Feasibility Study of a Hardware Implementation of a 4.8 kb/s RELP Speech coder

Peter Kabal

INRS-Télécommunications
a/s Recherches Bell-Northern Ltée
3, Place du Commerce
Ile des Soeurs, Qué.
H3E 1H6

INRS-Telecommunications Technical Report No. 81-08

May 1981

Final Report to Communications Research Centre

Contract 20SU 36001-0-3034

ABSTRACT

This report investigates the feasibility of a hardware implementation of a speech coder based on Residual Excited Linear Production (RELP) at 4.8 kb/s. To this end, the basic RELP algorithm has been restructured and simplified to be compatible with real-time processing. In addition, the computations have been implemented with integer arithmetic using 16 bit precision, augmented with the judicious use of double precision accumulation. An architecture based on a microprocessor supplemented with a peripheral processor built around a high speed multiplier/accumulator is proposed. This arrangement can be the basis for a simple, cost effective and flexible implementation of a hardware RELP coder.

SOMMAIRE

Le but de ce rapport est d'étudier la possibilité de réaliser un codeur de la voix opérant en temps réel à 4.8 kb/s. Le principe de ce codeur est basé sur la prédiction linéaire excitée par un signal résiduel (PLER). L'algorithme a été simplifié et restructuré pour réduire la complexité de réalisation. De plus, le calcul s'effectue avec 16 bits de précision sauf dans certains operations d'accumulation, on utilise une double précision. On propose de réaliser ce codeur avec un microprocesseur central aidé par un processeur périphérique construit autour d'un multiplicateur/accumulateur à haute vitesse. Cette architecture permet une réalisation simple, flexible et peu coûteux de ce codeur PLER.

ACKNOWLEDGEMENTS

The contributions of many colleagues to this project are gratifully acknowledged; J. Boyd for helping produce a RELP algorithm suitable for real-time implementation, R. Pinnell for his work on the hardware aspects of the coder, D. Stevenson for sharing his expertise on coding, and P. Mermelstein for helping to steer the project.

TABLE OF CONTENTS

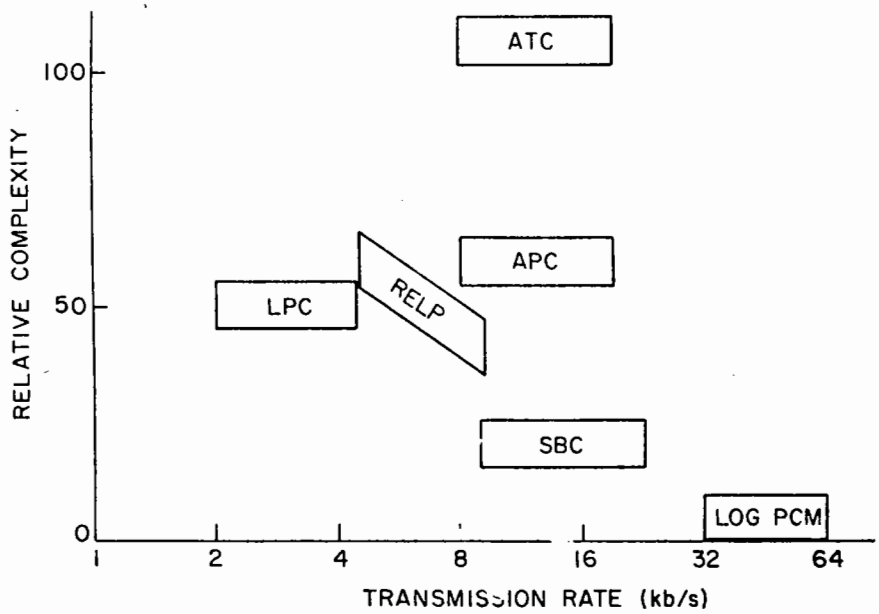
1. Introduction

The recent revolution in integrated circuit technology has spurred new interest in real-time implementations of speech coders. Prior to these developments, only the simplest speech coding algorithms could be realized in hardware. The advent of high speed large and very-large scale integrated circuits (LSI and VLSI) makes feasible the implementation of sophisticated algorithms which, apart from having all the attendant benefits of digital circuitry, utilize channel capacity more efficiently. In this chapter, we briefly discuss speech coding algorithms in an attempt to give the reader an understanding of the origins and objectives of this project.

1.1 Digital Speech Coding

Digital transmission facilities are the option of choice for new systems since are more reliable, flexible and economical than analog systems. Digital speech coders serve as the interface between end-users and digital transmission links. The choice of a coding algorithm is dependent on many factors. In particular, the tradeoffs available can be visualized as spanning 3 dimensions: complexity (or cost), transmission rate and speech quality (see Fig. 1-1). At data rates above 10 kb/s, bit rate reductions can be achieved by increasing the complexity of the coding schemes while maintaining relatively good speech quality. At rates below 10 kb/s, some speech quality degradation is inevitable. Often the use of low bit rate coders is dictated by the need for a bit rate compatible with existing analogue transmission facilities or the need to efficiently use digital facilities. For instance, low bit rate digital speech is used over analog voice

APC - Adaptive Predictive Coding
ATC - Adaptive Transform Coding
LPC - Linear Predictive Coding
RELP - Residual-Excited Linear Predictive Coding
PCM - Pulse-Code Modulation
SBC - Sub-Band Coding



a) Complexity (Cost) vs. Rate

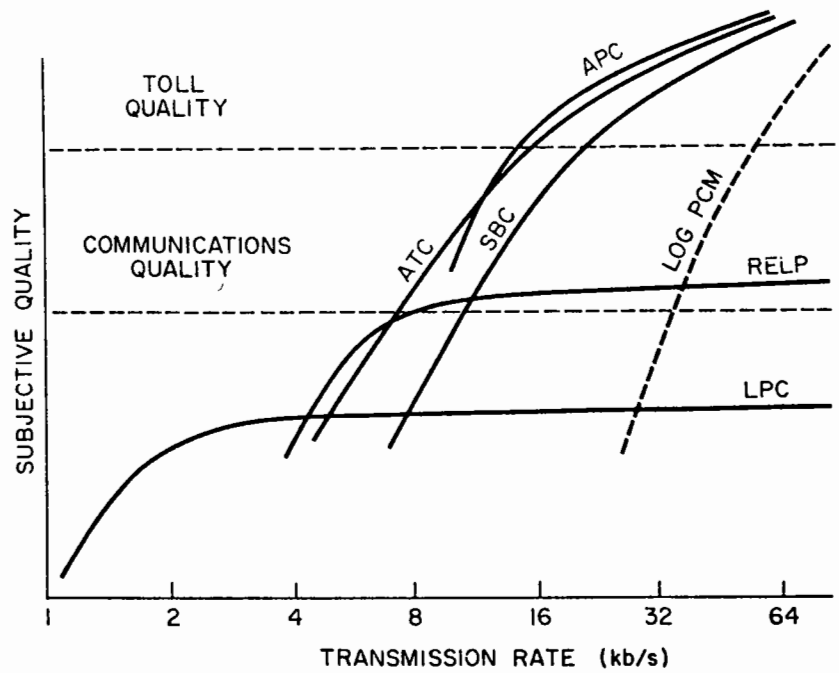


Figure 1-1 Complexity (Cost) / Rate / Quality Trade-offs for Speech Coders

circuits for reasons of security or to achieve cost reductions. For the first purpose, digital encryption is used for secure or private communication. The second application involves the multiplexing of several digital streams onto a single analog voice circuit.

Speech coding schemes can be separated into two main classes: waveform coders and analysis/synthesis coders. Coders in the former category attempt to replicate the input waveform at the receiving end. The complexity of waveform coding runs the gamut from pulse code modulation (PCM) up to sophisticated frequency domain techniques such as adaptive transform coding (ATC) [1]. High quality speech can be achieved with transmission rates as low as 10 kb/s using the more complex types of waveform coders; whereas the simpler schemes require higher rates.

Analysis/synthesis coders attempt to model the production of speech. The transmitter analyzes the input speech to produce a set of parameters each of which is digitized and sent to the receiver. At the other end the speech signal is synthesized using the received parameters. By employing a model appropriate to speech production, highly intelligible speech can be reproduced from a very compact parameterization of the input speech. Unfortunately, problems occur when the input speech deviates from the assumed model. Generally, these coding schemes are not robust to atypical speakers (e.g., high pitched individuals) or to speech corrupted by background acoustical noise (e.g., office noise or background noise in vehicules). Amongst analysis/synthesis coders, the best known technique is that of linear predictive coding (LPC) [2]. LPC uses a straightforward algorithm to produce highly intelligible speech at low transmission rates (2.4 kb/s

and below).

The speech coding technique explored in this report combines aspects of both waveform and analysis/synthesis coding. Because the target transmission rate for this project, 4.8 kb/s, does not allow for successful waveform coding and because the system is to be operated in potentially noisy environments, it is appropriate to employ some combination of the two types of coders. One such combination is residual-excited linear predictive coding (RELP) [3]. As in conventional LPC, RELP models the spectral envelope of the speech; however instead of using a two state excitation model, it transmits additional waveform information. This results in a coder which is more robust to noise and has a less synthetic quality than LPC, while still retaining a low transmission rate.

1.2 RELP

In the most basic form of LPC, speech production is modelled by an excitation source acted upon by a linear filter. The model assumes an all-pole filter (auto-regressive model) and an excitation signal which is either a periodic train of pulses or a noise signal. The filter coefficients are determined by solving a set of simultaneous linear equations which produce a minimum mean square error fit to the data. Efficient techniques for performing this analysis are available. To determine the type of excitation, it is necessary to determine whether segments of speech are voiced (periodic excitation) or unvoiced (noise exitation). Associated with voiced speech is a pitch corresponding to the vibration rate of the vocal folds. The tracking of the pitch frequency is one of the more difficult aspects of LPC.

Spurious background noise and transmission errors can render LPC completely incomprehensible. The former plays havoc with most pitch prediction algorithms by incorrectly determining a voiced/unvoiced segment of speech or by incorrectly estimating the pitch (e.g. pitch doubling). RELP avoids the hard decisions implied by the LPC model.

Instead of transmitting parameters to characterize the excitation source as in LPC, RELP calculates and transmits a portion of the residual signal. The residual signal is the correct excitation signal for the synthesizer using the filter coefficients of the model. The residual is determined by inverse filtering the speech signal, i.e. filtering using the inverse filter. This strategy results in better performance quality in non-ideal environments at the expense of an increased bit rate [3]. At a transmission rate of 4.8 kb/s, the residual must be characterized very crudely- in fact only the low-frequency components are sent. The decoder must reconstruct a suitable full residual using a high frequency regeneration scheme.

The quality of the speech produced by LPC and RELP differ. LPC produces highly intelligible but synthetic sounding speech (if the input speech is free of background noise). RELP on the other hand, produces more natural sounding but slightly more noisy speech.

1.3 Project Background

The overall objective of this project is to develop speech coders for narrow-band field communications. Since the coder is to be operated in the field, the system must be robust to transmission bit errors and noisy environments.

The project has been divided into separate tasks. The first phase, to determine if the intelligibility and quality criterion can be obtained at 4.8 kb/s, has been completed and documented by Mermelstein and Nakatsui [3]. Their results confirm that the RELP algorithm is significantly more robust to transmission errors and background noise than LPC.

The second phase of the project is to determine the feasibility of implementing the RELP coder in hardware. This report documents this study. The problems considered are means to reduce the complexity of the algorithm without reducing the overall quality. The effects of finite precision integer computations as required in a hardware implementation are also considered. The final aspect of this study is the recommendation of a signal processing architecture that combines versatility and ease of implementation with a moderate cost. The results of these inquiries will be discussed in detail in subsequent sections of this report.

2. RELP Algorithm

The basic algorithm for RELP coding will be described in this chapter. In addition, modifications to achieve a structure conducive to a real-time implementation are outlined. These modifications are of two types. First the algorithm has been restructured so that it becomes a hierarchical system utilizing common computational modules. This in general does not change the outcome of the calculations but does point to a hardware configuration which is built around a special purpose signal processor to implement one or more of the modules. The second type of modification is intended to reduce the computational complexity of the algorithm.

The problem of using integer arithmetic for the computations will be discussed in detail in the next chapter. However, some of the algorithm modifications suggested here were made with the need for integer arithmetic in mind.

2.1 Overview

The basic RELP algorithm that forms the basis for the work reported here is that used by Mermelstein and Nakatsui [3]. Their coder in turn had its genesis in one described by Un and MaGill [4]. The functional blocks of the coder and decoder are shown in Fig. 2-1 and Fig. 2-2 respectively. The coder calculates a set of filter (predictor) coefficients for each frame of data. The input speech signal is inverse filtered using a filter based on these coefficients to produce the residual signal. The residual signal is then low-pass filtered and digitized using a sub-band coding technique. The parameters that are transmitted by the coder include both the filter coefficients and the

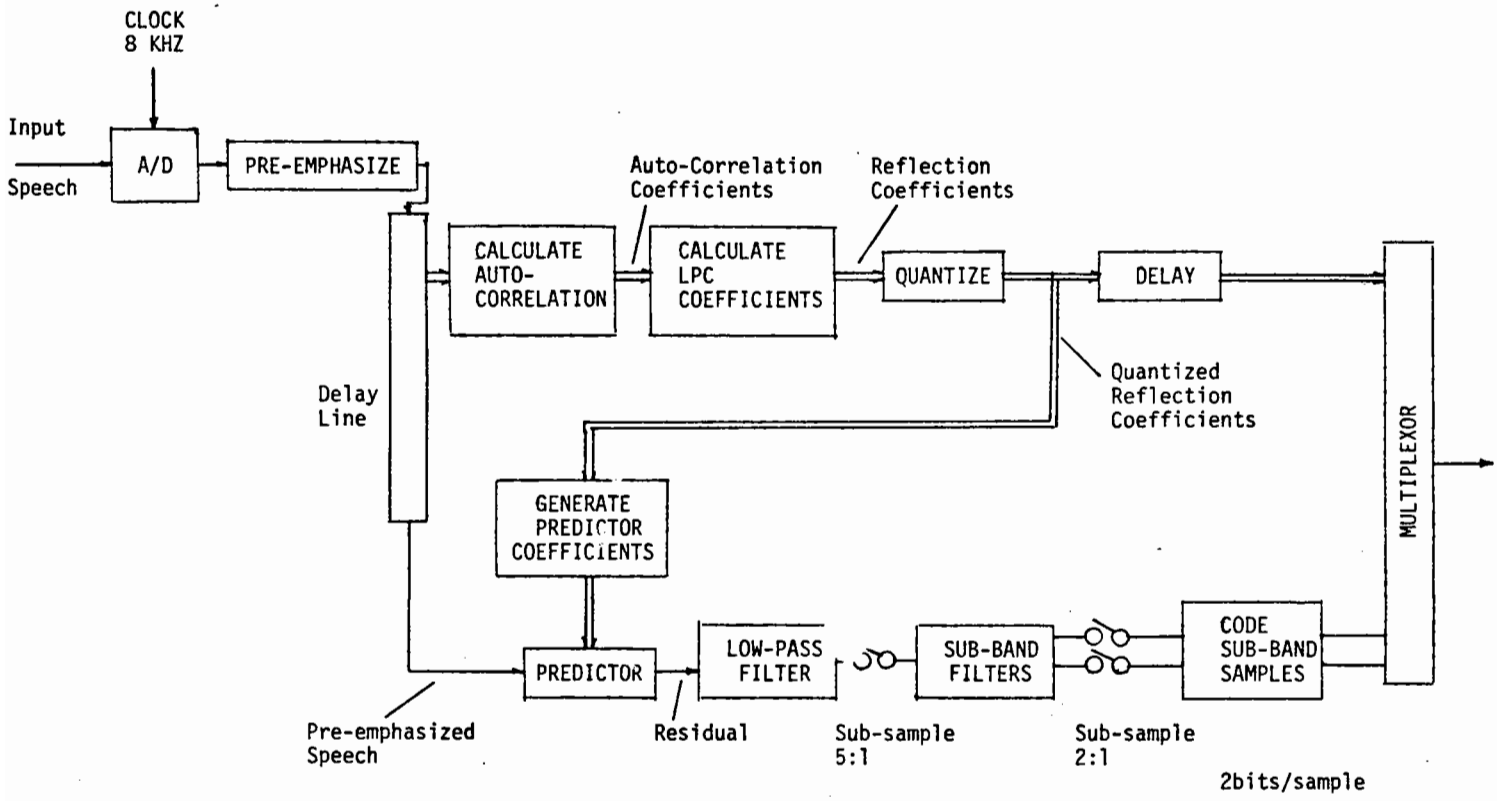


Figure 2-1 Block Diagram of a RELP Coder

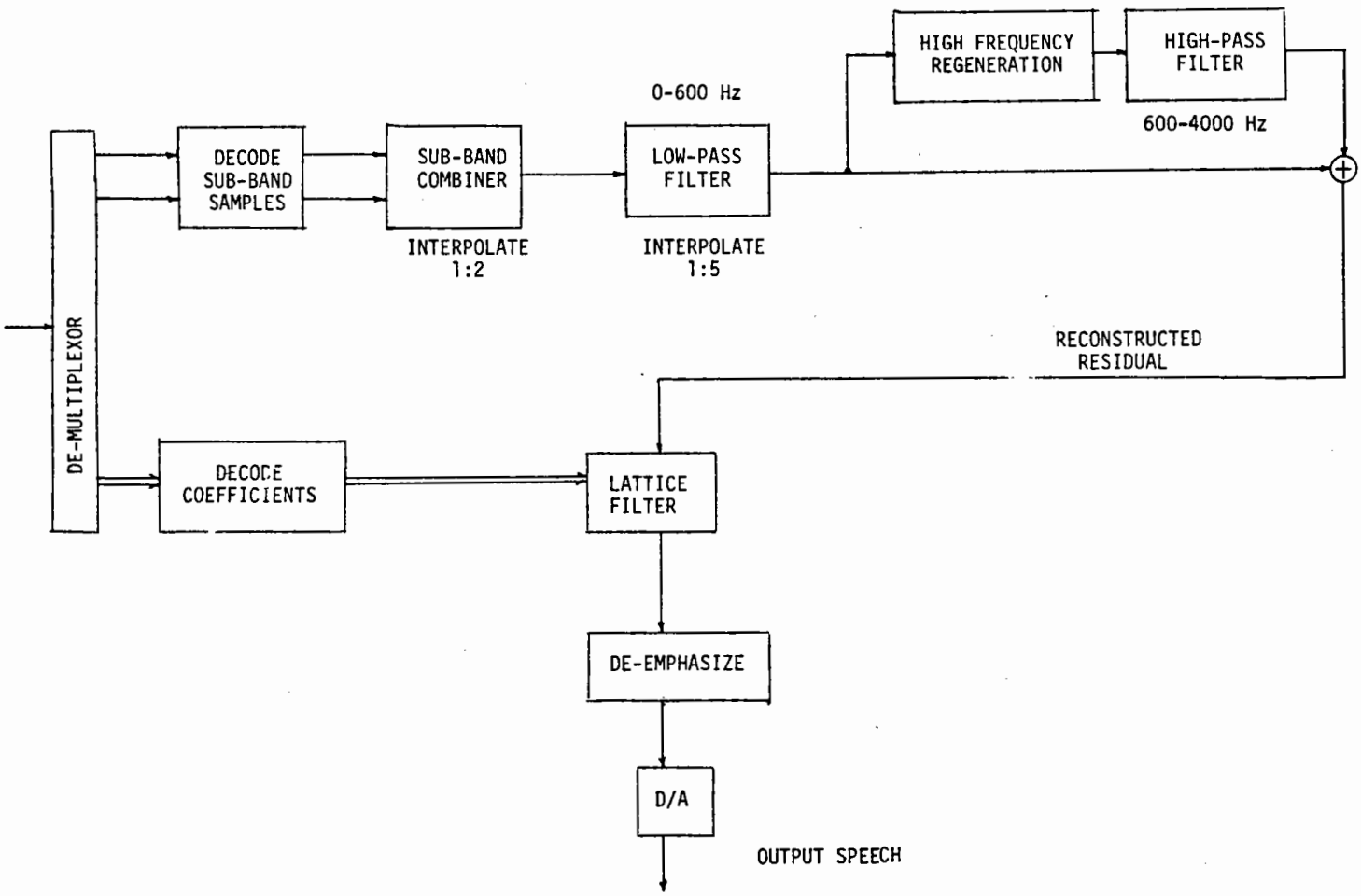


Figure 2-2 Block Diagram of a RELP Decoder

coded low-pass residual.

The decoder reconstructs the low-pass residual from the received data. The missing high frequency components are regenerated and added to the low-pass residual to form an excitation signal. This excitation signal drives the synthesis filter defined by the filter coefficients to form the output speech signal.

2.2 RELP Coder

The steps in the coding of a speech signal will now be described in more detail.

2.2.1 Pre-Emphasis

Before processing the input signal is pre-emphasized to boost the high frequency components. This helps compensate for the natural drop-off at high frequencies in speech. Pre-emphasis ensures that the higher formants are adequately represented and avoids ill-conditioning of the auto-correlation analysis [2]. The pre-emphasized signal is given by

$$s_p(n) = s(n) - a\, s(n-1), \qquad (2\text{-}1)$$

where s(n) is the input signal and a is the pre-emphasis factor. The pre-emphasis factor is near unity, here 0.95. Fig. 2-3 shows the frequency response of the pre-emphasis operation considered as a filter.

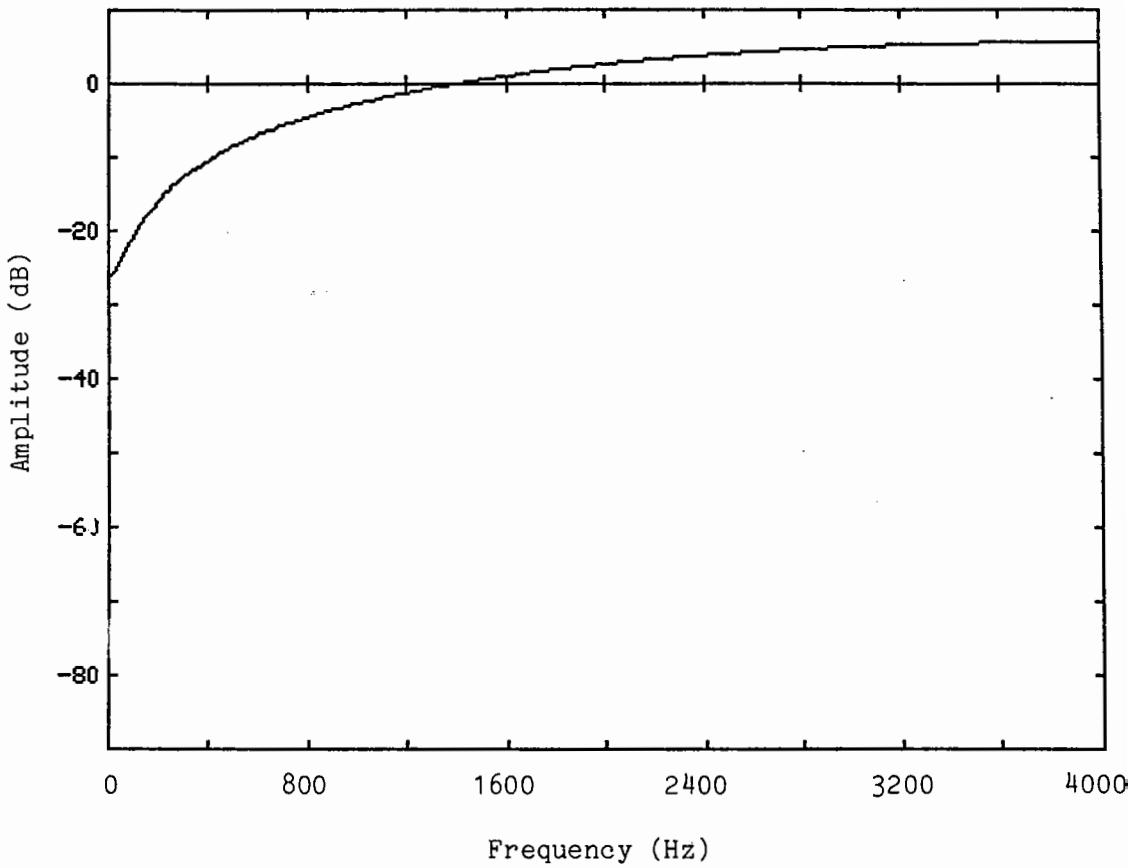


Figure 2-3 Frequency Response of the Pre-Emphasis Filter

2.2.2 Analysis

This stage produces the filter coefficients corresponding to a frame of data. The analysis frames are overlapped so that the resultant residual signal will not exhibit sharp discontinuities at the frame boundaries. In the present case the frames are updated 50 times per second, corresponding to 160 new samples per frame. However, the full analysis frame is 204 samples in length, overlapping 22 samples on each end with adjacent frames.

The analysis consists of five steps.

a) Calculate the auto-correlation coefficients for the analysis frame (9 values).
b) Lag window the auto-correlation coefficients.
c) Solve for the reflection coefficients (8 values) using a recursive algorithm.
d) Quantize the reflection coefficients.
e) Transform the quantized reflection coefficients to the equivalent set of filter coefficients.

In a conventional analysis, the data is windowed before calculating the auto-correlation coefficients. The procedure adopted here, termed lag windowing [5,6], reduces the computational complexity without altering the speech quality. Instead of 204 multiplications per frame, lag windowing requires only 8 multiplications for the windowing operations. The quality of the speech produced using lag windowing is indistinguishable from the original method if the residual is coded. With no coding, a slight change in tonality can be noted, but the quality rating remains the same.

The auto-correlation coefficients are calculated as

$$r(k) = \sum_{n=0}^{N-k-1} s_p(n)\, s_p(n+k), \qquad (2\text{-}2)$$

where k takes on values from 0 to K (here 8) and N (here 204) is the number of samples in a frame. The calculation of the auto-correlation for a sequence of data is closely related to convolution. The correlation of two sequences is equivalent to the convolution of one sequence with the other sequence time reversed. In this case

$$r(k) = \sum_{n=0}^{N-k-1} s_p(n)\, s'_p(k-n), \qquad (2\text{-}3)$$

where $s'_p(n)$ is a time reversed sequence,

$$s'_p(-n) = s_p(n). \qquad (2\text{-}4)$$

This identity between correlation and convolution has been shown to emphasize that a filtering or convolution module can also be used to calculate the auto-correlation coefficients in the analysis stage.

The conventional Durbin recursion used to calculate the reflection coefficients has been replaced by a recently described algorithm [7]. This algorithm uses a new set of intermediate variables which have magnitude less than unity. Since the number of operations is the same as that for the Durbin recursion, this variation has been adopted to facilitate conversion to integer based arithmetic. In a floating point implementation, the results are identical to the Durbin recursion.

The solution of the auto-correlation equations is obtained in terms of the reflection coefficients. These have the advantage of lying in the range -1 to +1 for stable filters. The quantizer can be then designed such that the quantized reflection coefficients also are guaranteed to represent a stable configuration.

In order to reduce the computational complexity, the quantizer is specified in terms of pre-computed quantizer levels. These have been designed based on log-area ratio quantization. The log-area ratio transformation produces a set of coefficients that have a uniform sensitivity over their ranges [2]. In the present implementation, instead of transforming the reflection coefficients to log-area ratios and then using a uniformly spaced quantizer, the equivalent non-uniform quantizer which acts directly on the reflection coefficients has been used. A binary search procedure is used to efficiently locate the index of the the quantizer level corresponding to the coefficient value. It is this index that is sent to the decoder. The coder also uses this index to form the quantized reflection coefficients. It is the quantized values that are used to define the inverse filter. The use of the quantized values ensures that the analysis filter will be a true inverse to the synthesis filter used in the decoder. The number of quantizer levels varies according to the coefficient being quantized as shown in Table 2-1. The range of each quantizer had been previously been chosen to give good results for a number of speakers.

The quantized reflection coefficients are transformed using a recursive procedure to the equivalent set of predictor coefficients for the inverse filter.

Coefficient	No. Levels	Range
K_1	32	(-0.9728,0.7677)
K_2	16	(-0.3721,0.8902)
K_3	16	(-0.8184,0.5677)
K_4	8	(-0.3006,0.7647)
K_5	4	(-0.4668,0.3822)
K_6	4	(-0.2022,0.6353)
K_7	2	(-0.2586,0.4079)
K_8	2	(-0.1651,0.5806)

Table 2-1 Reflection Coefficient Quantizers

2.2.3 Inverse Filtering

The inverse filter is implemented as a finite impulse response (FIR) filter. An alternative would have been to use a lattice form which uses the reflection coefficients directly, eliminating the need to transform them to the set of predictor coefficients. The lattice filter requires an additional multiplication per coefficient per sample over that of the FIR implementation [8]. This additional computation overwhelms the additional computations needed for the coefficient transformation which is done only once per frame. The use of an FIR filter also permits the use of a common module which can be used in subsequent filtering operations. a convolution module can be used here.

2.2.4 Low-pass Residual

An FIR low-pass filter is used to filter the residual. This operation is combined with a sub-sampling (decimation) of the output samples (here 5:1) to produce 32 low-pass residual samples per frame. Appendix A contains tutorial material that elaborates on filtering and sub-sampling. The choice of an FIR filter allows the reduction by a factor of five in the number of filtering operations. In fact, the filter used is a linear phase FIR structure. This implies that the filter coefficients have an even symmetry with respect to the mid-point of the filter. This symmetry can reduce the number of multiplications by an additional factor of two. However, the number of additions remains the same. It is not anticipated that the extra control structure needed to take advantage of this symmetry is warranted.

2.2.5 Sub-Band Coding

The low-pass residual is sub-band coded. This operation involves separating the signal into two distinct frequency bands. The sub-bands are then coded independently. Sub-band coding is advantageous when the signal is coarsely quantized. While the quantization noise in each sub-band is correlated with the signal in that sub-band, the overall quantization noise is perceived to be nearly white. In addition, the quantizers in each sub-band can be made to adapt to the signal levels in the respective sub-band. This has the effect of masking the quantization noise. In the case of RELP coding at 4.8 kb/s, sub-band coding is a key step in making the quantization noise that is superimposed on the residual acceptable.

The sub-band filters are implemented using quadrature mirror filters. These filters are discussed in detail in Appendix A. This implementation is very computationally efficient since both sub-bands can be generated essentially with a single filtering operation. An additional 2 to 1 saving is available since each sub-band is sub-sampled by a factor of 2.

There are 16 pairs of sub-band samples in each frame. Each sub-band stream is coded using a 4 level (2 bit) adaptive quantizer. The step size of each quantizer is adapted using the step-size multiplier scheme proposed by Jayant [9]. An additional feature is the ability to change from a mid-rise characteristic (4 level quantizer) to a mid-tread quantizer (3 level quantizer) for small signal levels [10]. This latter form has a zero output level, which reduces the idle channel quantization noise when the input signal is very small.

2.3 Transmission Format

The transmission format for a block of data is shown in Fig. 2-4. The data corresponding to a frame of speech has been kept intact as a block. The sub-band coded residual contributes 64 bits per frame. The reflection coefficient quantization requires an additional 20 bits per frame.

In the original version of RELP, an additional parameter representing the energy in the untransmitted high frequency components used up additional transmission capacity. The current version has 12 uncommitted bits out of the frame of 96 bits (corresponding to 4800 b/s). These may be used in a number of different ways.

a) Framing Information.

For synchronization, part or all of the spare 12 bits per frame can be used to mark the beginning of a frame. However, the use of all 12 bits for this purpose may be extravagant. LPC coders have been implemented using robbed frame synchronization. In this scheme frames corresponding to low levels of speech or silence are used for synchronization. This strategy can also be adapted to RELP.

b) Error Protection.

The unallocated bits can be used to provide redundancy for error protection. Both error correction and detection are possible. For instance, in an error detection scheme, the detection of an error condition can be used to signal the fact that for that frame, data from the previous frame should be used.

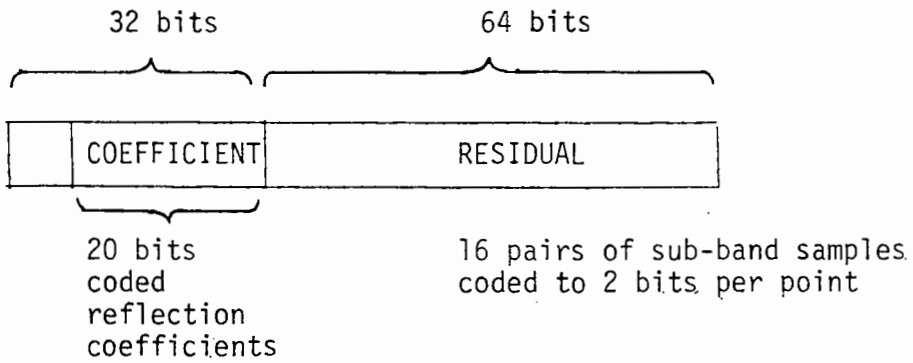


Figure 2-4 Data Format for Transmission

A cyclic code (92,84,4) based on an expurgated Hamming code (127,119,4) could be used to effect single error correction and double error detection for the whole block of data [11]. Double error correction is not possible for the whole block. However a more reasonable scheme would protect only a subset of the bits to keep the complexity within bounds. The use of any error protection may be unwarranted in most transmission environments since RELP is very robust with respect to transmission errors [3]

c) More Precise Representation of the Coefficients

A more precise representation of of the reflection coefficients can improve the quality of the reproduced speech. The energy in the residual would decrease, enabling better quality reproduction of the speech. However, the best improvement that can be expected is slight (see Section 2.5.4).

2.4 RELP Decoder

The decoder has available to it the quantized reflection coefficients and the coded sub-band samples. The processing steps which use this data to produce the output speech signal are described in more detail in the following sub-sections.

2.4.1 Sub-Band Reconstruction

The decoder tracks the sub-band quantizer step sizes that are used by the coder. This step size information is then used along with the quantizer output level index, which is part of the transmitted data, to form the sub-band samples. The sub-band samples are combined using

quadrature mirror filters to form an approximation to the sub-sampled low-pass residual signal. By forming sub-filters, consisting of the odd-numbered and even-numbered coefficients of the quadrature mirror filter, computational savings are realized (Appendix A).

The 16 pairs of sub-band samples in each frame are combined in this step to form 32 low-pass residual samples.

2.4.2 Residual Interpolation

A low-pass filter is used to form an interpolated low-pass residual with 160 samples per frame. The use of an FIR interpolating filter results in a computationally efficient procedure. As described more fully in Appendix A, sub-filters (5 in this case) are used in a round-robin fashion in this operation.

2.4.3 High Frequency Regeneration

The missing high-frequency components are generated by passing the low-pass residual through an absolute value function non-linearity. Double difference filters are used before and after the non-linearity to help produce spectrally balanced high frequency components. The regenerated components are high-pass filtered before they are added to the low-pass residual. The high-pass filter is the complementary filter to the low-pass filter used in the coder (except for a gain factor). For computational simplicity, the double difference filter and the high-pass filter are combined into a single filter.

For voiced speech, which is quasi-periodic, the harmonic structure in the low-pass residual is continued into the regenerated high frequency components. For unvoiced speech, the noise like characteristic is likewise continued into the high frequency portion.

2.4.4 Synthesis

The synthesizer is an all-pole filter. Because of the feedback inherent in such a filter, the numerical stability of the results depends on the form of the implementation. Lattice forms have a low sensitivity to parameter quantization but involve a more complicated structure [8]. A direct form implementation is more sensitive to parameter changes but involves fewer computations. This form was chosen because it allows the use of a convolution processor. Since the coefficients for the direct form are the linear prediction coefficients, a transformation from the reflection coefficients must be made once per frame.

The filter is implemented as shown in Fig. 2-5. In this form, an FIR filter is imbedded in the feedback path. Even with floating point computations, a small but measurable difference was evident between the output of the lattice form of the synthesizer and the direct form of the synthesizer. This small difference was inperceptable in the output speech.

The difference in the outputs occurs when the pole positions approach the unit circle. The pole positions could be scaled radially inward to avoid this problem. Indeed such a technique has been suggested for LPC synthesizers to improve the perceived quality of the speech by reducing the formant peaks. This effect is pronounced in LPC

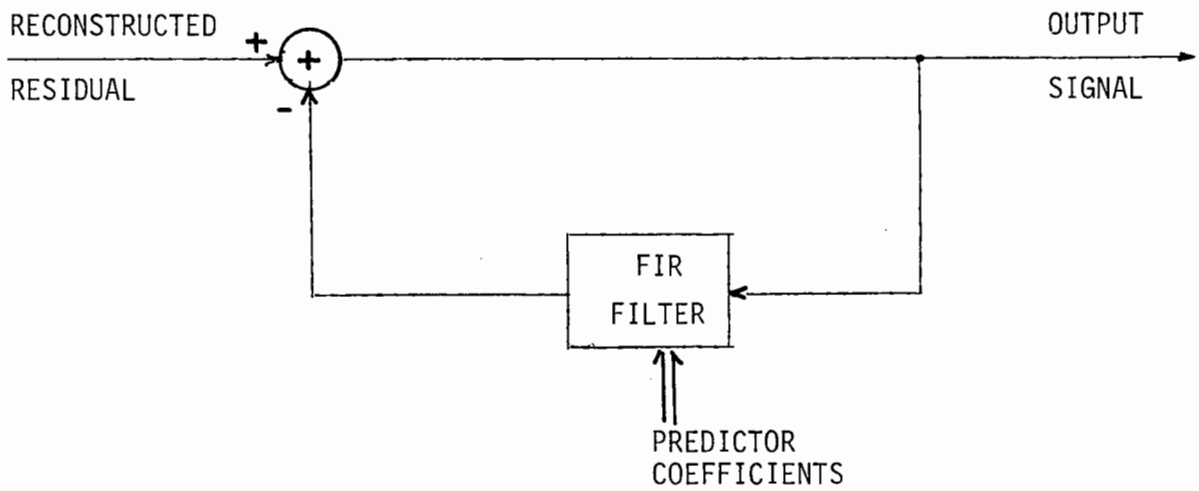


Figure 2-5 Direct Form Implementation for the Synthesis Filter

because of the artificial excitation signal. Due in part to the low order of the filter, all indications are that the direct form of the synthesizer without further modification is adequate for RELP.

2.4.5 De-Emphasis

The last step in the production of the output speech waveform is to de-emphasize the output signal. This operation uses a recursive filter which is the inverse of the pre-emphasis filter at the coder,

$$y_d(n) = y(n) + a\, y_d(n-1) , \qquad (2\text{-}5)$$

where y(n) is the output of the synthesizer and a is the pre-emphasis factor.

2.5 Algorithm Design Verification

Within the constraints of the RELP algorithm as described above, a great deal of latitude is possible. Of concern for this study is the choice of parameters such as filter lengths, which affect the computational complexity. In addition, changes that affect speech quality were also investigated.

2.5.1 Lag Windowing

The use of lag windowing of the auto-correlation coefficients has already been discussed. As noted earlier this change reduces the computational load without affecting the quality of the reconstructed speech.

2.5.2 Residual Energy Calculation

In an earlier version of the RELP algorithm, the ratio of the energy in the high frequency components of the residual to the energy in the low frequency components of the residual was calculated and transmitted to the decoder. This information was used in the decoder to scale the reconstucted high frequency components. However, since the residual has been "whitened" by the inverse filter, this scaling information produces little benefit. In fact, with quantization of the residual, there is no difference in the quality of output speech when the results of using this scaling information and using a fixed ratio are compared. Removing this calculation also reduces the overall coding delay since this ratio was averaged over a frame before transmission. The fixed gain in the high frequency regeneration branch was adjusted to give a good spectral balance between high and low frequencies in the resulting speech.

2.5.3 Coefficient Interpolation

The reflection coefficients calculated for a frame are used for the duration of that frame. An earlier version of RELP interpolated the reflection coefficients in an attempt to reduce frame boundary discontinuities. This effort was not rewarded with any perceptible difference in the speech quality.

2.5.4 Coefficient Quantization

Two experiments were conducted with the coefficient quantization. In the first, the 8 reflection coefficients were quantized with a large number of quantizer levels. Only a very small improvement in speech quality was realized. In the second experiment, the order of the analysis was increased to 9. Again only a small improvement was noted. These results were taken to mean that the quantization of the reflection coefficients is not a limiting factor of the quality of the output speech. This also indicates that little benefit will accrue from the allocation of additional transmission capacity to the quantization of the reflection coefficients.

2.5.5 Low-Pass Filter

The low-pass filter used in the RELP coder is a 31 tap linear phase FIR filter with a response as shown in Fig 2-6. It was designed using a design program based on [12]. A very sharp high order filter results in superior speech quality in the absence of quantization of the residual. This manifests itself as a fuller sound, since more of the residual is being transmitted. However, with quantization of the residual, this more nearly ideal filter produced a rougher quality in the speech than did the original filter.

Experiments with filters with fewer than 31 coefficients also produced inferior quality speech. This indicates that the present filter is well suited for the application. Additionally, the computational load attributable to this filter is not a large fraction of the overall computational complexity.

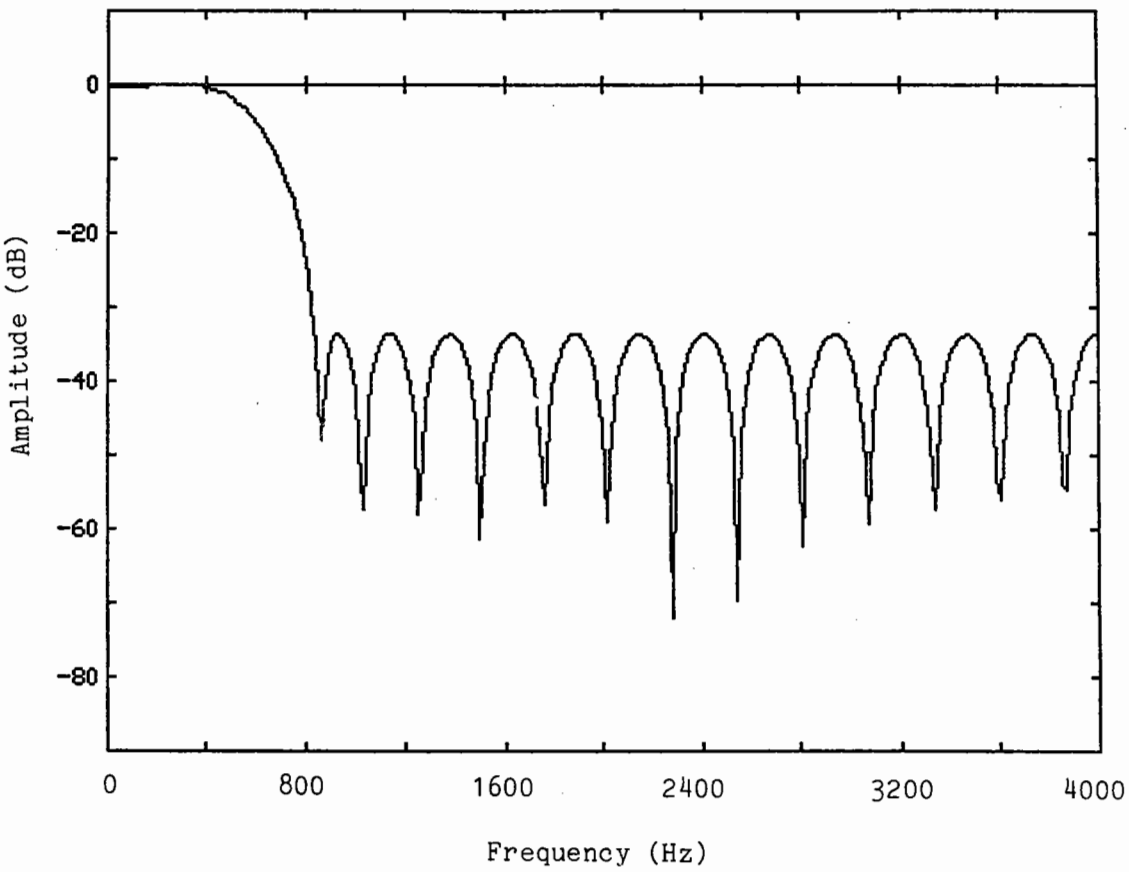


Figure 2-6 Frequency Response of the Coder Low-Pass Filter

2.5.6 Sub-Band Coders

The frequency responses of the quadrature-mirror filters are shown superimposed in Fig. 2-7. The quadrature-mirror filters were designed using an optimization process to maintain a flat frequency response across the sub-band coder and decoder [13]. The filter order was selected to give a reasonably sharp cut-off. Since these filters do not contribute significantly to the overall computation rate, no further optimization was attempted.

The major source of degradation in the system is due to the coding of the residual with only 2 bits per sample. The coding algorithm uses a quantizer step size adaptation scheme in which adaptation occurs independently for the two sub-bands. The adaptation constants used for the quantizers tread a fine line between sluggishness in following step changes and overshooting sudden changes. Considerable effort was put in to try to obtain a scheme which would allow some sharing of the step size information between the two quantizers. The modified scheme allows large step size changes when both sub-bands are changing in the same direction and smaller ones otherwise. This scheme did improve quality somewhat. However, most of the gain could also be achieved within the present scheme by modifying the step size adjustment parameters. The additional gains do not warrant the change.

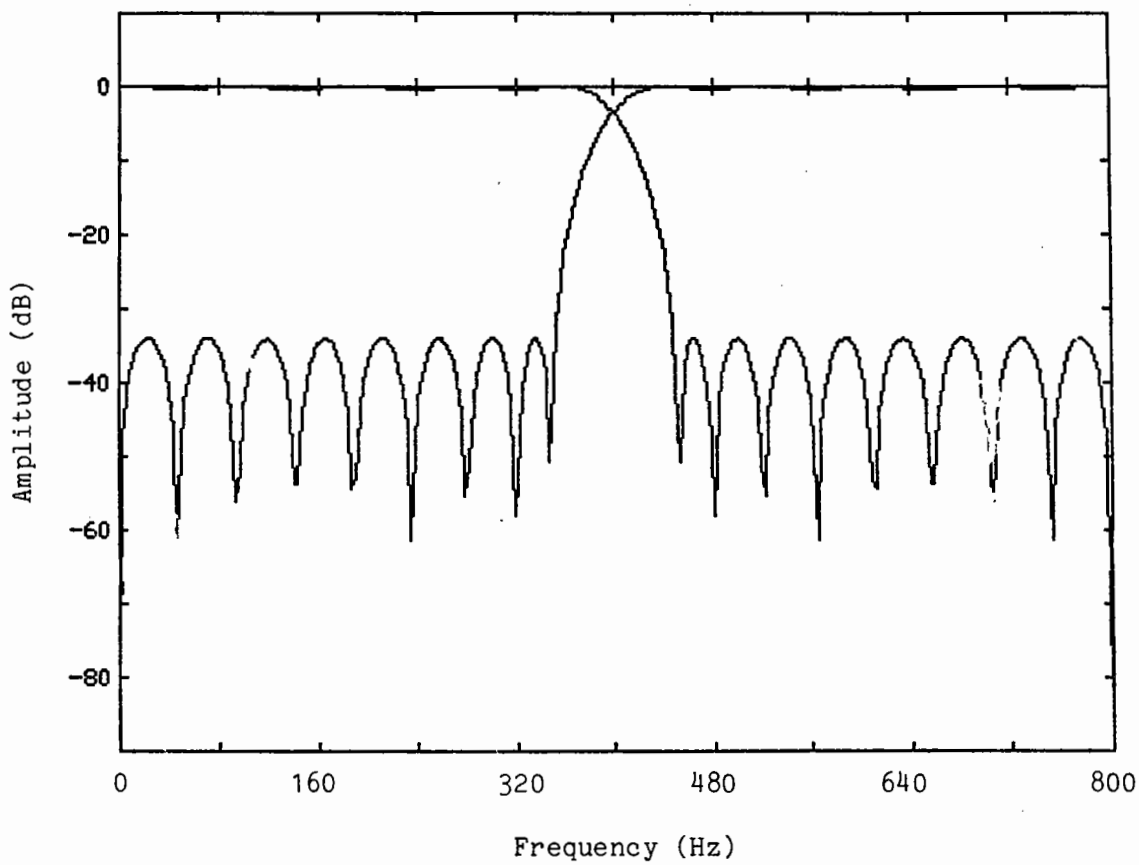


Figure 2-7 Frequency Response of the Quadrature-Mirror Filters

2.5.7 High Frequency Regeneration

A number of experiments were run to determine if some of the steps in the high frequency regeneration procedure could be simplified. The results indicated that one of the double difference operations could be eliminated at only a slight decrease in the quality. However, since the number of operations that could be saved is small, this change was not implemented. The frequency response of the double difference filter preceding the non-linearity is shown in Fig. 2-8.

The high-pass filter in the regeneration branch is matched to the low-pass filter used to generate the low-pass residual. A complementary pair of filters results in a reconstructed residual with no spectral gaps. Experimentation with modifications to the high-pass filter was an integral part of the experimentation with the low-pass filter in the decoder as noted above. The response of the high-pass filter (including the second double difference filter) is shown in Fig. 2-9.

2.6 Coding and Decoding Delay

The coder must buffer the speech samples corresponding to a full analysis frame of data before processing can begin. The delay inherent in this step is the duration of a frame plus the overlap into the subsequent frame (here 160 + 22 samples).

There are additional coding delays attributable to the fact that the filter memories extend into adjacent frames. In the present implementation, of the 16 pairs of sub-band samples produced during a frame interval, 10 pairs correspond to the previous frame and 6 correspond to the present frame. The remaining 10 pairs corresponding

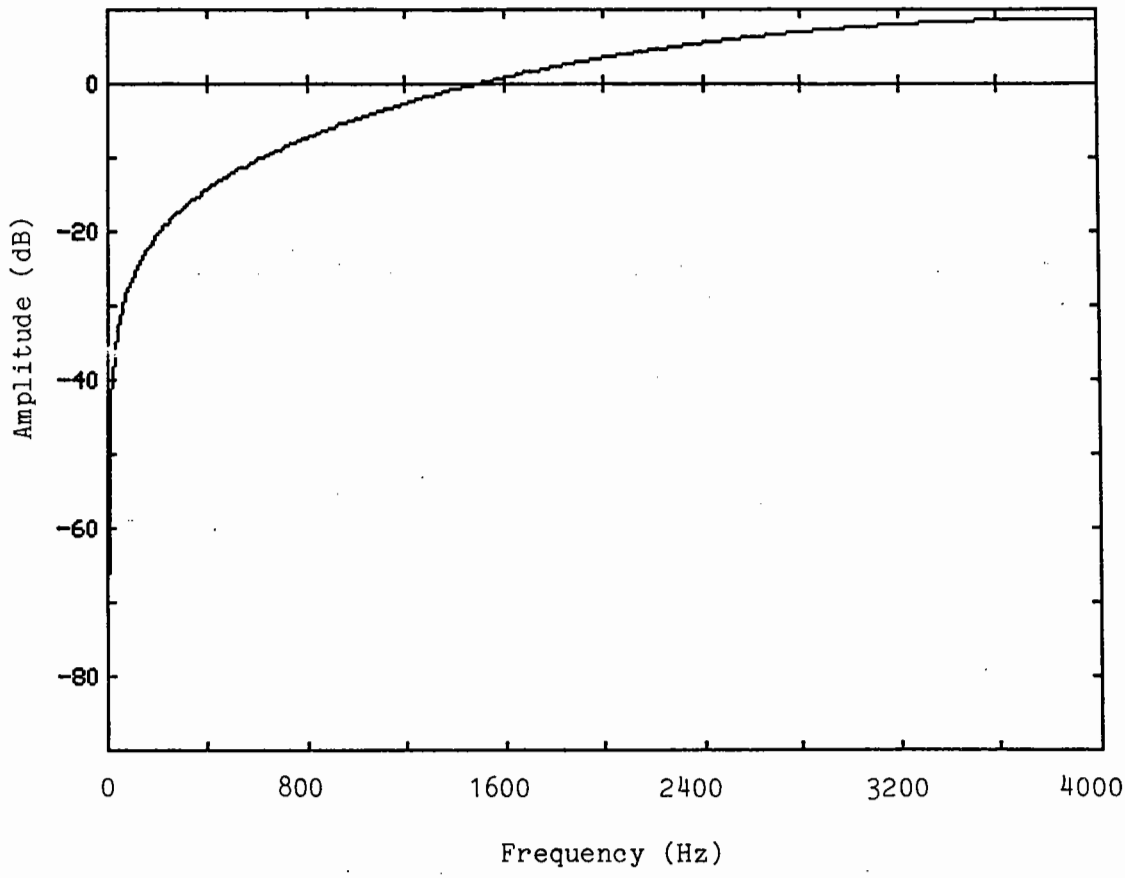


Figure 2-8 Frequency Response of the Double Difference Filter

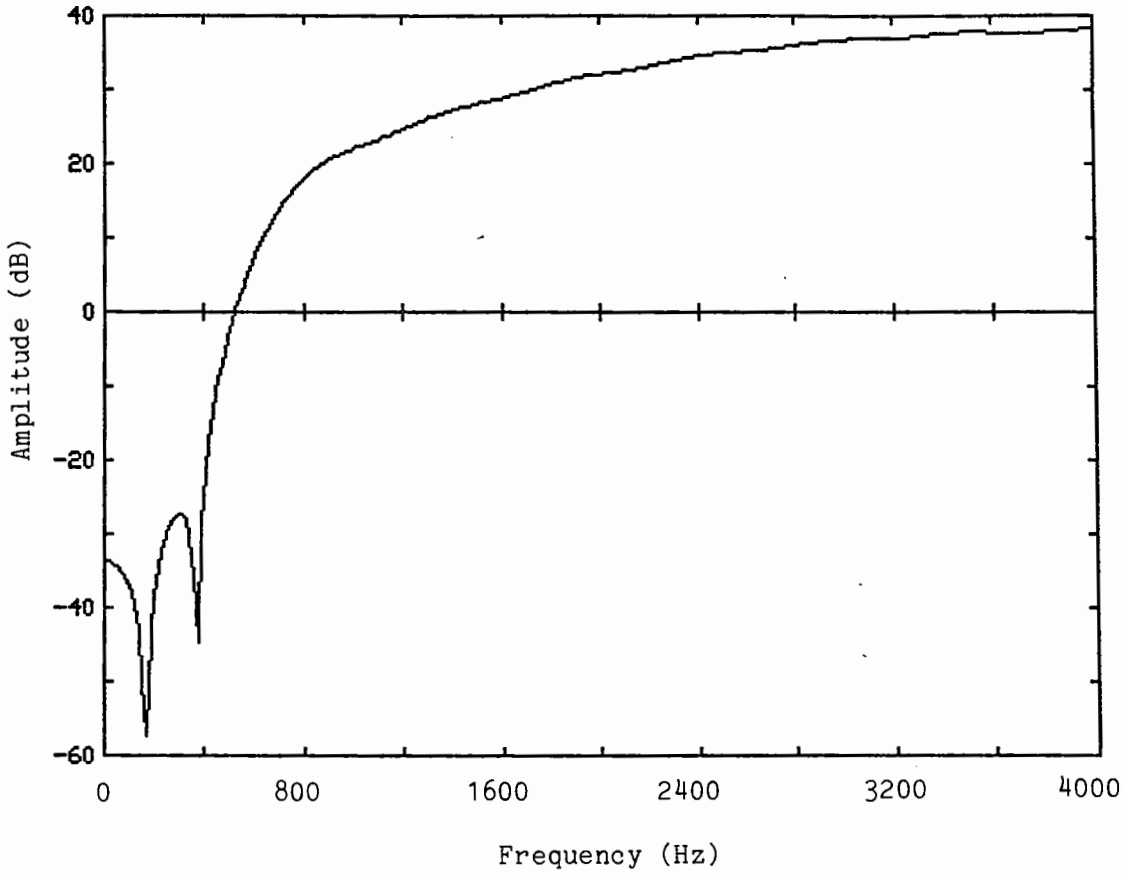


Figure 2-9 Frequency Response of the Decoder High-Pass Filter

to the present analysis frame are left in the "pipeline" to be pushed out during the next frame.

A similar effect occurs in the decoder. Assume the decoder operates in synchronism with each received block of data. For the first part of each block, the synthesizer generates samples corresponding to the previous frame of received data. Part way through the block, the synthesizer coefficients are updated to begin generating samples corresponding to the present frame of received data.

The minimum delay from the input of a speech sample to the coder to the output of a reconstructed speech sample at the decoder is 32 ms. This minimum delay assumes that the processing can occur in zero time. A more realisitic estimate must allow additional time for processing the data. The processing of a frame of data must be completed before a new frame of data is accepted. The processing delay for each of the decoder and encoder is less than a frame. If it is assumed that the processor is matched to the task and not overly powerful, then the processing time will be most of that frame. In addition, to simplify the control structure of the decoder, processing does not start until the whole frame has been received. This implies another frame of delay. With these assumptions, the overall input to output delay, not including any additional transmission delays, can be estimated at 92 ms.

3. Computation Using Integer Arithmetic

The RELP coder described in Chapter 2 was developed using a simulation which was implemented on a general purpose minicomputer using floating point arithmetic. The next step towards a real-time implementation is the use of integer based arithmetic for the calculations. In hardware configurations, integer arithmetic avoids the complexity and slow speed associated with floating point representations.

3.1 Integer Representation

3.1.1 Precision

The choice of precision depends to a large extent on the hardware implementation chosen. Microprocessor architectures, including bit slice processors, usually limit the precision to a multiple of 4 bits. The more conventional architectures offer 8, 16 or 32 bit arithmetic, with the latter available only in products that have been recently announced.

The precision needed for acceptable results depends on the coding algorithm used. The precision needed for LPC analysis has been well studied [3,14]. Based on these studies, 14 bits appears to be at the low end of the scale of acceptable precision for most of the computations. Some operations, notably the calculation of the auto-correlation coefficients, require extended precision to accommodate a larger dynamic range. The RELP algorithm uses the same analysis steps as LPC. Thus the choice of 16 arithmetic with double precision (32 bit) accumulation in certain operations, seems

appropriate. It remains to be verified that this precision is also applicable for the filtering operations in the RELP coder.

3.1.2 Fractional Integer Representation

In keeping with the arithmetic used in most microcomputer systems, twos complement notation is used to represent negative values. Fractional integer representation is used. In this representation, the binary point is assumed to lie immediately to the left of the most significant bit. Thus with twos complement arithmetic, numbers in the range [-1,+1) can be represented. When two numbers represented as fractional integers are multiplied, the product will not overflow. However, addition and subtraction can still lead to overflow.

A conventional multiplier which produces a 2N bit product from two N bit integers can be used for fractional integers. The two input numbers are represented as a sign bit and N-1 bits of precision. The product will have two sign bits and 2N-2 bits of precision. A correctly scaled result is obtained by using one of the sign bits and as many bits of precision as desired. The difference between this representation and the conventional integer representation (binary point to the right of the bits) manifests itself in the choice of the product bits which are retained. Conventional integer arithmetic retains the low order part of the product, while fractional integer arithmetic retains the high order part of the product.

A twos complement divider takes a double precision dividend and a single precision divisor to produce a single precision quotient (and possibly a remainder). Conventional integer arithmetic loads a single precision dividend into the low order part of the double precision

dividend, while fractional integer arithmetic loads it into the high order part.

3.2 Coder

The approach taken has been to integerize the computations in the coding and decoding algorithms step by step. The effect of integerizing each step can then be assessed.

3.2.1 Input Precision

The input samples need be digitized to only 12 bits. A 10 bit A/D converter may be used but with a reduced dynamic range. Automatic gain control circuitry can compensate for the reduced dynamic range, but this aspect was not explored further.

3.2.2 Analysis

The analysis stage which produces the quantized reflection coefficients and the inverse filter coefficients was converted to use integer arithmetic. This stage, though perhaps the most complex to integerize, produces very little effect on the resultant speech when compared with the floating point computations. No difference can be discerned between the two versions. In all except the auto-correlation computation, which can be likened to a filtering operation, single precision arithmetic was employed. Thus for multiplication, only a 16 bit result was kept. The individual steps followed are described more fully in the following subsections.

3.2.2.1 Auto-correlation Calculation

The auto-correlation coefficients cover a wide dynamic range. The zero lag coefficient, which has the largest magnitude, represents the square of the signal energy. These coefficients were calculated using a double precision accumulator. This accumulator is guaranteed not to overflow, viz. for a frame of 204 samples, each specified to 12 bits (11 bits plus sign), the maximum value for r(0) can be represented in 31 bits plus sign.

The analysis portion of the coding algorithm is insensitive to a scaling of the auto-correlation coefficients. In view of this, the double precision coefficients are scaled such that the largest magnitude coefficient, r(0), has a full 16 bits of precision. Since r(0) is positive, this is accomplished by searching for the first occurrence of a one bit in the double precision result. The coefficients are all shifted to the left by the number of bits required to bring this one bit in r(0) to the most significant bit position. Except for zero energy frames, the normalized coefficients then have

$$0.5 \leqslant r(0) < 1. \qquad (3\text{-}1)$$

3.2.2.2 Reflection Coefficient Solution

The algorithm for the solution of the linear equations which determine the reflection coefficients, uses a set of intermediate variables (corresponding to the normalized prediction error), which have magnitude less than unity. This normalization is fully compatible with fractional integer notation. The algorithm, as in the original Durbin recursion, uses a divide at each stage of the recursive solution

of the equations. All arithmetic for this step uses single precision arithmetic.

3.2.2.3 Reflection Coefficient Quantization

Since the reflection coefficients are bounded in the range -1 to +1, no problems occur in representing them. Quantization, implemented as a table look-up procedure, is also a straightforward conversion from a floating point implementation.

3.2.2.4 Reflection to Predictor Coefficient Transformation

This operation needs careful attention to normalization of the data values. It was found that the predictor coefficients need a dynamic range from -4 to 4. Thus this step returns the predictor coefficients scaled by 4. All arithmetic for this step is done in single precision.

3.2.3 Filtering Operations

The other coder operations involve filtering using FIR filters. Since these filters are feedforward, they eliminate any worries about limit cycles such as occur in recursive filters. The dot product implicit in the calculation of each output value was calculated using a double precision accumulator. This was done more to mimic the action of commercially available multiplier/accumulator devices than for reasons of accuracy, although in some cases careful scaling of the results is necessary in order to retain significance.

The double precision product is reduced to a 16 bit value by extracting the appropriate 16 bits. For instance, in the inverse filter, since the coefficients are scaled by a factor of 4, conceptually the result is shifted by two bits before converting the double precision result to a 16 bit value.

3.3 Decoder

The operations in the decoder are similar to those carried out for the coder. However, more care has to be exercised in scaling of the data before conversion to integer arithmetic.

3.3.1 Filtering

The filters in the decoder are used to interpolate the received sub-band samples to form the low-pass residual. These interpolation filters have gains larger than one (two for the quadrature mirror filters and five for the low-pass filter). This means that the coefficients have to undergo additional scaling to prevent overflows.

Additional filtering occurs in the high frequency regeneration branch. In this process, the low-pass residual is processed to produce high frequency components. These low-amplitude high frequency components must be separated from the relatively larger low frequency components using a high-pass filter. These are then amplified (by a factor 25) before they are added to the low-pass residual. This gain factor has been absorbed into the filter coefficients, resulting in a filter with the largest coefficient value being 40.

The scaling strategy used for all the filters will be illustrated with the example of the high-pass filter just described. The coefficients are scaled by a factor of 64 to reduce their magnitudes to below unity. The filtered result will also be scaled by a factor of 64. If the result is only available as a single precision value, scaling will render 6 bits not significant. With a double precision result available, the single precision result can have full significance after scaling. This procedure is implemented as a function which returns a selected group of 16 bits out of the 32 bit product.

3.3.2 Synthesis

The synthesizer is a direct form recursive filter which uses the predictor coefficients obtained by transforming the received reflection coefficients. The transformation procedure is identical to that employed in the coder.

The synthesis filter itself uses double precision accumulation. The double precision result is then truncated to a single precision value to become the output value. The single precision output value is used to form subsequent output values recursively.

3.4 Performance of the Integerized Algorithm

The analysis stage in the coder performed in integer arithmetic gives virtually identical results to a floating point implementation. The filtering operations in the coder and particularly in the decoder require the scaling described above to avoid additional perceptible noise. In spite of its recursive structure, the synthesis filter adds no discernible noise in its integer form. The overall result is such

that the speech resulting from a fully integerized algorithm cannot be distinguished from that obtained with floating point computations.

4. Real-Time Implementation Considerations

Many factors must be considered in choosing the specific hardware configuration for a real-time coder. The recent single chip central processing units (CPU's) are powerful enough to allow a degree of implementational flexibility and efficiency heretofore not available.

The configuration appropriate to these devices is a programmed microcomputer augmented with a hardware signal processing peripheral. The previous generation of low bit rate coders have had to be customized in hardware for each speech algorithm. The new programmable units can be made much more flexible. This flexibility is achieved with a general purpose signal processing capability that can be built around a single chip CPU. For example, the same configuration that can implement RELP coding should be able to be reprogrammed to implement an LPC coder.

The programmed approach also allows for the efficient implementation of real-time speech coding algorithms. Efficiency is used here in a global sense. It includes such factors as development time. The advantages are summarized below.

Flexibility

a) The same hardware configuration can be used to implement a variety of speech coding algorithms. The switch between algorithms need only involve changing the control program.
b) Modifications to an algorithm can be readily implemented. As new concepts in speech coding are developed or as new applications for a coder arise, modifications can be retrofitted easily into existing hardware.

Efficiency

a) The hardware arrangement can be compact, consisting of powerful subunits. The use of standard "chips" reduces the equipment costs.

b) Hardware development time is reduced. A well structured architecture allows the use of commercially available single board computers in the development stage. The amount of special purpose hardware that must be developed is kept to a minimum. The final coder can be a pared down version of the development system, omitting unneeded hardware.

c) The algorithm development time is reduced because speech algorithms pre-tested on general purpose computing facilities can be readily adapted to the single chip CPU. In fact high level languages such as PASCAL, C and FORTRAN are available for the new generation of micro-processors. The penalty for using a high level language such as C has been estimated to be about 30% over carefully optimized assembly language coding [15]. The judicious use of assembly language modules for the time consuming operations can often reduce this execution time penalty. The remaining penalty is then just in code size expansion. With reduced memory costs, the trend will be to accept this penalty in order to obtain the benefits of using a high level language.

The advantages of a high level language are numerous. The development time for an algorithm in a high level language is much less than for one coded in assembler. The capability of using a high level language also allows the easy transcription from a archetypical coding algorithm developed on a general purpose computer.

4.1 Choice of Micro-Processor

The options range from a fully customized "random" logic arrangement through bit-slice micro-processors to single-chip CPU's. The first option will be not be considered further. Bit-slice micro-processors are a middle ground. They can be somewhat faster than the present generation of single-chip CPU's. These devices are micro-coded with custom instruction sets for each application. This is both an asset and a liability. The custom instructions can be tailored for the application at hand as was done for a LPC coder developed at Lincoln Laboratories of MIT [16] in 1973-1977. The custom tailoring necessitates considerable software development for the micro-code and the assembler. The use of higher level languages is precluded unless considerable further software development is undertaken.

Single-chip CPU's are available from several vendors. Only 16 bit micro-processors were considered.

i) INTEL 8086

ii) Motorola 68000

iii) Fairchild 9445

iv) Zilog Z8000

A number of different performance measures may be used for micro-computers.

i) Speed - Instruction execution time, especially for arithmetic operations.

ii) Instruction Set - Appropriateness for the task at hand.

iii) Availability - Whether or not the device is readily available, including from second sources.

iv) Peripheral support - Availability of peripheral devices such as

I/O processors, DMA controllers etc.

v) Development System Support

The decision based on these factors is still largely subjective in that various weightings can be attached to these items. It was felt that speed was a prime objective with development system support also being important. Several single chip CPU's were assessed as signal processors in [17]. It should be noted that even the fastest of the single chip CPU's, the Fairchild 9445, cannot perform RELP coding in real-time without additional hardware. Based on these factors, the Motorola 68000 seems to be the best choice.

4.2 Choice of Special Purpose Hardware

The basic single-chip CPU can perform many of the tasks needed but in the case of the RELP algorithm, falls short in the area of filtering. This operation is very intense in its use of multiplications and additions. One option is provide a fast multiplication capability using unimplemented instruction codes or with a co-processor arrangement (as for example in the floating point operations of the Intel 8086/8087 combinations). Neither of these will give the performance required since the CPU is still decoding the instructions and fetching operands.

An alternative is the use of one of the new signal processing peripheral chips. However, these units were not considered to be adequate at this time. As an example, consider the most powerful of these devices, the NEC μPD7720 [18].

i) Speed. The multiply time is 250 ns without accumulation. Double precision accumulation would require at least two additional

cycles.

ii) Input/Output. The data paths in and out of the chip are only 8 bits wide.

iii) Restrictions. The coefficient storage is a Read-Only Memory, restricting the flexibility for use in certains operations such as the inverse filter, where the coefficients are dynamically determined.

iv) Availability. This is a new product, which is at present available only in a mask programmable version.

The option of choice is to provide a signal processor based on a fast multiply/accumulator. Such a device is available as a single chip multiplier that multiplies two 16 bit operands and accumulates the double precision product in under 150 ns (TRW 1010J). A signal processor built around this device would provide rudimentary control functions and fast local memory for the multiply/accumulator.

4.3 Multiplier / Accumulator

The philosophy behind the arrangement selected has been to provide a processor powerful enough to perform the task at hand but to leave as much of the control function as possible to the main CPU.

In order to reap the benefits of the fast multiply/accumulate, a local memory associated with the peripheral serves as a fast cache memory. The memory can be loaded with enough data to enable the computation of a number of output values.

A block diagram of one possible implementation of such a processor is shown in Fig. 4-1. Two register files are shown. These appear to the data bus as extensions of the main memory of the CPU. In a typical filtering application, one register file is loaded with coefficient values, while the other contains input data. The control registers contains pointers to the data and coefficients in a register designated as (PD,PC). Modifying the contents of this register can be used to automatically start the processing. Another register, designated as (NT) contains the number of terms in the accumulated products,

$$z = \sum_{i=0}^{NT-1} x_{i+PD}\, y_{i+PC} . \qquad (4-1)$$

The result is accessed by reading the double precision output register. Synchronization between the main CPU and the peripheral processor is maintained by stretching a read cycle until the computed data is available.

The unit as described is a very simple but adequate for the RELP coder/decoder application. A more sophisticated processor could automatically generate a number of output points in succession.

As an example consider a simple filter such as the high-pass filter used in the high frequency regeneration process. The register files would be primed with the coefficients and data. The number of data values would be equal to the number of samples in a frame (160) plus the filter memory (number of filter coefficients less one). The number of filter coefficients is then written into the (NT) register. The (PD,PC) register is set to (0,0) to initiate the calculation of the first ouput value. This value is then retrieved from the output

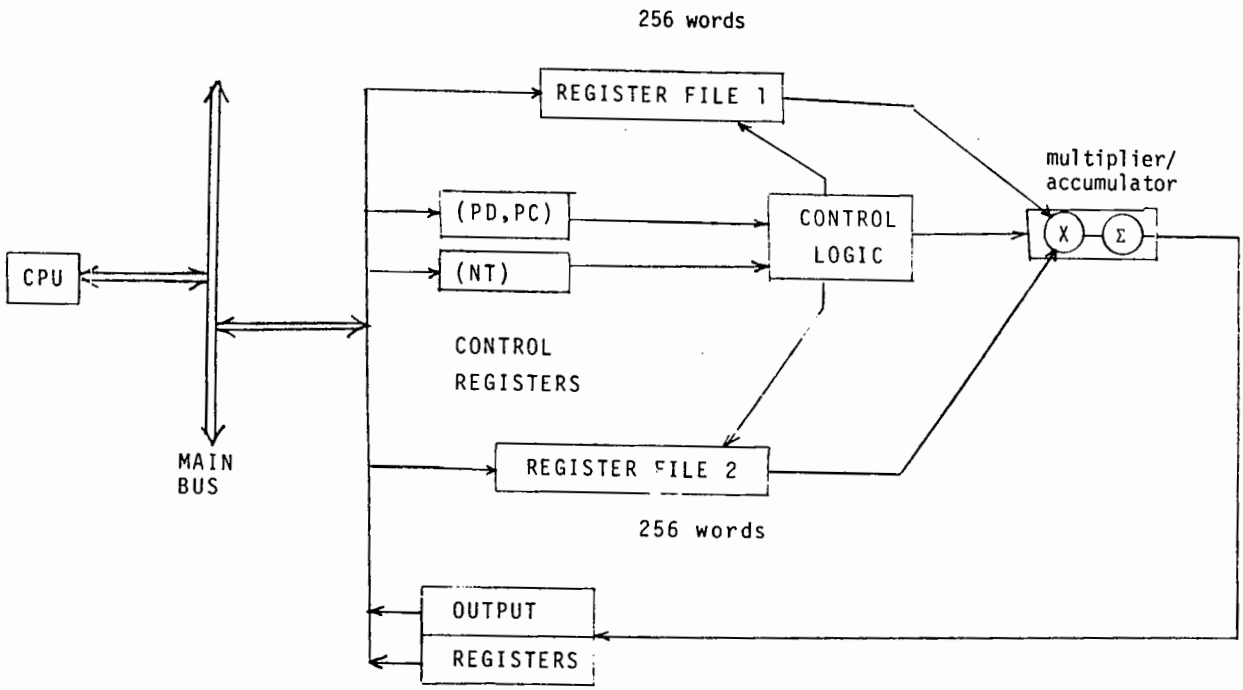


Figure 4-1 Peripheral Multiplier/Accumulator Processor

register. The next invokation is initiated with (PD,PC) set to (0,1). This computation uses the same set of coefficients, but the data array used is shifted by one point. Subsequent calculations for the data block proceed by writing a single word to the peripheral control register and retrieving the output value.

Filtering operations involving sub-sampling would be similar to that just described, except that the data array would be stepped through in steps equal to the sub-sampling ratio. For interpolation, the coefficients are separated into those corresponding to the sub-filters. The processor is then invoked using the sub-filters in succession. For the recursive synthesis filter, each output point must be written back into the register file before restarting the computation. For the auto-correlation calculation, the register files are both loaded with the same data. Thus this simple processing arrangement provides the capability of performing all of the required "filtering" operations needed in RELP.

4.4 Timing Considerations

With the proposed signal processing architecture, it remains to be verified that the unit is sufficiently fast to implement the RELP algorithm in real time. The processing will be split between the main CPU (a Motorola 68000) and the peripheral multiplier/accumulator. The operations involving the more intricate parts of the algorithm will be performed by the main CPU. Estimates of the execution times are shown in Tables 4-1 and 4-2 for each frame of data. These tables give only the times attributable to either filtering in the peripheral processing or the analysis time as explained more fully below.

Coder Operation	No. Transfers	Transfer Time ms	Execution Time ms
1. Data from D/A	160	0.16	-
2. Analysis			
2.1 Auto-Correlation	444	0.44	0.03
2.2 Lag Windowing	-	-	0.08
2.3 Recursive Solution	-	-	3.0
2.4 Quantize	-	-	-
2.5 Transform Coefficients	-	-	1.9
3. Inverse Filter	693	0.69	0.22
4. Low-Pass Filter	317	0.32	0.15
5. Sub-Band Filter	199	0.20	0.09
6. Code Sub-Bands	-	-	-

Table 4-1 Coder - Estimated Processing Time for a Frame of Data

Decoder Operation	No. Transfers	Transfer Time ms	Execution Time ms
1. Sub-Band Interpolation	163	0.16	0.09
2. Low-Pass Reconstruction	584	0.58	0.14
3. High-Pass Reconstruction	1352	1.35	0.79
4. Synthesis			
4.1 Transform Coefficients	-	-	1.9
4.2 Filter	706	0.71	0.19
5. Data to D/A	160	0.16	-

Table 4-2 Decoder - Estimated Processing Time for a Frame of Data

4.4.1 Filtering Operations

Considerable overhead is involved in setting up the multiplier/accumulator. Data must be transferred to and from the peripheral. The data setup will be assumed to be done using Direct Memory Access (DMA). The number of data transfers in and out of the peripheral processor uses up a significant amount of time. It is assumed that DMA operations proceed at 1 μs per value. Single transfers (MOVE instructions) proceed at only a little slower speed. The computation of an output value in the peripheral processor proceeds at 150 ns per multiply/accumulate. A certain amount of parallel processing is possible, but because of the speed of the peripheral processor, there is little opportunity for the main CPU to do more than increment its loop counters before a filtering operation is complete.

4.4.2 Equation Solution / Coefficient Transformation

These operations involve recursive operations which are performed by the main CPU. Its general purpose structure is well suited for this purpose. The execution times for these steps were estimated based on the use of the internal multiply and the internal divide available on the 68000. The times were estimated as follows. The routines were timed on a PDP-11/45 with a hardware multiply and divide option. The multiply and divide times were subtracted from the total time. This value was divided by two to reflect the difference in the CPU speeds. (The Motorola 68000 (8 MHz version) is faster by a factor of about 2 for for a mixed set of instructions.) To this was added the multiply and divide times for the 68000. These steps are shown in more detail in Table 4-3. These estimates lead to estimates that are probably conservative since

	PDP-11/45	MC68000
Recursive Solution (37 Multiplies, 8 Divides)	5.3 ms	3.0 ms
Coefficient Transformation (28 Multiplies)	2.8 ms	1.9 ms

Table 4-3 Estimated Execution Times for the Analysis Routines

the analysis routines were coded in a high level language.

4.4.3 Total Time

The times given in the Tables do not account for all of the necessary computations. The other computations for which execution times are hard to estimate account for only a small fraction of the computations required. Examples of routines not included are that which quantizes the reflection coefficients and that which codes the sub-band samples. The time to perform the pre-emphasis and de-emphasis has also not been included, since these operations can be performed using a simple analog network if necessary. For the cited operations, the coder uses 36% of the available time and the decoder uses 30% of the available time. Thus both the coding and decoding can be performed simultaneously with the proposed configuration.

Consideration must also be given to coordinating coder and decoder operations since these steps occur asynchronously. (The coder and decoder clocks are not in general synchronized.) One possible mechanism to accomplish this is to divide the processing of each of the coding and decoding operations into distinct steps (for instance, a filtering operation using the peripheral processor). When a step is completed, the processor selects the next step from the stream that is lagging behind. This simple mechanism will multiplex coder and decoder operations without incurring significant overhead.

4.5 Program Size

While it is difficult to estimate the program size reliably without actually coding some parts of it, estimates from the corresponding PDP-11/45 code will be used. The instruction sets are comparable, so that program sizes should be comparable. The sizes of the different modules is shown in Table 4-3. The array space has been used somewhat extravagantly in the implementation on the PDP-11/45. Considerable reduction in the data storage could be obtained by reusing the data vectors in succeeding steps of the algorithm. The module sizes were determined after excising some general purpose features not needed in the real-time implementation. The modules are still somewhat larger than necessary. However, additional code would have to be provided in a real-time coder to provide services normally offered by the executive or run-time system. Even with these provisions, the full real-time system should be able to be accommodated in 4K words of read/write memory and 4K words of read-only memory.

4.6 Cost

It is hard to estimate costs in units which use advanced technology, since the costs will vary rapidly with time. However in this case, commercial products of nearly the same complexity are available for comparison. The heart of the coder is a single chip CPU. Single board computers are available commercially at single quantity prices of below $2500. Such a board typically consists of a CPU, memory, bus interface circuitry, and serial or parallel ports. With the exception of the last item, all these fit in with the unit under consideration. A signal processing peripheral must be added to this

Module	Instruction Space	Read/Write Data	Read-Only Data
Coder	720	730	250
Decoder	650	1100	290
Common Modules	980	200	-

Table 4-4 RELP Program Size (Words)

complement. With the arrangement proposed above, this unit becomes very simple, not adding substantially to the cost of the unit. In addition to the multiplier/accumulator, this unit could be constucted with about 20 medium scale integrated circuits. (A much more complicated signal processor based on the same multiplier/accumulator, has been was implemented using 69 MSI chips [19].) This would add about $500 to the component cost based on single quantity prices ($400 for the multiplier/accumulator). Additional circuitry to interface with an analog input signal (filter plus A/D converter) and to generate an analog output signal (D/A converter plus filter) must be provided. These last items would add an additional $100 to the component cost.

The above component costs indicate that a RELP coder and decoder on a single board should be able to be produced for under $3000. The cost of packaging this board (power supplies, cabinets, etc.) will add to this basic cost.

5. Conclusions

The main conclusion of this study is that a RELP coder and decoder can be built with a simple hardware configuration readily available VLSI components. This microcomputer based architecture allows for a rapid progression from the algorithmic description as contained in this report to the final hardware version. In addition, with only minor modifications (if any at all), this structure is applicable to variety of speech coding algorithms.

The specific work performed for this study can be summarized as follows.

i) The RELP algorithm has been restructured to be compatible with real-time processing. This involves performing the calculations on a frame basis.

ii) The components of the coding algorithm have been modified to achieve a uniformity of structure. Many of the computationally intensive steps in the algorithm were found to be expressible as filtering operations. This allows a single filtering module to be reused in several steps in both the coder and the decoder. In addition, the complexity of the algorithm was reduced somewhat without impairing the quality of the resulting speech.

iii) A fully integerized version of the RELP algorithm using 16 bit integer arithmetic has been developed. With the scaling employed and the judicious use of double precision accumulation, there is no discernible difference in speech quality when compared to a simulation using floating point arithmetic.

iv) An architecture for a real-time implementation based on a single chip CPU (Motorola 68000) augmented with a fast

multiplier/accumulator unit (based on the TRW 1010J) has been proposed. This arrangement utilizes the microprocessor to control the processing for both itself and the peripheral processor. The simplicity of this combination requires minimal hardware development to produce the peripheral processor.

v) Timing studies of the above arrangement suggest that simultaneous coding and decoding (full duplex operation) is possible in real-time.

vi) Studies of the hardware complexity indicate that the overall complexity and organization of the real-time coder is comparable to that of single-board computers based on 16 bit microprocessors. These are available at single quantity prices below $2500. The cost of packaging (power supplies, cabinets, etc.) will add to this basic cost.

Appendix A Sub-sampling / Interpolation

A.1 Introduction

This Appendix develops the z-transform relationships used for the analysis of multi-rate operations, in particular for the analysis of quadrature-mirror filters.

A.2 Z-Transform

The z-transform of a sequence {x(n)} where n takes on integer values is defined as

$$X(z) = \sum_{n=-\infty}^{\infty} x(n) z^{-n} . \qquad \text{(A-1)}$$

The unit delay z^{-1} corresponds to the spacing between elements of the sequence when these are considered to be equi-spaced time samples. The frequency response corresponding to the z-transform X(z) is

$$X(e^{j2\pi ft}) = \sum_{n=-\infty}^{\infty} x(n) e^{-j2\pi fnT} , \qquad \text{(A-2)}$$

where T is the time interval between time samples.

A.3 Sub-sampling

The operation of sub-sampling a sequence can be interpreted as re-sampling a sequence. The sub-sampled sequence is

$$x_s(n) = x(nR+r) , \qquad \text{(A-3)}$$

where R is the (integer valued) sub-sampling ratio and r is an offset taking on values from 0 to R-1.

The z-transform of the sub-sampled sequence $\{x_s(n)\}$ will be derived in three steps.

i) Sample the original signal.

Initially the offset r will be taken to be zero. Sampling is accomplished by multiplying the sequence point-by-point by a periodic pulse train, namely,

$$s(n) = \begin{cases} 1 & n=mR \text{ or } n \bmod R=0, \\ 0 & \text{otherwise}, \end{cases}$$

$$= \frac{1}{R} \sum_{m=0}^{R-1} e^{j2\pi mn/R} , \qquad (A-4)$$

The geometric series above can be summed to give 0 when n mod R is non-zero and unity when n mod R is zero. Multiplying the sequence $\{x(n)\}$ by the sequence $\{s(n)\}$ has the effect of zeroing the undesired samples. The z-transform of the sampled sequence is

$$\sum_{n=-\infty}^{\infty} x(n)s(n)z^{-n} = \frac{1}{R} \sum_{m=0}^{R-1} \sum_{n=-\infty}^{\infty} x(n)e^{-j2\pi mn/R}z^{-n}$$

$$= \frac{1}{R} \sum_{m=0}^{R-1} X(e^{-j2\pi m/R}z) . \qquad (A-5)$$

ii) Sub-sample to remove the zero-valued samples.

Define the z-transform of the sub-sampled sequence using the new dummy variable w. The w represents the unit delay for the reduced rate sequence,

$$w^{-1} = z^{-R} . \tag{A-6}$$

The z-transform of the sub-samples sequence is

$$X_s(w) = \frac{1}{R} \sum_{m=0}^{R-1} X(e^{-j2\pi m/R} w^{1/R}) . \tag{A-7}$$

iii) Introduce the offset r.

The effect of the offset can be taken into account by sub-sampling the shifted sequence {x(n-r)}. The z-transform of the shifted sequence is

$$\{x(n-r)\} \longleftrightarrow z^{r}X(z) . \tag{A-8}$$

Finally the z-tansform of the sub-sampled sequence taking into account the offset is

$$X_s(w) = \frac{w^{r/R}}{R} \sum_{m=0}^{R-1} X(e^{-j2\pi m/R} w^{1/R}). \tag{A-9}$$

Fig. A-1 shows an example of the sub-sampling procedure for R=3 and r=1.

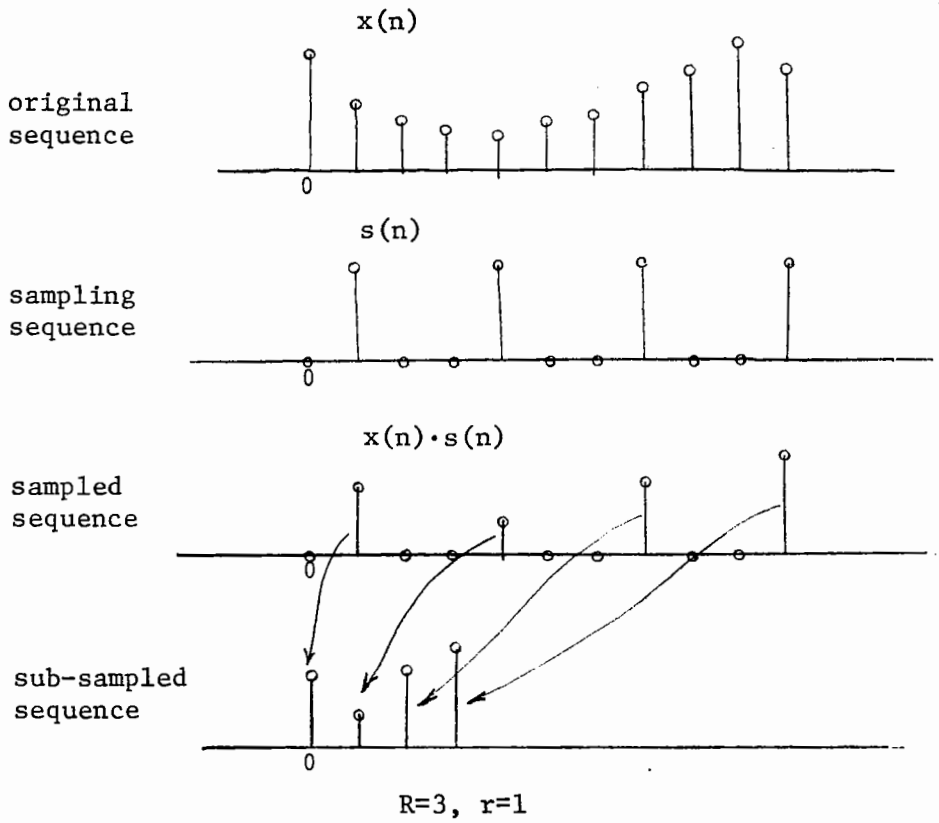


Figure A-1 Sub-Sampling a Sequence

The frequency domain expression for the sub-sampled sequence is

$$X_s(e^{j2\pi fRT}) = \frac{e^{j2\pi r/R}}{R} \sum_{m=0}^{R-1} X(e^{j2\pi(f-m/T)T}) . \qquad (A-10)$$

The frequency response of the sub-sampled sequence is the sum of frequency translations of the response of the original sequence. If these frequency translated responses have a non-zero value in regions of overlap, aliasing results. Fig. A-2 illustrates the frequency response of a sub-sampled sequence for R=3.

Often a sequence is filtered prior to sub-sampling to eliminate frequency components which lead to aliasing. Let the input to such a filter be the sequence {u(n)} with z-transform U(z) (see Fig. A-3). Let the filter have a z-transform H(z). Then

$$X(z) = U(z)\, H(z) , \qquad (A-11)$$

or written as a convolution,

$$x(n) = \sum_{k=-\infty}^{\infty} h(k)u(n-k) , \qquad (A-12)$$

where {h(n)} is the unit pulse response of the filter.

If the filter H(z) is a finite length impulse response (FIR) filter of length N,

$$x(n) = \sum_{k=0}^{N-1} h(k)u(n-k) . \qquad (A-13)$$

For such a filter, the output is expressed directly in terms of the input signal. If the output of the filter is to be sub-sampled, the

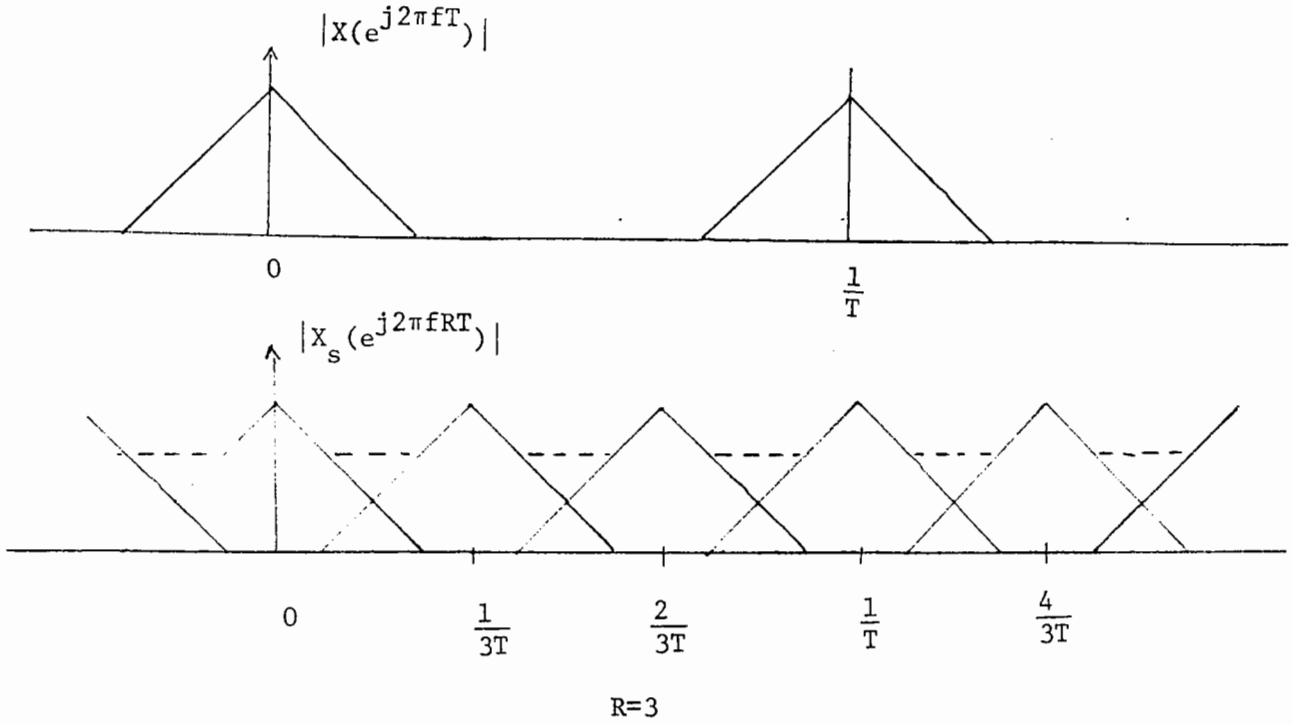


Figure A-2 Frequency Response of a Sub-Sampled Sequence

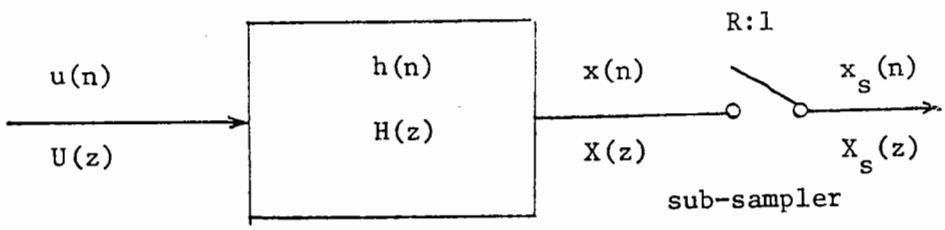


Figure A-3 Filtering / Sub-Sampling

intermediate output values need not be calculated,

$$x_s(n) = \sum_{k=0}^{N-1} h(k)\ u(nR+r-k)\ . \qquad \text{(A-14)}$$

This arrangement reduces the number of arithmetic operations by a factor R. For recursive filters, in which the output is expressed as a function of previous outputs as well as the inputs, this full reduction is not possible.

A.4 Rate Increase / Interpolation

Interpolation involves inserting new samples between existing samples. This operation can be viewed as occuring in two steps. The sampling rate is first increased by a factor R by inserting R-1 zero valued samples between each sample of the original sequence. This new sequence is processed to modify the values of the R-1 intermediate sample values. The processing considered here is linear filtering. Many common techniques usually thought of as belonging to the domain of numerical analysis can be viewed as linear filtering of the rate increased sequence. An example is polynomial interpolation, e.g. linear or quadratic interpolation [20]. Fig. A-4 and Fig. A-5 illustrate the interpolation process.

The rate increased sequence can be expressed as

$$u_i(n) = \begin{cases} u(m) & n=mR+r \text{ or } n \bmod R = r, \\ 0 & \text{otherwise.} \end{cases} \qquad \text{(A-15)}$$

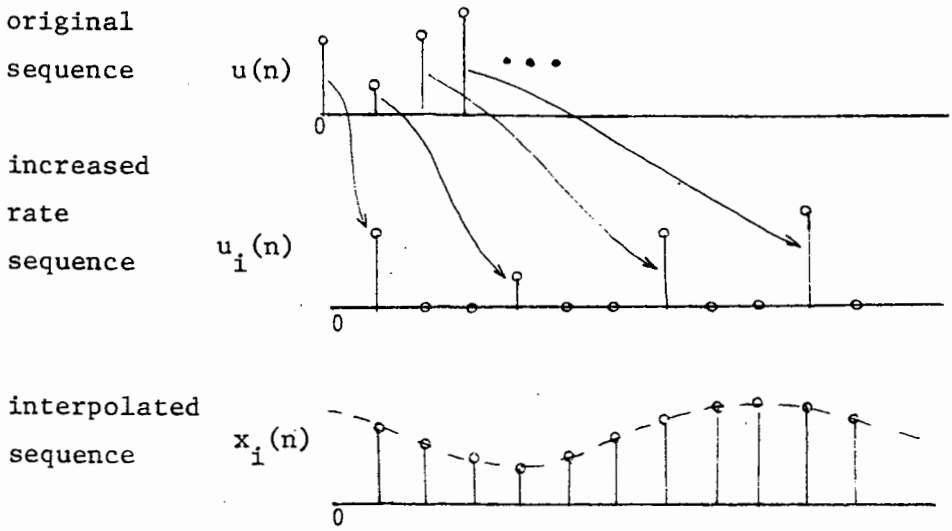


Figure A-4 Interpolating a Sequence

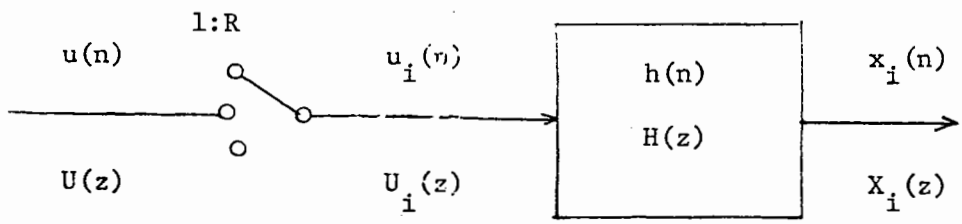


Figure A-5 Rate Increase / Filtering

The z-transform of $\{u_i(n)\}$ is

$$U_i(z) = z^{-r} U(z^R) \ . \qquad \text{(A-16)}$$

Finally the interpolated sequence is obtained by filtering $U_i(z)$,

$$X_i(z) = U_i(z)\ H(z) \qquad \text{(A-17)}$$

The frequency response corresponding to $X_i(z)$ is

$$X_i(e^{j2\pi fT}) = e^{-j2\pi frT}\ U(e^{j2\pi fRT})\ H(e^{j2\pi fT}), \qquad \text{(A-18)}$$

where T is the spacing between samples of the interpolated sequence. Thsi is illustrated in Fig. A-6.

An FIR filter used to interpolate a sequence can take advantage of the fact that R-1 out of R samples of the rate increased sequence are known a priori to be zero. The filter coefficients $\{h(n)\}$ can be separated into sub-sequences which are accessed in a round-robin fashion. More explicitly, let

$$n = pR+q+r \ , \qquad \text{(A-19)}$$

where $0 \leqslant q \leqslant R-1$. The R sub-filters are

$$h_q(m) = h(mR+q) \ , \qquad \text{(A-20)}$$

each with M coefficients, where

$$M_q = \left\lfloor \frac{N-1-q}{R} \right\rfloor , \qquad \text{(A-21)}$$

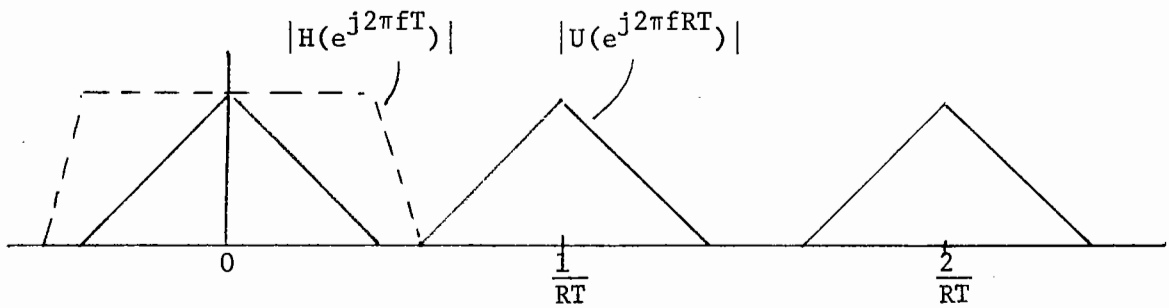


Figure A-6 Frequency Response of an Interpolated Sequence

where $[x]$ denotes the largest integer contained in x. Then

$$x_i(pR+q+r) = \sum_{m=0}^{M_q} h_q(m)\, u(p-m) \cdot \qquad \text{(A-22)}$$

From the above formulation, it can be seen that interpolation using linear filter forms R sub-sequences of $\{x_i(n)\}$ each of which was generated using a different sub-filter $\{h_q(n)\}$. In conventional interpolation, it is usually desirable that the samples of the original sequence appear unaltered in the interpolated sequence, i.e. only the R-1 intermediate values are to be modified. This constraint implies that

$$h_0(m) = \begin{cases} 1 & m=0, \\ 0 & \text{otherwise.} \end{cases} \qquad \text{(A-23)}$$

Design procedures for interpolation filters have been described in the literature [21].

A.5 Sub-Band Analysis

The purpose of sub-band analysis is to split a signal into a number of different frequency bands. An example is spectral analy. is achieved with a bank of filters. Consider the configuration shown in Fig. A-7. In many applications, the N filters are narrow band filters with different centre frequencies. In some applications, the output of each filter is frequency translated to become a low-pass signal, for instance in a spectrum analyzer for envelope extraction.

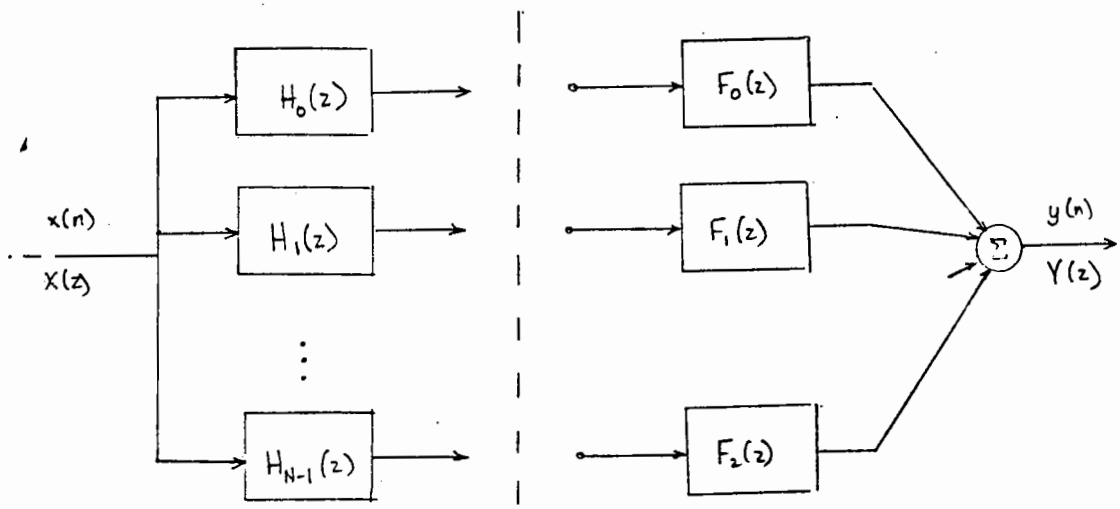


Figure A-7 Bank of Filters

In the case of signal coding, a sub-band analysis is used to derive a number of essentially non-overlapping sub-bands that can be individually coded.

A.5.1 Reversibility

One concern is whether the signals at the outputs of the filters contain all of the information of the input signal. Clearly, if the filter outputs can be used to reconstruct the input signal, this will be the case. This will be a major criterion in the design of sub-band systems. In the simple arrangement of Fig. A-7, the input signal can be reconstructed if

$$\sum_{i=0}^{N-1} H_i(z)\, F_i(z) = 1 \ . \qquad \text{(A-24)}$$

A.5.2 Information Rate

The sub-band system just described permits a representation of the input signal as the vector of filter outputs. This vector contains redundant information since the overall sampling of the system has been increased by a factor N.

Consider sub-sampling the filter outputs by a factor R for the n'th filter output. Reconstruction of the signal involves an interpolation operation in each sub-band. This is illustrated in Fig. A-8. For generality, a frequency translation operation has been introduced before sub-sampling. Conceptually, this operation frequency shifts the band-pass output of the filters to produce a low-pass signal. The inverse frequency translation is shown in the

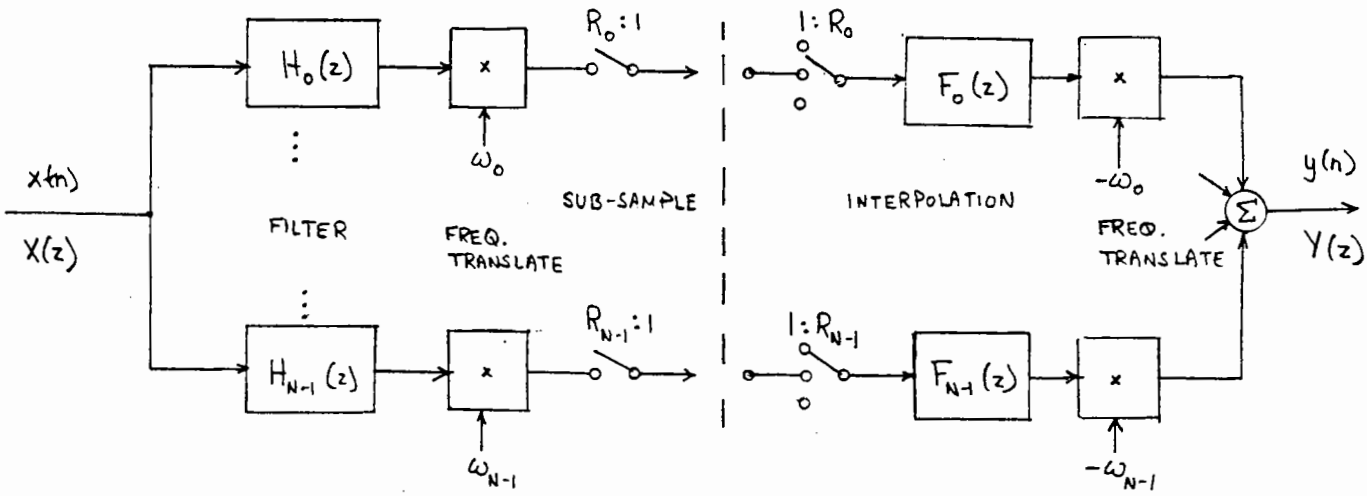


Figure A-8 Sub-Band Analysis

reconstruction side. The overall response of the system may be written as

$$Y(z) = \sum_{k=0}^{N-1} F_k(e^{-j\omega_k z}) \frac{1}{R_k} \sum_{m=0}^{R_k-1} H_k(e^{-j\omega_k e^{j2\pi m/R_k z}}) X(e^{j2\pi m/R_k z}) \quad . \tag{A-25}$$

Reversibility requires that

$$Y(z) = X(z). \tag{A-26}$$

There are additional terms in (A-25) which represent aliasing terms. Clearly these must be zero or cancel for reversibility.

A.5.3 Fractional-Band Filters

One case of interest involves a uniform structure with the same sub-sampling ratio in each sub-band,

$$R_k = N \quad . \tag{A-27}$$

This condition results in a system with no increase in the overall sampling rate. The N element vector of filter outputs is sampled at 1/N of the original sampling rate.

The translation frequencies will be assumed to be equally spaced,

$$\omega_k = \frac{2\pi k}{NT} \quad . \tag{A-28}$$

This additional constraint has the effect of making the first frequency translation unnecessary. The samples selected by the sub-sampler are always multiplied by unity.

The reconstuction process involves interpolation and then frequency translation. The order of these operations can be reversed. For instance if the interpolation filter is low-pass, an equivalent system first frequency translates the sub-band signal and then band-pass filters the signal to interpolate it. This change in order again makes the frequency translation step unnecessary since each non-zero sample is multiplied by unity. The restructured system is shown in Fig. A-9. The reconstruction filters are related to those in Fig. A-8 by

$$G_k(z) = F_k(e^{j\omega}z_k) \quad . \tag{A-29}$$

The output of the fractional-band system is

$$Y(z) = \frac{1}{N} \sum_{m=0}^{N-1} X(e^{j2\pi m/N}z) \sum_{k=0}^{N-1} G_k(z)\ H_k(e^{j2\pi m/N}z) \quad . \tag{A-30}$$

In order for the output to be equal to the input,

$$\sum_{k=0}^{N-1} G_k(z)\ H_k(e^{j2\pi m/N}z) = \begin{cases} N & m=0, \\ 0 & m\neq 0. \end{cases} \tag{A-31}$$

This is a set of simultaneous equations. Given the N filters $H_k(z)$, the system can be solved for the reconstruction filters $G_k(z)$ for each value of z. This procedure is analagous to a generalized sampling expansion described by Papoulis [22].

The simplest case occurs if there is no aliasing due to the sub-sampling. The filters $H_k(z)$ then are ideal band-pass filters each covering a non-overlapping portion of the spectrum of bandwidth $2\pi/N$. The reconstruction filters are also ideal band-pass filters. This

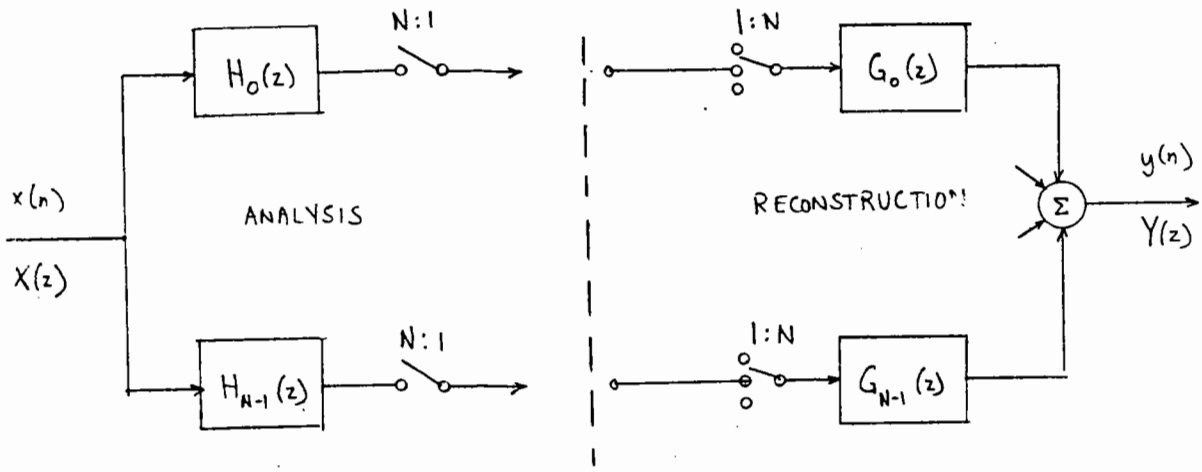


Figure A-9 Fractional-Band Analysis

situation is illustrated in Fig. A-10 for N=3.

For non-ideal filters, aliasing can occur in the individual sub-bands but be cancelled in the final result. The reconstruction filters necessary can be determined by solving the set of simultaneous equations (A-31). If the set of simultaneous equations is singular, the sub-sampled signals are not linearly independent.

The reconstruction filters can be written using Cramer's rule as

$$G_k(z) = N\frac{\Delta_k(z)}{\Delta(z)} \quad . \tag{A-32}$$

The denominator term is the determinant of the set of simultaneous equations and is common to all terms.

The conditions for reversibility can be relaxed slightly by using

$$G_k(z) = \Delta_k(z), \tag{A-33}$$

which will still cancel aliasing terms but give an overall input/ouput relationship

$$Y(z) = N\Delta(z)\ X(z) \quad . \tag{A-34}$$

A.5.4 Half-Band Filters

Half-band filters are fractional-band filters for N=2. The quadrature-mirror filters introduced by Estaban and Galand [23] are a special case of half-band filters. For N=2, (A-30) becomes

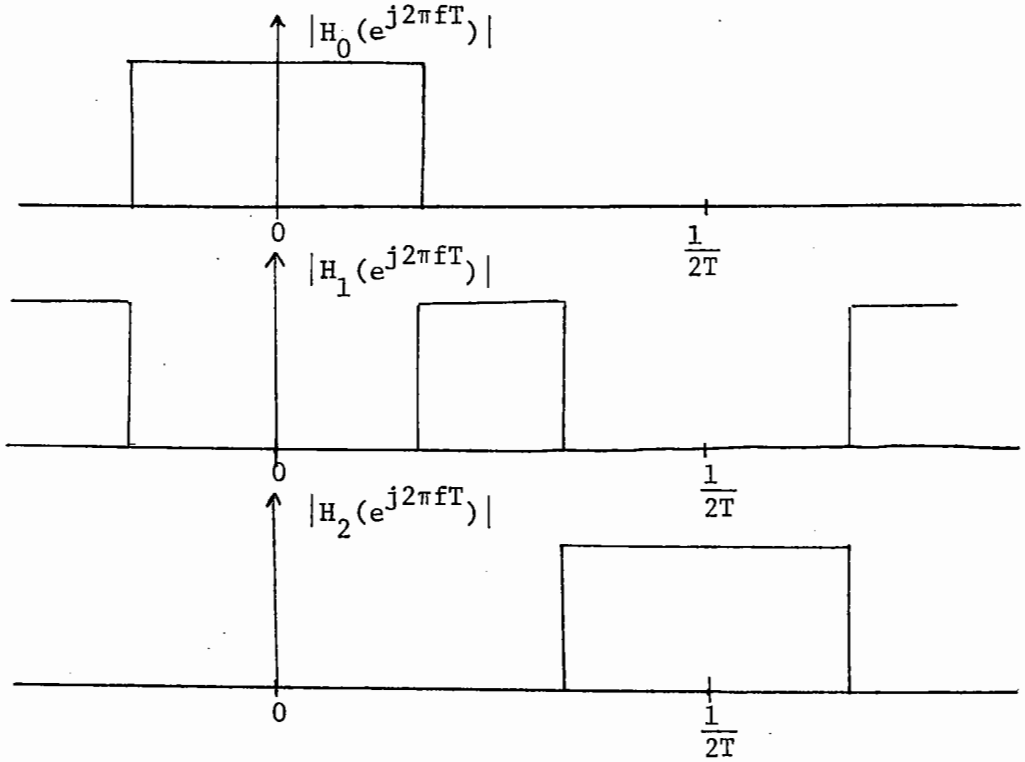


Figure A-10 Fractional-Band Filters (R=3)

$$Y(z) = \frac{1}{2} X(z) \{G_0(z)H_0(z) + G_1(z)H_1(z)\}$$

$$+ \frac{1}{2} X(-z)\{G_0(z)H_0(-z)+ G_1(z)H_1(-z)\} \quad . \qquad \text{(A-35)}$$

The second term represents aliasing. The conditions for cancellation of the aliasing terms give

$$G_0(z) = H_1(-z) \ ,$$

$$G_1(z) = -H_0(-z) \ . \qquad \text{(A-36)}$$

Then

$$Y(z) = \frac{1}{2} X(z) \{H_0(z)H_1(z) - H_0(-z)H_1(-z)\} \ . \qquad \text{(A-37)}$$

Let the half-band filters be linear phase FIR filters,

$$H_0(z) = \sum_{n=0}^{N_0-1} h_0(n) z^{-n} \ , \qquad \text{(A-38)}$$

$$H_1(z) = \sum_{n=0}^{N_1-1} h_1(n) z^{-n} \ .$$

The linear phase constraint requires that

$$h_0(n) = h_0(N_0-n-1) \ , \qquad \text{(A-39)}$$

$$h_1(n) = h_1(N_1-n-1) \ .$$

It is convenient to express the half-band filters in terms of filters

with zero delay,

$$H_0(z) = z^{-\frac{N_0-1}{2}} H_0'(z) , \qquad \text{(A-40)}$$

$$H_1(z) = z^{-\frac{N_1-1}{2}} H_1'(z) ,$$

where $H_0'(e^{j\pi})$ and $H_1'(e^{j\pi})$ are real because of the linear phase constraint. Then

$$Y(z) = \frac{1}{2}X(z)z^{-\frac{N_0+N_1}{2}} z\{H_0'(z)H_1'(z)+(-1)^{-\frac{N_0+N_1}{2}} H_0'(-z)H_1'(-z)\} . \qquad \text{(A-41)}$$

Note that if $(N_0+N_1)/2$ is odd, there is a spectral null at the half sampling frequency, i.e.

$$Y(e^{j\pi}) = 0, \quad \text{for } \frac{N_0+N_1}{2} \text{ odd.} \qquad \text{(A-42)}$$

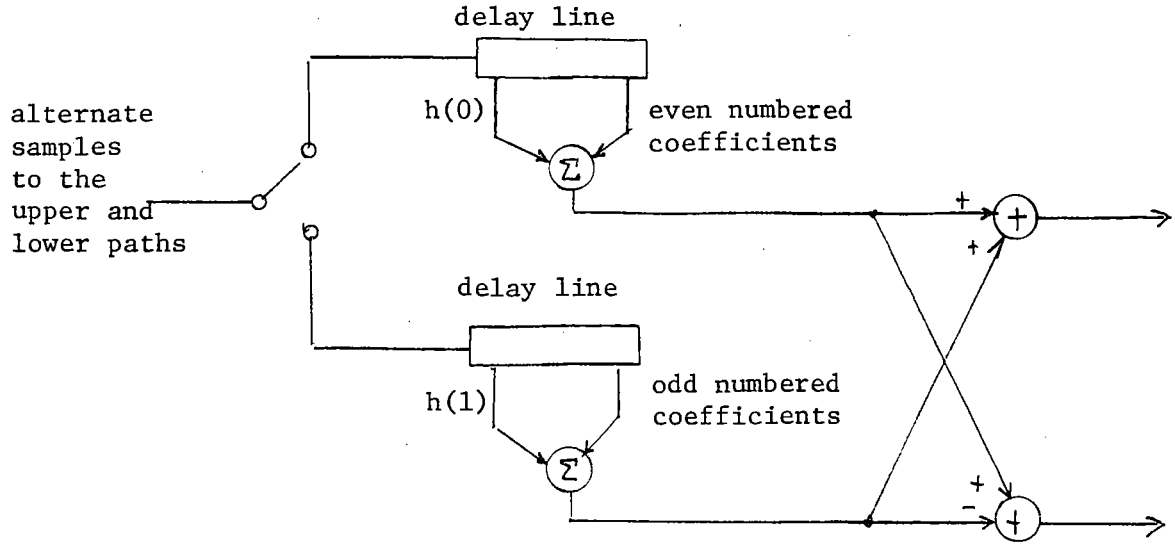
Except for trivial cases, with finite length filters, reversibility cannot be achieved exactly.

Quadrature mirror filters [23] are half-band filters with $H_0(z)=H_1(-z)$. This limits their application to even length filters. However, they have some interesting computational advantages. Note that,

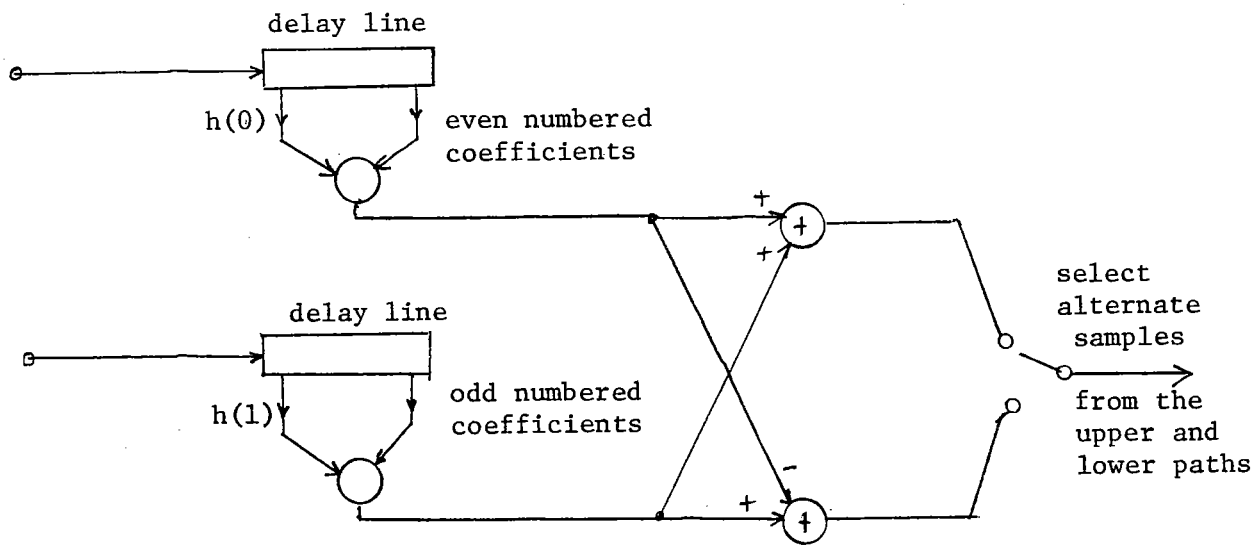
$$h_1(n) = (-1)^n h_0(n). \qquad \text{(A-43)}$$

This means that the half-band filters can be implemented as shown in

Fig. A-11, where the sub-band filters consisting of the even-numbered and odd-numbered coefficients have been separated.



a) Analysis



b) Reconstuction

Figure A-11 Quadrature-Mirror Filters

Appendix B Program Listings

B.1 Introduction

This Appendix gives the listings for the programs used in the simulation of the RELP coder. The listings are divided into three groups. The first group contains the listings for the floating point simulation of both the coder and decoder. These are presented in order that the reader can appreciate the program logic of the coder and decoder, for the integerized versions tend to obfuscate the algorithms somewhat. The second group contains listings of the algorithms employing integer arithmetic. The third group includes common modules used for the coder and decoder, both for the floating point and integer versions. These are given in alphabetical order. They include all modules that are used in the RELP algorithm itself. Modules specific to the file structure of the general purpose simulation system are not shown, though comments in the headers of the main programs indicate their functions.

An additional listing of the relevant program parameters as used in the simulations is also given. These parameters include the filter coefficient values and the quantizer table values used for the simulations.

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       CANAL1
C
C
C PURPOSE:
C       THIS PROGRAM SIMULATES THE ANALYSIS STAGE OF A RELP
C       CODER.
C
C
C DESCRIPTION:
C       THIS PROGRAM PRODUCES A FILE CONTAINING REFLECTION
C       COEFFICIENTS AND A DECIMATED RESIDUAL.
C
C
C PARAMETERS:
C       THE PARAMETERS ARE READ FROM A PARAMETER FILE, DEFAULT
C       NAME CANAL1.PAR.
C
C       NPOLES - NUMBER OF ANALYSIS POLES (MAXIMUM 12)
C       LENFRM - FRAME ADVANCE IN POINTS (MAXIMUM LENWIN-2*NPOLES,
C                MULTIPLE OF 2*IDECIM)
C       LENWIN - WINDOW LENGTH (MAXIMUM 300)
C       IDECIM - RESIDUAL DECIMATION FACTOR (MAXIMUM 8)
C       PRE    - PRE-EMPHASIS / DE-EMPHASIS FACTOR (0 TO 1)
C       WND    - WINDOW COEFFICIENTS (NPOLES+1 VALUES)
C       FLTLP  - ARRAY OF LOW-PASS FILTER COEFFICIENTS
C       NCFLP  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                LOW-PASS FILTER (MAXIMUM 64)
C       FLTSB  - ARRAY OF SUB-BAND FILTER COEFFICIENTS
C       NCFSB  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                SUB-BAND FILTER (EVEN, MAXIMUM 64)
C       FLTHP  - ARRAY OF HIGH-PASS FILTER COEFFICIENTS (NOT USED)
C       NCFHP  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                HIGH-PASS FILTER (MAXIMUM 64) (NOT USED)
C       DDF    - COEFFICIENTS FOR THE DOUBLE DIFFERENCE FILTER
C                (3 VALUES) (NOT USED)
C       NLEVR  - ARRAY CONTAINING THE NUMBER OF QUANTIZER
C                LEVELS FOR THE REFLECTION COEFFICIENT QUANTIZERS
C                (NPOLES VALUES, EACH AT MOST 64)
C       XQR    - ARRAY OF BREAK POINTS FOR THE REFLECTION
C                COEFFICIENT QUANTIZERS (NPOLES VECTORS OF
C                LENGTH 63, WITH THE FIRST NLEVR(I)-1
C                VALUES DEFINED IN THE I'TH VECTOR)
C       YQR    - ARRAY OF QUANTIZER OUTPUT LEVELS FOR THE
C                REFLECTION COEFFICIENT QUANTIZERS (NPOLES
C                VECTORS OF LENGTH 64, WITH THE FIRST
C                NLEVR(I) VALUES DEFINED IN THE I'TH VECTOR)
C       NLEVS  - NUMBER OF LEVELS FOR THE SUB-BAND QUANTIZER
C                (MINIMUM 3, MAXIMUM 64)
C       XQS    - VECTOR OF QUANTIZER BREAK POINTS FOR THE
C                SUB-BAND QUANTIZERS ((NLEVS+1)/2-1 VALUES).
C                THESE ARE SPECIFIED FOR THE POSITIVE PORTION
C                OF THE QUANTIZER RANGE.
C       YQS    - VECTOR OF QUANTIZER OUTPUT LEVELS FOR THE
C                SUB-BAND QUANTIZERS ((NLEVS+1)/2 VALUES).
C                THESE ARE SPECIFIED FOR THE POSITIVE PORTION
C                OF THE QUANTIZER RANGE.
C       QMLT   - VECTOR OF QUANTIZER ADAPTATION MULTIPLIERS
C                FOR THE SUB-BAND QUANTIZER ((NLEVS+1)/2 VALUES).
C       FSMN   - MAXIMUM FULL SCALE VALUE FOR THE SUB-BAND
C                QUANTIZERS
C       FSMX   - MAXIMUM FULL SCALE VALUE FOR THE SUB-BAND
C                QUANTIZERS
C       FSTH   - THRESHOLD FOR THE MID-TREAD / MID-RISER SWITCH
C
C       NOTES:
C         THE PROCESSING DELAY SHOULD BE LESS THAN ONE FRAME.
C         DEFINING
C             IDLYOU = ( 2*IDECIM-1 + (NCFLP-1) + IDECIM*(NCFSB-1) ) / 2,
C         THEN THE REQUIREMENT IS THAT  IDLYOU < LENFRM.
C
C
C ROUTINES REQUIRED:
C       ACORR  - CALCULATE THE AUTO-CORRELATION FOR A VECTOR
C       APCMQ  - ADAPTIVE QUANTIZER
C       AUTOK  - AUTO-CORRELATION ANALYSIS
C       CPARAM - READ THE ANALYIS PARAMETERS
C       DBGSET - SET THE DEBUG OUTPUT DEVICE
C       DEFNM  - SET A DEFAULT FILE NAME
C       FCNVL  - GENERATE A FILTER OUTPUT VALUE
C       GETNAM - DECODE A STRING OF FILE NAMES
C       GETPT1 - GET A NUMBER OF INPUT POINTS (ENTRY POINT GETIN1)
C       GTLINE - READ AN INPUT LINE
C       ISHIFT - SHIFT AN INTEGER VECTOR
C       ITOR   - COPY AN INTEGER ARRAY TO A REAL ARRAY
C       IZERO  - ZERO AN INTEGER ARRAY
C       KTOA   - CONVERT REFLECTION COEFFICIENTS TO FILTER COEFFICIENTS
C       OPNRD  - OPEN AN AUDIO FILE FOR READING
C       OPNWT  - OPEN AN AUDIO FILE FOR WRITING
C       PREEM  - PRE-EMPHASIZE A SIGNAL
C       PRTIBF - PRINT AN ARRAY OF INTEGER DATA
C       PRTRBF - PRINT AN ARRAY OF REAL DATA
C       PUT    - WRITE A NUMBER OF OUTPUT POINTS (ENTRY POINT PUTIN1)
C       QUANTZ - QUANTIZE A VARIABLE
C       RSHIFT - SHIFT A REAL VECTOR
C       RTOI   - CONVERT A REAL VECTOR TO AN INTEGER VECTOR
C       RTOR   - COPY A ONE REAL ARRAY TO ANOTHER
C       VMULN  - MULTIPLY ARRAYS POINT BY POINT
C       WRTFRM - WRITE A FRAME OF COEFFICIENTS
C       ZERO   - ZERO A REAL ARRAY
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       25-JUNE-80
C
C
C UPDATES:
C       81/02/24
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
```

```
      PARAMETER MXPOLE=12,MXWIN=300,MXFILT=64,MXLEV=64,MXDEC=8
      PARAMETER LIST=1,IFILE1=2,IFILE2=3,KBDO=5,CMF=3

      BYTE CSTR(81),FNAME(34,4)

      INTEGER NCA(4),IBUFF(MXWIN),IAPCM(2*MXWIN),NLEVR(MXPOLE)

      REAL WND(MXPOLE+1)
      REAL SIG(MXWIN),RESID(MXFILT+MXWIN-1),
     *     RESX(MXFILT+MXWIN+1),APCM(MXWIN)
      REAL RXX(MXPOLE+1),REFL2(MXPOLE),REFL1(MXPOLE),ACOF(MXPOLE+1)
      REAL FLTLP(MXFILT),FLTSB(MXFILT),FLTHP(MXFILT),DDF(3)
      REAL XQR(MXLEV-1,MXPOLE),YQR(MXLEV,MXPOLE),XQS((MXLEV+1)/2-1),
     *     YQS((MXLEV+1)/2),QMLT((MXLEV+1)/2)



C ********** OPEN THE INPUT AND OUTPUT FILES

C READ THE FILE NAME STRING

50    CALL GTLINE('$FILES (.PAR,.AUD>.CFR): ',CSTR,NCHR,ISRC)
        IF(NCHR.LE.0) GO TO 990

C SEPARATE THE FILE NAMES (INPUT.PAR,INPUT.AUD>OUTPUT.CFR,OUTPUT.LST)

      CALL GETNAM(CSTR,NCHR,'DQ:.PAR,.AUD;0>DQ:[].CFR;0,.LST;0',FNAME,NCA)

C SET UP THE DEBUG OUTPUT DEVICE

      IOUT=KBDO
      CALL DBGSET(LIST,FNAME(1,4),NCA(4),IDBG)
      IF(IDBG.GT.0) IOUT=IDBG

C READ THE PARAMETER FILE (USES LUNS 2 AND 3, TEMPORARILY)

      IF(NCA(1).LE.0) CALL DEFNM('DQ:CANAL1.PAR',13,FNAME(1,1),NCA(1),0)
      CALL CPARAM(FNAME(1,1),NPOLES,LENFRM,LENWIN,IDECIM,PRE,
     *       WND,FLTLP,NCFLP,FLTSB,NCFSB,FLTHP,NCFHP,DDF,
     *       NLEVR,XQR,YQR,NLEVS,XQS,YQS,QMLT,FSMN,FSMX,FSTH

C OPEN THE INPUT AUDIO FILE

      CALL OPNRD(IFILE1,FNAME(1,2),33,NBLKI,SFREQ,IOUT,IER)
        IF(IER.NE.0) GO TO 990

C OPEN THE OUTPUT COEFFICIENT / RESIDUAL FILE

      CALL OPNWT(IFILE2,FNAME(1,3),33,1,SFREQ/IDECIM,IOUT,IER)
        IF(IER.NE.0) GO TO 990

C******************** INITIALIZATION

C INITIALIZE VARIABLES

      FSVL=FSMN
      FSVH=FSMN
      IFRM=-1
      IDLYOU=(2*IDECIM-1+(NCFLP-1)+IDECIM*(NCFSB-1))/2
      LPOFFS=MOD(IDLYOU,IDECIM)
      LSOFFS=MOD(IDLYOU,2*IDECIM)/IDECIM
      LPMEM=NCFLP-1-LPOFFS
      LSMEM=NCFSB-1-LSOFFS

      NSMP=LENFRM/IDECIM
      NSMPO=NSMP-2*(IDLYOU/(2*IDECIM))
      IOVLP=LENWIN-LENFRM
      IOVLPH=IOVLP/2

C INITIALIZE THE DATA VECTORS

      AMEM=0.0
      CALL ZERO(SIG,IOVLP)
      CALL ZERO(RESID,LPMEM)
      CALL ZERO(RESX,LSMEM)
      CALL IZERO(IAPCM,NSMPO)

      CALL ZERO(REFL1,NPOLES)

C INITIALIZE THE INPUT AND OUTPUT ROUTINES

      CALL GETIN1(IFILE1)
      CALL PUTIN1(IFILE2)


      IF(IDBG.GT.0) WRITE(IDBG,1000) NPOLES,LENFRM,LENWIN,IDECIM,
     *    NCFLP,NCFSB,PRE
      CALL PRTRBF('0WINDOW COEFFICIENTS:',WND,NPOLES+1,IDBG)


C*********************
C MAIN LOOP
C*********************

C---------- PRE-EMPHASIZE THE INPUT SIGNAL

101   CALL GETPT1(IBUFF,NOUT,LENFRM)
        IF(NOUT.LE.0) GO TO 900

      CALL ITOR(IBUFF,SIG(IOVLP+1),LENFRM)
      CALL PREEM(SIG(IOVLP+1),SIG(IOVLP+1),LENFRM,PRE,AMEM)


      CALL PRTIBF('0***INPUT DATA:',IBUFF,LENFRM,IDBG)
```

```
C---------- CALCULATE A NEW SET OF PREDICTOR COEFFICIENTS

C CALCULATE AND LAG-WINDOW THE AUTO-CORRELATION

        CALL ACORR(SIG,LENWIN,RXX,NPOLES+1)
        CALL VMULN(RXX,WND,RXX,NPOLES+1)

C AUTO-CORRELATION ANALYSIS

        CALL AUTOK(RXX,NPOLES,REFL2)

C QUANTIZE THE REFLECTION COEFFICIENTS

        DO 220 I=1,NPOLES
          CALL QUANTZ(REFL2(I),L,XQR(1,I),NLEVR(I))
          REFL2(I)=YQR(L,I)
220     CONTINUE

C CONVERT TO FILTER COEFFICIENTS

        CALL KTOA(REFL2,ACOF,NPOLES)

        CALL PRTRBF('  ANALYSIS FRAME DATA:',SIG,LENWIN,IDBG)
        CALL PRTRBF('  AUTO-CORRELATION:',RXX,NPOLES+1,IDBG)
        CALL PRTRBF('  REFLECTION COEFFICIENTS:',REFL2,NPOLES,IDBG)
        CALL PRTRBF('  PREDICTOR COEFFICIENTS:',ACOF,NPOLES+1,IDBG)

C---------- CALCULATE THE RESIDUAL

        DO 440 I=1,LENFRM
          RESID(LPMEM+I)=FCNVL(SIG(IOVLPH+I),ACOF,NPOLES+1,1)
440     CONTINUE

        CALL RSHIFT(SIG,IOVLP,LENFRM)

        CALL PRTRBF('  RESIDUAL:',RESID(LPMEM+1),LENFRM,IDBG)

C---------- LOW-PASS FILTER AND SUB-SAMPLE BY IDECIM

        L=NCFLP
        DO 540 I=1,NSMP
          RESX(LSMEM+I)=FCNVL(RESID(L),FLTLP,NCFLP,1)
          L=L+IDECIM
540     CONTINUE

        CALL RSHIFT(RESID,LPMEM,LENFRM)

        CALL PRTRBF('  SUB-SAMPLED RESIDUAL:',RESX(LSMEM+1),NSMP,IDBG)

C---------- SUB-BAND CODING

        DO 640 I=1,NSMP,2
          ACC1=FCNVL(RESX(NCFSB+I-1),FLTSB(1),NCFSB/2,2)
          ACC2=FCNVL(RESX(NCFSB+I-2),FLTSB(2),NCFSB/2,2)

C APCM CODING OF THE SUB-BANDS

          CALL APCMQ(ACC1+ACC2,APCM(I),(NLEVS+1)/2,QMLT,XQS,YQS,
     *               FSVL,FSMN,FSMX,FSTH)
          CALL APCMQ(ACC2-ACC1,APCM(I+1),(NLEVS+1)/2,QMLT,XQS,YQS,
     *               FSVH,FSMN,FSMX,FSTH)

640     CONTINUE

        CALL RSHIFT(RESX,LSMEM,NSMP)

        CALL PRTRBF('  SUB-BAND POINTS:',APCM,NSMP,IDBG)

C---------- WRITE OUT A FRAME OF COEFFICIENTS AND DATA

        IFRM=IFRM+1
        CALL RTOI(APCM,IAPCM(NSMP0+1),NSMP)
        IF(IFRM.LE.0) GO TO 720

        IF(IFRM.GT.4) IDBG=0
        IF(IDBG.GT.0) WRITE(IDBG,1100) IFRM,(REFL1(I), I=1,NPOLES)
        IF(IDBG.GT.0) WRITE(IDBG,1200) (IAPCM(I), I=1,NSMP)

        CALL WRTFRM(IFRM,NPOLES,NSMP,REFL1,IAPCM)

720     CALL RTOR(REFL1,REFL1,NPOLES)
        CALL ISHIFT(IAPCM,NSMP0,NSMP)

C----------

        GO TO 101

C**********

C FLUSH THE WRITE BUFFER

900     CALL WRTFRM(0,0,0,REFL1,IAPCM)
        IF(ISRC.GE.CMF) GO TO 50

990     STOP
```

```
C DEBUG FORMAT STATEMENTS

1000    FORMAT('  NPOLES =',I5,'  LENFRM =',I5,'  LENWIN =',I5,
     #         '  IDECIM =',I5/'  NCFLP  =',I5,'  NCFSB  =',I5,
     #         '  PRE =',F7.4)
1100    FORMAT('0>>>FRAME',I4,':'/
     #         '    REFLECTION COEFFICIENTS:',10F10.5:/
     #        (28X,10F10.5))
1200    FORMAT('    SUB-BAND POINTS:',16I7:/(20X,16I7))

        END
```



```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       CSYN
C
C
C PURPOSE:
C       THIS PROGRAM SIMULATES THE SYNTHESIS STAGE FOR A RELP
C       CODER.
C
C
C DESCRIPTION:
C       AN OUTPUT SIGNAL IS SYNTHESIZED FROM A DECIMATED RESIDUAL
C       SIGNAL.
C
C
C PARAMETERS:
C       THE PARAMETERS ARE READ FROM A PARAMETER FILE, DEFAULT
C       NAME CSYN.PAR.
C
C       NPOLES - NUMBER OF ANALYSIS POLES (MAXIMUM 12)
C       LENFRM - FRAME ADVANCE IN POINTS (MULTIPLE OF 2*IDE: 1
C                MAXIMUM 300)
C       LENWIN - WINDOW LENGTH (NOT USED)
C       IDECIM - RESIDUAL DECIMATION FACTOR (MAXIMUM 8)
C       PRE    - PRE-EMPHASIS / DE-EMPHASIS FACTOR (0 TO 1`
C       WND    - WINDOW COEFFICIENTS (NOT USED)
C       FLTLP  - ARRAY OF LOW-PASS FILTER COEFFICIENTS
C       NCFLP  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                LOW-PASS FILTER (MINIMUM IDECIM, MAXIMUM 64)
C       FLTSB  - ARRAY OF SUB-BAND FILTER COEFFICIENTS
C       NCFSB  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                SUB-BAND FILTER (EVEN, MAXIMUM 64)
C       FLTHP  - ARRAY OF HIGH-PASS FILTER COEFFICIENTS
C       NCFHP  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                HIGH-PASS FILTER (MAXIMUM 64)
C       DDF    - COEFFICIENTS FOR THE DOUBLE DIFFERENCE FILTER
C                (3 VALUES)
C       NLEVR  - ARRAY CONTAINING THE NUMBER OF QUANTIZER
C                LEVELS FOR THE REFLECTION COEFFICIENT QUANTIZERS
C                (NPOLES VALUES, EACH AT MOST 64)
C       XQR    - ARRAY OF BREAK POINTS FOR THE REFLECTION
C                COEFFICIENT QUANTIZERS (NPOLES VECTORS OF
C                LENGTH 63, WITH THE FIRST NLEVR(I)-1
C                VALUES DEFINED IN THE I'TH VECTOR)
C       YQR    - ARRAY OF QUANTIZER OUTPUT LEVELS FOR THE
C                REFLECTION COEFFICIENT QUANTIZERS (NPOLES
C                VECTORS OF LENGTH 64, WITH THE FIRST
C                NLEVR(I) VALUES DEFINED IN THE I'TH VECTOR)
C       NLEVS  - NUMBER OF LEVELS FOR THE SUB-BAND QUANTIZER
C                (MINIMUM 3, MAXIMUM 64)
C       XQS    - VECTOR OF QUANTIZER BREAK POINTS FOR THE
C                SUB-BAND QUANTIZERS ((NLEVS+1)/2-1 VALUES).
C                THESE ARE SPECIFIED FOR THE POSITIVE PORTION
C                OF THE QUANTIZER RANGE.
C       YQS    - VECTOR OF QUANTIZER OUTPUT LEVELS FOR THE
C                SUB-BAND QUANTIZERS ((NLEVS+1)/2 VALUES).
C                THESE ARE SPECIFIED FOR THE POSITIVE PORTION
C                OF THE QUANTIZER RANGE.
```

```
C       QMLT   - VECTOR OF QUANTIZER ADAPTATION MULTIPLIERS
C                FOR THE SUB-BAND QUANTIZER ((NLEVS+1)/2 VALUES).
C       FSMN   - MAXIMUM FULL SCALE VALUE FOR THE SUB-BAND
C                QUANTIZERS
C       FSMX   - MAXIMUM FULL SCALE VALUE FOR THE SUB-BAND
C                QUANTIZERS
C       FSTH   - THRESHOLD FOR THE MID-TREAD / MID-RISER SWITCH
C
C       NOTES:
C         THE PROCESSING DELAY SHOULD BE LESS THAN ONE FRAME.
C         DEFINING
C             IDLYOU = ( IDECIM*(NCFSB-1) + (NCFLP-1) - (2*IDECIM-1) )/2
C                      + (NCFHP-1)/2 + 1
C         THEN THE REQUIREMENT IS THAT  IDLYOU < LENFRM
C
C
C ROUTINES REQUIRED:
C       CPARAM - READ THE ANALYIS / SYNTHESIS PARAMETERS
C       DBGSET - SET THE DEBUG OUTPUT DEVICE
C       DEEM   - DE-EMPHASIZE A SIGNAL
C       DFFNM  - SET A DEFAULT FILE NAME
C       FCNVL  - GENERATE A FILTERED POINT
C       GETNAM - DECODE A STRING OF FILE NAMES
C       GTLINE - READ AN INPUT LINE
C       KTOP   - CONVERT REFLECTION COEFFICIENTS TO PREDICTOR COEFFICIENTS
C       OPNRD  - OPEN AN AUDIO FILE FOR READING
C       OPNWT  - OPEN AN AUDIO FILE FOR WRITING
C       PRTIBF - PRINT AN ARRAY OF INTEGER DATA
C       PRTRBF - PRINT AN ARRAY OF REAL DATA
C       PUTPT1 - WRITE A NUMBER OF OUTPUT POINTS (ENTRY POINT PUTIN1)
C       REDFRM   READ A FRAME OF COEFFICIENTS
C       RSHIFT - SHIFT A REAL VECTOR
C       RTOI   - TRANSFER FROM A REAL TO AN INTEGER ARRAY
C       ZERO   - ZERO AN ARRAY
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       14-JULY-80
C
C
C UPDATES:
C       81/02/27
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
```

```
      PARAMETER MXPOLE=12,MXFRM=300,MXFILT=64,MXLEV=64,MXDEC=8
      PARAMETER LIST=1,IFILE1=2,IFILE2=3,KBDO=5,CMF=3

      BYTE CSTR(81),FNAME(34,4)

      INTEGER NCA(4),NLEVR(MXPOLE),NCF(MXDEC)
      INTEGER INBUFF(MXFRM),IOBUFF(MXFRM)

      REAL WND(MXFRM)
      REAL FLTSB(MXFILT),FLTEV(MXFILT/2),FLTOD(MXFILT/2),
     *     FLTLP(MXFILT),FLTLPS(MXFILT,MXDEC),FLTHP(MXFILT),
     *     DDF(3)
      REAL SUMBF(MXFILT/2-1+MXFRM/2),DIFBF(MXFILT/2-1+MXFRM/2),
     *     SUBRES(MXFILT-1+MXFRM),RESL((MXFILT-1)/2+1+MXFRM),
     *     BRH(MXFILT-1+MXFRM),RES(MXFRM),SIG(MXPOLE+MXFRM),
     *     DSIG(MXFRM)
      REAL REFL(MXPOLE),PRDCOF(MXPOLE)
      REAL XQR(MXLEV-1,MXPOLE),YQR(MXLEV,MXPOLE),XQS((MXLEV+1)/2-1),
     *     YQS((MXLEV+1)/2),QMLT((MXLEV+1)/2)



C ********** OPEN THE INPUT AND OUTPUT FILES

C READ THE FILE NAME STRING

50    CALL GTLINE('$FILES (.PAR,.CFR>.REL): ',CSTR,NCHR,ISRC)
        IF(NCHR.LE.0) GO TO 990


C SEPARATE THE FILE NAMES (INPUT.PAR,INPUT.CFR>OUTPUT.REL,OUTPUT.LST)

      CALL GETNAM(CSTR,NCHR,'DQ:.PAR,.CFR;0>DQ:[].REL;0,.LST;0',FNAME,NCA)

C SET UP THE DEBUG OUTPUT DEVICE

      CALL DBGSET(LIST,FNAME(1,4),NCA(4),IDBG)
      IOUT=KBDO
      IF(IDBG.GT.0) IOUT=IDBG

C READ THE PARAMETER FILE (USES LUNS 2 AND 3, TEMPORARILY)

      IF(NCA(1).LE.0) CALL DEFNM('DQ:CSYN.PAR',11,FNAME(1,1),NCA(1),0)
      CALL CPARAM(FNAME(1,1),NPOLES,LENFRM,LENWIN,IDECIM,PRE,
     *       WND,FLTLP,NCFLP,FLTSB,NCFSB,FLTHP,NCFHP,DDF,
     *       NLEVR,XQR,YQR,NLEVS,XQS,YQS,QMLT,FSMN,FSMX,FSTH)

C OPEN THE INPUT COEFFICIENT/ RESIDUAL FILE

      CALL OPNRD(IFILE1,FNAME(1,2),33,NBLKI,SFREQ,IOUT,IER)
        IF(IER.NE.0) GO TO 990
```

```
C OPEN THE OUTPUT AUDIO FILE

        CALL OPNWT(IFILE2,FNAME(1,3),33,1,IDECIM*SFREQ,IOUT,IER)
          IF(IER.NE.0) GO TO 990


C******************** INITIALIZATION


C CREATE THE INTERPOLATION FILTERS

        DO 180 I=1,IDECIM
          K=0
          DO 160 L=I,NCFLP,IDECIM
            K=K+1
            FLTLPS(K,I)=FLTLP(L)
160       CONTINUE
          NCF(I)=K
180     CONTINUE


C CREATE THE QUADRATURE MIRROR FILTERS

        DO 200 I=1,NCFSB/2
          FLTEV(I)=FLTSB(2*I)
          FLTOD(I)=FLTSB(2*I-1)
200     CONTINUE


C INITIALIZE THE VARIABLES

        IFRM=0
        AMEM=0.0

        LSMEM=NCFSB/2-1
        LPMEM=(NCFLP-1)/IDECIM
        LHMEM=NCFHP-1
        IDLYHP=(NCFHP-1)/2+1
        IDLYOU=( IDECIM*(NCFSB-1) + (NCFLP-1) - (2*IDECIM-1) )/2 + IDLYHP
        LPOFFS=0
        NSMP=LENFRM/IDECIM


        CALL ZERO(SUMBF,LSMEM)
        CALL ZERO(DIFBF,LSMEM)
        CALL ZERO(SUBRES,LPMEM)
        CALL ZERO(RESL,IDLYHP)
        CALL ZERO(BRH,LHMEM)
        CALL ZERO(SIG,NPOLES)
        CALL ZERO(PRDCOF,NPOLES)


C INITIALIZE THE INPUT/OUTPUT ROUTINES

        CALL GETIN1(IFILE1)
        CALL PUTIN1(IFILE2)


        IF(IDBG.GT.0) WRITE(IDBG,1000) NPOLES,LENFRM,LENWIN,IDECIM,
     *     NCFLP,NCFSB,NCFHP,PRE
```

```
C*********************
C MAIN LOOP
C*********************

C READ THE COEFFICIENTS FOR A FRAME

101     IFRM=IFRM+1
        CALL REDFRM(IF,NP,NS,REFL,INBUFF)
          IF(IF.NE.IFRM .OR. NP.NE.NPOLES .OR. NS.NE.NSMP) GO TO 920

        IF(IFRM.GT.6) IDBG=0
        IF(IDBG.GT.0) WRITE(IDBG,1100) IFRM,(REFL(I), I=1,NPOLES)
        CALL PRTIBF(' INPUT DATA:',INBUFF,NSMP,IDBG)


C--------------- RECONSTRUCT THE RESIDUAL SIGNAL

C FORM THE SUM AND DIFFERENCE SIGNALS

        K=LSMEM+1
        DO 320 I=1,NSMP,2
          SUMBF(K)=INBUFF(I)+INBUFF(I+1)
          DIFBF(K)=INBUFF(I)-INBUFF(I+1)
          K=K+1
320     CONTINUE


C RECONSTRUCT THE BASEBAND RESIDUAL BY SUB-BAND DECODING
C USING QUADRATURE MIRROR FILTERS AND INTERPOLATING (1:2).

        K=LSMEM+1
        DO 340 I=1,NSMP,2
          SUBRES(LPMEM+I)  =FCNVL(SUMBF(K),FLTOD,NCFSB/2,1)
          SUBRES(LPMEM+I+1)=FCNVL(DIFBF(K),FLTEV,NCFSB/2,1)
          K=K+1
340     CONTINUE


        CALL RSHIFT(SUMBF,LSMEM,NSMP/2)
        CALL RSHIFT(DIFBF,LSMEM,NSMP/2)

        CALL PRTRBF(' DECODED:',SUBRES(LPMEM+1),NSMP,IDBG)


C INTERPOLATE (1:IDECIM) TO FORM THE BASEBAND RESIDUAL.

        L=LPMEM
        DO 440 I=1,LENFRM
          RESL(IDLYHP+I)=FCNVL(SUBRES(L+1),FLTLPS(1,LPOFFS+1),
     *                         NCF(LPOFFS+1),1)
          LPOFFS=LPOFFS+1
          IF(LPOFFS.LT.IDECIM) GO TO 440
            LPOFFS=0
            L=L+1
440     CONTINUE


        CALL RSHIFT(SUBRES,LPMEM,NSMP)

        CALL PRTRBF(' LOW PASS RESIDUAL:',RESL(IDLYHP+1),LENFRM,IDBG)
```

```
C REGENERATE THE HIGH FREQUENCY COMPONENTS.

      DO 520  I=1,LENFRM
         BRH(LHMEM+I)=ABS( FCNVL(RESL(IDLYHP+I),DDF,3,1) )
520   CONTINUE


C HIGH-PASS FILTER THE REGENERATED COMPONENTS
C ADD THE LOW-PASS AND HIGH-PASS COMPONENTS

      DO 560 I=1,LENFRM
        RES(I)=RESL(I)+FCNVL(BRH(LHMEM+I),FLTHP,NCFHP,1)
560   CONTINUE


      CALL RSHIFT(BRH,LHMEM,LENFRM)
      CALL RSHIFT(RESL,IDLYHP,LENFRM)

      CALL PRTRBF('  FULL RESIDUAL:',RES,LENFRM,IDBG)


C---------------- FORM THE OUTPUT SIGNAL


C SYNTHESIZE THE OUTPUT SIGNAL

      DO 820 I=1,IDLYOU
        SIG(NPOLES+I)=RES(I)+FCNVL(SIG(NPOLES+I-1),PRDCOF,NPOLES,1)
820   CONTINUE


C UPDATE THE PREDICTOR COEFFICIENTS

      CALL KTOP(REFL,PRDCOF,NPOLES)


C CONTINUE SYNTHESIZING THE OUTPUT SIGNAL

      DO 840 I=IDLYOU+1,LENFRM
        SIG(NPOLES+I)=RES(I)+FCNVL(SIG(NPOLES+I-1),PRDCOF,NPOLES,1)
840   CONTINUE


C DE-EMPHASIZE THE SYNTHESIZED SIGNAL
C WRITE THE RECONSTRUCTED SIGNAL TO THE OUTPUT FILE

      CALL DEEM(SIG(NPOLES+1),DSIG,LENFRM,PRE,AMEM)
      CALL RTOI(DSIG,IOBUFF,LENFRM)
      CALL PUTPT1(IOBUFF,LENFRM)


      CALL PRTRBF('  SYNTHESIZER OUTPUT:',SIG(NPOLES+1),LENFRM,IDBG)
      CALL PRTRBF('  DE-EMPHASIZED OUTPUT SIGNAL:',DSIG,LENFRM,IDBG)
      CALL RSHIFT(SIG,NPOLES,LENFRM)


C----------------------

      GO TO 101


C FLUSH THE INTERNAL BUFFER IN PUTPT1

920   CALL PUTPT1(IOBUFF,0)
      IF(ISRC.GE.CMF) GO TO 50


990   STOP


C DEBUG FORMAT STATEMENTS

1000  FORMAT('  NPOLES =',I5,'  LENFRM =',I5,'  LENWIN =',I5,
     *       '  IDECIM =',I5/'  NCFLP  =',I5,'  NCFSB  =',I5,
     *       '  NCFHP  =',I5,'  PRE =',F7.4)
1100  FORMAT('0>>>FRAME',I4,':'/
     *       '   REFLECTION COEFFICIENTS:'10F10.5:/
     *       (28X,10F10.5))

      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       CANALI
C
C
C PURPOSE:
C       THIS PROGRAM SIMULATES THE ANALYSIS STAGE OF A RELP
C       CODER.
C
C
C DESCRIPTION:
C       THIS PROGRAM PRODUCES A FILE CONTAINING REFLECTION
C       COEFFICIENTS AND A DECIMATED RESIDUAL.
C
C
C PARAMETERS:
C       THE PARAMETERS ARE READ FROM A PARAMETER FILE, DEFAULT
C       NAME CANALI.PAR.
C
C       NPOLES - NUMBER OF ANALYSIS POLES (MAXIMUM 12)
C       LENFRM - FRAME ADVANCE IN POINTS (MAXIMUM LENWIN-2*NPOLES,
C                MULTIPLE OF 2*IDECIM)
C       LENWIN - WINDOW LENGTH (MAXIMUM 300)
C       IDECIM - RESIDUAL DECIMATION FACTOR (MAXIMUM 8)
C       PRE    - PRE-EMPHASIS / DE-EMPHASIS FACTOR (0 TO 1)
C       WND    - WINDOW COEFFICIENTS (NPOLES+1 VALUES)
C       FLTLP  - ARRAY OF LOW-PASS FILTER COEFFICIENTS
C       NCFLP  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                LOW-PASS FILTER (MAXIMUM 64)
C       FLTSB  - ARRAY OF SUB-BAND FILTER COEFFICIENTS
C       NCFSB  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                SUB-BAND FILTER (EVEN, MAXIMUM 64)
C       FLTHP  - ARRAY OF HIGH-PASS FILTER COEFFICIENTS (NOT USED)
C       NCFHP  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                HIGH-PASS FILTER (MAXIMUM 64) (NOT USED)
C       DDF    - COEFFICIENTS FOR THE DOUBLE DIFFERENCE FILTER
C                (3 VALUES) (NOT USED)
C       NLEVR  - ARRAY CONTAINING THE NUMBER OF QUANTIZER
C                LEVELS FOR THE REFLECTION COEFFICIENT QUANTIZERS
C                (NPOLES VALUES, EACH AT MOST 64)
C       XQR    - ARRAY OF BREAK POINTS FOR THE REFLECTION
C                COEFFICIENT QUANTIZERS (NPOLES VECTORS OF
C                LENGTH 63, WITH THE FIRST NLEVR(I)-1
C                VALUES DEFINED IN THE I'TH VECTOR)
C       YQR    - ARRAY OF QUANTIZER OUTPUT LEVELS FOR THE
C                REFLECTION COEFFICIENT QUANTIZERS (NPOLES
C                VECTORS OF LENGTH 64, WITH THE FIRST
C                NLEVR(I) VALUES DEFINED IN THE I'TH VECTOR)
C       NLEVS  - NUMBER OF LEVELS FOR THE SUB-BAND QUANTIZER
C                (MINIMUM 3, MAXIMUM 64)
C       XQS    - VECTOR OF QUANTIZER BREAK POINTS FOR THE
C                SUB-BAND QUANTIZERS ((NLEVS+1)/2-1 VALUES).
C                THESE ARE SPECIFIED FOR THE POSITIVE PORTION
C                OF THE QUANTIZER RANGE.
C       YQS    - VECTOR OF QUANTIZER OUTPUT LEVELS FOR THE
C                SUB-BAND QUANTIZERS ((NLEVS+1)/2 VALUES).
C                THESE ARE SPECIFIED FOR THE POSITIVE PORTION
C                OF THE QUANTIZER RANGE.
C       QMLT   - VECTOR OF QUANTIZER ADAPTATION MULTIPLIERS
C                FOR THE SUB-BAND QUANTIZER ((NLEVS+1)/2 VALUES).
C       FSMN   - MAXIMUM FULL SCALE VALUE FOR THE SUB-BAND
C                QUANTIZERS
C       FSMX   - MAXIMUM FULL SCALE VALUE FOR THE SUB-BAND
C                QUANTIZERS
C       FSTH   - THRESHOLD FOR THE MID-TREAD / MID-RISER SWITCH
C
C       NOTES:
C         THE PROCESSING DELAY SHOULD BE LESS THAN ONE FRAME.
C         DEFINING
C             IDLYOU = ( 2*IDECIM-1 + (NCFLP-1) + IDECIM*(NCFSB-1) ) / 2,
C         THEN THE REQUIREMENT IS THAT  IDLYOU < LENFRM.
C
C
C ROUTINES REQUIRED:
C       APCMQ  - ADAPTIVE QUANTIZER
C       CPARAM - READ THE ANALYIS PARAMETERS
C       DBGSET - SET THE DEBUG OUTPUT DEVICE
C       DEFNM  - SET A DEFAULT FILE NAME
C       GETNAM - DECODE A STRING OF FILE NAMES
C       GETPT1 - GET A NUMBER OF INPUT POINTS (ENTRY POINT GETIN1)
C       GTLINE - READ AN INPUT LINE
C       IACORR - CALCULATE THE AUTO-CORRELATION FOR A VECTOR
C       IAUTOK - AUTO-CORRELATION ANALYSIS
C       IFCNVL - GENERATE A FILTER OUTPUT VALUE
C       IKTOA  - CONVERT REFLECTION COEFFICIENTS TO FILTER COEFFICIENTS
C       IQUANT - QUANTIZE A VARIABLE
C       ISHIFT - SHIFT AN INTEGER VECTOR
C       ITOI   - COPY AN INTEGER ARRAY
C       ITOR   - COPY AN INTEGER ARRAY TO A REAL ARRAY
C       IVMULN - MULTIPLY ARRAYS POINT BY POINT
C       IZERO  - ZERO AN INTEGER ARRAY
C       OPNRD  - OPEN AN AUDIO FILE FOR READING
C       OPNWT  - OPEN AN AUDIO FILE FOR WRITING
C       PREEM  - PRE-EMPHASIZE A SIGNAL
C       PRTIBF - PRINT AN ARRAY OF INTEGER DATA
C       PRTRBF - PRINT AN ARRAY OF REAL DATA
C       PUTPT1 - WRITE A NUMBER OF OUTPUT POINTS (ENTRY POINT PUTIN1)
C       RTOI   - CONVERT A REAL VECTOR TO AN INTEGER VECTOR
C       VSMULT - MULTIPLY A REAL ARRAY BY A SCALAR
C       WRTFRM - WRITE A FRAME OF COEFFICIENTS
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       25-JUNE-80
C
C
C UPDATES:
C       81/03/09
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
```

```
      PARAMETER MXPOLE=12,MXWIN=300,MXFILT=64,MXLEV=64,MXDEC=8
      PARAMETER LIST=1,IFILE1=2,IFILE2=3,KBDO=5,CMF=3

      BYTE CSTR(81),FNAME(34,4)

      INTEGER NCA(4)
      INTEGER NLEVR(MXPOLE),IXQR(MXLEV-1,MXPOLE),IYQR(MXLEV,MXPOLE)
      INTEGER*4 IFCNVL,ID
      INTEGER*2 IEXTRT
      INTEGER*2 IFLTLP(MXFILT),IFLTSB(MXFILT)
      INTEGER*2 IRXX(MXPOLE+1),IWND(MXPOLE+1)
      INTEGER*2 IREFL1(MXPOLE),IREFL2(MXPOLE),IACOF(MXPOLE+1)
      INTEGER*2 IBUFF(MXWIN),ISIG(MXWIN),
     *          IRESID(MXFILT+MXWIN-1),IRESX(MXFILT+MXWIN-1),
     *          IAPCM(MXWIN)

      REAL WND(MXPOLE+1)
      REAL SIG(MXWIN),APCM(MXWIN)
      REAL REFL(MXPOLE)
      REAL FLTLP(MXFILT),FLTSB(MXFILT),FLTHP(MXFILT),DDF(3)
      REAL XQR(MXLEV-1,MXPOLE),YQR(MXLEV,MXPOLE),XQS((MXLEV+1)/2-1),
     *     YQS((MXLEV+1)/2),QMLT((MXLEV+1)/2)


C ********** OPEN THE INPUT AND OUTPUT FILES

C READ THE FILE NAME STRING

50    CALL GTLINE('$FILES (.PAR,.AUD>.CFR): ',CSTR,NCHR,ISRC)
        IF(NCHR.LE.0) GO TO 990

C SEPARATE THE FILE NAMES (INPUT.PAR,INPUT.AUD>OUTPUT.CFR,OUTPUT.LST)

      CALL GETNAM(CSTR,NCHR,'DQ:.PAR,.AUD;0>DQ:[].CFR;0,.LST;0',FNAME,NCA)

C SET UP THE DEBUG OUTPUT DEVICE

      IOUT=KBDO
      CALL DBGSET(LIST,FNAME(1,4),NCA(4),IDBG)
      IF(IDBG.GT.0) IOUT=IDBG

C READ THE PARAMETER FILE (USES LUNS 2 AND 3, TEMPORARILY)

      IF(NCA(1).LE.0) CALL DEFNM('DQ:CANALI.PAR',13,FNAME(1,1),NCA(1),0)
      CALL CPARAM(FNAME(1,1),NPOLES,LENFRM,LENWIN,IDECIM,PRE,
     *       WND,FLTLP,NCFLP,FLTSB,NCFSB,FLTHP,NCFHP,DDF,
     *       NLEVR,XQR,YQR,NLEVS,XQS,YQS,QMLT,FSMN,FSMX,FSTH)

C OPEN THE INPUT AUDIO FILE

      CALL OPNRD(IFILE1,FNAME(1,2),33,NBLKI,SFREQ,IOUT,IER)
        IF(IER.NE.0) GO TO 990

C OPEN THE OUTPUT COEFFICIENT / RESIDUAL FILE

      CALL OPNWT(IFILE2,FNAME(1,3),33,1,SFREQ/IDECIM,IOUT,IER)
        IF(IER.NE.0) GO TO 990


C********** INITIALIZATION

C CREATE THE WINDOW

      CALL RTOI(WND,IWND,NPOLES+1)

C CREATE THE FILTERS

      CALL RTOI(FLTLP,IFLTLP,NCFLP)
      CALL RTOI(FLTSB,IFLTSB,NCFSB)

C CREATE THE QUANTIZER TABLES

      DO 80 I=1,NPOLES
        CALL VSMULT(XQR(1,I),32768.,XQR(1,I),NLEVR(I)-1)
        CALL RTOI(XQR(1,I),IXQR(1,I),NLEVR(I)-1)
        CALL VSMULT(YQR(1,I),32768.,YQR(1,I),NLEVR(I))
        CALL RTOI(YQR(1,I),IYQR(1,I),NLEVR(I))
80    CONTINUE

C INITIALIZE VARIABLES

      FSVL=FSMN
      FSVH=FSMN
      IFRM=-1
      IDLYOU=(2*IDECIM-1+(NCFLP-1)+IDECIM*(NCFSB-1))/2
      LPOFFS=MOD(IDLYOU,IDECIM)
      LSOFFS=MOD(IDLYOU,2*IDECIM)/IDECIM
      LPMEM=NCFLP-1-LPOFFS
      LSMEM=NCFSB-1-LSOFFS

      NSMP=LENFRM/IDECIM
      NSMPO=NSMP-2*(IDLYOU/(2*IDECIM))
      IOVLP=LENWIN-LENFRM
      IOVLPH=IOVLP/2

C INITIALIZE THE DATA VECTORS

      AMEM=0.0
      CALL IZERO(ISIG,IOVLP)
      CALL IZERO(IRESID,LPMEM)
      CALL IZERO(IRESX,LSMEM)
      CALL IZERO(IAPCM,NSMPO)

      CALL IZERO(IREFL1,NPOLES)
```

```
C INITIALIZE THE INPUT AND OUTPUT ROUTINES

      CALL GETIN1(IFILE1)
      CALL PUTIN1(IFILE2)


      IF(IDBG.GT.0) WRITE(IDBG,1000) NPOLES,LENFRM,LENWIN,IDECIM,
     *    NCFLP,NCFSB,PRE
      CALL PRTIBF('0WINDOW COEFFICIENTS:',IWND,NPOLES+1,IDBG)


C*********************
C MAIN LOOP
C*********************


C---------- PRE-EMPHASIZE THE INPUT SIGNAL

101   CALL GETPT1(IBUFF,NOUT,LENFRM)
        IF(NOUT.LE.0) GO TO 900

      CALL ITOR(IBUFF,SIG,LENFRM)
      CALL VSMULT(SIG,0.125,SIG,LENFRM)
      CALL PREEM(SIG,SIG,LENFRM,PRE,AMEM)
      CALL RTOI(SIG,ISIG(IOVLP+1),LENFRM)


      CALL PRTIBF('0***INPUT DATA:',IBUFF,LENFRM,IDBG)


C---------- CALCULATE A NEW SET OF PREDICTOR COEFFICIENTS

C     ALCULATE AND LAG-WINDOW THE AUTO-CORRELATION

      CALL IACORR(ISIG,LENWIN,IRXX,NPOLES+1)
      CALL IVMULN(IRXX,IWND,IRXX,NPOLES+1)


C AUTO-CORRELATION ANALYSIS

      CALL IAUTOK(IRXX,NPOLES,IREFL2)


C QUANTIZE THE REFLECTION COEFFICIENTS

      DO 220 I=1,NPOLES
        CALL IQUANT(IREFL2(I),L,IXQR(1,I),NLEVR(I))
        IREFL2(I)=IYQR(L,I)
220   CONTINUE


C CONVERT TO FILTER COEFFICIENTS

      CALL IKTOA(IREFL2,IACOF,NPOLES)


      CALL PRTIBF('  ANALYSIS FRAME DATA:',ISIG,LENWIN,IDBG)
      CALL PRTIBF('  AUTO-CORRELATION:',IRXX,NPOLES+1,IDBG)
      CALL PRTIBF('  REFLECTION COEFFICIENTS:',IREFL2,NPOLES,IDBG)
      CALL PRTIBF('  PREDICTOR COEFFICIENTS:',IACOF,NPOLES+1,IDBG)


C---------- CALCULATE THE RESIDUAL

      DO 440 I=1,LENFRM
        ID=IFCNVL(ISIG(IOVLPH+I),IACOF,NPOLES+1,1)
        IRESID(LPMEM+I)=IEXTRT(ID,6)
440   CONTINUE


      CALL ISHIFT(ISIG,IOVLP,LENFRM)

      CALL PRTIBF('  RESIDUAL:',IRESID(LPMEM+1),LENFRM,IDBG)


C---------- LOW-PASS FILTER AND SUB-SAMPLE BY IDECIM

      L=NCFLP
      DO 540 I=1,NSMP
        ID=IFCNVL(IRESID(L),IFLTLP,NCFLP,1)
        IRESX(LSMEM+I)=IEXTRT(ID,1)
        L=L+IDECIM
540   CONTINUE


      CALL ISHIFT(IRESID,LPMEM,LENFRM)

      CALL PRTIBF('  SUB-SAMPLED RESIDUAL:',IRESX(LSMEM+1),NSMP,IDBG)


C---------- SUB-BAND CODING

      DO 640 I=1,NSMP,2
        ID=IFCNVL(IRESX(NCFSB+I-1),IFLTSB(1),NCFSB/2,2)
        IACC1=IEXTRT(ID,1)
        ID=IFCNVL(IRESX(NCFSB+I-2),IFLTSB(2),NCFSB/2,2)
        IACC2=IEXTRT(ID,1)


C APCM CODING OF THE SUB-BANDS

        TEMP=IACC1+IACC2
        CALL APCMQ(TEMP,APCM(I),(NLEVS+1)/2,QMLT,XQS,YQS,
     *             FSVL,FSMN,FSMX,FSTH)
        TEMP=IACC2-IACC1
        CALL APCMQ(TEMP,APCM(I+1),(NLEVS+1)/2,QMLT,XQS,YQS,
     *             FSVH,FSMN,FSMX,FSTH)

640   CONTINUE


      CALL ISHIFT(IRESX,LSMEM,NSMP)

      CALL PRTBBF('  SUB-BAND POINTS:',APCM,NSMP,IDBG)
```

```
C---------- WRITE OUT A FRAME OF COEFFICIENTS AND DATA

        IFRM=IFRM+1
        CALL RTOI(APCM,IAPCM(NSMPO+1),NSMP)
        IF(IFRM.LE.0) GO TO 720

        IF(IFRM.GT.4) IDBG=0
        CALL ITOR(IREFL1,REFL,NPOLES)
        CALL VSMULT(REFL,1./32768.,REFL,NPOLES)
        IF(IDBG.GT.0) WRITE(IDBG,1100) IFRM,(REFL(I), I=1,NPOLES)
        IF(IDBG.GT.0) WRITE(IDBG,1200) (IAPCM(I), I=1,NSMP)

        CALL WRTFRM(IFRM,NPOLES,NSMP,REFL,IAPCM)

720     CALL ITOI(IREFL2,IREFL1,NPOLES)
        CALL ISHIFT(IAPCM,NSMPO,NSMP)


C----------

        GO TO 101


C********************


C FLUSH THE WRITE BUFFER

900     CALL WRTFRM(0,0,0,REFL2,IAPCM)
        IF(ISRC.GE.CMF) GO TO 50


990     STOP

C DEBUG FORMAT STATEMENTS

1000    FORMAT('  NPOLES =',I5,'  LENFRM =',I5,'  LENWIN =',I5,
     *         '  IDECIM =',I5/'  NCFLP  =',I5,'  NCFSB  =',I5,
     *         '  PRE =',F7.4)
1100    FORMAT('0>>>FRAME',I4,':'/
     *         '    REFLECTION COEFFICIENTS:',10F10.5:/
     *        (28X,10F10.5))
1200    FORMAT('    SUB-BAND POINTS:',16I7:/(20X,16I7))

        END
```



```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       CSYNI
C
C
C PURPOSE:
C       THIS PROGRAM SIMULATES THE SYNTHESIS STAGE FOR A RELP
C       CODER USING SCALED INTEGER ARITHMETIC.
C
C
C DESCRIPTION:
C       AN OUTPUT SIGNAL IS SYNTHESIZED FROM A DECIMATED RESIDUAL
C       SIGNAL.
C
C
C PARAMETERS:
C       THE PARAMETERS ARE READ FROM A PARAMETER FILE, DEFAULT
C       NAME CSYNI.PAR.
C
C       NPOLES - NUMBER OF ANALYSIS POLES (MAXIMUM 12)
C       LENFRM - FRAME ADVANCE IN POINTS (MULTIPLE OF 2*IDECIM,
C                MAXIMUM 300)
C       LENWIN - WINDOW LENGTH (NOT USED)
C       IDECIM - RESIDUAL DECIMATION FACTOR (MAXIMUM 8)
C       PRE    - PRE-EMPHASIS / DE-EMPHASIS FACTOR (0 TO 1)
C       WND    - WINDOW COEFFICIENTS (NOT USED)
C       FLTLP  - ARRAY OF LOW-PASS FILTER COEFFICIENTS
C       NCFLP  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                LOW-PASS FILTER (MINIMUM IDECIM, MAXIMUM 64)
C       FLTSB  - ARRAY OF SUB-BAND FILTER COEFFICIENTS
C       NCFSB  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                SUB-BAND FILTER (EVEN, MAXIMUM 64)
C       FLTHP  - ARRAY OF HIGH-PASS FILTER COEFFICIENTS
C       NCFHP  - NUMBER OF FILTER COEFFICIENTS FOR THE
C                HIGH-PASS FILTER (MAXIMUM 64)
C       DDF    - COEFFICIENTS FOR THE DOUBLE DIFFERENCE FILTER
C                (3 VALUES)
C       NLEVR  - ARRAY CONTAINING THE NUMBER OF QUANTIZER
C                LEVELS FOR THE REFLECTION COEFFICIENT QUANTIZERS
C                (NPOLES VALUES, EACH AT MOST 64)
C       XQR    - ARRAY OF BREAK POINTS FOR THE REFLECTION
C                COEFFICIENT QUANTIZERS (NPOLES VECTORS OF
C                LENGTH 63, WITH THE FIRST NLEVR(I)-1
C                VALUES DEFINED IN THE I'TH VECTOR)
C       YQR    - ARRAY OF QUANTIZER OUTPUT LEVELS FOR THE
C                REFLECTION COEFFICIENT QUANTIZERS (NPOLES
C                VECTORS OF LENGTH 64, WITH THE FIRST
C                NLEVR(I) VALUES DEFINED IN THE I'TH VECTOR)
C       NLEVS  - NUMBER OF LEVELS FOR THE SUB-BAND QUANTIZER
C                (MINIMUM 3, MAXIMUM 64)
C       XQS    - VECTOR OF QUANTIZER BREAK POINTS FOR THE
C                SUB-BAND QUANTIZERS ((NLEVS+1)/2-1 VALUES).
C                THESE ARE SPECIFIED FOR THE POSITIVE PORTION
C                OF THE QUANTIZER RANGE.
C       YQS    - VECTOR OF QUANTIZER OUTPUT LEVELS FOR THE
C                SUB-BAND QUANTIZERS ((NLEVS+1)/2 VALUES).
C                THESE ARE SPECIFIED FOR THE POSITIVE PORTION
C                OF THE QUANTIZER RANGE.
```

```
C       QMLT   - VECTOR OF QUANTIZER ADAPTATION MULTIPLIERS
C                FOR THE SUB-BAND QUANTIZER ((NLEVS+1)/2 VALUES).
C       FSMN   - MAXIMUM FULL SCALE VALUE FOR THE SUB-BAND
C                QUANTIZERS
C       FSMX   - MAXIMUM FULL SCALE VALUE FOR THE SUB-BAND
C                QUANTIZERS
C       FSTH   - THRESHOLD FOR THE MID-TREAD / MID-RISER SWITCH
C
C       NOTES:
C         THE PROCESSING DELAY SHOULD BE LESS THAN ONE FRAME.
C         DEFINING
C             IDLYOU = ( IDECIM*(NCFSB-1) + (NCFLP-1) - (2*IDECIM-1) )/2
C                      + (NCFHP-1)/2 + 1
C         THEN THE REQUIREMENT IS THAT  IDLYOU < LENFRM
C
C
C ROUTINES REQUIRED:
C       CPARAM - READ THE ANALYIS / SYNTHESIS PARAMETERS
C       DBGSET - SET THE DEBUG OUTPUT DEVICE
C       DEEM   - DE-EMPHASIZE A SIGNAL
C       DEFNM  - SET A DEFAULT FILE NAME
C       GETNAM - DECODE A STRING OF FILE NAMES
C       GETPT1 - READ A NUMBER OF INPUT POINTS (ENTRY GETIN1)
C       GTLINE - READ AN INPUT LINE
C       IFCNVL - GENERATE A FILTERED VALUE
C       IKTOP  - CONVERT REFLECTION COEFFICIENTS TO PREDICTOR COEFFICIENTS
C       ISHIFT - SHIFT AN INTEGER VECTOR
C       ITOR   - CONVERT INTEGER VALUES TO REAL VALUES
C       IZERO  - ZERO AN INTEGER ARRAY
C       OPNRD  - OPEN AN AUDIO FILE FOR READING
C       OPNWT  - OPEN AN AUDIO FILE FOR WRITING
C       PRTIBF - PRINT AN ARRAY OF INTEGER DATA
C       PRTRBF - PRINT AN ARRAY OF REAL DATA
C       PUTPT1 - WRITE A NUMBER OF OUTPUT POINTS (ENTRY POINT PUTIN1)
C       READFRM - READ A FRAME OF COEFFICIENTS
C       RTOI   - TRANSFER FROM A REAL TO AN INTEGER ARRAY
C       VSMULT - MULTIPLY A VECTOR BY A SCALAR
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/03/02
C
C
C UPDATES:
C       81/03/09
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
      PARAMETER MXPOLE=12,MXFRM=300,MXFILT=64,MXLEV=64,MXDEC=8
      PARAMETER LIST=1,IFILE1=2,IFILE2=3,KBDO=5,CMF=3

      BYTE CSTR(81),FNAME(34,4)

      INTEGER NCA(4),NLEVR(MXPOLE),NCF(MXDEC)
      INTEGER*4 IFCNVL,ID
      INTEGER*2 IEXTRT
      INTEGER*2 IFLTEV(MXFILT/2),IFLTOD(MXFILT/2),
     *          IFLTLP(MXFILT,MXDEC),IFLTHP(MXFILT),IDDF(3)
      INTEGER*2 INBUFF(MXFRM),ISUMBF(MXFILT/2-1+MXFRM),
     *          IDIFBF(MXFILT/2-1+MXFRM),ISUBRS(MXFILT-1+MXFRM),
     *          IBRH(MXFILT-1+MXFRM),IRES(MXFRM),
     *          IRESL((MXFILT-1)/2+1+MXFRM),ISIG(MXPOLE+MXFRM),
     *          IOBUFF(MXFRM)
      INTEGER*2 IREFL(MXPOLE),IPRDCF(MXPOLE)

      REAL WND(MXFRM)
      REAL FLTSB(MXFILT),FLTLP(MXFILT),FLTHP(MXFILT),DDF(3)
      REAL SIG(MXFRM),DSIG(MXFRM)
      REAL REFL(MXPOLE)
      REAL XQR(MXLEV-1,MXPOLE),YQR(MXLEV,MXPOLE),XQS((MXLEV+1)/2-1),
     *     YQS((MXLEV+1)/2),QMLT((MXLEV+1)/2)



C ***********  OPEN THE INPUT AND OUTPUT FILES

C READ THE FILE NAME STRING

50    CALL GTLINE('$FILES (.PAR,.CFR>.REL): ',CSTR,NCHR,ISRC)
        IF(NCHR.LE.0) GO TO 990

C SEPARATE THE FILE NAMES (INPUT.PAR,INPUT.CFR>OUTPUT.REL,OUTPUT.LST)

      CALL GETNAM(CSTR,NCHR,'DQ:.PAR,.CFR;0>DQ:[].REL;0,.LST;0',FNAME,NCA)

C SET UP THE DEBUG OUTPUT DEVICE

      CALL DBGSET(LIST,FNAME(1,4),NCA(4),IDBG)
      IOUT=KBDO
      IF(IDBG.GT.0) IOUT=IDBG

C READ THE PARAMETER FILE (USES LUNS 2 AND 3, TEMPORARILY)

      IF(NCA(1).LE.0) CALL DEFNM('DQ:CSYN1.PAR',12,FNAME(1,1),NCA(1),0)
      CALL CPARAM(FNAME(1,1),NPOLES,LENFRM,LENWIN,IDECIM,PRE,
     *       WND,FLTLP,NCFLP,FLTSB,NCFSB,FLTHP,NCFHP,DDF,
     *       NLEVR,XQR,YQR,NLEVS,XQS,YQS,QMLT,FSMN,FSMX,FSTH)

C OPEN THE INPUT COEFFICIENT/ RESIDUAL FILE
```

```
      CALL OPNRD(IFILE1,FNAME(1,2),33,NBLKI,SFREQ,IOUT,IER)
        IF(IER.NE.0) GO TO 990

C OPEN THE OUTPUT AUDIO FILE

      CALL OPNWT(IFILE2,FNAME(1,3),33,1,IDECIM*SFREQ,IOUT,IER)
        IF(IER.NE.0) GO TO 990


C********************* INITIALIZATION

C CREATE THE INTERPOLATION FILTERS

      DO 180 I=1,IDECIM
        K=0
        DO 160 L=I,NCFLP,IDECIM
          K=K+1
          IFLTLP(K,I)=NINT(FLTLP(L))
160     CONTINUE
        NCF(I)=K
180   CONTINUE

C CREATE THE QUADRATURE MIRROR FILTERS

      DO 200 I=1,NCFSB/2
        IFLTEV(I)=NINT(FLTSB(2*I))
        IFLTOD(I)=NINT(FLTSB(2*I-1))
200   CONTINUE

C CREATE THE HIGH-PASS FILTER AND DOUBLE DIFFERENCE FILTER

      CALL RTOI(FLTHP,IFLTHP,NCFHP)
      CALL RTOI(DDF,IDDF,3)

C INITIALIZE THE VARIABLES

      IFRM=0
      AMEM=0.0

      LSMEM=NCFSB/2-1
      LPMEM=(NCFLP-1)/IDECIM
      LHMEM=NCFHP-1
      IDLYHP=(NCFHP-1)/2+1
      IDLYOU=( IDECIM*(NCFSB-1) + (NCFLP-1) - (2*IDECIM-1) )/2 + IDLYHP
      LPOFFS=0
      NSMP=LENFRM/IDECIM


      CALL IZERO(ISUMBF,LSMEM)
      CALL IZERO(IDIFBF,LSMEM)
      CALL IZERO(ISUBRS,LPMEM)
      CALL IZERO(IRESL,IDLYHP)
      CALL IZERO(IBRH,LHMEM)
      CALL IZERO(ISIG,NPOLES)

      CALL IZERO(IPRDCF,NPOLES)

C INITIALIZE THE INPUT/OUTPUT ROUTINES

      CALL GETIN1(IFILE1)
      CALL PUTIN1(IFILE2)


      IF(IDBG.GT.0) WRITE(IDBG,1000) NPOLES,LENFRM,LENWIN,IDECIM,
     *    NCFLP,NCFSB,NCFHP,PRE


C*********************
C MAIN LOOP
C*********************

C READ THE COEFFICIENTS FOR A FRAME

101   IFRM=IFRM+1
      CALL REDFRM(IF,NP,NS,REFL,INBUFF)
        IF(IF.NE.IFRM .OR. NP.NE.NPOLES .OR. NS.NE.NSMP) GO TO 920

      CALL VSMULT(REFL,32768.,REFL,NPOLES)
      CALL RTOI(REFL,IREFL,NPOLES)

      IF(IFRM.GT.6) IDBG=0
      IF(IDBG.GT.0) WRITE(IDBG,1100) IFRM,(IREFL(I), I=1,NPOLES)
      CALL PRTIBF(' INPUT DATA:',INBUFF,NSMP,IDBG)


C---------- RECONSTRUCT THE RESIDUAL SIGNAL

C FORM THE SUM AND DIFFERENCE SIGNALS

      K=LSMEM+1
      DO 320 I=1,NSMP,2
        ISUMBF(K)=4*(INBUFF(I)+INBUFF(I+1))
        IDIFBF(K)=4*(INBUFF(I)-INBUFF(I+1))
        K=K+1
320   CONTINUE

C RECONSTRUCT THE BASEBAND RESIDUAL BY SUB-BAND DECODING
C USING QUADRATURE MIRROR FILTERS AND INTERPOLATING (1:2).

      K=LSMEM+1
      DO 340 I=1,NSMP,2
        ID=IFCNVL(ISUMBF(K),IFLTOD,NCFSB/2,1)
        ISUBRS(LPMEM+I)  =IEXTRT(ID,1)
        ID=IFCNVL(IDIFBF(K),IFLTEV,NCFSB/2,1)
        ISUBRS(LPMEM+I+1)=IEXTRT(ID,1)
        K=K+1
340   CONTINUE


      CALL ISHIFT(ISUMBF,LSMEM,NSMP/2)
      CALL ISHIFT(IDIFBF,LSMEM,NSMP/2)

      CALL PRTIBF(' DECODED:',ISUBRS(LPMEM+1),NSMP,IDBG)
```

```
C INTERPOLATE (1:IDECIM) TO FORM THE BASEBAND RESIDUAL.

        L=LPMEM
        DO 440 I=1,LENFRM
          ID=IFCNVL(ISUBRS(L+1),IFLTLP(1,LPOFFS+1),NCF(LPOFFS+1),1)
          IRESL(IDLYHP+I)=IEXTRT(ID,1)
          LPOFFS=LPOFFS+1
          IF(LPOFFS.LT.IDECIM) GO TO 440
            LPOFFS=0
            L=L+1
440     CONTINUE


        CALL ISHIFT(ISUBRS,LPMEM,NSMP)

        CALL PRTIBF(' LOW PASS RESIDUAL:',IRESL(IDLYHP+1),LENFRM,IDBG)


C REGENERATE THE HIGH FREQUENCY COMPONENTS.

        DO 520 I=1,LENFRM
           ID=IFCNVL(IRESL(IDLYHP+I),IDDF,3,1)
           IBRH(LHMEM+I)=IABS( IEXTRT(ID,2) )
520     CONTINUE


C HIGH-PASS FILTER THE REGENERATED COMPONENTS
C ADD THE LOW-PASS AND HIGH-PASS COMPONENTS

        DO 560 I=1,LENFRM
          ID=IFCNVL(IBRH(LHMEM+I),IFLTHP,NCFHP,1)
          IRES(I)=IRESL(I)+IEXTRT(ID,7)
560     CONTINUE


        CALL ISHIFT(IBRH,LHMEM,LENFRM)
        CALL ISHIFT(IRESL,IDLYHP,LENFRM)

        CALL PRTIBF(' FULL RESIDUAL:',IRES,LENFRM,IDBG)

C---------------- FORM THE OUTPUT SIGNAL

C SYNTHESIZE THE OUTPUT SIGNAL

        DO 820 I=1,IDLYOU
          ID=IFCNVL(ISIG(NPOLES+I-1),IPRDCF,NPOLES,1)
          ISIG(NPOLES+I)=IRES(I)+IEXTRT(ID,3)
820     CONTINUE

C UPDATE THE PREDICTOR COEFFICIENTS

        CALL IKTOP(IREFL,IPRDCF,NPOLES)

C CONTINUE SYNTHESIZING THE OUTPUT SIGNAL

        DO 840 I=IDLYOU+1,LENFRM
          ID=IFCNVL(ISIG(NPOLES+I-1),IPRDCF,NPOLES,1)
          ISIG(NPOLES+I)=IRES(I)+IEXTRT(ID,3)
840     CONTINUE


C DE-EMPHASIZE THE SYNTHESIZED SIGNAL
C WRITE THE RECONSTRUCTED SIGNAL TO THE OUTPUT FILE

        CALL ITOR(ISIG(NPOLES+1),SIG,LENFRM)
        CALL VSMULT(SIG,0.25,SIG,LENFRM)
        CALL DEEM(SIG,DSIG,LENFRM,PRE,AMEM)
        CALL RTOI(DSIG,IOBUFF,LENFRM)
        CALL PUTPT1(IOBUFF,LENFRM)


        CALL PRTRBF(' SYNTHESIZER OUTPUT:',SIG,LENFRM,IDBG)
        CALL PRTRBF(' DE-EMPHASIZED OUTPUT SIGNAL:',DSIG,LENFRM,IDBG)
        CALL ISHLST(ISIG,NPOLES,LENFRM)


C--------------------

        GO TO 101


C FLUSH THE INTERNAL BUFFER IN PUTPT1

920     CALL PUTPT1(IOBUFF,0)
        IF(ISRC.GE.CMF) GO TO 50


990     STOP


C DEBUG FORMAT STATEMENTS

1000    FORMAT(' NPOLES =',I5,' LENFRM = ',I5,' LENWIN =',I5,
     *         ' IDECIM =',I5/' NCFLP  = ',I5,' NCFSB  =',I5,
     *         ' NCFHP  =',I5,' PRE =',F7.4)
1100    FORMAT('0>>>FRAME',I4,':'/
     *         ' REFLECTION COEFFICIENTS:'10I10:/
     *        (28X,10I10))

        END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       ACORR (X, NPTS, RXX, NTERM)
C
C
C PURPOSE:
C       THIS SUBROUTINE CALCULATES THE AUTO-CORRELATION FOR
C       A DATA VECTOR.
C
C
C DESCRIPTION:
C       THIS SUBROUTINE CALCULATES THE AUTO-CORRELATION FOR A
C       GIVEN DATA VECTOR. RXX(I) GIVES THE AUTO-CORRELATION AT
C       LAG I-1 FOR I RUNNING FROM 1 TO NTERM.
C
C                  NPTS
C         RXX(I) = SUM X(J) * X(J-I+1)
C                  J=I
C
C
C PARAMETERS:
C       X       - INPUT DATA VECTOR WITH NPTS ELEMENTS
C       NPTS    - NUMBER OF DATA POINTS
C     (*) RXX   - AUTO-CORRELATION VECTOR WITH NTERM ELEMENTS
C       NTERM   - NUMBER OF AUTO-CORRELATION TERMS
C
C
C ROUTINES REQUIRED:
C       NONE
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/02/25
C
C
C UPDATES:
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------

      SUBROUTINE ACORR(X,NPTS,RXX,NTERM)

      REAL X(NPTS),RXX(NTERM)

      DO 160 I=1,NTERM
        SUM=0.0
        IF(I.GT.NPTS) GO TO 140
        DO 120 J=I,NPTS
          SUM=SUM+X(J)*X(J-I+1)
120     CONTINUE
140     RXX(I)=SUM
160   CONTINUE

      RETURN
      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       APCMQ (XIN, XOUT, NLEV, QMLT, XQ, YQ, FSV, FSVMN, FSVMX, FSVTH)
C
C
C PURPOSE:
C       THIS ROUTINE QUANTIZES A SAMPLE USING AN ADAPIVE QUANTIZER.
C
C
C DESCRIPTION:
C       THE INPUT SAMPLE IS QUANTIZED, USING THE GIVEN SCALING FACTOR
C       FOR THE QUANTIZER. THE SCALING FACTOR IS UPDATED ON RETURN.
C
C
C PARAMETERS:
C       XIN     - INPUT SAMPLE
C   (*) XOUT    - OUTPUT QUANTIZED SAMPLE
C       NLEV    - NUMBER OF POSITIVE QUANTIZER LEVELS. THE QUANTIZER
C                 IS ASSUMED TO BE SYMMETRIC ABOUT ZERO. THE TOTAL
C                 NUMBER OF QUANTIZER LEVELS IS 2*NLEV.
C       QMLT    - ARRAY OF NLEV, QUANTIZER MULTIPLIERS
C       XQ      - ARRAY OF NLEV-1 NORMALIZED QUANTIZER BREAK POINTS
C                 (IN INCREASING ORDER)
C       YQ      - ARRAY OF NLEV NORMALIZED QUANTIZER OUTPUT VALUES
C                 (IN INCREASING ORDER)
C   (*) FSV     - SCALING FACTOR FOR THE QUANTIZER. THIS VALUE IS
C                 UPDATED ON OUTPUT.
C       FSVMN   - MINIMUM SCALING FACTOR
C       FSVMX   - MAXIMUM SCALING FACTOR
C       FSVTH   - THRESHOLD VALUE. IF THE SCALE FACTOR IS LESS THAN
C                 FSVTH, A MID-TREAD QUANTIZER IS USED INSTEAD OF
C                 A MID-RISE QUANTIZER. THE MID-TREAD QUANTIZER USES
C                 ZERO AS AN OUTPUT LEVEL INSTEAD OF YQ(1).
C
C
C ROUTINES REQUIRED:
C       QUANTZ - QUANTIZE A POINT
C
C
C AUTHOR / MAINTAINED BY:
C       P. Kabal
C
C
C DATE CREATED:
C       80/07/23
C
C
C UPDATES:
C       80/08/08
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------

      SUBROUTINE APCMQ(XIN,XOUT,NLEV,QMLT,XQ,YQ,
     *                 FSV,FSVMN,FSVMX,FSVTH)

      REAL QMLT(NLEV),XQ(NLEV-1),YQ(NLEV)

C QUANTIZE THE SAMPLE

      XA=ABS(XIN/FSV)
      CALL QUANTZ(XA,L,XQ,NLEV)
      XOUT=FSV*YQ(L)
      IF(FSV.LT.FSVTH .AND. L.EQ.1) XOUT=0.0
      IF(XIN.LT.0.0) XOUT=-XOUT

C UPDATE THE STEP SIZE

      FSV=FSV*QMLT(L)
      IF(FSV.GT.FSVMX) FSV=FSVMX
      IF(FSV.LT.FSVMN) FSV=FSVMN

      RETURN

      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       AUTOK (RXX, NPOLE, REFL)
C
C
C PURPOSE:
C       THIS ROUTINE SOLVES A SET OF TOEPLITZ EQUATIONS TO OBTAIN
C       A SET OF REFLECTION COEFFICIENTS.
C
C
C DESCRIPTION:
C       THIS SUBROUTINE IMPLEMENTS A MODIFIED FORM OF DURBIN'S
C       ALGORITHM FOR SOLVING A SET OF AUTO-CORRELATION EQUATIONS.
C       THIS ROUTINE USES AN INTERMEDIATE VARIABLE WHICH IS
C       CONSTRAINED TO BE LESS THAN THE ENERGY OF THE SIGNAL
C       I.E. RXX(1). THIS VARIABLE TAKES THE ROLE OF THE PREDICTOR
C       COEFFICIENTS WHICH NORMALLY ARE USED IN DURBIN'S FORM
C       OF THE ALGORITHM. THIS ROUTINE RETURNS ONLY THE
C       REFLECTION COEFFICIENTS.
C         REFERENCE: J. LEROUX AND C.J. GUEGUEN, "A FIXED POINT
C                    COMPUTATION OF THE PARTIAL CORRELATION
C                    COEFFICIENTS", IEEE TRANS. ASSP, JUNE 1977,
C                    PP. 257-259.
C
C
C PARAMETERS:
C       RXX    - ARRAY OF NPOLE+1 AUTO-CORRELATION COEFFICIENTS
C       NPOLE  - NUMBER OF REFLECTION COEFFICIENTS (MAXIMUM 13)
C   (*) REFL   - OUTPUT ARRAY OF NPOLE REFLECTION COEFFICIENTS
C
C
C ROUTINES REQUIRED:
C       NONE
C
C
C AUTHOR / MAINTAINED BY:
C       J. BOYD
C
C
C DATE CREATED:
C       20-08-80
C
C
C UPDATES:
C       81/02/24
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------

      SUBROUTINE AUTOK(RXX,NPOLE,REFL)


      PARAMETER MXPOLE=13
      REAL RXX(NPOLE+1),REFL(NPOLE)
      REAL EP(MXPOLE),EN(MXPOLE)


C  INITIALIZATION
      DO 140 I=1,NPOLE
        EP(I)=RXX(I+1)
        EN(I)=RXX(I)
140   CONTINUE
      IF(EN(1).LE.0.0) EN(1)=1.0


C  RECURSIVE SOLUTION OF THE EQUATIONS
      DO 180 K=1,NPOLE
        REFL(K)=-EP(K)/EN(1)

        IF(K.EQ.NPOLE) GO TO 900

        EN(1)=EN(1)+REFL(K)*EP(K)
        EP(NPOLE)=EP(NPOLE)+REFL(K)*EN(NPOLE-K+1)

        IF(K.GE.NPOLE-1) GO TO 180
        L=2
        DO 160 I=K+1,NPOLE-1
          SUM=EP(I)+REFL(K)*EN(L)
          EN(L)=EN(L)+REFL(K)*EP(I)
          EP(I)=SUM
          L=L+1
160     CONTINUE

180   CONTINUE

900   RETURN

      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       DEEM (X, Y, NPTS, PRE, AMEM)
C
C
C PURPOSE:
C       THIS SUBROUTINE DE-EMPHASIZES A DATA VECTOR.
C
C
C DESCRIPTION:
C       THIS SUBROUTINE IMPLEMENTS A FIRST ORDER RECURSIVE FILTER
C       TO DE-EMPHASIZE A SIGNAL VECTOR.
C
C         Y(I) = X(I) + PRE * Y(I-1))
C
C PARAMETERS:
C       X       - INPUT VECTOR WITH NPTS ELEMENTS
C   (*) Y       - OUTPUT VECTOR WITH NPTS ELEMENTS. THE VECTOR Y MAY
C                 BE THE SAME VECTOR AS X.
C       NPTS    - NUMBER OF POINTS IN EACH OF X AND Y.
C       PRE     - PRE-EMPHASIS FACTOR (NORMALLY POSITIVE).
C   (*) AMEM    - MEMORY ELEMENT. THIS LOCATION IS USED TO STORE THE
C                 THE LAST OUTPUT ELEMENT. INITIALLY, AMEM SHOULD BE
C                 SET TO ZERO. AS A LONG ARRAY OF DATA IS PROCESSED
C                 BLOCK BY BLOCK, THE VALUE OF AMEM RETURNED FROM ONE
C                 CALL TO THIS SUBROUTINE IS SUITABLE AS THE INITIAL
C                 VALUE FOR THE NEXT CALL.
C
C
C ROUTINES REQUIRED:
C       NONE
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/02/28
C
C
C UPDATES:
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------

      SUBROUTINE DEEM(X,Y,NPTS,PRE,AMEM)

      REAL X(NPTS),Y(NPTS)

      DO 100 I=1,NPTS
        AMEM=X(I)+PRE*AMEM
        Y(I)=AMEM
100   CONTINUE

      RETURN

      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       FCNVL (DATA, FCOF, NTERM, INC)
C
C
C PURPOSE:
C       THIS FUNCTION CONVOLVES A DATA VECTOR WITH A FILTER
C       COEFFICIENT VECTOR.
C
C
C DESCRIPTION:
C       THIS FUNCTION CALCULATES A SINGLE VALUE WHICH IS THE
C       RESULT OF CONVOLVING AN ARRAY OF DATA POINTS WITH AN
C       ARRAY OF FILTER COEFFICIENTS.
C
C         FCNVL  =  SUM   FCOF(I) * DATA(2-I)
C
C           WHERE I TAKES ON VALUES 1, INC+1, ... , (NTERM-1)*INC+1
C
C
C PARAMETERS:
C   (*) FCNVL   - RESULTANT FUNCTION VALUE
C
C       DATA    - ARRAY OF DATA SPECIFIED BY THE LAST ELEMENT TO BE USED
C                 IN CALCULATING THE CONVOLUTION SUM.
C       FCOF    - ARRAY OF NTERM FILTER COEFFICIENTS.
C       NTERM   - NUMBER OF TERMS IN THE CONVOLUTION SUM.
C       INC     - INDEX INCREMENT
C
C
C ROUTINES REQUIRED:
C       NONE
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/02/09
C
C
C UPDATES:
C       81/02/24
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
      FUNCTION FCNVL(DATA,FCOF,NTERM,INC)

      REAL DATA(1),FCOF(1)

      FCNVL=0.0
      K=1
      L=1
      DO 100 I=1,NTERM
        FCNVL=FCNVL+FCOF(L)*DATA(K)
        K=K-INC
        L=L+INC
  100 CONTINUE

      RETURN

      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       IACORR (IX, NPTS, IRXX, NTERM)
C
C
C PURPOSE:
C       THIS SUBROUTINE CALCULATES THE AUTO-CORRELATION FOR
C       A DATA VECTOR USING SCALED INTEGER ARITHMETIC.
C
C
C DESCRIPTION:
C       THIS SUBROUTINE CALCULATES THE AUTO-CORRELATION FOR A
C       GIVEN DATA VECTOR. IRXX(I) GIVES THE AUTO-CORRELATION AT
C       LAG I-1 FOR I RUNNING FROM 1 TO NTERM.
C
C                   NPTS
C         IRXX(I) = SUM IX(J) * IX(J-I+1)
C                   J=I
C
C
C PARAMETERS:
C       IX      - INPUT DATA VECTOR WITH NPTS ELEMENTS
C       NPTS    - NUMBER OF DATA POINTS
C    (*) IRXX   - AUTO-CORRELATION VECTOR WITH NTERM ELEMENTS
C       NTERM   - NUMBER OF AUTO-CORRELATION TERMS (MAXIMUM 13)
C
C
C ROUTINES REQUIRED:
C       MULD    - MULTIPLY TWO SCALED INTEGER VALUES
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/03/06
C
C
C UPDATES:
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
```

```
      SUBROUTINE IACORR(IX,NPTS,IRXX,NTERM)


      PARAMETER MXTERM=13

      INTEGER*2 IX(NPTS),IRXX(NTERM),IS(2)
      INTEGER*4 IRXXD(MXTERM),ISUMD,ID,ISCD,MULD
      EQUIVALENCE (ID,IS)


      DO 160 I=1,NTERM
        ISUMD=0
        IF(I.GT.NPTS) GO TO 140
        DO 120 J=I,NPTS
          ISUMD=ISUMD+MULD(IX(J),IX(J-I+1))
120     CONTINUE
140     IRXXD(I)=ISUMD
160   CONTINUE


C  COUNT THE NUMBER OF LEADING ZEROS IN IRXXD(1) UP TO
C  A MAXIMUM OF 16

      ISCD=1
      ID=IRXXD(1)
      DO 180 I=1,30
        IF(ID.LE.0) GO TO 200
        ID=2*ID
        ISCD=2*ISCD
180   CONTINUE
200   IF(ISCD.NE.1) ISCD=ISCD/2

C  SHIFT THE MOST SIGNIFICANT 16 BITS TO THE TOP
C  STORE THESE IN IRXX(I)
      DO 220 I=1,NTERM
        ID=ISCD*IRXXD(I)
        IRXX(I)=IS(2)
220   CONTINUE

      RETURN

      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       IAUTOK (IRXX, NPOLE, IREFL)
C
C
C PURPOSE:
C       THIS ROUTINE SOLVES A SET OF TOEPLITZ EQUATIONS TO OBTAIN
C       A SET OF IREFLECTION COEFFICIENTS USING SCALED INTEGER
C       ARITHMETIC.
C
C
C DESCRIPTION:
C       THIS SUBROUTINE IMPLEMENTS A MODIFIED FORM OF DURBIN'S
C       ALGORITHM FOR SOLVING A SET OF AUTO-CORRELATION EQUATIONS.
C       THIS ROUTINE USES AN INTERMEDIATE VARIABLE WHICH IS
C       CONSTRAINED TO BE LESS THAN THE ENERGY OF THE SIGNAL
C       I.E. IRXX(1). THIS VARIABLE TAKES THE ROLE OF THE PREDICTOR
C       COEFFICIENTS WHICH NORMALLY ARE USED IN DURBIN'S FORM
C       OF THE ALGORITHM. THIS ROUTINE RETURNS ONLY THE
C       IREFLECTION COEFFICIENTS.
C         REFERENCE: J. LEROUX AND C.J. GUEGUEN, "A FIXED POINT
C                    COMPUTATION OF THE PARTIAL CORRELATION
C                    COEFFICIENTS", IEEE TRANS. ASSP, JUNE 1977,
C                    PP. 257-259.
C
C
C PARAMETERS:
C       IRXX    - ARRAY OF NPOLE+1 AUTO-CORRELATION COEFFICIENTS
C       NPOLE   - NUMBER OF IREFLECTION COEFFICIENTS (MAXIMUM 13)
C   (*) IREFL   - OUTPUT ARRAY OF NPOLE IREFLECTION COEFFICIENTS
C
C
C ROUTINES REQUIRED:
C       IDIVS   - DIVIDE USING SCALED INTEGER ARITHMETIC
C       MULS    - MULTIPLY TWO SCALED INTEGER VALUES
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/03/06
C
C
C UPDATES:
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------

      SUBROUTINE IAUTOK(IRXX,NPOLE,IREFL)


      PARAMETER MXPOLE=13
      INTEGER*2 IRXX(NPOLE+1),IREFL(NPOLE),ISUM,MULS,IDIVS
      INTEGER*2 IEP(MXPOLE),IEN(MXPOLE)


C  INITIALIZATION
      DO 140 I=1,NPOLE
        IEP(I)=IRXX(I+1)
        IEN(I)=IRXX(I)
140   CONTINUE
      IF(IEN(1).LE.0) IEN(1)=32767


C  RECURSIVE SOLUTION OF THE EQUATIONS
      DO 180 K=1,NPOLE
        IREFL(K)=IDIVS(-IEP(K),IEN(1))

        IF(K.EQ.NPOLE) GO TO 900

        IEN(1)=IEN(1)+MULS(IREFL(K),IEP(K))
        IEP(NPOLE)=IEP(NPOLE)+MULS(IREFL(K),IEN(NPOLE-K+1))

        IF(K.GE.NPOLE-1) GO TO 180
        L=2
        DO 160 I=K+1,NPOLE-1
          ISUM=IEP(I)+MULS(IREFL(K),IEN(L))
          IEN(L)=IEN(L)+MULS(IREFL(K),IEP(I))
          IEP(I)=ISUM
          L=L+1
160     CONTINUE

180   CONTINUE

900   RETURN

      END
```

```
;--------------Bell-Northern Research / INRS Computer Laboratory-------------
;
;
; MODULE:
;       IDIVS (I, J)
;
;
; PURPOSE:
;       THIS FUNCTION PERFORMS INTEGER DIVISION.
;
;
; DESCRIPTION:
;       THIS FUNCTION DIVIDES TWO INTEGER VALUES. THE NUMERATOR
;       IS PLACED IN THE HIGH ORDER PART OF A DOUBLE PRECISION
;       REGISTER AND IS THEN DIVIDED BY THE DENOMINATOR VALUE.
;       THIS OPERATION IS APPROPRIATE IF THE NUMERATOR AND
;       DENOMINATOR ARE CONSIDERED TO BE INTEGER VALUES SCALED
;       BY 2**15. IN THIS CASE THEY REPRESENT VALUES WITH MAGNITUDE
;       LESS THAN ONE. THE OUTPUT VALUE IS SCALED LIKEWISE.
;
;
; PARAMETERS:
;   (*) IDIVS  - RETURNED INTEGER VALUE
;
;       I      - INPUT INTEGER VALUE
;       J      - INPUT INTEGER VALUE
;
;
; ROUTINES REQUIRED:
;       NONE
;
;
; AUTHOR / MAINTAINED BY:
;       P. KABAL
;
;
; DATE CREATED:
;       81/03/09
;
;
; UPDATES:
;
;
;--------------Bell-Northern Research / INRS Computer Laboratory-------------

        .TITLE IDIVS
        .MCALL RETURN
        .PSECT IARITH

IDIVS::
        MOV @2(R5),R0
        CLR R1                  ;(R0,R1)=I
                                ;  HIGH ORDER BITS IN R0

        ASR R0
        ROR R1                  ;(R0,R1)=(R0,R1)/2

        DIV @4(R5),R0           ;R0=(R0,R1)/J
                                ;  RETURN THE QUOTIENT IN R0

        RETURN

        .END
```

```
;--------------Bell-Northern Research / INRS Computer Laboratory-------------
;
;
; MODULE:
;       IEXTRT (ID, NS)
;
;
; PURPOSE:
;       THIS FUNCTION RETURNS A SELECTED GROUP OF 16 BITS FROM
;       A DOUBLE PRECISION INTEGER.
;
;
; DESCRIPTION:
;       THIS FUNCTION SHIFTS THE BITS IN A 32 BIT INTEGER AND
;       RETURNS THE 16 MOST SIGNIFICANT BITS OF THE 32 BIT INTEGER.
;
;
; PARAMETERS
;   (*) IEXTRT - RETURNED 16 BIT VALUE
;
;       ID      - INPUT 32 BIT VALUE
;       NS      - NUMBER OF PLACES TO BE SHIFTED. NORMALLY NS IS
;                 IN THE RANGE 0 TO 16. VALUES FROM -32 TO +31 ARE
;                 PERMISSIBLE.
;                  0 - IEXTRT IS BITS 16-31 (TOP 16 BITS)
;                  1 - IEXTRT IS BITS 15-30
;                  ...
;                 16 - IEXTRT IS BITS  0-15 (BOTTOM 16 BITS)
;
;
; ROUTINES REQUIRED:
;       NONE
;
;
; AUTHOR / MAINTAINED BY:
;       P. KABAL
;
;
; DATE CREATED:
;       81/03/09
;
;
; UPDATES:
;
;
;--------------Bell-Northern Research / INRS Computer Laboratory-------------

        .TITLE IEXTRT
        .MCALL RETURN
        .PSECT IARITH

IEXTRT::
        MOV 2(R5),R2
        MOV (R2),R1
        MOV 2(R2),R0            ;(R0,R1)=ID
                                ;  HIGH ORDER BITS IN R0, LOW ORDER BITS IN R1

        ASHC @4(R5),R0          ;SHIFT (R0,R1) BY NS PLACES
                                ;  RETURN THE HIGH ORDER BITS IN R0

        RETURN

        .END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       IFCNVL (IDATA, IFCOF, NTERM, INC)
C
C
C PURPOSE:
C       THIS FUNCTION CONVOLVES A DATA VECTOR WITH A FILTER
C       COEFFICIENT VECTOR USING INTEGER ARITHMETIC.
C
C
C DESCRIPTION:
C       THIS FUNCTION CALCULATES A SINGLE VALUE WHICH IS THE
C       RESULT OF CONVOLVING AN ARRAY OF DATA POINTS WITH AN
C       ARRAY OF FILTER COEFFICIENTS. THE RETURNED VALUE IS
C       DOUBLE PRECISION.

C         IFCNVL  =  SUM   IFCOF(I) * IDATA(2-I)
C
C           WHERE I TAKES ON VALUES 1, INC+1, ... , (NTERM-1)*INC+1
C


C PARAMETERS:
C   (*) IFCNVL - RESULTANT FUNCTION VALUE, 32 BIT VALUE
C
C       IDATA  - ARRAY OF DATA SPECIFIED BY THE LAST ELEMENT TO BE USED
C                IN CALCULATING THE CONVOLUTION SUM.
C       IFCOF  - ARRAY OF NTERM FILTER COEFFICIENTS.
C       NTERM    NUMBER OF TERMS IN THE CONVOLUTION SUM.
C       INC      INDEX INCREMENT
C
C
C ROUTINES REQUIRED:
C       MULD  - MULTIPLY SCALED INTEGER VALUES
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/03/02
C
C
C UPDATES:
C       81/03/09
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------

      FUNCTION IFCNVL(IDATA,IFCOF,NTERM,INC)


      INTEGER*4 IFCNVL,MULD
      INTEGER*2 IDATA(1),IFCOF(1)


      IFCNVL=0
      K=1
      L=1
      DO 100 I=1,NTERM
        IFCNVL=IFCNVL+MULD(IFCOF(L),IDATA(K))
        K=K-INC
        L=L+INC
  100 CONTINUE

      RETURN

      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory----------
C
C
C MODULE:
C       IKTOA (IREFL, IERRCF, NPOLE)
C
C
C PURPOSE:
C       THIS ROUTINE CONVERTS A SET OF REFLECTION COEFFICIENTS TO
C       THE EQUIVALENT SET OF INVERSE FILTER COEFFICIENTS USING
C       SCALED INTEGER ARITHMETIC.
C
C
C DESCRIPTION:
C       THIS ROUTINE GENERATES THE VECTOR OF INVERSE FILTER COEFFICIENTS
C       CORRESPONDING TO THE GIVEN SET OF REFLECTION COEFFICIENTS.
C       THESE ARE RELATED TO THE PREDICTOR COEFFICIENTS AS FOLLOWS.
C         IERRCF(1)       =   8192
C         IERRCF(2)       =  -IPRDCF(1)
C           ...                ...
C         IERRCF(NPOLE+1) =  -IPRDCF(NPOLE)
C
C
C PARAMETERS:
C       IREFL  - ARRAY OF NPOLE REFLECTION COEFFICIENTS
C     (*) IERRCF - ARRAY OF NPOLE+1 INVERSE FILTER COEFFICIENTS
C       NPOLE  - NUMBER OF REFLECTION COEFFICIENTS
C
C
C ROUTINES REQUIRED:
C       MULS   - MULTIPLY TWO SCALED INTEGER VARIABLES
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/03/06
C
C
C UPDATES:
C
C
C---------------Bell-Northern Research / INRS Compute· Laboratory----------
```

```
      SUBROUTINE IKTOA(IREFL,IERRCF,NPOLE)

      INTEGER*2 IERRCF(NPOLE+1),IREFL(NPOLE),ISUM,MULS


      IERRCF(1)=8192
      IERRCF(2)=IREFL(1)/4
      IF(NPOLE.LE.1) GO TO 900

      DO 200 J=2,NPOLE
        IERRCF(J+1)=IREFL(J)/4
        L=J-1
        DO 100 I=1,J/2
          ISUM=IERRCF(I+1)+MULS(IREFL(J),IERRCF(L+1))
          IF(I.NE.L) IERRCF(L+1)=IERRCF(L+1)
     *                           +MULS(IREFL(J),IERRCF(I+1))
          IERRCF(I+1)=ISUM
          L=L-1
  100   CONTINUE
  200 CONTINUE

  900 RETURN

      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       IKTOP (IREFL, IPRDCF, NPOLE)
C
C
C PURPOSE:
C       THIS ROUTINE CONVERTS A SET OF REFLECTION COEFFICIENTS TO
C       THE EQUIVALENT SET OF PREDICTOR COEFFICIENTS USING SCALED
C       INTEGER ARITHMETIC.
C
C
C DESCRIPTION:
C       THIS ROUTINE GENERATES THE VECTOR OF PREDICTOR COEFFICIENTS
C       CORRESPONDING TO THE GIVEN SET OF REFLECTION COEFFICIENTS.
C
C
C PARAMETERS:
C       IREFL  - ARRAY OF NPOLE REFLECTION COEFFICIENTS
C   (*) IPRDCF - ARRAY OF NPOLE PREDICTOR COEFFICIENTS
C       NPOLE  - NUMBER OF COEFFICIENTS
C
C
C ROUTINES REQUIRED:
C       MULS   - MULTIPLY TWO SCALED INTEGER VARIABLES
C

C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/03/04
C
C
C UPDATES:
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
```

```
      SUBROUTINE IKTOP(IREFL,IPRDCF,NPOLE)


      INTEGER*2 IPRDCF(NPOLE),IREFL(NPOLE),ISUM,MULS


      IPRDCF(1)=-IREFL(1)/4
      IF(NPOLE.LE.1) GO TO 900

      DO 200 J=2,NPOLE
        IPRDCF(J)=-IREFL(J)/4
        L=J-1
        DO 100 I=1,J/2
          ISUM=IPRDCF(I)+MULS(IREFL(J),IPRDCF(L))
          IF(I.NE.L) IPRDCF(L)=IPRDCF(L)+MULS(IREFL(J),IPRDCF(I))
          IPRDCF(I)=ISUM
          L=L-1
  100   CONTINUE
  200 CONTINUE

  900 RETURN

      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       IQUANT (IX, L, IXQ, NLEV)
C
C
C PURPOSE:
C       THIS ROUTINE QUANTIZES AN INPUT VALUE.
C
C
C DESCRIPTION:
C       THIS SUBROUTINE RETURNS THE INDEX OF THE QUANTIZER OUTPUT REGION
C       CORRESPONDING TO A GIVEN INPUT VALUE. THE QUANTIZER IS SPECIFIED
C       BY AN ARRAY OF QUANTIZER BREAK POINTS.
C
C
C PARAMETERS:
C       IX      - QUANTIZER IN. T VALUE
C    (*) L      - OUTPUT INDEX OF THE QUANTIZER OUTPUT REGION CORRESPONDING
C                 TO IX. L TAKES ON VALUES FROM 1 TO NLEV.
C                    L = 1,       IX =< IXQ(1)
C                    L = 2,       IXQ(1) < IX =< IXQ(2)
C                     ...           ...
C                    L = NLEV,    IXQ(NLEV-1) < IX
C       IXQ     - INPUT ARRAY OF NLEV-1 QUANTIZER BREAK POINTS WHICH DELIMIT
C                 THE NLEV QUANTIZER REGIONS. THESE MUST BE IN ASCENDING ORDER.
C       NLEV    - NUMBER OF QUANTIZER REGIONS (MINIMUM 2)
C
C
C ROUTINES REQUIRED:
C       NONE
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/03/06
C
C
C UPDATES:
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
      SUBROUTINE IQUANT(IX,L,IXQ,NLEV)
C
C
      INTEGER IXQ(NLEV-1)
C
C
C  BINARY SEARCH FOR THE INTERVAL ( IXQ(IL),IXQ(IU) ) WHICH BRACKETS
C  THE VALUE IX.
      IL=0
      IU=NLEV
C
100   I=(IL+IU)/2
      IF(IX.GT.IXQ(I)) GO TO 200
        IU=I
        GO TO 300
C
200     IL=I
C
C  SEE IF THE INTERVAL HAS BEEN ISOLATED
300   IF(IU.GT.IL+1) GO TO 100
C
      L=IU
C
      RETURN
C
      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       ISHIFT (IX, NKEEP, NSHIFT)
C
C
C PURPOSE:
C       THIS ROUTINE SHIFTS THE ELEMENTS OF AN INTEGER VECTOR.
C
C
C DESCRIPTION:
C       THIS ROUTINE SHIFTS THE ELEMENTS OF AN INTEGER VECTOR.
C       IF THE NUMBER OF ELEMENTS TO BE RETAINED IS ZERO, NO
C       ACTION IS TAKEN.
C
C       FOR A SHIFT DOWN (NSHIFT POSITIVE), THE ELEMENTS OF
C       THE ARRAY ARE SHIFTED DOWN TO THE BOTTOM OF THE
C       ARRAY. THE INPUT ARRAY MUST HAVE AT LEAST
C       NKEEP+NSHIFT ELEMENTS.
C         IX(NSHIFT+I) --> IX(I) , FOR I FROM 1 TO NKEEP.
C
C       FOR A SHIFT UP (NSHIFT NEGATIVE), THE ELEMENTS OF
C       THE ARRAY ARE SHIFTED UP TO THE TOP OF THE ARRAY.
C       THE INPUT ARRAY MUST HAVE AT LEAST NKEEP-NSHIFT
C       ELEMENTS (RECALL THAT NSHIFT IS NEGATIVE).
C         IX(I) --> IX(I-NSHIFT), FOR I FROM NKEEP TO 1.
C
C
C PARAMETERS:
C       IX     - INPUT ARRAY
C       NKEEP  - NUMBER OF ELEMENTS TO BE RETAINED
C       NSHIFT - NUMBER OF POSITIONS TO BE SHIFTED. NSHIFT
C                IS POSITIVE FOR A SHIFT DOWN AND NEGATIVE
C                FOR A SHIFT UP. NSHIFT EQUAL TO ZERO IS
C                ALSO PERMISSIBLE.
C
C
C ROUTINES REQUIRED:
C       NONE
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/02/15
C
C
C UPDATES:
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
```

```
      SUBROUTINE ISHIFT(IX,NKEEP,NSHIFT)
C
C
      INTEGER IX(1)
C
C
      IF(NKEEP.LE.0) GO TO 900
C
C SHIFT DOWN
      IF(NSHIFT.LT.0) GO TO 200
      DO 100 I=1,NKEEP
        IX(I)=IX(NSHIFT+I)
100   CONTINUE
      GO TO 900
C
C SHIFT UP
200   DO 300 I=NKEEP,1,-1
        IX(I-NSHIFT)=IX(I)
300   CONTINUE
C
900   RETURN
C
      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       KTOA (REFL, ERRCOF, NPOLE)
C
C
C PURPOSE:
C       THIS ROUTINE CONVERTS A SET OF REFLECTION COEFFICIENTS TO
C       THE EQUIVALENT SET OF INVERSE FILTER COEFFICIENTS.
C
C
C DESCRIPTION:
C       THIS ROUTINE GENERATES THE VECTOR OF INVERSE FILTER COEFFICIENTS
C       CORRESPONDING TO THE GIVEN SET OF REFLECTION COEFFICIENTS.
C       THESE ARE RELATED TO THE PREDICTOR COEFFICIENTS AS FOLLOWS.
C         ERRCOF(1)       =   1
C         ERRCOF(2)       =  -PRDCOF(1)
C           ...                 ...
C         ERRCOF(NPOLE+1) =  -PRDCOF(NPOLE)
C
C
C PARAMETERS:
C       REFL   - ARRAY OF NPOLE REFLECTION COEFFICIENTS
C   (*) ERRCOF - ARRAY OF NPOLE+1 INVERSE FILTER COEFFICIENTS
C       NPOLE  - NUMBER OF REFLECTION COEFFICIENTS
C
C
C ROUTINES REQUIRED:
C       NONE
C
C
C AUTHOR / MAINTAINED BY:
C       J. TURNER
C
C
C DATE CREATED:
C       15-APR-80
C
C
C UPDATES:
C       81/02/27  P. KABAL
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------

      SUBROUTINE KTOA(REFL,ERRCOF,NPOLE)

      REAL ERRCOF(NPOLE+1),REFL(NPOLE)

      ERRCOF(1)=1.
      ERRCOF(2)=REFL(1)
      IF(NPOLE.LE.1) GO TO 900

      DO 200 J=2,NPOLE
        ERRCOF(J+1)=REFL(J)
        L=J-1
        DO 100 I=1,J/2
          SUM=ERRCOF(I+1)+REFL(J)*ERRCOF(L+1)
          IF(I.NE.L) ERRCOF(L+1)=ERRCOF(L+1)+REFL(J)*ERRCOF(I+1)
          ERRCOF(I+1)=SUM
          L=L-1
  100   CONTINUE
  200 CONTINUE

  900 RETURN

      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       KTOP (REFL, PRDCOF, NPOLE)
C
C
C PURPOSE:
C       THIS ROUTINE CONVERTS A SET OF REFLECTION COEFFICIENTS TO
C       THE EQUIVALENT SET OF PREDICTOR COEFFICIENTS.
C
C
C DESCRIPTION:
C       THIS ROUTINE GENERATES THE VECTOR OF PREDICTOR COEFFICIENTS
C       CORRESPONDING TO THE GIVEN SET OF REFLECTION COEFFICIENTS.
C
C
C PARAMETERS:
C       REFL   - ARRAY OF NPOLE REFLECTION COEFFICIENTS
C   (*) PRDCOF - ARRAY OF NPOLE PREDICTOR COEFFICIENTS
C       NPOLE  - NUMBER OF COEFFICIENTS
C
C
C ROUTINES REQUIRED:
C       NONE
C
C
C AUTHOR / MAINTAINED BY:
C       J. TURNER
C
C
C DATE CREATED:
C       15-APR-80
C
C
C UPDATES:
C       81/02/27  P. KABAL
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-----------
```

```
      SUBROUTINE KTOP(REFL,PRDCOF,NPOLE)


      REAL PRDCOF(NPOLE),REFL(NPOLE)


      PRDCOF(1)=-REFL(1)
      IF(NPOLE.LE.1) GO TO 900

      DO 200 J=2,NPOLE
        PRDCOF(J)=-REFL(J)
        L=J-1
        DO 100 I=1,J/2
          SUM=PRDCOF(I)+REFL(J)*PRDCOF(L)
          IF(I.NE.L) PRDCOF(L)=PRDCOF(L)+REFL(J)*PRDCOF(I)
          PRDCOF(I)=SUM
          L=L-1
  100   CONTINUE
  200 CONTINUE

  900 RETURN

      END
```

```
;--------------Bell-Northern Research / INRS Computer Laboratory-------------
;
;
; MODULE:
;       MULD (I, J)
;
;
; PURPOSE:
;       THIS FUNCTION PERFORMS INTEGER MULTIPLICATION, RETURNING
;       A DOUBLE PRECISION RESULT.
;
;
; DESCRIPTION:
;       THIS FUNCTION CALCULATES THE PRODUCT OF TWO INTEGER
;       VALUES.
;
;
; PARAMETERS:
;   (*) MULD    - RETURNED DOUBLE PRECISION SCALED INTEGER VALUE
;
;       I       - INPUT SCALED INTEGER VALUE
;       J       - INPUT SCALED INTEGER VALUE
;
;
; ROUTINES REQUIRED:
;       NONE
;
;
; AUTHOR / MAINTAINED BY:
;       P. KABAL
;
;
; DATE CREATED:
;       81/03/02
;
;
; UPDATES:
;       81/03/09
;
;
;--------------Bell-Northern Research / INRS Computer Laboratory-------------
```

```
        .TITLE MULD
        .MCALL RETURN
        .PSECT IARITH

MULD::
        MOV @2(R5),R2
        MUL @4(R5),R2           ;(R2,R3)=I*J
                                ;  HIGH ORDER BITS IN R2, LOW ORDER BITS IN R1
        MOV R2,R1
        MOV R3,R0               ;(R1,R0)=(R2,R3)
                                ;  HIGH ORDER BITS IN R1, LOW ORDER BITS IN R0
        RETURN

        .END
```

```
;--------------Bell-Northern Research / INRS Computer Laboratory------------
;
;
; MODULE:
;       MULS (I, J)
;
;
; PURPOSE:
;       THIS FUNCTION PERFORMS SCALED INTEGER MULTIPLICATION.
;
;
; DESCRIPTION:
;       THIS FUNCTION CALCULATES THE PRODUCT OF TWO INTEGER
;       VALUES. THE HIGH ORDER PART OF THE DOUBLE PRECISION
;       PRODUCT IS RETURNED. THIS OPERATION IS APPROPRIATE IF
;       THE OPERANDS ARE CONSIDERED TO BE INTEGER VALUES SCALED
;       BY 2**15. IN THIS CASE THEY REPRESENT VALUES WITH MAGNITUDE
;       LESS THAN ONE. THE 16 BIT PRODUCT IS SCALED LIKEWISE.
;
;
; PARAMETERS:
;    *) MULS    - RETURNED SCALED INTEGER VALUE
;
;       I       - INPUT SCALED INTEGER VALUE
;       J       - INPUT SCALED INTEGER VALUE
;
;
; ROUTINES REQUIRED:
;       NONE
;
;
; AUTHOR / MAINTAINED BY:
;       P. KABAL
;
;
; DATE CREATED:
;       81/03/02
;
;
; UPDATES:
;
;
;--------------Bell-Northern Research / INRS Computer Laboratory------------
```

```
.TITLE MULS
.MCALL RETURN
.PSECT IARITH

MULS::
        MOV @2(R5),R0
        MUL @4(R5),R0           ;(R0,R1)=I*J
                                ;  HIGH ORDER BITS IN R0, LOW ORDER BITS IN R1
        ASL R1
        ROL R0                  ;(R0,R1)=2*(R0,R1)
                                ;  SHIFT THE BITS BY ONE POSITION
                                ;  ONLY R0 IS RETURNED BY THIS FUNCTION
        RETURN

        .END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       PREEM (X, Y, NPTS, PRE, AMEM)
C
C
C PURPOSE:
C       THIS SUBROUTINE PRE-EMPHASIZES A DATA VECTOR.
C
C
C DESCRIPTION:
C       THIS SUBROUTINE FORMS THE FIRST DIFFERENCE WITH THE PAST
C       SAMPLE WEIGHTED BY PRE OF A VECTOR TO FORM A NEW VECTOR.
C
C         Y(I) = X(I) - PRE * X(I-1)
C
C PARAMETERS:
C       X       - INPUT VECTOR WITH NPTS ELEMENTS
C   (*) Y       - OUTPUT VECTOR WITH NPTS ELEMENTS. THE VECTOR Y MAY
C                 BE THE SAME VECTOR AS X.
C       NPTS    - NUMBER OF POINTS IN EACH OF X AND Y.
C       PRE     - PRE-EMPHASIS FACTOR (NORMALLY POSITIVE).
C   (*) AMEM    - MEMORY ELEMENT. THIS LOCATION IS USED TO STORE THE
C                 THE LAST INPUT ELEMENT. INITIALLY, AMEM SHOULD BE
C                 SET TO ZERO. AS A LONG ARRAY OF DATA IS PROCESSED
C                 BLOCK BY BLOCK, THE VALUE OF AMEM RETURNED FROM ONE
C                 CALL TO THIS SUBROUTINE IS SUITABLE AS THE INITIAL
C                 VALUE FOR THE NEXT CALL.
C
C
C ROUTINES REQUIRED:
C       NONE
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       81/02/25
C
C
C UPDATES:
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
```

```
      SUBROUTINE PREEM(X,Y,NPTS,PRE,AMEM)

      REAL X(NPTS),Y(NPTS)

      DO 100 I=1,NPTS
        TEMP=X(I)
        Y(I)=X(I)-PRE*AMEM
        AMEM=TEMP
100   CONTINUE

      RETURN

      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       QUANTZ (X, L, XQ, NLEV)
C
C
C PURPOSE:
C       THIS ROUTINE QUANTIZES AN INPUT VALUE.
C
C
C DESCRIPTION:
C       THIS SUBROUTINE RETURNS THE INDEX OF THE QUANTIZER OUTPUT REGION
C       CORRESPONDING TO A GIVEN INPUT VALUE. THE QUANTIZER IS SPECIFIED
C       BY AN ARRAY OF QUANTIZER BREAK POINTS.
C
C
C PARAMETERS:
C       X       - QUANTIZER INPUT VALUE
C   (*) L       - OUTPUT INDEX OF THE QUANTIZER OUTPUT REGION CORRESPONDING
C                 TO X. L TAKES ON VALUES FROM 1 TO NLEV.
C                    L = 1,       X =< XQ(1)
C                    L = 2,       XQ(1) < X =< XQ(2)
C                     ...           ...
C                    L = NLEV,    XQ(NLEV-1) < X
C       XQ      - INPUT ARRAY OF NLEV-1 QUANTIZER BREAK POINTS WHICH DELIMIT
C                 THE NLEV QUANTIZER REGIONS. THESE MUST BE IN ASCENDING ORDER.
C       NLEV    - NUMBER OF QUANTIZER REGIONS (MINIMUM 2)
C
C
C ROUTINES REQUIRED:
C       NONE
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       79/05/24
C
C
C UPDATES:
C       80/06/17
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
      SUBROUTINE QUANTZ(X,L,XQ,NLEV)
C
C
      REAL XQ(NLEV-1)
C
C
C  BINARY SEARCH FOR THE INTERVAL ( XQ(IL),XQ(IU) ) WHICH BRACKETS
C  THE VALUE X.
      IL=0
      IU=NLEV
C
100   I=(IL+IU)/2
      IF(X.GT.XQ(I)) GO TO 200
        IU=I
        GO TO 300
C
200     IL=I
C
C  SEE IF THE INTERVAL HAS BEEN ISOLATED
300   IF(IU.GT.IL+1) GO TO 100
C
      L=IU
C
      RETURN
C
      END
```

```
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
C
C
C MODULE:
C       RSHIFT (X, NKEEP, NSHIFT)
C
C
C PURPOSE:
C       THIS ROUTINE SHIFTS THE ELEMENTS OF A REAL VECTOR.
C
C
C DESCRIPTION:
C       THIS ROUTINE SHIFTS THE ELEMENTS OF A REAL VECTOR.
C       IF THE NUMBER OF ELEMENTS TO BE RETAINED IS ZERO, NO
C       ACTION IS TAKEN.
C
C       FOR A SHIFT DOWN (NSHIFT POSITIVE), THE ELEMENTS OF
C       THE ARRAY ARE SHIFTED DOWN TO THE BOTTOM OF THE
C       ARRAY. THE INPUT ARRAY MUST HAVE AT LEAST
C       NKEEP+NSHIFT ELEMENTS.
C         X(NSHIFT+I) --> X(I) , FOR I FROM 1 TO NKEEP.
C
C       FOR A SHIFT UP (NSHIFT NEGATIVE), THE ELEMENTS OF
C       THE ARRAY ARE SHIFTED UP TO THE TOP OF THE ARRAY.
C       THE INPUT ARRAY MUST HAVE AT LEAST NKEEP-NSHIFT
C       ELEMENTS (REMEMBER THAT NSHIFT IS NEGATIVE).
C         X(I) --> X(I-NSHIFT), FOR I FROM NKEEP TO 1.
C
C
C PARAMETERS:
C       X      - INPUT ARRAY
C       NKEEP  - NUMBER OF ELEMENTS TO BE RETAINED
C       NSHIFT - NUMBER OF POSITIONS TO BE SHIFTED. NSHIFT
C                IS POSITIVE FOR A SHIFT DOWN AND NEGATIVE
C                FOR A SHIFT UP. NSHIFT EQUAL TO ZERO IS
C                ALSO PERMISSIBLE
C
C
C ROUTINES REQUIRED:
C       NONE
C
C
C AUTHOR / MAINTAINED BY:
C       P. KABAL
C
C
C DATE CREATED:
C       80/08/01
C
C
C UPDATES:
C
C
C---------------Bell-Northern Research / INRS Computer Laboratory-------------
```

```
      SUBROUTINE RSHIFT(X,NKEEP,NSHIFT)
C
C
      REAL X(1)
C
C
      IF(NKEEP.LE.0) GO TO 900
C
C SHIFT DOWN
      IF(NSHIFT.LT.0) GO TO 200
      DO 100 I=1,NKEEP
        X(I)=X(NSHIFT+I)
100   CONTINUE
      GO TO 900
C
C SHIFT UP
200   DO 300 I=NKEEP,1,-1
        X(I-NSHIFT)=X(I)
300   CONTINUE
C
900   RETURN
C
      END
```

B.5 Program Parameters - Simulation using Floating Point Arithmetic

```
NPOLES:
       8
LENFRM:
     160
LENWIN:
     204
IDECIM:
       5
PRE:
   0.9500

WND:
    1.000       0.9995       0.9980       0.9956       0.9921       0.9877       0.9824       0.9761       0.9689

FLTLP (CODER):
   1.2674E-02   3.2073E-03  -2.7730E-04  -6.3792E-03  -1.3978E-02  -2.0927E-02  -2.4485E-02  -2.1886E-02  -1.1060E-02   8.6959E-03
   3.6272E-02   6.8712E-02   0.1016       0.1300       0.1492       0.1560       0.1492       0.1300       0.1016       6.8712E-02
   3.6272E-02   8.6959E-03  -1.1060E-02  -2.1886E-02  -2.4485E-02  -2.0927E-02  -1.3978E-02  -6.3792E-03  -2.7730E-04   3.2073E-03
   1.2674E-02

FLTLP (DECODER):
   6.3372E-02   1.6037E-02  -1.3865E-03  -3.1896E-02  -6.9892E-02  -0.1046      -0.1224      -0.1094      -5.5299E-02   4.3480E-02
   0.1814       0.3436       0.5081       0.6500       0.7461       0.7801       0.7461       0.6500       0.5081       0.3436
   0.1814       4.3480E-02  -5.5299E-02  -0.1094      -0.1224      -0.1046      -6.9892E-02  -3.1896E-02  -1.3865E-03   1.6037E-02
   6.3372E-02

FLTSB (CODER):
   4.6269E-04   1.4714E-02  -1.2980E-03  -1.1001E-02   2.6800E-03   1.5114E-02  -5.2885E-03  -2.0392E-02   9.4255E-03   2.7451E-02
  -1.6094E-02  -3.7699E-02   2.7715E-02   5.5191E-02  -5.2157E-02  -9.7253E-02   0.1393       0.4593       0.4593       0.1393
  -9.7253E-02  -5.2157E-02   5.5191E-02   2.7715E-02  -3.7699E-02  -1.6094E-02   2.7451E-02   9.4255E-03  -2.0392E-02  -5.2885E-03
   1.5114E-02   2.6800E-03  -1.1001E-02  -1.2980E-03   1.4714E-02   4.6269E-04

FLTSB (DECODER):
   9.2538E-04   2.9428E-02  -2.5960E-03  -2.2002E-02   5.3599E-03   3.0229E-02  -1.0577E-02  -4.0784E-02   1.8851E-02   5.4902E-02
  -3.2188E-02  -7.5398E-02   5.5431E-02   0.1104      -0.1043      -0.1945       0.2786       0.9185       0.9185       0.2786
  -0.1945      -0.1043       0.1104       5.5431E-02  -7.5398E-02  -3.2188E-02   5.4902E-02   1.8851E-02  -4.0784E-02  -1.0577E-02
   3.0229E-02   5.3599E-03  -2.2002E-02  -2.5960E-03   2.9428E-02   9.2538E-04

FLTHP:
  -0.3169       0.4268      -6.7566E-02   9.7070E-02   9.8732E-02   6.6113E-02  -1.3248E-03  -9.7389E-02  -0.2072      -0.3096
  -0.3820      -0.4061      -0.3726      -0.2845      -0.1563       24.99       -39.88        16.22        0.2720       0.2672
   0.2156       0.1381       4.3982E-02  -3.4369E-02  -8.6380E-02  -0.1060      -9.5880E-02  -6.4821E-02  -2.4579E-02   1.0792E-02
  -0.1841       0.4557      -0.2028

DDF:
    1.000      -1.400        0.4000

 IQR(*,1):
   -0.9670      -0.9599      -0.9518      -0.9411      -0.9287      -0.9138      -0.8960      -0.8747      -0.8495      -0.8196
   -0.7845      -0.7435      -0.6960      -0.6416      -0.5799      -0.5107      -0.4342      -0.3508      -0.2515      -0.1675
   -7.0297E-02   2.8227E-02   0.1262       0.2218       0.3153       0.3993       0.4788       0.5512       0.6161       0.6737
    0.7240
 IQR(*,1):
   -0.9700      -0.9636      -0.9558      -0.9465      -0.9352      -0.9216      -0.9053      -0.8858      -0.8626      -0.8351
   -0.8027      -0.7647      -0.7206      -0.6697      -0.6117      -0.5462      -0.4733      -0.3933      -0.3068      -0.2150
   -0.1192      -2.1086E-02   7.7402E-02   0.1744       0.2681       0.3570       0.4399       0.5159       0.5846       0.6458
    0.6997       0.7467

 IQR(*,2):
   -0.2706      -0.1627      -5.0737E-02   6.2497E-02   0.1741       0.2815       0.3822       0.4746       0.5576       0.6308
    0.6942       0.7485       0.7942       0.8325       0.8642
 IQR(*,2):
   -0.3223      -0.2173      -0.1070       5.8987E-03   0.1187       0.2285       0.3328       0.4295       0.5173       0.5958
    0.6637       0.7225       0.7724       0.8142       0.8491       0.8778

 IQR(*,3):
   -0.7778      -0.7294      -0.6724      -0.6062      -0.5298      -0.4450      -0.3506      -0.2486      -0.1407      -2.9397E-02
    8.2660E-02   0.1927       0.2980       0.3966       0.4858
 IQR(*,3):
   -0.7990      -0.7546      -0.7020      -0.6405      -0.5695      -0.4889      -0.3989      -0.3005      -0.1952      -8.5325E-02
    2.6715E-02   0.1381       0.2461       0.3483       0.4428       0.5285

 IQR(*,4):
   -0.1445       1.9223E-02   0.1819       0.3352       0.4725       0.5902       0.6873
 IQR(*,4):
   -0.2240      -6.3042E-02   0.1012       0.2602       0.4061       0.5340       0.6413       0.7283

 IQR(*,5):
   -0.2718      -5.1623E-02   0.1737
 IQR(*,5):
   -0.3734      -0.1638       6.1828E-02   0.2613

 IQR(*,6):
    3.3781E-02   0.2661       0.4711
 IQR(*,6):
   -8.5407E-02   0.1520       0.3731       0.5586

 IQR(*,7):
    8.4042E-02
 IQR(*,7):
   -8.9940E-02   0.2530

 IQR(*,8):
    0.2434
 IQR(*,8):
    4.0850E-02   0.4267

 IQS:
    0.5000
 IQS:
    0.2500       0.7500
QMLT:
    0.8500       1.900
FSMN:
   10.00
FSMX:
   1000.
FSTH:
   15.00
```

B.6 Program Parameters - Simulation using Integer Arithmetic

```
NPOLES:
       8
LENFRM:
     160
LENWIN:
     204
IDECIM:
       5
PRE:
   0.9500

WND:
   3.2767E+04   3.2751E+04   3.2702E+04   3.2622E+04   3.2509E+04   3.2365E+04   3.2190E+04   3.1984E+04   3.1746E+04

FLTLP (CODER):
     415.3          105.1         -9.087         -209.0         -458.0         -685.7         -802.3         -717.2         -362.4          284.9
     1189.          2252.          3330.          4260.          4890.          5113.          4890.          4260.          3330.          2252.
     1189.          284.9         -362.4         -717.2         -802.3         -685.7         -458.0         -209.0         -9.087          105.1
     415.3

FLTLP (DECODER):
     2077.          525.5         -45.43         -1045.         -2290.         -3429.         -4012.         -3586.         -1812.          1425.
     5943.      1.1258E+04     1.6651E+04     2.1301E+04     2.4449E+04     2.5566E+04     2.4449E+04     2.1301E+04     1.6651E+04     1.1258E+04
     5943.          1425.         -1812.         -3586.         -4012.         -3429.         -2290.         -1045.         -45.43          525.5
     2077.

FLTSB (CODER):
     15.16          482.2         -42.53         -360.5          7.82          495.3         -173.3         -668.2          308.9          899.5
    -527.4         -1235.          908.2          1808.          709.         -3187.          456.4      1.5049E+04     1.5049E+04          456.4
    -3187.         -1709.          1808.          908.2         -1235.         -527.4          899.5          308.9         -668.2         -173.3
     495.3          87.82         -360.5         -42.53          482.2          15.16

FLTSB (DECODER):
     30.32          964.3         -85.07         -721.0          175.6          990.5         -346.6         -1336.          617.7          1799.
    -1055.         -2471.          1816.          3617.         -3418.         -6374.          9128.      3.0098E+04     3.0098E+04          9128.
    -6374.         -3418.          3617.          1816.         -2471.         -1055.          1799.          617.7         -1336.         -346.6
     990.5          175.6         -721.0         -85.07          964.3          30.32

FLTHP:
    -162.2          218.5         -34.59          49.70          50.55          33.85         -0.6783         -49.86         -106.1         -158.5
    -195.6         -207.9         -190.8         -145.6         -80.04      1.2794E+04     -2.0417E+04          8306.          139.3          136.8
     110.4          68.65          22.52         -17.60         -44.23         -54.28         -49.09         -33.19         -12.58          5.525
    -94.28          233.3         -103.8

DDF:
   1.6384E+04   -2.2938E+04          6554.

XQR(*,1):
    -0.9670        -0.9599        -0.9514        -0.9411        -0.9287        -0.9138        -0.8960        -0.8747        -0.8495        -0.8196
    -0.7845        -0.7435        -0.6960        -0.6416        -0.5799        -0.5107        -0.4342        -0.3508        -0.2615        -0.1675
   -7.0297E-02     2.8227E-02     0.1262         0.2218         0.3133         0.3993         0.4788         0.5512         0.6161         0.6737
     0.7240
YQR(*,1):
    -0.9700        -0.9636        -0.9558        -0.9465        -0.9352        -0.9216        -0.9053        -0.8858        86  8
    -0.8027        -0.7647        -0.7206        -0.6697        -0.6117        -0.5462        -0.4733        -0.3933        -0.3068        -0.2150
    -0.1192        -2.1086E-02     7.7402E-02     0.1744         0.2681         0.3570         0.4399         0.5159         0.5846         0.6458
     0.6997         0.7467

XQR(*,2):
    -0.2706        -0.1627        -5.0737E-02     6.2497E-02     0.1741         0.2815         0.3822         0.4746         0.5576         0.6308
     0.6942         0.7485         0.7942         0.8325         0.8642
YQR(*,2):
    -0.3223        -0.2173        -0.1070        5.8987E-03      0.1187         0.2285         0.3328         0.4295         0.5173         0.5954
     0.6637         0.7225         0.7724         0.8142         0.8491         0.8778

XQR(*,3):
    -0.7778        -0.7298        -0.6724        -0.6062        -0.5304        -0.4450        -0.3506        -0.2486        -0.1407        -2.9397E-02
     8.2660E-02     0.1927         0.2980         0.3966         0.4868
YQR(*,3):
    -0.7990        -0.7546        -0.7020        -0.6405        -0.5695        -0.4889        -0.3989        -0.3005        -0.1952        -8.5325E-02
     2.6715E-02     0.1381         0.2461         0.3483         0.4428         0.5285

XQR(*,4):
    -0.1445         1.9223E-02     0.1819         0.3352         0.4725         0.5902         0.6873
YQR(*,4):
    -0.2240        -6.3042E-02     0.1012         0.2602         0.4061         0.5340         0.6413         0.7283

XQR(*,5):
    -0.2718        -5.1623E-02     0.1737
YQR(*,5):
    -0.3734        -0.1638         6.1828E-02     0.2813

XQR(*,6):
     3.3781E-02     0.2661         0.4711
YQR(*,6):
    -8.5407E-02     0.1520         0.3731         0.5586

XQR(*,7):
     8.4042E-02
YQR(*,7):
    -8.9940E-02     0.2530

XQR(*,8):
     0.2838
YQR(*,8):
     4.0850E-02     0.4267

XQS:
     0.5000
YQS:
     0.2500         0.7500
QMLT:
     0.8500         1.900
FSMN:
     10.00
FSMX:
     1000.
FSTH:
     15.00
```

References

1. D. Sloan, "Adaptive Transform Coding of Speech", INRS Technical Report 79-05, July 1979.
2. J. Markel and A.H. Gray, Linear Prediction of Speech, Springer-Verlag, 1976.
3. P. Mermelstein and M. Nakatsui, "Intelligibility Evaluation of a Simulated 4.8 kb/s Residual Excited Prediction Coder", INRS Technical Report 80-05, Report for the Department of Communications, March 1980.
4. C.K. Un and D.T. Magill, "The Resiual-Excited Linear Prediction Vocoder with Transmission Rate Below 9.6 kbits/s", IEEE Trans. Commun., vol. COM-23, pp. 1466-1474, Dec. 1975.
5. Y. Tohkura, F. Itakura and S. Hashimoto, "Spectral Smoothing Technique in PARCOR Speech Analysis-Synthesis", IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-26, pp. 587-596, Dec. 1978.
6. Y. Tohkura and F. Itakura, "Spectral Sensitivity Analysis of PARCOR Parameters for Speech Data Compression", IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-27, pp. 273-280, June 1979.
7. J. LeRoux and C.J. Gueguen, "A Fixed Point Computation of Partial Correlation Coefficients", IEEE Trans. Acoust., Speech, Signal Processing, vol. ASSP-25, pp. 257-259, June 1977.
8. L.R. Rabiner and R.W. Schafer, Digital Processing of Speech Signals, Prentice Hall, 1978.
9. N.S. Jayant, "Adaptive Quantization with a One-Bit Memory", Bell Syst. Tech. J., vol. 55, pp. 1119-1144, Sept. 1973.
10. R.E. Crochiere, "A Mid-Rise/Mid-Tread Quantizer Switch for Improved

Idle-Channel Performance in Adaptive Coders", Bell Syst. Tech. J., vol. 576, pp. 2953-2955, Oct. 1978.

11. F.J. MacWilliams and N.J.A. Sloane, The Theory of Error-Correcting Codes, North-Holland, 1977.
12. J.H. McClellan, T.W. Parks and L.R. Rabiner, "A Computer Program for Designing Optimum FIR Linear Phase Digital Filters", IEEE Trans. Audio Electroacoust., vol. AU-21, pp.506-526, Dec. 1973.
13. A. Roset, "Application des Filtres Mirroirs a un Procede de Codage par Decoupage en Sous-Bandes", INRS Technical Report 80-03, Jan. 1980.
14. J.D. Markel and A.H. Gray, Jr., "Fixed-Point Truncation Arithmetic Implementation of a Linear Prediction Autocorrelation Vocoder", IEEE Trans. Acoustics, Speech, Signal Processing, vol. ASSP-22, pp. 273-282, Aug. 1974.
15. M.S. Krieger and P.J. Plauger, "C Language's Grip on Hardware Makes Sense for Small Computers", Electronics, pp. 130-133, May 8, 1980.
16. E.M. Hofstetter, J. Tierney and O. Wheeler, " Microprocessor Realization of a Linear Predictive Vocoder", IEEE Trans. Acoustics, Speech, Signal Processing, vol. ASSP-25, pp. 379-387, Oct. 1977.
17. H.T. Nagle, Jr. and V.P. Nelson, "Digital Filter Implementation on 16-Bit Microcomputers", IEEE Micro, pp. 23-41, Feb. 1981.
18. D.F. Millet, "A Signal Processor Fast Enough to do Real-Time Voice-Band", Electronic Design, pp. 86-89, Feb. 15, 1980.
19. H.-J. Schloss and G. Ballenberger, "A High Speed Small Scale Signal Processor", Signal Processing, vol. 3, pp. 29-36, Jan. 1981.
20. R.W. Schafer and L.R. Rabiner, "A Digital Signal Processing Approach to Interpolation", Proceedings of the IEEE, vol. 61, pp. 692-702, June 1973.

21. G. Oetken, T.W. Parks and H.W. Schussler, "New Results in the Design of Digital Interpolators", IEEE Trans. Acoustics, Speech, Signal Processing, vol. ASSP-23, pp.301-309, June 1975.

22. A. Papoulis, Signal Analysis, pp. 193-196, McGraw-Hill, 1977.

23. D. Estaban and C. Galand, "Application of Quadrature Mirror Filters to Split Band Voice Coding Schemes", Record IEEE Conf. on Speech, Acoust., Signal Processing, pp. 191-195, May 1977.