EVALUATION OF A POISSON WINDOWED TEST SIGNAL FOR PCM CODECS

Bernhard Cronje

A project report submitted to the Faculty of Engineering, University of the Witwatersrand, Johannesburg, in partial fulfilment of the requirements for the degree of Master of Science in Engineering

Johannesburg, 1988

DECLARATION

I declare that this project report is my own, unaided work. It is being submitted for the Degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg. It has not been submitted before for any degree or examination in any other University.

17th day of March 1989

ABSTRACT

This project deals with the critical evaluation of a Poisson windowed test signal which was proposed for in-service and production testing of filtered pulse code modulation systems. The Poisson windowed test signal has various advantages over the existing two test methods specified by the International Consultive Committee for Telephone and Telegraph (CCITT). The main goals of the new test signal are to minimise the test time and to accommodate the send and recieve filters normally found in communication systems.

The evaluation process included the compilation of performance curves of signal to noise ratio vs. signal level and gain variation vs. signal level. These curves were compiled by simulating the codec and filters on a digital computer and processing the output data. Three sets of programs were developed.

The results obtained revealed some fundamental limitations of the proposed test signal. Using the basic properties of the Poisson windowed signal, an alternative test signal was proposed. It was the single sided exponentially windowed sinusoidal which proved to be suitable for the testing of filtered pulse code modulation stems.

ACKNOWLEDGEMENTS

I would like to express my gratitude to the following people :

My Supervisor, Professor M.E.Hanrahan, for his advice and guidance.

My sister, Stephanie, for preparing the typed script in record time.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

a	Final value of exponential function as a fraction of vpeak
b	Exponential time constant
dBm0	dB scale with a reference level of 2840 units
d_E	Exponential signal
d_P	Poisson window
d_R	Rectangular window
D_E	Frequency spectrum of the exponential signal
D_R	Frequency spectrum of the rectangular window
DFT	Digital Fourier Transform
DUT	Decoder under test
EUT	Encoder under test
f_r	Signal frequency
FFT	Fast Fourier Transform
k	FFT component index
K	Encoder output code
m	Gain
m_E	Encoder side gain
m_D	Decoder side gain
N	Number of samples used in the FFT
PAM	Pulse amplitude modulation
PCM	Pulse code modulation
rms	Root mean square
r_n	Dither signal
SNR	Signal to noise ratio
tblk	Time duration of one block (32E-3 s)
tdur	Time duration of test signal

TCTC	Transposed codec test configuration
u	Decoder output
U_i	Decoder decision level
v	Encoder output
V_i	Encoder decision level
vpeak	Peak signal amplitude
w	Windowing function
x	Output of analogue process/input of EUT
a_n	FFT components
σ_{ND}^2	Decoder noise power
σ_{NE}^2	Encoder noise power
σ_S^2	Output signal power
σ_V^2	Input signal power

1 INTRODUCTION

1.1 Problem statement

The CCITT recommendations G.712, G.714 and O.131 (1984a-c) specify limits on the performance of PCM codecs. These limits are essential to ensure adequate subjective quality in PMC speech links. The CCITT specifies two methods to test codecs, both with their advantages and disadvantages.

The use of digital subscriber loops has introduced new constraints on test methods. Codecs and filters are now being applied on a per-line basis in large numbers. Remote testing of encoders and decoders is a necessity due to practical problems arising when considering that the devices to be tested are located on the subscribers premises. With in-sevice testing the separation of half channel performance parameters is desirable because of the fact that any encoder associated with any decoder, via the transmission and switching system, must be able to perform within the limits specified by the CCITT. During production testing short times are essential because testing accounts for a significant proportion of the total device cost.

The new test method has been proposed and tries to deal with these problems. It consists of the connection of the encoder and decoder in a back to back configuration and injecting a Poisson windowed sine wave. It is referred to as a one-shot measurement. The performance curves (signal to noise ratio vs. signal level and gain variation vs. signal level) are derived from these

measurements.

The purpose of this project is to critically examine and evaluate this new proposal. It involves a thorough simulation to see whether the new test signal can produce performance figures called for in the CCITT recommendations.

1.2 Background

The CCITT (Recommendation G.712, 1984a, Recommendation O.131, 1984c) recommends two methods for testing PCM codes, one using a pure sinusoidal test signal and the other using bandlimited Gaussian pseudo random noise. Both of these have disadvantages when considering production testing and in-service testing of codecs. Due to the large number of units involved and the cost of testing time during production, short test times are very desirable.

With Gaussian noise as a test signal, all decision levels are covered equally well and this makes it very appealing from a theoretical point of view. A disadvantage though is the large bandwidth involved because restrictive assumptions have to be made about the filter gain and phase. It is difficult to model filters and assumptions of flat magnitude gain and linear phase over the bandwidth concerned is too restrictive. This prevents practical comparison of actual and theoretical outputs.

Sinusoidal test signals are simple and well defined. Including filters does not present any problems because once the magnitude

and phase of the output have been estimated it is possible to calculate the theoretical output waveform. A *disadvantage* is the sensitivity of the *reference curves* of gain and SNR to signal level. Measurements are required at a number of input levels to overcome the problem of the non-uniform amplitude distribution of the sine wave. This in turn has the undesirable effect of increased measuring times and thus cost.

The various disadvantages of the above mentioned test signals led to alternative proposals. Due to the need to minimise the measuring time, methods that calculate the required curves from a set of measurements, rather than measuring them explicitly, are more appealing. Such measurements are *referred to as 'one-shot'* methods. The *main objectives of one-shot measurement techniques are* stated below.

1. The test signal must be short. (In the order of one to two seconds.)
2. Half channel measurements *should be possible.*
3. *Calibration* of the test system must be minimised. (Avoid analogue signal generation and measurement.)
4. The reference curves of the proposed method must be relatable to the CCITT reference curves.
5. The method must be suitable for *filtered codecs.*

The reason for (1) has been discussed and finds its roots in the cost of the time it takes to test a device during production. When considering the large number of digital subscriber stations

that are going to be installed in future. It is obvious that the in-service testing time must be kept as short as possible.

The migration of PMC codecs to the subscriber sets brings about a situation where a specific encoder can be connected to any arbitrary decoder. This explains the second objective of being able to make half channel tests. Both the encoder and decoder must be subjected to their own specifications or requirements to ensure adequate full channel performance for any arbitrary encoder-decoder pair. The CCITT recommendation G.714 (1984b) specifies limits on half channel performance parameters.

Conventional tests on encoders and decoders used analogue stimulation and measurement. This presented difficulties because the components of the test system required accurate calibration. By using standard analogue-to-digital and digital-to-analogue converters, digital stimulation and processing can be achieved. The standard converters are easily and accurately implemented in software.

The last objective (5) requires suitability for filtered codecs. In the new generation of integrated codecs, filters are included in the same package and the unfiltered signals are often inaccessible.

Another existing 'one-shot' measuring technique uses a stroboscopic effect on one cycle of a sine wave to measure the decision levels of the encoder and the decoder output levels. The

result is then used to calculate the required curves of the gain and SNR to signal level (Cordt, Hahn, 1982)

The proposed test signal that will be the subject of this project is a Poisson windowed sine wave with the equation :

$$v(t) = v_{peak}.\exp(-b|t|).\sin(2\pi f_c t) \qquad -t_{dur}/2 <= t < t_{dur}/2$$
$$= 0 \qquad \text{otherwise.}$$

The duration of the signal (tdur) will be in the order of one to two seconds. A thorough computer simulation will be executed and a set of reference curves have to be determined. As mentioned these should be capable of being related to the two CCITT methods.

The configuration that is going to be used in this project is called the transposed codec test configuration. It consists of the encoder-under-test (EUT) and decoder-under-test (DUT), connected up in a back to back configuration. Stimulation and measurement is digital which is a very desirable feature. This test configuration and its theory will be discussed in section 1.3.4.

1.3 Review of the theory

1.3.1 Introduction

This chapter will cover all the relevant theory for the testing of codecs. The two principal methods for performing tests on

codecs will be discussed. The transposed codec test configuration (TCTC) and its theory will be covered and finally some of the properties of the new Poisson windowed test will be highlighted. This information is needed to successfully analyse and interpret measured results.

1.3.2 Conventional tests

A coventional test configuration is shown in figure 1.1 below.

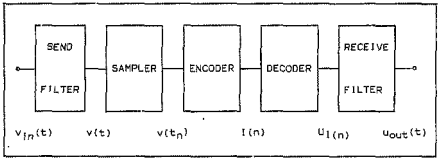

Figure 1.1 Conventional PCM test connection

It consists of an encoder, preceded by a send filter and sample and hold circuit. The PCM output of the encoder is then connected to the decoder which produces a quantised pulse amplitude modulated (PAM) signal. The receive filter would then remove the amplitude discontinuities from the decoder output and produce a waveform that closely resembles the input waveform.

The input waveform is a band limited signal with a specific probability density function which will either be for a Gaussian or a sinusoidal test signal. Specifications for both of these

waveforms are given by the CCITT (1984c). The signal is sampled at a rate which must be high enough to satisfy the Nyquist Criterion. The encoder will then produce a code I(n) for a specific sample v(t), based on the decision of whether the sample is lying between decision levels $V_{I(n)}$ and $V_{I(n)+1}$. The suffix n is added to identify a specific sample and to follow it through the chain without accounting for the exact time delays introduced by the encoder and the decoder.

The CCITT recommendation G711 (1984d) defines A and μ-law characteristics in terms of the decision- and decoder output levels. The relationship between the N+1 encoder decision levels, V_I and the N decoder output levels U_I is given for an ideal encoder-decoder pair as

$$U_I = (V_I + V_{I+1})/2 \qquad I = 1, 2, 3, \ldots, N$$

In the beginning only full channel tests were performed. The test setup looked like the one in figure 1.1 above and analogue stimulation and measurement was used. Later, with the introduction of switching between different encoders and decoders, the necessity for performing half channel tests arose. CCITT recommendation G714 (1984b) specifies limits on the half channel performance. Signal to noise ratio (SNR) and gain are the principle parameters measured. In both of the above mentioned cases filters were used to separate the signal and noise components.

1.3.3 Measurement techniques

There are a number of test methodologies and seven are identified by Hanrahan (1984b). The two principal methods that exist are :

1) Frequency domain separation.
2) Linear regression based analysis.

The first uses filters to separate the signal and noise components. These filters are either implemented physically or via a Discrete Fourier Transform (DTF). The latter method is normally used during digital stimulation and measurement. In both cases the signal and noise powers are measured in their respective bands. By assuming that the noise spectrum is flat the total signal to noise power ratio can be found. The gain is found by comparing the input signal power to the output signal power.

The second method is not used very often in practical situations because it is more difficult to implement. It is however a more fundamental approach and it is free of assumptions such as the flatness of the noise spectrum. It also allows the signal to occupy the entire audio band but still allows sufficient separation of the signal and noise components on the output. This method uses the variances of the input and output respectively and the covariance between them to calculate the required curves. The theory of both these methods is discussed in detail by Hanrahan (1984a) and in this report only the theory relevant to the transposed test connection will be discussed briefly.

Regardless of the technique or the type of test signals used, the performance of a given encoder-decoder pair is assessed in terms of the departure from ideal values. Graphs of gain and signal to noise ratio are obtained for ideal encoder-decoder pairs and are termed reference curves. Performance evaluation of non-ideal pairs is done with these reference curves as a basis. Each specific test method will have its own reference curves and these must be relatable to the CCITT specifications.

Up to now most of the work and tests done on codecs excluded the filters. With the emergence of combinational codec-filter units, it has become impossible to separate the filters from the other components. With filters present the observed output signal and noise powers are influenced by the filter transfer functions. Some corrections have to be made to the reference curves of gain and SNR.

1.3.4 Transposed test configuration

The restrictions of analogue measuring systems have been mentioned. In addition to this, practical difficulties arise when considering that the analogue inputs and outputs of the codecs are located on the subscribers premises. Access is obviously a major problem. These problems can be overcome by digital stimulation and measurement. The transposed test configuration, where the encoder-under-test (EUT) and the decoder-under-test (DUT) are connected in a back to back configuration is shown in figure 1.2.

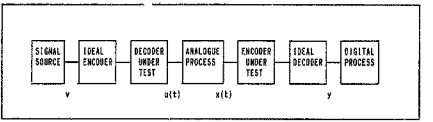

Figure 1.2 Transposed test configuration

The DUT is stimulated by a digital sequence of samples generated by a software implemented ideal encoder and signal source. The output of the DUT is then fed to the input of the EUT after some analogue process has been performed on it. Reasons for this process will be discussed in chapter 1.3.5. The encoder output is a digital sequence of encoded samples and is fed into a software implemented ideal decoder and measuring device. The configuration can also easily be implemented in an in-service test situation. See figure 1.3. Stimulation and measurement is done via the PCM subscriber loop as indicated. Low error rates on these channels and the fact that digital signals are used, makes this remote testing configuration feasible.

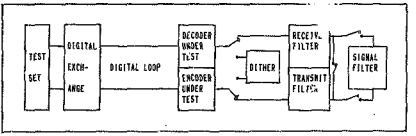

Figure 1.3 In-service test connection

Closer inspection reveals that the transposed test configuration is built up of two half channel test systems, which are the decoder-side half-channel test pair and the encoder-side half-channel test pair. These two halves are connected via the analogue signal processing or modification unit. This unit is necessary to ensure that the entire encoder input range is probed and that any decoder imperfections are reflected in the output of the encoder. The various operations on the analogue signal will be described in a later section.

The CCITT recommendations allow both sinusoidal and pseudo-random noise signals to be used as test signals. Sinusoidals are easy to generate and are therefore preferred in existing tests. It is difficult to generate analogue pseudo-random noise with good amplitude and spectral properties. With digital stimulation more effort can be expended to obtain input sequences with near-ideal properties. Once these have been generated, tested and found suitable, it is easy to store them as a sequence of numbers in memory. Regeneration of an exact test waveform is thus simplified

and is without the uncertainties normally associated with analogue test signals.

1.3.5 Analogue signal modifications

As mentioned, some type of analogue operation is needed to 'smooth' out the PAM output of the DUT. The encoder input signal must have a continuous probability density function and it must exercise the full range of amplitudes. When considering the combined effect of the EUT and the DUT a situation can arise where small variations in the decoder output levels and the encoder decision levels can produce no change in the output code. Gain and distortion contributions due to these small errors can thus go undetected if the two halves are connected directly. Four possible operations on u(t) which give a more continuous amplitude distribution in x(t) are given by Hanrahan (1984b):

1. Analogue filtering
2. Addition of a dither signal
3. Programmed attenuation
4. Dither signals that adapt to the input amplitude.

Analogue filters ensure that the encoder input has a continuous amplitude distribution. Errors in encoder decision levels along with decoder errors from the first half of the channel will now produce an incorrect output code, K. With the filters included there is no need for dithering. The group delay and amplitude distortion must however be taken into account when calculating the required curves.

Addition of a dither signal, which is uncorrelated to the input signal, is another way of achieving the *desired effect*. This *signal must* be known exactly and is added to the input signal before encoding. An identical signal is removed from the output analogue signal after encoding. One of the requirements for the dither signal is that its peak to *peak value must* be at least *equal* to the width of the highest quantising interval. Selection of the dither signal is also dependent on the type of test signal used. The selection of a triangular dither waveform with the right frequency is described by *Hanrahan (1984b)*.

A variation on the above mentioned additive dither signal is interval-related dither. The peak to peak value of the dither signal is varied in accordance *with the input* signal. This is a very attractive method but the complexity of the equipment needed to implement this is high.

An alternative method *described by Hanrahan* (1984b,1984c), is dithering by means of a programmable attenuator/gain unit installed in the analogue part of the loop. This is switched in a sequence uncorrelated to the test signal, according to

$$x(t) = u(t)(1+r_n)$$

where r_n is a sampled function uniformly distributed about zero. *The optimum number* of attenuator steps and their spacing were found by simulation (Hanrahan, 1984b,1984c). The range of attenuations $\pm r_n$ should spread the pulse heights by at least one

quantising interval on either side of the decoder output. Uniformly spaced steps of at least 32 but preferably 64 were found to be satisfactory. All previous equations for obtaining the gain and SNR hold, provided the input and the dither signal are uncorrelated.

1.3.6 Theory of the transposed connection

The performance of the decoder side half-channel is quantified in terms of the gain m_D and the output noise power σ_{ND}^2 for a given input signal power σ_V^2. Similarly the encoder half will have gain m_E and noise power σ_{NE}^2. The output of the DUT is given by

$$\sigma_U^2 = m_D^2\sigma_V^2 + \sigma_{ND}^2 \qquad (1)$$

which is the sum of the signal power and the noise power. The input signal to the EUT is x(t) and is derived from u(t). The digital output of the ideal decoder is y_n and has signal and noise components

$$\sigma_Y^2 = m_E^2\sigma_X^2 + \sigma_{NE} \qquad (2)$$

For the simplest case, where the filters are excluded x(t) = u(t) and thus

$$\sigma_Y^2 = m_E^2m_D^2\sigma_V^2 + m_E^2\sigma_{ND}^2 + \sigma_{NE}^2 \qquad (3)$$

We can now define the overall gain

$$m = m_E m_D \tag{4}$$

and the total noise power as

$$\sigma_{ND}^2 = m_E^2 \sigma_{ND}^2 + \sigma_{NE}^2 \tag{5}$$

The gain m is measured in terms of the ratio of output and input signal powers

$$m^2 = \sigma_S^2 / \sigma_V^2$$

where σ_S^2 is the output signal power i.e. the component of y_n that can be correlated with the input v_n. Examples of reference curves obtained for ideal encoder-decoder pairs are given in figure 1.4.

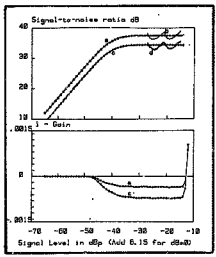

Figure 1.4 Examples of reference curves (Hanrahan, 1984b)

The parameters needed to calculate these curves are obtained by using the Fast Fourier Transform (FFT). The method described here is particularly suited to testing with sinusoidal signals and can be extended to other test signals as well. The number of samples, N, should be larger or equal to 256 (Debbo,1984). The signal frequency, f_r, is chosen to coincide with one of the spectral lines of the FFT i.e. $f_r = K\Delta f$, where Δf is the spacing of the frequencies. Signal energies at other frequencies than f_r are negligible if this condition is met. A diagram illustrating the spreading of the spectrum is shown in figure 1.5.

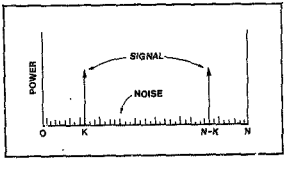

Figure 1.5 Spreading of signal and noise power

The signal and noise powers are given by:

$$\sigma_S^2 = |Y_k|^2 + |Y_{N-K}|^2 \qquad (6)$$

$$\sigma_{NO}^2 = \Sigma_k |Y_k|^2 \qquad k=1,\ldots,k-1,k+1,\ldots,N-K-1,N-K+1,\ldots,N-1 \qquad (7)$$

The summation is not over all the values as indicated. It neglects the noise component at the signal frequency (components K and N-K) and omits the DC term. The SNR is $10\log_{10}.\sigma_S^2/\sigma_{ND}^2)$

and the gain is found from the ratio $m_E^2 m_D^2 = \sigma_S^2/\sigma_V^2$. By modifying equations (6) and (7), separation of signal and noise powers can be achieved for the Gaussian test method as well.

1.3.7 New test signal

The various advantages and disadvantages of the two test signals, as specified by CCITT, have been discussed. This led to the proposal of the new Poisson windowed sine wave as a test signal. Examination of the spectral properties of the signal reveals some attractive features which will have to be examined thoroughly in the computer simulation. The test signal is given by:

$$v(t) = vpeak.\exp(-b|t|).\sin(2\pi F_r t) \qquad -tdur/2 <= t < tdur/2$$
$$= 0 \qquad \text{otherwise}$$

It is a sine wave with peak value 'vpeak' and it is windowed by the double sided exponential and a rectangular window. The equation for the window is thus given by :

$$d_P(t) = d_E(t).d_R(t)$$

where

$$d_E(t) = e^{(-b|t|)}$$

and

$$d_R(t) = 1 \qquad -tdur/2 <= t <= tdur/2$$
$$= 0 \qquad \text{otherwise}$$

Another parameter, a, is introduced to conveniently describe the Poisson window. a is the final value as indicated in figure 1.6.

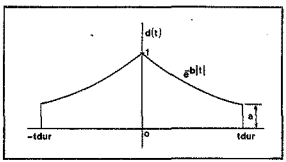

Figure 1.6 Poisson window

Thus, at t = tdur/2

$$a = \exp(-b.tdur/2)$$

$$\text{or } b = -(2/tdur).\ln(a)$$

Two of the parameters a, b and tdur fully define the window.

Examination of the spectrum of the waveform is necessary to get an idea of the bandwidth required. The Fourier Transform of equation (2) will reveal this information.

$$d_P(t) = d_E(t)d_R(t)$$

$$D(f) = D_E * D_R(f)$$

where

$$D_E(f) = F\{d_E(t)\} = (2/b) \;/\; (1 + (2\pi f/b)^2)$$

$$D_R(f) = F\{d_R(f)\} = tdur.Sa(\pi F.tdur)$$

Figures 1.7 and 1.8 below show these functions and the order of the magnitudes of the bandwidth associated with each can be seen.

The bandwidth of the rectangular window is in the order of 1/tdur.

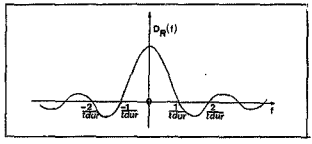

Figure 1.7 Power spectrum of rectangular window

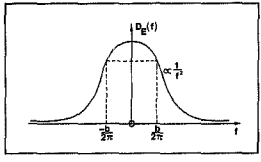

Figure 1.8 Power spectrum of the exponential function

The bandwidth of $D_E(f)$ is a function of b, which can be specified in terms of a and tdur as follows

$f_{6dB} = b/2\pi$

$f_{6dB} = [-2\ln(a)]/[2\pi tdur]$

A table with typical values for a and tdur shows that the 6dB frequency falls between *1/tdur and 2/tdur*, see table 1. Vm is used to denote the final amplitude value in number of units and

is equal to a * vpeak.

Vm	a	tdur(s	f_{6dB} Hz
8	0.0020	1	1.99
16	0.0039	1	1.77
32	0.0078	1	1.54
64	0.0156	1	1.32
8	0.0020	2	0.99
16	0.0039	2	0.88
32	0.0078	2	0.77
64	0.0156	2	0.66

Table 1.1 Typical 6dB bandwidths with vpeak = 4096

As the first null of the rectangular window is at 1/tdur we can see that this test signal achieves a bandwidth in the order of a few Hz. The value of the spectrum falls off according to $1/f^2$.

The FFT on the total test signal is performed in a series of blocks as shown in figure 1.9 below.

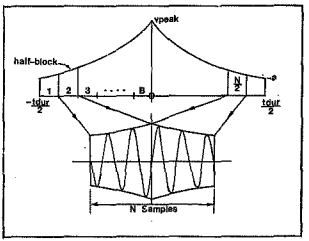

Figure 1.9 Subdivision of test signal

If N is the number of points used in the FFT, the test signal is divided into 2B blocks, each having N/2 points. Each half-block on the one side of the test signal is then joined to its corresponding half-block on the other side of the signal. The resulting blocks, with N samples each, are FFT'ed and the gain and SNR's are calculated for this signal level. It will now be shown that processing of the shorter blocks does not change the bandwidth, which is also the reason for performing this joining process.

If the signal frequency is chosen such that there is an integer number of cycles per block, the effect of the rectangular window will be hidden and all the signal power will be confined to one spectral line. This is due to the fact that the frequency line separation is equal to the distance between the nulls of the rectangular window. Convolving this single spectral line with the frequency spectrum of the exponential function, merely gives a replica of the exponential spectrum at the signal frequency. The line of reasoning is illustrated in figure 1.10.

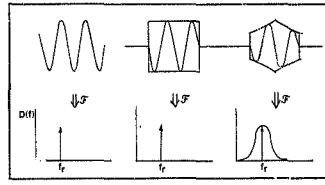

Figure 1.10 Illustration of bandwidth conservation

This is a very favorable result because separate processing of each block gives a set of values for gain and SNR of increasing signal levels without a change in bandwidth.

Calculations of the gain and SNR are made as described under the theory for the transposed test connection in section 1.3.5. The narrow bandwidth occupied by the signal allows us to neglect the noise components within that band.

The following chapters will describe the method used to subject the test signal to practical testing situations as well as the results obtained. The development of an alternative test signal is also described.

2 TEST SYSTEM LAYOUT

2.1 Introduction

A set of programs were developed for the evaluation of the proposed test signal. The programming language used was Turbo Pascal. Although some other languages are much more time efficient, Turbo Pascal allowed the quick development of a number of flexible modules.

Due to fairly long running times, the programs used were split into three main sections namely simulation, processing and plotting. These shown in figure 2.1 as well as the way they interact.

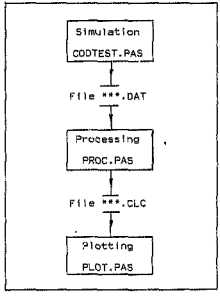

Figure 2.1 Program subdivision

The first section simulates the physical test setup consisting of the encoders, decoders and filters. The output data of this section is stored in a file on disk, and is then processed by the second section. The results are stored in another file, which is read by the plotting program for the plotting of the SNR and gain graphs. Each of these sections will now be described briefly.

2.2 Simulation

Three of these programs exist viz. CODTESTS.PAS, CODTESTP.PAS and CODTESTB.PAS. The only difference between them lies in the test signals that they use. The first is for a pure sinusoidal test signal and will be described in chapter 2.5. The second is for the proposed Poisson windowed test signal and the third is for the single sided exponentially windowed signal, which originated later on. The main modules of the simulation package are shown in figure 2.2 below.

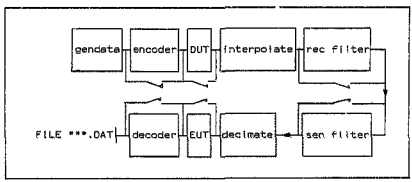

Figure 2.2 Main modules of the simulation program

The switches in the figure indicate the option of including or

excluding specific modules during the simulation. Each one of the modules will be discussed in detail in chapter 5 and program listings will be given in the appendices.

Due to the restrictions of Turbo Pascal on the total size of data variables, the input signal was divided into a number of blocks, each containing 256 samples. The blocks are sent through the sequence one by one and are stored in one array, called 'out_dta', at the end. After all the blocks have passed through, 'out_dta' is dumped onto disk in a file with a relevant name and the extension .DAT. The name gives information regarding the test signal used as well as the modules which were included in the simulation (eg. filtered versus unfiltered codecs).

2.3 Processing

Different processing programs exist, having the names PROC***.PAS. The '***' suffix is used to indicate the type of test signal used and the number of output points generated. A block diagram of a processing program is shown in figure 2.3.

The input data for the simulation is regenerated by the procedure 'gen_data' and the simulation output is obtained from the ***.DAT file by the procedure called 'get_data'. Processing is done block by block (256 samples per block). The procedure 'calculate' calculates the SNR from the FFT data and the gain by taking the ratio of the regenerated input signal level and the output signal level. This information is stored in a file called ***.CLC.

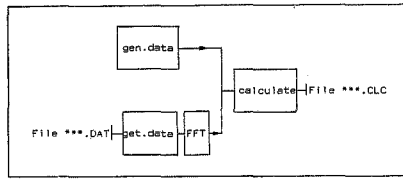

Figure 2.3 Main modules of the processing program

2.4 Plotting

The plotting programs, called PLOT***.PAS, each has a suffix indicating the number of data points used (eg. PLOT99, PLOT785 etc.). The data is read from a ***.CLC file and is then passed to two procedures viz. 'plot_snr' and 'plot_gain', which plot graphs of SNR versus signal level and gain variation versus signal level respectively. The high resolution graphics mode (640 X 200 pixels) of Turbo Pascal was used for plotting the graphs.

2.5 Reference curves

Before the proposed test signal could be evaluated, the *generation of reference curves for a standard* test signal was necessary. Assessment of the new signal could then be done by comparing the different reference curves. The sinusoidal test method as specified by the CCITT, (CCITT Recommendation G.712, 1984a), was used. This method uses a pure sinusoidal test signal at all possible input signal levels.

A known property and disadvantage of the reference curves for pure sinusoidal test signals is its severe sensitivity for signal level. This sensitivity to signal level is caused by the non-uniform amplitude distribution of the sinusoidal test signal. The resolution of the graphs have to be fairly high (± 0,1 dBm0), in order to get the exact shape of the curves.

The graphs for the SNR and gain variation for an ideal encoder-decoder pair, obtained by the simulation program CODTESTS.PAS, are shown in figures 2.4 and 2.5. Psophometric weighing for the SNR was done according to CCITT Recommendation G.223 (1984e).

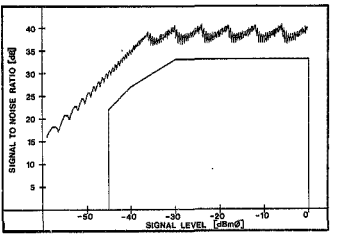

Figure 2.4 SNR graph for an ideal encoder-decoder pair using a sinusoidal test signal

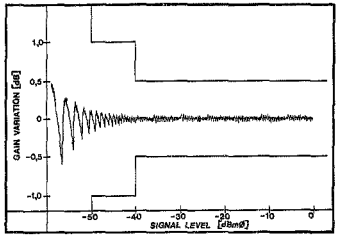

Figure 2.5 Gain variation for an ideal encoder-decoder pair using a sinusoidal test signal .

The CCITT masks for sinusoidal test signals, as specified in CCITT Recommendation G.712 (1984a) were included in each one of the figures.

Now that an overview has been given of the test strategy and reference curves have been obtained, the detailed study and the evaluation of the proposed Poisson windowed signal can be described. This will be the topic of chapter 3.

3 THE POISSON WINDOWED SIGNAL

3.1 Signal parameters and properties

Most of the properties of the Poisson windowed test signal making it attractive have been discussed in chapter 1. The most favorable one was the possibility to process the signal in a blockwise format without any increase in bandwidth.

In order to use the signal in a physical test setup, a zero-shift had to be introduced and values had to be assigned to all the variables in the equation. The signal became :

$$v(t) = vpeak.exp(-b|t-tdur/2|).sin(2\pi f_r(t-tdur/2))$$

$$0 <= t < tdur$$

The signal is shown in figure 3.1.

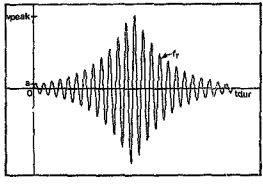

Figure 3.1 Test signal as used in program

The values of a and vpeak were chosen such that the whole signal range was covered (-60 dBm0 to 0 dBm0). This corresponded to values vpeak = 4096 units and $a = 0,001$. (The dBm0 scale uses the

value 2840 units as reference level.)

The number of points used in the FFT influence the measurement accuracy. Values of N = 256 or greater have been found to be satisfactory by Debbo (1984).

According to CCITT specifications G712 (1984a), the signal frequency f_r must lie in the frequency range 700 - 1100 Hz (excluding submultiples of the 8 kHz sampling frequency). The use of the FFT for processing the data, implied another restriction on the signal frequency. Debbo (1984) showed that when using a 256 point FFT,the signal frequency has to be an integer multiple of Δf = 31,25 Hz. This is to avoid leakage of the signal component to other frequencies without the use of complex windows. A test signal frequency of 1031,25 Hz was chosen.

With the number of points per block as well as the signal frequency fixed, the test time per block is fixed as well. The parameter 'tblk' was used to denote the time duration of one block and is equal to

$$\begin{aligned} tblk &= 256 * (1/f_r) \\ &= 256 * (1/1031{,}25) \\ &= 32 \text{ ms} \end{aligned}$$

The duration of the whole test signal, tdur, will thus only depend on the number of blocks it consists of. Table 3.1 shows different test times in the order of one to two seconds.

B	tdur
30	0,96 s
40	1,28 s
50	1,60 s
60	2,24 s

Table 3.1 Test time as a function of the number of blocks

A test time of 1,6 s, corresponding to 50 blocks, was chosen. The 50 blocks used during processing were each made up of half a block, coming from either side of the test signal. This was described in chapter 1.3.7.

3.2 Test setup

The spectral properties of the Poisson windowed signal were discussed in chapter 1 and some tests had to be performed to determine the maximum possible separation of the signal and the noise components. This was achieved by doing a FFT on the test signal in its pure form and examining the SNR. See figure 3.2.

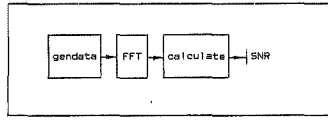

Figure 3.2. Determining the maximum SNR.

The FFT results showed that most of the signal energy was confined to the frequency line f_r plus one line on either side. These three spectral lines were added to get the signal component and all the remaining lines (except 0 Hz) were added to get the noise component. The resulting SNR for the pure signal was :

$$SNR \approx 46{,}3 \text{ dB}.$$

When looking at the overall SNR of two separate devices in series (A and B), it can be shown that this value is too low for accurate measurement. Let device A be an ideal encoder-decoder pair with an average SNR of :

$$SNR_A = 20 \log(S/N_A) = 37 \text{ dB}$$
$$\text{or } N_A = S \times 14{,}13 \times 10^{-3}$$

Device B is the measuring device with a maximum signal-to-noise separation of :

$$SNR_B = 20 \log(S/N_B) = 46{,}3 \text{ dB}$$
$$\text{or } N_B = S \times 4{,}84 \times 10^{-3}$$

The measured SNR of the ideal encoder-decoder will be the combined SNR, given by :

$$SNR_T = 20 \log(S/(N_A + N_B))$$
$$= 34{,}4 \text{ dB}$$

It is obvious that the margin for error in the measurement is not large enough. It could thus be said that accurate separation of signal and noise, using this method, proved to be impossible due to the specific spectral properties of the signal. An alternative measurement technique had to be adopted and this led to the idea of using an inverse window on the block of data just before doing the FFT. This window is the inverse of the exponential window used by the test signal itself and it flattens out the block as shown in figure 3.3.

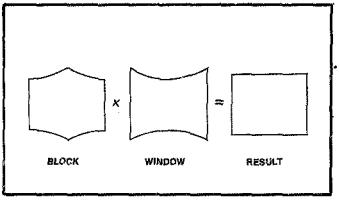

Figure 3.3 Inverse windowing of blocks

Due to the fact that the signal frequency was chosen such that no spillage occurred to other frequency components, the windowed signal was a pure sinusoidal with only one spectral line at f_r. The SNR obtained for a pure uncoded signal was 94 dB. This was a much more realistic value for accurate measurement.

One disadvantage however of the inverse windowing, is that a

certain amount of distortion is being introduced. The effect of this is assumed to be negligible on the following grounds :

- -The difference between the amplitudes at the beginning of the block and at the end of the block are very small for the chosen values of b and tblk. The distortion introduced will thus also be very small.
- -Noise components at the beginning and at the end of the block are increased or boosted, whereas the noise components in the middle of the block are decreased. To a certain degree, this has the effect of cancellation.

An equation describing the shape of the window had to be derived. This was done by looking at the basic formula for the block.

$$v(t) = vpeak.exp(b|t|).sin(2\pi f_r t) \qquad -tblk/2 =< t < tblk/2$$

The window must remove the exponential term and is thus given by:

$$w(t) = wconst.exp(-b|t|)$$

The constant wconst is needed to make sure that the rms value of the signal stays the same before and after the windowing. This is a critical requirement for accurate measurement of the gain variation. It is done by setting the rms values of the unwindowed signal equal to that of the windowed signal and then solving wconst.

$$v(t)|_{rms} = v(t).w(t)|_{rms}$$

After a rigorous integration process, shown in appendix A, the value of wconst was found to be equal to :

$$wconst = \sqrt{(1-exp(-b.tblk))/(b.tblk)}$$

The value proved to be correct because the value obtained for the gain of an uncoded signal was exactly one. Tests could now be performed to obtain the SNR and gain graphs of ideal encoders and decoders.

3.3 Results

In the first simulation, the reference curves for an ideal encoder-decoder pair had to be found. The transposed test configuration as described in chapter one was used. All filters were excluded. The resulting graphs had to be compared with those obtained for a pure sinusoidal signal in chapter 2.5, in order to determine the effect of the Poisson window. At first, only the 50 blocks of which the test signal was built up, were used for processing. This provided a signal level resolution of roughly 1,2 dBm0. Results are shown in figure 3.4 and 3.5.

The basic shape of the curves looked the same as for the pure sinusoidal tests obtained in chapter 2.5. It was impossible though to continue the assessment due to the fact that the exact shape of the graphs were unknown. At this stage no additional information regarding the sensitivity to signal level, could be obtained. Tests with higher resolutions thus had to be performed.

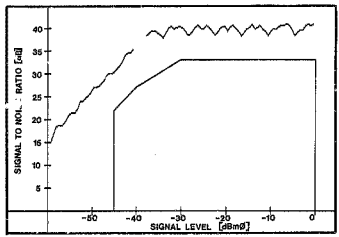

Figure 3.4 SNR for an ideal encoder-decoder pair using a Poisson windowed signal

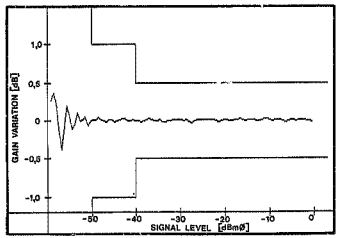

Figure 3.5 Gain variation for an ideal encoder-decoder pair using a Poisson windowed signal

In order to get graphs with higher resolutions, more blocks had to be used for processing. This could be achieved by shifting the block-starts by only fractions of the total blocklength and overlapping them. Attempts to do this revealed a severe limitation on the overlapping of blocks for this test signal. Problems arose in the middle of the blocks due to the fact that two half-blocks from either side of the test signal had to be linked up for processing.

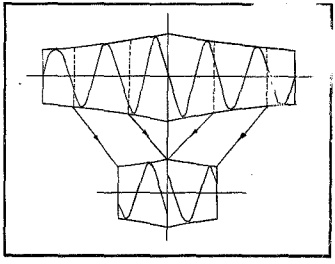

Figure 3.6 The linking of half blocks

The signal frequency was chosen such that an integer number of cycles fall into one block. There will thus always be an integer number of half cycles in half a block. If a half-block starts on a zero-crossing, it will also end on a zero-crossing and by linking the other half-block to it the continuity of the sine

wave will be preserved. The same argument holds for blocks starting and ending exactly on peaks of the sinusoidal. If the blocks start at any other point, a section of the sine wave will be missing in the middle of the block. This is illustrated in figure 3.6. The effect of these discontinuities in the sinusoidal is distortion of the frequency spectrum which makes the separation of signal and noise impossible.

The shifting of blockstarts had to be done in such a way that each block started on either a zero crossing or a peak of the sinusoidal. With the chosen frequency (1031,25 Hz) and the sampling frequency (8000 Hz) there are only two such points per block available that coincide with samples. All other zero crossings and peaks fall between samples, and this makes it impossible to start blocks at these points. The maximum obtainable resolution, using this measurement method, is (2X50)-1 = 99 blocks. (The last block cannot be shifted by half a block because it will overlap with the other half of the test signal.) This corresponds to a signal level resolution of approximately (60/99) = 0,61 dBm0.

There exists thus a fundamental limit on the resolution obtainable due to the specific choice of the signal frequency and sampling frequency. Changing the signal frequency to 1062,5 Hz, which is also an acceptable multiple of 31,25 Hz, results in twice as many available block starting points. The resolution can thus be improved to 0,3 dBm0. This though is still not sufficient to determine the exact shape of the SNR and gain graphs. An

alternative measurement method or test wave had to be found to overcome this restriction.

3.4 Summary

Though all tests performed in the previous section were without filters, it must be pointed out that one of the main reasons for using the Poisson windowed signal, was the need for a gradual change in amplitude level when probing the whole range. The filter response would be upset by a stepped amplitude increase as used in chapter 2.5. The property of the test signal being a one-shot method was also a major consideration for its choice.

Two major drawbacks of this test signal were identified in the previous section. The first one was the limit of 46 dB on the maximum signal to noise separation. This was due to the shape of the power spectrum of the blocks used for processing. The idea of inversely windowing the block just before taking the FFT solved this problem. The signal energy was now confined to one spectral line and this made accurate measurement possible. A certain degree of distortion was introduced by the inverse windowing, but the effect of it on the reference curves was assumed to be negligible.

The second problem was the limitation on the signal level resolution. This was due to the problems that resulted during the merging of half-blocks from either side of the test signal. Up to now, the exact shapes of the SNR an gain graphs were unknown due to a lack of resolution and this made assessment of the effect of

the Poisson windowing on the reference graphs impossible.

With these problems in mind, the idea of a different test signal without such limitations originated. It was that of a single sided exponentially windowed sinusoidal and it will be the topic of the next chapter.

4 SINGLE SIDED EXPONENTIAL WINDOW

4.1 Signal properties

This signal has all the favorable properties of the Poisson windowed signal, without the restrictions identified in the previous chapter. It is shown in figure 4.1.

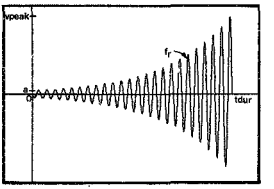

Figure 4.1 The single sided exponentially windowed test signal

It can be seen that the amplitude starts off at a very small value and increases gradually to finish at the clipping level of the codec, vpeak. The signal thus adheres to the requirements of a test signal for filtered codecs. It is still a one-shot method, covering the whole amplitude range in the same time as the Poisson windowed signal.

The most favorable property is that there is no limit on the overlapping of blocks. Each block of 256 points, no matter where it is chosen, will have an integer number of cycles and will be phase coherent. The signal level resolution can thus be increased

to any required level.

The signal equation is given by :

$$v(t) = vpeak.exp(b.(t - tdur)).sin(2\pi f_r t)$$
$$0 < t < tdur$$

The exponential time constant, b, is half of that for the Poisson windowed signal.

$$b = -ln(a)/tdur$$

All the other variables stay the same. The blocks used for processing still consist of 256 points, corresponding to tblk = 32 ms.

A close look at the blocks used for processing, revealed that the same amplitude range was covered per block as with the Poisson windowed signal. The distortion introduced by the inverse windowing just before taking the FFT, was thus of the same order and could still be neglected.

4.2 Test setup

Various tests could now be performed with the required resolution. Exactly the same procedure was followed as with the Poisson windowed signal. The whole test signal would first be sent through the simulation package. The result would then be processed blockwise to give the data needed to compile the SNR

end gain graphs.

The inverse windowing just before taking the FFT had to be changed slightly. A block of data and its corresponding window are shown in figure 4.2.

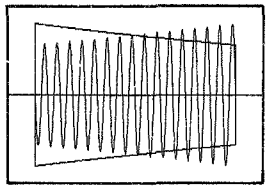

Figure 4.2 Data block and its corresponding window.

With the basic equation for a data block given by :

$$v(t) = vpeak.exp(b.t).sin(2\pi f_r t)$$

the window must remove the exponential section and will be given by :

$$w(t) = wconst * exp(-b.t)$$

Here again, wconst is a constant needed to make sure that the unwindowed signal and the windowed signal have the same rms

value.

$$v(t)|_{rms} = v(t)w(t)|_{rms}$$

The integration process is similar to the one shown in appendix A and it was found that :

$$wconst = \sqrt{(\exp(2.b.tblk) - 1)/(2.b.tblk)}$$

Performing the same test as in chapter 3.2 on an uncoded signal to determine the maximum signal-to-noise separation gave values of more than 200 dB. This indicated the possibility of very accurate measurement.

Two tests were performed. The first test was on an unfiltered ideal codec and the second test was on a filtered codec. Figure 4.3 shows the test setup where the first test was performed with the switch in a closed position, bypassing the filters. In this way the effect of the filters on the reference curves could be studied.

Two important points had to be taken into consideration. The first one was the need for a 'startup' block. This is an extra block of data at the beginning of the test signal used to give the filters a chance to settle down and let all the transient responses die away. This block is not used for processing.

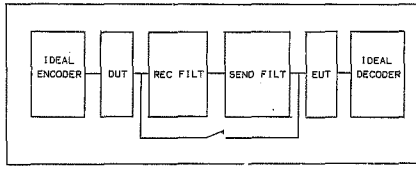

Figure 4.3 Test setup with and without filters

The other filter effect that had to be compensated for is the phase delay it introduced. The theoretical phase delay was calculated and the output data was shifted correspondingly during the decimation procedure. The exact method used will be described in chapter 5.3.10.

The results of all the tests performed are shown and discussed in the next section. The effect of changing the test time, tdur, will also be discussed.

4.3 Results

4.3.1 Unfiltered Codecs

The first test run was performed on an ideal encoder-decoder pair. The resolution used to plot the graphs was very high in order to get the exact shapes of the curves. The SNR and gain variation graphs, each with a resolution of 785 points (60/785 = 0,076 dBm0), are shown in figures 4.4 and 4.5.

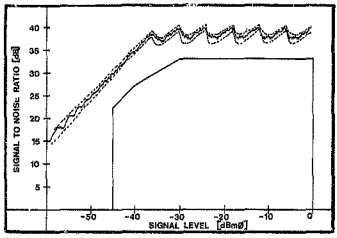

Figure 4.4 SNR graph for an ideal encoder-decoder pair using the single sided exponential test signal with dotted lines showing the envelope of the pure sinusoidal test results

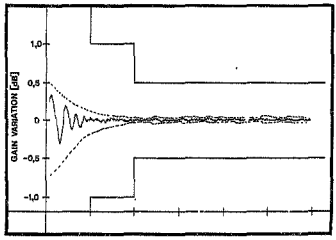

Figure 4.5 Gain variation for an ideal encoder-decoder pair using the single sided exponential test signal with dotted lines showing the envelope of the pure sinusoidal test results

The outsides of the envelopes of the graphs for the pure sinusoidal signal are shown by the dotted lines. The windowing of the test signal thus produced much smoother curves in both cases. This is a very desirable property because devices can now be tested with fewer points, without missing sharp peaks or dips. The processing time can thus be reduced.

The exact effect of the windowing of the test signal is an increase in bandwidth. This gives a better amplitude distribution of the input signal, decreasing the sensitivity of the SNR and gain variation graphs with respect to signal level.

The bandwidth of the test signal is dependant on the exponential time constant b. This led to the idea of changing b, and observing the effect on the reference curves. Changing b merely implied changing the test time. Decreasing the test time was the only acceptable thing to do, which meant increasing b. Two tests were performed with different values of tdur. The one used half the test time (tdur = 0,8 s) and the other used one eighth of the original test time (tdur = 0,2 s) to cover the same amplitude range. The two SNR graphs are shown in figures 4.6 and 4.7. Figures 4.8 and 4.9 are the gain variation graphs for tdur = 0,8 s and tdur = 0,2 s respectively.

The SNR graphs became smoother as tdur decreased. Due to the much larger bandwidth covered, this could be interpreted as an approximation to the measurement method specified by the CCITT using noise.

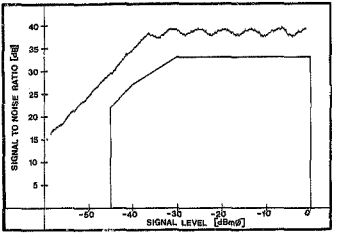

Figure 4.6 SNR graphs for tdur = 0,8 s

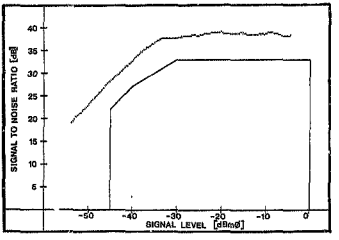

Figure 4.7 SNR graph for tdur = 0,2 s

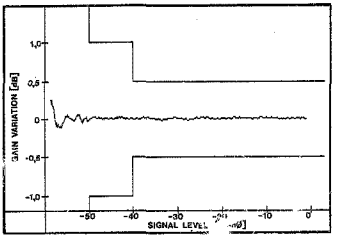

Figure 4.8 Gain variation graph for tdur = 0,8 s

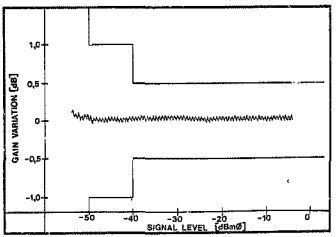

Figure 4.9 Gain variation graph for tdur = 0,2 s

There is however an upper limit to the value of b, due to the increasing amount of distortion being introduced during the inverse windowing process. Changing b too much will require an alteration of the masks as specified by the CCITT as well.

The gain graphs are also affected adversely if the value of b is too high. The curves are much smoother at low signal levels but an oscillatory behavior is developed if the value of b becomes too high.

It is thus preferable to keep to the original value of tdur (1,6 s), in order to minimise distortion and to be able to use the same windows as specified by the CCITT.

4.3.2 Filtered codecs

Next the receive and send filters were included in the simulation. The results are shown in figures 4.10 and 4.11. The dotted lines on the graphs represent the results of the previous section viz. for unfiltered codecs.

It can be seen that there is a drop in the SNR graph of about 1,5 dB. This is not an unexpected result due to the fact that the filters ensure a more continuous amplitude distribution of the input of the encoder under test. It can be shown (Hanrahan 1984b), that a chain of two ideal half-channel test pairs in cascade has the same performance as two ideal full channel test pairs.

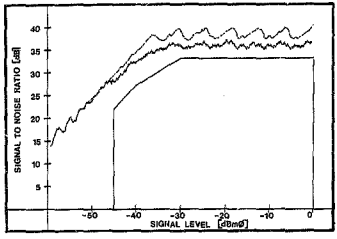

Figure 4.10 SNR graph for an *ideal filtered* codec using a single sided exponential test wave, with dotted lines showing results for an unfiltered codec

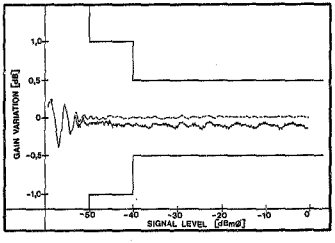

Figure 4.11 Gain variation for an ideal filtered codec using a single sided exponential test wave, with dotted lines showing results for an unfiltered codec

If however, the amplitude distribution of the test signal is changed by some analogue process like the filters or dithering in the middle, the quantisation distortion of the second two half-channel test pairs will be added.

The gain variation graph has an offset of approximately -0,1 dB. This corresponds to the theoretical filter gain of 0,988 at the signal frequency (1031,25 Hz). A correction factor can be introduced to eliminate the offset, but it is suggested that a new gain variation mask is defined. This mask will specifically be for the transposed test configuration on filtered codecs and will merely be the old one dropped by 0,1 dB.

4.4 Summary

The advantages of this test signal were pointed out in the beginning. The two main ones were :

- The continuous increase in the input amplitude, which is a necessity for filtered codecs.
- The fact that there is no limit to the overlapping of blocks during processing. The exact shape of the curves could thus be determined.

The need for the inverse windowing of blocks just before taking the FFT's still existed. After defining this window, two sets of tests could be performed.

The first test performed was on an ideal encoder-decoder pair and

a favorable property of the test signal was identified. It was the elimination of the severe sensitivity of the SNR graph experienced for the pure sinusoidal test signal. This was due to the increase in bandwidth, caused by the windowing process. The increased bandwidth implied a better amplitude distribution of the input signal. The effect of increasing b, or decreasing the test time (tdur) was also investigated. It was found that too short test times were undesirable due to distortion being introduced during the inverse windowing process.

The second test was on an ideal encoder-decoder pair with the send and receive filters. A drop of both curves occurred. The drop of the SNR graph was due to the more continuous amplitude distribution of the filtered signal. It was mentioned in chapter 1.3.5 that the input signal for the encoder under test should have a continuous probability density function to prevent small errors in the encoder and decoder from going undetected. Four operations on the intermediate signal were mentioned, one of which was analogue filtering. By using the same mask for the SNR, a much smaller margin for error is allowed. This has the same effect as performing half-channel tests with their more stringent specifications. It will thus ensure that the encoder can form an acceptable communication link with any decoder and vice versa. The drop in the gain variation graph was merely due to the non unity gain of the filters at the specified filter frequency. The use of a new gain mask was suggested for testing filtered codecs.

5 PROGRAM DESCRIPTION

5.1 Introduction

Chapter 2 briefly described the three main sections into which the programs were divided viz. simulation, processing and plotting. A block diagram combining these programs and showing all the procedures used, is shown in figure 5.1. All the main variables used for the intelligible transfer of data between these procedures are shown as well. The hardware requirements of the system will be given in section 5.2. After that, each one of the three programs, with all its procedures, will be described in detail under the headings as shown in figure 5.1.

5.2 Hardware requirements

Programs have to be run in Turbo Pascal (version 3) on an IBM PC, XT or AT computer (or compatible) with :

- 640 kbytes of RAM,
- 8087 co-processor,
- a graphics card and monitor.

Due to the long running times of the simulation and processing programs, it is suggested that an AT-computer is used.

5.3 Simulation

Three simulation programs were used and the names of the files each one generated are shown in table 5.1.

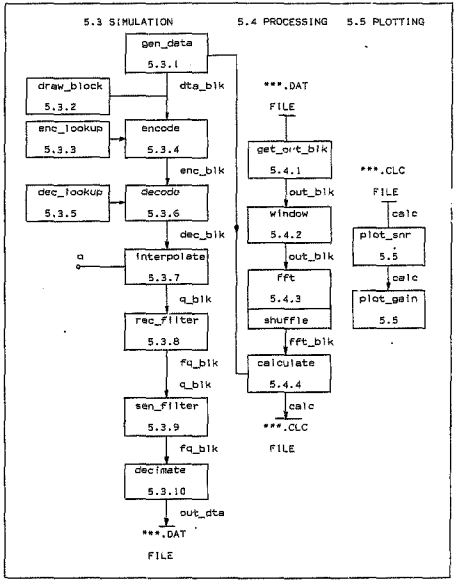

Figure 5.1 Block diagram of overall test system layout.

Test signal	Program name	File generated	Filters
Sinusoidal	CODTESTS.PAS	SCOD.DAT	Excluded
Poisson windowed	CODTESTP.PAS	PCOD.DAT	Excluded
Single sided exponential	CODTESTB.PAS	BCOD.DAT BFULL.DAT	Excluded Include.

Table 5.1 Different simulation programs and files

As mentioned in chapter 2 data was handled in blocks of 256 points. This was due to the fundamental restrictions of Turbo Pascal on a maximum of 64 kbytes of memory available for data variables. Pointer variables were also used to overcome this limitation. One big variable 'out_dta', containing the whole output response of the simulation, exists. It is filled up by the smaller blocks as the simulation progresses. Once the simulation is completed, 'out_dta' is written to a file. The listing of the simulation program can be found in appendix B. The declaration of constants and variables is shown on the first page and is followed by all the procedures used. Each of these will now be discussed in detail.

5.3.1 Procedure gen_data

Inputs : Blocknum

Outputs : dta_blk : array[0..255] of real.

Description : This procedure generates a block of input data, which corresponds to a section of the test signal. There are three different gen_data procedures, one for each one of the three CODTEST programs. This is also where the main difference between these programs lies. Each program uses the relevant signal equation i.e. sinusoidal, Poisson windowed or single sided exponential to generate the input sequence. Blocknum specifies the block being generated and a time shift of teqv = (blocknum-1)*tblk is introduced to get the correct offset on the time axis. The time trel is the time relative to the blockstart and is incremented by one sampling interval (125 µs) from sample to sample.

The pure sinusoidal test signal is easy to generate. The amplitude is incremented form block to block according to an exponential function. This is done to ensure a linear amplitude change on the logarithmic signal level scale (dBm0). A slight offset was introduced in the sine function, (+1E-9), just to distinguish between +0 and -0 values during encoding.

For the Poisson windowed signal, only the first 128 values are generated. The first half is then duplicated onto the second half, with opposite signs. It must be noted that the time shift between blocks is only half a block due to the fact that half-blocks from either side of the test signal are merged

The single sided exponential test signal has a block repetition interval of tblk. There is an additional time delay of tblk to

implement the extra block of data needed for starting up the filters.

5.3.2 Procedure draw_block

Inputs : dta_blk

Outputs : Display output.

Description : This procedure merely draws the block of data. It was used in the initial stages of programming as a rough check on the input data as well as to ensure correct linking of half-blocks. It calls procedure 'frame' to draw a frame around the picture and then merely plots out the data points.

5.3.3 Procedure enc_lookup

Inputs : None

Outputs : idl_enc : array[1..128] of integer

Description : This procedure generates a lookup table with the 128 decision levels of an ideal A-law encoder. The transfer function of the encoder is implemented in segments, each segment being a linear staircase with 16 intervals. The equations used are given by :

$$idl\_enc[int] = 2*int \qquad \text{for seg} = 0$$

$$idl\_enc[int + 16*seg] = 2^{(seg-1)}.(32 + 2*int) \qquad \text{for seg} > 0$$

where int = interval (1 to 16)

seg = segment (0 to 7)

5.3.4 Procedure encode

Inputs : dta_blk, idl_enc

Outputs : enc_blk : array[0..255] of byte.

Description : This procedure takes each one of the input values of dta_blk and encodes them using the lookup table generated by the previous procedure. The most significant bit is set according to the sign of the signal. The encoder output value is then found by using the successive approximation technique. The output is of the type byte which corresponds to the real life 8-bit code produced by an encoder.

5.3.5 Procedure dec_lookup

Inputs : None

Outputs : idl_dec : array[1..128] of integer.

Description : This procedure generates a lookup table with the 128 decoder output levels of an ideal A-law decoder. The equations used are :

$$idl\_dec[int] = (2*int)-1 \qquad \text{for seg} = 0$$

$$idl\_dec[int + 16*seg] = 2^{(seg-1)}.(31 + 2*int) \qquad \text{for seg} > 0$$

where int = interval (1 to 16)

seg = segment (0 to 7)

5.3.6 Procedure decode

Inputs : idl_enc, enc_blk

Outputs : dec_blk : array [0..255] of real

Description : This procedure merely checks the sign of the signal

and then returns the decoder output level corresponding to the encoder code. The decoder lookup table generated by the procedure dec_lookup is used.

5.3.7 Procedure interpolate

Inputs : dec_blk

Outputs : q_blk : pointer variable with q * 256 points.

Description : This procedure interpolates by a factor q between the samples. The interpolation is required to prevent distortion due to the overlapping of the two frequency spectrum halves of the digital filters. A factor of q = 32 was found to be satisfactory. Indexes 'sampl' and 'offst' were used to address all the points. 'Sampl' points to the specific sample and 'offst' points to one of its 32 interpolated values.

5.3.8 Procedure rec_filter

Inputs : q_blk, last_rec, rec_del

Outputs : fq_blk : pointer variable with q * 256 points.

Description : This procedure is a digital simulation of a typical receive filter used in PCM codecs. It is a fifth order elliptical filter with the following poles and zeros :

Poles : p_0 = -16800 rad/s

p_1, p_2 = -9626 ± j 17379 rad/s

p_3, p_4 = -2230 ± j 21370 rad/s

Zeros : w_1, w_2 = 0 ± j 31420 rad/s

w_3, w_4 = 0 ± j 44610 rad/s

A standard filter design package was used to split the filter up into three parallel sections for easier realization. One first order section and two second order sections are used and are shown in figre 5.2.

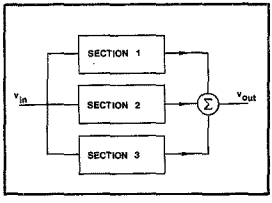

Figure 5.2 Breakdown of filter into parallel sections

Each section is of the form :

$$H(z) = \frac{a_0 + a_1 z^{-1}}{b_0 + b_1 z^{-1} + b_2 z^{-2}}$$

Values for the coefficients of each section were found using the filter design package. A standard realization method was used and is shown in figure 5.3.

Additional variables last_rec and rec_del were needed to store the delayed input and output values of each section. Variable last_rec stores the previous input and is the same for each

section. Variable rec_del is a 2 X 3 array, storing the last two output values for each filter section. These variables have to be intialized at the beginning of the program and are thus set equal to zero. After a block has been sent through the filter section, these variables are passed to the main program and stored until filtering of the next block.

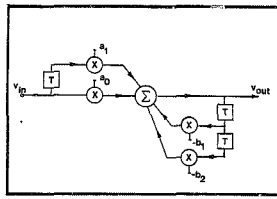

Figure 5.3 Realization method for filter sections

5.3.9 Procedure sen_filter

Inputs : q_blk, last_sen, sen_del

Outputs : fq_blk

Description : This procedure simulates a typical send filter used in PCM codecs. It is also a fifth order elliptical filter with poles and zeros :

Poles : P_0 = -16800 rad/s

P_1, P_2 = -9700 ± j 17500 rad/s

P_3, P_4 = -2360 ± j 22400 rad/s

Zeros : w_1, w_2 = 0 ± j 29200 rad/s

w_3, w_4 = 0 ± j 43200 rad/s

Implementation of this filter is exactly the same as for the receive filter. The output of the receive filter, fq_blk, is transferred to variable q_blk, which is used as the input of this procedure. Variable fq_blk is then used for the output of the send filter.

5.3.10 Procedure decimate

Inputs : fq_blk, blocknum, z_sampl, z_offst

Outputs : out_dta : *pointer variable with 12800 real points.*

: point : integer.

Description : This procedure removes all the redundant data points which were introduced by the interpolation procedure. For each original sample there are q = 32 interpolated values. This procedure takes one of these 32 values per original sample and discards the other data points inbetween. It is necessary to ensure that the correct one of the 32 possible samples is taken. The beginning of a block must for example start on a zero crossing. This is illustrated in figure 5.4.

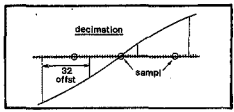

Figure 5.4 Illustration of the decimation procedure

Variable blocknum is used to identify the beginning of the output waveform. The procedure ignores the first block (blocknum = 1), which is the startup block for the filters. It only starts filling up 'out_dta', when the second block of data (blocknum = 2) arrives.

Two additional variables, z_sampl and z_offst, give the position of the first zero crossing in the second block. The values of these variables are set in the main program according to the theoretical phase shift introduced by the filters.

If the filters are excluded, the first zero crossing will occur at sample number 0 and interpolation offset number 16, the middle of the first sample. This would correspond to values of z_sampl = 0 and z_offst = 16. If the filters are included, a time delay will occur and the position of the first zero crossing will have to be determined. This is done by running the simulation without encoding or decoding. The position of the first zero crossing for the filters used was found at sampl = 2, offst = 13. This corresponded to the theoretical value, calculated as shown below.

Receive filter phase delay at f_p = 44,4°
Send filter phase delay at f_p = 40,0°
Total phase delay = 88,4°

Time delay in terms of interpolation intervals
= (88,4°/360°) * (8000*32)/1031,25
= 61 interpolation intervals.

The zero crossing will thus occur at

(sampl, offst) = (0, 16) + 32 intervals

= (1, 16) + 29 intervals

= (2, 13)

Thus z_sampl = 2 and z_offst = 13.

The variable 'point' is used to point at the next free position in 'out_dta'. If 'point' reaches the value of 12800 + 1, 'out_dta' is full and a message will be displayed. The main program then transfers 'out_dta' to a file with extension .DAT.

5.4 Processing

There are a number of processing programs for the different test signals and for the different resolutions. The suffix P is used to denote the Poisson windowed signal and the suffix B is for the single sided exponential. It is followed by a number giving the number of points at which calculations for the SNR and gain are done.

The constants and variables used are similar to the ones used in the simulation program. The declaration of the former and the heading of the processing program are shown on the first page of appendix C. After that, each of the procedures as well as the main program are listed. The procedures will now be described.

5.4.1 Procedure get_out_blk

inputs : blocknum, out_dta

Outputs : out_blk : array [1..255] of real

Description : The main program reads the relevant .DAT file and transfers the data points to the big variable 'out_dta'. 'Out_dta' will thus contain the whole output waveform of the simulation. This procedure merely collects a block of 256 data points from 'out_dta' for processing. The block is then called out_blk. The program also uses a gen_data procedure to regenerate the input signal for calculation of the gain variation. The overlapping of blocks in both the gen_dta and get_out_dta procedures have to be the same.

5.4.2 Procedure window

Inputs : out_blk

Outputs : out_blk

Description : This procedure applies the inverse windowing just before doing the FFT. There are two of these procedures, one for each of the test signals. It also calculates the relevant window constant as described in chapters 3.2 and 4.2.

5.4.3 Procedure fft

Inputs : out_blk

Outputs : fft_blk : array[0..255] of real

Description : This procedure performs a Fast Fourier Transform on the block of input data. The procedure can be split up into two sections, viz. shuffling and merging. The former is performed by a separate procedure called shuffle and merely performs a shuffling process, also called bit reversal (Stelglitz, 1974). The procedure fft then performs the merging process which is also

referred to as the butterfly process. This is a standard procedure also described by Steiglitz (1974). For the sampling frequency of 8000 Hz and 256 samples per block, the frequency line separation is 31,25 Hz. The signal will thus occur at the (1031,25/31,25) = 33rd spectral line.

5.4.4 Procedure calculate

Inputs : blocknum, in_blk, fft_blk

Outputs : calc : array[1..mxblk, 1..3] of real

Description : This procedure calculates all the values required to plot the SNR and gain variation graphs. First of all it *calculates* the input rms signal level. It then converts this value to a dBm0 value, using 2840 units as a reference level :

$$v[dBm0] = 20 * log(v_{rms}/2840)$$

Next is the calculation of the signal and noise components of the output signal. Due to the inverse *windowing process, all the* signal power will be confined to the spectral line at f_n (1031,25 Hz). This corresponds to line number (1031,25/31,25) = 33 and the output rms signal value will thus be given by

$$vrms\_o = \sqrt{fft\_blk[33]^2 + fft\_blk[256-33]^2}$$

The mean square value of the noise is found by summing the square values of the remaining components, excluding the line at 0 Hz. The SNR can now be calculated by :

SNR = 20 log(rms value of signal/rms value of noise)

The gain is calculated by taking the ratio of the input signal level to the output signal level.

gain = rms value of input/rms value of output

The signal level (in dBm0) for each block and the corresponding values of the SNR and gain, are stored in variable 'calc'. After the whole signal has been processed the variable 'calc' is transferred to a file with a relevant name and .CLC extension.

5.5 Plotting

The different plotting programs merely differ in the number of data points they use for plotting the graphs. The value of mxblk as well as the names of the .CLC files gives the number of points. There are two different plotting procedures, one for the SNR (plot_snr) and the other for the gain variation graph (plot_gain). They are basically the same and will be described in general.

The main program reads the .CLC file created by the processing program and then passes variable calc to the relevant procedure. The psophometric weighing of the SNR graph is done at this stage. According to the CCITT recommendation G223 (1984e) this corresponds to adding 0,9 dB to the SNR. It must also be noted that the gain values are converted to dB before the plotting of the graphs. This is done according to :

$gain_{dB}$ = 20 log (rms value of input/rms value of output)

Now each procedure sets up the axis and plots the graph. The masks for the graphs are also drawn. The high resolution mode of Turbo Pascal, corresponding to 640 X 200 pixels, was used. Graphs were printed out using the normal 'print screen' command. The listing of the program PLOT785.PAS can be found in appendix D.

6 CONCLUSIONS

6.1 Review of the work done

The purpose of this project was to critically evaluate a proposal for a new test signal, used to test PCM codecs and filters in the transposed test configuration. The test signal to be evaluated was a Poisson windowed sinusoidal and it had a few properties which made it attractive for testing purposes.

- Due to the gradual change in the amplitude level, it was suitable for filtered codecs.
- It was a 'one shot' measurement technique.
- FFT processing could be done blockwise without an increase in bandwidth.

A set of programs were developed to simulate the transposed test configuration and to obtain the reference curves of the Poisson windowed test signal. The three main programs used were described in chapters 2 and 5. An additional program was written to obtain the reference curves of an ideal A-law encoder-decoder pair, using a pure sinusoidal waveform as specified by the CCITT. This was required to make a comparative assessment of the newly proposed signal.

In chapter 3 some tests were performed to determine the maximum possible SNR separation of the signal. It was found that accurate measurement was impossible with the specific signal and method used. The spectral properties of the test signal were such that

too much energy lay in bands adjacent to the si frequency. This made sufficient separation of signal and noise components during processing difficult. The problem was overcome by using an inverse window on a block of data just before taking the FFT. Although some distortion was introduced by the windowing, it was assumed to be negligible. Arguments were put forward to show that the distortion introduced was negligibly small.

The reference curves for an ideal A-law codec were obtained using the Poisson windowed signal. It was found however, that a fundamental limit to the number of blocks overlapping each other during processing existed. There was thus an upper limit to the obtainable resolution. The sensitivity of the graphs to signal level could not be determined. It was at this stage, taking the above mentioned problems into consideration, that the idea of the one sided exponential test signal originated.

In chapter 4 the advantages of the single sided exponential test signal were listed. The most important ones were :

- Its suitability for filtered codecs.
- No limit to the resolution of graphs.
- Blockwise processing without the problems associated with the linking of half-blocks.

The first tests performed were to obtain the reference curves for an ideal A-law codec. The curves obtained were much smoother than those for the pure sinusoidal test signal because of the larger

bandwidth occupied by the windowed signal. The windowed signal had a more even amplitude distribution causing the graphs to be smoother. It was a favorable property because the graphs could now be compiled using less points, without losing detail like in the case with the pure sinusoidal test. The processing time could thus be decreased.

The effect of increasing the exponential time constant b, or indirectly decreasing the test time, tdur, was also investigated. It was found that high values of b gave a more continuous amplitude distribution and it could be looked at as an approximation to the noise measurement method specified by the CCITT. Problems associated with too high values of b were :

- Increased distortion of the signal and quantisation noise during the inverse windowing process.
- A large bandwidth could present problems with filtered codecs because the magnitude response is not always flat and the phase response is not always linear over the bandwidth in question. This would make modeling difficult.

There was thus some limit to the value of the exponential time constant at which these problems were still tolerable. In order not to have to change the CCITT masks, a value for b, corresponding to a test time of 1,6 s was suggested.

The final tests performed were on an ideal A-law encoder-decoder pair with typical send and receive filters. The effect of the

filters on the SNR graph was a slight drop at higher amplitudes. The filtering corresponded to an analogue operation on the signal which gave it a more continuous amplitude distribution. This ensured that the encoder under test had its entire input range probed and prevented small errors from going undetected. A new mask was proposed for the gain variation graph of the filtered codec to compensate for the non-unity gain of the filters at the signal frequency.

6.2 Conclusions

An evaluation of the Poisson windowed test signal for the transposed test configuration was conducted. Some disadvantages and fundamental limitations of this test signal were identified. Using the basic properties of the Poisson windowed signal and information obtained from the tests performed, the idea of a single sided exponentially windowed test signal emerged. The reference curves for this test signal were obtained for both filtered and unfiltered codecs. Most of the properties of the reference curves were favorable.

It can thus be said that a 'one shot' test signal for testing PCM codecs with their filters in in-service situations has been developed. In the case of the gain variation graph, a new mask which takes the filter gain into consideration, was proposed. The shape of the reference curves for the signal were determined and can now be used for future tests.

6.3 Future work

Now that the exact shape of the new test signal and its reference curves are known, tests can be performed to determine the effect of imperfection in the system components on the SNR and gain variation graphs. These tests can easily be performed by replacing the ideal lookup tables in the encoding and decoding procedures by imperfect ones. The sensitivity of the reference curves to various imperfections can then be determined.

It is also suggested that the influence of the half-channel performance on the overall performance be investigated. Various analogue operations on the test signal have been suggested, in order to obtain more information on the half-channel performance (Henrghan, 1984b). Analogue filtering was one of these operations and its effect on the reference curves was demonstrated in the tests conducted. Other methods for separating half-channel values, like dithering, are beyond the scope of this paper and still have to be investigated thoroughly.

It was noted that the FFT procedure used was inefficient, which led to longer calculation times. Other more efficient FFTs could be used to speed up the running time. In an example shown in appendix E, the bit reversal is done only once and the resulting addresses are stored in an address array. These addresses are used later to reduce the shuffling and merging steps into one procedure. The different sine and cosine terms are also calculated only once in the beginning and the results are stored in an array for later use.

These two techniques will already give a significant improvement in the calculation times. Availability of memory space for these precalculated values could present problems on the computer used, due to its limited memory capacity, but this issue should be investigated thoroughly.

Another technique used to speed up the FFT is to fully write out the first two merging steps and thus do them directly. After this the merging gets too complicated and is handled in the normal manner viz. looping. This technique is shown in appendix E nd could also be used in this application.

Note that t_b was used instead of the normal tbik.

$$v(t)_{rms}^2 = (1/t_b)\int_{-t_b/2}^{t_b/2} v(t)^2\,dt \qquad (1)$$

where $v(t) = v.e^{(-b|t|)}.\sin(\omega t)$

$$= v.e^{(-b|t|)}.(e^{j\omega t} - e^{-j\omega t})/(2j) \qquad (2)$$

Substituting (2) in (1) :

$$v(t)_{rms}^2 = 1/t_b\int_{-t_b/2}^{t_b/2} [v.e^{(-b|t|)}.(e^{j\omega t} - e^{-j\omega t})/(2j)]^2\,dt$$

$$= 2/t_b\int_{0}^{t_b/2} [v.e^{(-bt)}.(e^{j\omega t} - e^{-j\omega t})/(2j)]^2\,dt$$

$$= -2v^2/4t_b\int_{0}^{t_b/2} [e^{(-2bt+2j\omega t)} - 2e^{(-2bt)} + e^{(-2bt-2j\omega t)}]\,dt$$

$$= -v^2/(2t_b)\left[\, 1/(-2b+2j\omega).e^{((-2b+2j\omega)t)} + 1/b\,.\,e^{(-2bt)} + 1/(-2b-2j\omega).e^{((-2b-2j\omega)t)}\,\right]_0^{t_b/2}$$

Substituting the boundery values and noting that :

$$\Delta f = 1/t_b$$

$$\omega = 2\pi f_r$$

$$= 2\pi(k\Delta f)$$

$$= 2\pi k/t_b \qquad (3)$$

$$v(t)_{rms}^2 = v^2/(2t_b)\ [\ C.(1-\exp(-bt_b))\ ] \qquad (4)$$

where $C = 1/(-2b+2j\omega) + 1/b + 1/(-2b-2j\omega)$

$$= -4b/(4b^2 + 4\omega^2) + 1/b$$

$$= -b/(b^2 + \omega^2) + 1/b \qquad (5)$$

For the chosen signal frequency f_r = 1031,25 Hz

and Δf = 31,25 Hz :

$$k = f_r/\Delta f$$

$$= 33$$

and $b = (-2/tdur).\ln(a)$

$$= 8,63 \quad (tdur = 1,6\ s,\ a = 0,001)$$

With these values the first term in (5) can be neglected and (4) becomes :

$$v(t)_{rms}^2 = v^2/(2t_b)\ [\ (1-\exp(-bt_b))/b\ ]$$

$$v(t)_{rms} = v/(\sqrt{2})\ [\ (1-\exp(-bt_b))/(bt_b)\ ]^{1/2} \qquad (6)$$

The equation of the windowed signal is :

$$v(t).w(t) = v.wconst.\sin(2\pi f_r t)$$

with rms value :

$$v(t)_{rms} = v.wconst/(\sqrt{2})$$

Thus

$$wconst = [\ (1-\exp(-bt_b))/(bt_b)\ ]^{1/2}$$

```
{*************************************************************}
{              PROGRAM SIMULATING A CODEC TEST                }
{           USING A ONE SIDED EXPONENTIAL WINDOW              }
{              By B.Cronje    Date : Sept 1987                }
{*************************************************************}
program codec_test ;
{$R+}

const  mxblk = 50          ; {Number of blocks                  }
       ntot  = 12800       ; {Total no of data points = mxblk*256 }
       tdur  = 1.6         ; {Time duration of whole test signal  }
       tblk  = 32E-3       ; {Time duration of one block of data  }
       vpeak = 4096        ; {Clipping level of signal          }
       a     = 0.001       ; {Final value of the Poisson window   }
       fr    = 1031.25     ; {Signal frequency                  }
       q     = 32          ; {Interpolation factor              }
      slewlim = 1.6        ; {Units per millisecond             }


type   blockb   = array[0..255]        of byte    ;
       blockr   = array[0..255]        of real    ;
       blockq   = array[0..255]        of^qcomp   ;
       qcomp    = array[1..q]          of real    ;
       outmat   = array[1..ntot]       of^real    ;
       table    = array[1..128]        of integer ;
       del_mat  = array[1..3,1..2]     of real    ;


var    dta_blk,out_blk     : blockr ;
       enc_blk             : blockb ;
       dec_blk             : blockr ;
       idl_enc,idl_dec     : table  ;
       a_blk,fq_blk        : blockq ;
       out_dta             : outmat  ;
       point               : integer ;
       blocknum,x,y        : integer ;
       z_sampl,z_offst     : integer ;
       last_rec,last_sen   : real    ;
       rec_del,sen_del     : del_mat ;
       filename            : string[20] ;
       outfile             : file of blockr ;
       option              : char ;
```

```
{*************************************************************}
{                  PROCEDURE TO GENERATE DATA BLOCK           }
{                    FOR THE CODTESTS.PAS PROGRAM             }
{*************************************************************}
procedure gen_data(blocknum     : integer ;
                   var dta_blk : blockr
                  );
var  trel,teqv,b,ampl       : real ;
     sampl                  : integer ;

begin
  b := - ln(a) / tdur ;
  ampl := vpeak * 1.1 * exp(b*((blocknum-1)*1.75E-3 - tdur)) ;
  for sampl := 0 to 255 do
    begin
     trel := sampl*125E-6 ;
     dta_blk[sampl] := ampl * sin(2*pi*fr*trel+1E-9) ;
    end;
end;




{*************************************************************}
{                  PROCEDURE TO GENERATE DATA BLOCK           }
{                    FOR THE CODTESTP.PAS PROGRAM             }
{*************************************************************}
procedure gen_data(blocknum     : integer ;
                   var dta_blk : blockr
                  );

var  trel,teqv,b     : real ;
     sampl           : integer ;

begin
  b := (-2/tdur) * ln(a) ;
  for sampl := 0 to 128 do
    begin
     trel := sampl*125E-6 ;
     teqv := (blocknum-1)*tblk + trel - tblk - tdur/2 ;
     dta_blk[sampl] := vpeak*exp(-b*abs(teqv))*sin(pi*fr*teqv) ;
    end;
  for sampl := 1 to 127 do
    dta_blk[sampl+128] := -dta_blk[128-sampl] ;
end;
```

```
{*************************************************************}
{                PROCEDURE TO GENERATE DATA BLOCK             }
{                  FOR THE CODTESTB.PAS PROGRAM               }
{*************************************************************}
procedure gen_data(blocknum    : integer ;
                   var dta_blk : blockr
                  );

var  trel,teqv,b    : real ;
     sampl          : integer ;

begin
  b := -ln(a) / tdur ;
  for sampl := 0 to 255 do
   begin
    trel := sampl*125E-6 ;
    teqv := (blocknum-1)*tblk - tblk + trel ;
    dta_blk[sampl] := vpeak * exp(b*(teqv-tdur))
                            * sin(2*pi*fr*teqv) ;
   end;
end;
```

```
{*************************************************************}
{        PROCEDURE TO DRAW FRAME AROUND PICTURE               }
{*************************************************************}
procedure frame ;

begin
  draw(0,0,0,199,15) ;
  draw(0,199,319,199,15) ;
  draw(319,199,319,0,15) ;
  draw(319,0,0,0,15) ;
end;


{*************************************************************}
{                PROCEDURE TO DRAW BLOCK OF DATA              }
{*************************************************************}
procedure draw_block(dta_blk : blockr) ;

var  x,yold,ynew     : integer ;

begin
  clrscr ;
  graphmode ;
  frame ;
  draw(0,100,219,100,15) ;
  draw(160,0,160,199,15) ;
  for x := 0 to 254 do
   begin
     yold := round((4096 - dta_blk[x]) / 41 ) ;
     ynew := round((4096 - dta_blk[x+1]) / 41 ) ;
     draw(x+32,yold,x+33,ynew,15) ;
    end;

end;
```

```
{*************************************************************}
{                    PROCEDURE ENC_LOOKUP                     }
{*************************************************************}
procedure enc_lookup(var idl_enc : table ) ;

var  seg,int      : integer ;

begin
  for int := 1 to 16 do
  idl_enc[int] := 2*int ;
  for seg := 1 to 7 do
      for int := 1 to 16 do
        idl_enc[int + 16*seg] := round(exp((seg-1)*ln(2))
                                        * (32 + 2*int)) ;
end;



{*************************************************************}
{            PROCEDURE TO ENCODE A BLOCK OF DATA              }
{*************************************************************}
procedure encode(tbl_enc          : table ;
                 dta_blk          : blockr ;
                 var  enc_blk     : blockb
                ) ;

var  sign, index, step     : integer ;
     x                     : integer ;
     s_value               : real ;

begin

 for x := 0 to 255 do
   begin

     {Get and store the sign of the value  }
      if dta_blk[x] < 0
      then sign := 128
      else sign := 0 ;
      s_value   := abs(dta_blk[x]) ;


      index := 128 ;
      step := 128 ;
      repeat
         step := step div 2;
         if s_value<tbl_enc[index-step]
         then index := index - step ;
      until step=1 ;
      enc_blk[x] := index-1 + sign ;

    end;


end;
```

```
{*************************************************************}
{                  PROCEDURE ID_DEC_LOOKUP                    }
{*************************************************************}
procedure dec_lookup(var idl_dec : table ) ;

var  seg,int      : integer ;

begin
  for int := 1 to 16 do
  idl_dec[int] := 2*int - 1 ;
  for seg := 1 to 7 do
    begin
     for int := 1 to 16 do
     idl_dec[int + 16*seg] := round(exp((seg-1)*ln(2))
                                    * (31 + 2*int)) ;
    end;

end;


{*************************************************************}
{                  PROCEDURE IDEAL DECODING                   }
{*************************************************************}
procedure decode(tbl_dec        : table  ;
                 enc_blk        : blockb ;
                 var dec_blk    : blockr
                ) ;

var  sign, index          : integer ;
     s_value              : real ;

begin

  for x := 0 to 255 do
    begin

     {Get and store the sign of the value  }
      sign := enc_blk[x] and 128 ;
      index := (enc_blk[x] and 127) + 1 ;

      if sign=128
      then dec_blk[x] := -tbl_dec[index]
      else dec_blk[x] := tbl_dec[index] ;
    end;

end;
```

```
{*********************************************************}
{       PROCEDURE TO INTRODUCE INTERPOLATION FACTOR q      }
{            By B.Cronje      Date : 19/11/87              }
{*********************************************************}

procedure Interpolate(dec_blk       : blockr ;
                      var q_blk     : blockq
                     );

var  sampl,offst       :      integer ;

begin

 for sampl := 0 to 255 do
  for offst := 1 to q do
   q_blk[sampl]^[offst] := dec_blk[sampl] ;

end;
```

```
{*************************************************************}
{          PROCEDURE TO IMPLEMENT THE RECEIVE FILTER          }
{              By B.Cronje      Date : 09/12/87               }
{*************************************************************}
procedure rec_filter(q_blk          : blockq  ;
                     var  last_rec  : real    ;
                     var  rec_del   : del_mat ;
                     var  fq_blk    : blockq
                     ) ;

const  s1a0 = 7.4250112E-2  ;
       s1a1 = 0             ;
       s1b1 = -0.93648198   ;
       s1b2 = 0             ;

       s2a0 = -0.0666878l   ;
       s2a1 =  6.7647895E-2 ;
       s2b1 = -1.921761     ;
       s2b2 = 0.92755506    ;

       s3a0 = -1.4807212E-3 ;
       s3a1 = -8.3313987E-4 ;
       s3b1 = -1.9757499    ;
       s3b2 = 0.98272901    ;

var  sampl,offst           : integer ;
     s1_del1,s1_del2       : real    ;
     s2_del1,s2_del2       : real    ;
     s3_del1,s3_del2       : real    ;
     in_one                : real    ;

{-------------------------------------------------------------}
function recfilt(in_zero : real) : real ;

var sec1res,sec2res,sec3res    : real ;

begin
 sec1res := s1a0 * in_zero + s1a1 * in_one
          - s1b1 * s1_del1 - s1b2 * s1_del2 ;
 s1_del2 := s1_del1 ;
 s1_del1 := sec1res ;


 sec2res := s2a0 * in_zero + s2a1 * in_one
          - s2b1 * s2_del1 - s2b2 * s2_del2 ;
 s2_del2 := s2_del1 ;
 s2_del1 := sec2res    ;
```

```
  sec3res := s3a0 * in_zero + s3a1 * in_one
           - s3b1 * s3_del1 - s3b2 * s3_del2 ;
  s3_del2 := s3_del1 ;
  s3_del1 := sec3res    ;

  in_one := in_zero ;

  recfilt := sec1res + sec2res + sec3res ;

end;
{-------------------------------------------------------------------}

begin

  s1_del1 := rec_del[1,1] ;
  s1_del2 := rec_del[1,2] ;

  s2_del1 := rec_del[2,1] ;
  s2_del2 := rec_del[2,2] ;

  s3_del1 := rec_del[3,1] ;
  s3_del2 := rec_del[3,2] ;

  in_one := last_rec ;

  for sampl := 0 to 255 do
   for offst := 1 to q do
    fq_blk[sampl]^[offst] := recfilt(q_blk[sampl]^[offst]) ;

  last_rec := q_blk[255]^[q] ;

  rec_del[1,1] := s1_del1 ;
  rec_del[1,2] := s1_del2 ;

  rec_del[2,1] := s2_del1 ;
  rec_del[2,2] := s2_del2 ;

  rec_del[3,1] := s3_del1 ;
  rec_del[3,2] := s3_del2 ;


end;
```

```
{*************************************************************}
{            PROCEDURE TO IMPLEMENT THE SEND FILTER           }
{              By B.Cronje     Date : 06/01/88                }
{*************************************************************}
procedure sen_filter(q_blk                : blockq  ;
                     var  last_sen        : real    ;
                     var  sen_del         : del_mat ;
                     var  fq_blk          : blockq
                     ) ;

const  s1a0 = 8.0622542E-2  ;
       s1a1 = 0             ;
       s1b1 = -0.93648198   ;
       s1b2 = 0             ;

       s2a0 = -7.6213496E-2 ;
       s2a1 =  7.5919373E-2 ;
       s2b1 = -1.9211390    ;
       s2b2 = 0.92701897    ;

       s3a0 = 3.9673138E-3  ;
       s3a1 = -5.6151008E-3 ;
       s3b1 = -1.9740661    ;
       s3b2 = 0.98173143    ;

var  sampl,offst            : integer ;
     s1_dell,s1_del2        : real    ;
     s2_dell,s2_del2        : real    ;
     s3_dell,s3_del2        : real    ;
     in_one                 : real    ;

{-------------------------------------------------------------}
function senfilt(in_zero  :  real) : real ;

var seclres,sec2res,sec3res    : real ;

begin
 seclres := s1a0 * in_zero + s1a1 * in_one
          - s1b1 * s1_dell - s1b2 * s1_del2 ;
 s1_del2 := s1_dell ;
 s1_dell := seclres    ;


 sec2res := s2a0 * in_zero + s2a1 * in_one
          - s2b1 * s2_dell - s2b2 * s2_del2 ;
 s2_del2 := s2_dell ;
 s2_dell := sec2res    ;
```

```
  sec3res := s3a0 * in_zero + s3a1 * in_one
           - s3b1 * s3_del1 - s3b2 * s3_del2 ;
  s3_del2 := s3_del1 ;
  s3_del1 := sec3res    ;

  in_one := in_zero ;

  senfilt := sec1res + sec2res + sec3res ;

end;
{-------------------------------------------------------------------}

begin

  s1_del1 := sen_del[1,1] ;
  s1_del2 := sen_del[1,2] ;

  s2_del1 := sen_del[2,1] ;
  s2_del2 := sen_del[2,2] ;

  s3_del1 := sen_del[3,1] ;
  s3_del2 := sen_del[3,2] ;

  in_one := last_sen ;

  for sampl := 0 to 255 do
   for offst := 1 to q do
    fq_blk[sampl]^[offst] := senfilt(q_blk[sampl]^[offst]) ;

  last_sen := q_blk[255]^[q] ;

  sen_del[1,1] := s1_del1 ;
  sen_del[1,2] := s1_del2 ;

  sen_del[2,1] := s2_del1 ;
  sen_del[2,2] := s2_del2 ;

  sen_del[3,1] := s3_del1 ;
  sen_del[3,2] := s3_del2 ;


end;
```

```
{*************************************************************}
{                      PROCEDURE DECIMATE                     }
{            By B.Cronje     Date : 17/12/87                  }
{*************************************************************}
procedure decimate(fq_blk                    : blockq ;
                   blocknum                  : integer ;
                   z_samp1,z_offst           : integer ;
                   var   out_dta             : outmat ;
                   var   point               : integer
                  ) ;


var  samp1,offst           : integer ;

begin
 if blocknum = 1 then ; {do nothing}
 if blocknum = 2 then
   begin
    for samp1 := z_samp1 to 2** do
      begin
       out_dta[point]^ := fq_blk[samp1]^[z_offst] ;
       point := point + 1 ;
      end;
   end;

 if blocknum >2 then
   begin
    samp1 := 0 ;
    repeat
      out_dta[point]^ := fq_blk[samp1]^[z_offst] ;
      samp1 := samp1 + 1 ;
      point := point + 1 ;
    until (samp1 = 255+1) or (point = ntot+1) ;
     if point = ntot+1 then writeln('Output block filled !') ;
   end;
 end;
```

```
{*************************************************************}
{                        MAIN PROGRAM                         }
{*************************************************************}

begin

 for x := 0 to 255  do new(q_blk[x]) ;
 for x := 0 to 255  do new(fq_blk[x]) ;
 for x := 1 to ncot do new(out_dta[x]) ;

 enc_lookup(id1_enc) ;
 dec_lookup(id1_dec) ;

 point := 1 ;
 last_slew := 0 ;
 last_rec  := 0 ;
 last_sen  := 0 ;
 for x := 1 to 3 do
  begin
   rec_del[x,1] := 0 ;
   rec_del[x,2] := 0 ;
   sen_del[x,1] := 0 ;
   sen_del[x,2] := 0 ;
  end;

clrscr ;
writeln('AVAILABLE OPTIONS : ') ;
writeln ;
writeln(' 1 : enc-dec.                    -   *COD .DAT') ;
writeln(' 2 : enc-dec-rec-sen-enc-dec.    -   *FULL.DAT') ;
writeln ;

repeat
 write('Enter your option : ') ;
 readln(option) ;
until option in ['1','2'] ;
writeln ;
```

```
case option of

  '1' :  begin
           z_sampl  := 0 ;
           z_offst  := 16 ;
           { These two constants give the theotetical zero }
           { crossing of each block beginning.             }
           filename := 'bcod.dat' ;
           for blocknum := 1 to mxblk+1 do
             begin
               writeln('busy with block  ',blocknum ) ;
               gen_data(blocknum,dta_blk) ;

               encode(id1_enc,dta_blk,enc_blk) ;
               decode(id1_dec,enc_blk,dec_blk) ;

               interpolate(dec_blk,q_blk) ;

               decimate(q_blk,blocknum,z_sampl,z_offst,
                        out_dta,point
                        ) ;
             end;
           end ;

  '2' :  begin
           z_sampl  := 2;
           z_offst  := 13 ;
           { These two constants give the theotetical zero }
           { crossing of each block beginning. Receive and }
           { send filters included.                        }
           filename := 'bfull.dat' ;
           for blocknum := 1 to mxblk+2 do
             begin
               writeln('busy with block  ',blocknum ) ;
               gen_data(blocknum,dta_blk) ;

               encode(id1_enc,dta_blk,enc_blk) ;
               decode(id1_dec,enc_blk,dec_blk) ;

               interpolate(dec_blk,q_blk) ;

               rec_filter(q_blk,last_rec,rec_del,fq_blk) ;
               for x := 0 to 255 do q_blk[x]^ := fq_blk[x]^ ;
               sen_filter(q_blk,last_sen,sen_del,fq_blk) ;

               decimate(fq_blk,blocknum,z_sampl,z_offst,
                        out_dta,point
                        ) ;
             end;
           end ;

end;{case}
```

```
    for blocknum := 1 to mxblk do
     begin
      writeln('busy with block  ',51-blocknum ) ;

      for x := 0 to 255 do
        out_blk[x] := out_dta[x + 1 + (blocknum-1)*256]^ ;

      encode(idl_enc,out_blk,enc_blk) ;
      decode(idl_dec,enc_blk,dec_blk) ;

      for x := 0 to 255 do
        out_dta[x + 1 + (blocknum-1)*256]^ := dec_blk[x] ;

     end ;


    writeln('writing to file ',filename) ;
    assign(outfile,filename) ;
    rewrite(outfile) ;
    for blocknum := 1 to mxblk do
     begin
      for x := 0 to 255 do
        out_blk[x] := out_dta[x + 1 + (blocknum-1)*256]^ ;
      write(outfile,out_blk) ;
     end ;
    close(outfile) ;

    clrscr ;
    writeln('Simulation finished.') ;

end.
```

```
{*************************************************************}
{              PROGRAM PROCESSING CODEC TEST OUTPUT           }
{                 By B.Cronje   Date : 4 Jan 1988             }
{*************************************************************}
program process_data ;
{$R+}

const  mxblk = 785      ;   {Number of blocks                  }
       ntot  = 12800    ;   {Total no of points = 256*mxblk    }
       tdur  = 1.60     ;   {Duration of test signal(50 blocks) }
       vpeak = 4096     ;   {Clipping level of signal          }
       a     = 0.001    ;   {Final value of the Poisson window  }
       fr    = 1031.25  ;   {Signal frequency                  }

type   block1   = array[0..255]          of integer ;
       blockr   = array[0..255]          of real    ;
       outmat   = array[1..ntot]         of real    ;
       calcadr  = array[1..mxblk]        of^calcmat ;
       calcmat  = array[1..3]            of real    ;


var    in_blk,out_blk    : blockr ;
       fft_blk           : blockr ;
       out_dta           : outmat  ;
       calc              : calcadr ;
       calcfile          : file of calcmat ;
       point,blk         : integer ;
       blocknum,x,y      : integer ;
       filename          : string[20] ;
       infile            : file of blockr ;
```

```
{*************************************************************}
{                PROCEDURE TO GET THE OUTPUT BLOCK            }
{                  By : B.CRONJE  Date : 30/01/88             }
{*************************************************************}
procedure get_out_blk(blocknum      : integer ;
                      out_dta       : outmat ;
                      var out_blk   : blockr
                     ) ;

var  sampl,point      : integer ;

begin
 for sampl := 0 to 255 do
  begin
   point := (blocknum-1)*16 + 1 + sampl ;
   out_blk[sampl] := out_dta[point]^ ;
  end ;
end;
```

```
{*************************************************************}
{                       PROCEDURE WINDOW                      }
{          FOR THE SINGLE SIDED EXPONENTIAL TEST SIGNAL       }
{                  By : B.Cronje  Date : 14/01/88             }
{*************************************************************}
procedure window(var   out_blk : blockr ) ;

const  tblk = 32E-3 ;

var teqv,b,wconst    :  real ;

begin
  b := -ln(a) / tdur ;
  wconst := sqrt((exp(2*b*tblk)-1)/(2*b*tblk)) ;
  for x := 0 to 255 do
   begin
    teqv := x*125E-6 ;
    out_blk[x] := wconst * exp(-b*teqv) * out_blk[x] ;
   end ;

end;
```

```
{*************************************************************}
{       PROCEDURE PERFORMING THE FOURIER TRANSFORM             }
{*************************************************************}
procedure fft(out_blk      : blockr ;
              var  fft_blk : blockr
             );

var  fre,fim                      : blockr  ;
     sec,secnum,merge,k,kl,n,n2   : integer ;
     arg c,s,dumre,dumim          : real    ;

begin
 shuffle(out_blk,fre,fim) ;

 for merge := 1 to 8 do
  begin
   sec := round(exp((8-merge) * ln(2)));     {2^(8-merge)}
   n  := round(exp(merge * ln(2))) ;         {2^(merge)}
   n2 := round(n/2) ;

   for secnum := 0 to (sec-1) do
     begin
       for kl := 0 to n2-1 do
         begin
           arg := 2*pi*kl/n ;
           c   := cos(arg) ;
           s   := sin(arg) ;
           k   := kl + secnum*n ;
           dumre  := fre[k+n2]*c + fim[k+n2]*s ;
           dumim  := fim[k+n2]*c - fre[k+n2]*s ;
           fre[k+n2] := fre[k] - dumre ;
           fim[k+n2] := fim[k] - dumim ;
           fre[k]    := fre[k] + dumre ;
           fim[k]    := fim[k] + dumim ;
         end;
     end;
  end;

  for x:= 0 to 255 do
  fft_blk[x] := sqrt(fre[x]*fre[x] + fim[x]*fim[x]) ;

end;
```

```
{*************************************************************}
{                  PROCEDURE TO SHUFFLE DATA                  }
{*************************************************************}
procedure shuffle(out_blk       : blockr ;
                  var  fre,fim  : blockr
                  ) ;

var  bits           : array[0..7] of byte ;
     bit,index      : integer ;

begin
 for x := 0 to 255 do
  begin
   {Get the binary code  }
   index := x ;
   for bit := 0 to 7 do
    begin
     if odd(index)
     then bits[bit] := 1
     else bits[bit] := 0 ;
     index := trunc(index/2) ;
    end;

   {Reverse bits and convert back to decimal }
   index := 0 ;
   for bit := 0 to 7 do
    index := index + round(bits[7-bit] * exp(bit*ln(2))) ;

   {Assign new block value}
   fre[index] := out_blk[x] ;
   fim[x] := 0 ;

  end;

end;
```

```
{*************************************************************}
{          PROCEDURE DOING CALCULATIONS (SNR, GAIN)           }
{*************************************************************}
procedure calculate(blocknum          : integer ;
                    in_blk,fft_blk    : blockr ;
                    var calc          : calcadr
                   );

var  smsqr,vrms_i,vrms_o      : real ;
     nrms,snr,vdbm_i,gain     : real ;
     r,sb                     : integer ;

begin

 {Signal level calculation}
 smsqr := 0 ;
 for x := 0 to 255 do
   smsqr := smsqr + sqr(in_blk[x]) ;
 vrms_i := sqrt(smsqr/256) ;

 {Signal level in dBm0 }
 vdbm_i := 20 * ln(vrms_i/2840) / ln(10) ;

 {Rms value of signal out }
 r := round(fr/31.25) ;
 sb := 0 ;  {No of sidebands on each side }
 smsqr := 0 ;
 for x := r-sb to r+sb do
   smsqr :=smsqr + sqr(fft_blk[x]) ;
 vrms_o := sqrt(2 * smsqr) ;

 {Rms value of noise}
 smsqr := 0 ;
 for x := 1 to r-sb-1 do
  smsqr :=smsqr + sqr(fft_blk[x]) ;
 for x := r+sb+1 to 127 do
  smsqr :=smsqr + sqr(fft_blk[x]) ;
 nrms  := sqrt(2 * smsqr) ;

 snr := 20 * ln(vrms_o/nrms) / ln(10) ;

 gain := vrms_o/vrms_i/256 ;

 calc[blocknum]^[1] := vdbm_i ;
 calc[blocknum]^[2] := snr ;
 calc[blocknum]^[3] := gain ;

end;
```

```
{*************************************************************}
{                          MAIN PROGRAM                       }
{*************************************************************}
begin
  for x := 1 to ntot do new(out_dta[x]) ;
  for x := 1 to mxblk do new(calc[x]);

  clrscr ;
  writeln('AVAILABLE OPTIONS :') ;
  writeln(' 1 : BCOD785') ;
  writeln(' 2 : BFULL785') ;

  repeat
   write('Enter your option : ') ;
   readln(option) ;
  until option in ['1','2'] ;

  case option of
   '1' :  filename := 'bcod785' ;
   '2' :  filename := 'bfull785' ;
  end; {case}

  writeln('Reading data from file ',filename) ;
  assign(infile,filename + '.dat') ;
  reset(infile) ;
  for blk := 1 to 50 do
    begin
      read(infile,out_blk) ;
      point := 1 + (blk-1)*256 ;
      for x := 0 to 255 do
       out_dta[x + point]^ := out_blk[x] ;
    end ;
  close(infile) ;

   for blocknum := 1 to mxblk do
    begin
     gen_in_blk(blocknum,in_blk);
     get_out_blk(blocknum,out_dta,out_blk) ;
     window(out_blk) ;
     fft(out_blk,fft_blk) ;
     calculate(blocknum,in_blk,fft_blk,calc) ;
    end ;

   filename := filename + '.cic' ;
   writeln('Writing data to file ',filename) ;
   assign(calcfile,filename) ;
   rewrite(calcfile) ;
    for x := 1 to mxblk do
     write(calcfile,calc[x]^) ;
   close(calcfile) ;

end.
```

```
{*********************************************************}
{                                                         }
{          PROGRAM TO PLOT RESULTS OF CODEC TESTS         }
{                                                         }
{*********************************************************}
program plotres ;

const mxblk = 785 ;

type   calcadr = array[1..mxblk]  of ^calcmat ;
       calcmat = array[1..3]      of  real ;


var blocknum,blk        : integer ;
    calc                : calcadr ;
    calcfile            : file of calcmat ;
    filename            : string[20] ;
    option              : char ;
    x,y                 : integer       ;
```

```
{*************************************************************}
{                  PROCEDURE TO PLOT SNR GRAPH                }
{*************************************************************}
procedure plot_snr(calc  :  calcadr) ;

const weigh = 0.9 ;

var xx,yy,xcrd,ycrd     : integer ;
    xcrdn,ycrdn         : integer ;

begin
  hires ;
  frame ;
  draw(0,180,639,180,1) ;
  draw(80,0,80,199,1) ;
  draw(210,180,210,92,1) ; plot(253,72,1) ;
  draw(340,48,600,48,1)  ; draw(600,48,600,180,1) ;
   for yy:=1 to 9 do
   begin
    ycrd := 180 - round(180*yy/9);
    draw(74,ycrd,86,ycrd,1) ;
   end;
  for xx := 1 to 8 do
   begin
    xcrd := 80 + round(520*xx/6) ;
    draw(xcrd,177,xcrd,183,1) ;
   end;
   for blocknum := 1 to mxblk-1 do
    begin
     xcrd  := 80 + round(520/60*(60+calc[blocknum]^[1])) ;
     xcrdn := 80 + round(520/60*(60+calc[blocknum+1]^[1])) ;
     ycrd  := 180 - round(180/45*(calc[blocknum]^[2]+weigh)) ;
     ycrdn := 180 - round(180/45*(calc[blocknum+1]^[2]+weigh)) ;
     draw(xcrd,ycrd,xcrdn,ycrdn,1) ;
    end;

  gotoxy(7,24) ;
  repeat until keypressed ;

end ;
```

```
{*************************************************************}
{                PROCEDURE TO PLOT GAIN GRAPH                 }
{*************************************************************}
procedure plot_gain(calc  : calcadr) ;

var xx,yy,xcrd,ycrd     : integer ;
    xcrdn,ycrdn         : integer ;
    gain1,gain2         : real    ;

begin
  clrscr ;
  hires ;
  frame ;
  draw(0,180,640,180,1) ;
  draw(167,0,167,33,1)  ; draw(167,33,253,33,1) ;
  draw(253,33,253,67,1) ; draw(253,67,626,67,1) ;
  draw(167,180,167,167,1)  ; draw(167,167,253,167,1) ;
  draw(253,167,253,133,1)  ; draw(253,133,626,133,1) ;
  draw(80,0,80,199,1) ;
  for yy:=0 to 6 do
   begin
    ycrd := round(yy*200/6) ;
    draw(74,ycrd,86,ycrd,1) ;
   end;
  for xx := 1 to 8 do
   begin
    xcrd := 80 + round(520*xx/6) ;
    draw(xcrd,177,xcrd,183,1) ;
   end;

   for blocknum := 1 to mxblk-1 do
    begin
     gain1 := 20 * ln(calc[blocknum]^[3]) / ln(10) ;
     gain2 := 20 * ln(calc[blocknum+1]^[3]) / ln(10) ;
     xcrd  := 80 + round(520/60*(60+calc[blocknum]^[1])) ;
     xcrdn := 80 + round(520/60*(60+calc[blocknum+1]^[1])) ;
     ycrd  := 100 - round(100*gain1/1.5) ;
     ycrdn := 100 - round(100*gain2/1.5) ;
     draw(xcrd,ycrd,xcrdn,ycrdn,1) ;
    end;

  gotoxy(7,24) ;
  writeln ;
  repeat until keypressed ;

end;
```

```
{*************************************************************}
{                        MAIN PROGRAM                         }
{*************************************************************}

begin

  for x := 1 to mxblk do
  new(calc[x]) ;

  clrscr ;
  writeln('AVAILABLE OPTIONS :') ;
  writeln ;
  writeln(' 1 : BJOD785') ;
  writeln(' 2 : BFULL785') ;
  writeln ;

  repeat
   write('Enter your option : ') ;
   readln(option) ;
  until option in ['1','2'] ;

  case option of
    '1' :  filename := 'bcod785.CLC' ;
    '2' :  filename := 'bfull785.CLC' ;
  end; {case}

  writeln(' ') ;
  writeln('Reading data from file ',filename) ;
  writeln(' ') ;
  assign(calcfile,filename) ;
  reset(calcfile) ;
   for x := 1 to mxblk do
    read(calcfile,calc[x]^) ;
  close(calcfile) ;


  plot_snr(calc) ;
  plot_gain(calc) ;


end.
```

```
Type
   WP = ^Wrec;
   Wrec = Record
      R          : Array[0 .. 511] of Double;
      Q          : Array[0 .. 511] of Double;
   End;

Var  BRAddress : Array[0 .. 1023] of Integer;
     W : WP;

{-------------------------------SP2FFT-------------------------------}
FUNCTION bitrev ;
VAR i,j1,j2,BitRevAddr:integer;
   BEGIN
      j1:=Address;
      BitRevAddr:=0;
      FOR i:=1 TO NoOfBits DO
         BEGIN
            j2:=j1 DIV 2;
            BitRevAddr :=BitRevAddr*2+(j1-2*j2);
            j1:=j2;
         END; (for)
         bitrev:=BitRevAddr;
END;{bitrev}
{-------------------------------SP2FFT-------------------------------}

Procedure FFT ;
Begin
   FastFT(Y1,Y2);
   YD!VN(Y2);
   Message(1, 'Direct FFT of '+RecNum('Y',Y1)+' now in
'+RecNum('Y',Y2));
   Y[Y2]^.RUnits   := 'FFT output real part';
   Y[Y2]^.QUnits   := ' " output Imaginary part';
End; { }
{-------------------------------SP2FFT-------------------------------}

Procedure FFTInitialise;
Var
   I : Integer;
   A : Double;
Begin
   Message (1,'Initialising FFT Twiddles ....');
   New(W);
   A :=2*PI/1024;
   For I := 512 to 1023 do
      BRAddress[I] := BitRev(I,10);
   For I := 0 to 511 do
      Begin
         W^.R[I] := cos(A*I);
         W^.Q[I] := -sin(A*I);
         BRAddress[I] := BitRev(I,10);
      End;
End;
{-------------------------------SP2FFT-------------------------------}
```

```
Procedure FastFT  ;
Var
   N,Ratio, BRI,BRJ,BRK,BRL, Stage, Stages, Span, Last, Group,
Groups, I,J,K,L,Kstep : Integer;
   Rt,Qt,Rp,Qp,RI,QI,RJ,QJ,RK,QK,RL,QL : real;
Begin
   N := Y[Y1]^.Num + 1;
   Y[Y2]^.Num := Y[Y1]^.Num ;
   Stages := WhatPower(N,2);
   Ratio := 10 - Stages;
   {First two stages}
   I := 0;
   While I < N do
         Begin
            J := I + 1;
            K := J + 1;
            L := K + 1;
            BRI := BRAddress[I] shr Ratio;
            BRJ := BRAddress[J] shr Ratio;
            BRK := BRAddress[K] shr Ratio;
            BRL := BRAddress[L] shr Ratio;
            RI  := Y[Y1]^.R[BRI] + Y[Y1]^.R[BRJ];
            QI  := Y[Y1]^.Q[BRI] + Y[Y1]^.Q[BRJ];
            RJ  := Y[Y1]^.R[BRI] - Y[Y2]^.R[BRJ];
            QJ  := Y[Y1]^.Q[BRI] - Y[Y1]^.Q[BRJ];
            RK  := Y[Y1]^.R[BRK] + Y[Y1]^.R[BRL];
            QK  := Y[Y1]^.Q[BRK] + Y[Y1]^.Q[BRL];
            RL  := Y[Y1]^.R[BRK] - Y[Y1]^.R[BRL];
            QL  := Y[Y1]^.Q[BRK] - Y[Y1]^.Q[BRL];
            Y[Y2]^.R[I] := RI + RK;
            Y[Y2]^.Q[I] := QI + QK;
            Y[Y2]^.R[J] := RJ + QL;
            Y[Y2]^.Q[J] := QJ - RL;
            Y[Y2]^.R[K] := RI - RK;
            Y[Y2]^.Q[K] := QI - QK;
            Y[Y2]^.R[L] := RJ - QL;
            Y[Y2]^.Q[L] := QJ + RL;
            I := I + 4;
         End; { First and Second stages}
```

```
        {Stages 3 to finish}
        Span := 4;
        Groups := N shr 3;   {N div 8}
        For Stage := 3 to Stages do
          Begin
            I := 0;
            KStep := Groups shl Ratio;
            For Group := 1 to Groups do
              Begin
                K := 0;
                Last := Span -1;
                L      := I + Span;   { Bottom of
                                        butterfly}
                Rt     := Y[Y2]^.R[I];
                Qt     := Y[Y2]^.Q[I];
                Y[Y2]^.R[I] := Rt + Y[Y2]^.R[L];
                Y[Y2]^.Q[I] := Qt + Y[Y2]^.Q[L];
                Y[Y2]^.R[L] := Rt - Y[Y2]^.R[L];
                Y[Y2]^ Q[L] := Qt - Y[Y2]^.Q[L];
                I      := I + 1;
                For  J := 1 to Last do
                  Begin
                    K := K + KStep; { Twiddle power}
                    L := I + Span;   { Bottom of
                                       butterfly}
                    Rt := Y[Y2]^.R[I];
                    Qt := Y[Y2]^.Q[I];
                    Rp := W^.R[K]*Y[Y2]^.R[L] -
                          W^.Q[K]*Y[Y2]^.Q[L]
                    Qp := W^.R[K]*Y[Y2]^.Q[L] +
                          W^.Q[K]*Y[Y2]^.R[L]
                    Y[Y2]^.R[I] := Rt + Rp;
                    Y[Y2]^.Q[I] := Qt + Qp;
                    Y[Y2]^.R[L] := Rt - Rp;
                    Y[Y2]^.Q[L] := Qt - Qp;
                    I := I + 1;
                  End;
                I  := I + Span;
              End;
            Span := Span shl 1;
            Groups := Groups shr 1;
          End; {3 to end}
End;
{=======================End of unit SP2FFT=======================}
```

REFERENCES

CCITT Recommendation G.712 (1984a) "Performance characteristics of PCM channels at audio frequencies." Red book III.3, 1984.

CCITT Recommendation G.714 (1984b) "Seperate performance characteristics for the send and receive sides of PCM channels applicable to 4-wire voice-frequency interfaces." Red book III.3, 1984.

CCITT Recommendation O.131 (1984c) "Specification for a quantisation distortion measuring apparatus using a pseudo-random test stimulus. Red book IV.3, 1984.

CCITT Recommendation G.711 (1984d) " Pulse code modulation (PCM) of voice frequencies." Red book III.3, 1984.

CCITT Recommendation G.223 (1984e) "Assumptions for the calculation of noise on hypothetical reference circuits for telephony." Red book, Volume III, Fascicle III.2, 1984.

CORDT,W. und HAHN,H. (1982) "Rechnergestuetztes verfahren zur *analyse der daempfungs- und verzerrungseigerschaften von PCM* systemen." Bd. 35 Heft 5, 1982.

DEBBO,G.A. (1984) "A Proposed test system and strategy for production testing of PCM codecs." MSc (Eng) Dissertation, University of the Witwatersrand, 1984.

HANRAHAN, H.E. (1984a) "Relationship between regression-based and DFT-based PCM codec gain and SNR measurements." Research report, Johannesburg, Electrical Engineering Department, University of the Witwatersrand, 1984.

HANRAHAN, H.E. (1984b) "Testing of PCM codecs in a Transposed configuration." Research report, Johannesburg, Electrical Engineering Department, University of the Witwatersrand, 1984.

HANRAHAN, H.E. (1984c) "Performance evaluation of PCM codecs in digital subscriber loops." University of the Witwatersrand, 1984.

STEIGLITZ, K. (1974) "An introduction to discrete systems", New York, John Wiley & Sons, 1974.

DEBBO,G.A. (1984) "A Proposed test system and strategy for production testing of PCM codecs." MSc (Eng) Dissertation, University of the Witwatersrand, 1984.

HANRAHAN, H.E. (1984a) "Relationship between regression-based and DFT-based PCM codec gain and SNR measurements." Research report, Johannesburg, Electrical Engineering Department, University of the Witwatersrand, 1984.

HANRAHAN, H.E. (1984b) "Testing of PCM codecs in a Transposed configuration." Research report, Johannesburg, Electrical Engineering Department, University of the Witwatersrand, 1984.

HANRAHAN, H.E. (1984c) "Performance evaluation of PCM codecs in digital subscriber loops." University of the Witwatersrand, 1984.

STEIGLITZ, K. (1974) "An introduction to discrete systems", New York, John Wiley & Sons, 1974.