-depth discussion of the implementation. With the knowledge of the transcoder in mind, the programming approach towards assembly code is described. This is followed by simulation and laboratory results. Chapter 6 summarises the conclusions and recommendations.

In order to facilitate easy referencing to Formulas, Tables and Figure, these are NOT numbered in consecutive numbers but contain the *pagenumber*. If more than one of the same type is found on the same page an extra alphabetic index is added. E.g., Formula 24b points to the second formula on page 24.

1.3 GLOSSARY

ADPCM	Adaptive Differential Pulse Code Modulation
BSP	Buffered Serial Port
CSS	Composite Source Signal
CTM	Cordless Terminal Mobility
DAT	Digital Audio Tape
DCME	Digital Circuit Multiplication Equipment
DCT	Discrete Cosine Transform
DECT	Digital Enhanced/European Cordless Telecommunications
DFT	Discrete Fourier Transform
DHT	Discrete Hadamard Transform
DL2	Digital Link 2
DP	Device Processor
DSI	Digital Speech Interpolation
DSP	Digital Signal Processor (or Processing)
DTMF	Dual Tone Multi-Frequency
DWHT	Discrete Walsh-Hadamard Transform
CODEC	Coder Decoder
ERLE	Echo Return Loss Enhancement
ETSI	European Telecommunications Standards Institute
FL	Fixed Loss
FP	Fixed Part (within DECT systems)
HPI	Host Port Interface
ITU(-T)	International Telecommunication Union (Telecommunication Standardisation Sector)
KLT	Karhunen-Lòeve Transform
LAN	Local Area Network
LE	Local Exchange (within PSTN)
LUT	Look-Up Table
MIPS	Million Instructions Per Second
LMS	Least Mean Square
MOS	Mean Opinion Score
MUSSE	Multi-Usage 2Mbit Switching and Signal processing Equipment
N-ISDN	Narrow-band Integrated Services Digital Network
NLMS	Normalised Least Mean Square
NLP	Non-Linear Processor
PCM	Pulse Code Modulation
PCME	Packet Circuit Multiplication Equipment
PSTN	Public Switched Telephone Network
QDU	Quantisation Distortion Unit
RLL	Radio in the Local Loop
SBC	Sub-Band Coding
SIFT	Simple Inverse Filtering Tracking
SNR	Signal-to-Noise Ratio
STM	Synchronous Transfer Mode
TELR	Talker Echo Loudness Rating
TASI	Time Assignment Speech Interpolation
TI	Texas Instruments
TS	TimeSlot
VAD	Voice Activity Detection
VNL	Via Net Loss
VOCODER	Voice Coder

CHAPTER 2

THE TEST ENVIRONMENT

The first chapter discussed the true demands for the test environment. The first is that a "real-world" situation has to be created. The second is that speech codecs and echo canceller algorithms could be tested on it. The questions that arise from this are:

1. What is a real-world situation?
2. What are the demands for the testing of such algorithms?

A real-world situation is created if comparable conditions are found in modern telecommunication networks. This is because the mentioned algorithms could or will be used within such networks. Modern Public Switched Telecommunication Networks (PSTNs) are all digital. Switches, the links between these switches, as well as processing hardware and software are digital. This means that the analogue signal that is about to enter such networks will be filtered to avoid aliasing, sampled at a fixed sampling rate, and quantised at a fixed number of bits. Networks generally use a bandwidth between 200 and 3400 Hz, a sampling frequency of 8000 Hz, and quantise the sampled data to 8 bits using companding techniques like A-law and μ-law. The inverse applies for the data that leave the network and are sent towards the user's premises.

Either speech, voice-band data (fax/modem), or tones (DTMF) excite the algorithms. People still use speech as the main important means of transporting information. This speech is picked up by the telephone in the user's premises and transmitted over a twisted pair, generally referred to as local loop, towards the Local Exchange (LE) of the network. Modern networks merely use 4 wires, half for the reception of the data and the other half for the transmission of the data. To provide interfacing between the 4-wire network and the 2-wire twisted pair between the network and the user a hybrid is used. The hybrid is especially needed to test the echo cancellation algorithms. Hybrids actually cause the echo in telecommunication networks.

Concerning the physical requirements of the system we are about ready. What was left unmentioned was the type of device that performs the calculations of the algorithm. Ericsson uses the TMS320C542 DSP from Texas Instruments for all the application software within their switches (AXE 10). This type of processor is also used for this project. From this point only the requirements regarding the tests are missing. These can be subdivided into tests needed for the improved echo cancellation algorithm and the speech compression algorithms, with the emphasis on the implemented ADPCM speech codec. Until now nothing is known about future test requirements for speech compression algorithms. Therefore, it seems best to focus on the requirements that are needed for the mentioned algorithms.

The echo cancellation algorithm that was developed by Ericsson was simulated on a Device Simulator from Texas Instruments. A synthesised hybrid response was used to circumvent using real hybrids. An estimate was made about the noise in the network, for instance. And, as input signal merely a single speech file was used, generated with the Audiotool of the Sparcstation. From the previous it becomes clear that the major test objective is the verification of the performance under real circumstances. To facilitate debugging it should be possible to view internal parameters to see what kind of fault occurred. The residual echo should be observable, either with a Signal Analyser (oscilloscope or spectrum analyser) or should be made audible on the speaker of a handset. A delay had to be implemented since echo is best observed with delays. More details about the exact tests can be found in Chapter 3.

Regarding the testing of speech compression algorithms not much can be said. Neither the exact algorithms nor the desired tests are yet specified. Ericsson still did not finalise this part. The ultimate test requirement will always be the quality of the transcoded signal. As input signal (like for the echo canceller) speech, voice-band data and tones may excite the algorithm. The amount of distortion of the speech, the bit error rate of the voice-band data and the amplitude distortion of the tones may be useful parameters to investigate. For demonstration purposes it may be valuable to send the speech signal to two telephones. On one telephone the original speech may be listened to, on the other the transcoded signal could be heard.

The contents of this chapter are shown next. All relevant matters regarding the test bed are described in section 2.1 of this chapter. Following, the second section starts with a brief introduction on digital signal processing by means of DSPs. Also, the TMS320C542 DSP that is used is further explained. Since the Echo Canceller and ADPCM codec are embedded in pre-written Firmware, this is briefly discussed in the third section. The last section of this chapter contains a description of how the correct functioning of the test bed was checked, and how it was set-up for the two algorithm types.

2.1 Description of the test bed
2.2 The Digital Signal Processor
2.3 The DSP Firmware
2.4 Set-up and testing of the test bed

2.1 DESCRIPTION OF THE TEST BED

The laboratory was not equipped with recording equipment for speech (e.g., DAT), nor with a modem or a fax. Thus, the signal generator had to be a common telephone. A signal generator was available that could generate waveforms like triangles, sinusoids and rectangles. For the visualisation of the residual echo or the transcoded speech, a Signal Analyser on a PC and oscilloscopes were at hand.

Figure 6 shows the test bed that was used for the various tests. All the components are described in the next pages. With this test environment the desired tests could be performed, what becomes clear later.

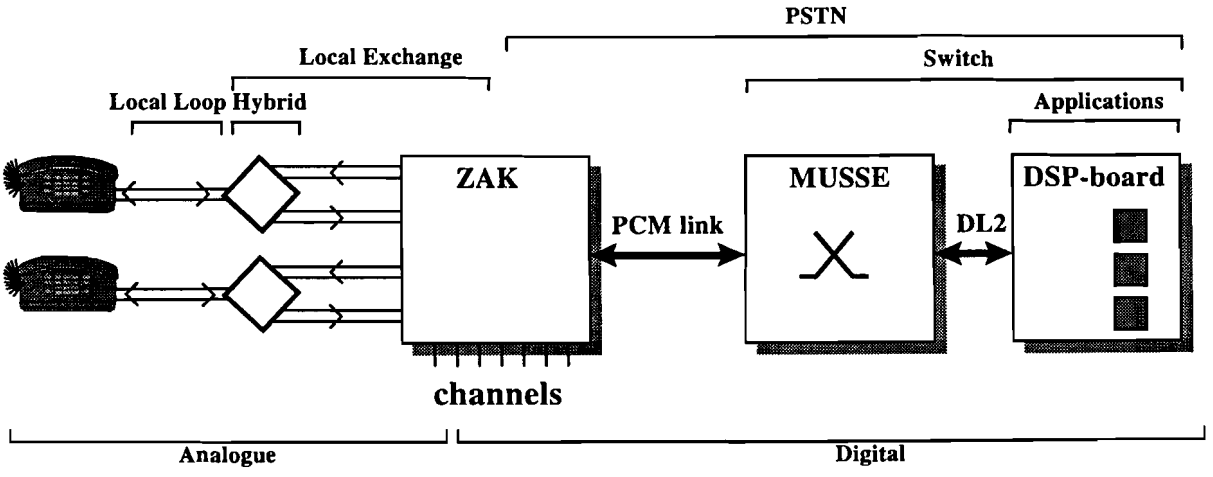


Figure 6 Block diagram of the test bed

With the **telephones** in Figure 6 the speech was generated to excite the algorithms and the result could be listened to on the speaker of the handset. The telephones are connected to the hybrid. This **hybrid** connects the 2-wire subscriber-line to the 4-wire network. The hybrid loss is about 20 dB, which means that the signal coming from the MUSSE are coupled back to the network but is 20 dB attenuated. Hybrids used in networks generally range between 6 and 11 dB. It was felt that this increased performance was not very relevant. The hybrids have in-built functions like on/off-hook detection and ringing. However, these are not used.

The 4-wire parts of the hybrids used are connected to their own 4-wire channel of the **ZAK,** which is an Ericsson code for this type of equipment. Figure 7 shows the block diagram. It may consist of several Channel Unit (Ch. U) and has a Control Unit (CU). The Control Unit controls the data on the internal bus and performs the switching between the Channel Units and the bi-directional PCM link. The board multiplexes 30 voice or 64 kbit/s data channels into a data flow of 2048 kbit/s, and vice versa. The PCM link has a STM 2 Mbit/s E1 frame structure and is in compliance with ITU-T recommendation [G. 704].

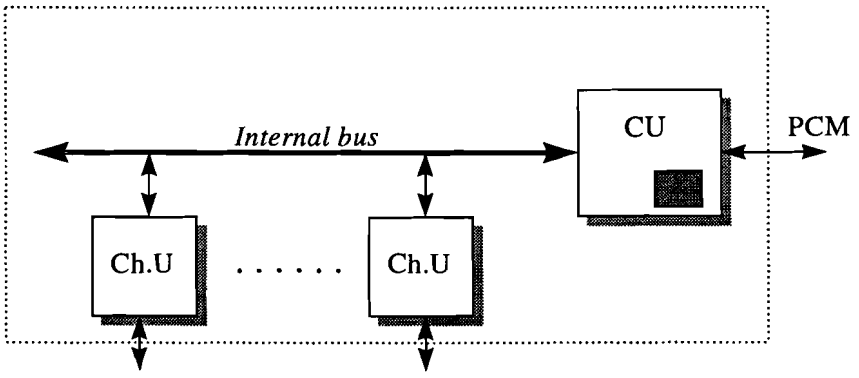


Figure 7 The ZAK subsystem

The CU-board has an on-board EEPROM that is programmed via the RS-232 serial port of a PC. It enables the user to set the right timeslot from the PCM link to the desired channel on the Channel Unit, and vice versa. Incoming timeslots can only be addressed to a single channel. The board takes care of frame alignment.

The Channel Units that can be connected to the internal bus of the ZAK may have analogue or digital inputs. The units that where installed are:

- 10 channel, 2/4 wire voice unit;
- 15 channel, 4-wire voice unit.

The first unit has the possibility to select the 2-wire or 4-wire configuration on a channel basis. The logic automatically places a balancing circuit between the receiving and transmitting part. This option seemed perfect, but the board did not provide loop-current for the telephones. Therefore the hybrids where needed. The 2-wire Channel Unit was used for connecting several channels to a PC. On this PC the required data acquisition hardware and the mentioned Signal Analyser were installed. Both units contain PCM codecs that filters, samples and compresses (A-law) the analogous input data, and performs the inverse operation for the digital data coming from the PCM link.

The full-duplex PCM link interconnects the ZAK and the MUSSE. The full name of the **MUSSE** is: "Multi Usage 2 Mbit Switching and Signal processing Equipment". In order to eliminate the need for the Ericsson AXE 10 Group Switch this board was developed. It constitutes interfacing between the PCM links and applications

offered by the Switch. It offers an easy-to-use interface handling for testing the algorithms. A block diagram of the MUSSE is shown in Figure 8.

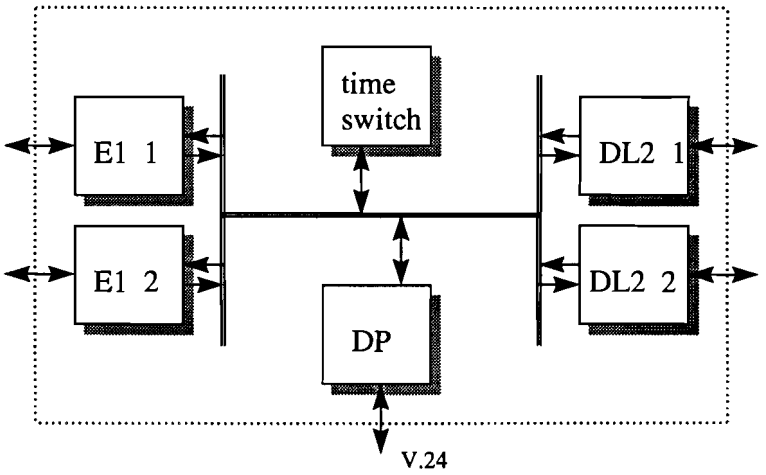


Figure 8 The MUSSE subsystem

The board has two full-duplex DL2 (Digital Link 2) transceivers, and two E1 PCM transceivers. The DL2 standard is an internal Ericsson standard, and these links are used to connect various parts within the Group Switch. It has a 32 channel synchronous frame structure with a bitrate of 4096 kbit/s. Half of this bandwidth is used for control and synchronisation. External clocks will be mastered by the MUSSE.

The time switch can connect whatever incoming timeslots to whatever outgoing timeslots of any link. Total links can be copied or specific timeslots can be selected. The V.24 interface can be connected with the RS-232 serial port of a PC or UNIX station. The V.24 is connected to the DP, which takes care of control of the platform as well as communication with the outside world via the V.24 interface. Software has been written to ease the communication with the DP. Every link has a special number and so has every timeslot (TS) on that link. The most important commands and their syntax are given below:

set <in_link> <in_TS> <out_link> <out_TS>	copies a specific TS of a link to a specific TS of a link;
switchlink <in_link> <out_link>	connects a complete frame from the in_link to the out_link;
fixpattern <TS> <pattern>	from the pattern generation "link" fixed the 32 patterns of either a certain slot can be "set" to a slot of another link, or, all 32 patterns may be "switched" to another link;
reset	forces a full HW reset of the platform.

The **DSP-board** is used to perform application processing within the Ericsson AXE 10 Group Switch. It contains 3 TMS320C542 DSPs from Texas Instruments (TI) to provide the processing power, a Motorola DP to orchestrate the DSPs and to communicate with the outside world, and a DL2 interlink with the necessary interfacing hardware. The other outside connection is the V.24, which has only access to the DP. The board has additional RAM for the DSPs and a flash EEPROM to load the program code on "power-up".

2.2 THE DIGITAL SIGNAL PROCESSOR

Digital signal processing is one of the fastest growing fields in modern electronics. Since the 1980s many of the analogue transport, storage and processing has subsequently been exchanged by digital schemes. This will still continue, for consumer equipment such as television, telephones and radio are on the verge of becoming digital. Digital signals have the strong advantage that processing can be done much easier. Digital Signal Processors (DSPs) are specially designed to fulfil the job.

The architecture of standard microprocessors are unsuited to perform the desired processing. DSPs are specially tailored to basic operations like convolution, correlation, filtering, FFT, multiplications and additions. The built-in hardware and instruction set, mostly provide parallel processing that enlarge the throughput of the data. Because of its tremendous speed, accessing external memory would decrease the performance. On-chip RAM (and sometimes ROM) alleviates this problem.

Every DSP-manufacturer offer chips with special advantages and features. Architecture and instruction set will vary strongly. The drawback is that if an algorithm has been written for a certain DSP it cannot be used on another one. Portability can be achieved by high-level programming languages like C or Turbo Pascal. But, compiling the high-level program to use all the features of the specific DSP and thus minimising speed and program size is extremely troublesome. For clear reasons, programming in a high-level language is much more speedy than in assembly.

One of the main differences between DSPs is the existence of fixed-point and floating-point versions. Floating-point has a much higher dynamic range and therefore it is almost impossible to get an overflow. The limited precision of 16-bit fixed-point compared to 32-bit floating-point is another drawback for fixed-point versions. But, there are also advantages for the latter. Generally, they are faster, smaller, use less power and are less expensive (upto a factor 10). Fixed-point DSPs are mainly used for commercial products that are not high-end and high-performance.

Texas Instruments still is the leading force in DSPs. Since 1982 they have introduced 8 generations: fixed-point, floating-point and multiprocessors.

2.2.1 THE TMS320C542 DSP

The TMS320C54x generation is a fixed-point processor and is designed to perform complex computation-intensive signal processing in real time. It has a high-performance pipelined architecture that enables it to execute each instruction at the maximum rate of 25 ns per instruction (40 MIPS). The Enhanced Harvard architecture is built around eight major 16-bit buses. There are three data buses and one program bus. The main characteristics are:

CPU:	A 40-bit Arithmetic Logic Unit Two 40-bit accumulators (8 guard bits, a high-order and low-order word) A barrel shifter A 17 x 17-bit multiplier (2s-complement) A compare, select and store unit (CSSU)
Memory:	10K words RAM (dual access per clock cycle) 2K words ROM

Peripherals: A Buffered Serial Port (BSP)
A time-division multiplexed (TDM) serial port
A 16-bit timer
Wait-state generator
On-chip phase-locked loop (PLL) clock generator

Instruction set: The total instruction set consists of about 124 instructions. They can be found in the User's Guide [TI_user]. Some special instructions are:

MAC: performs multiply and accumulation in one machine cycle
FIRS: to implement symmetrical FIR filters, produces MACs in parallel with additions
LMS: performs MAC and a parallel add with rounding to efficiently update FIR-coefficients
SQDST: performs a MAC and a subtract in parallel to calculate Euclidean distance

The pipeline of the 'C542 is six levels deep. In the pipeline, the instruction pre-fetch, fetch, decode, access, operand read, and execute operations are independent, which allows overlapping execution of instructions. During any given cycle one-to-six different instructions can be active, each at a different stage of completion. Incorrect operation may occur when an instruction is updating a CPU register in phase *p* and another instruction is reading the register at the same time or during phases *p-1* or *p-2*. Then, extra time is needed to ensure the register values are correct. The time period is called pipeline latency. When latencies should be implemented is described in the User's Guide [TI_user]. More Pipeline management is not required.

Next, the reader may find the most important features depicted in Figure 10 and Table 11.

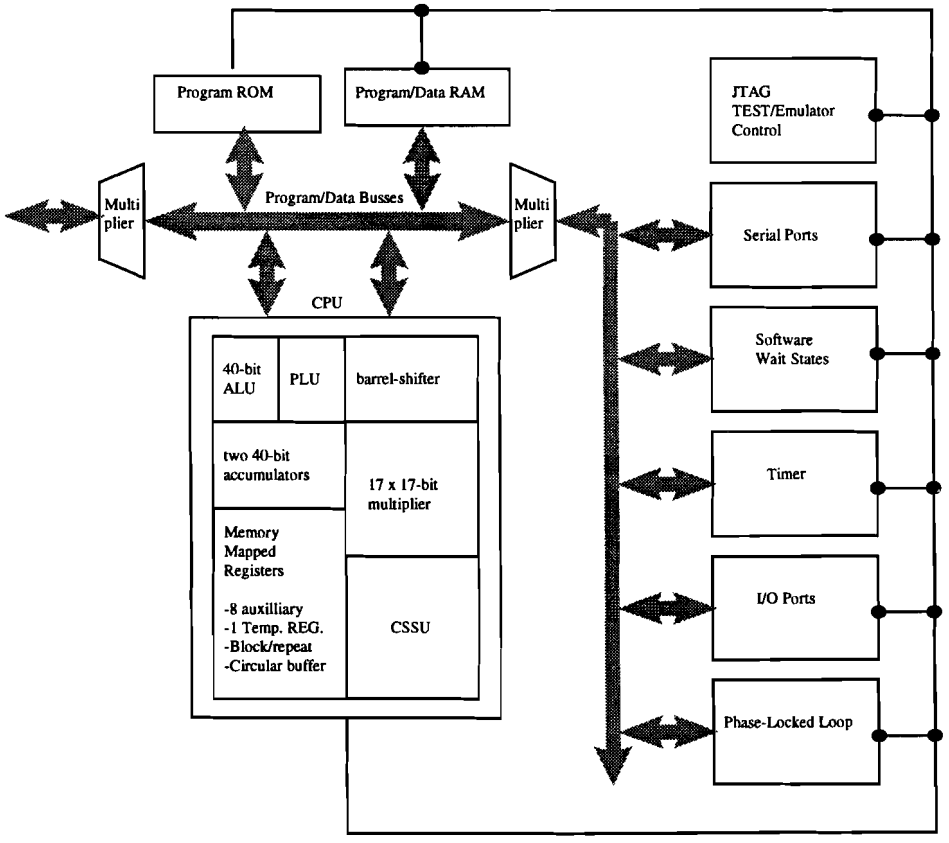


Figure 10 Key Features of the TMS320C542 Architecture

FEATURES	BENEFITS
Enhanced Harvard architecture	Simultaneously accesses instructions and data operands
Parallel Logic Unit	Allows direct bit manipulation on memory operands
Repeat-block loop	Reduce overhead of looped code
Memory-mapped I/O ports	Efficiently handle peripheral data transfer
Circular Buffers	Implement queues, delay lines, circular convolution, etc.
Hardware multiplier	Supports single-cycle signed and unsigned integer manipulation

Table 11 Features of the TMS320C542

2.3 THE DSP FIRMWARE

For another application for the Ericsson Switch, DSP Firmware was written. Since the same board is used and thus all communication and synchronisation via the DL2 has already been resolved, the Echo Canceller and speech compression algorithms will be embedded in this software-environment. This paragraph briefly describes the Firmware.

Synchronisation will be provided by the other hardware on the board. An interupt ('BspIntr') is generated to all DSPs at the same time via the Buffered Serial Port. The interupt will start the Timer unit, which checks the synchronisation and sends a 'tick' to the scheduler. In other words, it calls the scheduler. This tick is the pacemaker of the program. The scheduler is of the weighted round-robin type.

The processes in the Firmware have three levels of impartance. The most important are 'Frameprocesses' that need to be finished within the end of the current frame (within 5000 clock cycles), thus, they may never be interupted. 'Process 1' and 'Process 2' functions may be interupted, where the first has a higher hierarchical level. Regarding the background jobs, all required contents of registers are preserved when interupted. The scheduler is the master controller for the execution of the units. The Program Control Flow is shown in the Figure 11.

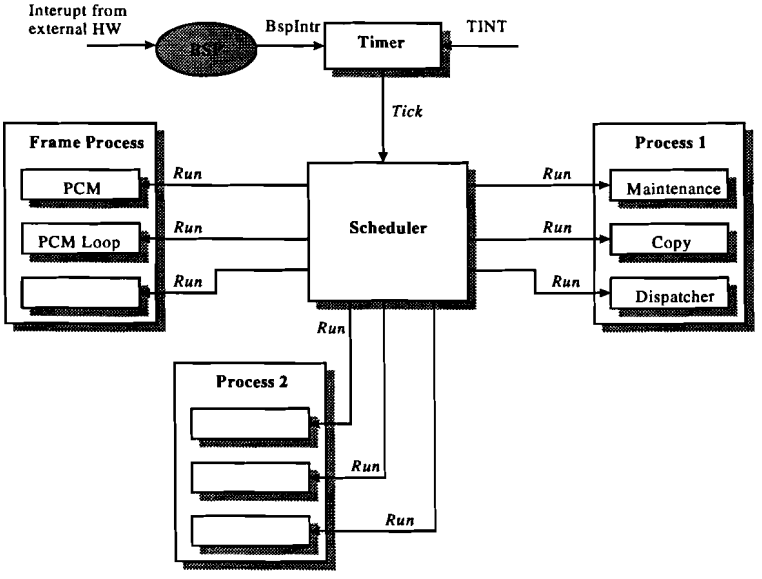


Figure 11 Program Control Flow

The next part explains the various aspects of the units. The information is derived from several internal company documents.

PCM: The unit is activated almost immediatly after the 'BspIntr' has been received. It contains a PCM receiver and transmitter block. Two double 32 word buffers are allocated in memory, one for PCM-receive and the other for PCM-transmit. During a frame, half of the buffer will be stable, this part will be used by the DSP to either read from the PCM-receive buffer, or write to the PCM transmit buffer. The other half will be used by the Buffered Serial Port (BSP) to read or write the incoming or outgoing frame. The stable 32 timeslots are expanded to linear samples. The input samples can be encoded per timeslot according to A-law or μ-law. The appropriate coding is set by the DP through the Host Port Interface. The DP has direct access to the third 2K block of the DSP memory via the Host Port Interface (HPI). The DP can either read or write in this memory block.

Dispatcher: The Dispatcher must read host's commands via the Host unit (if the 'NewMsg' flag is set) and dispatch them to the destination unit and sending an acknowledgement signal to the host upon completion of host signal (the 'NewMsg' flag will be reset). Moreover, the Dsipatcher handles the communication between DSP and DP.

Maintenance: This program performs two different self test routines: an on command comprehensive self-test (e.g., program checksum) and self-test tasks that run continuously as background jobs (such as transparent or attenuate loop tests).

Copy: Provides means to access the DSP memory from the host. Memory refers to the complete DSP data memory, the host memory included. If a large block is specified, a few words are transferred each frame.

PCMloop: A test pattern is transmitted through the BSP. If the loop is set up correctly, the transmitted pattern will return to the receive port within a few frames.

Other units that can be seen as 'overhead' are listed below.

Host: It has to interface the DP through the HPI. It contains symbolic definitions for memory addresses and subroutines to access the HPI memory. It is directly connected to the Dispatcher.

Timer: The Timer unit takes care of generating the system tick (the pace maker) by receiving a frame synch from the BSP. If no synchs arrives, the ticks are generated by interrupts from the DSP's HW timer ('TINT'). This ensures that the host can continue to communicate with the DSP, even when no frame synch is present. After generating the tick it calls the Scheduler unit.

Status: The Status unit takes care of sending status information to the host. It contains symbolic definitions for memory addresses and macros to access the status word in the HPI memory. The status word contains the information regarding Clock errors, Frame synch errors, loop errors, illegal argument errors, parity error and so on.

Kernel: It calls the initialisation routines of the other units and the idle loop of the scheduler unit.

Except for the PCM unit all other units written in C programming language. The PCM unit was analysed to be time-critical and thus was written in assembly. Still, PCM-receive and PCM-transmit consume near 1500 out of 5000 clock cycles per frame.

2.4 SET-UP AND TESTING OF THE TEST BED

Before the Firmware could be used some modifications had to be made. The PCM-transmit and PCM-receive had to be combined on a single DSP. This was necessary because the original application required them to be separated. The specific application software was removed to make place for the ADPCM and echo canceller algorithm. Because the DP's Specific Application Program (SAP) was not used, the Firmware had to be altered to automatically select the compression law. A string had to be added to the calls of the PCM-transmit and PCM-receive routines. Another important issue was that the unused outputs of each DSP had to be put in tristate. Otherwise they would corrupt the correct outputs, because the outputs of all DSPs are connected together. And, in order to be able simulate the developed programs Look-Up-Tables (LUTs) had to be added to PCM-receive. On the real DSP the LUTs for expansion of the incoming samples are copied from the ROM.

After connecting all subsystems the EEPROM of the ZAK was programmed to connect the appropriate channels to the desired timeslots of the PCM link. The initial test was to loop back voice from one telephone to another via the MUSSE. When this proved working, the DSP-board was connected and application code was downloaded onto the DSP(s).

In order to test the entire test bed, the assembly code for the delay, necessary to simulate long distances telephoning, was written. This tiny program has been included in Appendix A (Note, due to strict implementation constraints of the circular buffer it was difficult to fit the buffer in the existing memory configuration). It has been simulated using the Texas Instruments simulator version 2.10 for the 'C54x series [TI_sim]. The simulations were performed without and within the Firmware mentioned in the previous paragraph. Texas Instruments states that the device simulator should have identical performance as the real device.

After simulating the delay it was added as a Frameprocess within the schedular of the Firmware and the entire code was compiled and downloaded onto the board via the V.24 interface. It took a pretty long time before all the errors were solved! Since they were too specific in nature, they are not explained any further. When this work was done a second delay was added, but now with a different DSP. Since the first showed correct functioning, it seemed best to try another to see if multi-processors were able to work together. The idea was to use two DSPs, one for the application, being the Echo Canceller, and the other for the simulated transmission time. The adding of the second delay was much less troublesome.

In all, it proved that the Pattern Generator of the MUSSE was very helpful for debugging. Fixed values could be sent to the DSPs. Subsequently, the delay would send fixed values to the output and the MUSSE would loop this timeslot back towards the DSP-platform. This was necessary because only the remote terminal that was connected to this platform has means to view the contents of timeslots. For this, the 'Copy'-function within the Firmware was a big help, since it gave a means to copy data outside the HPI-memory to a specific part in this memory. Again, via the DP, the remote terminal could retrieve these values. The maximum number that could be copied at a time was bounded to a value of 16.

From this point the test bed was ready for the Echo Canceller and the ADPCM transcoder. The basic set-ups of the system for both type of algorithms are shown in the Figure 14a and 14b

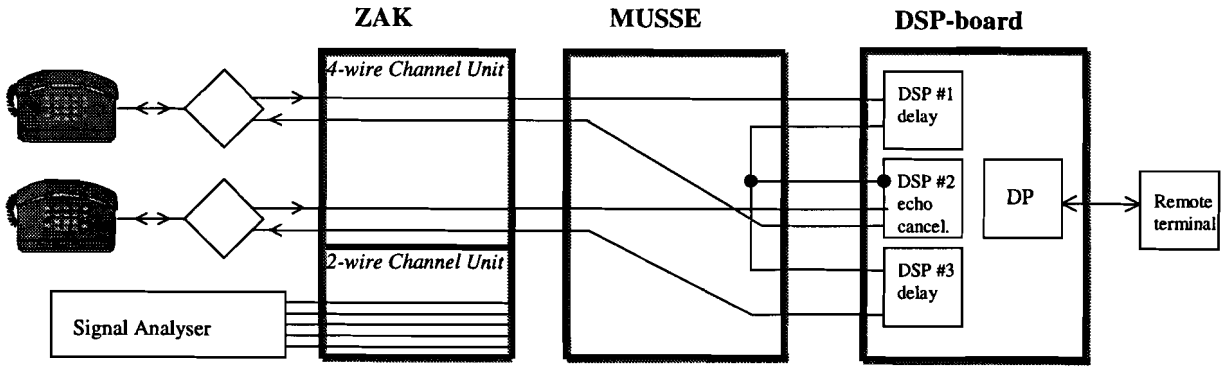


Figure 14a Basic set-up of the system for testing the echo cancellation algorithm

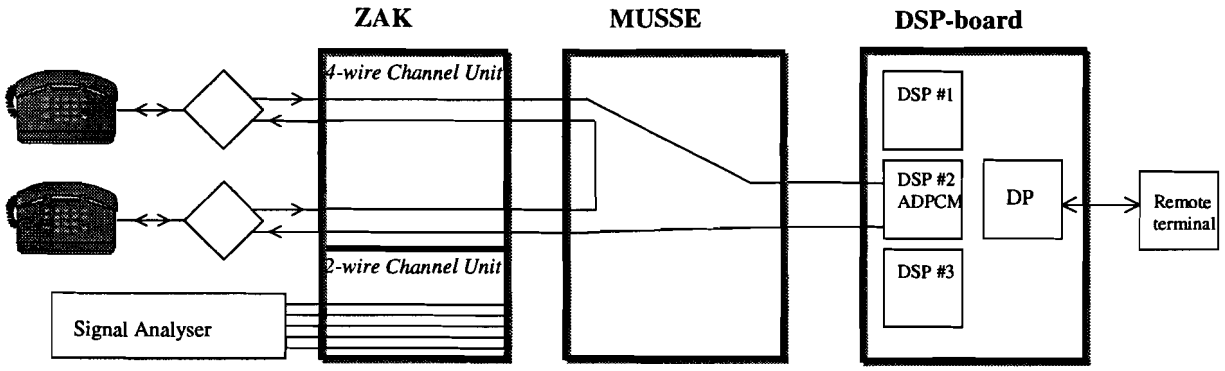


Figure 14b Basic set-up of the system for testing the ADPCM speech codec

Figure 14a shows that two delays are used for the testing of the echo canceller. Why this was done in this way is explained in Chapter 3.

A maximum of 5 different signals may also be viewed on the Signal Analyser. The MUSSE has to connect these signals to the appropriate timeslots of the PCM link that interconnects the ZAK and the MUSSE. The Signal Analyser is a product from Analog Instruments and is called LabVIEW. It is a virtual instrument on a PC and has oscilloscopes and spectrum analysers.

The disadvantage of the system is that there is no digital output from which the data could be monitored. If this is indeed required an extra board has to be bought for the ZAK that has digital output channels. These digital channels may be linked to a PC or UNIX station. It is also possible to use one DSP for the performance evaluation. The (intermediate) results can be read on the remote terminal. The test environment provides the possibility to test with voice-band data from modems of faxes. It requires a simple exchange with the telephones. The mentioned signal generator may also substitute a telephone to test with common analogue signals and not with speech. Finally, it is possible to connect additional hybrids and telephones if necessary.

CHAPTER 3

IMPROVEMENTS TO A TIME-DOMAIN ECHO CANCELLER

Due to a constant globalisation, international communication via telephony networks shows a continuous upgoing trend. Because of imperfection of the network, to be explained later, a delayed, distorted replica from the talker will be echoed back to the speaker. In case of local calls this echo will just be added up to the talkers sidetone. But, since international calls involve long delays the echo becomes more noticeable and more annoying. In order to improve the quality of the network this echo ought to be cancelled.

The type of echo cancellation mentioned above is called 'line echo cancellation'. Another type is 'acoustic echo cancellation'. The latter is increasing in importance since handsfree telephony and videoconferencing are becoming more popular. The acoustic signal from the loudspeakers(s) is reflected by the walls and the ceiling, are intercepted by the microphone(s) and echoed back to the talker.

Both echo types are mentioned, since they can be cancelled by means of the same adaptive filtering principles. The emphasis of this chapter will be on the first, however. This is, because the improvements of the algorithm that was proposed by an Ericsson employee only deals with this type. The proposal is outlined in the third paragraph and is followed by the simulation and laboratory results in the fourth and fifth paragraph. The structure of this chapter is shown below.

3.1 GENERAL ISSUES OF ECHO CONTROL

As was mentioned, line echo (further denoted as just: echo) is caused by an imperfection of the network. In order to lower the costs of a telephony network, the subscriber lines (local loop) are a 2-wire twisted-pair. Communication in both directions takes place over these 2 wires. On the other hand, the network consists of 4 wires. The 4-wire network is needed since for long distances amplification is required for either direction. To interface the 4-wire network with the 2-wire subscriber line a hybrid is used. In Figure 6 it is shown where the hybrid can be found in the network.

If the impedance of the local loop is not perfectly balanced in the hybrid, the 2-wire port coming from the PSTN will be coupled to 2-wire port returning to the PSTN, thereby giving rise to an echo. By definition, the part where the echo originates is called "near-end", the opposite side the "far-end". A "Double-talk" situation occurs when at least the near-end talker is speaking. During this condition adaptation is inhibited, but filtering will proceed. International agreements have imposed the demand on network operators to control the echo at its origin. However, this is not always done.

The degree of echo observability depends on two main factors, round-trip delay and the loudness of the echo that will arrive at the earpiece of the talker. Round-trip delay will increase with the transmission distance. But, it also depends on the transmission type, like: coaxial cables, optical fibre and the number of satellite hops. These and other values can be found in ITU-T Recomendation G.114. Furthermore, extra delay is

caused by multiplexing, switching, voice processing, etcetera. Especially vocoders, which will be explained in the next chapter, may cause fairly large delays.

The ITU-T developed a standard to determine when Echo Control should be applied. Two basic parameters are used: Talker Echo Loudness Rating (TELR) and Mean One-way Transmission Time. The loudness of the echo returned to the talker is defined as TELR. This parameter takes the losses of the network, the local loop and across the far-end hybrid into account. Once the TELR and the mean transmission time T are known, the following graph, taken from [G.131], shows when to use Echo Control. For a certain mean one-way transmission time the TELR should be higher than the top curve. This curve is equivalent to 1% probability of encountering echo. The limiting case (-6dB) ought to be used only in exceptional circumstances. Because the conditions of links are not known before, and therefore neither are the parameters, ITU-T recommends to implement Echo Control on all connections which exceed the total one-way talker transmission path time of 25 ms.

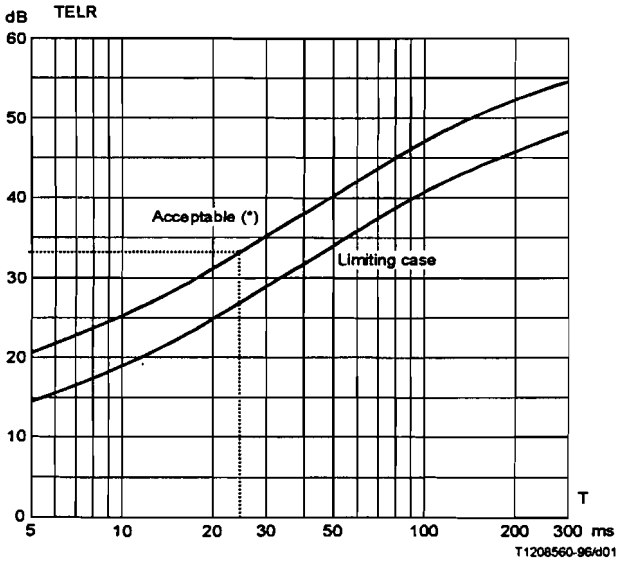


Figure 16 Talker echo tolerance curves (G.131)

The European Telecommunications Standards Institute (ETSI) more recently developed a computation model to guide network operators in assessing transmission quality of connections. This is called the E-model, and it uses ideas from the older models, complementing them with results from more recent subjective tests. Among the important additions is the inclusion of psychological factors and low bit-rate effects. E.g., GSM users already take a lower expected voice quality for granted. Recently, some ITU-T documents on the planning of private networks have used this model. A comparison between the ITU-T limits and the ETSI predictions has been made by Johannesson [2], who indicates that in order to obtain 85% satisfaction, a TELR of 34 dB is required. This figure confirms the ITU-T Reccomendation for the need of Echo Control with a mean one-way transmission time higher than 25 ms. The E-model is explained in [ETSI_3].

There are three ways to Control Echo. One is to increase the TELR by adding a loss in the trunk network. These losses are part of the **loss plans** (e.g., VNL and FL) in telephone networks in order to increase the stability margins during the actual connection and the setting-up and clearing of the link. But it can also be utilised to raise the echo loss. Knowing that the SNR of communication on the PSTN does not exceed 37 dB, it is clear that there is a limit to augmenting these losses.

A second method is the use of **Echo Suppressors**. This type physically blocks the echo return path. This method mainly causes problems during interruptions. Suppressors mostly are low-cost devices that have little sophisticated means to determine if the returned signal is either low-level near-end speech or echo. Parts of the conversation will then be erased. The drawback of this type of echo control is enlarged with increasing transmission delays. This is clear, for interruption occurs more with higher delays. With a one-

way transmission time of 500 ms more than 60% of the people perceived considerable problems (this was only about 15% when Echo Cancellers were used). The described suppressor is a *hard* type [G.164]. The *soft* type does not inhibit passing through of the signal but attenuates the signal by a fixed loss. A loss of more than 12 dB is not recommended, otherwise the false intrusions would become too noticeable.

The best solution is attained when **Echo Cancellers** are implemented. They are much more complex than echo suppressors but yield far better results. Nowadays network operators opt for quality. Firstly, the increasing competition requires a good product. Secondly, because good quality of speech increases the revenue of the operator, the call duration will be longer.

3.1.1 ECHO CANCELLATION

Echo cancellation requires adaptive filtering techniques. Since these are computationally complex this can only been done thanks to increased computation power of DSPs and ordinary processors. The generic form of an echo canceller is shown in Figure 17.

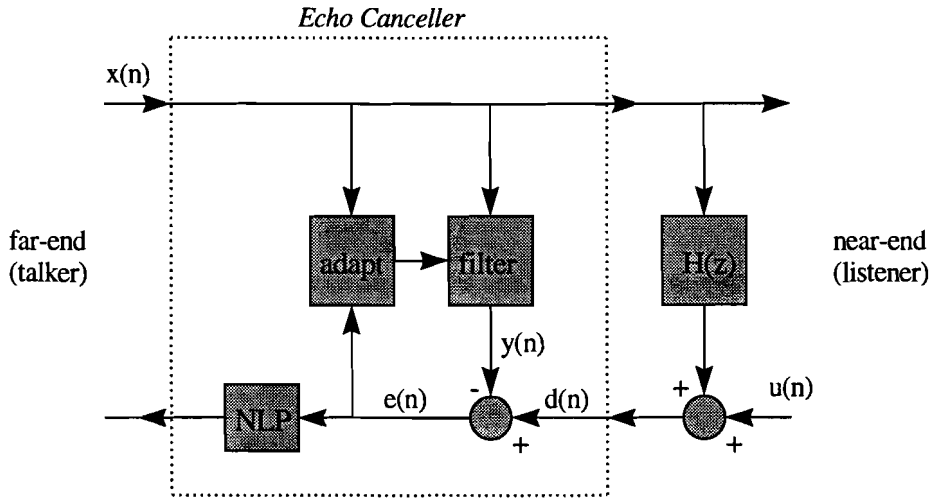


Figure 17 Generic form of an echo canceller

Generally, the algorithm attempts to model the unknown response H(z). This response (echo) may be caused by a hybrid (electric echo), or by a room (acoustic echo). The echo canceller is divided in three parts: a filter, a filter adapter, and a Non-Linear Processor (NLP). The filter will generate an estimate, *y(n)*, of the actual echo, *d(n)*. The adaptation adjusts the filter in order to yield a better estimate of the echo. The estimated error will be subtracted from the real echo. The remaining error (residual echo), *e(n)*, will then pass through the NLP which will further erase the signal, below a predefined value. See Figure 18 for a standard center-clipped NLP.

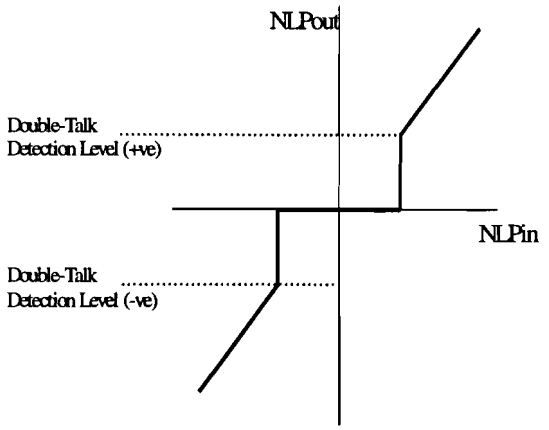


Figure 17b Standard center-clipped NLP

The "Double-talk" levels, in the Figure is determined by the noise-level in the network and the residual echo that was not erased by the Echo Canceller. Below this level the signal is replaced with noise with an equivalent level as the typical network noise. This noise is designated as 'Comfort noise'. Input signals that are higher than the "double-talk levels" are assumed to be near-end speech and will be passed untouched. If the NLP was in "clipping-mode" it will take a certain amount of time before the signal is again fully passed through. This time is generally called 'hangover-time'. The difference between echo suppressors and NLPs is that the intrusion on the signal is far less for a NLP. The Echo Canceller has already lowered the returned signal by a level between 10 and 35 dB. These values depend on the performance of the adaptive filter and on non-linearities in the echo path. The non-linearities include noise, the maximum obtainable echo reduction is always the limited by the noise-level.

The next list shows the main parameters that the performance of the adaptive filter can be judged on:

complexity: the computational load and memory requirements of the algorithm for the hardware, as well as the time needed to program the algorithm on a computer;

convergence rate: the number of iterations that are needed to closely reach the optimum solution;

misadjustment: the difference between the optimum solution and the mean-squared error, after convergence;

robustness: the capability of the adaptive filter to operate satisfactory in spite of ill-conditioned input data, being numerically stable and insensitive to variations in wordlength (numerical inaccuracy);

tracking: the ability of the filter to track the statistical variations in the environment.

The aforementioned parameters are very important for the determination of the quality of the adaptive filter. Regarding echo cancellers, it is commonly known that the double-talk detector and NLP are also very dominant factors. While the first can cause serious divergence of the echo canceller, the latter may erroneously pass through the echo, or prevent near-end speech from passing.

3.1.2 TESTING OF AN ECHO CANCELLER

A large number of subjective and objective tests exist in order to test the performance of Echo Cancellers. The ITU-T has recommended a set of tests that are described in [G.165]. Modifications and additions were made to this Recommendation and these are combined in [G.168]. The last Recommendation is surely a better representation of the 'Real World'. But, still a lot of tests are unfinished, either the procedure or the

requirements are not yet finalised. Despite this fact, the tests from G.168 will be described and reference will be made to the older version.

Main difference between [G.165] and [G.168] are:

- the test signal: The test signal has been changed from white, or coloured, noise to Composite Source Signal (CSS) [P.340]. This CSS represents the real situation much more than noise. E.g., the CSS includes pauses that will cause the canceller to converge at a slower rate. Also, the speech level is increased by 10 dB since the level in todays networks are higher.

- the echo tail: It has been recognised that a more complex echo response can be found in the network than specified in [G.165]. Therefore, the echo tail in [G.168] consists of a multi-path echo with different losses. The values may be chosen to represent real figures. This echo-tail still does not resemble the real situation, but comes closer than the old version.

Some tests are done with and without NLP, for Network Operators sometimes do not want to use NLPs. Reccomendation [G.131] advises not to use an NLP when Echo Cancellers are placed in tandem. Also, tests without the NLP obviously reveal the true quality of the Echo Canceller.

The separate tests can be found below, specific values and figures can be found in Recommendation [G.168].

TEST 1 MISADJUSTMENT: Determining the residual echo for any input signal level. Maximum values can be found for every input level. The test should be done with and without NLP.

Requirements: see [G.168].

TEST 2 CONVERGENCE RATE: To insure that the Echo Canceller converges rapidly for all combinations of input signal levels and echo paths and that the returned echo level is sufficiently low within a fairly short time. Divided in:

2A) With NLP.
Requirements: converged within 1 seconde.

2B) Without NLP.
Requirements: ERLE greater than 10 dB within 1 second.

2C) In the presence of noise (-10 dB compared to input signal).
Requirements: convergence time and value not yet determined.

TEST 3 DOUBLE-TALK DETECTOR:

3A) Double talk test with low near-end signal.
Requirement: 9 dB cancellation with in less than 5 (merely guideline) seconds.

3B) double talk test with high near-end levels.
Requirements: Echo Canceller should remain converged to D dB (value D under study) of the requirements in test 1.

3C) double talk under simulated conversation (**new**); to test if EC does not produce undesirable artifacts during and after double talk.
Requirements: Test has not been finalised.

TEST 4 LEAK RATE TEST: This test is meant to ensure that the leak time is not too fast, i.e. that the content of the H register (tap-weightsof the filter) does not go to zero rapidly. The only difference with [G.165] is the use of CSS.

TEST 5 INFINITE RETURN LOSS CONVERGENCE TEST: This test is designed to ensure that the Echo Canceller has some means to prevent the unwanted generation of echo. This may occur when the H register contains an erroneous echo path model, either from a previous connection or the current connection, and the echo path is opened. The test procedure are the same as [G.165], the requirements however are not yet determined.

TEST 6 NON-DIVERGENCE ON NARROW-BAND SIGNALS: Determination if the Echo Canceller diverges when subscriber-originated narrow-band signals, for voice-mail and other services that require DTMF, or other tones. This test was optional in G.165, now mandatory. Procedure:

1) let the Echo Canceller totally converge,
2) add mono or bi-frequency,
3) inhibit adaptation,
4) add CSS input signal,
5) and, measure the residual echo.

TEST 7 STABILITY TEST: The object of the test is to verify that the Echo Canceller will remain stable for narrowband signals. The residual echo is measured after the application of a sinusoidal tone. The Echo Canceller has to converge fully within 2 minutes. Convergence is defined as in Test 1.

TEST 8 (optional) NON-CONVERGENCE OF ECHO CANCELLERS ON SPECIFIC ITU-T NO. 5,6,7 IN-BAND SIGNALLING AND CONTINUITY CHECK TONES:

This test is designed to ensure that Echo Cancellers ,which are not externally disabled by the switch, will not cancel signals transmitted in a handshaking protocol in the transmit direction either before or after receiving an identical signal (except for amplitude and phase) in the receive direction. This is intended to allow correct transmission of specific signalling and check tones without requiring that the Echo Canceller be externally disabled.

TEST 9 COMFORT NOISE TEST: Intended to test if the Echo Canceller is able to provide a comfort noise signal, and if the Echo Canceller is able to compensate for changes in the level of input noise. White noise is used instead of CSS.

TEST 10 (new) FACSIMILE TEST: Several tests are included to ensure that the Echo Canceller converges rapidly on initial FAX handshaking sequences. Therefore a variety of FAX signals are transmitted.

TEST 11 ECHO CANCELLERS PLACED IN TANDEM: Because it has been found that some Echo Cancellers do not work well when placed in tandem this test is added. The test is still under study.

TEST 12 (new) RESIDUAL ACOUSTIC ECHO TEST: The test is meant to ensure that the Echo Canceller continues to operate properly in the presence of acoustic echo. The test is still under study.

TEST 13 (new) PERFORMANCE WITH LOW BIT RATE CODERS IN END-PATH: Still under study.

TEST 14 (new) PERFORMANCE WITH V.SERIES LOW SPEEDS DATA MODEMS:
Mentioned modems do not turn off Echo Canceller. Bir-error rate test is made while the Echo Canceller is operating in a simulated network connecting two modems.

3.2 ADAPTIVE FILTERING ALGORITHMS

Basically, we may identify three distinct methods for deriving recursive algorithms for adaptive filters. These will be discussed in the following three sub-paragraphs. Each section will contain the main characteristics and performance. The emphases will lie on the Wiener filter theory, since the implemented algorithm is a derivative of this theory as will become clear in the next paragraph. Additional information can be found in [7-10].

3.2.1 ALGORITHMS BASED ON WIENER FILTER THEORY

The first studies of minimum mean-square estimation in stochastic processes were made during the 1930s. A major contribution came from Wiener. He formulated the continuous-time prediction problem and derived an explicit formula for the optimum predictor. The optimum estimate required the solution of an integral equation known as the Wiener-Hopf equation (1931). In 1947, Levinson formulated the Wiener problem in discrete-time.

When looked at Figure 17 we may define the error signal e(n) as

$$e(n) = d(n) - y(n) \qquad \textbf{[21a]}$$

where d(n) denotes the desired response (the near-end signal $u(n)$ is assumed to be zero). The estimate of the response y(n) is defined as a linear convolution sum

$$y(n) = \sum_{k=1}^{M} w_k x(n-k) \qquad \textbf{[21b]}$$

To optimise the filter design, the mean-square estimation error (*mse*) is chosen to be minimised. The *mse* is mathematically tractable and has a uniquely defined bottom. The cost function will thus be defined as

$$J(n) = E[\,|e(n)|^2] \qquad \textbf{[21c]}$$

The optimum will be attained if all first-order derivatives equal zero. This leads us to

$$\nabla_k(J) = -2E[x(n-k)e(n)] = 0 \qquad \textbf{[21d]}$$

The formula above states that the cost function J attains its minimum value when the estimation error is orthogonal to each input sample, or in other words, the input samples are uncorrelated to the estimation error e(n). Rewriting the formula leads us directly to the Wiener-Hopf equations

$$\sum_{i=1}^{M} w_{oi} E[x(n-k)x(n-i)] = E[x(n-k)d(n)] \qquad \textbf{[21e]}$$

The first expectation is equal to the autocorrelation function (acf) of the filter input for a lag of i-k. The second expectation is equal to the cross-correlation between the filter input and the desired response d(n) for a lag of -k. Using matrix formulation in order to obtain the optimum tap weights, yields

$$\mathbf{w}_o = \mathbf{R}^{-1}\mathbf{p} \qquad \textbf{[22a]}$$

where $\mathbf{w}_o$ denotes the M-by-1 optimum tap-weight vector of the transversal filter, $\mathbf{R}^{-1}$ is the inverse of the M-by-M correlation matrix of the tap inputs (the matrix **R** is assumed to be nonsingular), and **p** denotes the M-by-1 cross-correlation matrix between the tap inputs of the filter and the desired response d(n). The optimum situation of the filter is generally referred to as the *Wiener solution.* The filter coefficients may be computed using the recursive Levinson-Durbin algorithm that makes practical use of the Toeplitz structure of the autocorrelation matrix.

The oldest gradient-based adaptation that converges to the Wiener solution is the *Method of Steepest Descent.* The method is recursive in the sense that it starts form a arbitrary starting point value for the tap-weight vector and it updates the tap weights in the following manner

$$\begin{aligned} w(n+1) &= w(n) + \tfrac{1}{2}\mu[-\nabla(J(n)] \\ &= w(n) + \mu[p - Rw(n)] \end{aligned} \qquad \textbf{[22b]}$$

The gradient vector $\nabla(J(n)$ has already been derived in Formula 21d. The step size μ is a positive real-valued constant. The method of Steepest Descent still requires a fairly large amount of computations. Therefore, Widrow and Hoff developed the algorithm known as the *least-mean-square (LMS) algorithm.* They replaced the estimators for **R** and **p** by instantaneous estimates based on sample values of the tap-input vector $\boldsymbol{u}(n)$ and the desired response $d(n)$, as follows

$$\mathbf{R} \rightarrow \mathbf{x}(n)\mathbf{x}^T(n) \qquad \textbf{[22c]}$$

and

$$\mathbf{p} \rightarrow \mathbf{x}(n)d(n) \qquad \textbf{[22d]}$$

These instantaneous estimates yield a simplified recursive relation for updating the tap-weight vector

$$\mathbf{w}(n+1) = \mathbf{w}(n) + \mu\mathbf{x}(n)e(n) \qquad \textbf{[22e]}$$

Widrow proved in 1970 that the LMS algorithm would be convergent in the mean-square provided that the step size parameter would not exceed the next boundaries

$$0 < \mu < \frac{2}{\lambda_{max}} \qquad \textbf{[22f]}$$

$$\sum_{i=1}^{M} \frac{\mu\lambda_i}{2 - \mu\lambda_i} < 1 \qquad \textbf{[22g]}$$

where λ_i ,i=1,2,..,M denote the eigenvalues of the correlation matrix **R,** and M is the number of taps; λ_{max} is the largest eigenvalue of **R** (if $\mu \ll 2/\lambda_{max}$, then λ_{max} may be substituted by the *total input power).* The final misadjustment is proportional to the step size, the number of taps and the eigenvalues as can be seen in the next formula

$$M = \frac{J_{ex}(\infty)}{J_{min}} = \frac{\sum_{i=1}^{M} \mu\lambda_i/(2 - \mu\lambda_i)}{1 - \sum_{i=1}^{M} \mu\lambda_i/(2 - \mu\lambda_i)} \qquad \textbf{[22h]}$$

where $J_{ex}(\infty)$ is difference between the mean-squared error at time $n\rightarrow\infty$, minus the minimum value, J_{min} (Wiener solution). The excess mse would be zero if *a priori* knowledge of the gradient was available. Now, the algorithm suffers from gradient noise. The bigger the step-size μ the faster the algorithm will converge, but the higher the misadjustment and steady-state fluctuations. The first may be seen when looked at the ensemble-averaged learning curve of the LMS algorithm. It was developed for the method of steepest

descent and now approximated by a single exponential with time constant $\tau_{mse,av}$., which is a measure for the convergence rate:

$$\tau_{mse,av} \approx \frac{1}{2\mu\,\lambda_{av}} \qquad \textbf{[23a]}$$

The simplicity is the main benefit of the LMS algorithm. Its computational complexity increases linearly with the number of filter taps, *M*. The memory burden is very small, about twice the number of filter taps. The performance depends mainly upon the number of filter taps, the step-size parameter and the eigenvalue spread. The spread in eigenvalues is defined as $\chi(\mathbf{R}) = \lambda_{max}/\lambda_{min}$, and its negative influence on the performance can be diminished by using the *normalised* LMS (NLMS) algorithm. The modified algorithm is shown below

$$\hat{\mathbf{w}}(n+1) = \hat{\mathbf{w}}(n) + \frac{\mu^{*}\mathbf{x}(n)e(n)}{\delta + \|x(n)\|^{2}} \qquad \textbf{[23b]}$$

where ||x(n)|| is the Euclidean norm of the tap-input vector. The optional constant δ prevents division by small numbers. The step size may now be viewed as being time-varying. The *dimensionless* adaptation constant μ^{*} was thoroughly studied and several researchers found that the next condition has to be satisfied for the NLMS algorithm to be convergent in the mean-square sense [7].

$$0 < \mu^{*} < 2 \qquad \textbf{[23c]}$$

Several different variations of the (N)LMS algorithm exist today, but it will take to far to describe them all. They may use different *time-varying step-sizes* or use Block-adaptation techniques. Or, in order to avoid computationally intensive multiplications between *e(n)* and *x(n)* in (formula 22e) the signs of these variables are multiplied. Three methods exist, using the signs of either or both variables. The Sign-Family is rarely used due to poor performance. Finally, to improve stability of the LMS algorithm in a limited-precision environment a leakage constant may be added to the tap-weights to limit the *memory* of the algorithm. All at the cost of an increased excess *mse* [11].

3.2.2 APPROACH BASED ON KALMAN FILTER THEORY

The Kalman filtering problem for a linear dynamic system is formulated in terms of two basic equations: the *process equation* that describes the dynamics of the system, and the *measurement equation* that describe the measurement errors incurred in the system. The theory requires that we postulate a model of the optimum operating conditions, which serves as a reference for the filter. For a stationary environment a fixed model may be applied. In case of a non-stationary environment, a noisy state-model can be used. For example, the tap weights or state vector of the model executes a random walk around some mean value.

The solutions of the equations are solved recursively. The convergence rate is much higher than for the LMS algorithm. Also the steady-state error is far less. The algorithm is robust in the sense that it is insensitive to the eigenvalue spread. The computational complexity is high $O(M^2)$ and also the memory burden is considerable. Therefore, its main applications lie in the field of aerospace, aeronautics and other demanding purposes.

3.2.3 METHOD OF LEAST SQUARES

The algorithms described before are based on statistical concepts. The approach based on the classical method of least squares differs from these in that it involves the use of time averages. The goal is to minimise a cost function that consists of the sum of error squares:

$$\xi(w_0,\ldots,w_{M-1}) = \sum_{i=i_1}^{i_2} |e(i)|^2 \qquad [24]$$

where i_1 and i_2 define the index limits at which the error minimisation occurs. During this interval the tap weights of the filter are held constant.

Several recursive algorithms exist today that use the least square method. The most important is the *Recursive Least Squares* (RLS) algorithm, which is a special case for the Kalman algorithm for adaptive filters. The algorithm converges in the mean square sense in about 2M iterations, much faster than for the LMS algorithm. In theory it does not produce any excess mse if the environment is stationary. In the case the algorithm is used in a non-stationary environment an *exponential weighting factor* λ may be introduced within the cost function. This increases the excess mse, but diminishes the memory of the algorithm which is roughly equivalent to $(1 - \lambda)^{-1}$ [7,11] Despite this factor the LMS algorithm still is found to have a superior tracking performance compared to the RLS algorithm. Another disadvantage is the computational complexity that increases linearly with M^2. Faster algorithms, named fast RLS or fast Kalman, exist that utilise the redundancy in the Toeplitz structure of the input data matrix and exploit this redundancy through the use of linear least-square prediction in both forward and backward directions. The drawback of these algorithms is that they suffer from numerical instability. This parameter drift may require a periodic resetting of the parameters, or the use of error-feedback [7]. The obvious advantage is that the computational complexity increases with about *O(8M).*

3.3 THE PROPOSED IMPROVEMENTS OF AN ECHO CANCELLER

This paragraph describes the tested algorithm. Information is presented that is relevant for this thesis, more information can be found in the Ph.D. dissertation by K. Elmalki [1].

The core of the Echo Canceller is the NLMS based algorithm. This algorithm has proven to have good properties as mentioned in the previous paragraph. Since a commercial implementation is intended, filtering and adaptation is done in the time-domain. Frequency-domain cancellers typically use much more memory space. They are more appropriate for 'acoustic echo cancellation'.

Before executing the improvements the Echo Canceller is given time to converge. Convergence is started the moment a power-threshold on x(n) is exceeded. After 3000 clock cycles the algorithm is said to be in a sufficiently converged state. Generally, for this type of algorithms convergence is reached after about 10 times the number of filter taps. This number of filter taps is chosen to be 512 in order to be able to accommodate near-end delays up to 64 ms. The H register and the 512 old samples of x(n) are stored in circular buffers.

After 3000 cycles the 'Echo Path Optimisation' (which will be explained in the next sub-paragraph) is performed, which will give the necessary information about the echo path to the 'Silence Detection' (which will be described in sub-paragraph 3.3.2). The 'Silence Detection' is started subsequently.

Unlike the ITU-T Recommendations [G.165] and [G.168], a more sophisticated echo path tail has been used for the simulations. The synthesized impulse response of the hybrid is a sync-like response with a length of 40 taps (5 ms) and an echo loss of 6 dB. Other researchers have used similar patterns but echo losses of around 11 dB. Also ITU-T subscribes this number. It also recommends testing with slow varying or stationary hybrid responses, since there is too little knowledge on this topic. The response is depicted in Figure 25a. An exemplary pure delay is set to 200 samples. The amplitude has been normalised to one.

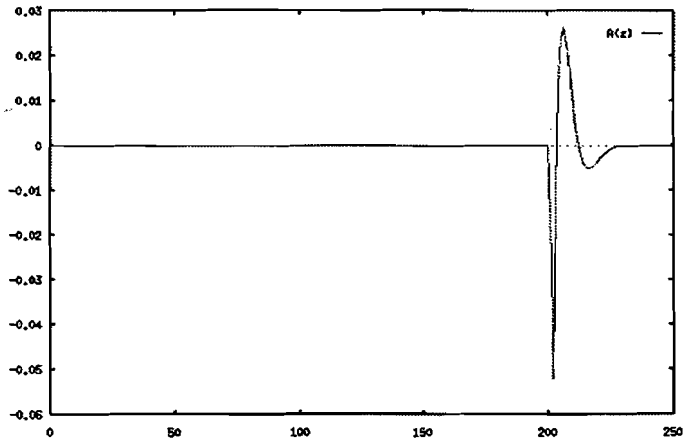


Figure 25a Echo Path Impulse response

Providing 'Comfort noise' can take relatively many cycles. Elmalki attempts to tackle this problem by applying a different NLP-characteristic than has been depicted in Figure 25b. Low-level signals will be passed, medium levels (too high residual echoes) will be clipped and above the double-talk threshold will also be untouched. The modified characteristic is shown below.

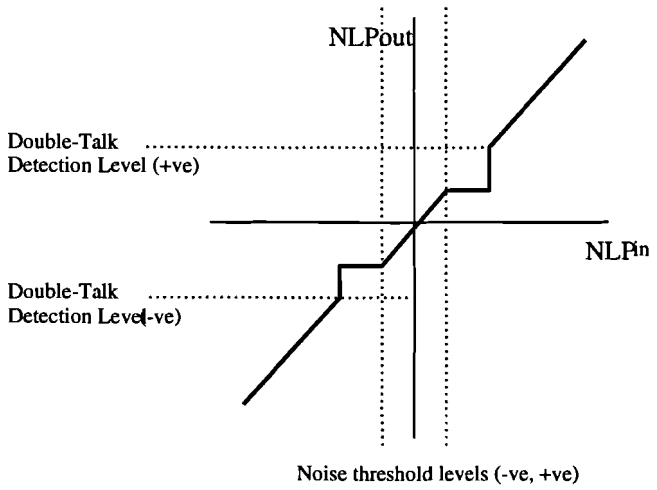


Figure 25b Modified NLP characteristic

3.3.1 ESTIMATION OF THE ECHO PATH CHARACTERISTICS

In order to be able to obtain a good model of the system, adaptive filters need to have a number of filter taps that is large enough to represent the impulse response. For the use in Echo Cancellers there is a second important constraint. Since the echo path delay is unknown beforehand, the filter should be able to accommodate echo path pure delays up to 64 ms. This number is derived from the fact that (near-end) terrestrial delays almost never exceed this figure. There are three major drawbacks to this increased number of filter taps:

1) expensive memory usage with DSPs;
2) degraded performance;
3) computational load.

The first has already been explained in Chapter 2. The second is due to the fact that the irrelevant filter coefficients are not zero, but are corrupted by noise. These accumulative errors give rise to an inaccurate determination of the echo. Also formula 22h shows an increase in mse which is proportional to the number of taps. Thirdly, because only about 10% of the number of filter taps are relevant, the estimation of the echo as well as updating the coefficients consume many computations. These reasons have urged researchers to investigate 'echo path optimisation'.

One of the existing solutions can be found in U.S. Patent 4,736,414 first proposed in 1983. The patent describes the search for the highest absolute value of the filter coefficients. Once this peak, *Wmax,* is found the echo path response is said to be a fixed number of taps away from this peak. In Figure 26 *T1* resembles the echo path pure delay.

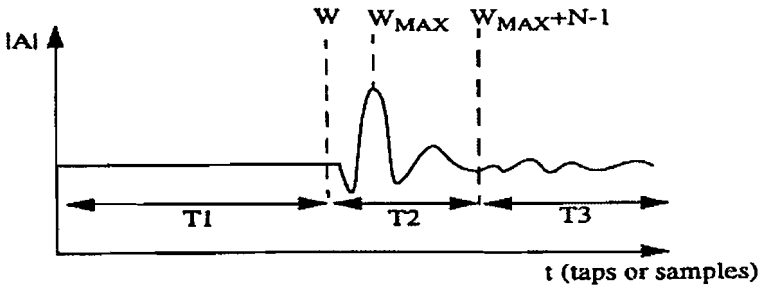


Figure 26 Echo path delay calculations

One drawback is that all filter taps have to be examined, and every new maximum has to be stored. The other is the assumption that W can always be found a fixed integer away from Wmax. This is not necessarily true. Other algorithms are proposed by Sugiyama and Ikeda [3], and, Kawamura and Hatori [4]. While the first is considered to suffer from slower convergence, both are said to be computationally inefficient.

The method proposed by Elmalki will, after convergence, search for the absolute peak in the first 10 samples of the impulse response. Generally this will be a coefficient corrupted by noise. Then, it will look for the absolute peak 'K' times bigger than this value. Where 'K' is a constant larger than four. This peak should NOT be the maximum peak, but the first relevant maximum of the impulse response. When the peak is found, the search is stopped and the first and last relevant taps are determined, being a fixed number away from the relative maximum. The result from this is graphically shown in Figure 27.

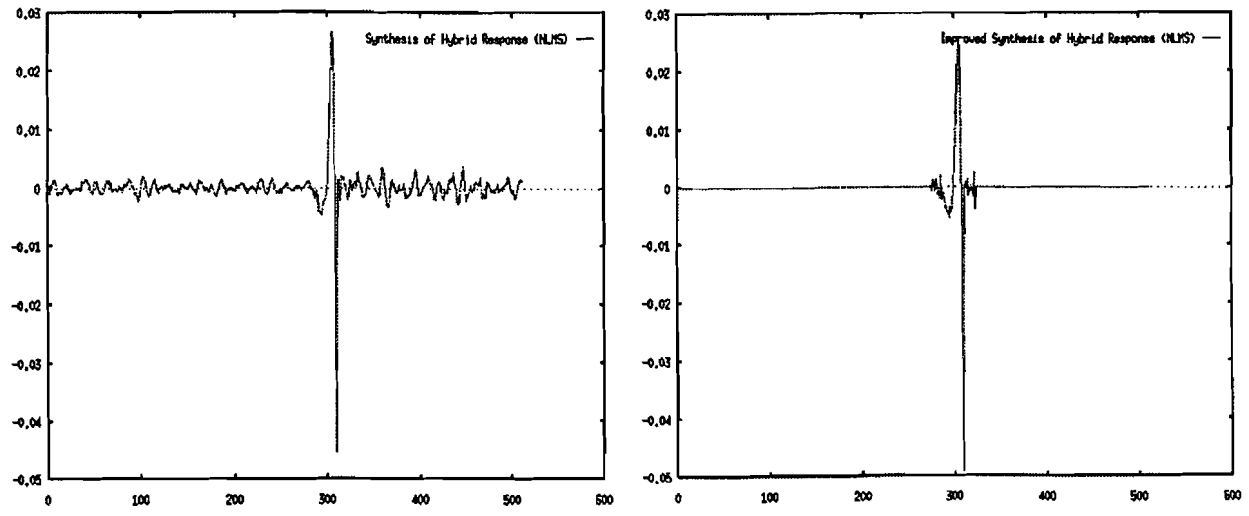


Figure 27 Synthesis of the Hybrid response without (left) and with optimisation (right)

The Figures above are simulation plots retrieved after 8000 samples. For the left Figure, no echo path optimisation is performed, for the right this is the case. For these simulation a hybrid impulse response length of 48 taps (equivalent to 6ms) was taken. The pure delay was set to be 200 samples, which can be found on the right side of the Figure (the impulse response is mirrored). It can be observed that the Hybrid response differs from either graph.

3.3.2 SILENCE-IMPROVED ECHO CANCELLER

Voice Activity Detection (VAD) has already been studied since the 1970s. In order to increase the utilisation of expensive satellite communication, during pauses in a conversation the channel is assigned to another user. This technique is used in Time Assignment Speech Interpolation (TASI) and Digital Speech Interpolation (DSI). Many researchers have studied the duration of pauses in speech and a speech activity range between 35-45% has been found. The differences can mainly be explained by different background noise thresholds. It is widely accepted that speech activity averages around 40%, including inter-syllable pauses. Silence may be determined using the next two parameters.

- short-term / long-term power thresholds
- number of zero-crossings

These parameters are all analysed in a window of arbitrary size.

Nakada and Sato [5] use the above parameters in their VAD algorithm. They found that the relative threshold between voiced and unvoiced speech is 30 dBr, and between unvoiced and silence 20 dBr. In the case the speech signal was 30 to 50 dBr lower with respect to voiced sounds, the decision between unvoiced and silence was made by the number of zero-crossings. If the number of zero-crossings was less than 30% out of 16 samples, the signal was decided to be unvoiced. Their simulations indicated a rough average of 50% talker activity. It must be said that be results were based on only three telephone conversations.

Yatsuzuka [6] developed a more complicated algorithm that was able to detect speech activity of 36%. To achieve this, additional parameters such as variation of short-time energy and Sign Bit sequence processing were used. The window size was set to 4ms and a hangover time of 30ms lowered the probability of going into silent state during short low energy periods of speech.

The use of VAD for Echo Cancellers is new. Generally, filtering occurs in 100% of the time. With a speech activity average of 40%, the filter is 'wasting its time' during the remaining 60%. During silence also the

adaptation of the filter-coefficients is no longer necessary. The Echo Canceller proposed by Elmalki [1] contains a silence detector that will inhibit adaptation and filtering during pauses of the far-end talker. It also provides for an outgoing signal for the bandwidth optimisation mentioned (TASI). The latter may also contain near-end pauses.

Figure 28a depicts the changes that where made compared to the classical structure.

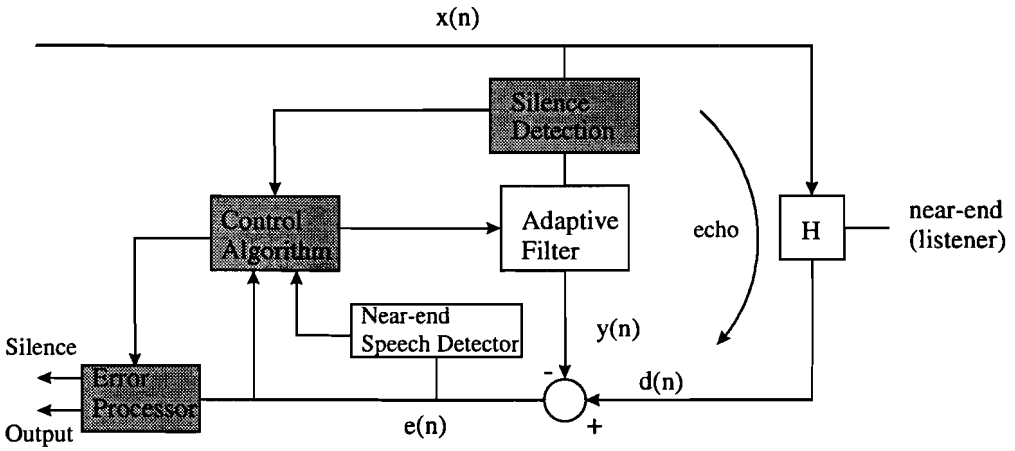


Figure 28a Modified Block Echo Canceller with Silence Improvement

Silence detection is performed by calculating the power of the input signal over a specific period. If during this time window the power-level is not bigger than a fixed pre-defined level, silence has been detected. Then, adaptation and filtering will be stopped. The moment a new incoming sample on x(n) exceeds the silence threshold, adaptation and filtering will immedeatly be resumed, even if the end of the silence window has not been reached.

The length of the silence window determines the silence detection efficiency. The size limits the minimum (inter-syllable) pause that can be detected. Inter-syllable pauses range from about a few tens of milliseconds to about 100ms. An example of a real speech waveforms is shown in Figure 28b. A silence threshold will be used which will depend on the maximum expected noise level. It should be high enough to distinguish noise from unvoiced speech, and, low enough to yield high efficiency.

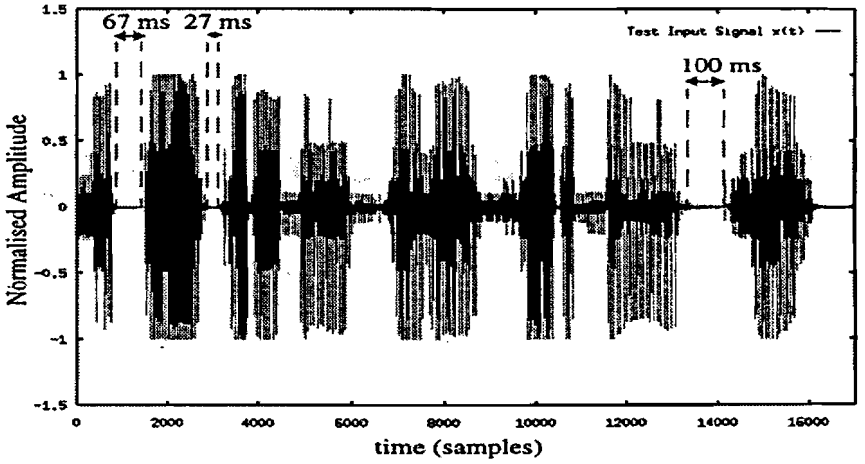


Figure 28b Inter-syllable pause measurements on example speech waveform

Depending on the characteristics of the total echo path, two modes of operation may be distinguished. The two modes are:

mode A: Echo path delay is shorter than the silence decision window
mode B: Echo path delay is longer than the silence decision window

In mode A the window is larger than the echo path delay. This is to allow the echo canceller to cancel relevant samples up until the echo path is "empty". Then, the silence-detection may become active. This mode becomes inactive the instant a sample power higher than the maximum noise level enters the silence window.

Mode B caters for the drawback of mode A in case of large echo paths, since for the latter a larger window than the echo path delay is needed. When silence is detected in mode B, the Silence Control Signal will become "TRUE" after the echo path pure delay. So, also for this mode the echo path will be "empty". From the moment a non-silent sample is detected the echo path delay minus an anticipation time later the Silence Control Signal will become "FALSE". The anticipation time is needed to compensate for possible silent *fricatives* at the beginning of syllables. This anticipation constant will be set to:

Silence-Window length / F.

Where F is a constant, which value has been experimentally determined to be ideal between 3 and 5.

From an implementation point of view Mode A has clear benefits. Mode B needs to have an in-built counter, after (non-)silence detection. Mode B provides for higher detection-efficiency, which is determined by the Silence Detection window and the anticipation time. For both modes knowledge of the echo path length is needed.

The Error Processor will provide a Silence signal for external equipment to improve bandwidth usage. The Silence flag may be set using two different methods. The easiest is to take the inverse of the Double-talk detection signal. The second is to use a Silence Detection window as has been explained before. During silence the Error Processor may replace the outgoing signal by noise, or function as an NLP.

3.4 SIMULATION RESULTS

Basically, the two modes and two adaptive Echo Canceller algorithms are simulated that will be selected automatically. The settings are shown in Table 29. All simulations are done with the assembly code using the Texas Instruments 'C54x device simulator.

Parameter	*TEST 1*	*TEST 2*
Number of filter taps (M taps)	512	
Silence Detection window (ms)	10	
Anticipation-time constant F	4	
Convergence time (samples)	3000	
Hybrid response length (taps)	48	
Algorithms	NLMS and FNLMS	
Length of Pure Delay (samples)	8	200

Table 29 Simulation parameters

The FNLMS is merely a filtered version of the NLMS in order to prevent an accumulation of arithmetic errors.

As input signal neither white or coloured noise, nor an artificial signal was used for the simulations. Instead, a recorded sentence was taken. With 5 dB additive noise it was said to have an equivalent SNR compared to conditions with mobile telephones. The speech signals were recorded using the internal audio device of a Sun Sparcstation, which compressed the signal according to μ-law. Two speech signals were recorded, one to represent the far-end talker and the other near-end talker, to create a double-talk situation. The normalised speech waveforms can be found in Figure 30.

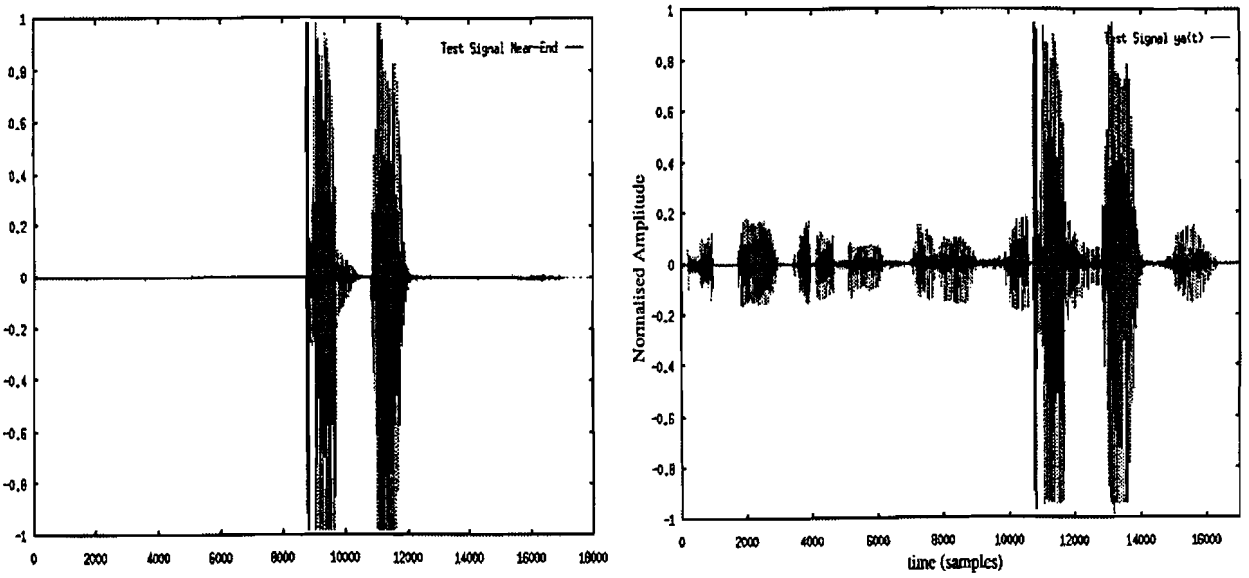


Figure 30 Waveforms for near-end double-talk (left) and the test signal d(n) (right)

The sentence was chosen for it was proven to be 'ill-conditioned' and did not contain pauses. No specific choices were made for the near-end speech, other than being distinct. In order to create the ya(n) test signal, x(n) was convolved with the hybrid impulse response (including the pure delay) and added to the near-end speech.

The results that were obtained from the various simulations are summarised in the Table 30. **Note**, all ERLE values that are, and will be mentioned do not take into account the fixed hybrid loss of 6 dB. In Figure 31, the formula is shown. In the formula $y_a(t)$ equals the desired response $d(n)$, as was used throughout this thesis.

Parameter	*TEST 1*		*TEST 2*	
Algorithm	NLMS	FNLMS	NLMS	FNLMS
Delay estimate (samples)	0	1	189	189
Double-talk (samples)	4047		4042	
Silence detection (%)	5,80		3,25	
Average ERLE overall (dB)	21,64	20,45	19,54	18,88
Average ERLE post-convergence (dB)	26,92	26,36	25,57	24,97

Table 30 Simulation results

The calculations of ERLE are performed when no double-talk situation occurred. Double-talk was detected almost equally in all tests. The tiny difference is due to the different delays, which creates a different superposition for either test. The number of samples is higher than the simulated Double-talk condition, which tells us that the 800 samples hangover-time is operating well.

Silence Detection with mode B seems more effective, but these results are preliminary. Actually, efficiency is determined more by the Silence detection window. The differences are caused by the shifted operation on the speech signal due to the different pure delay values. Because the test signal did not contain real pauses, the found Silence can be seen as inter-syllable pauses, which were not taken into account for the 40% average active speech.

The correct hybrid response was found in all cases. When looked at the estimated delay, the first tap has found to be a little too early. This "error" compensates for different hybrid responses. Since the synthesis of the hybrid response contains 40 samples, the value of 48 taps is just enough.

The differences between the average ERLEs of the two tests ought to be discarded. The test signal was far too small to get a stable result. I.e., no general conclusions can be drawn from this. The extra delay distorts the results, like explained before. What can be said is that the standard NLMS is likely to perform a bit better than the filtered version. It seems that the accumulation of errors does less harm than the extra filtering that is supposed to prevent this.

Finally, a comparative study was performed in order to gauge the performance increase introduced by the improved algorithm. The novel algorithm is tested using 2 different convergence times. The earlier the 'echo path optimisation' is initiated, the faster the convergence. In other words, the transient towards the final misadjustment is obtained in a faster way. The next plot shows this graphically. The classical NLMS algorithm is added to have a good reference. A faster convergence time does not only improve the quality for the user, since the perceived echo will fade away more speedy. From a network operator point of view, it will also create free processing power on the DSP to perform cancellation on another channel.

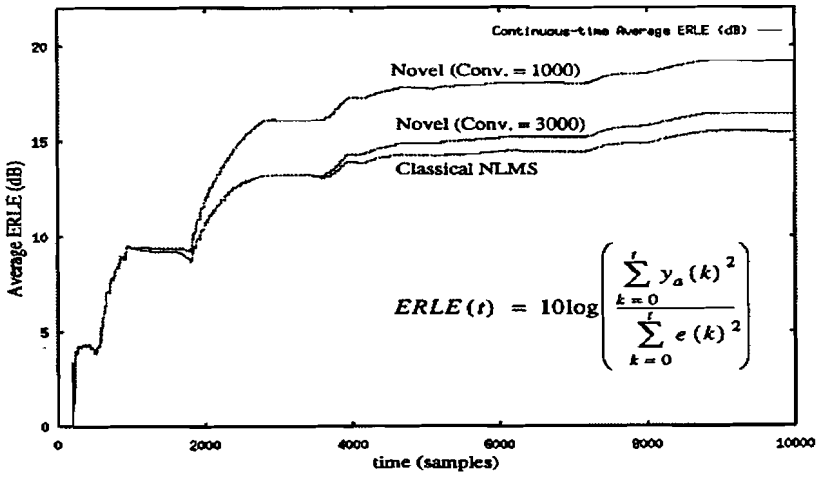


Figure 31 Comparative ERLE results

3.5 LABORATORY RESULTS

Before the laboratory testing could begin, a whole bundle of tests were needed, where all the constants had to be re-set to retain the same results as were obtained with the simulations. The real environment that was created in the laboratory caused the Echo Canceller to diverge and even to go into a complete crisis. To facilitate debugging, the initial tests were done without the novel improvements. Subsequently, additional tests were able to provide proper working of the total program, and to improve the performance.

The most important part of the testing in the lab was listening tests. Not many other means were present. No digital output was present to connect it to a PC or workstation and to calculate the ERLE. Nor means to record speech to be able to test all changes with the same speech sample. This made it more difficult to compare the different settings. To obtain more vivid proof of the capabilities of the algorithm a dual sine

wave was used to excite the Echo Canceller. The result was guided to a spectrum analyser and these results were printed. Two simulated transmission times were used, one far-end round-trip delay of 100 ms and a near-end round-trip delay of 4 and 25 ms. The value of 4 ms was chosen since this is a concrete value that is used for Echo Cancellers within DECT systems.

For various reasons mentioned earlier, test procedures according to what the ITU recommends (G.165 and G.168) could not be carried out. They were of great importance in the determination of which tests could be valuable. Taking the test objectives and means of testing into account, the various aspects that were examined are:

- The standard NLMS;
- The standard NLMS (with increase precision calculation of the filter tap updates);
- The Delayed NLMS;
- Working of the algorithm with and without NLP;
- Correct Double-talk detection;
- Different echo path pure delays (4ms and 25ms);
- Different convergence times (3000 and 1000 samples).

The FNLMS version was not tested in the laboratory since the differences between the standard NLMS was very small. The extra computations did not give sufficient benefits. In case long-term lab tests might have caused serious distorsion due to the accumulation of errors, the algorithm could have been implemented.

As has been stated before, after convergence far less clock cycles are required for filtering; this enlarged the possibilities to increase performance with more precise calculations. Due to savings in computational load the filter tap updates were calculated less accurate than possible. Implementing the changed calculation yielded an increased average ERLE of 5 dB.

The delayed NLMS uses older near-end signals to update the filter coefficients. This has the advantage in the sense that it may prevent divergence during double-talk. The newer near-end signals can first be analysed before they are used to update coefficients. Drawbacks are a slower convergence time and explicit bounds on the step size to prevent instability.

Since the tests were carried out with the near-end and far-end telephone in the same room, the direct and indirect acoustic coupling caused problems. Therefore, the handset was physically disconnected from the telephone for all tests except for the one to test the double-talk detector. Other principal results are:

- The hangover time had to be enlarged to cause less transient distortion;
- The multiplication factor 'K' for the echo path optimisation was increased to 8 in order to not mistake with noisy spikes;
- Double-talk detector threshold had to be lowered considerably to not cause serious divergence (additional noise and returned echo were considerably lower than for the simulations);
- A convergence time of 1000 samples worked well even in a practical situation;
- With a hybrid loss that was 9 dB higher than the one used for simulations, the algorithm performed well;
- The NLP did not work very well, probably due to incorrect threshold settings;
- The NLP was no longer needed with the last update of the algorithm, only with loud speech the echo became noticeable. This was the opinion of all testers involved;
- Apparently, the Delayed NLMS did not create sufficient benefits;
- The true echo path was hardly ever found when a sinusoid was applied as input signal (Test 7, G.168);
- A 30-minute test period did not cause the algorithm to diverge.

Some retrieved results can be found in the Table 33. The tested algorithm was the standard NLMS with a more precise calculation of the filter tap updates. Two tones were applied as far-end signals with frequencies of 770Hz and 1336 Hz. Four different tests were carried out, two different near-end delays and two different

impulse response lengths. Snapshots of the results can be found in Appendix B. The appendix also contains the original excitation of the Hybrid as well as the returned electric echo.

Test	*Silence Mode*	*Near-end delay (ms)*	*Impulse Response length*	*Calculated delay*	*D(n) (dBVrms) 700/1336 Hz*	*E(n) (dBVrms) 700/1336 Hz*	*average ERLE dB*
1	B	25 ms	48 taps	25 ms	38,68/39,57	67,67/67,69	28.6
2	B	25 ms	64 taps	25 ms	38,68/39,57	69,92/69,55	30.6
3	A	4 ms	64 taps	4.125 ms	38,68/39,57	68,93/69,01	29.9
4	A	4 ms	48 taps	4.125 ms	38,68/39,57	67,45/67,46	28.8

Table 33 ***Laboratory Convergence Test Results***

Note to the results: In order to obtain more stable (thus more real) results, the values were averaged over 20 time frames of 500ms. It ought to be said that the average ERLE should only be seen as indicative. The time-variations and measurement inaccuracies do not really allow an honest comparison. In future test phases, more advanced performance extraction tools within the Firmware are foreseen.

CHAPTER 4

CONCLUSIONS AND RECOMMENDATIONS TO THE IMPROVED ECHO CANCELLER

4.1 CONCLUSIONS

The proposed improvements for echo cancellers, implemented to reduce electric echo, have shown great improvements on computational load and performance. The 'echo path optimisation' brought the active number of filter taps from 512 to 48, which saves 91% on additions and multiplications. The savings in taps also yields a lower burden on the memory, and an increased ERLE by several decibels. The proposed method has clear advantages over existing ones, since a full search of all taps is not required, nor intermediate storage.

Normally, filtering occurs 100% of the cycles. With an average voice activity of 40%, "Silence Detection" can save 60% of the redundant calculations. The same percentage holds for adaptation, but this block is only active when no double-talk situation occurs. The VAD and 'echo path optimisation' enables the DSP to cancel echo from multiple channels, which reduces the hardware costs of Group Switches.

Simulations confirmed the theoretical gain of a reduced computational load by 96%. The echo path optimisation always found the correct hybrid response "window", although about 10 taps too early. The latter is required in order to accommodate for different hybrid patterns that exist. The small Silence detection window increased the efficiency by a percentage between about 3 and 6 percent, this was dependent on the mode that was set. The latter increase was due to inter-syllable detection, which was not taken into account in the 40% average active speech. No problems were encountered with the double-talk detector. The hangover-time appeared to be set correct.

Laboratory results mainly confirmed the simulated results. With an increased multiplication factor K the echo path pure delay was always found. Like with simulations, a bit too early but within the hybrid response "window". The test often failed when a sine was applied as input signal instead of speech. An additional test with a convergence time of 1000 also worked. This is important from a perceptual point of view, because the perceived echo fades rapidly, and also from a hardware point of view, since the DSP may then start with another channel. "Silence Detection" could not be checked.

The following values yield a TELR of 46 dB, which is enough for a mean one-way transmission time up to 80 ms (for this, see Figure 16). With an extra inserted loss of 6 dB the figure increases to about 170 ms. The values correspond to the results in the lab, where a round-trip delay of 125 ms was used and good results were achieved.

ERLE hybrid	11 dB	(value used by more researchers and advised by ITU-T)
ERLE Echo Canceller	25 dB	(lower value of average ERLE after convergence, without NLP)
Extra network loss	10 dB	(typical loss value across the twisted-pair and the hybrid)
Inserted loss (Loss Plan)	0 dB	

The simulation results obtained by simulations are a "best case"-situation since the hybrid loss was chosen to be 6 dB. It is commonly known that Echo Cancellers work well with high returned echoes. Despite this fact the laboratory results proofed outstanding performance. The increased ERLEs obtained in the laboratory, compared to the simulations, may be due to:

- inaccurate measurements;
- the longer test time compared to the simulations;
- low near-end background noise, since the handset was disconnected from the telephone.

Despite the great achievements with echo path optimisation it may be dangerous to use it in the proposed way. There is a real change that the echo path will not be determined correctly. In order to miss the noisy spikes the multiplication factor K was chosen to be fairly high. This may cause to miss the relative peak and hit the global maximum. The ERLE will therefore be degraded. Secondly, the national network may reroute the call and cause a different echo path pure delay.

The effect may be seriously worsened because of the fixed convergence time of either 1000 or 3000. It is not said that the echo canceller has already obtained a reasonable ERLE. Convergence starts the moment a threshold on x(n) has been exceeded. This may be the case when the handset is picked up. Even if the far-end speaker remains mute, the counter will continue. With voice-band data and, even worse, DTMF tones convergence will be reached much later. The erroneous behaviour of the implemented algorithm with tones was observed in the laboratory test phase. Also, in case the DSP with the Echo Canceller was started before the two fifo's (delays), the first mentioned DSP totally blocked. The 'remedy' was to execute the DSPs with fifo's first.

4.2 RECOMMENDATIONS

The recommendations in this section may improve the performance of the implemented algorithm. No time was available for the implementation and testing. This will be done by Ericsson in a later stage.

In order to obtain a more clear knowledge of the time when the Echo Canceller has converged, either of the following changes can be made:

1. Counting a fixed number of samples with sufficient amplitude (not just any)
2. Analysing the updating of the coefficients (around the optimum the sign of $\mu \bullet e(n) \bullet x(n)$ changes)

The second may create a fairly large computational burden, for all 512 values ought to be looked at. The first may be a simple alternative. The second option may also be used to determine if the call has been rerouted. If the filter taps are updated rapidly it could be an indication that a different echo path pure delay is the case. It is clearly the best to investigate a single filter tap, being the biggest in amplitude. It takes only a few clock cycles to find the global maximum. This procedure also clarifies if the global maximum lies within the hybrid "window" that has been found.

A further reduction in computational load can be obtained by completely stopping the adaptation procedure. Since the hybrid response is assumed to be (almost) time-invariant this may be easily done. It will also prevent the Echo Canceller to diverge under undetected double-talk conditions or low-level near-end background noise. Periodically, the adaptation procedure could be re-started. Alternatively, the earlier mentioned update of largest filter tap coefficient could be observed.

Already in 1961 the "Computer of Average Transients" (CAT) was introduced in the field of Electro EncephaloGram (EEG)-research. By means of a summation of repeating short signals and afterwards dividing by the total number of summations, the obtained SNR was increased by a number of $\sqrt{N}$, where N is the number of additions. It might well be possible to do the same with the synthesised hybrid response. The noise on the filter taps is zero-mean and can therefore be reduced. This scheme is merely possible if the exact location of the synthesised response is known, otherwise far too much memory would be required. Another issue is the increased *memory* of the adaptive filter. The algorithm will react slowly to a varying environment. It should be possible to find a way to diminish this drawback.

Silence detection could be improved by cancelling only voiced sounds. In the proposed Silence detector, the detection level is placed between noise and unvoiced sounds. It was stated that the unvoiced sounds are relatively 30 dB lower than voiced sounds. When looked at the TELR, only voiced sounds cause an audible echo. This method would further diminish calculation cycles, but causes non-linearities in the transmission path.

Without the use of an NLP, perceived echoes, occurring during the transient before convergence, could be lowered by using a soft suppressor with a time-dependent loss between say 10 dB an 0 dB. The decay should be about equal to the convergence time. This way convergence becomes almost not noticeable to the far-end speaker.

Even if the algorithm is able in perfectly model the hybrid response, H(z), an error may occur due to the companding procedure. The estimated echo does not pass a compander, whereas the true echo is compressed and then expanded. It is easy to verify with simulations whether a "companded" estimation of the echo yields an improvement.

CHAPTER 5

THE ADPCM CODEC

The era of information (technology) has started some years ago and this will remain so for a considerable time. This rising trend is responsible for an increased transmission rate as well as a high storage capacity. Since both are costly, a reduction of data is very desirable. This may prevent overloading transmission channels and storage resources, which cause a abatement in service. Of the aforementioned data, speech still plays a major role in telecommunications. For the data reduction Speech Compression may be utilised. Speech compression was first practised by the military industry in order to allow additional encryption to secure the transmission. Later, network operators applied the method to increase revenue.

The ADPCM codec that will be described in this chapter has been studied for many years. Hundreds of different versions exist, all with different features. This type of codec has gained substantial importance. This is mainly due to the fact that only a minor decrease in speech quality can be observed, comparing the 32 kbit/s ADPCM to standard 64 kbit/s log-PCM. But, it is also oftem used because of the relative ease of implementation. One of the versions was approved in 1984 by the ITU-T, then called CCITT, and standardised in Recommendation [G.721]. This initial version was revised in 1990 and substituted by Recommendation [G.726].

This chapter is subdivided in the following way:

5.1 Introduction to speech compression
5.2 General explanation of ADPCM
5.3 The ITU standard of ADPCM
5.4 Implementation and testing of ADPCM
5.5 Test results

This chapter starts with an introduction to speech compression. Section 5.1 provides for general information on the existing schemes to compress speech data and gives a more detailed explanation of speech prediction. Speech prediction is an important technique also used in ADPCM. The second section mentions general issues of ADPCM, whereas the third explains the implemented ITU standard of ADPCM specifically. Section 5.4 reveals the approach that was chosen for the development of the low-level code of ADPCM. The final section illustrates how the developed programs were tested and shows the test results.

5.1 INTRODUCTION TO SPEECH COMPRESSION

The information that is communicated through speech is intrinsically of a discrete nature; i.e., it can be represented by a concatenation of elements from a finite set of symbols. The symbols from which every sound can be classified are called *phonemes*. Each language has its own distinct set of phonemes, typically numbering between 30 and 50. For example, English can be represented by a set of around 42 phonemes.

For speech a crude estimate of the information rate can be obtained by noting that physical limitations on the rate of motion of the articulators require that humans produce speech at an average rate of about 10 phonemes per second. Using a 6-bit binary code to represent the various English phonemes yields an

estimate of 60 bit/sec for the average information rate of speech. Of course the true lower bound to represent the actual speech signal is considerably higher than this rate. The above estimate does not take into account factors such as the identity and emotional state of the speaker, the rate of speaking, the loudness of the speech, etc. In general, *what* is said and *who* said it has to be preserved.

There are three distinct families of speech compression techniques. The first and only one that uses a non-parametric representation, observes the statistical characterisation of the speech waveforms. The general name is *Waveform Coders*. As the name implies, these coders essentially strive for facsimile reproduction of the waveform. In principal they are designed to be signal-independent, hence they can code equally well a variety of signals (speech, music, tones, voice-band data). The coded speech generally provides toll quality, being comparable to that of an analogue signal having approximately the properties: Frequency range 200-3200 Hz; SNR $\geq$ 30 dB; Harmonic distortion $\leq$ 2-3%. To preserve these advantages with minimal complexity, waveform coders typically aim for moderate economies in transmission rate. Examples are (Adaptive) Delta Modulation and (Adaptive) Differential PCM.

The second class of speech coders is termed *Source Coders,* generally referred to as *Vocoders* (a contraction of the words voice coders). It exploits a priori knowledge about how the speech is produced. A model of the vocal apparatus is used. The goal of the existing algorithms is to obtain the various parameters with which the speech signal can be synthesised. Large bandwidth reductions may be achieved. The speech quality will be below *toll quality,* and is often talker-dependent. The algorithms require a fairly high amount of computations. In telecommunication the additional encoding delay may be a problem if the TELR is below the recommended value (Chapter 3). An example is Linear Prediction Coding (LPC) and its many derivatives.

The third and last kind is *(Adaptive) Transformation Coding.* The greatest advantage of this coding class is that it operates in the frequency domain. This technique involves block transformation of windowed input segments of the speech waveform. Each segment is represented by a set of transform coefficients, which are quantised and transmitted. At the receiver the inverse action will take place. Transform examples are the suboptimal , DWHT, DHT, DFT, DCT and the optimally orthogonalising Karhunen-Lòeve Transform (KLT). The latter merely provides an upper bound for the other time-to-frequency transforms, it will not be used due to the computational complexity. The DCT has been found to be well suited for speech coding. A well-known speech coder that, in a sense, uses the frequency domain is Sub-Band Coding (SBC). With this type, the speech signal is subdivided into a number of separate frequency components and each are encoded separately. In both types the speech quality may be substantially increased by dynamically allocating a number of bits to the transmitted frequency components. Thus, the encoding accuracy may be placed where it is needed. Low bit rates (even below 1 bit/sample) may be achieved, attaining a high subjective quality. All at the cost of the highest computational complexity of the three speech compression families.

In order to apply digital signal processing techniques to speech communication problems, it is essential to understand the fundamental issues of the speech production process. Figure 41a shows a schematic diagram of the vocal apparatus. The sub-glottal system, lungs, bronchi and trachea, serves as a source of energy for the production of speech. Speech sounds can be classified into three classes according to their mode of excitation. *Voiced* sounds are produced by forcing air through the glottis with the tension of the vocal cords adjusted so that they vibrate in a relaxation oscillation, thereby producing quasi periodic pulses of air which excite the vocal tract. *Fricative or unvoiced* sounds are generated by forming a constriction at some point in the vocal tract. This creates a broad-spectrum noise source to excite the vocal tract, with relatively low energy. *Plosive* sounds result from making a complete closure, building up pressure behind the closure, and abruptly releasing it.

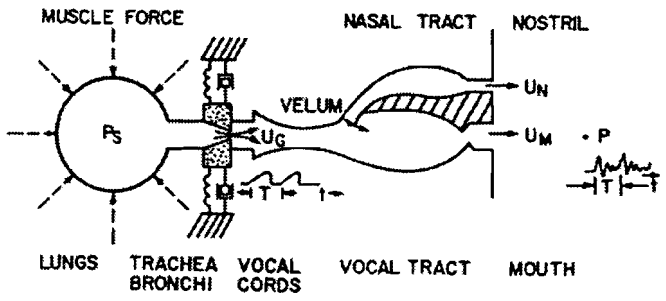


Figure 41a Schematised diagram of the vocal apparatus [16]

A general discrete-time model for speech production can be found in Figure 41b. As can be seen, this model comes close to the real situation depicted in the above schematic. Three different effects may be observed: excitation, vocal tract and radiation. The basic assumption is that the excitation and the vocal tract may be decoupled. The voiced/unvoiced switch determines if the source should produce either a quasi-periodic pulse waveform or a random noise waveform. The choice may be made according to the zero-crossings rate and/or energy during a certain period of time. Voiced sounds typically have a lower number of zero-crossings during an interval *N* and a much higher energy than unvoiced sounds. The pitch period sets the fundamental period of the impulse train. The pitch frequency is typical to all individual human beings and depends on the speed at which the vocal cords vibrate. Typically ranging between 50-200Hz for men and 100-400Hz for women. Determining the correct pitch period is both troublesome and important. It can be done by means of, for instance, the short-time autocorrelation function (ACF), where the successive peaks in the ACF represents this period. An algorithm that uses the ACF for pitch detection is called SIFT (simple inverse filtering tracking). It also delivers a signal whether the speech sounds are either voiced or unvoiced

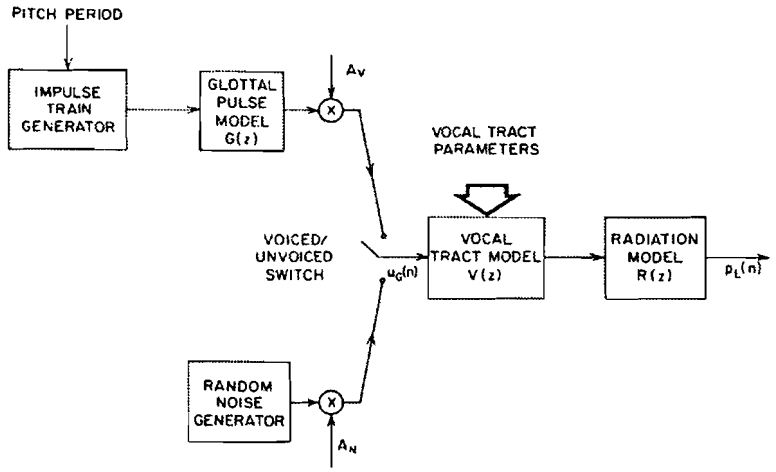


Figure 41b General discrete-time model for speech production

The resonances (formants) of speech correspond to the poles of the vocal tract transfer function. An all-pole model is a very good representation for the majority of speech sounds; however, the acoustic theory tells us that nasals and fricatives require both resonances and anti-resonances (poles and zeroes). We may either include zeroes in the transfer function or reason with Atal [14] that the effect of a zero of the transfer function can be achieved by including more poles. The transfer function of the vocal tract is given by:

$$V(z) = \frac{G}{1 - \sum_{k=1}^{p} \alpha_k z^{-k}} \qquad \textbf{[42a]}$$

where G and $\{\alpha_k\}$ depend upon the area function (which is a function of the length of the vocal tract) of the lossless tube that models the vocal tract. The parameter *p* characterises this lossless tube by a set of areas or, equivalently, reflection coefficients. This figure depends upon the sampling rate. The transfer function V(z) relates air velocity at the source to air velocity at the lips. If we wish to obtain a model for pressure at the lips, then the effects of radiation must be included. In the case of linear predictive analysis it is convenient to combine the glottal pulse, radiation and vocal tract components all together and represent them as a single transfer function:

$$H(z) = G(z)\, V(z)\, R(z) \qquad \textbf{[42b]}$$

of the all-pole type.

The basic idea behind linear predictive analysis is that a speech sample can be approximated as a linear combination of past speech samples. By minimising the sum of the squared differences (over a finite interval) between the actual speech samples and the linearly predicted ones, a unique set of predictor coefficients can be determined. The philosophy is intimately related to the fact that speech can be modelled as the output of a linear time-varying system excited by quasi-periodic pulses or random noise. The linear prediction method provides a robust, reliable and accurate method for estimating the parameters that characterise the linear, time-varying system.

The basic problem of linear prediction is to determine a set of predictor coefficients directly from the speech signal in such a manner as to obtain a good estimate of the spectral properties of the speech signal. The general approach leading to useful results is to minimise the mean-squared prediction error (mmse). The cost function that has to minimise the short-time average prediction error is defined as:

$$J_{\min}(n) = E_n(n) = \sum_m (s_n(m) - \check{s}_n(m))^2 \qquad \textbf{[42c]}$$

where *s(n)* denotes the actual speech samples. The predicted speech samples, *š(n)*, is defined as

$$\check{s}(n) = \sum_{k=1}^{p} \alpha_k s(n-k) \qquad \textbf{[42d]}$$

We can find the values of $\partial E_n / \alpha_k$ that minimise E_n by setting $\alpha_i = 0$, $i = 1,2,...,p$. There are a number of different approaches to obtain the solutions of these equations. The Least Mean Square family explained in Chapter 3 offers the simplest solution. A by-product of the LPC analysis is the generation of the prediction error signal. From this signal the pitch period may be retrieved. For many voiced sounds the prediction error will be large at the beginning of each pitch period. Unfortunately it has its own difficulties in locating pitch markers for a wide variety of voiced sounds. The SIFT algorithm mentioned earlier provides better results.

Finally, the *spectral flatness measure* (sfm) is defined as:

$$\gamma_x^2 = \frac{\exp\left[\frac{1}{2\pi}\int_{-\pi}^{\pi} \ln\left(S_{xx}(e^{j\omega})\right)d\omega\right]}{\frac{1}{2\pi}\int_{-\pi}^{\pi} S_{xx}(e^{j\omega})d\omega} = \frac{\eta_x^2}{\sigma_x^2} \qquad \textbf{[43a]}$$

where $S_{xx}(e^{j\omega})$ is the power spectrum density (psd) of the input signal x(n), σ_x^2 denotes the variance, and γ_x^2 can be seen as a minimum prediction error variance [12]. Note that $\gamma_x^2 = 1$ if $S_{xx}(e^{j\omega}) = \sigma_x^2$, which is the psd that describes a white noise process. The sfm is used as an important tool to describe the shape of a psd-function by a single value. The sfm bounds the performance of two classes of coding schemes, classes that exploit waveform redundancy (as shown by a non-flat psd) by *predictive coding* and by *transform coding*. The inverse of the sfm is a measure of waveform predictability. This number is typically 3 for the long-term psd's of speech. Short-term psd's are sometimes characterised by much higher values of γ_x^{-2}, implying higher predictability. The sfm ranges between zero and unity, leading from processes that are predictable with zero error to unpredictable white noise processes, respectively.

5.2 GENERAL EXPLANATION OF ADPCM

The amplitude of a speech signal can vary over a wide range depending on the speaker, communication environment, and within a given utterance, from voiced to unvoiced segments. One attempt to minimise the variance of the speech signal is the use of Differential PCM (DPCM). The difference between adjacent samples has a lower variance than the variance of the coder input itself. The next Figure shows the adjacent sample correlation. The Figure illustrates a long-time-averaged autocorrelation function (acf) based on measurements with a 55-second speech sample that is either lowpass-filtered (LPF: 0 to 3400 Hz) or bandpass-filtered (BPF: 300 to 3400 Hz). The sampling frequency is 8 kHz. In each set of curves, the upper and lower limits represent acf maxima and minima, respectively, over four speakers (two male and two female).

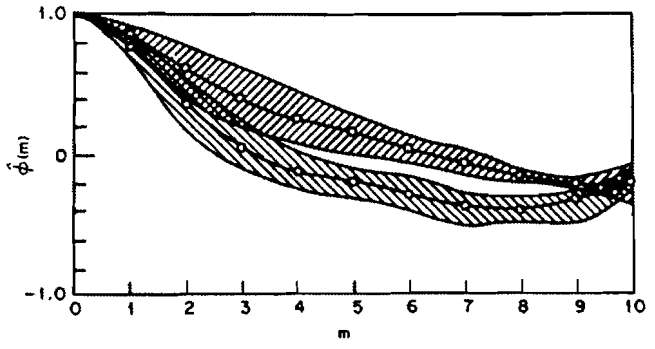


Figure 43 Autocorrelation functions of speech signals; upper curves for lowpass-filtered speech, lower curves for bandpass-filtered speech. (After Noll [17])

The correlation is high (~0.9) between adjacent samples and it decreases rapidly for greater spacing. Also evident is the fact that lowpass-filtered speech is more highly correlated than bandpass-filtered speech. Thus, lower frequencies in speech causes higher correlations for a given separation *m*. In general, the quantiser input in a DPCM coder is a prediction error or difference signal

$$d(n) = x(n) - \hat{x}(n) \qquad \textbf{[43b]}$$

where $\hat{x}$ (n) is a prediction of x(n). Note, the high sample-to-sample correlation supports the assumption that speech may be predicted from a number of past speech samples. It is easy to verify that if $\hat{x}$ (n) = x(n-1), this would only lead to a positive gain if ρ_1>0.5. The predictor tries to minimise the variance of the difference signal, yielding and increased SNR. This Signal-to-quantising Noise Ratio of the DPCM system can be written as

$$SNR = \frac{E[x^2(n)]}{E[e^2(n)]} = \frac{\sigma_x^2}{\sigma_d^2} \bullet \frac{\sigma_d^2}{\sigma_e^2} = G_P \bullet SNR_Q \qquad \textbf{[44a]}$$

where σ_x^2, σ_d^2, and σ_e^2 denote the variances of the input signal, the difference signal and the quantisation error, respectively. The quantity SNR_Q is dependent upon the particular quantiser that is used, and, given knowledge of the properties of d(n). The quantity G_P, if greater than unity, represents the gain in SNR that is due to the differential scheme. Clearly, the objective is to maximise G_P by appropriate choice of the predictor system. This leads us back to the upper bound provided by the reciprocal of the sfm

$$\max_{p\to\infty}({}^{p}G_P) = \frac{\sigma_x^2}{\min({}^{\infty}\sigma_d^2)} = (\gamma_x^2)^{-1} \qquad \textbf{[44b]}$$

where p is the number of predictor coefficients used, in other words: the prediction order. There are two basic possibilities for the prediction system. The first is the use of fixed predictor coefficients. The second is the use of adaptive predictor coefficients. Noll [12] has used the data used for Figure 43 to compute the maximum obtainable prediction gain $(G_P)_{opt}$ as a function of the number of fixed prediction coefficients, *p,* for a 55 second segment of speech that was bandpass-filtered. The results are depicted in Figure 44. The shaded region shows the amount of variation obtained over four speakers. It is clear that, even with the simplest predictor, it is possible to realise about a 6dB improvement in SNR.

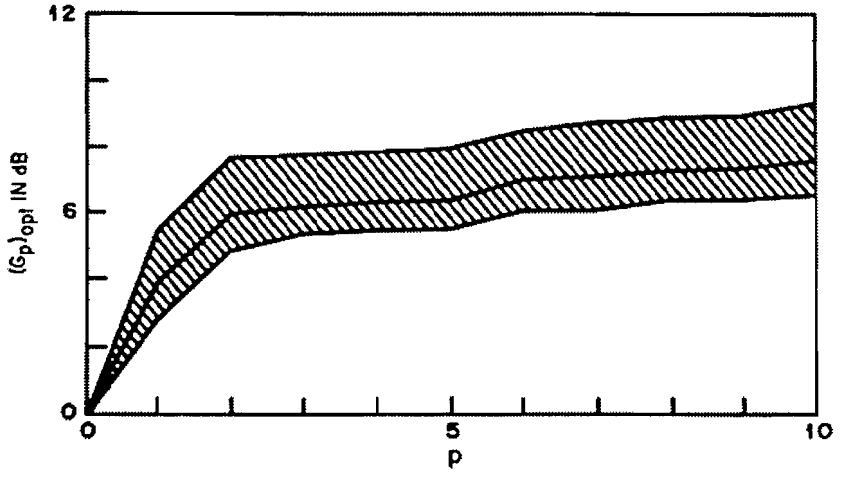


***Figure 44 Optimum SNR gain G versus number of predictor coefficients; bandpass-filtered speech. (After Noll* [15])**

The shaded region in the above Figure shows the amount of variation obtained for the four speakers. It displays a wide variation of prediction gain. Similar variations are observed among different speech utterances. All of these effects are a result, off course, of the non-stationarity of the speech signal. No fixed set of predictor coefficients can be optimum for a wide variety of speech material or a wide range of speakers. Short-term psd's often provide a much higher number for the reciprocal of the sfm, thus higher waveform predictability, as was mentioned before. For this reason often the prediction is made adaptive at the cost of increased complexity of the coder. Figure 45 shows the difference between adaptive and fixed prediction for a single (female) speaker. The speech data was sampled at 8 kHz. The speech was lowpass-filtered which explains the lower SNR gain for fixed prediction compared to Figure 44.

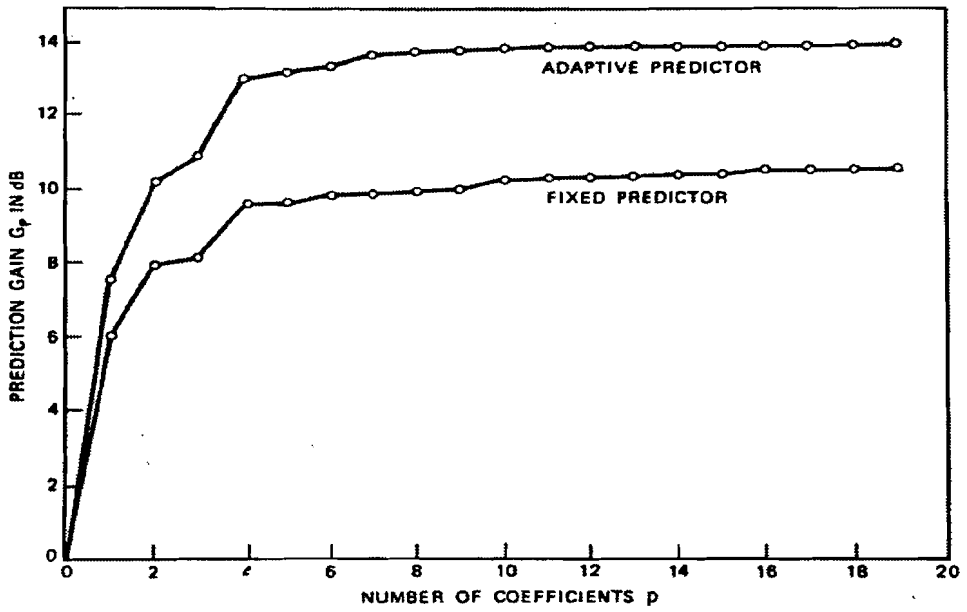


Figure 45 Prediction gain versus number of predictor coefficients for one female speaker (lowpass-filtered speech) After Noll **[15])**

Figure 45 expresses the qualitative benefits of adaptive prediction. Regarding fixed prediction, the maximum gain is about 10.5 dB, for adaptive prediction close to 14 dB. A prediction order higher than 4 or 5 leads to very little extra gain. Not evident in the curves of Figure 45 is the fact that the optimum fixed predictor is likely to be very sensitive to speaker and speech material. Whereas the adaptive prediction scheme is inherently much less sensitive.

As was remarked before, the choice of the quantisation of the difference signal may yield an additional SNR gain. The ideal quantiser ought to limit the quantisation noise and idle channel noise, and has to avoid exceeding the amplitude range. The quantisation may be either uniform or non-uniform, adaptive or non-adaptive. With an adaptive quantiser the step size may be varied according to changes in the input signal. It is also possible to use a time varying gain that tends to keep the variance constant. The adaptation may take place on a sample-to-sample basis, referred to as *instantaneous* changes, or remain essentially unchanged for relatively long time intervals, referred to as *syllabic* variations. The step size (or gain for that matter) may be derived from the input signal or from the quantised output. The first is named feed-forward adaptation (AQF), the second feed-backward adaptation (AQB). For the latter there is no need for transmitting gain or step size information to the decoder. The disadvantage of such systems is increased sensitivity in the code words, since these errors imply not only an error in the quantiser level but also in the step size. The advantage is that, with the same bit rate, more quantiser levels will be available.

Adapting the step size of the quantiser may be done in the way shown in Formula 45a.

$$\Delta(n) = \Delta_0 \sigma(n) \qquad \textbf{[45a]}$$

where $\Delta(n)$ is the step size at time n, Δ_0 is a constant and $\sigma(n)$ denotes the standard deviation of the input signal. Step size adaptation may also be achieved by scaling the input with a suitable gain, like $y(n) = x(n)G(n)$, where $G(n)$ could be calculated as follows

$$G(n) = \frac{G_0}{\sigma(n)} \qquad \textbf{[45b]}$$

where G_0 is a constant. The window size in which the standard deviation is estimated, determines if we have to do with *syllabic* or *instantaneous* adaptation.

Jayant [18] extensively studied the AQB-system that updates the step size as shown in Formula 46. The step size multiplier, P, is a function only of the magnitude of the previous code word. The procedure is generally referred to as *adaptive quantisation with a one-word memory.*

$$\Delta(n) = P\Delta(n-1) \qquad [46]$$

Rabiner and Schafer [13] report an increase in SNR of several dB's, up to about 6dB. Adding this figure to the 14 dB obtained from the adaptive prediction scheme in Figure 45 yields a total number of 20 dB. This objective figure tells us that an ADPCM coder could work well with about 3-4 bits less than log-PCM. Subjective tests show this is indeed the case [23]. Besides, transmission errors have a more serious impact on the perception of speech with log-PCM than with 4-bit ADPCM.

5.3 THE ITU STANDARD OF ADPCM

After the standardisation of ADPCM in 1984 the use of this algorithm has increased dramatically. In the USA it is used by the network operators to compress speech data on the STM T1 links between the city of New York and San Fransisco, and, New York and Chicago. This led to savings of about $600.000 over a 3 year period. Also the DECT air protocol uses the standard because of its outstanding speech quality.

In this paragraph the latest approved standard (G.726) will be explained and reference will be made to the minor changes compared the older standard (G.721). The main difference between the standards is that the last version is able to operate at four different bit rates. It uses an identical "core" for these bit rates, only some constants vary. The bit rates are: 16, 24, 32 and 40 kbit/s. The G.721 merely uses the 32 kbit/s bit rate, which is used within DECT.

The principal application of 16 and 24 kbit/s channels is for overload channels carrying voice in Digital Circuit Multiplication Equipment (DCME). The principal application of 40 kbit/s channels is to carry data modem signals in DCME and Packet Circuit Multiplication Equipment (PCME), especially for modems operating at bit rates greater than 4800 bit/s. The major use for 32 kbit/s ADPCM is the substitution of log-PCM within the PSTN or private telephone networks.

A simplified block diagram of ADPCM is shown in Figure 46. Note that the decoder logic is also used in the encoder. Then, assuming an errorless transmission condition, the same information in both Encoder and Decoder is available.

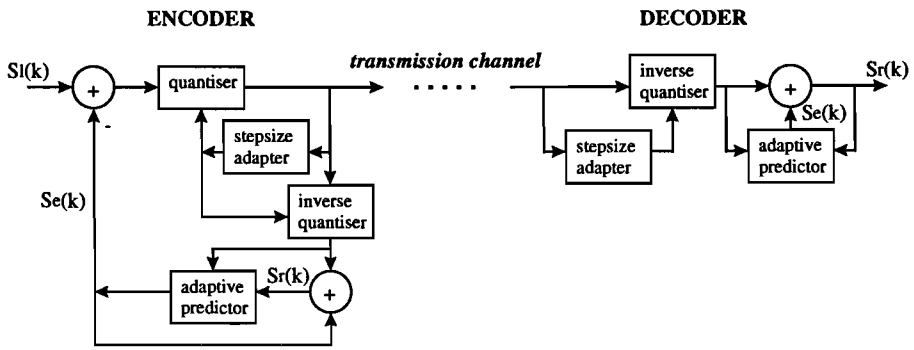


Figure 46 Simplified block diagram of ADPCM

ENCODER

Figure 47 shows the diagram of the encoder according to the ITU-T Recommendation. The Encoder will be described extensively, whereas with the Decoder only the unmentioned items will be touched upon. Only 32 kbit/s ADPCM will be depicted, but the differences with the other bit rates will be specified.

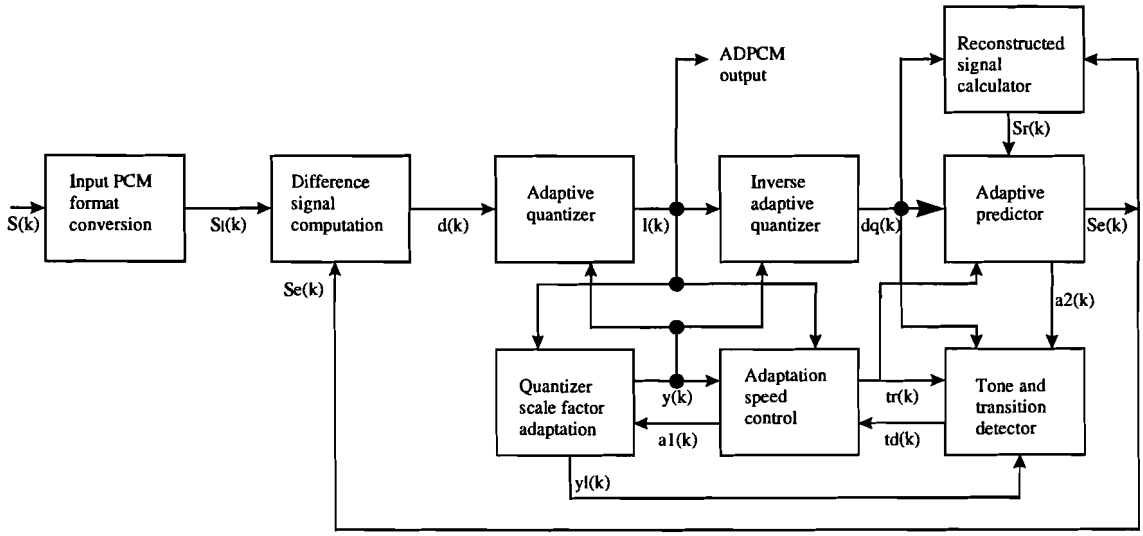


Figure 47 Block diagram of the ADPCM encoder

The input of the ADPCM encoder is 64 kbit/s log-PCM. The first block retrieves the samples and converts them to linear signals. From this signal, $S_l(k)$, the signal estimate $S_e(k)$ is subtracted, yielding the difference signal d(k).

$$d(k) = S_l(k) - S_e(k) \qquad \textbf{[47a]}$$

where k represents the sample moments
(current sample → k = 0)

ADAPTIVE QUANTISATION

This part is comprised of the 'Adaptive quantiser', 'Quantiser scale factor adaptation' and the 'Adaptation speed control' blocks in Figure 47. This multistage process is used to determine the quantisation scale factor and the speed control that controls the rate at which the scale factor is adapted. The output of the encoder is determined by take the log (base 2) of the differences signal, d(k), and subtracting the quantiser scale factor as shown below.

$$Log_2 |d(k)| - y(k) \rightarrow |I(k)| \qquad \textbf{[47b]}$$

The logarithmic domain offers an advantage because multiplications and divisions are substituted by additions and subtractions, respectively. The result will directly provide a value for the output. These can be found in Table 48, where the sign of the difference signal is equal to that of the output value.

Normalised quantiser input range $\log_2 \lvert d(k) \rvert - y(k)$	$\lvert I(k) \rvert$	Normalised quantiser output $\log_2 \lvert d_q(k) \rvert - y(k)$	Scale factor multiplier W[I]	Rate-of-change weighting function F[I]
[3.12, + ∞)	7	3.32	70.13	7
[2.72, 3.12)	6	2.91	22.19	3
[2.34, 2.72)	5	2.52	12.38	1
[1.91, 2.34)	4	2.13	7.00	1
[1.38, 1.91)	3	1.66	4.00	1
[0.62, 1.38)	2	1.05	2.56	0
[-0.98, 0.62)	1	0.031	1.13	0
(- ∞ , -0.98)	0	- ∞	-0.75	0

Table 48 Constants determination from the normalised difference signal

Notes: *Except for the second and fifth column the values have been adjusted with respect to [G.721]. I(k) value +0 does not exist, therefore only 31 different values are transmitted.*

The quantiser scale factor is comprised of two parts, and therefore bimodal in nature. The two parts, $y_l(k)$ and $y_u(k)$, are weighted by the speed control factor, $a_l(k)$, see Formula 48a. For speech signals, $a_l(k)$ will tend towards a value of one; for voiceband data, $a_l(k)$ will tend towards zero. The dual structure effectively caters for the variety of input signals that might be encountered.

$$y(k) = a_l(k) \bullet y_u(k-1) + [1-a_l(k)] \bullet y_l(k-1) \qquad \textbf{[48a]}$$

where $0 \leq a_l(k) \leq 1$

One of the factors, $y_u(k)$, is considered to be unlocked, since it adapts quickly to rapidly changing signals (e.g. speech) and has a relatively short-term memory:

$$y_u(k) = [1 - 2^{-5}] \bullet y(k) + 2^{-5} \bullet W[I(k)] \qquad \textbf{[48b]}$$

where $1.06 \leq y_u(k) \leq 10$

The unlocked scale factor is recursively determined from the quantiser scale factor and the discrete function W[I], see Table 48 for the values of W[I]. The latter is a function of I(k) which causes $y_u(k)$ to adapt by larger steps for larger values of I(k). This gives $y_u(k)$ the freedom to track a signal almost instantaneously. Since $y_u(k)$ is in the logarithmic domain, W[I] is effectively a multiplier of the scale factor.

The other factor, $y_l(k)$ adapts more slowly and tracks signals which change slowly (e.g., voiceband data). This factor includes a lowpass filtering of the unlocked factor $y_u(k)$:

$$y_l(k) = [1-2^{-6}] \bullet y_l(k-1) + 2^{-6} \bullet y_u(k) \qquad \textbf{[48c]}$$

By including $y_u(k)$ in the manner shown in Formula 48c, $y_l(k)$ is implicitly limited to the explicit boundaries imposed on $y_u(k)$. The same could be said about y(k). A leakage factor is which limits the memory. In this way internal discrepancies between the Encoder and Decoder, due to transmission errors or different initial conditions, exponentially fade away.

A speed-control, $a_l(k)$ adjusts the relative weighting of the locked and unlocked scale factors. It makes use of the short- and long-term averages of the coded output to determine how rapidly the signal is changing. Actually, the predicted speed control, $a_p(k)$, determines the speed control:

$$a_l(k) = \begin{cases} 1, & a_p(k-1) > 1 \\ a_p(k-1) > 1, & a_p(k-1) \le 1 \end{cases} \quad \textbf{[49a]}$$

The speed control factor is augmented each time the difference between the short- and long-term averages becomes too large, or in case of an idle channel (i.e. y(k) < 3). If neither of these conditions exist, a uniform, slowly varying signal can be assumed, such as occurs in data transmission.

$$a_p(k) = \begin{cases} (1-2^{-4})a_p(k-1) + 2^{-3}, if \ |d_{ms}(k) - d_{ml}(k)| \ge 2^{-3} d_{ml}(k) \\ (1-2^{-4})a_p(k-1) + 2^{-3}, if \ y(k) < 3 \\ (1-2^{-4})a_p(k-1) + 2^{-3}, if \ t_d(k) = 1 \\ 1, if \ t_r(k=1) \\ (1-2^{-4})a_p(k-1), otherwise \end{cases} \quad \textbf{[49b]}$$

The short-term and long-term averages of the transmitted ADPCM signal, I(k), are determined by averaging a weighted function of the output, F[I(k)] (Table 48).

$$d_{ms}(k) = [1 - 2^{-5}] \bullet d_{ms}(k - 1) + 2^{-5} \bullet F[I(k)] \quad \textbf{[49c]}$$

$$d_{ml}(k) = [1 - 2^{-7}] \bullet d_{ml}(k - 1) + 2^{-7} \bullet F[I(k)] \quad \textbf{[49d]}$$

INVERSE ADAPTIVE QUANTISATION

Inverse adaptive quantisation is the process in which the encoder output is used to determine the normalised log of the difference signal from Table 48. The result is a quantised version of the difference signal, d(k), determined by adding the scale factor, y(k), to the value specified by Table 48 and calculating, the inverse log (base 2) of this sum.

$$d_q(k) = log_2^{-1}[\{log_2 |d_q(k)| - y(k)\} + y(k)] \quad \textbf{[49e]}$$

ADAPTIVE PREDICTION

The adaptive predictive filter is a two-pole, six-zero filter used to determine the signal estimate. The combination of both poles and zeroes allows the filter to model effectively any general input signal. The sixth-order all-zero section helps to stabilise the filter and prevent if from drifting into oscillation. For both poles and zeroes, the coefficients, $a_i(k)$ and $b_i(k)$, respectively, are adapted. The adaptation is based upon a gradient algorithm to further adjust the filter model to the input signal. Figure 50 shows the sixth-order and second-order filters, respectively.

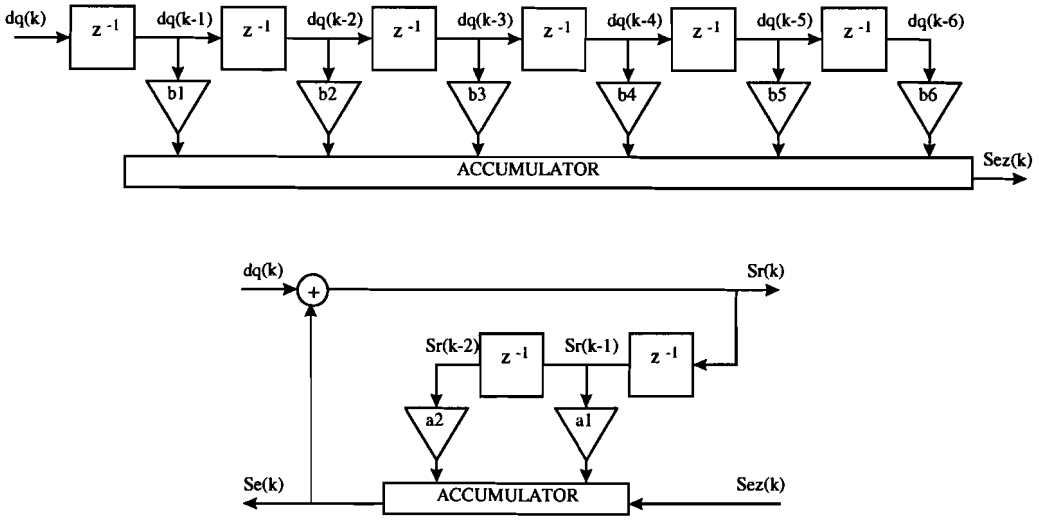


Figure 50 Sixth-order All-zero FIR-filter (above) and second-order IIR-filter (below)

The signal estimate, $S_e(k)$, represents the sum of the both filters. The reconstructed signal, $S_r(k)$, being the output in the receiver, is obtained by summing the quantised difference signal and the signal estimate:

$$S_e(k) = \sum_{i=1}^{2} a_i(k-1) S_r(k-i) + S_{ez}(k) \qquad \textbf{[50a]}$$

$$S_{ez}(k) = \sum_{i=1}^{6} b_i(k-1) d_q(k-i) \qquad \textbf{[50b]}$$

$$S_r(k-i) = S_e(k-i) + d_q(k-i) \qquad \textbf{[50c]}$$

The adaptation of the pole coefficients, $a_i(k)$, is shown below. The gradient function is determined from a signal, p(k), that is equivalent to the reconstructed signal minus the contribution of the pole filter output. The pole filter was implemented in the way shown below as to shorten the memory in the adaptation of the pole filter at the Decoder. Without the modification transmission errors would cascade through the filter for a long time [20]. Adaptation is performed by the Sign Algorithm (SA) of the LMS family. The reason for this is just to avoid computations of products. Stability of the filter is further provided by explicitly limiting the coefficients. To ensure uniform stability and adjustment of the AR portion, especially regarding narrow bandwidth signals, the standard stability constraints of a second order IIR filter are altered, but only slightly.

$$a_1(k) = [1 - 2^{-8}] \bullet a_1(k-1) + 3 \bullet 2^{-8} \bullet sgn[p(k)] \bullet sgn[p(k-1)] \qquad \textbf{[50d]}$$

where $|a_1(k)| \leq 1 - 2^{-4} - a_2(k)$

$$a_2(k) = [1 - 2^{-7}] \bullet a_2(k-1) + 2^{-7} \bullet \{sgn[p(k)] \bullet sgn[p(k-2)] - f[a_1(k-1)] \bullet sgn[p(k)] \bullet sgn[p(k-1)]\} \qquad \textbf{[50e]}$$

where $|a_2(k)| \leq 0.75$

$$p(k) = d_q(k) + S_{ez}(k) \quad \textbf{[51a]}$$

$$f(a_1) = \begin{cases} 4a_1, & |a_1| \le 2^{-1} \\ 2\,\mathrm{sgn}(a_1), & |a_1| > 2^{-1} \end{cases} \quad \textbf{[51b]}$$

where sgn(0) = +0, except when p(k) = 0 and k = 0

For the coefficients, $b_i(k)$, of the sixth order all-zero filter, the adaptation procedure is similar.The gradient function, in this case, is determined by the current difference signal, $d_q(k)$, and corresponding difference signal, $d_q(k-i)$ at the specific filter tap. See the next formula. Note that the coefficients are implicitly limited to ±2.

$$b_i(k) = [1 - 2^{-8}] \bullet b_i(k-1) + 2^{-7} \bullet sgn[d_q(k)] \bullet sgn[d_q(k-i)] \quad \textbf{[51c]}$$

where i = 1,2,.. 6 and $-2 \le b_i(k) \le +2$

Note: for 40 kbit/s ADPCM the leakage factor '2^{-8}' in Formula 51c is set to '2^{-9}'.

TONE AND TRANSITION DETECTOR

In order to improve performance for signals from frequency shift keying (FSK) modems operating in the character mode, a two-step detection process is defined. First, partial band signal (e.g. tone) detection is invoked so that the quantiser can be driven into the fast mode adaptation:

$$t_d(k) = \begin{cases} 1, & a_2(k) < -0.71875 \\ 0, & otherwise \end{cases} \quad \textbf{[51d]}$$

In addition, a transition from a partial band signal is define so that the predictor coefficients can be set to zero and the quantiser can be forced into the fast mode of operation:

$$t_r(k) = \begin{cases} 1, & a_2(k) < -0.71875 \; and \; |d_q(k)| > 24 \bullet 2^{y_l(k)} \\ 0, & otherwise \end{cases} \quad \textbf{[51e]}$$

if $t_r(k) = 1$ then: $a_1 = a_2 = 0$

Note: this block is new in [G.726].

DECODER

This part only contains the matters that were not explained before. As stated, the Encoder also contains the Decoder logic with the exception of the "Synchronous Coding Adjustment"-block, which will be explained.

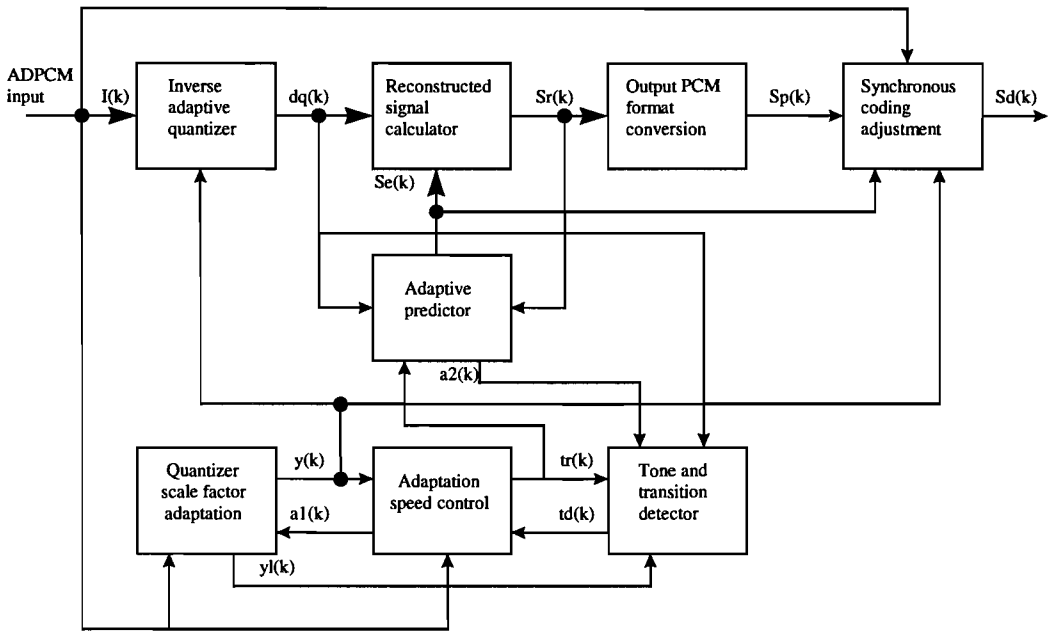


Figure 52 Block diagram of the ADPCM decoder

The reconstructed signal, $S_r(k)$, will be converted to a 8-bit log-PCM representation in the 'Output PCM format conversion' block.

RECONSTRUCTED SIGNAL SYNCHRONISATION

To avoid a cumulative distortion in synchronous tandem codings, an adjustment to the reconstructed signal is specified. The adjustment block estimates the quantisation of the encoder by determining a difference signal and executing the adaptive quantisation logic. The quantisation result is an estimate of the received value of I(k). The difference signal, $d_x(k)$, is determined by subtracting the signal estimate $S_e(k)$ from the linear-PCM signal $S_{lx}(k)$, which itself is determined by expanding the log-PCM signal $S_p(k)$.

$$d_x(k) = S_{lx}(k) - S_e(k) \qquad \textbf{[52a]}$$

The adaptive quantisation process produces the estimate of the ADPCM code value, $I_d(k)$. If the estimate implies a difference signal that is lower than the received interval boundary, the log-PCM code is changed to the next most positive value. On the other hand, when the estimate implies a difference signal larger than the received interval boundary requires the log-PCM code to be changed to the next most negative value. The adjustment is denoted as $S_d(k)$ in the following equation:

$$S_d(k) = \begin{cases} s_p^+, d_x(k) < \text{lower interval bound} \\ s_p^-, d_x(k) \geq \text{upper interval bound} \\ s_p(k), \text{otherwise} \end{cases} \qquad \textbf{[52b]}$$

Note: *The synchronous coding adjustment prevents distortion only with connections remain fully digitally, the transmitted data arrives error free, and, without disturbance of the bit streams by signal processing devices.*

5.4 IMPLEMENTATION AND TESTING OF THE ADPCM

As stated in the first chapter, the main objective was to provide for a low-level code to be used on a DSP, using fixed-point arithmetic. It had to be embedded in the company firmware. In order to be able to implement a full-duplex version and to be able to retrieve accurate benchmarks, the code had to be as compact and as fast as possible. The first since the data memory of a DSP is small, the second since the firmware already uses a considerable part of the clock cycles during a frame. Basically, three different classes of programming methods exist:

1) higher programming languages (like C and Pascal)
2) SPOX
3) assembly language

SPOX is a medium level programming language that provides portability between DSPs, but it was unavailable within the company, thus discarded. Assembler has the clear benefit of yielding the most efficient program, both in memory requirement and in clock cycles. Drawbacks are that the specific instruction set of the DSP has to be known, debugging is very troublesome and the programming-time is relatively high. Also, the program is difficult to read which make future updates hard, and, portability to other DSPs is not possible.

Higher programming languages are much less efficient compared to assembler due to the fact that the specific instruction set and available registers are not fully utilised. Major advantages are the programming speed, readability, and the debugging means. Another advantage is the reduced simulation time. E.g., simulations with TI's device simulator using a speech file of 14.000 samples took about 9 hours to encode and decode on a 200 MHz Sparcstation. With the high-level program in C this took only a few minutes. Thus, the outcome of tiny errors and different settings could be revealed rapidly. And, the values of all variables can be visualised on the screen without any difficulty. This is impossible with the TI Device Simulator. Combining all the arguments mentioned before, supports the choice for the development approach as shown in Figure 53.

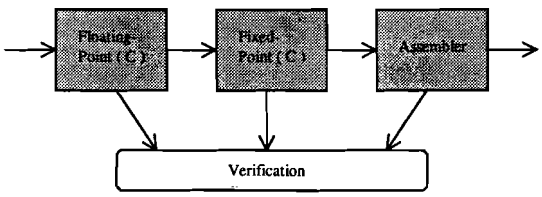


Figure 53 Programming towards an efficient implementation

It may seem a lot of extra work to develop all three realisations, but this is deceptive. The emphasis was placed on the development of the first program. After validation, this program served as an important reference for developing the program with only integer variables (fixed-point). The influence of the limited-precision of the fixed-point arithmetic could be made as small as possible. When this second program proved to work correctly the, inefficient, TI-compiler provided the low-level code that was modified afterwards. Assuming identical working of the ALU of the DSP and the Sparcstation would imply the variable values in both implementations to be identical, of course. TI's C compiler requires the program to be written with 16-bit integers only.

Testing the diverse programs:

Evaluating speech codecs may be done with either subjective or objective test methods. The easiest objective measure is the SNR, but is felt to be inadequate for a perceptual meaningful measure of digitised waveform quality [12,19]. The truly definitive measures of coded signal quality are perceptual

ones, as measured by careful subjective experimentation [P.830]. But, these are often very costly and time-consuming. They involve field-experiments and are mostly given an overall MOS-rating (Mean Opinion Score). Objective tests that didn't require field-tests, were developed [P.861] but they still don't provide the same reliable measures as with subjective testing. Especially the generation of an 'Artificial Voice' [P.50] is extremely troublesome. In [P.861] the following test factors regarding *Waveform* coders were specified:

1) Speech input levels to a codec
2) Talker dependencies
3) Bit rates if a codec has more than one bit-rate mode
4) (a)synchronous tandeming
5) Transmission channel errors

The fifth factor wasn't actually assigned to *Waveform* coders but it might as well be an appropriate factor to verify. This was not done because there was a lack of time and transmission errors are not to be expected in the laboratory. What was chosen for the tests is the following. Six different speech files were produced with the Audiotool of the Sparcstation. Four male and two female uttered a self-chosen phrase between 6000 and 26000 samples. These speech files where encoded and decoded up to ten times. Synchronous tandem coding was performed because modern telecommunication networks are close to 100% digital. The ten times tandem coding is an exaggerated number, real figures reach up to about five. The number is chosen this high to verify the quality after many tandem codings. The quality of the coding algorithm and the implementation becomes much more clear after several tandem codings. If the mentioned figure of a loss of SNR of 3dB in [21] after each synchronous tandem coding would be correct, the speech would not be intelligible at all. The test situation is depicted in Figure 54. The implemented code will only be able to use the 32 kbit/s bit-rate, the other bit-rates are discarded.

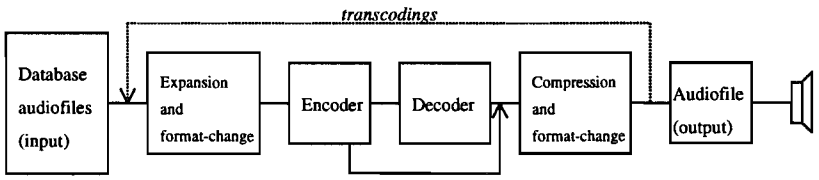


Figure 54 The test situation

The speech samples were μ-law compressed by the Audiotool. Compressing and expanding the speech may be done by means of a Look-Up-Table (LUT) and an algorithm. For this codec it was, arbitrarily, chosen to be an algorithm. The algorithm can be found in [12, TI_appl]. Companding routines were preferred to be outside the programs. The final DSP-version does not need the companding routines, as was clarified in Chapter two. The programs in C language became smaller, and, simulation time was reduced. Initial simulations were always done with the same speech file. The Encoder picked the linearised samples from the file that was saved on the harddisk. The companding routines, as they were programmed in C language, may be viewed in Appendix C.1.

Floating-point

The tone and transition detector was omitted, because no tests with modems are foreseen. If this is the case the required changes are only minor. Also the 'Synchronous Coding Adjustment' has not been implemented. Successive tandem codings is not foreseen in the laboratory and the procedure is computationally intensive. The latter is obvious because the existing Firmware has no means to compress a single channel during a frame and feed the result back to the Decoder. If in the future it would become necessary to test the other bit rates of [G.726], being 16, 24 and 40 kbit/s, the implementation is made

very easy. Because only the constants in Table 48 differ between the various bit rates these are grouped together in the program. The DP may select the bit rate and this has to be checked only once every frame.

Another important issue is the execution sequence of the equations. It is not important where to start searching for a possible execution sequence, an arbitrary equation may be chosen. If one or more variables in that equation requires an update at an earlier stage, this (these) equations(s) that do(es) so is (are) put before. Equations that update the other variables in this equation are put *after* this equation. The equation will be marked and the next will be selected. This always leads to a correct sequence. Care must be taken with equations using recursions. Newer values have to be transferred to the older ones, e.g. $S_r(k-1)$ has to be transferred to $S_r(k-2)$. It is convenient to shift the values at the of the program. This approach was followed for the Encoder and the Decoder. The complete Encoder and Decoder program can be found in Appendix C.2.

Because the Encoder contains the Decoder, the decoder was programmed *after* the encoder was debugged. This way, changes did not have to be made directly in both Encoder and Decoder and making mistakes was virtually impossible. Another method to prevent erroneous programming was to save all variables on the hard disk. This way it could be checked whether or not the variables in both encoder and decoder were identical, assuming errorfree transmission. The calculated values of the reconstructed signal in the Encoder, being $S_r(k)$, were μ-law compressed and the total file could be 'fed' to the Audiotool.

Fixed-point

Often converting a floating-point in a fixed-point implementation is done by changing the 32-bit floating-point variables to 32-bit fixed-point variables. The fixed point is set as far to the left as possible, preventing a high chance of causing overflow. From this point variable-by-variable will be changed to a 16-bit value, assuming the goal is to run the code on 16-bit hardware using fixed-point arithmetic. In case the quality decreases drastically, the variable will remain fixed at 32-bit [26]. Why this method was not adopted becomes clear later.

Thanks to implicit or explicit bounds placed on most of the variables in paragraph 5.3, overflow can never occur. The variables without these boundaries are the output of a stable filter and are thus bounded. Where the fixed point was placed in this implementation can be found in Appendix D. Most are conform the ITU-T Recommendations. The ones that differ are depicted in the same Appendix, together with the reasoning. The choice where to place the fixed point is based on the bounds, the ease of implementation and the required accuracy. For instance, when looking at Formula 48c it becomes clear that it is convenient to choose the fixed point of y_u to be 6-bits to the left compared to y_l. This way the right-shift of y_l is unnecessary. The major reason why the implementation differs from the ITU-T Recommendation is that the latter is based on ancient hardware. Modern DSPs can perform multiplications in only one clock-cycle, and, load a variable and perform a shift at the same time. Moreover, modern DSP ALUs have accumulators that are greater than 16 bits. This enables multiplication to take place with much more accuracy, since overflow will almost never occur. In the trade-off between accuracy and reducing clock cycles, the first may win. The fixed-points program can be found in Appendix C.3.

Assembler

Even with the automatically generated low-level code it took a fairly long time to obtain an efficient version. Luckily, the values of all variables of the fixed-point program could be compared with the output of the low-level code. The instruction sequence of each variable was made efficient before the next was chosen. The size of the original micro-code was reduced to about a third of the length. From this figure it becomes clear that re-writing the code is worthwhile.

To facilitate debugging, first the simulations with the TI-device simulator were performed *without* the Firmware. The format of the linearised input samples was changed and the output of the Encoder was saved to disk. With a second simulation the Decoder would generate the reconstructed speech signal and

save it on the hard disk. When this proved to work the assembly programs were altered to work within the Firmware.

To work well within the Firmware the Encoder and Decoder programs were modified. Internal Firmware variables had to be used for inputs and outputs and appropriate timeslots were chosen. To avoid looping the Encoder-output via the MUSSE to the Decoder-input a global variable in the Kernel-file was employed. The initialisation of the encoder and decoder was combined in a dedicated file, named *Init_adpcm*. Together with all other Firmware initialisations the ADPCM Encoder and Decoder are initialised. The Encoder and Decoder are obviously set to be Frame processes. The speech file(s) had to be modified to present the 8-bit speech samples at the right timeslot of the 32 slots. After simulations proved to yield the same results as the one without the Firmware, the loadable file for the DSP could be produced. The Encoder, Decoder and Init_ADPCM programs in assembly can be found in Appendix C.4.

5.5 TEST RESULTS

Simulations

Several simulations were performed using fixed-point and floating-point arithmetic. For comparison the Recommendations G.721 and G.726 were simulated. The differences are only minor, the ITU even claims compatibility. But, it might be interesting to see if there are clear differences between the old and new standards. In all simulations the speech input was tandem coded ten times. Of all tandem codings the output was written to disk. It is clear that a decrease in quality may be observed better after a certain number of (synchronous) tandem codings.

The tests that were conducted are shown in Table 56. Only the programs written in C language are simulated, due to the similarity of the fixed-point and the assembler implementations.

Test	Standard	Arithmetic	Extra
1	G.726	floating-point	Inclusion +0 in output
2	G.721	floating-point	Inclusion +0 in output
3	G.726	fixed-point	Inclusion +0 in output
4	G.721	fixed-point	Inclusion +0 in output
5	G.721	fixed-point	Exclusion +0 in output

Table 56 Simulation tests

It was mentioned in paragraph 5.3 that the value +0 would not be transmitted by the Encoder and only received by the Decoder in case of a transmission error. Clearly, it does not transmit a value but rather a sign. Initial simulations with the G.721 standard showed a considerable increase in quality if this value would indeed be transmitted! This is not hard to imagine since the sign of the quantiser scale factor, *y*, that will be corrupted, results in an error that has twice the value of this variable. An additional check verified that the Decoder and Encoder were identical *with* the transmission of +0 and not *without*. For these reasons the fifth test was added.

A major outcome of the simulations is whether the coding algorithm would be stable with the speech material, even with a number of tandem codings. As was mentioned in the introduction, *waveform* coders are generally designed to be independent from the input signals, therefore should be stable at all times. This *objective* demand was met with all tests, successive tandem codings and all speech material. Regarding the first two tests, the G.721 standard seems slightly better than the G.726 standard. This is supported by the fact that 2 out of six speech files became identical after respectively 3 and 5 tandem codings, whereas Test 1 yields no identical file. With both standards, the first coding resulted in just a small decrease in speech quality, almost inaudible. After ten tandem codings the speech still was highly intelligible. The quality degrading was audible, but the quality remained 'good'.

Roughly the same is true for the comparison between Test 3 and 4. With the fourth test, one identical result was obtained after four tandem codings. Again, with the third test no identical files were obtained. With both standards, the first tandem coding yielded an excellent result. After 10 tandem codings the degrading in speech quality was noticeable, and sometimes somewhat annoying. Regarding Test 3 there were some 'vibrations' observed in the voice. The results with the G.721 standard, again, appear better than with the other standard.

With the last test the results were much less impressive. Even after the first coding the decrease in quality was pretty audible. The results were intelligible, though. After ten tandem codings the perceived speech quality was extremely poor. Only two speech files were intelligible, but of a dreadful quality. The qualification *synthetic* is even exaggerated.

Appendix E.1 shows a number of variables which values were saved during a simulation of a fixed-point program. Also the error-signal of both fixed-point and floating-point was added. This enables the reader to see that the behaviour is fairly independent of the implemented arithmetic.

Assembler

In case the tests were done using the TI device simulator and the programs written in assembler, the simulation time would take approximately 2700 hours. This supports the idea that programs written with a high-level programming language are ideal for simulation purposes. The low-level code was tested in the laboratory, using a dedicated processor for signal processing. After downloading the microcode onto the Flash memory of the DSP-board and setting up the test bed some minor tests were performed. One was a perceptual one, the speech signal coming from one telephone was copied and transmitted to *two* telephones. One signal was left untouched and the other was coded on a DSP and then sent to the other telephone. It enabled several people to distinguish the differences between the original signal and the coded speech signal.

For the people involved with the listening tests it was almost impossible to notice differences. The noise was slightly higher which was discernible during silences. The increased noise is best observed with a sine and triangle as input, being the second test phase. With these tests the telephone sets were left unused and fixed signals were generated by a Waveform Generator on a PC. The signal was compressed according to A-law and switched to the DSP that executed the coding algorithm. The output was sent to a Waveform Analyser, also on a PC. One example is shown in Figure 58. Other examples can be found in Appendix E.2. The amplitudes of the signals seem distorted, but this is due to the poor graphical resolution of the Waveform Analyser. When looking at the psd, the increased "noise-floor" is apparent for both sine and triangular input signals. Apart from this, the inputs and outputs are highly similar. It may be important to know that coding a pure sine instead of speech results in a SNR that is 7 dB lower [20].

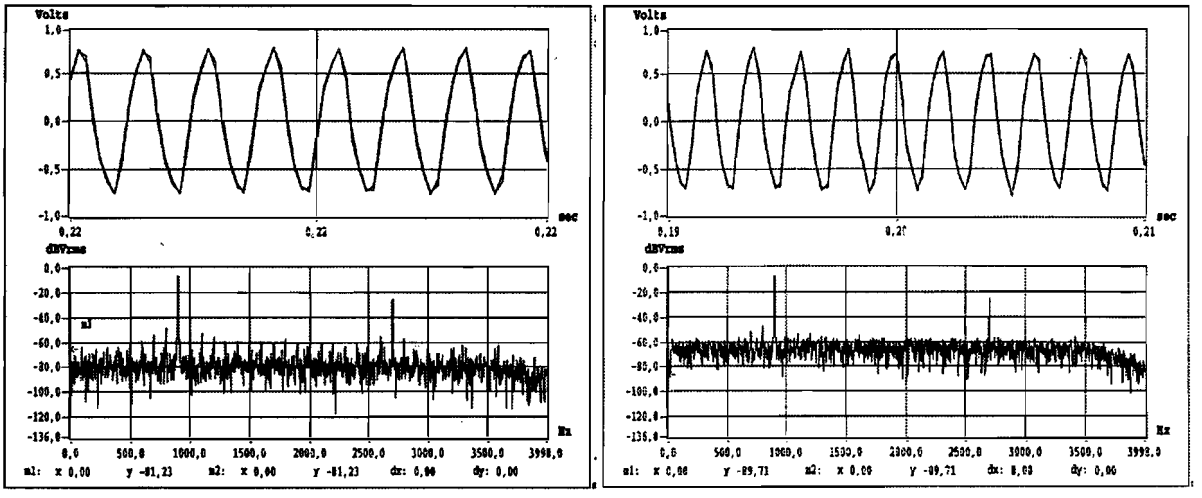


Figure 58 Laboratory test results with a triangle of 900Hz and an amplitude of 1.5V t-t. The left Figure shows the signal coming from the Waveform Generator. The right Figure shows the output signal after the signal was encoded and decoded by the ADPCM algorithm.

CHAPTER 6

CONCLUSIONS AND RECOMMENDATIONS TO THE IMPLEMENTATION OF ADPCM

6.1 CONCLUSIONS

With an ever increasing burden on transmission and storage equipment speech compression is a helpful means to increase the capacity of this equipment. Of the available Waveform, Transform and Voice Coders, Waveform coders offer a good quality at moderate price of computations and memory. It became clear from several articles that the ITU standard of ADPCM performs well, the same could be said regarding the simulations and lab-tests with this speech compression algorithm.

Still, no compilers exist that generate an efficient low-level code from high-level programming languages. In order to tackle this drawback the microcode that was obtained after writing the algorithm in C language was optimised. The implementation sequence enabled the comparison between the floating-point and the fixed-point program, and, the fixed-point program and the program written in assembly. This way debugging was much easier. From the microcode the following benchmark values were obtained

	Program memory	Data memory	Clock cycles (worst-case)
Encoder	682 words	48 words	586
Decoder	534 words	48 words	475

Table 59 ***Benchmark values of the implemented assembly program***

As depicted in Table 59 the number of clock cycles are taken to be the worst-case situation. This means that the maximum number of clock cycles was taken from a number of speech files for both Encoder and Decoder. For the DSP this would lead to about 8.5 MIPS, and, theoretically, 10 channels could be either encoded or decoded. Taking in account a margin of 20% for companding the in- and output, the changes in order to obtain the full ITU standard and the required overhead still means an ability to encode or decode 8 channels. Both Encoder and Decoder use 48 word of data memory, so every additional channel that needs to be encoded or decoded requires only an extra of 48 word of memory.

Lacking objective measurements to verify the performance of the codec, it was chosen to use 6 speech files and to tandem code the output of the codec a certain number of times. It was felt that this approach worked well. The available speech material contained enough variations to test worst-case situations. With the successive tandem codings it was possible to test the stability of the algorithm even after several times the speech was encoded and decoded. Besides, the quality of the implemented algorithm is much better observed after some synchronous tandem codings than after the first. Simulations yielded the following *subjective* quality order:

1) floating-point over fixed-point;
2) The G.721 standard over the G.726 standard (32 kbit/s);
3) *with* the transmission of +0 over *without* the transmission of +0.

The first shows that with an increased resolution of the parameters will augment the performance of the codec, though not spectacularly. Since the older ADPCM appears to yield an improved speech quality over the newest standard, it raises the question why it was altered in the first place. Perhaps the input of voiceband data shows an improved quality. A comparison between the G.721 standard test with and without the transmission of the value +0 indicated a huge improvement in speech quality when this

value was indeed transmitted. The speech material that was 10 times synchronously tandem coded still was highly intelligible. This was done without the 'Synchronous Coding Adjustment'-block of the Decoder.

6.2 RECOMMENDATIONS

In all, the programs that were developed are a valuable basis for the intended practical tests and demonstrations in the laboratory. Several simulation and tests could be carried out (for which no time was yet found), like:

- Voiceband data and tones as input signal;
- The different bit rates of ADPCM (16,24 and 40 kbit/s);
- A comparison with the full ITU standard;
- The performance with transmission errors;
- A-law versus μ-law companding;
- The linear prediction with the original LMS algorithm (instead of the Sign-LMS);
- Voice Activity Detection with ADPCM;
- Co-working of ADPCM and an echo canceller (DECT);
- Increasing the order of the MA-portion;
- Finding out whether or not the G.721 and G.726 are compatible, as is claimed by the ITU;
- Investigating proposed modifications in [24,25].

REFERENCES AND BIBLIOGRAPHY

ETSI Recommendations

[ETSI_1] Technical Report ETR 015, Radio equipment and Systems; Digital European Cordless Telecommunications (DECT); Reference document, March 1991

[ETSI_2] ETSI standard ETS 300 175-8, final draft, Radio Equipment and Systems (RES); Digital Enhanced Cordless Telecommunications (DECT); Common Interface (CI); Part 8: speech coding and transmission, June 1996

[ETSI_3] ETSI Technical Report ETR 250, Transmission and Multiplexing (TM); Speech communication quality from mouth to ear for 3.1 kHz handset telephony across networks, March 1996

ITU-T Recommendations

[G.100] Definitions used in Recommendations on general characteristics of international telephone connections and circuits, March 1993
[G.114] One-way transmission time, February 1996
[G.131] Control of talker echo, August 1996
[G.164] Echo suppressors, 1988
[G.165] Echo cancellers, March 1993
[G.168] Digital echo cancellers, July 1997
[G.704] Synchronous frame structures at 1544, 6312, 2048, 8488 and 44736 kbit/s hierarchical levels, July 1995
[G.711] Pulse Code Modulation (PCM) of voice frequencies, October 1984
[G.721] Adaptive Differential Pulse Code Modulation (ADPCM), August 1986
[G.726] 40, 32, 24, 16 kbit/s Adaptive Differential Pulse Code Modulation (ADPCM), Dec. 1990
[P.830] Subjective performance assessment of telephone-band and wideband codecs, Febr.1996
[P.861] Objective quality measurement of telephone-band (300-3400 Hz) speech codecs, Aug.1996
[P.340] Transmission characteristics of hands-free telephones, August 1996
[P.50] Artificial voices, March 1993

Other references

(the reference-numbers that are written in italics are obtained from the first reference above this (these) one(s), which has NOT been written in italics)

[TI_user] Texas Instruments, TMS320C5xx C Source Debugger; User's Guide, 1994
[TI_sim] Texas Instruments, Assembly Language Tools; User's Guide, 1995
[TI_appl] Texas Instruments, Digital Signal Processing with the TMS320 Family; Theory, Algorithms and Implementations, volume 1, 1989

[1] Elmalki, K., "A novel approach to high quality voice using echo cancellation and silence detection", Ph.D. thesis, faculty of engineering dept. of computer science, University of Sheffield (UK), April 1998

[2] Johannesson, N. O., "The ETSI Computation Model: A Tool for Transmission Planning of Telephone Networks", IEEE Communications Magazine, p.70-79, Jan. 1997

[3] Sugiyama, A., Ikeda, S., "A Fast Convergence Algorithm for Adaptive FIR Filters with Sparse taps", IEICE Trans. Fundamentals, Vol. E77-A, No.4, Apr. 1994

[4] Kawamura, S., Hatori, M., "A Tap Selection Algorithm for Adaptive Filters", Proceedings International Conference on ASSP, ICASSP 86, p. 2979-2982, Apr. 1986

[5] Nakada, H., Sato, K., "Variable Rate Speech Coding for Asynchronous Transfer Mode", IEEE trans. on Comm.s, Vol. 38, No. 3, p. 277-284, March 1990

[6] Yatsuzuka, Y., "Highly Sensitive Speech Detector and High-Speed Voiceband Data Discriminator in DSI-ADPCM Systems", IEEE Trans. on Comm.s., Vol. 30, No. 4, p. 739-750, April 1982

[7] Haykin, S., "Adaptive Filter Theory", second edition, Prentice-Hall, Englewood Cliffs, 1991

[8] Eriksson, A. et al, "Ericsson echo cancellers- a key to improved speech quality", Ericsson Review, No. 1, p.25-33, 1996

[9] Gritton, C.W.K., Lin, D.W., "Echo Cancellation Algorithms", IEEE ASSP Magazine, p.30-38, April 1994

[10] Lewis, A., "Adaptive filtering – applications in telephony", BT Technol. J., Vol. 10, No. 1, p.49-63, January 1992

[11] Cioffi, J.M., "Limited-Precision Effects in Adaptive Filtering", IEEE transactions on circuits and systems, Vol. Cas-34, No. 7, p.821-832, July 1987

[12] Jayant, N.S., Noll, P., "Digital Coding of Waveforms; Principles and Applications to Speech and Video ", Prentice-Hall, Englewood Cliffs, 1984

[13] Rabiner, L.R., Schafer, R.W., "Digital Processing of Speech Signals", Prentice-Hall, Englewood Cliffs, 1978

[14] Atal, B.S. and Hanauer, S.L., "Speech analysis and Synthesis by linear prediction of the speech wave", J. acoust. Soc. Am., Vol. 50, No. 2 (part 2), pp.637-655, August 1971

[15] Noll, P., "A comparative study of various schemes for speed encoding", Bell Systems Tech. J., Vol. 54, No. 9, pp. 1597-1614, November 1975

[16] Flanagan, J.L., Coker, C.H., Rabiner, L.R., Schafer, R.W., and Umeda, N., "Synthetic voices for computers", IEEE spectrum, Vol. 7., No. 10, pp.22-45, October 1970

[17] Noll., P., "Non-adaptive and adaptive DPCM of speech signals", Polytech. Tijdschr. ED. Elektrotech/Elektron (The Netherlands), No. 19, 1972

[18] Jayant, .S., "Adaptive quantisation with a one-word memory", Bell System tech. J., pp.1119-1144, September 1973

[19] Flanagan, J.L., Schroeder, M.R., Jayant, N.S., Atal, B.S., Crochiere, R.E., and Tribolet, J.M., "Speech Coding", IEEE trans. on Comm, Vol. COM-27, No. 4, p. 710-733, April 1979

[20] Bonnet, M., Macchi, O., and Jaidane-Saidane, M., Theoretical Analyses of the ADPCM CCITT Algorithm", IEEE trans. on Comm., Vol. 38, No. 6, p. 847-856, June 1990

[21] Bonnet, M., Macchi, O., and Jaidane-Saidane, M., "Mistracking in Successive PCM/ADPCM Transcoders", IEEE trans. on Comm., Vol. 37, No. 8, p.843-850, August 1989

[22] Watkins, C.R., Bitmead, R.R., and Crisafulli, S., "Destabilization Effects of Adaptive Quantization in ADPCM", IEEE trans. on Speech and Audio Processing, Vol. 3, No. 2, p. 137-141, March 1995

[23] Rabiner, L.R., "Toward Vision 2001: Voice and Audio Processing Considerations", AT&T Technical Journal, p. 4-13, March/April 1995

[24] Crisafulli, S., Rey, G.J., Johnson R., Jr., and Kennedy, R.A., "A coupled Approach to ADPCM Adaptation", IEEE trans. on Speech and Audio Processing, Vol. 2, No. 1, Part I, p. 90-93, Jan. 1994

[25] Pesquet, J.C., Tziritas G., and Macchi O., "Modified LMS algorithms for robust ADPCM", ICASSP 90, 1990 International Conference on Acoustics, Speech and Signal Processing, vol. 3, p. 1405-8, 1990

[26] Koops, E., "Fixed-point implementation of the digital AMPS speech codec", University of Technology Eindhoven, Master's thesis, ARL 91 ELE (5798), Eindhoven, 1992

APPENDICES

APPENDIX A Simulated delay

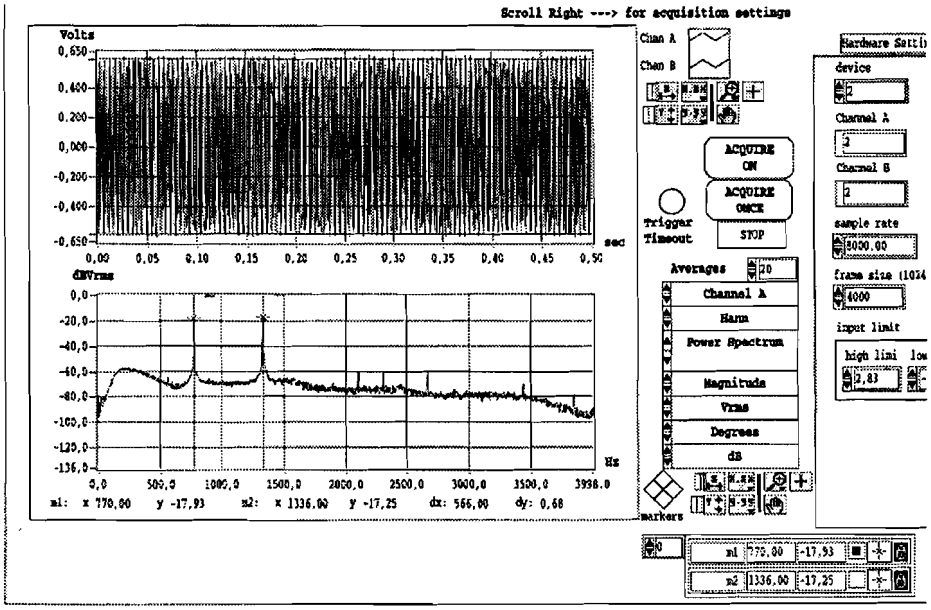
INITFIFO

```
            .mmregs
            N           .set   200          ; fixed delay set to 200/1000 samples BUFF
            .usect      "circ1", N          ; externally defined value for address
                                            ; BUFF pointing to "circ1". 1 word is
                                            ; reserved for this.

_InitFifo:  STM         #BUFF,AR2           ; retrieving the starting address of the
            MVMD        AR2,*(_fifoptr)     ; Circular Buffer and saving it to "_fifoptr"
            LD          #0,A                ; reset memory space used by CB
            RPT         #N-1                ; repeat the next line N (!) times
            STL         A,*AR2+             ; store value in acc. A in address (AR2)
                                            ; post-increment *AR2
            RET

            .global     _fifoptr
            .global     N
            .global     _InitFifo

            .end
```

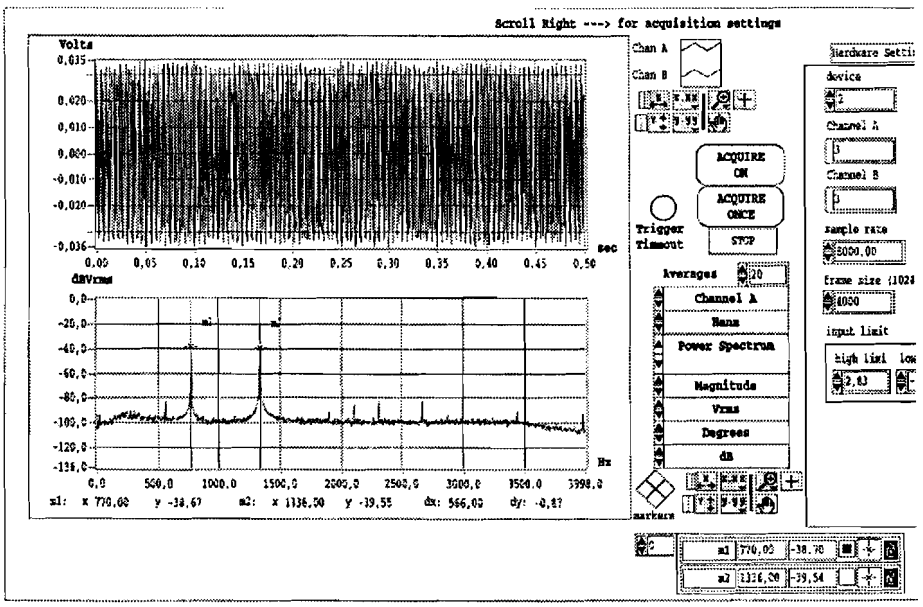
FIFO

```
            .mmregs                         ; define symbolic names for memory mapped registers
            INTS        .set   2            ; input time slot

_fifo:      STLM        A,AR2               ; copy address fifoptr to AR2
            STM         INTS,AR3            ; input timeslot saved in AR3
            STM         #N,BK               ; size CB set to N

            LD    *AR3(_Data_exp),A         ; load input sample in acc. A
            STL   A,*AR2+%                  ; store input sample in CB and incr.
                                            ; AR2 and stay within CB
            LD    *AR2,A                    ; load oldest sample in CB in acc. A
            STL   A,*(_DELSAMP)             ; store sample in address "DELSAMP"
            MVMD  AR2,*(_fifoptr)           ; save current value of AR2 in fifoptr

            RET                             ; replaces PC by value from top stack

            .global _DELSAMP                ; symbols can be externally referenced
            .global _Data_exp
            .global _fifoptr
            .global N
            .global _fifo

            .end
```

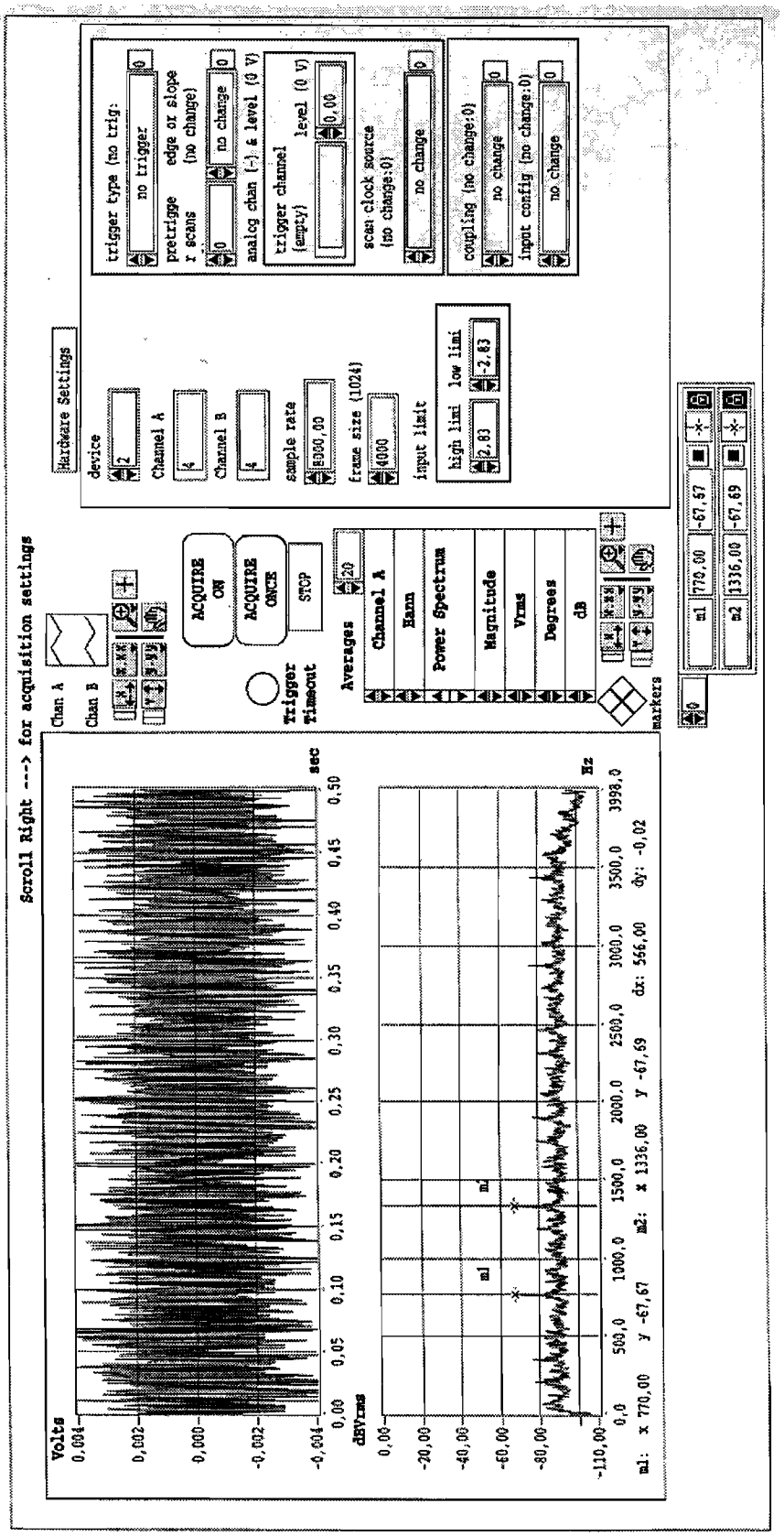
APPENDIX B LABORATORY RESULTS ECHO CANCELLER



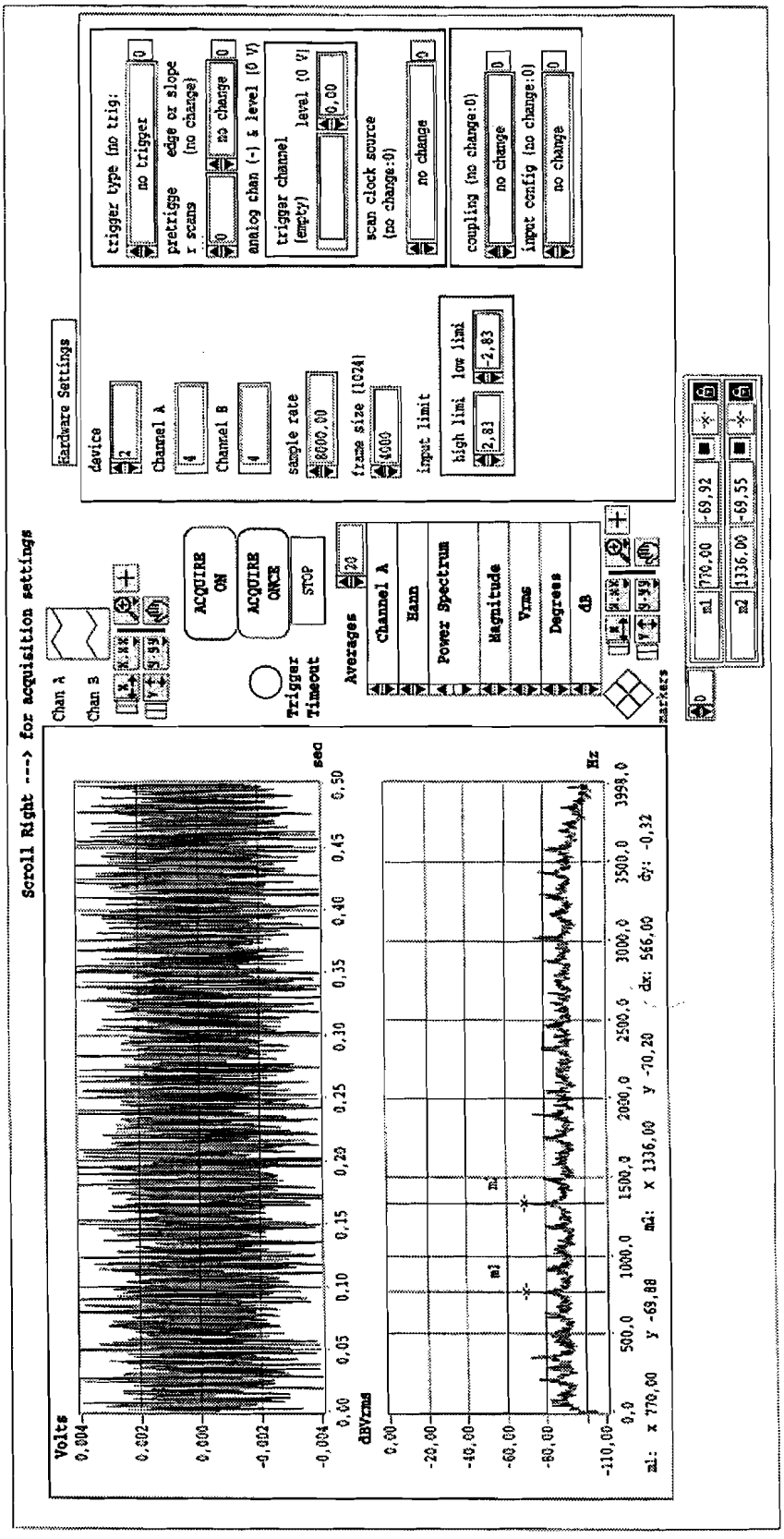
Snapshot of the input signal going to the hybrid



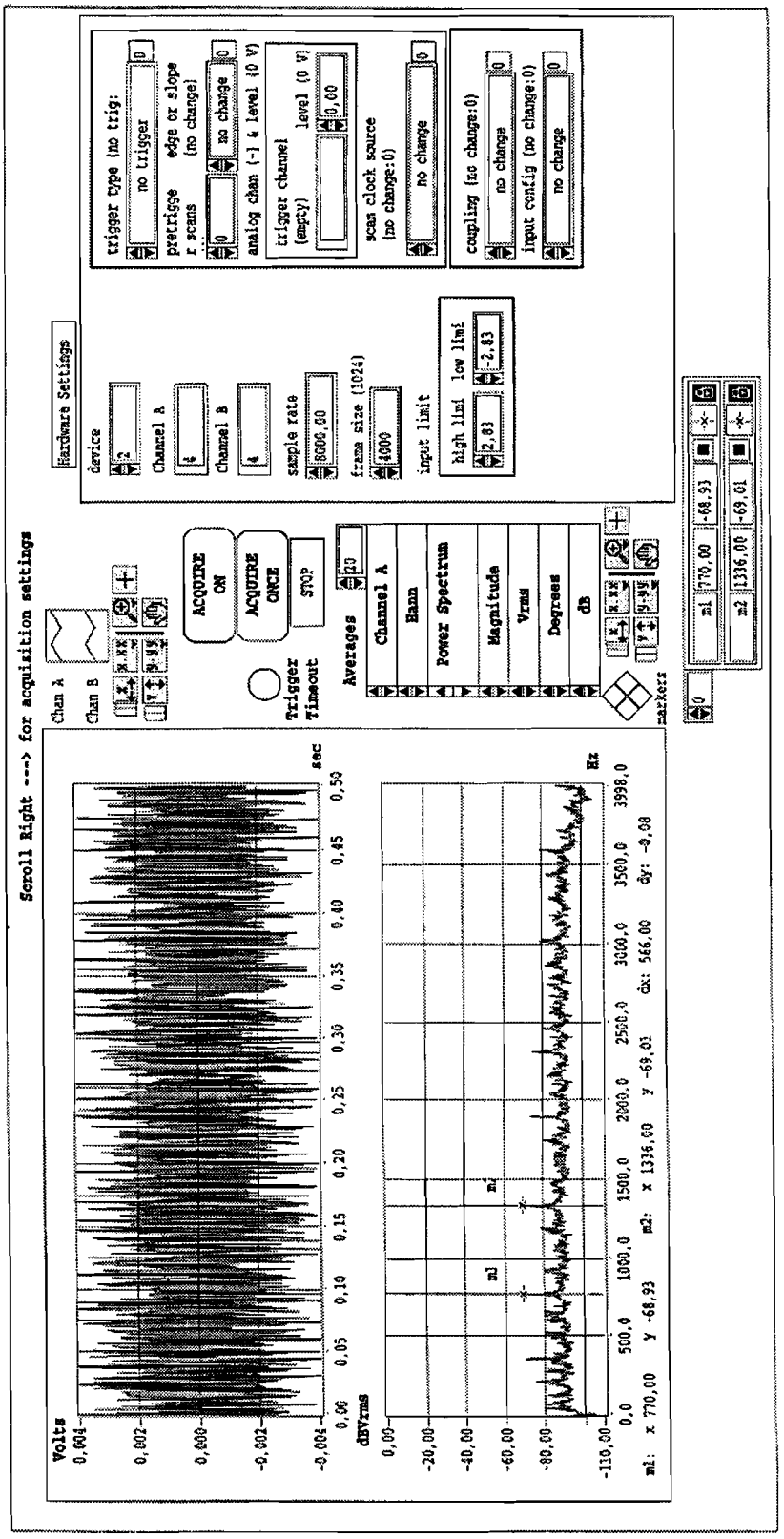
Snapshot of the signal returning from the hybrid
(about 20dB lower than the signal that enters the hybrid)



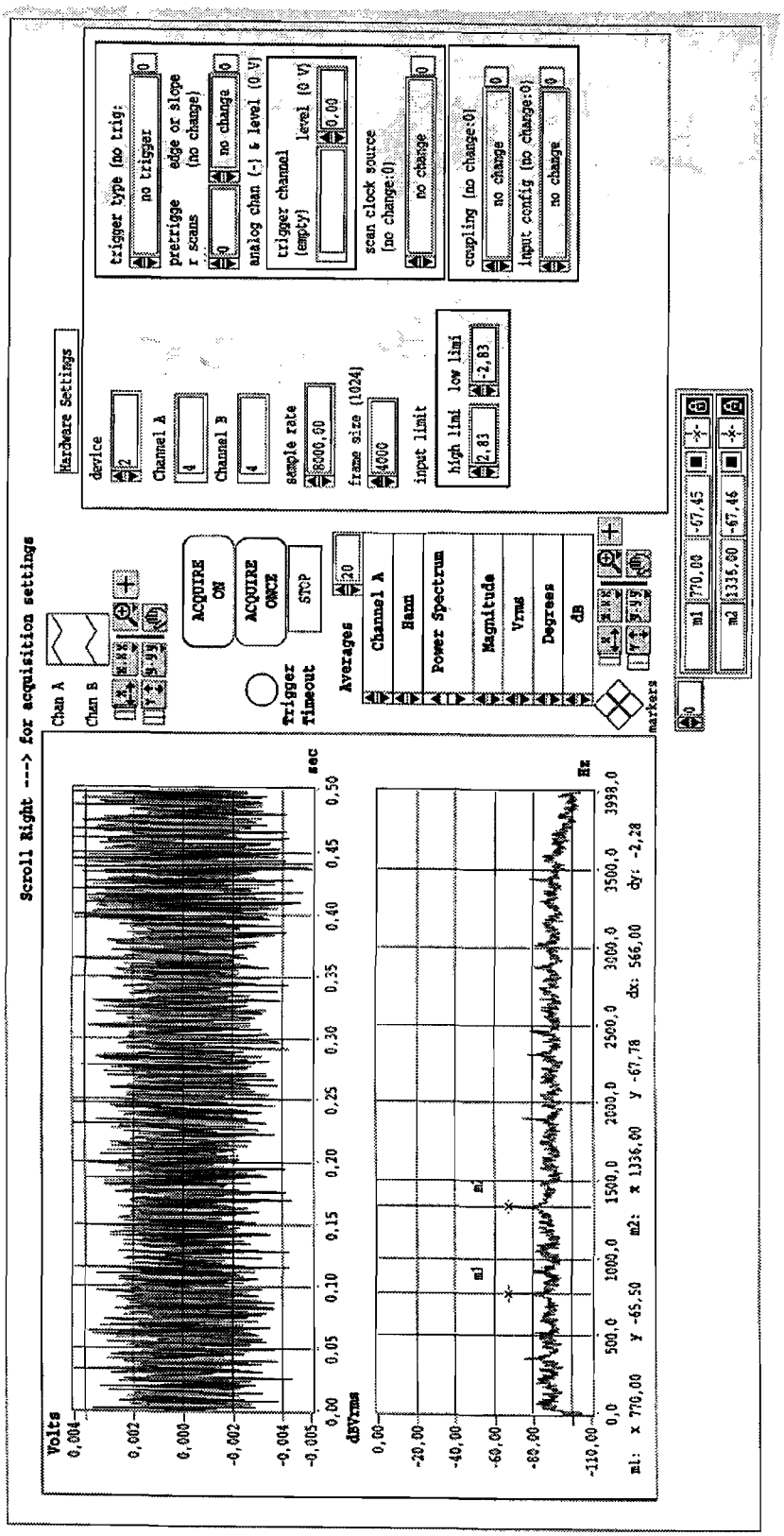
Snapshot belonging to Test 1 (see Table 33)



Snapshot belonging to Test 2 (see Table 33)



Snapshot belonging to Test 3 (see Table 33)



Snapshot belonging to Test 4 (see Table 33)

APPENDIX C.1 Companding routines

```
#include <stdio.h>

 /* this program will convert m-law encoded samples to the linear format. */
 /* Both input- and output-file are binary files, so the output is saved as two 8-bit characters */
 /* Audio files on a Unix-station contain 32 bytes at the beginning, these are erased */


main()
{
 short sign,s,value2,value;
 FILE *fp1,*fp2;

 fp1=fopen("data/sound8.in", "r");                  /* open "sound8.in" in read-mode */
 fp2=fopen("data/enc.in", "w");                     /* open "enc.in" in write-mode */

 for (s = 0; s < 33; ++s)                           /* exclude first 32 information byte */
  value = fgetc(fp1);                               /* get sample in file */

 while(value != EOF)
 {

  value = value^255;                                /* invert value */
  if (value > 127 )                                 /* remove polarity bit */
  {
   sign = 1;
   value = value - 128;
  }
  else sign = 0;

  s = (value >> 4);                                 /* determine SSS */
  value =  ((value - (s << 4)) << 1) + 33;          /* determine 2*QQQQ + 33 */

  for (s = s; s > 0; s--)                           /* right-shifting SSS times*/
  value = value << 1;

  value = value - 33;                               /* remove bias */
  if (sign==1) value = value + 8192;                /* add polarity bit */
  value2 = ((value >> 8) & 255);                    /* obtain high-byte */
  value = (value & 255);

  fprintf(fp2,"%c%c",value2,value);                 /* high-byte,low-byte to file */
  value = fgetc(fp1);

 }      /* while */

 fclose(fp1);                                       /* close both files */
 fclose(fp2);

}
```

```
#include <stdio.h>

/* This file will compress the linear output from the ADPCM-decoder and */
/* add the 32 information byte. */

long count(FILE *f)                /* in this function the length of the file is */
{                                  /* read. Since all samples are written in */
 unsigned int nc,c;                /* seperate lines, it's best to count these. */

 for(nc=0; c!=EOF; nc++)
   c = fgetc(f);
 nc = ((nc-1) >> 1);
 return nc;                        /* the number will be usable in the "main" */
}

main()
{
 short          value,sign,s,begin[32],p,q;
 unsigned int   nc;
 FILE           *fp1,*fp2;

 begin[0]=46;                      /* the array below contains the first 32 */
 begin[1]=115;                     /* information-bytes. */
 begin[2]=110;
 begin[3]=100;
 begin[4]=0;
 begin[5]=0;
 begin[6]=0;
 begin[7]=32;                      /* number of control-bytes */
 begin[8]=0;
/* begin[9..11]                   contains the file-length */
 begin[12]=0;
 begin[13]=0;
 begin[14]=0;
 begin[15]=1;                      /* sets the file-type */
 begin[16]=0;
 begin[17]=0;
 begin[18]=31;                     /* sampling frequency */
 begin[19]=64;
 begin[20]=0;
 begin[21]=0;
 begin[22]=0;
 begin[23]=1;                      /* number of channels, 1=mono, 2=stereo */
 begin[24]=0;
 begin[25]=0;
 begin[26]=0;
 begin[27]=0;
 begin[28]=0;
 begin[29]=0;
 begin[30]=0;
 begin[31]=0;

 p = 0;
 q = 0;
 fp1 = fopen("data/dec.out", "r");          /* open "dec.out" in read-mode */
 fp2 = fopen("data/sound8.out", "w");       /* open "sound8.out" in write-mode */
 nc = count(fp1);                           /* count the number of samples and */
```

```
p = nc >> 8;                          /* split the number in 3 characters */
q = p >> 8;

for (s = 0; s < 9; s++)               /* write the first 9 byte to file 2 */
   fprintf(fp2,"%c",begin[s]);

fprintf(fp2,"%c%c%c",q,p,nc);         /* write the file length as 3 separate characters */

for (s = 12; s < 32; s++)
   fprintf(fp2,"%c",begin[s]);        /* the remaining information-bytes */

fp1 = fopen("data/dec.out", "r");     /* rewinding */

value = fgetc(fp1);
if (value!=EOF)
   value = (value << 8) +             /* value of high-byte */
   fgetc(fp1);                        /* value of low-byte */

while(value!=EOF)
{
 if (value > 8191)                    /* remove sign-bit */
 {
  sign = 1;
  value = value - 8192;
 }
 else sign = 0;
if ((value+33) < 8192)  value = value + 33;   /* add bias if no overflow */
else
{
 value = 127;                         /* saturating */
 goto sat;
}
for(s = 0; value > 31; s++)           /* retreive SSS */
   value = value >> 1;

 value = (value - 16) + ((s-1) << 4);  /* determine QQQQ and add SSS */
sat:
   if (sign==1) value = value + 128;  /* add sign bit */
   value = value^255;                 /* invert value */

   fprintf(fp2,"%c",value);           /* writing result to file */

   value = fgetc(fp1);
   if (value!=EOF)
      value = (value << 8) +          /* value of high-byte */
  fgetc(fp1);                         /* value of low-byte */

 } /* while */

 fclose(fp1);                         /* close both files */
 fclose(fp2);
}
```

APPENDIX C.2 Floating-point program ADPCM in C

ENCODER

```
#include <stdio.h>

/* this program will encode the linear input samples to 4-bit output samples */

main()
{

/* variables */

  float W,dq[7],b[6],p1,p2,p3,sr1,sr2,al,ap,y,yu,yl,dms,dml,sez,x,z,a1,a2,f,se,d,minus;
  short n,sign1,sign2,sign3,signx,sl,I,srp,srq;
  unsigned short F;
  FILE *fp1,*fp2,*fp3,*fp4,*fp5,*fp6,*fp7,*fp8,*fp9,*fp10,*fp11,*fp12,*fp13,
            *fp14,*fp15,*fp16,*fp17,*fp18,*fp19,*fp20, *fp21,*fp22,*fp23;
unsigned int k;                    /* additional variable to count the number of samples */

/* initialisation */

  fp1  = fopen("data/enc.in", "r");          /* open "enc.in" in read-mode */
  fp2  = fopen("data/enc_f.out", "w");       /* open "dec_f.out" in write-mode */
  fp3  = fopen("data/sl.f", "w");
  fp4  = fopen("data/sez.f", "w");
  fp5  = fopen("data/se.f", "w");
  fp6  = fopen("data/d.f", "w");
  fp7  = fopen("data/al.f", "w");
  fp8  = fopen("data/ap.f", "w");
  fp9  = fopen("data/y.f", "w");
  fp10 = fopen("data/x.f", "w");
  fp11 = fopen("data/I.f", "w");
  fp12 = fopen("data/dq1.f", "w");
  fp13 = fopen("data/sr1.f", "w");
  fp14 = fopen("data/yu.f", "w");
  fp15 = fopen("data/yl.f", "w");
  fp16 = fopen("data/dms.f", "w");
  fp17 = fopen("data/dml.f", "w");
  fp18 = fopen("data/p1.f", "w");
  fp19 = fopen("data/a2.f", "w");
  fp20 = fopen("data/a1.f", "w");
  fp21 = fopen("data/e.f", "w");             /* error signal, sl-sr1 */
  fp22 = fopen("data/b1.f", "w");
  fp23 = fopen("data/snd.f", "w");

  dq[2,3,4,5,6,7] = 0;
  b[1,2,3,4,5,6] = 0;

  p1 = 1;
  p2 = 1;
  sr1 = 0;
  sr2 = 0;

  ap = 0;
  yl = 1.06;
  yu = 1.06;
  dms = 0;
  dml = 0;
  a1 = 0;
  a2 = 0;
/* MAIN PROGRAM */
```

```
k=0;
while( 1==1)
{
k=k+1;

/* READ LINEARISED INPUT X-BIT SAMPLE */

 sl = fgetc(fp1);
 if (sl!=EOF) sl = (sl << 8) + fgetc(fp1);        /* combine two bytes */
 else goto end;

 if (sl > 8191)                                   /* produce sign-extention */  {
  sl = 8192 - sl;
 }

/* COMPUTE SIGNAL ESITMATE */

 sez = 0;
 for ( n = 1 ; n < 7 ; n++ )
  sez = sez + ( b[n] * dq[n+1] );

 se  = sez + ( a1 * sr1 ) + ( a2 * sr2 );

fprintf(fp5,"%f\n",se);
fprintf(fp4,"%f\n",sez);
fprintf(fp20,"%f\n",a1);
fprintf(fp19,"%f\n",a2);
fprintf(fp13,"%f\n",sr1);

/* COMPUTE DIFFERENCE SIGNAL */

  d = sl - se;

fprintf(fp6,"%f\n",d);
fprintf(fp3,"%d\n",sl);

/* COMPUTE ADAPTIVE QUANTISER */

 if (ap > 1)  al = 1;
 else         al = ap;

 y = al * (yu-yl) + yl;

fprintf(fp7,"%f\n",al);
fprintf(fp8,"%f\n",ap);
fprintf(fp9,"%f\n",y);
fprintf(fp15,"%f\n",yl);
fprintf(fp14,"%f\n",yu);

/* COMPUTE QUANTISED OUTPUT */

 if (d < 0) sign1 = -1;
 else sign1 = 1;
 d = sign1 * d;

 if (d<1) d = 1;                                  /* compensation for "d-1" */
 for (n = 0; d > 2 ; n++)                         /* determine log2(abs(d)) */
  d = d / 2 ;
 x = (n + ( d - 1 ))  - y;                        /* using log(1+x)=x */
fprintf(fp10,"%f\n",x);
```

NOTE: the constants below are identical to the standard G.721, the minor differences compared to the standard G.726 (Table 48) can now be observed

```
if ( x >= 3.16 )
 {
  I = 7;   W = 69.25;    F = 7;   z = 3.34;                  /* z=log2abs(dq(k))-y(k) */
  goto eif;
 }
 if ( x >= 2.78 )
 {
  I = 6;   W = 21.25;   F = 3;   z = 2.95;
  goto eif;
 }
 if ( x >= 2.42 )
 {
  I = 5;   W = 11.5;   F = 1;   z = 2.59;
  goto eif;
 }
 if ( x >= 2.04 )
 {
  I = 4;   W = 6.12;   F = 1;   z = 2.23;
  goto eif;
 }
 if ( x >= 1.58 )
 {
  I = 3;   W = 3.12;   F = 1;   z = 1.81;
  goto eif;
 }
 if ( x >= 0.96 )
 {
  I = 2;   W = 1.69;   F = 0;   z = 1.29;
  goto eif;
 }
 if ( x >= -0.05 )
 {
  I = 1;   W = 0.25;   F = 0;   z = 0.53;
  goto eif;
 }
  I = 0;   W = -0.75;   F = 0;   z = -1.05;                  /*  "else"  */

eif:


/* WRITE THE ADPCM OUTPUT  I */

 if (sign1 == -1)  I = 15 - I ;          /* add sign-bit */

 fprintf(fp2,"%c",I);

 if  (I > 7)  I = -(I - 8);
 fprintf(fp11,"%d\n",I);

/* COMPUTE RECONSTRUCTED SIGNAL */

 dq[1] = 1;
z = z + y;
 for (z ; z >= 1 ; z--)                  /* determine dq(k)=log2^-1(abs(d)) */
    dq[1] = dq[1] * 2;
 dq[1] = dq[1] * (z + 1) * sign1;                /* using log(1+x)=x */

fprintf(fp12,"%f\n",dq[1]);

 sr2 = sr1;
```

```
 sr1  = se + dq[1];

fprintf(fp21,"%f\n",sl-sr1);

 srp = sr1;
 if (srp >  8191) srp =  8191;
 if (srp < -8191) srp = -8191;
/*sr1=srp;*/
 if (srp < 0)     srp = (-srp) + 8192;
 srq = (srp >> 8) & 255;
 srp = (srp & 255);

fprintf(fp23,"%c%c",srq,srp);            /* write Sr1 to "audio-file" */


/* COMPUTE SCALE FACTOR */

 yu  = (0.96875 * y) + (0.03125 * W);
 if    (yu < 1.06) yu = 1.06;
       else if (yu > 10)   yu = 10;

 yl  = 0.984375 * yl + 0.015625 * yu;


/* COMPUTE SPEED CONTROL */

 dms = 0.96875 * dms + 0.03125 * F;
 dml = 0.9921875 * dml + 0.0078125 * F;
 ap  = 0.9375 * ap;
 if ( dms > dml )  minus = dms - dml;
 else              minus = dml - dms;

 if ( (minus >= (0.125 * dml) ) || (y < 3) )
    ap = ap + 0.125;

fprintf(fp16,"%f\n",dms);
fprintf(fp17,"%f\n",dml);


/* COMPUTE PREDICTOR ADAPTATION */

 if (dq[1] > 0)      sign2 =  1;
 else if (dq[1] < 0) sign2 = -1;
 else                sign2 =  0;
 for ( n = 1 ; n < 7 ; n++ )
 {
  if (dq[n+1] < 0)      sign2 = -sign2;
  b[n] = (0.99609375 * b[n]) + (0.0078125 * sign2);
 }

fprintf(fp22,"%f\n",b[1]);

 for (n = 7; n > 1; n--)            /* shifting dq's to an 'older' position */
    dq[n] = dq[n-1];

 p3 = p2;
 if (p1==0) p2 = 1;
 else p2 = p1;
 if      ((dq[1] + sez) > 0)   p1 =  1;
 else if  ((dq[1] + sez) < 0)  p1 = -1;
    else                       p1 =  0;
fprintf(fp18,"%f\n",p1);
```

```
if (a1 < 0)  sign3 = -1;
else         sign3 =  1;

if ( (sign3 * a1) > 0.5)  f = 2 * sign3;
  else                    f = 4 * a1;

a2 = ( 0.9921875 * a2 ) +
        0.0078125 * ((p1 * p3) - (f * p1 * p2));
if (a2 > 0) signx =  1;
else        signx = -1;
if ( (signx * a2) > 0.75 ) a2 = signx * 0.75;              /* stability boundary */

a1 = ( 0.99609375 * a1 ) + ( 0.01171875 * p1 * p2);
if (a1 > 0) signx =  1;
else        signx = -1;
if ((signx * a1)  > (0.9375-a2))
        a1 =  signx * (0.9375 - a2);              /* stability boundary */

}   /* while */

end:                                              /* go out of loop in case of EOF */
printf("%d",k);

fclose(fp1);                                      /* close both files */
fclose(fp2);


} /* main */
```

DECODER

```
#include <stdio.h>

/* this program will decode the 4-bit input samples to a linear output signal */

main()
{

/* variables */

 float W,dq[7],b[6],p1,p2,p3,sr1,sr2,al,ap,y,yu,yl,dms,dml,sez,z,a1,a2,f,se,minus;
 short n,sign1,sign2,sign3,sign4,sign5,signx,I,srp,srq;
 unsigned short F;
 FILE *fp1,*fp2,*fp3,*fp4,*fp5,*fp6,*fp7,*fp8,*fp9,*fp10,*fp11,*fp12,
      *fp13,*fp14,*fp15,*fp16,*fp17,*fp18,*fp19,*fp20;
 int k;

/* initialisation */

 fp1  = fopen("data/dec_f.in", "r");          /* open "enc.in" in read-mode */
 fp2  = fopen("data/dec_f.out", "w");         /* open "dec_f.out" in write-mode */
 fp4  = fopen("data/Sez.f", "w");
 fp5  = fopen("data/Se.f", "w");
 fp7  = fopen("data/Al.f", "w");
 fp8  = fopen("data/Ap.f", "w");
 fp9  = fopen("data/Y.f", "w");
 fp11 = fopen("data/Id.f", "w");
 fp12 = fopen("data/Dq1.f", "w");
 fp13 = fopen("data/Sr1.f", "w");
 fp14 = fopen("data/Yu.f", "w");
 fp15 = fopen("data/Yl.f", "w");
 fp16 = fopen("data/Dms.f", "w");
 fp17 = fopen("data/Dml.f", "w");
 fp18 = fopen("data/P1.f", "w");
 fp19 = fopen("data/A2.f", "w");
 fp20 = fopen("data/A1.f", "w");

 dq[2,3,4,5,6,7]=0;
 b[1,2,3,4,5,6]=0;

 p2=1;
 p1=1;

 sr1=0;
 sr2=0;

 ap=0;
 yl=1.06;
 yu=1.06;
 dms=0;
 dml=0;
 a1=0;
 a2=0;

/* MAIN PROGRAM */

k=0;
while( 1==1)
{
k=k+1;


 /* READ 4-BIT ENCODED SIGNAL */
```

```
 I = fgetc(fp1);
 if (I==EOF) goto end;

 if (I > 7)                    /* eliminate sign */
 {
  I = 15 - I;
  sign1 = - 1;
 }
 else sign1 = 1;

fprintf(fp11,"%d\n",sign1*I);


/* COMPUTE SIGNAL ESITMATE */

 sez = 0;
 for ( n = 1 ; n < 7 ; n++ )
 sez = sez + ( b[n] * dq[n+1] );

 se  = sez + ( a1 * sr1 ) + ( a2 * sr2 );

fprintf(fp5,"%f\n",se);
fprintf(fp4,"%f\n",sez);
fprintf(fp20,"%f\n",a1);
fprintf(fp19,"%f\n",a2);
fprintf(fp13,"%f\n",sr1);


/* COMPUTE ADAPTIVE QUANTISER */

 if (ap > 1) al = 1;
 else   al = ap;

 y = al * (yu-yl) + yl;

fprintf(fp7,"%f\n",al);
fprintf(fp8,"%f\n",ap);
fprintf(fp9,"%f\n",y);
fprintf(fp15,"%f\n",yl);
fprintf(fp14,"%f\n",yu);
```

NOTE: the constants below are identical to the standard G.721, the minor differences can be compared with the standard G.726 (Table 48).

```
if ( I == 7 )
 {
  W = 69.25;  F = 7; z = 3.34;          /* z=log2abs(dq(k))-y(k) */
  goto eif;
 }
 if ( I == 6 )
 {
  W = 21.25; F = 3;  z = 2.95;
  goto eif;
 }
 if ( I == 5 )
 {
  W = 11.5;F = 1;  z = 2.59;
  goto eif;
 }
 if ( I == 4 )
 {
```

```
  W = 6.12;  F = 1;  z = 2.23;
  goto eif;
 }
 if ( I == 3 )
 {
  W = 3.12;  F = 1;  z = 1.81;
  goto eif;
 }
 if ( I == 2 )
 {
  W = 1.69;   F = 0;   z = 1.29;
  goto eif;
 }
 if ( I == 1 )
 {
  W = 0.25;  F = 0;  z = 0.53;
  goto eif;
 }
  W = -0.75;  F = 0;  z = -1.05;                    /*  "else"  */

 eif:


/* COMPUTE RECONSTRUCTED SIGNAL */

 dq[1] = 1;
 z = z + y;
 for (z ; z > 1 ; z--)                    /* determine dq(k)=log2^-1(abs(d)) */
    dq[1] = dq[1] * 2;
 dq[1] = dq[1] * (z + 1) * sign1;         /* using log(1+x)=x */

fprintf(fp12,"%f\n",dq[1]);


/* COMPUTE SCALE FACTOR */

 yu  = (0.96875 * y) + (0.03125 * W);
 if (yu<1.06) yu=1.06;
 else if (yu > 10) yu=10;

 yl  = 0.984375 * yl + 0.015625 * yu;


/* COMPUTE SPEED CONTROL */

 dms = 0.96875 * dms + 0.03125 * F;
 dml = 0.9921875 * dml + 0.0078125 * F;
 ap  = 0.9375 * ap;
 if ( dms > dml ) minus =  dms - dml;
 else             minus =  dml - dms;
 if ( ( minus >= (0.125 * dml) ) || (y < 3) )
    ap = ap + 0.125;

fprintf(fp16,"%f\n",dms);
fprintf(fp17,"%f\n",dml);
 sr2 = sr1;
 sr1  = se + dq[1];


/* COMPUTE PREDICTOR ADAPTATION */

 if (dq[1] > 0)     sign2 =  1;
 else if (dq[1] < 0) sign2 = -1;
```

```
    else        sign2 =  0;
 for ( n = 1 ; n < 7 ; n++ )
 {
  if (dq[n+1] < 0)      sign2 = -sign2;
  b[n] = (0.99609375 * b[n]) + (0.0078125 * sign2);
 }

 for (n = 7; n > 1; n--)            /* shifting dq's to an 'older' position */
   dq[n] = dq[n-1];


 p3 = p2;
 if (p1==0) p2 = 1;
 else p2 = p1;
 if     ((dq[1] + sez) > 0)  p1 =  1;
 else if ((dq[1] + sez) < 0)  p1 = -1;
    else                p1 =  0;

fprintf(fp18,"%f\n",p1);

 if (a1 < 0)  sign3 = -1;
 else       sign3 =  1;

 if ( (sign3 * a1) > 0.5)  f = 2 * sign3;
 else                      f = 4 * a1;

  a2 = ( 0.9921875 * a2 ) +
       0.0078125 * (( p1 * p3) - (f * p1 * p2 ));
 if (a2 > 0) signx = 1;
 else       signx = -1;
 if ( (signx * a2) > 0.75 ) a2 = signx * 0.75;              /* stability boundary */

 a1 = ( 0.99609375 * a1 ) + ( 0.01171875 * p1 * p2 );
if (a1 > 0) signx = 1;
 else signx = -1;
 if ( (signx * a1)  > (0.9375-a2) )
        a1 =  signx * (0.9375 - a2);              /* stability boundary */


/* WRITE THE OUTPUT sr(k) */

 srp = sr1;
 if (srp >  8191) srp =  8191;
 if (srp < -8191) srp = -8191;
 if (srp < 0)     srp = (-srp) + 8192;

 srq = (srp >> 8) & 255;
 srp = (srp & 255);

 fprintf(fp2,"%c%c",srq,srp);                     /* high-byte, low-byte */

 } /* while */

 end:

 printf("%d\n",k);

 fclose(fp1);                                     /* close both files */
 fclose(fp2);

} /* main */
```

APPENDIX C.3 Fixed-point program ADPCM in C

ENCODER

```
#include <stdio.h>

/* this program will encode the linear input samples to 4-bit output samples */

main()
{

/* variables */

 unsigned short yl;
 short F,dq[7],y,al,ap,dms,dml,W,yu,sr1,sr2,se,sez,p1,p2,p3,d,f,a1,a2,b[6],n,sign1,sign2,sign3,signx,sl,I,srp,srq;
 FILE *fp1,*fp2,*fp23;
 int k,set;

/* initialisation */

 fp1  = fopen("data/enc.in", "r");          /* open "enc.in" in read-mode */
 fp2  = fo pen("data/enc.out", "w");        /* open "enc.out" in write-mode */
 fp23 = fopen("data/snd", "w");

 dq[2,3,4,5,6,7]=0;

 b[1,2,3,4,5,6]=0;

 p1=1; p2=1;

 sr1=0; sr2=0;

 ap=0;
 yl=4342;
 yu=543;
 dms=0; dml=0;
 a1=0; a2=0;

/* MAIN PROGRAM */

k=0;
while(1==1)
{

k=k+1;


/* READ LINEARISED INPUT X-BIT SAMPLE */

 sl = fgetc(fp1);
 if (sl!=EOF) sl = (sl << 8) + fgetc(fp1);
 else goto end;

 if ((sl >> 13) == 1)                /* produce sign-extension */
    sl = 8192 - sl;

/* COMPUTE SIGNAL ESITMATE */

 set=0;
 for( n=1; n < 7; n++)
     set = set + (b[n] * dq[n+1]);

sez = set >> 13;                     /* the variable 'set' was introduced in
                                        order to yield the same results as
                                        with the DSP */

 se = (((set >> 1) + ( a1 * sr1 ) + ( a2 * sr2 )) >> 14);

/* COMPUTE DIFFERENCE SIGNAL */
```

```
d = sl - se;

/* COMPUTE ADAPTIVE QUANTISER */

if (ap > 256) al = 64;
else     al = (ap >> 2);
y = ((((al * (yu  - (yl>>3))) >> 3) + yl) >> 3);

/* COMPUTE QUANTISED OUTPUT */

if (d < 0) sign1 =  -1;
else       sign1 = 1;
d = abs(d);

if (d < 1) d = 1;
for (n = 0; (d >> n) > 1 ; n++);                    /* look for highest "1" */
if (n>7) d =  (((d - (1 << n)) >> (n - 7)) + (n << 7)) - (y >> 2);
else     d =  (((d - (1 << n)) << (7 - n)) + (n << 7)) - (y >> 2);
```

NOTE: the constants below are, apart from being implemented in fixed-point notation, identical to the standard G.726, as they were depicted in Table 48

```
if ( d > 399 )
{
 I = 7;   W = 1122;    F = 112;   dq[1] = 425;                    /* dq[1]=log2abs(dq(k))-y(k) */
 goto eif;
}
if ( d > 348 )
{
 1 = 6;    W = 355;   F = 48;   dq[1] = 372;
 goto eif;
if ( d > 300 )
{
I = 5;    W = 198;   F = 16;   dq[1] = 323;
 goto eif;
}
if ( d > 244 )
{
 1 = 4;   W = 112;   F = 16;   dq[1] = 273;
 goto eif;
}
if ( d > 277 )
{
 I = 3;   W = 64;   F = 16;   dq[1] = 212;
 goto eif;
}
if ( d > 79        )
{
 I = 2;   W = 41;   F = 0;   dq[1] = 134;
 goto eif;
}
if ( d > -125 )
{
 I = 1;   W = 18;   F = 0;   dq[1] = 4;
 goto eif;
}
 I = 0;    W = -12;   F = 0;   dq[1] = - (y>>2);              /* "else" */

eif:

/* COMPUTE RECONSTRUCTED SIGNAL */

dq[1] = dq[1] + (y >> 2);

n = dq[1] >> 7;
if (n>7) dq[1] = (((dq[1] - (n << 7)) << (n-7)) + (1 << n)) * sign1;
  else    dq[1] = (((dq[1] - (n << 7)) >> (7-n)) + (1 << n)) * sign1;
```

```
 sr2 = sr1;
 sr1  = se + dq[1];

 srp = sr1;

 if (srp >  8191) srp =  8191;
 if (srp < -8191) srp = -8191;
 if (srp < 0)     srp = (-srp) + 8192;

 if (srp < 0)     srp = (-srp) + 8192;

 srq = (srp >> 8) & 255;
 srp = (srp & 255);

fprintf(fp23,"%c%c",srq,srp);                /* write to "audiofile"*/


/* WRITE THE ADPCM OUTPUT I */

 if (sign1 == -1)  I = (15 - I);             /* add sign-bit, */

fprintf(fp2,"%c",I);                         /*write to output-file*/


/* COMPUTE SCALE FACTOR */

 yu  = (y - (y >> 5)) + W;
 if (yu < 543)  yu =  543;
 else if (yu > 5120) yu = 5120;              /* boundaries */
 yl  =  (yl - (yl >> 6)) + (yu >> 3);
/* COMPUTE SPEED CONTROL */
 dms = (dms - (dms >> 5)) + F;
 dml = (dml - (dml >> 7)) + F;

 if ( ( abs ((dms << 2) - dml) >= (dml >> 3) ) | (y < 1636) )
 ap = (ap - (ap >> 4)) + 32;
  else  ap  = (ap - (ap >> 4));


/* COMPUTE PREDICTOR ADAPTATION */

 if     (dq[1] > 0)  sign2 =  1;
 else if (dq[1] < 0)  sign2 = -1;
    else          sign2 =  0;

 for ( n = 1 ; n < 7 ; n++ )
 {
  if (dq[n+1] < 0)      sign2 = -sign2;
  b[n] = (( (b[n] - (b[n] >> 8)) + (sign2 << 6) ) );
 }

 for (n = 7; n > 1; n--)                     /* shifting dq's to an 'older' position */
   dq[n] = dq[n-1];
 if (a1 < 0) sign3 = -1;
 else        sign3 =  1;

 if ( (sign3 * a1) > 8192)  f = sign3 << 8;
    else                    f =  a1 >> 5;

 p3 = p2;
 if (p1==0) p2 = 1;
 else p2 = p1;
 if     ((dq[1] + sez) > 0)  p1 =  1;
 else if ((dq[1] + sez) < 0)  p1 = -1;
    else                  p1 =  0;

 a2 =     ((a2 - (a2 >> 7)) +
      (((p1 * p3) << 7) - (f * p1 * p2)));
 if (a2 < 0) signx = -1;
 else          signx =  1;
```

```
if ( (signx * a2) > 12288 )
    a2 = signx * 12288;                    /* stability boundary */

a1 = ((a1 - (a1 >> 8)) + ((3 * p1 * p2) << 6));
if (a1 < 0) signx = -1;
else        signx =  1;

if ( (signx * a1) > (15360 - a2 ))
        a1 = signx * (15360 - a2);                /* stability boundary */


} /* while */

end:                                       /* go out of loop in case of EOF */
printf("%d\n",k);

fclose(fp1);                                      /* close both files */
fclose(fp2);


} /* main */
```

DECODER

```
#include <stdio.h>

/* this program will encode the linear input samples to 4-bit output samples */

main()
{


/* variables */

  unsigned short yl;
  short F,dq[7],y,al,ap,dms,dml,W,yu,sr1,sr2,se,p1,p2,p3,d,sez,f,a1,a2,b[6],n,sign1,sign2,sign3,sign4,sign5,signx,sl,I,srp,srq;
  FILE *fp1,*fp2;

 int k,set;


/* initialisation */

 fp1  = fopen("data/dec.in", "r");          /* open "enc.in" in read-mode */
 fp2  = fopen("data/dec.out", "w");         /* open "enc.in" in write-mode */

 dq[2,3,4,5,6,7=0;

 b[1,2,3,4,5,6]=0;

 p1=1;
 p2=1;

 sr1=0;
 sr2=0;

 ap=0;
 yl=4342;
 yu=543;
 dms=0;
 dml=0;
 a1=0;
 a2=0;


/* MAIN PROGRAM */

k=0;
while( 1==1)
{

k=k+1;


/* READ 4-BIT ENCODED SIGNAL */

 I = fgetc(fp1);
 if (I==EOF) goto end;

 if (I > 7)                                 /* retrieve sign */
 {
  I = 15 - I;
  sign1 = -1;
 }
 else sign1 = 1;


/* COMPUTE SIGNAL ESITMATE */

 set=0;
 for( n=1; n < 7; n++)
    set = set + ((b[n] * dq[n+1]));
```

```
sez = set >> 13;                    /* the variable set was introduced to
                                       yield the same results as with the DSP*/

 se = (((set >> 1) + ( a1 * sr1 ) + ( a2 * sr2 )) >> 14) ;

/* COMPUTE ADAPTIVE QUANTISER */

 if (ap > 256) al = 64;
  else     al = (ap >> 2);

 y = ((((al * (yu  - (yl>>3))) >> 3) + yl) >> 3);

/* COMPUTE QUANTISED OUTPUT */
```

NOTE: the constants below are, apart from being implemented in fixed-point notation, identical to the standard G.726, as they were depicted in Table 48

```
 if ( I == 7 )
 {
  W = 1122;    F = 112;   dq[1] = 425;              /* dq[1]=log2abs(dq(k))-y(k) */
  goto eif;
 }
 if ( I == 6 )
 {
  W = 355;   F = 48;   dq[1] = 372;
  goto eif;
 }
 if ( I == 5 )
 {
  W = 198;   F = 16;   dq[1] = 323;
  goto eif;
 }
 if ( I == 4 )
 {
  W = 112;   F = 16;   dq[1] = 273;
  goto eif;
 }
 if ( I == 3 )
 {
  W = 64;   F = 16;   dq[1] = 212;
  goto eif;
 }
 if ( I == 2 )
 {
  W = 41;   F = 0;   dq[1] = 134;
  goto eif;
 }
 if ( I == 1 )
 {
  W = 18;   F = 0;   dq[1] = 4;
  goto eif;
 }
  W = -I2;  F = 0;   dq[1] = - (y>>2);          /*  "else"  */

 eif:


/* COMPUTE RECONSTRUCTED SIGNAL */

  dq[1] = dq[1] + (y >> 2);

  n = dq[1] >> 7;
  if (n>7) dq[1] = (((dq[I] - (n << 7)) << (n-7)) + (1 << n)) * sign1;
  else     dq[1] = (((dq[1] - (n << 7)) >> (7-n)) + (1 << n)) * sign1;



/* COMPUTE SCALE FACTOR */
```

```
 yu  = (y - (y >> 5)) + W;
 if (yu < 543)  yu =  543;
 else if (yu > 5120) yu = 5120;                /* boundaries */
 yl  =  (yl - (yl >> 6)) + (yu >> 3);


/* COMPUTE SPEED CONTROL */

 dms = (dms - (dms >> 5)) + F;
 dml = (dml - (dml >> 7)) + F;

 if ( ( abs ((dms << 2) - dml) >= (dml >> 3) ) | (y < 1636) )
 ap = (ap - (ap >> 4)) + 32;
 else  ap  = (ap - (ap >> 4));

 sr2 = sr1;
 sr1  = se + dq[1];


/* COMPUTE PREDICTOR ADAPTATION */

 if     (dq[1] > 0)   sign2 =  1;
 else if (dq[1] < 0)  sign2 = -1;
    else            sign2 =  0;

 for ( n = 1 ; n < 7 ; n++ )
 {
  if (dq[n+1] < 0)      sign2 = -sign2;
  b[n] = (( (b[n] - (b[n] >> 8)) + (sign2 << 6) ) );
 }

 for (n = 7; n > 1; n--)                  /* shifting dq's to an 'older' position */
    dq[n] = dq[n-1];

 p3 = p2;
 if (p1==0) p2 = 1;
 else  p2 = p1;
 if     ((dq[1] + sez) > 0)  p1 =  1;
 else if ((dq[1] + sez) < 0)  p1 = -1;
    else                 p1 =  0;

 if (a1 < 0)  sign3 = -1;
  else       sign3 =  1;

  if ( (sign3 * a1) > 8192)  f = sign3 << 8;
 else                  f =  a1 >> 5;

  a2 =     ((a2 - (a2 >> 7)) +
       (((p1 * p3) << 7) - (f * p1 * p2)));
  if (a2 < 0) signx = -1;
  else          signx =  1;

  if ( (signx * a2) > 12288 )
      a2 = signx * 12288;                       /* stability boundary */

  a1 = ((a1 - (a1 >> 8)) + ((3 * p1 * p2) << 6));
  if (a1 < 0) signx = -1;
  else          signx =  1;

  if ( (signx * a1) > (15360 - a2 ))
          a1 = signx * (abs(15360 - a2));               /* stability boundary */

 /* WRITE THE OUTPUT sr(k) */

  srp = sr1;
  if (srp >  8191) srp =  8191;
  if (srp < -8191) srp = -8191;

  /* if (sr1 >  8191) sr1 =  8191;
  if (sr1 < -8191) sr1 = -8192;
  srp = sr1;*/
```

```
 if (srp < 0)    srp = (-srp) + 8192;

 srq = (srp >> 8) & 255;
 srp = (srp & 255);

fprintf(fp2,"%c%c",srq,srp);                    /* high-byte, low-byte */

 } /* while */

 end:

 printf("%d\n",k);

 fclose(fp1);                                   /* close both files */
fclose(fp2);


} /* main */
```

APPENDIX C.4 Init_ADPCM, Encoder and Decoder in Assembly

Init_ADPCM:

```
        .mmregs
FP      .set    AR7
        .global _Init_adpcm

_Init_adpcm:

        LDM     SP, A           ; save SP
        STLM    A, AR4
        STM     #0c00h, SP      ;start address in datamem
        ST      #0, *SP(4)      ;dq2
        ST      #0, *SP(5)      ;dq3
        ST      #0, *SP(6)      ;dq4
        ST      #0, *SP(7)      ;dq5
        ST      #0, *SP(8)      ;dq6
        ST      #0, *SP(9)      ;dq
        ST      #0, *SP(27)     ;b1
        ST      #0, *SP(28)     ;b2
        ST      #0, *SP(29)     ;b3
        ST      #0, *SP(30)     ;b4
        ST      #0, *SP(31)     ;b5
        ST      #0, *SP(32)     ;b6
        ST      #1, *SP(20)     ;p1
        ST      #1, *SP(21)     ;p2
        ST      #0, *SP(16)     ;sr1
        ST      #0, *SP(17)     ;sr2
        ST      #0, *SP(11)     ;ap
        ST      #4342, *SP(0)   ;yl
        ST      #543, *SP(15)   ;yu
        ST      #0, *SP(12)     ;dms
        ST      #0, *SP(13)     ;dml
        ST      #0, *SP(25)     ;a1
        ST      #0, *SP(26)     ;a2

; variables Decoder

        FRAME   #48             ; modify SP
        nop
        ST      #0, *SP(4)      ;Dq2
        ST      #0, *SP(5)      ;Dq3
        ST      #0, *SP(6)      ;Dq4
        ST      #0, *SP(7)      ;Dq5
        ST      #0, *SP(8)      ;Dq6
        ST      #0, *SP(9)      ;Dq6
        ST      #0, *SP(27)     ;B1
        ST      #0, *SP(28)     ;B2
        ST      #0, *SP(29)     ;B3
        ST      #0, *SP(30)     ;B4
        ST      #0, *SP(31)     ;B5
        ST      #0, *SP(32)     ;B6
        ST      #1, *SP(20)     ;P1
        ST      #1, *SP(21)     ;P2
        ST      #0, *SP(16)     ;Sr1
        ST      #0, *SP(17)     ;Sr2
        ST      #0, *SP(11)     ;Ap
        ST      #4342, *SP(0)   ;Yl
        ST      #543, *SP(15)   ;Yu
        ST      #0, *SP(12)     ;Dms
        ST      #0, *SP(13)     ;Dml
        ST      #0, *SP(25)     ;A1
        ST      #0, *SP(26)     ;A2
        LDM     AR4, A
        STLM    A, SP           ;return to original SP
        nop                     ;pipeline latencies
        nop
        nop

        RET
```

ENCODER

```
        .mmregs
FP      .set    AR7
        .global _encoder, sez, output, se, d, Al, y, logd, sr1,
                case, inv_log, yu, yl, dms, dml, ap, sign2,
                b_n, p1, a2, a1, shift, _PASSON,
                _Data_exp
_encoder:

INTS    .set    2                       ; input time slot

        LDM     SP, A                   ;save stack
pointer
        STLM    A, AR4
        STM     #0c00h, SP              ;SP points to
datamem
        STM     INTS, AR3       ;input timeslot saved in
                                ;AR3
        LD      *AR3(_Data_exp), A      ;load linearised
input sample in acc. A
        STL     A, *SP(38)              ;Sl

sez:    LDM     SP, A
        ADD     #4, A                   ;dq2
        STLM    A, AR2
        ADD     #23, A                  ;b1
        STLM    A, AR3
        SSBX    SXM                     ;variables are
"signed"
        RPTZ    A, 5                    ;clear acc. A,
repeat next line 6 times
        MAC     *AR2+, *AR3+, A         ;dqn(k) * bn(k)
        SFTA    A, #-13, B              ;left shift A and
                                        store result in
acc. B
        STL     B, *SP(19)              ;storelow word
Sez
L6:
se:     SFTA    A, #-12, A              ;sez<<1
        LDM     SP, B
        ADD     #16, B                  ;sr1
        STLM    B, AR2
        ADD     #9, B                   ;a1
        STLM    B, AR3
        SSBX    SXM
        RPT     #1                      ;repeat twice
        MAC     *AR2+, *AR3+, A
        SFTA    A, #-14, A
        STL     A, *SP(18)              ;save Se

Al:     LD      *SP(11), #-2, A         ;load ap
        SUB     #64, A
        BCD     L8, ALEQ                ;conditional
delayed
                                        brach to L8 if
ap>=1
        ADD     #64, A
        LD      #64, A                  ;saturate al to 1
L8:
        STL     A, *SP(10)              ;al
L9:
y:      LDU     *SP(0), B       ; load yu unsigned
        SFTA    B, #-3, B
        NEG     B, B                    ;store -yu in B
        ADD     *SP(15), B              ;yu-yl
        STLM    B, T                    ;store contents of
B
                                        in temp. reg. T
        MPY     *SP(10), A              ;T * al
        SFTA    A, #-3, A
        LDU     *SP(0), B
        ADD     B, A                    ;+yl
        SFTA    A, #-3, A
        STL     A, *SP(2)

d:      LD      *SP(38), A              ;Sl
        SUB     *SP(18), A              ;Se
logd:   BC      L11, ALEQ       ;branch conditional to
L11

        BD      L12                     ;execute next
line and jump to L12
        ST      #1, *SP(35)             ;sign1
L11:
        ST      #-1, *SP(35)
        ABS     A, A
L12:    STL     A, *SP(23)              ;abs(d)

        SUB     #1, A, B
        BCD     L17, BLEQ               ;d<=1?
        nop                             ;pipeline latency
        LD      #0, B                   ;logd=0
L14:
        LD      #-1, B                  ;n(1toomany)
L15:
        ADD     #1, B
        SFTA    A, #-1, A               ;highest "1"
        BC      L15, AGT                ;jump if A>0
        STL     B, *SP(34)              ;n
L16:
        LD      *SP(34), ASM    ;n->Arithmic Shift
Memory
        LD      #1, A
        LD      A, ASM, A               ;1<<n
        STL     A, *SP(36)              ;n
        LD      *SP(23), B
        SUB     *SP(36), B              ;d-(1<<n)

        LD      #7, A
        SUB     *SP(34), A              ;7-n
        LD      *(AL), ASM      ;load low word of A in
ASM
        LD      B, ASM, B               ;d>>7-n
        LD      *SP(34), #7, A          ;n<<7
        ADD     A, B
L17:
        LD      *SP(2), #-2, A
        SUB     A, B                    ;-y
        STL     B, *SP(23)              ;logd
L19:
case:   LD      *SP(23), A              ;d
        SUB     #404, A, B
        BC      L21, BLEQ

        ST      #7, *SP(39)             ;I
        ST      #1108, *SP(14)          ;W
        ST      #112, *SP(1)            ;F
        BD      L34
        ST      #428, *SP(3)            ;log|dq|-y
L21:
        SUB     #355, A, B
        BC      L23, BLEQ

        ST      #6, *SP(39)
        ST      #340, *SP(14)
        ST      #48, *SP(1)
        BD      L34
        ST      #378, *SP(3)
L23:
        SUB     #309, A, B
        BC      L25, BLEQ

        ST      #5, *SP(39)
        ST      #184, *SP(14)
        ST      #16, *SP(1)
```

```
        BD      L34
        ST      #332, *SP(3)
L25:
        SUB     #261, A, B
        BC      L27, BLEQ
        ST      #4, *SP(39)
        ST      #98, *SP(14)
        ST      #16, *SP(1)
        BD      L34
        ST      #285, *SP(3)
L27:
        SUB     #202, A, B
        BC      L29, BLEQ

        ST      #3, *SP(39)
        ST      #50, *SP(14)
        ST      #16, *SP(1)
        BD      L34
        ST      #232, *SP(3)
L29:
        SUB     #122, A, B
        BC      L31, BLEQ

        ST      #2, *SP(39)
        ST      #27, *SP(14)
        ST      #0, *SP(1)
        BD      L34
        ST      #165, *SP(3)
L31:
        SUB     #-6, A, B
        BC      L33, BLEQ

        ST      #1, *SP(39)
        ST      #4, *SP(14)
        ST      #0, *SP(1)
        BD      L34
        ST      #68, *SP(3)
L33:
        ST      #0, *SP(39)
        ST      #-12, *SP(14)
        ST      #0, *SP(1)
        ST      #-134, *SP(3)
L34:                                    ;eif
inv_log: LD     *SP(2), #-2, A
        ADD     *SP(3), A               ;dq1+y
        SFTA    A, #-7, B
        SFTA    B, #7, B
        SSBX    SXM
        NEG     B, B
        ADD     B, A
        NEG     B, B
        SFTA    B, #-7, B
        SUB     #7, B
        LD      *(BL), ASM
        LD      A, ASM, A
        ADD     #7, B
        LD      *(BL), ASM
        LD      #1, B
        LD      B, ASM, B
        ADD     B, A
        LD      *SP(35), B
        BC      L35, BGEQ               ;sign1>=0?
        NEG     A, A                    ;-dq1
L35:
        STL     A, *SP(3)               ;dq1

sr1:    LD      *SP(16), A
        STL     A, *SP(17)              ;sr1 -> sr2
        LD      *SP(3), A
        ADD     *SP(18), A              ;se+dq1
        STL     A, *SP(16)              ;save sr1

output: LD      *SP(39), A              ;1
        LD      *SP(3), B               ;dq1
        BC      L38, BLT                ;dq1<0?
        B       L39
L38:
        SUB     #15, A                  ;TC
        NEG     A, A
L39:    STL     A, *SP(39)
        STL     A, *(_PASSON) ;save output in"PASSON"

yu:     LD      *SP(2), #-5, A
        NEG     A, A
        ADD     *SP(2), A
        ADD     *SP(14), A
        SUB     #543, A, B              ;low_bound
        BC      L41, BGEQ

        BD      L43
        LD      #543, A
L41:
        SUB     #5120, A, B
        BC      L43, BLEQ

        LD      #5120, A                ;high_bound
L43:
        STL     A, *SP(15)              ;yu
yl:     SFTA    A, #-3, B
        LDU     *SP(0), A
        SFTA    A, #-6, A
        NEG     A, A
        ADD     B, A                    ;-(yl>>6)+yu
        LDU     *SP(0), B
        ADD     A, B
        STL     B, *SP(0)

dms:    LD      *SP(12), #-5, A
        NEG     A, A
        ADD     *SP(12), A
        ADD     *SP(1), A
        STL     A, *SP(12)

dml:    LD      *SP(13), #-7, B
        NEG     B, B
        ADD     *SP(13), B
        ADD     *SP(1), B
        STL     B, *SP(13)

ap:     SFTA    A, #2, A                ;dms
        SUB     B, A                    ;dml
        ABS     A, A
        SFTA    B, #-3, B
        SUB     B, A
        LD      #0, B
        BC      plus, AGEQ
L45:
        LD      *SP(2), A               ;y<3?
        SUB     #1636, A, A
        BC      plus, AGEQ
        B       L47

plus:   LD      #32, B
L47:
        LD      *SP(11), #-4, A
        NEG     A, A
        ADD     *SP(11), A
        ADD     B, A
        STL     A, *SP(11)              ;ap

sign2:  LD      *SP(3), A
        BC      L52, ALEQ
        BD      L55
        ST      #1, *SP(36)
L52:
        BC      shift, AGEQ             ;dq1==0

        BD      L55
```

```
        ST      #-1, *SP(36)
L55:
b_n:
        LDM     SP, A
        ADD     #4, A
        STLM    A, AR2              ;dq2
        ADD     #23, A              ;b1
        STLM    A, AR3
        STM     #5, BRC             ;6 times
        RPTB    end_block-1
        LD      *SP(36), A
        LD      *AR2+, B
        BC      go, BGEQ            ;DQx>=0?
        NEG     A, A                ;-sign2
go:     SFTA    A, #6, A
        LD      *AR3, #-8, B
        NEG     B, B
        ADD     *AR3, B
        ADD     A, B
        STL     B, *AR3+            ;b_n
end_block:

shift:  LDM     SP, A
        ADD     #8, A
        STLM    A, AR2              ;dq6
        ADD     #1, A               ;dq7
        STLM    A, AR3
        nop                         ;pipeline latency
        RPT     #5
        MVDD    *AR2-, *AR3-        ;shift dqs

p2_p3:  LD      *SP(21), A
        STL     A, *SP(22)          ;p2 -> p3
        LD      *SP(20), A
        BC      L63, ANEQ           ;p1=0?

        LD      #1, A
L63:
        STL     A, *SP(21)

p1:     LD      *SP(19), A
        ADD     *SP(3), A
        BC      L65, ALEQ

        BD      L68
        ST      #1, *SP(20)
L65:
        BC      L67, AGEQ

        BD      L68
        ST      #-1, *SP(20)
L67:
        ST      #0, *SP(20)

L68:    LD      *SP(25), A
        BC      L70, AGEQ

        BD      L71
        ST      #-1, *SP(37)        ;sign3
L70:
        ST      #1, *SP(37)
L71:
a2:     LD      *SP(25), A
        ABS     A, A
        SUB     #8192, A
        BC      L73, ALEQ

        BD      L74
        LD      *SP(37), #8, A      ;sign<<8
L73:
        LD      *SP(25), #-5, A     ;a1>>5
L74:
        STL     A, *SP(24)          ;f
        LD      *SP(22), 16, A
        MPYA    *SP(20)             ;goes to B!
        SFTL    B, #7, B        ;
        LD      *SP(26), #-7, A
        NEG     A, A
        ADD     *SP(26), A
        ADD     A, B
        STL     B, *SP(36)          ;temp store
        LD      *SP(20), 16, A
        MPYA    *SP(24)
        STLM    B, T
        nop
        MPY     *SP(21), B
        LD      *SP(36), A
        SUB     B, A
        STL     A, *SP(26)          ;a2

        BC      L76, ALT

        BD      L77
        nop
        LD      #1, B
L76:
        LD      #-1, B
        ABS     A, A
L77:
        SUB     #12288, A           ;bounda2
        BC      L79, ALEQ
        LD      #12288, A
        BC      L78, BGT
        NEG     A, A
L78:    STL     A, *SP(26)          ;a2
L79:
a1:     LD      *SP(21), 16, A
        MPYA    *SP(20)
        STLM    B, T
        LD      *SP(25), #-8, B
        MPY     #192, A         ;*3<<6
        NEG     B, B
        ADD     *SP(25), B
        ADD     B, A
        STL     A, *SP(25)

        BC      L81, ALT

        BD      L82
        ST      #1, *SP(37)         ;signx
L81:
        ST      #-1, *SP(37)
        ABS     A, A
L82:
        LD      #15360, B
        SUB     *SP(26), B
        SUB     *(AL), B
        BC      L84, BGEQ

        ADD     *(AL), B
        LD      *SP(37), A
        BC      L83, AGEQ
        NEG     B, B                ;-(15360-a2)
L83:    BD      L84
        nop
        STL     B, *SP(25)

L84:    LDM     AR4, A
        STLM    A, SP
        nop                         ;pipeline latency
        nop
        nop

        RET
```

DECODER

```
        .mmregs
FP      .set    AR7
        .global _decoder,Sez,Output,Se,Al_d,Y,Logd,Sr1,
                Case,Inv_log,Yu,Yl,Dms,Dml,Ap,Sign2,B_
          n,P1,A2,A1,Shift,_PASSON,_DELSAMP

_decoder:
        LDM     SP, A
        STLM    A, AR4
        STM     #0c30h, SP

        LD      *(_PASSON), A
        STL     A, *SP(39)              ;I
        SUB     #8, A
        nop
        BCD     L4, ALT
        ST      #1, *SP(35)             ;sign1

        SUB     #7, A                   ;I-15
        ABS     A, A
        STL     A, *SP(39)              ;I
        ST      #-1, *SP(35)            ;sign1
L4:
Sez:    LDM     SP, A
        ADD     #4, A                   ;dq2
        STLM    A, AR2
        ADD     #23, A  ;b1
        STLM    A, AR3
        SSBX    SXM
        RPTZ    A, 5
        MAC     *AR2+, *AR3+, A
        SFTA    A, #-13, B
        STL     B, *SP(19)
L6:
Se:     SFTA    A, #-12, A              ;sez<<1
        LDM     SP, B
        ADD     #16, B
        STLM    B, AR2
        ADD     #9, B
        STLM    B, AR3
        SSBX    SXM
        RPT     #1
        MAC     *AR2+, *AR3+, A
        SFTA    A, #-14, A
        STL     A, *SP(18)

Al_d:   LD      *SP(11), #-2, A
        SUB     #64, A
        BCD     L8, ALEQ                ;ap>=1?
        ADD     #64, A
        LD      #64, A
L8:
        STL     A, *SP(10)              ;al
L9:
Y:      LDU     *SP(0), B
        SFTA    B, #-3, B
        NEG     B, B
        ADD     *SP(15), B              ;yu-yl
        STLM    B, T
        MPY     *SP(10), A              ;*al
        SFTA    A, #-3, A
        LDU     *SP(0), B
        ADD     B, A                    ;+yl
        SFTA    A, #-3, A
        STLA, *SP(2)
L19:
        NEG     B, B
        SFTA    B, #-7, B
        SUB     #7, B

Case:   LD      *SP(39), A
        SUB     #7, A, B
        BC      L21, BNEQ

        ST      #1108, *SP(14)
        ST      #112, *SP(1)
        BD      L34
        ST      #428, *SP(3)
L21:
        SUB     #6, A, B
        BC      L23, BNEQ

        ST      #340, *SP(14)
        ST      #48, *SP(1)
        BD      L34
        ST      #378, *SP(3)
L23:
        SUB     #5, A, B
        BC      L25, BNEQ

        ST      #184, *SP(14)
        ST      #16, *SP(1)
        BD      L34
        ST      #332, *SP(3)
L25:
        SUB     #4, A, B
        BC      L27, BNEQ

        ST      #98, *SP(14)
        ST      #16, *SP(1)
        BD      L34
        ST      #285, *SP(3)
L27:
        SUB     #3, A, B
        BC      L29, BNEQ

        ST      #50, *SP(14)
        ST      #16, *SP(1)
        BD      L34
        ST      #232, *SP(3)
L29:
        SUB     #2, A, B
        BC      L31, BNEQ

        ST      #27, *SP(14)
        ST      #0, *SP(1)
        BD      L34
        ST      #165, *SP(3)
L31:
        SUB     #1, A, B
        BC      L33, BNEQ

        ST      #4, *SP(14)
        ST      #0, *SP(1)
        BD      L34
        ST      #68, *SP(3)
L33:
        ST      #-12, *SP(14)
        ST      #0, *SP(1)
        ST      #-134, *SP(3)
L34:                                    ;eif

Inv_log: LD     *SP(2), #-2, A
        ADD     *SP(3), A               ;dq1+y
        SFTA    A, #-7, B
        SFTA    B, #7, B
        SSBX    SXM
        NEG     B, B
        ADD     B, A


        LD      *(BL), ASM
        LD      A, ASM, A
        ADD     #7, B
```

```
        LD      *(BL), ASM
        LD      #1, B
        LD      B, ASM, B
        ADD     B, A
        LD      *SP(35), B
        BC      L35, BGEQ               ;sign1>=0?
        NEG     A, A                    ;-dq1
L35:
        STL     A, *SP(3)

Yu:     LD      *SP(2), #-5, A
        NEG     A, A
        ADD     *SP(2), A
        ADD     *SP(14), A
        SUB     #543, A, B              ;low_bound
        BC      L41, BGEQ

        BD      L43
        LD      #543, A
L41:
        SUB     #5120, A, B
        BC      L43, BLEQ

        LD      #5120, A                ;high_bound
L43:
        STL     A, *SP(15)              ;yu
Yl:     SFTA    A, #-3, B
        LDU     *SP(0), A
        SFTA    A, #-6, A
        NEG     A, A
        ADD     B, A                    ;-(yl>>6)+yu
        LD      *SP(11), #-4, A
        NEG     A, A
        ADD     *SP(11), A
        ADD     B, A
        STL     A, *SP(11)              ;ap

Sr1:    LD      *SP(16), A
        STL     A, *SP(17)              ;sr2=sr1
        LD      *SP(3), A               ;dq1
        ADD     *SP(18), A
        STL     A, *SP(16)              ;sr1
        STL     A, *SP(23)              ;sr1 (output)
        nop
        BC      ploes, AGEQ

        ADD     #8192, A
        BC      L48, AGEQ
        LD      #-8192, B
        BD      L48
        STL     B, *SP(23)              ;sat sr1

ploes:  SUB     #8191, A
        BC      L48, ALEQ
        LD      #8191, A
        STL     A, *SP(23)              ;sat sr1
L48:
Sign2:  LD      *SP(3), A
        BC      L52, ALEQ

        BD      L55
        ST      #1, *SP(36)
L52:
        BC      Shift, AGEQ             ;dq1==0

        BD      L55
        ST      #-1, *SP(36)
L55:
B_n:
        LDM     SP, A
        ADD     #4, A
        STLM    A, AR2          ;dq2
        ADD     #23, A                  ;b1

        LDU     *SP(0), B
        ADD     A, B
        STL     B, *SP(0)

Dms:    LD      *SP(12), #-5, A
        NEG     A, A
        ADD     *SP(12), A
        ADD     *SP(1), A
        STL     A, *SP(12)

Dml:    LD      *SP(13), #-7, B
        NEG     B, B
        ADD     *SP(13), B
        ADD     *SP(1), B
        STL     B, *SP(13)

Ap:     SFTA    A, #2, A
        SUB     B, A
        ABS     A, A
        SFTA    B, #-3, B
        SUB     B, A
        LD      #0, B
        BC      plus, AGEQ
L45:
        LD      *SP(2), A               ;y<3?
        SUB     #1636, A, A
        BC      plus, AGEQ
        B       L47

plus:   LD      #32, B

L47:
        STLM    A, AR3
        STM     #5, BRC                 ;6 times
        RPTB    end_block-1
        LD      *SP(36), A
        LD      *AR2+, B
        BC      go, BGEQ                ;DQx>=0?
        NEG     A, A                    ;-sign2
go:     SFTA    A, #6, A
        LD      *AR3, #-8, B
        NEG     B, B
        ADD     *AR3, B
        ADD     A, B
        STL     B, *AR3+                ;b_n
end_block:

Shift:  LDM     SP, A
        ADD     #8, A
        STLM    A, AR2          ;dq6
        ADD     #1, A                   ;dq7
        STLM    A, AR3
        nop                             ;pipeline latency
        RPT     #5
        MVDD    *AR2-, *AR3-            ;shiftdq's

p2_p3:  LD      *SP(21), A
        STL     A, *SP(22)              ;p3=p2
        LD      *SP(20), A
        BC      L63, ANEQ               ;p1=0?
```

```
        LD      #1, A
L63:
        STL     A, *SP(21)

P1:     LD      *SP(19), A
        ADD     *SP(3), A
        BC      L65, ALEQ

        BD      L68
        ST      #1, *SP(20)
L65:
        BC      L67, AGEQ

        BD      L68
        ST      #-1, *SP(20)
L67:
        ST      #0, *SP(20)

L68:    LD      *SP(25), A
        BC      L70, AGEQ

        BD      L71
        ST      #-1, *SP(37)            ;sign3

L70:
        ST      #1, *SP(37)
L71:
A2:     LD      *SP(25), A
        ABS     A, A
        SUB     #8192, A
        BC      L73, ALEQ

        BD      L74
        LD      *SP(37), #8, A          ;sign<<8
L73:
        LD      *SP(25), #-5, A         ;a1>>5
L74:
        STL     A, *SP(24)              ;f

        LD      *SP(22), 16, A
        MPY     A*SP(20)                ;goes to B!
        SFTL    B, #7, B ;
        LD      *SP(26), #-7, A
        NEG     A, A
        ADD     *SP(26), A
        ADD     A, B
        STL     B, *SP(36)              ;temp store
        LD      *SP(20), 16, A
        MPYA    *SP(24)
        STLM    B, T
        nop
        MPY     *SP(21), B
        LD      *SP(36), A
        SUB     B, A
        STL     A, *SP(26)              ;a2

        BC      L76, ALT

        BD      L77
        nop
        LD      #1, B

L76:
        LD      #-1, B
        ABS     A, A
L77:
        SUB     #12288, A               ;bounda2
        BC      L79, ALEQ
        LD      #12288, A
        BC      L78, BGT
        NEG     A, A

L78:    STL     A, *SP(26)              ;a2
L79:
A1:     LD      *SP(21), 16, A
        MPYA    *SP(20)
        STLM    B, T
        LD      *SP(25), #-8, B
        MPY     #192, A         ;*3<<6
        NEG     B, B
        ADD     *SP(25), B
        ADD     B, A
        STL     A, *SP(25)

        BC      L81, ALT

        BD      L82
        ST      #1, *SP(37)             ;signx
L81:
        ST      #-1, *SP(37)
        ABS     A, A
L82:
        LD      #15360, B
        SUB     *SP(26), B
        SUB     *(AL), B
        BC      Output, BGEQ

        ADD     *(AL), B
        LD      *SP(37), A
        BC      L83, AGEQ
        NEG     B, B                    ;-(15360-a2)
L83:
        STL     B, *SP(25)

Output: LD      *SP(23), A
        STL     A, *(_DELSAMP)          ;sat(sr1) in
                                address "DELSAMP"
        LDM     AR4, A
        STLM    A, SP
        nop                             ;pipeline latencies
        nop
        nop

        RET
```

APPENDIX D Table fixed-point variables

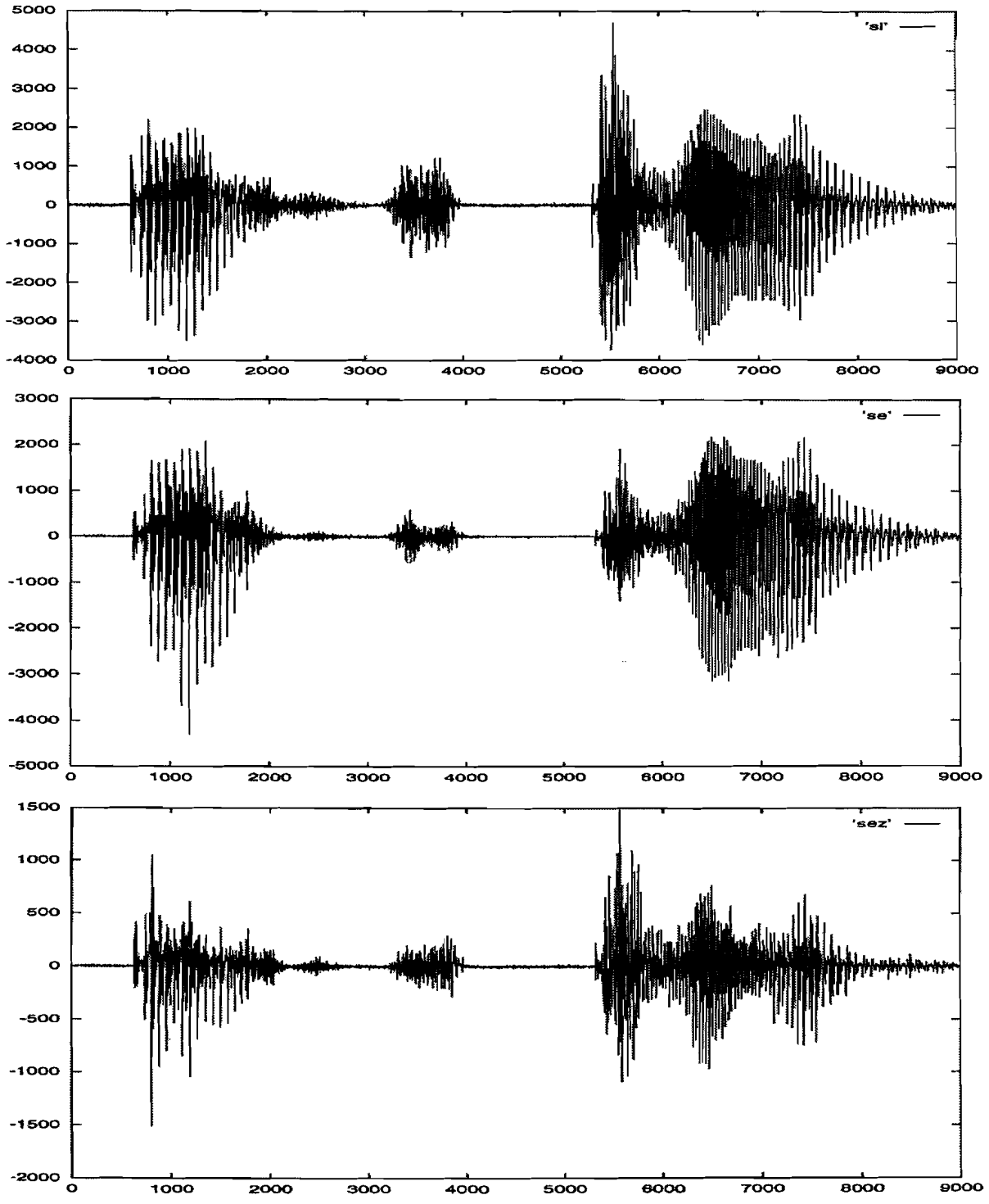
Parameter symbol	Parameter boundaries	Binary representation	Parameter description
a_1	(-1.699,1.699)	S, 2 . 13	Second order predictor coefficients
a_2	(-0.768,0.768)	S, 1 . 14	Second order predictor coefficients
a_l	[0,1]	-, 1 . 6	Limited speed control parameter
a_p	[0,2]	-, 2 . 8	Unlimited speed control parameter
b_1,..,b_6	[-2, +2]	S, 2 . 13	Sixth order predictor coefficients
d	Unbounded	S, 15 . 0	Difference signal
d_{ml}	[0, 7]	-, 3 . 11	Long-term average of F(I) sequence
d_{ms}	[0, 7]	-, 3 . 9	Short-term average of F(I) sequence
d_{q0}..d_{q7}	[-10226, +10226]	S, 15 . 0	Quantised difference signal
F	[0,7]	-, 3 . 4	Energy-measure
I	[-7, +7] discrete	-, S 3	Output
Log | d | -y	[-y, log|d|]	S, 4 . 7	Normalised difference signal
p_1,.,p_3	-	S, 15 . 0	Signal used for coefficient update of AR-portion
S_e	-	S, 15 . 0	Signal estimate
S_{ez}	-	S, 15 . 0	Sixth order partial signal estimate
S_l	[-8191,+8191]	S, 15 . 0	Linear signal input
S_{r1},S_{r2}	-	S, 15 . 0	Reconstructed signal
W	[-0.75, 70.13	S, 7 . 4	Quantiser multiplier
y	[1.06, 10]	-, 4 . 9	Quantiser scale factor
y_l	[1.06, 10]	-, 4 . 12	Slow quantiser scale factor
y_u	[1.06, 10]	-, 4 . 9	Fast quantiser scale factor

Table A_101 Internal processing variables as they were implemented

In the third column in the table above the first token denotes if the variable contains a sign, if this is the case an 'S' is placed otherwise a '-'. The following figure shows the number before the fixed-point and the last figure sets the number of bits used after the fixed-point.

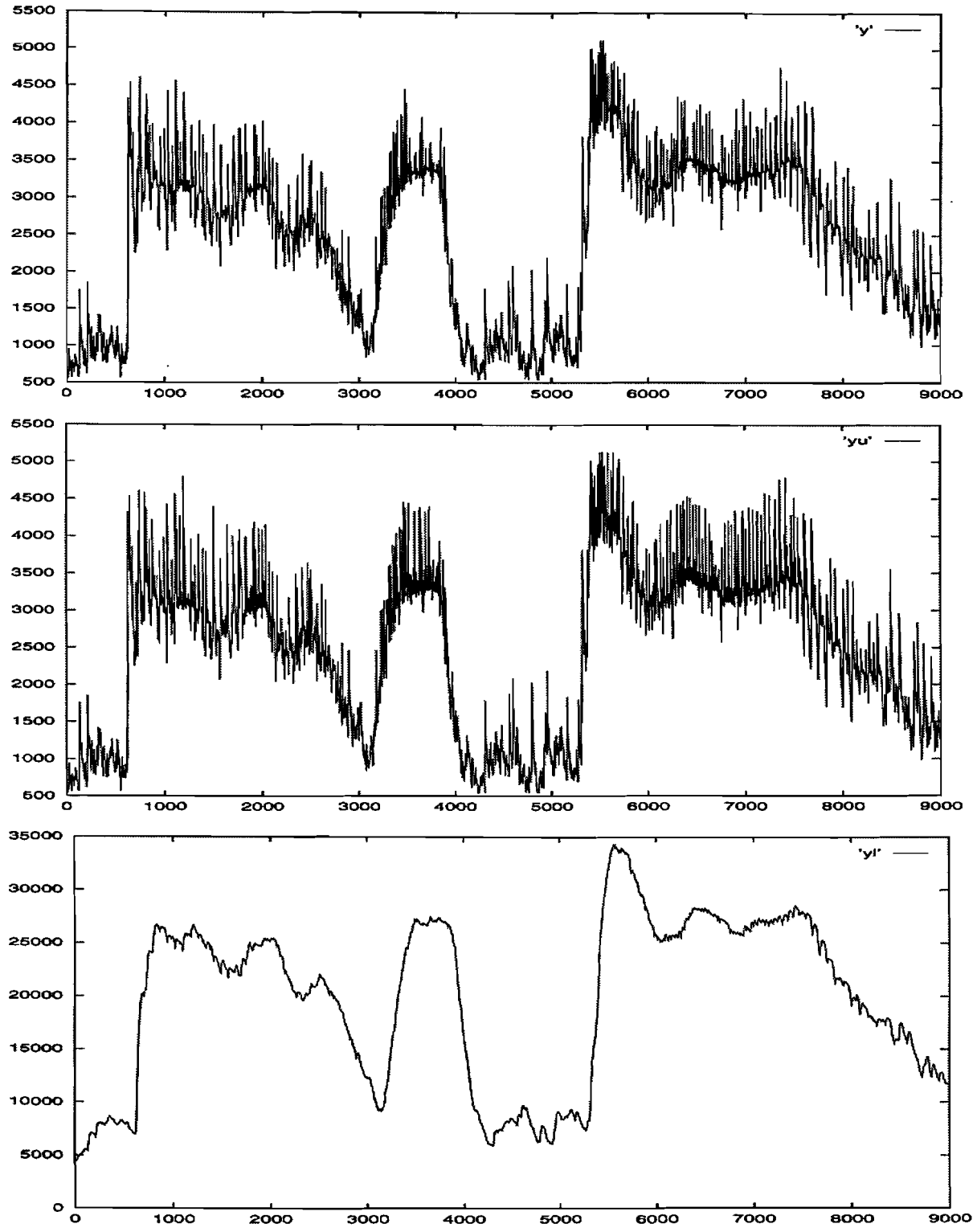
The major difference between the standard and the actual implementation is that both quantised difference signal2, d_q, and reconstructed signals, S_r, are not implemented in floating-point notation as was done in Recommendations G.721 and G.726. Calculations with such floating-point variables are not efficient in a fixed-point environment. The standard tells you to implement these variables with a mantissa of 6 bits and an exponent of 4 bits. Because values smaller than 1 are not likely to be of any importance an implementation with only fixed-point variables is even more accurate. Another difference is that the parameter, y_l, is not represented in 19 bits but rather in 16 bits (unsigned!). The resolution should be high enough, and it is obvious that the representation was chosen to be a 6-bits left-shift of the fixed-point compared y_u and y to facilitate an easy updating of $y(k)$.

APPENDIX E.1[1] Simulation results ADPCM

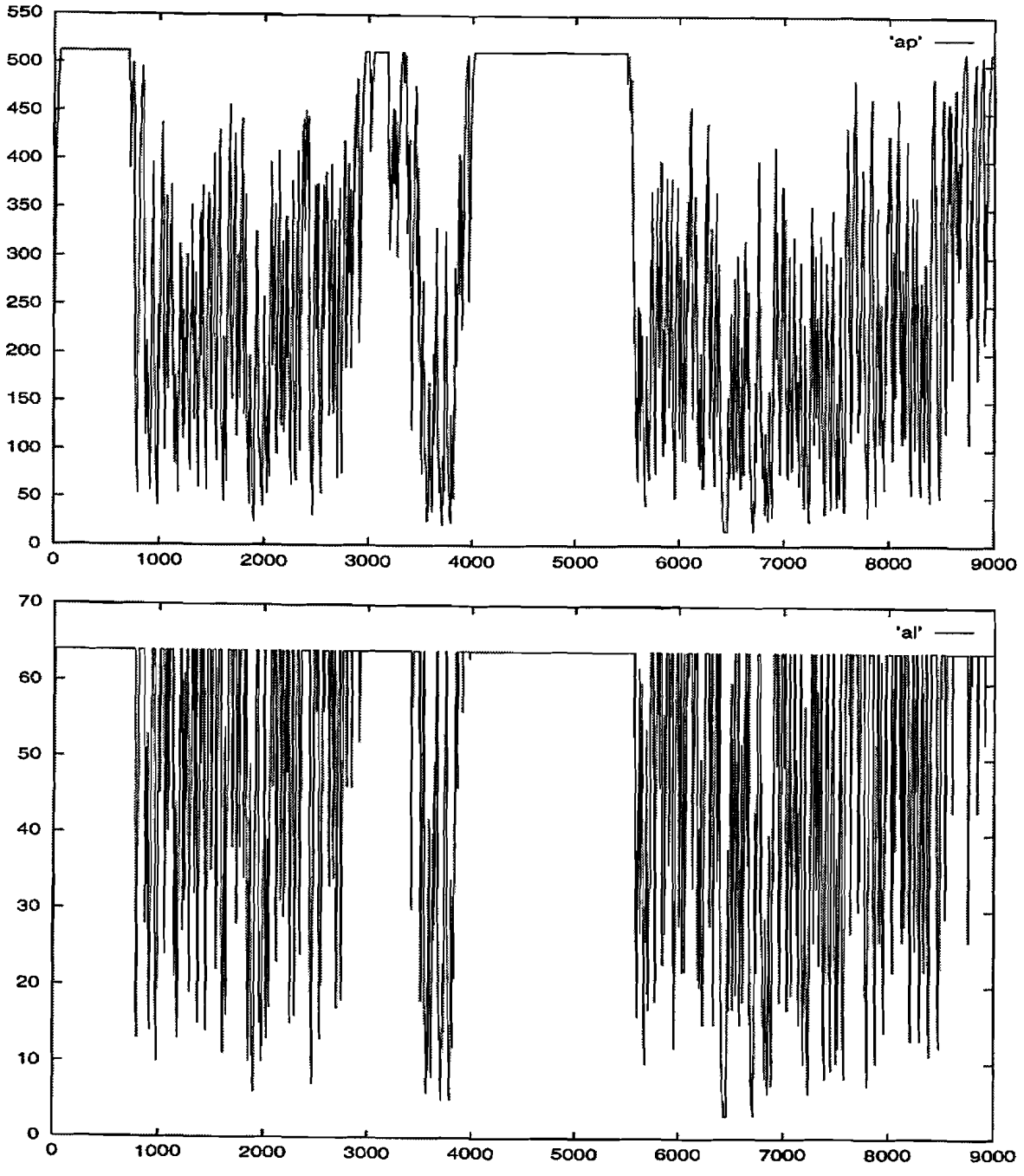


From the Figures above it is obvious that the signal estimate is close to the linear input signal. The equations of the variables that are shown above can be found in Formula 47a, 50a, and 50b, respectively.

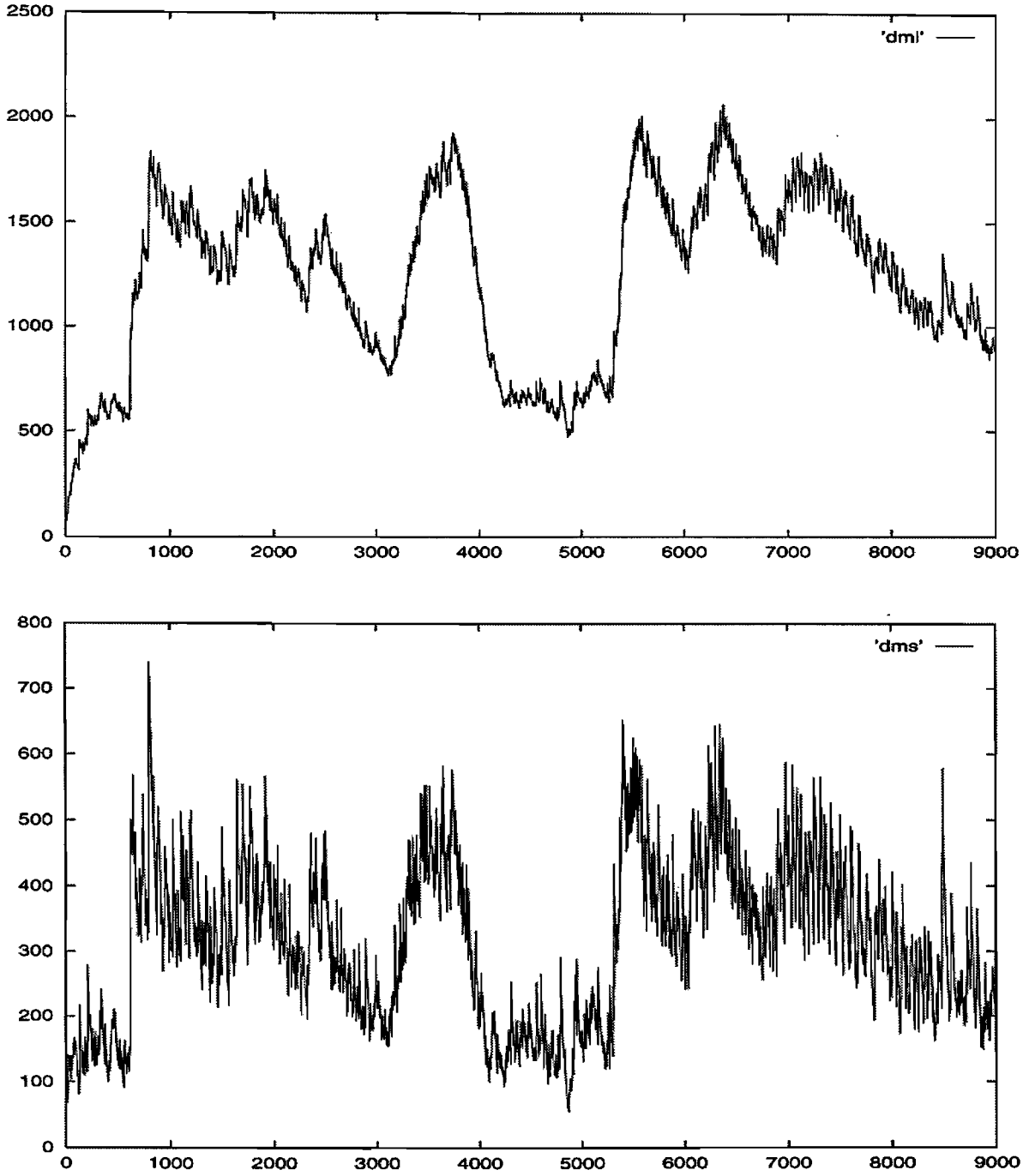
[1] The figures in Appendix E.1 are snapshots taken from performed simulations. The actual variable name can be found in the top-right corner of all Figures. The extension 'f' denotes floating-point variables, all other are fixed-point variables. The vertical axis is the amplitude of the signal, the horizontal axis shows the sample-moments.



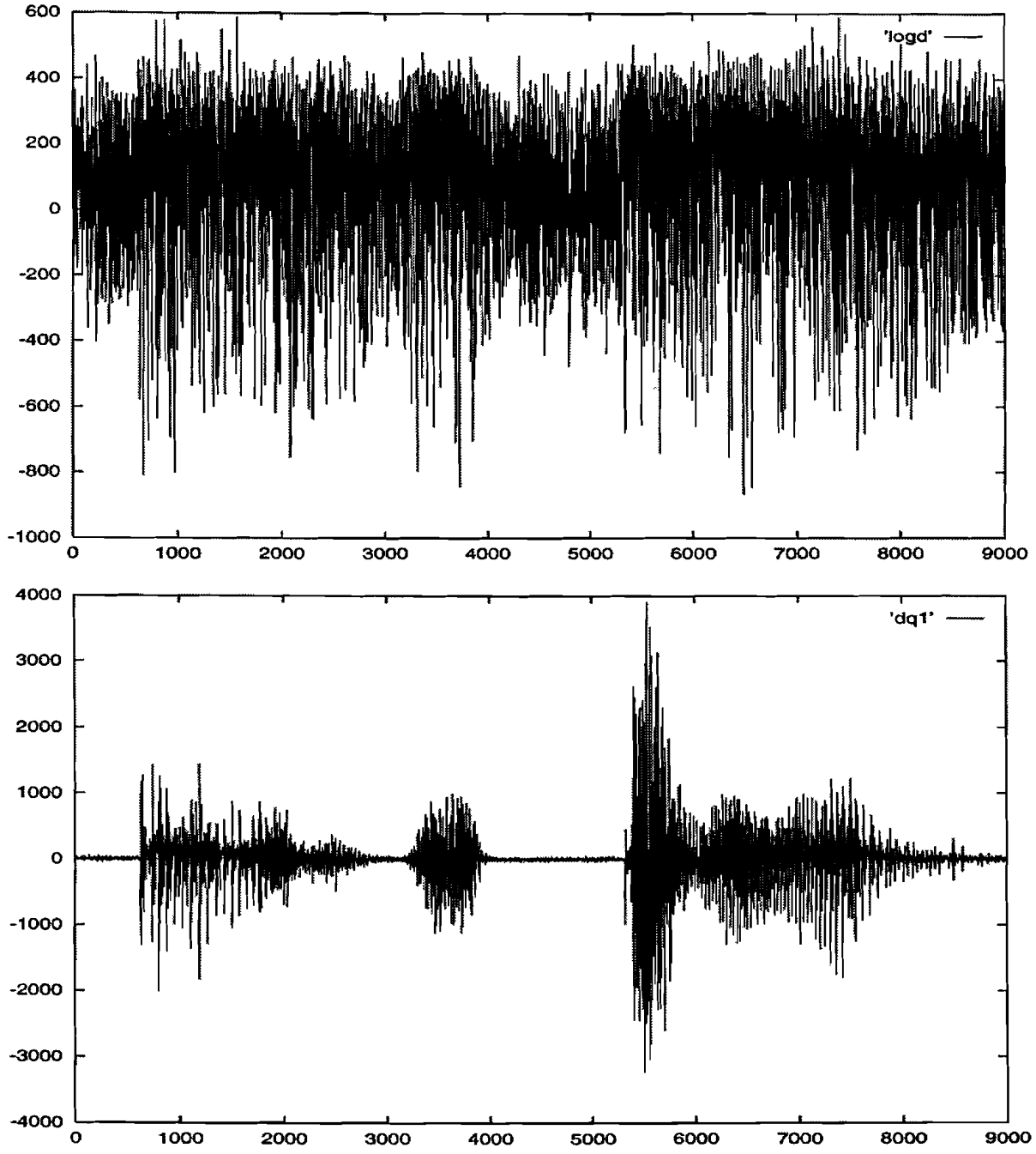
It becomes clear from the Figures above that the unlocked quantiser scale factor is adapted quasi-*instantaneous* whereas the locked quantiser scale factor adapts *syllabically*. The quantiser scale factor (top Figure) resembles most with the unlocked scale factor (y_u), which indicates that the input signal was speech. The equations of the variables that are shown above can be found in Formula 48a, 48b, and 48c, respectively.



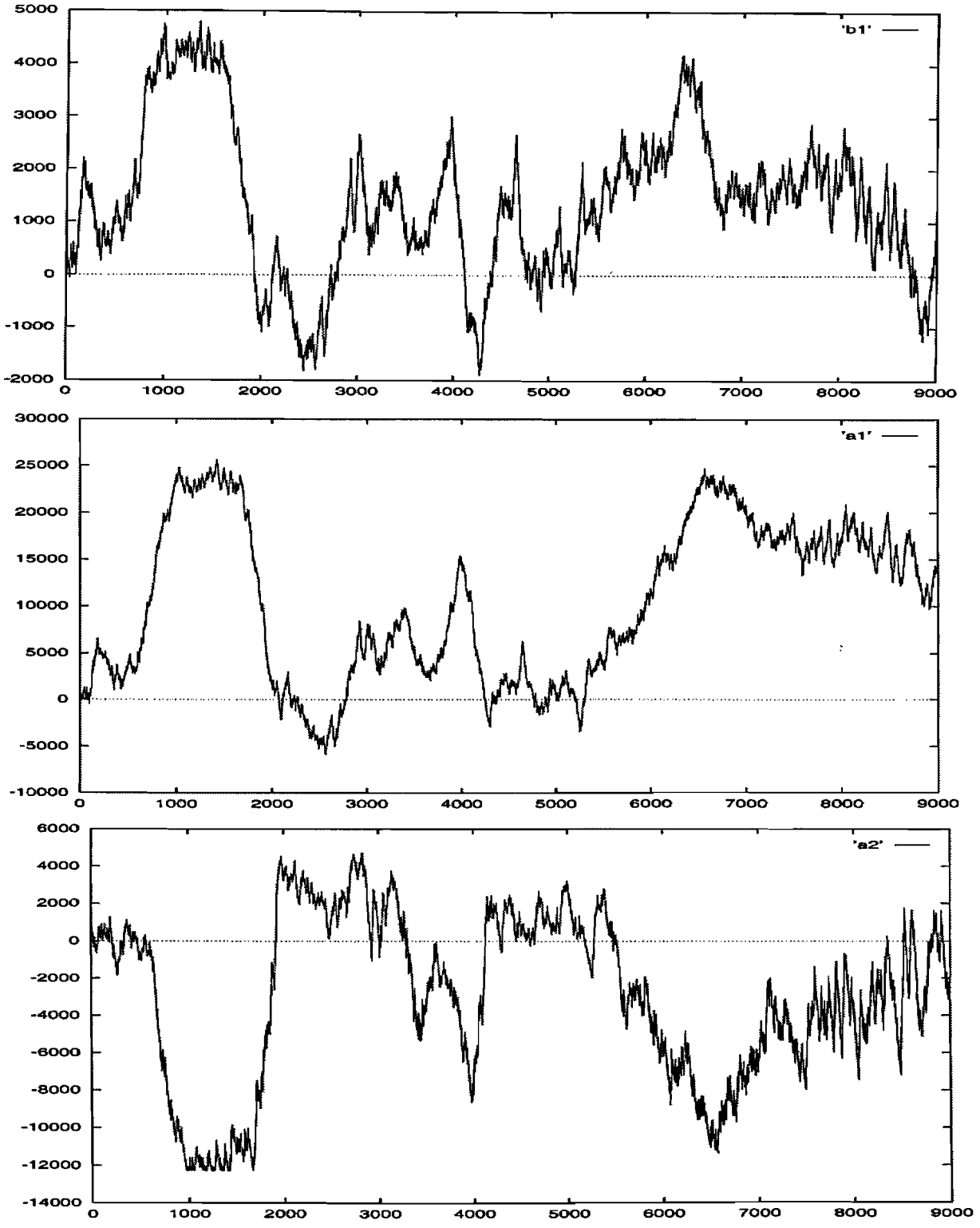
In the case of non-stationary input signals the speed control parameter tends towards unity, as was mentioned in Chapter 5. If the input signal is close to zero (noise) the speed control parameter is always unity, due to the fact that y(k)<3. The equations of the variables that are shown above can be found in Formula 49b and 49a, respectively.



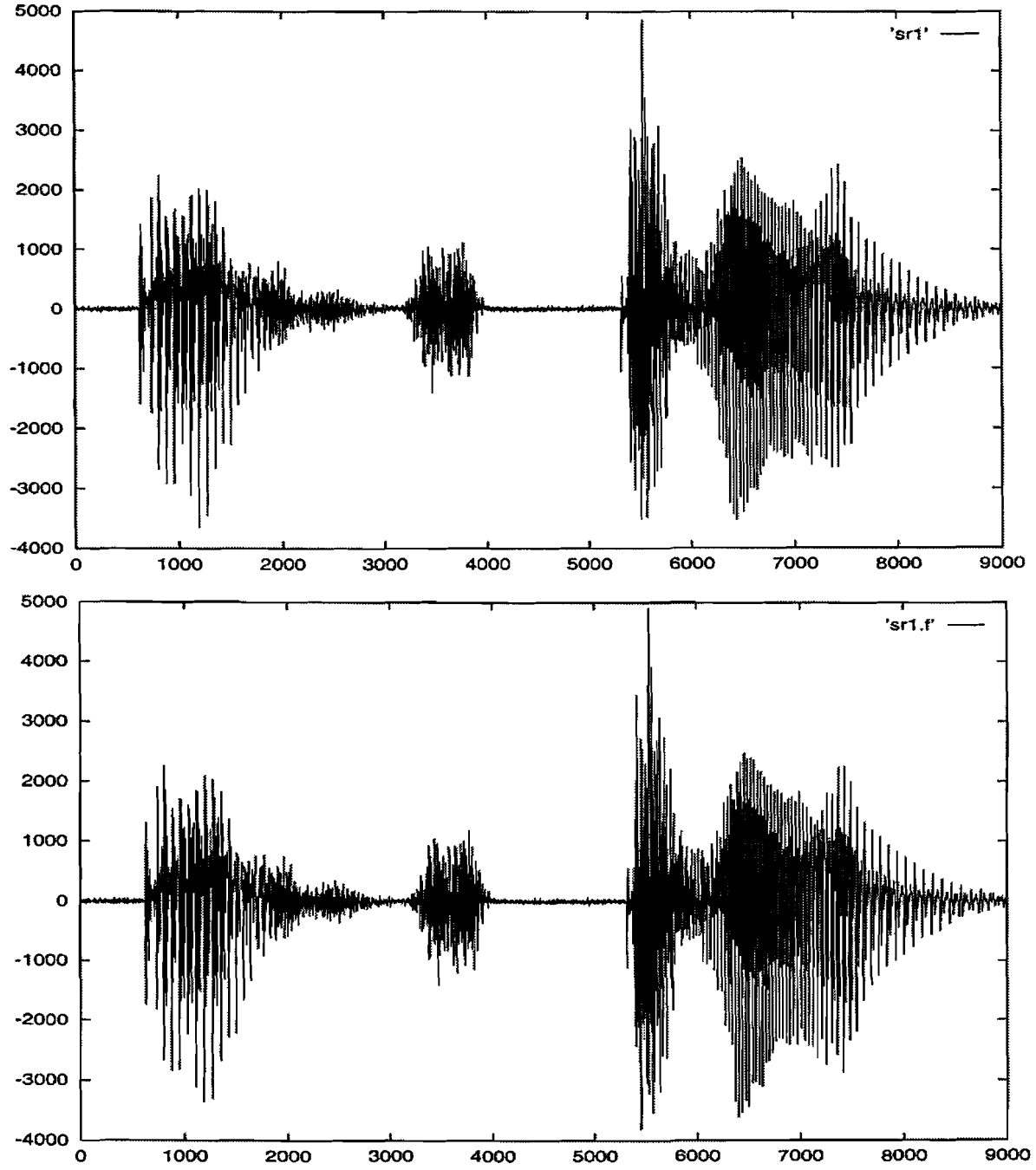
During voiced sounds of the input signal both short-term and long-term averages of the F(I) sequence increase. The short-term average adapts, off course, much more rapidly than the long-term average. The equations of the variables that are shown above can be found in Formula 49c and 49d, respectively.



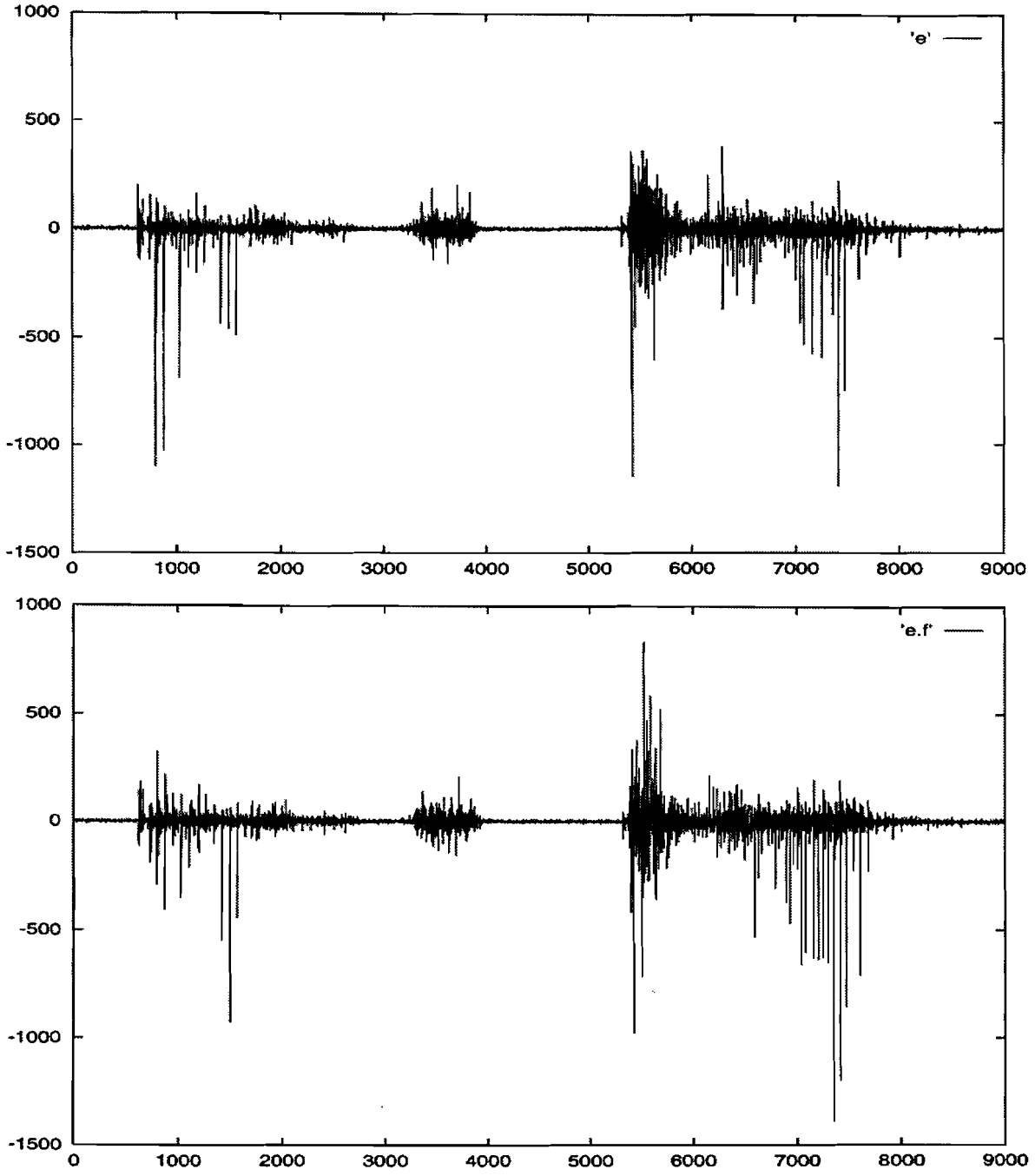
The data form the top figure is the input of the quantiser. therefore logd denotes the normalised difference signal. It appears to have no great correlation with the input signal. Clearly, most of this signal fits well within the range of the quantiser. This is surely the case for values greater than zero. For values smaller than zero still a fairly large amount of samples exist that will be totally erased. Every quantiser attempts to limit the number of cases overflow occurs, but here this is not the case. It may improve the coding quality if the step sizes would be chosen to be larger in order to account for the 'logd'-values that are subzero. Note, similar logd-outputs were observed with other speech material. The equations of the variables that are shown above can be found in Formula 47b and 49e respectively.



The adaptation speed of the filter coefficients, as shown above, is not very high. This indicates that the quantiser scale factor has a larger influence on the stability of the ADPCM codec than the adaptive filtering. The boundaries of the coefficients are only reached for a_2, not for the other variables. The equations of the variables that are shown above can be found in Formula 51c, 50d, and 50e, respectively.

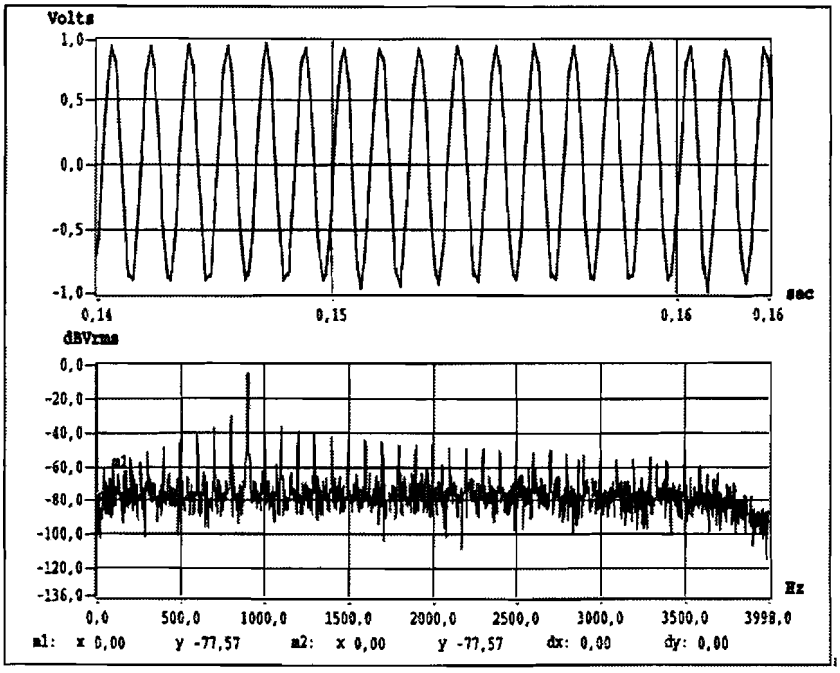


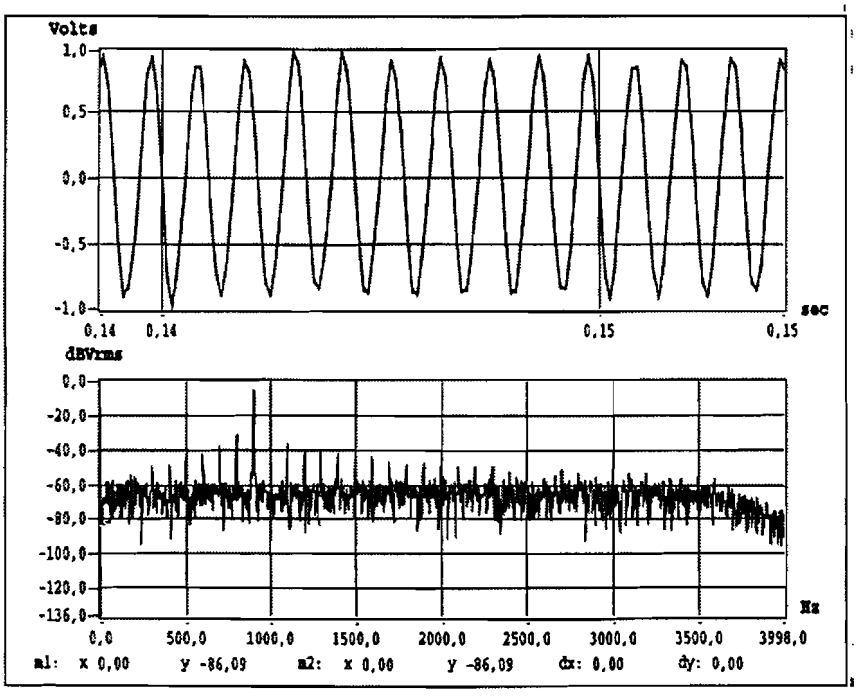
The plots of S_{r1} and $S_{r1}.f$ support the perceptual results from the simulations. The floating-point version of the codec does not produce an output that is much different than for the fixed-point version. Also two figures on the next page show this finding. The equation of the variable that is shown above can be found in Formula 50c.



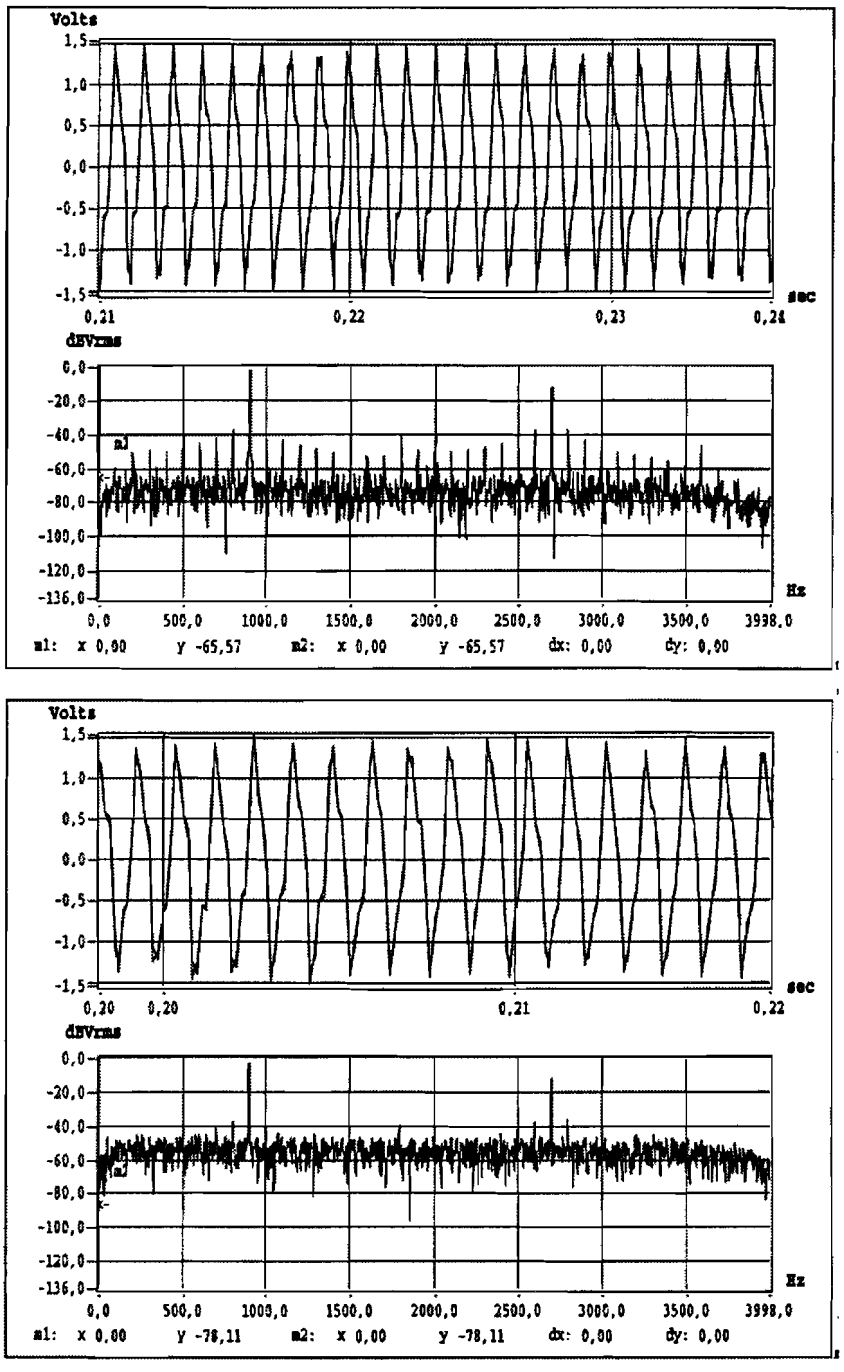
The plots above show the unrecoverable error that was caused by the coding algorithm. This error is the difference between the linear input, $S_l(k)$, and the reconstructed signal, $S_r(k)$. The error signal is correlated with the input signal and shows high peaks that are the result from the overflow of the quantiser range, as was mentioned before.

APPENDIX E.2 Laboratory results ADPCM





The top figure shows the input signal of the ADPCM codec, being a sine with an amplitude of 2V t-t and a frequency of 900Hz. The bottom figure shows the output of the codec. It is clear that the predominant frequencies are not distorted. Frequencies and amplitudes are almost the same. The "noise-floor" is somewhat higher than for the ADPCM output.



The top figure shows the input signal of the ADPCM codec, being a triangle with an amplitude of 3V t-t and a fundamental frequency of 900Hz. The lower figure shows the output of the codec. Also this example makes clear that the predominant frequencies are not distorted. Frequencies and amplitudes are almost the same, also the third harmonic. The "noise-floor" is higher for the ADPCM output. It seems that triangular input signal cause a bit more noise than sinusoidal input signals. This may be due to the fact that the AR(2)-portion is able to model sinusoidal s, but not noisy sinusoidals.