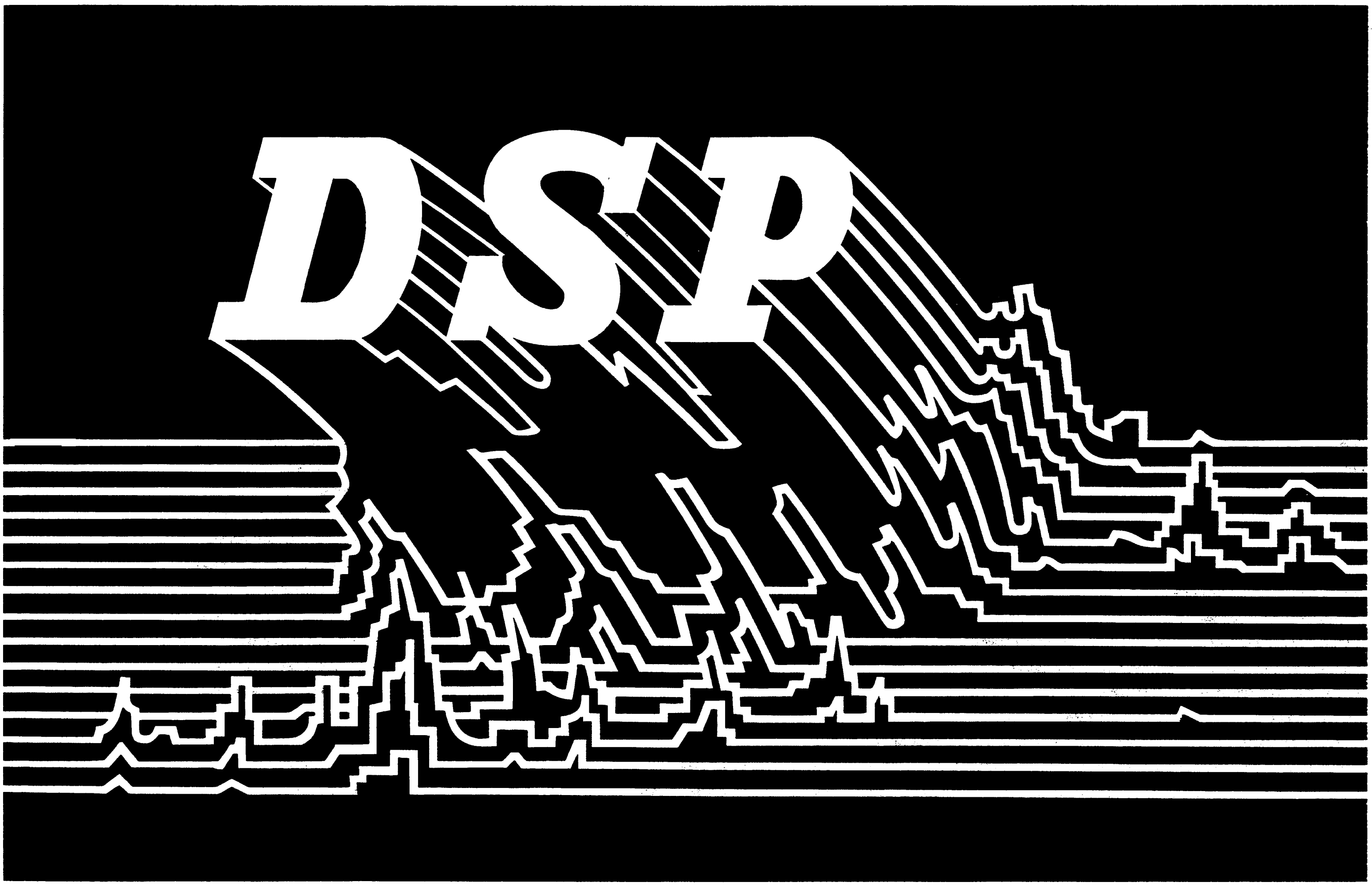
APR404/D

# G.722 Audio Processing on the DSP56100 Microprocessor Family




MOTOROLA

# Motorola Digital Signal Processors

## G722 Audio Processing on the DSP56100 Microprocessor Family

**by Phil Atherton**

**DSP Applications Engineer**

**Motorola Ltd.**

**East Kilbride**

**The application described in this document was developed in collaboration with the Centre National d'Études des Télécommunications, Lannion, France. Motorola gratefully acknowledge their valuable contribution.**



APR404/D

# Table of Contents

# List of Illustrations

# List of Tables

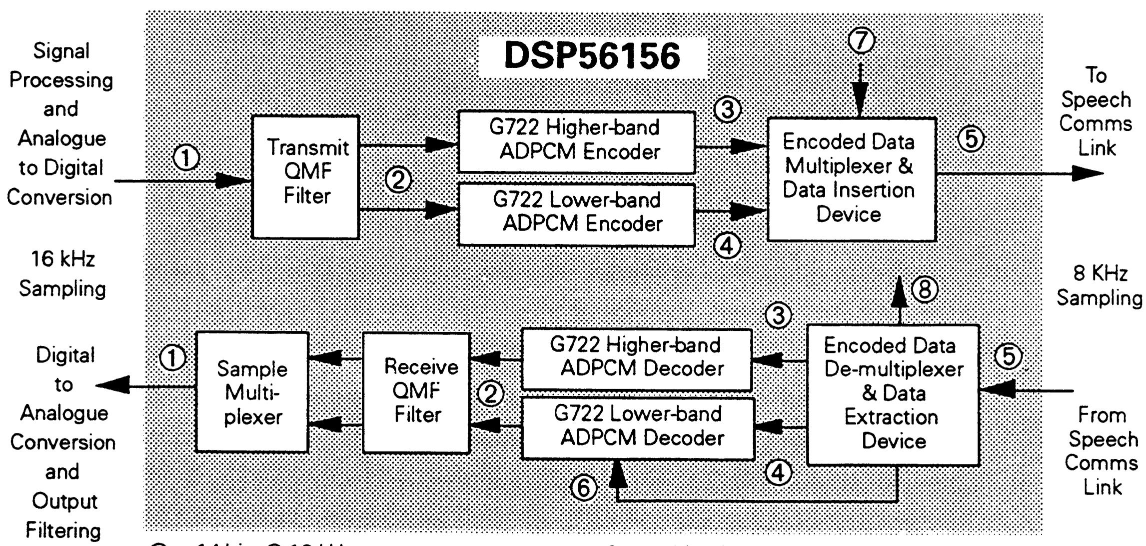
| Table Number | Title | Page Number |
|---|---|---|

# SECTION 1
# INTRODUCTION

This application note describes the implementation of a speech codec that conforms to the CCITT standardised G.722 specification for 7kHz audio-coding within 64 kbit/s [1]. The target processor for which the code was developed is the first in Motorola's 56100 16-bit Digital Signal Processor family, the DSP56156 [2].

The G.722 specification details the characteristics of an audio (50Hz to 7kHz) coding system that may be used for a variety of higher quality speech applications. These characteristics relate to everything from the anti-aliasing filter mask at the transmitting terminal, to the reconstructing filter mask at the receiving terminal. Within this document, however, we concern ourselves only with the software coding aspects of the specification.

The coding system uses Sub-Band Adaptive Differential Pulse Code Modulation (SB-ADPCM) to decimate a signal sampled at 16 kHz and 14 bits or 224 kbit/s to digital data for transmission at 8 bits and 8kHz or 64 kbit/s. Figure 1-1 gives an overview of the signal flow through the processor and the status of the signals at relevant points within the codec.




① : 14 bits @ 16 kHz
② : 14 bits @ 8 kHz
③ : 2 bits @ 8 kHz
④ : 6 bits @ 8 kHz
(6, 5 or 4 bits for the Decoder)
⑤ : 8 bits @ 8 kHz
⑥ : Mode Indicator from transmitter
(requires a comms. protocol)
⑦ : Auxiliary data channel input 0, 8 or 16 kbit/s
⑧ : Auxiliary data channel output 0, 8 or 16 kbit/s

**Figure 1-1** G.722 Signal Flow through the DSP56156

In the SB-ADPCM technique used, the frequency band is split into two sub-bands (lower and higher) of 50Hz – 4kHz and 4kHz – 7kHz and the signals from each sub-band are subsequently encoded using ADPCM. The 50 Hz lower cut-off frequency is set by analogue filtering before A/D conversion.

The G.722 specification details three basic modes of system operation using the algorithm, two of which allow the insertion of data into 1 or 2 lsb's of the 8 bits of transmission data immediately prior to their being sent. The three modes give rise to the 8-bit data transmission formats indicated by Figure 1-2 below. This means that modes 2 and 3 allow the insertion of data into the transmitted byte providing an auxiliary data channel of either 8 or 16 kbit/s respectively, by making use of bits from the lower sub-band. This has a slightly detrimental effect on the Signal-to-Noise Ratio ( SNR) of the reconstructed signal.

M.S.B. — Data Transmission Format for Mode 1 : 64 kbit/s speech — L.S.B.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| H.B | H.B | L.B. | L.B. | L.B. | L.B. | L.B. | L.B. |

M.S.B. — Data Transmission Format for Mode 2 : 56 kbit/s speech, 8 kbit/s data — L.S.B.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| H.B | H.B | L.B. | L.B. | L.B. | L.B. | L.B. | DATA |

M.S.B. — Data Transmission Format for Mode 3 : 48 kbit/s speech, 16 kbit/s data — L.S.B.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| H.B | H.B | L.B. | L.B. | L.B. | L.B. | DATA | DATA |

KEY : L.B. = Lower Band Encoded Signal H.B. = Higher Band Encoded Signal

**Figure 1-2** Data Formats for 64 Kbit/s Channel

The G.722 algorithm is well suited to the DSP56156 in so far as:

a) the signal information is contained in 14 bits, allowing direct interface of 16-bit Analogue to Digital converters, such as Motorola's 16-bit Sigma-Delta 56ADC16;

b) intermediate accumulation results for the sub-band filters require a minimum resolution of 24 bits which can be adequately accommodated in the processor's 40-bit accumulators;

c) full duplex operation of the algorithm requires 9.41 MIPS peak processor performance, and the excess power of the device allows other algorithms such as echo-cancelling and protocols such as the CCITT's H.221 and G.725 to be included on a single chip;

d) when used as an ISDN terminal with G.722 speech coding and the H.221 and G.725 protocols, the standard G.711 $\mu$ and A-Law PCM fallback modes can be easily included by using the companding hardware built into both of the processor's two Synchronous Serial Interfaces or SSI's.

This application note also gives a brief overview of the latest trends in speech coding techniques and where the G.722 algorithm fits into these developments.

# SECTION 2
# THE G.722 ALGORITHM

This section describes the implementation of the G.722 algorithm on the DSP56156 processor. Figure 1-1 details the G.722 data flow through the processor, from which the sub-sections that follow have been derived.

## 2.1 Quadrature Mirror Filters

Since the development of the QMF filter pair by Esteban and Galand [3] the technique of frequency sub-band splitting has been researched extensively with interesting results. These particular digital filter structures allow signals to be divided into frequency sub-bands and de-sampled or decimated without loss of information upon reconstruction at the original sampling frequency. Originally the technique was developed as a result of work undertaken by Crochiere *et al* [9] as a means of reducing the effects of quantisation noise due to coding. The main advantages of this approach are:

a) the localisation of quantisation noise into the frequency sub-bands and in so doing preventing noise interference between the bands and,

b) the enabling of bit resource allocation to the frequency sub-band signals according to certain spectral criteria.

Originally the technique allowed only approximate reconstruction of the decomposed signal. However, with the wealth of development work that has been carried out it is now known that sub-band signals can be reconstructed perfectly [5][7][8] with linear-phase FIR filters, allowing alias and phase distortion free reconstruction. The work has been extended to include multiple sub-bands and multirate filter banks are comprehensively covered in [7].

The following description is a brief introduction to QMF filtering [4]. Initially, consider the two sub-band coder of Figure 2-1. The blocks A-B-C represent the cascade of an encoder, transmission channel and decoder of some nature. Their presence will, however, be ignored for the purposes of the present discussion.

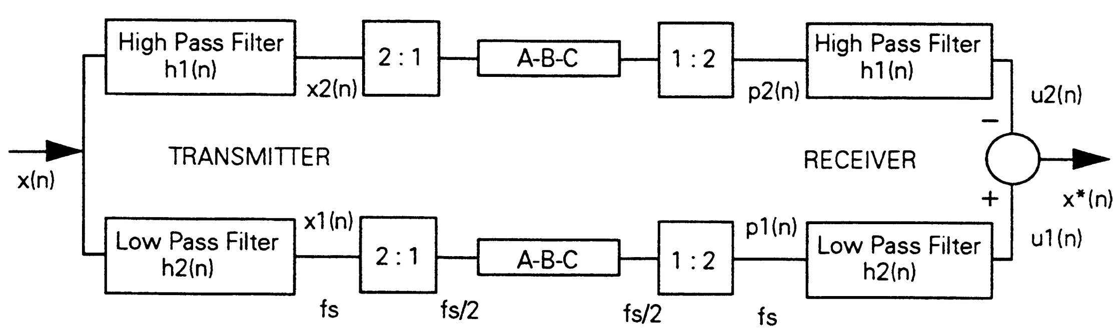



**Figure 2-1** Quadrature Mirror Filters in a 2-band Sub-band Coder

The basic process of QMF filtering is designed to overcome the effect of non-ideal transition-band and stop-band filtering. With real-world filters, the non-zero signal energy in the transition and stop bands is reflected back into the pass-band during the interpolation process at the receiver causing aliasing. This aliasing is cancelled in the QMF bank during reconstruction of the signal at the summing junction indicated in Figure 2-1.

To obtain the cancellation property, filters h1(n) and h2(n) must respectively be symmetrical and anti-symmetrical FIR filters with an even number of taps, i.e.

$$h1(n) = h2(n) = 0 \qquad \text{for } n<0 \text{ and } n\geq N \tag{1}$$

where N is the number of taps. For symmetry and asymmetry the following restrictions are implied:

$$\begin{aligned} h1(n) &= h1(N-1-n) \\ & \qquad\qquad n = 0,1,2,3,\ldots\ldots,(N/2)-1 \\ h2(n) &= -h2(N-1-n) \end{aligned} \tag{2}$$

Equation 2 gives the necessary condition for FIR filters with constant group delay. This eliminates signal distortion due to different phase shifting of the individual frequency components that constitute the input signal as it passes through the filter.

The filter bank indicated must also satisfy the relationship indicated in equation (3), which is the mirror image relationship of the filters.

$$h2(n) = (-1)^n h1(n) \qquad n = 0,1,2,\ldots\ldots,N-1 \tag{3}$$

In order to obtain perfect reconstruction the combined filter passband responses must be flat, and to satisfy this requirement the filters must have responses which conform to:

$$\left|H1\,(e^{jw})\right|^2 + \left|H2\,(e^{jw})\right|^2 = 1 \tag{4}$$

where $Hx(e^{jw})$ is the Fourier Transform of hx(n). A more detailed analysis of this structure is presented in [3].

In the G.722 specification the implementation of the QMF filters is described by equations 5 through 12. Equations 5 to 8 represent the transmit QMF filter and 9 to 12 the receive QMF filter.

The transmit output variables, xL(n) and xH(n), are computed in the following manner:

$$xA = \sum_{i=0}^{11} h(2i) * xin(j-2i) \tag{5}$$

$$xB = \sum_{i=0}^{11} h(2i+1) * xin(j-2i-1) \tag{6}$$

$$xL(n) = xA + xB \tag{7}$$

$$xH(n) = xA - xB \tag{8}$$

For the receive filter, the D/A output variables are calculated as follows:

$$xout(j) = 2\sum_{i=0}^{11} h(2i) * xd(i) \tag{9}$$

$$xout(j+1) = 2\sum_{i=0}^{11} h(2i+1) * xs(i) \tag{10}$$

$$xd(i) = rL(n-i) - rH(n-i) \tag{11}$$

$$xs(i) = rL(n-i) + rH(n-i) \tag{12}$$

where:

| | |
|---|---|
| index (j-1) | = value corresponding to the previous 16 kHz sampling interval. |
| index (j) | = value corresponding to the current 16 kHz sampling interval. |
| index (j+1) | = value corresponding to the next 16 kHz sampling interval. |
| index (n-1) | = value corresponding to the previous 8 kHz sampling interval. |
| index (n) | = value corresponding to the current 8 kHz sampling interval. |
| rL(n-i) | = lower band reconstructed signal delay line |
| rH(n-i) | = higher band reconstructed signal delay line |

An efficient implementation of the band-splitting and reconstruction can be realised using the structures indicated in Figure 2-2.

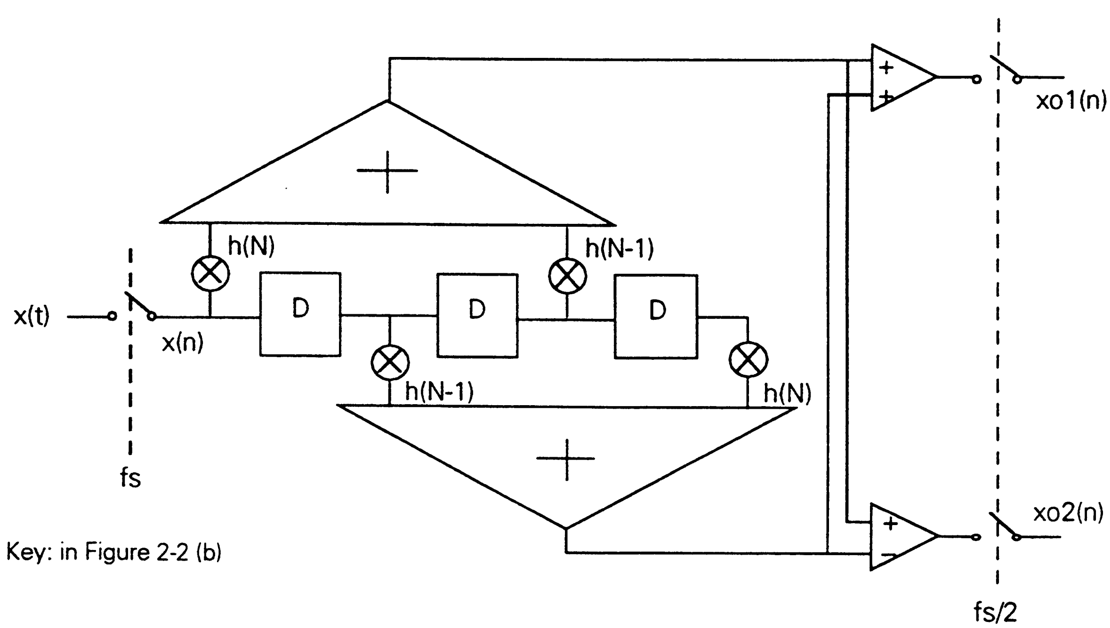
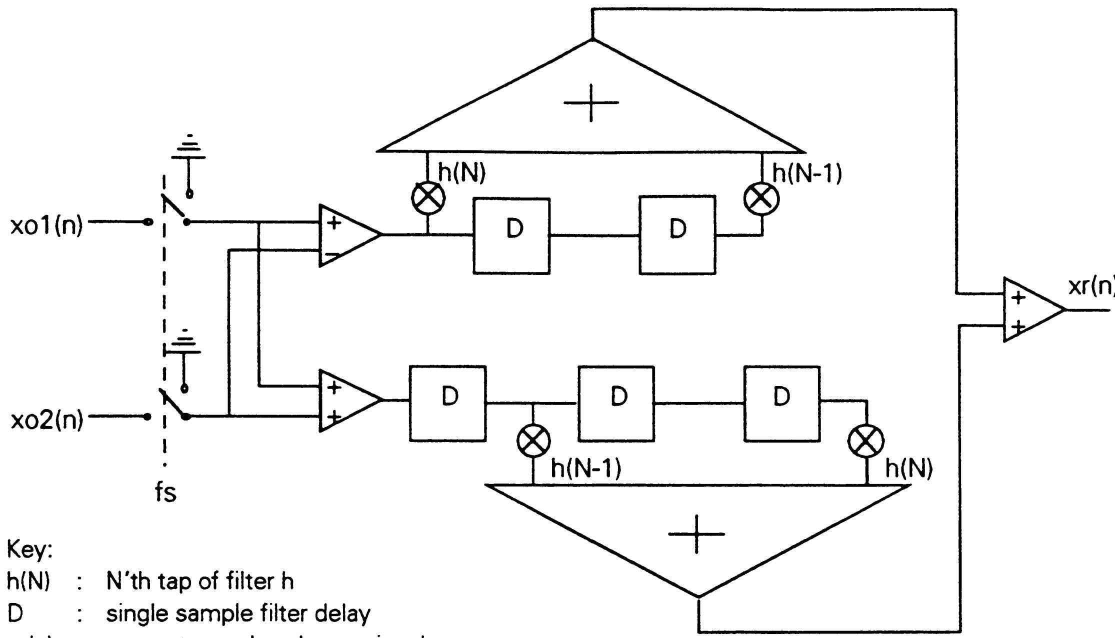



**Figure 2-2** Efficient QMF Processing Implementation
(a) Channel Splitting Using QMF Structure




Key:
h(N) : N'th tap of filter h
D : single sample filter delay
xr(n) : reconstructed analogue signal
fs/2 : half the analogue sampling frequency

**Figure 2-2** Efficient QMF Processing Implementation
(b) Channel Reconstruction Using QMF Structure

Mathematical analysis of the QMF processing shows the reconstructed signal, xr(n), is a perfect replica of the input signal, x(n), but half its magnitude and delayed by an amount equal to (N-1) sample periods.

In the QMF splitter of Figure 2-2(a), xo1(n) represents the lower band channel and xo2(n) the higher band channel. In the DSP56156 implementation these outputs are stored before subsequent encoding by the G.722 lower and higher band ADPCM algorithms in the variables 'xl_cod' and 'xh_cod' respectively.

The FIR filters represented are 2-tap, but this is just for reference. The actual number of taps per filter implemented in the code is 12.

For the filter calculations, in order to avoid excessive signal distortion due to rounding and truncation errors, intermediate multiply-accumulate results require an accumulator with at least 24 bits of resolution. The 56100 core has 40-bit accumulators.

The decimation of 2 between the analogue and digital conversion sampling process at 16 kHz and the ISDN line communication frequency of 8 kHz can be efficiently implemented using the interrupt processing structure shown in Figure 2-3. This structure has not been included in the code as it stands but may be used as a working reference.

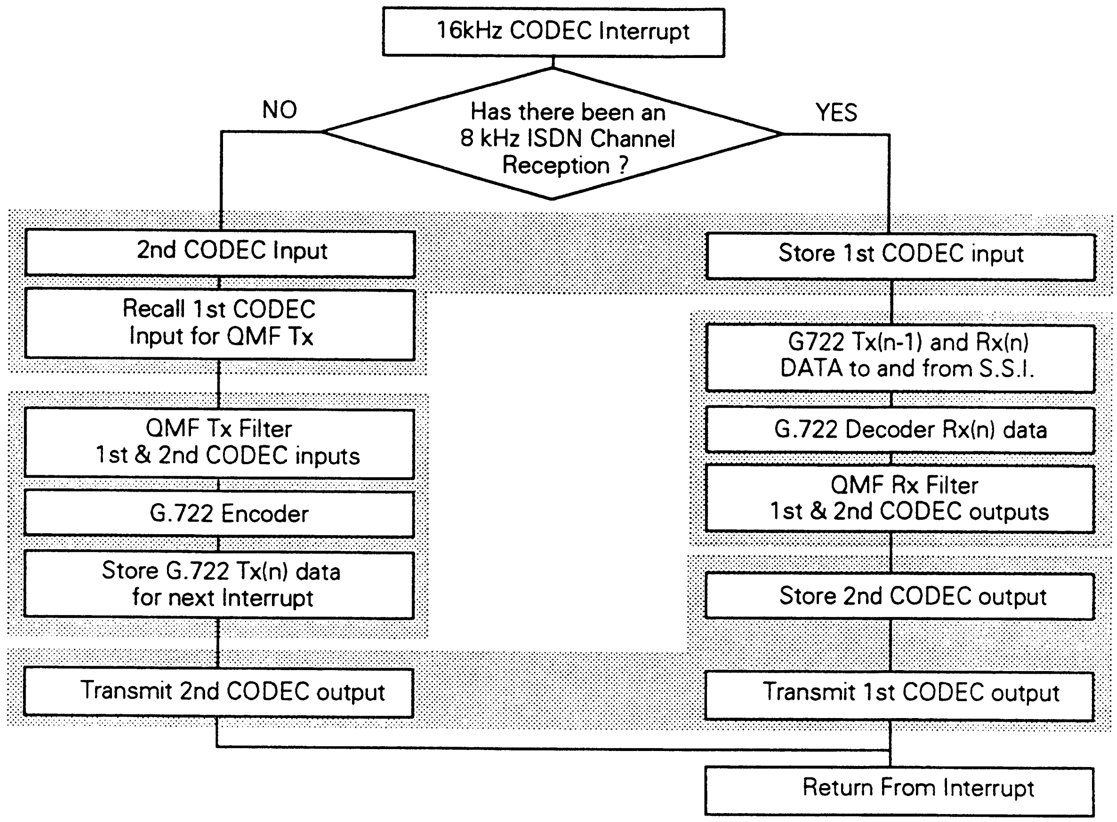



**Figure 2-3** G.722 Processing after A/D Conversion Reception

Table 2-1 shows the coefficients used in the G.722 QMF filters.

**Table 2-1** QMF Coefficient Values

| Coefficient | | Value | | $2^{13}$ Scaled Value |
|---|---|---|---|---|
| $h_0$ | $h_{23}$ | 0.366211 | exp–03 | 3 |
| $h_1$ | $h_{22}$ | –0.134277 | exp–02 | 11 |
| $h_2$ | $h_{21}$ | –0.134277 | exp–02 | 11 |
| $h_3$ | $h_{20}$ | 0.646973 | exp–02 | 53 |
| $h_4$ | $h_{19}$ | 0.146484 | exp–02 | 12 |
| $h_5$ | $h_{18}$ | –0.190430 | exp–01 | –156 |
| $h_6$ | $h_{17}$ | 0.390625 | exp–02 | 32 |
| $h_7$ | $h_{16}$ | 0.441895 | exp–01 | 362 |
| $h_8$ | $h_{15}$ | –0.256348 | exp–01 | –210 |
| $h_9$ | $h_{14}$ | –0.982666 | exp–01 | –805 |
| $h_{10}$ | $h_{13}$ | 0.116089 | exp–00 | 951 |
| $h_{11}$ | $h_{12}$ | 0.473145 | exp–00 | 3876 |

The scaled values are the ones implemented in the G.722 code. They have been multiplied by a factor of $2^{13}$ but to maintain proper scaling within the DSP56156 the scaling should be by $2^{15}$.

**Table 2-2** Data I/O Formats used

| Variable Name | Binary Representation | Source |
|---|---|---|
| Xin | S S –2 –3 ........ –14 –15 | G.722 Spec. |
| Xout | S S –2 –3 ........ –14 –15 | G.722 Spec. |
| Xin | S –1 –2 –3 ........ –14 –15 | A/D input |
| Xout | S –1 –2 –3 ........ –14 –15 | D/A output |

Key: S = Sign   $-x = 2^{-x}$

Bearing in mind the scaling of the coefficients and specified input/output data given in Table 2-1 and Table 2-2, a scaling of 2 still has to be applied to each data and coefficient multiplication on the DSP56156 to maintain the scaling indicated in the G.722 specification. This is taken into account in software by pre-multiplying the coefficients by 2 before storage in the DSP data memory.

The QMF sections of the G.722 assembly code implemented on the DSP56156 can be found in Appendix A, sections 1 and 2, at the end of this document.

## 2.2 G.722 ADPCM Encoders

Some familiarity with Differential Pulse Code Modulation techniques as discussed in [10][11] is assumed.

ADPCM coders combine principles from two basic speech coding techniques, Adaptive PCM (APCM) and Differential PCM (DPCM) [10][11]. There are two basic differences between these and uniform and log PCM techniques : APCM and DPCM codecs a) require knowledge of previous speech sample values to make decisions upon signal scaling factors and signal prediction levels and b) make use of adaptive quantisation step sizes.

Speech signals tend to have gradual transitions in amplitude. Adaptive PCM codecs exploit this property by changing or adapting the quantisation characteristics of the algorithm in sympathy with the amplitude of the speech signal being coded. This gives the impression of a greater dynamic range from the codec. There are two ways of adapting the quantisation characteristics of the coder, which are:

a) direct modification of the quantiser step sizes; and

b) scaling the captured speech signal by multiplication with a gain factor.

The G.722 algorithm uses the second technique, which employs a gain factor for scaling the incoming speech signal.

Signal Gain Factor updating can be performed in one of two ways:

a) the feedforward approach, whereby the update information is passed to the receiving decoder across the transmission channel; and

b) the feedback approach, which uses previously coded values to determine the update.

With the feedback approach, no extra information is passed to the receiving decoder, freeing channel bandwidth for more of the signal.

Differential PCM codecs use the fact that speech signals have a high degree of sample-to-sample correlation to estimate the magnitude of the next speech sample. This signal estimate is stored and subtracted from the next actual speech sample (when it is received) to provide a difference value. The difference value is then quantised. The magnitude of this difference value is therefore dependent upon the accuracy of the signal prediction scheme used and ideally should be considerably smaller than the original speech sample. With the number of quantisation levels remaining the same as for a full-scale speech level, the step size of the original speech quantiser can therefore be reduced allowing more precise quantisation.

The G.722 algorithm [1] utilises two, independent, feedback Adaptive Differential Pulse Code Modulation (ADPCM) encoders for the compression of the two signals output from the QMF transmit filter. The two signals, already decimated to the ISDN line communication frequency of 8kHz, are now represented by a linearly quantised and filtered 14 bits. These signals are then applied to the ADPCM encoders, which encode the 'xh_cod' 14-bit signal into 2 bits for the higher-band and the 'xl_cod' 14-bit signal into 6 bits for the lower-band. The number of bits allocated to the lower-band is greater due to statistically higher energy spectral density in the frequency band up to 4kHz for speech signals.

The CCITT higher band and lower band encoders are represented in Figures 2-4 (a) and (b).

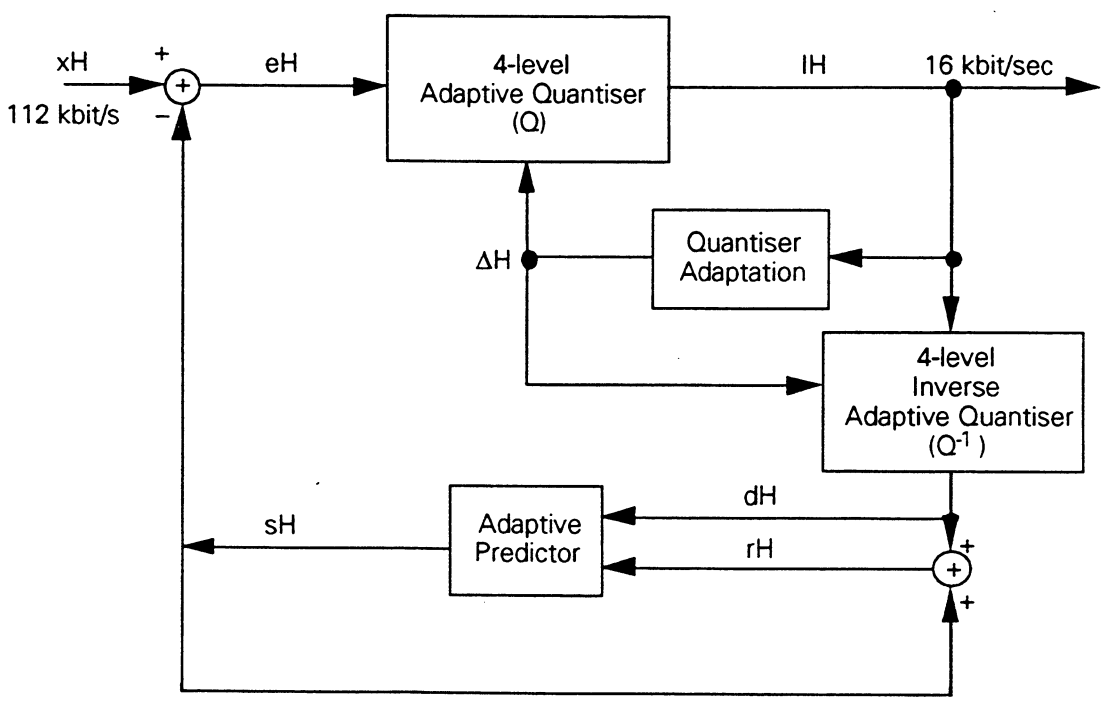



**Figure 2-4** (a) Higher Band G.722 Encoder

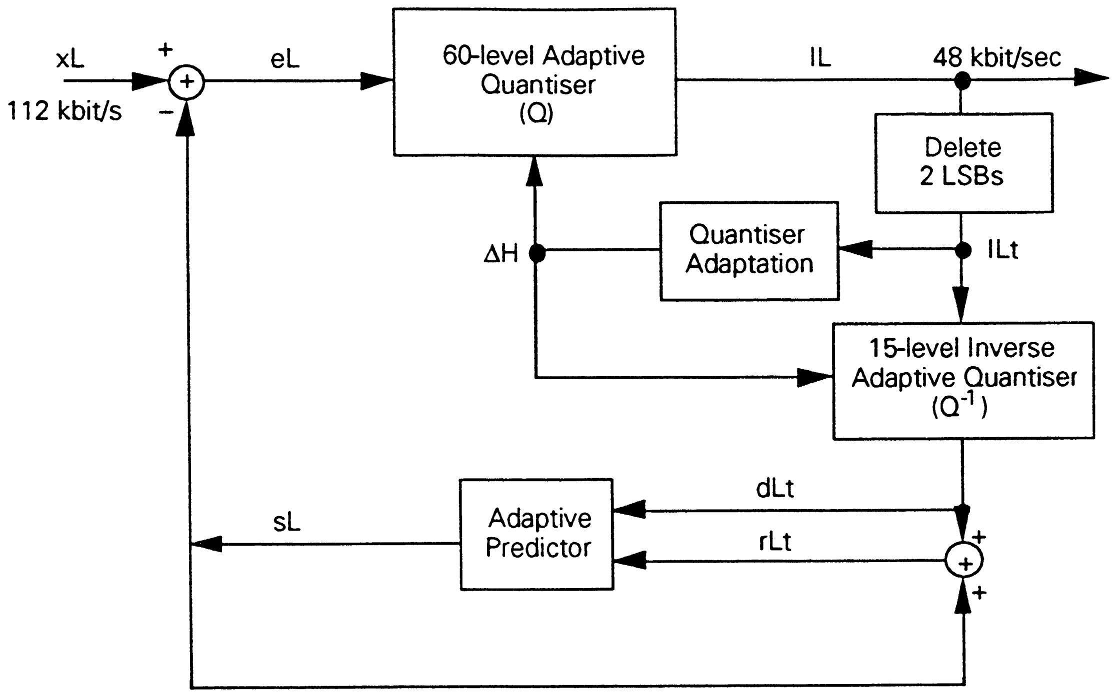



**Figure 2-4** (b) Lower Band G.722 Encoder

Figure 2-5 depicts the encoder implementations in modular form. Each block performs a specific function of the G.722 algorithm and for each of these reusable assembly code modules have been written that simplify the code structure.

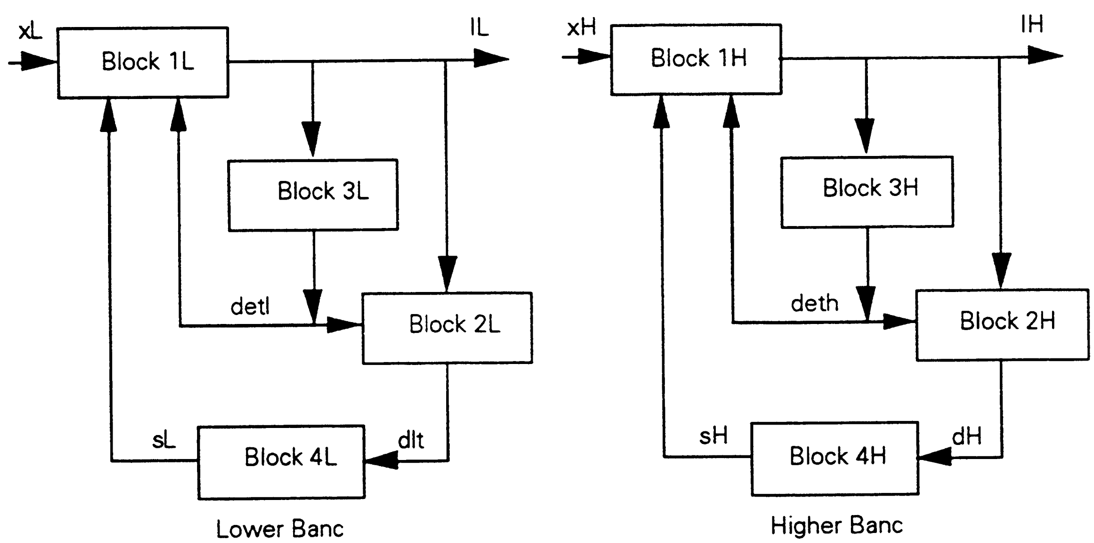



**Figure 2-5** G.722 Encoders in Modular Format

A major portion of the G.722 code involves scale factor adaptation and lower and higher band signal prediction procedures (blocks 2x, 3x and 4x). The algorithm for performing these procedures is identical for both lower and higher bands, the only difference being the array pointers and data values used. This allows the same assembly code to be used for both sub-bands without unnecessary duplication of code.

Consider first blocks 1L and 1H. These represent the error signal calculation and its subsequent quantisation for each band. These blocks therefore indicate two independent processes. In the G.722 specification these functions are performed by the routines 'Subtra', 'Quantl' and 'Quanth'. The operations performed by these blocks are described by equations 13 through 18.

The difference signals, eL(n) and eH(n), are calculated according to equations 13 and 14 before quantisation into 6 bits for the lower band and 2 bits for the higher band respectively.

$$eL(n) = xL(n) - sL(n-1) \quad (13)$$

$$eH(n) = xH(n) - sH(n-1) \quad (14)$$

where:

| | | |
|---|---|---|
| xL(n) | = | Lower-band speech value in current 8 kHz sampling interval. |
| xH(n) | = | Higher-band speech value in current 8 kHz sampling interval. |
| sL(n-1) | = | Lower-band speech prediction in previous 8 kHz sampling interval. |
| sH(n-1) | = | Higher-band speech prediction in previous 8 kHz sampling interval. |

Tables 2-2 and 2-3 give the quantiser decision levels and corresponding output codes for the 6-bit and 2-bit quantisers. The interval boundaries, LL6, LU6, HL and HU, are scaled by computed scale factors, $\Delta L(n)$ and $\Delta H(n)$, according to equations 15 and 16. Once the appropriate quantiser interval has been determined for each band the indices (or offsets) mL and mH are then used to select the corresponding output codes IL and IH according to equations 17 and 18.

$$LL6(mL)*\Delta L(n) \leq |eL(n)| < LU6(mL)*\Delta L(n) \quad (15)$$

$$HL(mH)*\Delta H(n) \leq |eH(n)| < HU(mH)*\Delta H(n) \quad (16)$$

The output codes, ILN and IHN, represent negative quantiser intervals whilst, ILP and IHP, represent positive intervals (the quantiser decision levels are symmetric about zero).

$$IL(n) = \begin{cases} ILP(mL), & \text{if } eL(n) \geq 0 \\ ILN(mL), & \text{if } eL(n) < 0 \end{cases} \quad (17)$$

$$IH(n) = \begin{cases} IHP(mH), & \text{if } eH(n) \geq 0 \\ IHN(mH), & \text{if } eH(n) < 0 \end{cases} \quad (18)$$

Blocks 2L and 2H consist of the Inverse Quantisation routines 'INVQAL' and 'INVQAH' respectively for the higher and lower sub-bands prior to signal prediction. The inverse quantisation process is based upon the 4 msb bits of IL(n) for the lower band regardless of the G.722 mode of operation. This enables consistent signal prediction performance even in G.722 mode 3 (48 kbit/s speech, 16 kbit/s data) communication.

Blocks 3L, 4L, 3H and 4H form the remaining scale factor ($\Delta L(n)$ and $\Delta H(n)$) adaptation and signal prediction portions of the lower band and higher band G.722 encoders respectively. The individual routines detailed in the G.722 specification that comprise block 3L are 'Logscl'and 'Scalel' for scale factor adaptation. For block 4L the routines are: 'Parrec', 'Recons', 'Upzero', 'Uppol2', 'Uppol1', 'Filtez', 'Filtep' and 'Predic' for signal prediction. For higher band blocks 3H and 4H the scale factor adaptation routines become 'Logsch' and 'Scaleh' whereas the signal prediction routines are the same. The scale factor adaptation procedure is represented in section 2-4, 'G.722 Signal Prediction', in modular format.

The G.722 encoder quantisation processes within the DSP56156 implementation adhere to the equations presented on pages 279 to 288 of the G.722 specification [1]. These have not been included in this report for simplicity but it is recommended that reference should be made to these as an aid when reading the code provided.

In the lower sub-band encoder assembly code, the pointer addresses 'cod_6_pl' and 'cod_6_mi' represent the quantiser positive and negative output codes corresponding to the codes presented in columns ILP and ILN of Table 2-2.

In the implementation described here the signal decision levels represented by columns LL6 and LU6 in Table 2-2, and HL and HU in Table 2-3 have been pre-multiplied by 8 before storage. This eliminates the need for the left shift of 3 places demanded by the 'Quantl' and 'Quanth' routine descriptions in the G.722 specification.

As the signal predictor routines are common to both higher and lower band encoders and decoders the description of the implementation and the associated assembly code is given in section 2-3, entitled 'G.722 Signal Prediction'.

The assembly code for the G.722 encoder is provided in Appendix A, section 3.

**Table 2-3** Decision Levels and Output Codes for the Lower Band Quantiser

| Index (Offset+1) mL | Decision Level LL6 | Decision Level LU6 | Output Code ILN | Output Code ILP |
|---|---|---|---|---|
| 1 | 00 | 35 | 111111 | 111101 |
| 2 | 35 | 72 | 111110 | 111100 |
| 3 | 72 | 110 | 011111 | 111011 |
| 4 | 110 | 150 | 011110 | 111010 |
| 5 | 150 | 190 | 011101 | 111001 |
| 6 | 190 | 233 | 011100 | 111000 |
| 7 | 233 | 276 | 011011 | 110111 |
| 8 | 276 | 323 | 011010 | 110110 |
| 9 | 323 | 370 | 011001 | 110101 |
| 10 | 370 | 422 | 011000 | 110100 |
| 11 | 422 | 473 | 010111 | 110011 |
| 12 | 473 | 530 | 010110 | 110010 |
| 13 | 530 | 587 | 010101 | 110001 |
| 14 | 587 | 650 | 010100 | 110000 |
| 15 | 650 | 714 | 010011 | 101111 |
| 16 | 714 | 786 | 010010 | 101110 |
| 17 | 786 | 858 | 010001 | 101101 |
| 18 | 858 | 940 | 010000 | 101100 |
| 19 | 940 | 1023 | 001111 | 101011 |
| 20 | 1023 | 1121 | 001110 | 101010 |
| 21 | 1121 | 1219 | 001101 | 101001 |
| 22 | 1219 | 1339 | 001100 | 101000 |
| 23 | 1339 | 1458 | 001011 | 100111 |
| 24 | 1458 | 1612 | 001010 | 100110 |
| 25 | 1612 | 1765 | 001001 | 100101 |
| 26 | 1765 | 1980 | 001000 | 100100 |
| 27 | 1980 | 2195 | 000111 | 100011 |
| 28 | 2195 | 2557 | 000110 | 100010 |
| 29 | 2557 | 2919 | 000101 | 100001 |
| 30 | 2919 | ∞ | 000100 | 100000 |

**Table 2-4** Decision Levels and Output Codes for the Higher Band Quantiser

| Index (Offset+1) mH | Decision Level HL | Decision Level HU | Output Code IHN | Output Code IHP |
|---|---|---|---|---|
| 1 | 00 | 564 | 01 | 11 |
| 2 | 564 | ∞ | 00 | 10 |

## 2.3 G.722 ADPCM Decoders

The G.722 lower and higher band decoders are represented below in Figure 2-6(a) and (b) respectively. From an implementation point of view the modular representation of the decoders given in Figure 2-7 is more illustrative. It can be seen that some code modules are duplicates of the ones implemented in the encoder sections of the G.722 algorithm, i.e., blocks 2L, 3L, 4L, 2H, 3H and 4H. These blocks comprise the Signal Prediction portion of the G.722 decoders and as such are described in section 2-3.

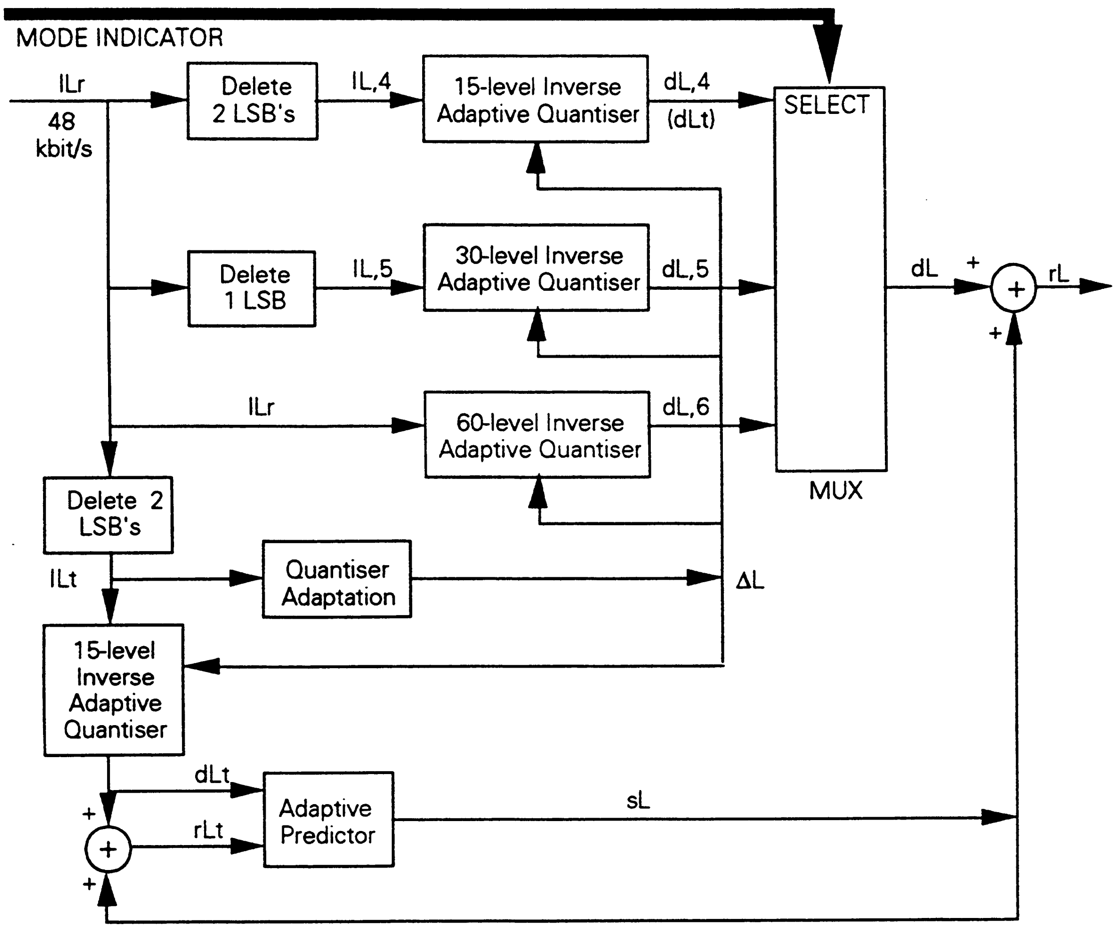



**Figure 2-6** (a) Lower Band G.722 Decoder

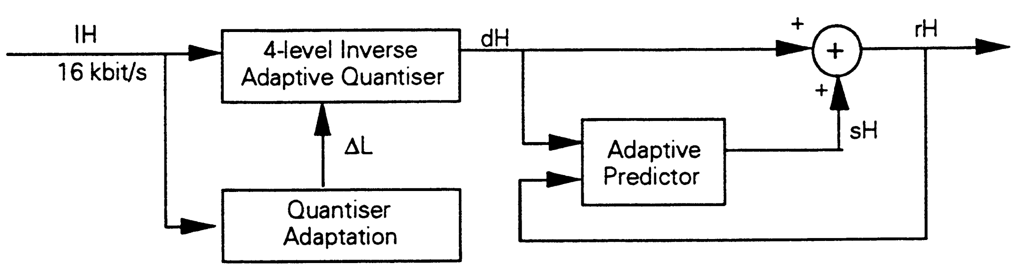



**Figure 2-6** (b) Higher Band G.722 Decoder

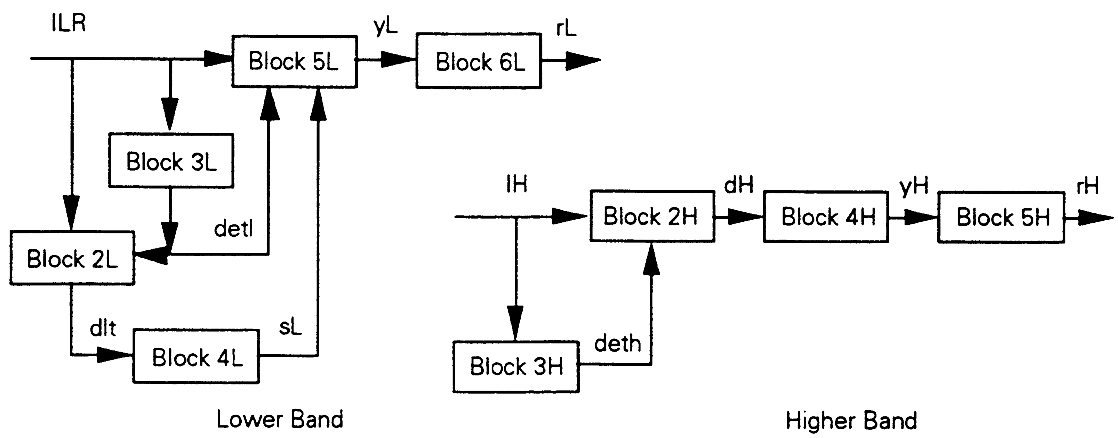



**Figure 2-7** G.722 Decoders in modular form

In Figure 2-7, block 5L represents the inverse quantisation (mode dependent) and signal reconstruction processes in the lower band. Block 5H represents the inverse quantisation and signal limiting processes for the higher band decoder and block 6L performs the signal limiting or saturation process of the lower band decoder that restricts the output signals to 14 bits linearly quantised. The outputs from these blocks are then fed to the receive QMF interpolation filter.

In blocks 5L and 5H the inverse quantisation processes produce the difference signals corresponding to the G.722 encoder signals eL(n) and eH(n). These signals, when added to their respective signal predictor outputs, sL(n-1) and sH(n-1), should produce the original QMF transmit filter outputs from the G.722 transmitter. The relationships between the received IL and IH codes and their corresponding inverse quantisation levels are given below in equations 19 and 20.

$$dL(n) = QL6^{-1}( ILr(n) ) * \Delta L(n) * sign( ILr(n) ) \quad (19)$$

$$dH(n) = Q2^{-1}( IH(n) ) * \Delta H(n) * sign( IH(n) ) \quad (20)$$

Note that here $QL6^{-1}$ and ILr(n) represent the case for mode 1 or 64 kbit/s speech. In the case of mode 2 these will be replaced by $QL5^{-1}$ and IL,5(n), while mode 3 utilises $QL4^{-1}$ and IL,4(n).

These reconstructed difference signals are related to the reconstructed output signals by the following equations 21 and 22.

$$rL(n) = sL(n-1) + dL(n) \quad (21)$$

$$rH(n) = sH(n-1) + dH(n) \quad (22)$$

where:

| | | |
|---|---|---|
| rX(n) | = | Current 8 kHz sampling interval reconstructed output signals. |
| sX(n-1) | = | Previous 8 kHz sampling interval signal predictions. |
| dX(n) | = | Current 8 kHz sampling interval reconstructed difference signals. |

From Figure 2-6(a) it can be seen that the decoder mode is dependent upon the mode signal received from the transmitting terminal. It is important to realise that the decoder mode of operation can only be changed via reception of a mode change signal from the transmitting terminal, and that this signal requires the implementation of separate communication protocols (such as the CCITT's H.221 and G.725) before any such mode change can take place. If the G.722 algorithm is used alone as a speech codec without such associated protocols, then the algorithm will be locked into operation in mode 1 or 64 kbit/s speech.

Once the mode of operation has been determined, the appropriate number of lsb's are deleted from the received 6 lower band bits before entering the appropriate decoder section, as indicated in Figure 2-6(a).

In the G.722 implementation presented, the inverse quantiser arrays have been rearranged to allow a more structured approach to inverse quantisation. The arrays are ordered such that the received lower and higher band words represent the *offsets* from the array base address. This rule also applies for each mode of operation of the lower band inverse quantisation process. During the array search the possibility of transmission errors is accounted for by organising the arrays in a manner which causes any invalid received codeword to select appropriate output values. These have been arranged to be the minimum selectable values for each G.722 mode or corresponding to a received codeword of all 1's (see Table 2-4 Inverse Quantisation Codes and Levels).

The inverse quantiser arrays contain the G.722 specified values which have been pre-multiplied by 8 in order to perform the left shift scaling by 3 places as specified in the inverse quantiser routines 'Invqal', 'Invqbl' and 'Invqah'.

In the lower band decoder two separate inverse quantisation processes take place. The first, 'Invqah', which is located within the main decoder routine, involves the reconstruction of the lower band signal from the received codeword and which is dependent upon the G.722 mode of operation. The second, 'Invqbl', involves the reconstruction of the signal estimate from the 4 msb bits of the 6 received lower band bits and is located within the signal prediction subroutine. The latter is required to maintain consistent decoder signal prediction performance even during G.722 mode 3 communication.

In the higher band decoder the inverse quantisation process takes place within the signal prediction routine and is subsequently stored in the variable location pointed to by the symbol combination x:(dat_hsbdec+dlt0). This variable is recalled after exit from the signal prediction routine, 'Pred_h', for higher band signal reconstruction.

Once reconstructed, the lower and higher band signals are stored in the variables 'yl_dec' and 'yh_dec' prior to their use in the receive QMF filter routine.

The decoder routine assembly code has been included as section 4 of Appendix A.

**Table 2-5** Inverse Quantisation Codes and Associated Levels

| Received Codewords | | | | Inverse Quantisation Level | | | |
|---|---|---|---|---|---|---|---|
| QQ6 | QQ5 | QQ4 | QQ2 | Level 6 | Level 5 | Level 4 | Level 2 |
| 000000 | 00000 | 0000 | 00 | -17 | -35 | 00 | -926 |
| 000001 | 00001 | 0001 | 01 | -17 | -35 | -2557 | -202 |
| 000010 | 00010 | 0010 | 10 | -17 | -2919 | -1612 | 926 |
| 000011 | 00011 | 0011 | 11 | -17 | -2195 | -1121 | 202 |
| 000100 | 00100 | 0100 | | -3101 | -1765 | -786 | |
| 000101 | 00101 | 0101 | | -2738 | -1458 | -530 | |
| 000110 | 00110 | 0110 | | -2376 | -1219 | -323 | |
| 000111 | 00111 | 0111 | | -2088 | -1023 | -150 | |
| 001000 | 01000 | 1000 | | -1873 | -858 | 2557 | |
| 001001 | 01001 | 1001 | | -1689 | -714 | 1612 | |
| 001010 | 01010 | 1010 | | -1535 | -587 | 1121 | |
| 001011 | 01011 | 1011 | | -1399 | -473 | 786 | |
| 001100 | 01100 | 1100 | | -1279 | -370 | 530 | |
| 001101 | 01101 | 1101 | | -1170 | -276 | 323 | |
| 001110 | 01110 | 1110 | | -1072 | -190 | 150 | |
| 001111 | 01111 | 1111 | | -982 | -110 | 00 | |
| 010000 | 10000 | | | -899 | 2919 | | |
| 010001 | 10001 | | | -822 | 2195 | | |
| 010010 | 10010 | | | -750 | 1765 | | |
| 010011 | 10011 | | | -682 | 1458 | | |
| 010100 | 10100 | | | -618 | 1219 | | |
| 010101 | 10101 | | | -558 | 1023 | | |
| 010110 | 10110 | | | -501 | 858 | | |
| 010111 | 10111 | | | -447 | 714 | | |
| 011000 | 11000 | | | -396 | 587 | | |
| 011001 | 11001 | | | -347 | 473 | | |
| 011010 | 11010 | | | -300 | 370 | | |
| 011011 | 11011 | | | -254 | 276 | | |
| 011100 | 11100 | | | -211 | 190 | | |
| 011101 | 11101 | | | -170 | 110 | | |
| 011110 | 11110 | | | -130 | 35 | | |
| 011111 | 11111 | | | -91 | -35 | | |
| 100000 | | | | 3101 | | | |
| 100001 | | | | 2738 | | | |
| 100010 | | | | 2376 | | | |
| 100011 | | | | 2088 | | | |
| 100100 | | | | 1873 | | | |
| 100101 | | | | 1689 | | | |
| 100110 | | | | 1535 | | | |
| 100111 | | | | 1399 | | | |
| 101000 | | | | 1279 | | | |

N.B. Underlined codewords represent invalid received combinations.

**Table 2-5** Inverse Quantisation Codes and Associated Levels

| Received Codewords | | | | Inverse Quantisation Level | | | |
|---|---|---|---|---|---|---|---|
| QQ6 | QQ5 | QQ4 | QQ2 | Level 6 | Level 5 | Level 4 | Level 2 |
| 101001 | | | | 1170 | | | |
| 101010 | | | | 1072 | | | |
| 101011 | | | | 982 | | | |
| 101100 | | | | 899 | | | |
| 101101 | | | | 822 | | | |
| 101110 | | | | 750 | | | |
| 101111 | | | | 682 | | | |
| 110000 | | | | 618 | | | |
| 110001 | | | | 558 | | | |
| 110010 | | | | 501 | | | |
| 110011 | | | | 447 | | | |
| 110100 | | | | 396 | | | |
| 110101 | | | | 347 | | | |
| 110110 | | | | 300 | | | |
| 110111 | | | | 254 | | | |
| 111000 | | | | 211 | | | |
| 111001 | | | | 170 | | | |
| 111010 | | | | 130 | | | |
| 111011 | | | | 91 | | | |
| 111100 | | | | 54 | | | |
| 111101 | | | | 17 | | | |
| 111110 | | | | -54 | | | |
| 111111 | | | | -17 | | | |

N.B. Underlined codewords represent invalid received combinations.

## 2.4 G.722 Signal Prediction

The signal prediction section is the most computationally intensive portion of the G.722 algorithm. This portion of the application note describes how the adaptive scaling factors and signal prediction filter coefficients are updated within the DSP processor. The theory behind this section is not covered in this report for reasons of simplicity but for a more detailed description of these techniques, reference may be made to [10][11].

As the processes within these routines are common to both higher and lower band encoder and decoder, a code kernel was written that satisfied the needs for each. As a result certain restrictions have been placed upon the entry requirements of certain registers when entering the predictor routines. These restrictions are detailed in the comment header associated with the predictor code (see Appendix A, section 6). When exiting the predictor routine the signal prediction value is contained in the 'a' accumulator and should be stored in its relevant location before continuing in the calling routine. The particular variable structure required for correct operation of the signal prediction scheme is indicated in Appendix A, section 5. This structure details the relevant offsets required from the base address pointer which is passed to the signal predictor routine in the 'r2' address register from the calling subroutine.

Upon entry into the predictor routines – 'Pred_l' for the lower band and 'Pred_h' for the higher band – the first process encountered is the inverse quantisation of the lower or higher band transmission code for use in the predictor routines. This process is performed in the subroutines 'Invqal' and 'Invqah' for the lower and higher bands respectively. As previously described, this process uses 4 bits from the lower band quantiser code and 2 bits from the higher band quantiser code and the reconstructed difference signals are stored in memory for subsequent use.

Next, the adaptive scale factors are updated as per the routine descriptions given for 'Logscl' and 'Logsch' and 'Scalel' and Scaleh' in the G.722 Specification. A block schematic diagram for this implementation is shown below in Figure 2-8.

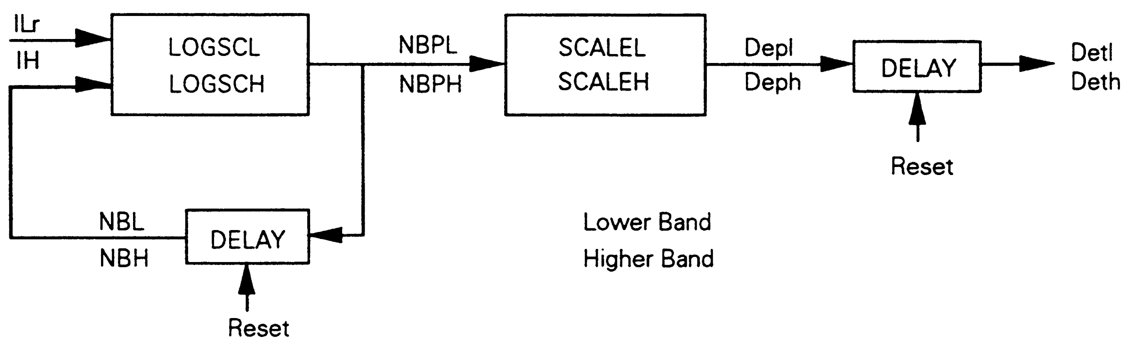



**Figure 2-8** Scale Factor Adaptation in Lower and Higher Bands for both Encoder and Decoder

The scaling factors are updated in the log domain and then subsequently converted to a linear representation. The log-domain scale factors are updated according to equations 23 and 24 and the linear domain scale factors are then updated using these new log-domain factors. To ensure that the scaled signals remain within the bounds of 16-bit arithmetic, the log-domain scaling factors are limited to the ranges specified by equations 25 and 26. The order of processing that takes place therefore follows that of the equations, i.e., 23 through 28 in the appropriate bands.

$$p\_nbl\_L(n) = B * p\_nbl\_L(n-1) + wL( ILt(n-1) ) \quad (23)$$

$$p\_nbl\_H(n) = B * p\_nbl\_H(n-1) + wH( IH(n-1) ) \quad (24)$$

where:

B = leakage factor equal to $^{127}/_{128}$ scaled by $2^{15}$ to 32512 and,

wX = logarithmic scaling factor multipliers corresponding to the received codewords.

$$0 \leq p\_nbl\_L(n) \leq 9 \quad (25)$$

$$0 \leq p\_nbl\_H(n) \leq 11 \quad (26)$$

Finally, the linear scaling factors are calculated from;

$$\Delta L(n) = 2 * ( p\_nbl\_L(n) + 2 ) \Delta_{min} \quad (27)$$

$$\Delta H(n) = 2 * p\_nbl\_H(n) * \Delta_{min} \quad (28)$$

where $\Delta_{min}$ equals half the quantiser step size of a 14-bit A/D converter.

As described in the implementing equations within the G.722 specification, there are two possible methods for performing the linear scale factor updates in the routines 'Scalel' and 'Scaleh'. The difference between the two methods lies in the memory usage and processing complexity of the solutions. In the first method the updated log-domain factor is indirectly used to select the linear scale factor from a table of 353 values, all of which would have to reside in the DSP's memory map. The advantage of this method lies in the speed and simplicity of processing required to implement the search and subsequent manipulation.

The second method employs a 32 value table but requires extra processing to obtain the desired linear scale factor. For the G.722 implementation presented here the second method was chosen as the instruction set of the DSP56156 processor allowed a straightforward implementation of the extra processing necessary by using the multiple left and right shift and repeat instructions (see G.722 spec. [1] pp 282).

Whichever of the above methods is used, the logarithmic scale factor multipliers are chosen corresponding to the quantiser output codes. Table 2-5 shows the relationships between the log multiplication factors, w4 and w2 (lower and higher bands respectively) and the quantiser codewords for the scale factor adaptation process.

**Table 2-6** Relationship between Received Codewords and Log Multiplication Factor

| Received Codewords | | Log Multiplication Factor | |
|---|---|---|---|
| QQ4 | QQ2 | w4 | w2 |
| <u>0000</u> | 00 | -60 | 798 |
| 0001 | 01 | 3042 | -214 |
| 0010 | 10 | 1198 | 798 |
| 0011 | 11 | 538 | -214 |
| 0100 | | 334 | |
| 0101 | | 172 | |
| 0110 | | 58 | |
| 0111 | | -30 | |
| 1000 | | 3042 | |
| 1001 | | 1198 | |
| 1010 | | 534 | |
| 1011 | | 334 | |
| 1100 | | 172 | |
| 1101 | | 58 | |
| 1110 | | -30 | |
| 1111 | | -60 | |

N.B. Underlined codewords represent invalid received combinations.

Upon completion of the inverse quantisation and scale factor adaptation processes the main signal predictor routines follow. A block schematic of the signal predictor processing is shown in Figure 2-9.

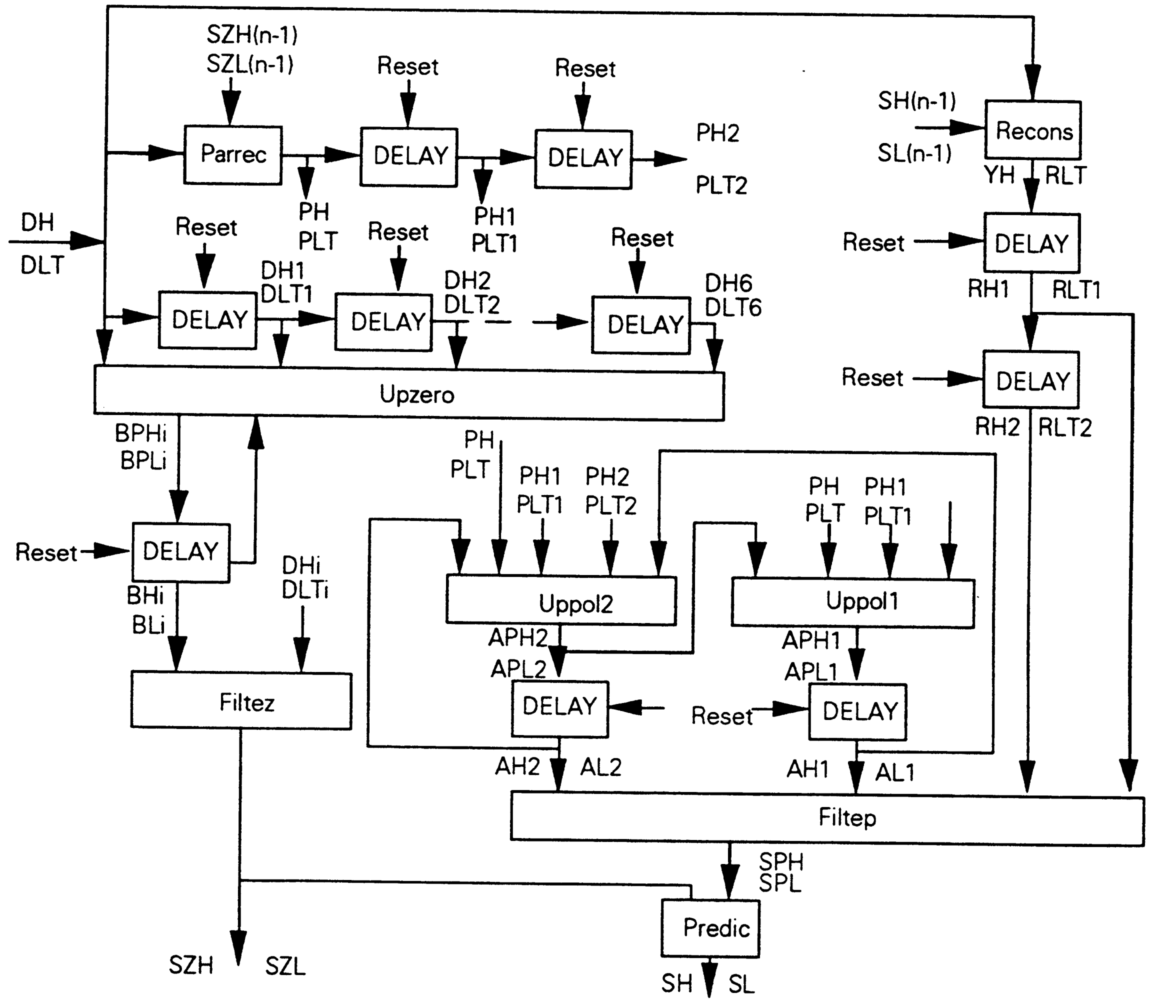



**Figure 2-9** G.722 Signal Prediction Process showing Higher and Lower Band Signals

The adaptive signal prediction routines comprise of two main sections: a second order section that models poles in the input signal, and a sixth order section that models zeros in the input signal. Again, for simplicity the theory behind the development of the routines is not touched upon in this application note. This report concerns itself quite simply with the implementation of the G.722 adaptive signal prediction routines on the DSP56100 core.

Equations 29 through 34 perform the pole and zero signal predictions and identify two-tap and six-tap FIR filter structures respectively. The two outputs from these filters are subsequently mixed (added) to generate the reconstructed difference signals sL(n) and sH(n). These signals are then added to the inversely quantised difference signals from their respective bands in the next 8 kHz sampling interval to produce the decoded output signals rL(n) and rH(n).

The pole predictor signals are calculated according to equations 29 and 30 for the two bands.

$$SPL(n) = \sum_{i=1}^{2} aLi((n-1)*rLt)(n-i \qquad (29)$$

$$SPH(n) = \sum_{i=1}^{2} aHi((n-1)*rH)(n-i \qquad (30)$$

where:

rLt(x) = 6-bit quantiser codeword truncated to 4 bits.

The zero predictor filter produces outputs according to the following equations 31 and 32;

$$SZL(n) = \sum_{i=1}^{6} bLi((n-1)*dLt)(n-i \qquad (31)$$

$$SZH(n) = \sum_{i=1}^{6} bHi((n-i)*dH(n-i) \qquad (32)$$

From the outputs of these two filters the partially reconstructed signals sL(n) and sH(n) are generated according to equations 33 and 34.

$$sL(n) = SPL(n) + SZL(n) \qquad (33)$$

$$sH(n) = SPH(n) + SZH(n) \qquad (34)$$

The updated signal predictions produced according to equations 33 and 34 are subsequently stored for use in reconstructing the signal received in the next 8 kHz sampling interval.

The first procedures in the signal prediction subroutine are the 'Upzero' adaptive filter coefficient (bL1-bL6) and differential signal delay line (dLt1-dLt6) updates. It should be noted here that the differential delay line variables dLt1-dLt6, are stored as twice their reconstructed values. This is purposely done in order to simplify the operation of the 'Filtez' routine later in the code.

The equations governing the operation of the 'Filtez' filter coefficient updates in the 'Upzero' routine are given below in equations 35 to 39. The filter coefficient updating procedure follows a simplified gradient algorithm.

$$bLi = (1-2^{-8}) * bLi(n-1) + 2^{-7} * sign3(dLt(n)) * sign2(dLt(n-i)) \qquad (35)$$

$$bHi = (1-2^{-8}) * bHi(n-1) + 2^{-7} * sign3(dH(n)) * sign2(dH(n-i)) \qquad (36)$$

where:

i = 1 to 6, and bLi and bHi are implicitly limited to ± 2

$$sign2(q) = \begin{cases} +1, & q \geq 0 \\ -1, & q < 0 \end{cases} \qquad (37)$$

$$\text{sign3}(q) = \begin{cases} +1, & q > 0 \\ 0, & q = 0 \\ -1, & q < 0 \end{cases} \quad (38)$$

Once the zero predictor filter coefficients have been updated the next procedure is the update of the partially reconstructed PLT, PLT0, PLT1 and RLT0 signals associated with the pole predictor filter coefficient updates. The RLT delay line values are multiplied by 2 before storage in order to simplify the 'Filtep' procedure.

The next operation updates the pole predictor filter coefficients according to the processes specified by the routines 'Uppol1' and 'Uppol2'. The procedure names, 'Uppol1' and 'Uppol2', represent the pole predictor coefficient updates corresponding to aL1 and aL2 respectively. The procedures governing the operation of these routines are given below in equations 39 to 49 and again follows a simplified gradient algorithm.

$$PLT(n) = DLT(n) + SZL(n-1) \quad (39)$$

$$PH(n) = DH(n) + SZH(n-1) \quad (40)$$

$$aL1(n) = (1-2^{-8}) * aL1(n-1) + 3 * 2^{-8} * PA \quad (41)$$

$$aH1(n) = (1-2^{-8}) * aH1(n-1) + 3 * 2^{-8} * PA \quad (42)$$

$$aL2(n) = (1-2^{-7}) * aL2(n-1) + 2^{-7} * PB - 2^{-7} * f * PA \quad (43)$$

$$aH2(n) = (1-2^{-7}) * aH2(n-1) + 2^{-7} * PB - 2^{-7} * f * PA \quad (44)$$

where:

$$PA = \text{sign2}(pX(n)) * \text{sign2}(pX(n-1)) \quad (45)$$

$$PB = \text{sign2}(pX(n)) * \text{sign2}(pX(n-2)) \quad (46)$$

$$f = \begin{cases} 4 * aX1(n-1) & |aX1| \le 1/2 \\ 2 * \text{sign}(aX1(n-1)) & |aX1| > 1/2 \end{cases} \quad (47)$$

pX = pLt or pH dependent upon the sub-band being processed.

aX = aL or aH dependent upon the sub-band being processed.

sign2(q) = see equation 37

In order to maintain stability, the two pole predictor coefficients are limited to the following ;

$$|aX2| \le 0.75 \quad (48)$$

$$|aX1| \le 1 - 2^{-4} - aX2 \quad (49)$$

Once these coefficients have been updated the prediction filter routines are executed according to equations 29 to 34. The program flow then returns to the calling G.722 procedure with the reconstructed signal, sL(n) or sH(n), in accumulator 'a'.

The assembly code for the signal prediction portion of the G.722 algorithm is provided in Appendix A, section 6.

# SECTION 3
## G.722 CODE INITIALISATION AND TESTING

To ensure correct operation of any G.722 code implementation the CCITT have prepared a set of test vector files that the code must process correctly. These test vector files comprise files for testing encoder and decoder separately. For testing purposes, and to simplify the production of test vector files, the vectors passed into the decoder for processing make up the files that are used to compare the outputs from the encoder after processing of its own test vectors. The construction and use of the data contained in these files is detailed in the G.722 specification [1] pp 304-318. When passing the test vectors during code development, the QMF filters are bypassed and should be tested independently of the encoder and decoder code sections. This may be achieved by passing tones of varying frequencies within the bandwidth of interest (0-7 kHz) through the filters connected back to back.

The G.722 code as it stands has passed all available CCITT test vectors satisfactorily. When the code is shipped, two versions of the G.722 code are provided on the disk:

1) the version required to pass the CCITT test vectors, i.e., minus the QMF filters; and

2) the complete version including QMF filters and interface to one of the 56156's SSI's for speech sampling at 16 kHz.

The directory structure on the disk provided may be used to pass all the test vectors. In the top level directory there is a help file that gives a comprehensive user's guide to passing the test vectors using a 56156 Application Development System.

Certain internal variables within the G.722 code require initialisation to specific values to ensure correct operation of the algorithm and to pass the test vectors. Table 3-1 shows these variables and the values to which they should initially be set. The table also includes variables that do not require initialisation but that have been reset anyway. The code sections that perform the initialisation are provided in Appendix A, section 7 figures (a),(b),(c) and (d). These initial values are valid for variables in both the higher and lower band encoders and decoders.

As the G.722 code has been written to enable bootstrap from EPROM, all the storage constants within the DSP X memory that are used in the algorithm (run-time) are pre-loaded with their relevant values on power-up and hardware reset of the DSP (load-time). This includes all the variables indicated in Table 3-1 and more.

**Table 3-1** Variables requiring initialisation and the associated values

| Variable | Description | Initialised Value | |
|---|---|---|---|
| detl | Signal scaling factor in lower band | 32 | * |
| deth | Signal scaling factor in higher band | 8 | * |
| sL,sH | Lower and Higher band signal predictions | 0 | |
| SZL,SZH | Reconstructed signals from the Zero predictor | 0 | |
| p_nbl | The last output of the quantiser signal scale factor adaptation routine (NBL in G.722 spec.) | 0 | * |
| aL1,aL2, aH1,aH2 | The Pole predictor filter coefficients | 0 | * |
| bL1..bL6, bH1..bH6 | The Zero predictor filter coefficients | 0 | * |
| rLt0,rLt1, rH0,rH1 | Adaptive predictor reconstructed signal delay lines | 0 | * |
| dLt0..dLt6, dH0..dH6 | Zero predictor quantised difference signal delay lines | 0 | * |
| PLT0, PLT1, PH0,PH1 | Partially reconstructed Pole predictor signal delay line | 0 | * |

* = Variables to be initialised as specified by the G.722 specification.

Some points to note regarding the code initialisation procedures are:

a) the delay line buffers associated with the transmit and receive QMF filters are reset to zero and the modulo addressing pointers associated with each filter are set to the address of their respective first buffer locations in the routines 'init_q_tx' and 'init_q_rx';

b) the bootstrapping of G.722 constants from program memory into X data memory is performed in the routine 'init_const'; and

c) the initialisation of the encoder and decoder internal variables as detailed in Table 3-1 is performed in the routines 'reset_cod' and 'reset_dec' respectively.

The G.722 encoder and decoder software provided in this report have both passed all the available test vector files provided by the CCITT. When testing the G.722 algorithm both acoustically and for passing the test vectors, the mode of operation of the G.722 algorithm must be read in from an external file that resides in the host computer. This is because the mode of operation of the G.722 code in an end application can only be changed with the use of auxiliary communications protocols.

The DSP56156 X data and program memory map structures are shown in Figure 3-1 below. The internal RAM of the 56156 comprises 2K data and 2K program words and, as indicated in Figure 3-1, the complete G.722 software implementation fits within the internal memory space of the device with room to spare.

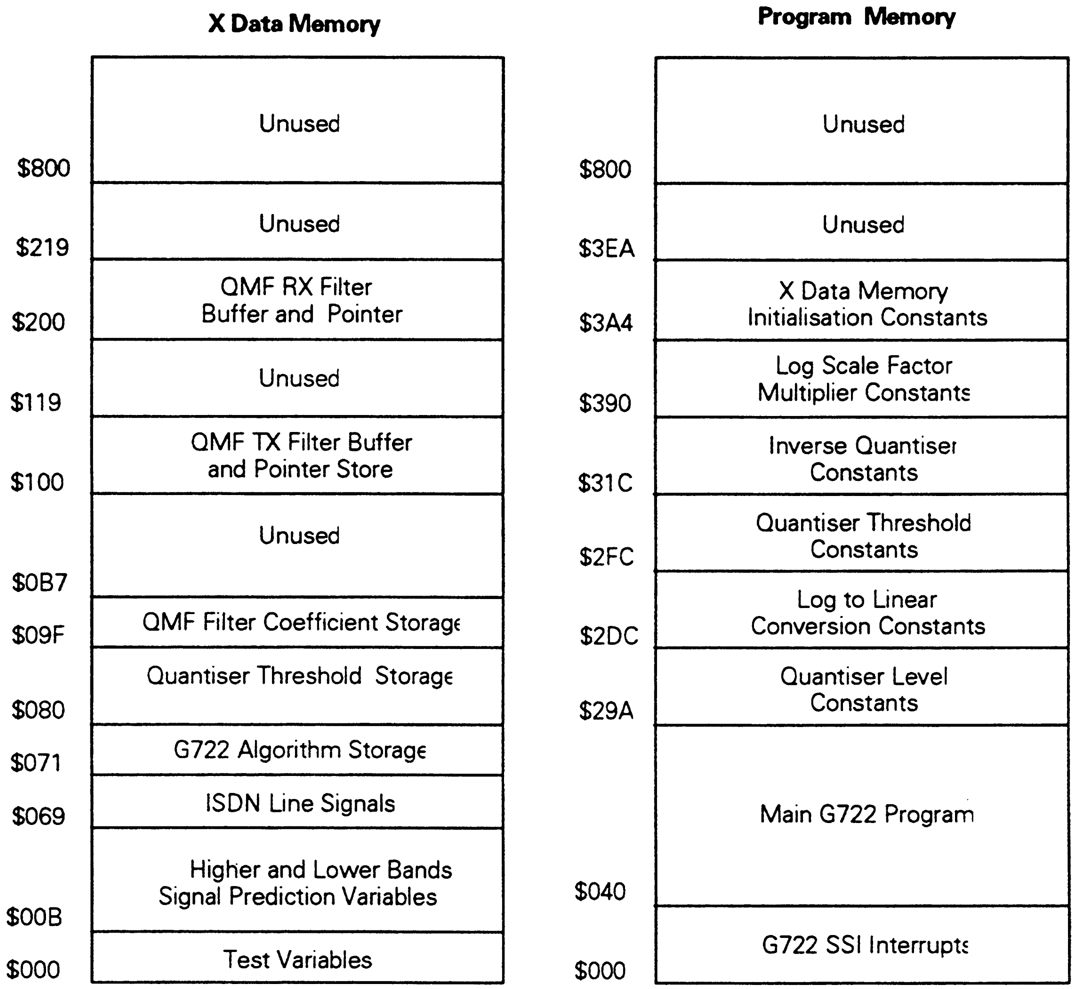



**Figure 3-1** Internal Data RAM Memory Map

# SECTION 4
## PERFORMANCE SPECIFICATIONS

The G.722 software as it stands uses 1001 words of program memory and 232 words of X data memory and further optimisation of the code is possible. The worst case execution times in instruction cycles of the G.722 algorithm modules are given below in Table 4-1. As the G.722 algorithm only needs to execute either the transmit QMF and G.722 encoder *or* the G.722 decoder and the receive QMF in a single 16 kHz sampling period (due to the decimation of the 16 kHz A/D/A sampling frequency to the ISDN line frequency of 8 kHz) the performance required from the processor is the worst case of these two scenarios. The number of instructions per second (IPS) required from the DSP is given by equation 50 and from this the performance required in order to provide full-duplex operation is shown in Table 4-2.

**Table 4-1** Execution Times of the G.722 Sections

| Code Section | Instruction Cycles |
|---|---|
| Transmit QMF Filter | 64 |
| G.722 Encoder | 116 |
| G.722 Decoder | 69 |
| Receive QMF Filter | 64 |
| Lower OR Higher Band Predictor | 204 |

The MIPS performance can be calculated as follows:

$$\text{Millions of Instruction Cycles Per Second (MIPS)} = \text{Instruction Cycles} * \text{Sampling Frequency} \div 10^6 \qquad (50)$$

**Table 4-2** MIPS Requirements of the G.722 Code

| Code Section | Processing Frequency | Instruction Cycles | MIPS |
|---|---|---|---|
| Transmit QMF Filter | (a) 16 kHz | 64 | 1.03 |
| G.722 Encoder | (b) | 116 | 1.86 |
| G.722 Decoder | (c) | 69 | 1.10 |
| Receive QMF Filter | (d) | 64 | 1.03 |
| Lower OR Higher Band Predictor | (e) | 204 | 3.27 |
| Effective G.722 Transmit (= a + b + 2 * e) | | 588 | 9.41 |
| Effective G.722 Receive (= c + d + 2 * e) | | 541 | 8.66 |

From Table 4-2 it can be seen that the peak performance requirement of the DSP processor is 9.41 MIPS. These figures include the instruction cycles incurred by the subroutine jump instructions in the main program loop.

# SECTION 5
# RECENT TRENDS IN SPEECH CODING TECHNIQUES

In recent years, with lower-cost Digital Signal Processors, Mobile Communications and a desire for better quality speech transmission, the race has been on to develop algorithms that compress speech into ever lower bit rates but with increasing bandwidth and quality.

Many speech coding techniques have been developed based upon both time-domain and frequency-domain processing and latterly the trend has been towards psycho-acoustic coding with algorithms such as OCF (Optimum Coding in the Frequency domain) [12]. This, as its name suggests, means that coding techniques are now being developed which base their design upon the acoustic response of the human auditory system. For example, baseband speech signals are broken down into 'critical bands' using complex filter banks the outputs of which are allocated different bit rates dependent upon where the highest spectral density lies within the speech spectrum. New subjective measurement techniques are also being developed that gear their operation to the responses of the human auditory system and may result in the displacement of the usual Signal to Noise Ratio (SNR) measurement for perceived quality. The Noise to Mask Ratio (NMR) metric [13] is currently being developed based upon the audibility of error signals according to the laws of psycho-acoustics. At present, NMR measurements cannot substitute completely for listening tests but they can deliver objective, reproducible results, helping to highlight critical pieces of music and the weaknesses of the algorithms being evaluated. Psycho-acoustic algorithms are however, extremely computationally intensive and at present most require multiple processors to execute in real time.

With the imminent arrival of Pan-European Mobile Communication standards such as the GSM (Group Special Mobile) 06.10 [14], and with the limited availability of frequency airspace, the need to reduce bit rates to make the systems feasible is of paramount importance. Subsequently, the quality of the speech encoding algorithms must reflect this demand and evolve, thus consideration is now being given to speech data transmission rates of 6.5 kbit/s and less.

The G.722 specification [1] has been in existence for a number of years and its relative simplicity combined with increased speech bandwidth and quality make it a good choice for systems with limited processing power but which require good quality speech combined with 8 and 16 kbit/s data channels. As the arrival of the ISDN (Integrated Systems Digital Network) becomes more real this ability to compress good quality speech and data into a single 64 kbit/s channel makes G.722 the ideal choice for low cost ISDN terminals.

# SECTION 6
# REFERENCES

## 6.1 Introduction

[1] CCITT, "7 kHz Audio Coding within 64 kbit/s", *Recommendation G.722*, Fascicle 111.3, Red book. 6.

[2] Motorola Ltd, DSP56116, Digital Signal Processor User's Manual, *Motorola Technical Publications*, DSP56116UM/AD

## 6.2 The G.722 Algorithm : Quadrature Mirror Filtering

[3] D. Esteban and C. Galand, "Application of Quadrature Mirror Filters to Split Band Voice Coding Schemes", *IEEE Int. Conf. on Acoustics, Speech, Signal Proc.*, pp. 191-199, 1977

[4] Vijay K. Jain and Ronald E. Crochiere, "Quadrature Mirror Filter Design in the Time Domain", *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. ASSP-32, pp. 353-361, April 1984

[5] M.J.T. Smith and T.P. Barnwell, "A Procedure for Designing Exact Reconstruction Filter Banks for Tree-Structured Subband Coders", *Proc. IEEE ICASSP-84*, San Diego, California, March 1984, pp. 27.1.1-4.

[6] P.P. Vaidyanathan and Kumar Swaminathan, "Alias-Free, Real Coefficient *m*-Band QMF Banks for Arbitrary *m*" *IEEE Trans. on Circuits and Systems*, vol. CAS-34, Nº 12, pp. 1485-1496, December 1987.

[7] Martin Vetterli, "A Theory of Multirate Filter Banks", *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. ASSP-35, Nº 3, pp. 356-372, March 1987.

[8] P.P. Vaidyanathan and Phuong-Quan Hoang, "Lattice Structures for Optimal Design and Robust Implementation of Two-Channel Perfect Reconstruction QMF Banks", *IEEE Trans. Acoustics, Speech, Signal Processing*, vol. ASSP-36, Nº 1, pp. 81-94, January 1988.

[9] R.E. Crochiere, S.A. Webber and J.L. Flanagan, "Digital Coding of Speech in Sub-Bands", *Bell Systems Technical Journal*, vol. 55, pp. 1069-1085, October 1976

## 6.3 The G.722 Algorithm : G.722 ADPCM Encoding

[10] Kent Terry, "Full-duplex 32 kbit/s CCITT ADPCM Speech coding on the Motorola DSP56001" *Application Note APRD/9*, 1990.

[11] P. Cummiskey, N.S. Jayant, and J.L. Flanagan, "Adaptive Quantisation in Differential PCM Encoding of Speech" *B.S.T.J. vol. 52 number 7*, September 1973.

## 6.4 Recent Trends in Speech Coding Techniques


[12] K. Brandenburg, "OCF - "A new coding algorithm for high quality sound signals" *presented at ICASSP,* 1987, Dallas.

[13] K. Brandenburg, "Evaluation of quality for audio encoding at low bit rates" *presented at 82 nd AES convention, London,* March 10-13, 1987.

[14] ETSI/GSM, Speech Coding for mobile communications, *Recommendation* GSM 06.10

# APPENDIX A
# ASSEMBLY CODE SECTIONS

## A.1. QMF Transmit Filter Code on the DSP56156

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;      Subroutine qmf_tx: transmit QMF filter operating at 8 kHz
;;      input two samples at 16 kHz and output a low_band value and
;;      a high_band value
;;      The input values are x1_in and x0_in (the more recent is x0)
;;      The output values are xl_cod and xh_cod
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
qmf_tx          move    x:ptr_q_tx,r0; recall pointer
                move    m0,x:ptr_q_tx; save m0 value
                move    #23,m0    ; modulo 24 for the delay line
                move    #q_coef,r3; for QMF TX and RX coefficients
;;
;;      Read the two values from ADC converter and scale them
;;      ADC is supposed to be 16 bits  (left justfified,14 bits precison)
;;      ================================================
;;
                move    x:x1_in,b               ; ADC sample (N-1)
                move    x:x0_in,a               ; ADC sample (N)
                move    b,x:(r0)+               ; save x1 in modulo delay line
                move    a,x:(r0)+               ; save x0 in modulo delay line
                                                ; r0 points on H[23]
;;
;;      Begin mac operation: ACCUMA in a and ACCUMB in b ( See G.722 spec. for ACCUMA and ACCUMB )
;;      ========================================================================
;;
                clr     a       x:(r0)+,y0      ; read xin[23]
                clr     b       x:(r3)+,x0      ; read h[23]
;;
                do      #12,end_q_tx                    ; for 12 values
                mac     x0,y0,b  x:(r0)+,y1  x:(r3)+,x1 ; mac and read next values
                mac     x1,y1,a  x:(r0)+,y0  x:(r3)+,x0 ; mac and read next values
end_q_tx; end of do loop
;;
;;      Now save updated pointer and end the computation of xl_cod and xh_cod
;;      =====================================================
;;
        tfr     b,x                             ; save 32-bit result in x
        add     a,b     (r0)-                   ; compute xl_cod
                                                ; decrement modulo buffer pointer to correct address
        sub     x,a                             ; compute xh_cod
;;
```

```
;;      Limiting the output values
;;      ===================
;;
                asl     a                               ; times 2
                asr     a                               ; limited to -16384 and +16383
                asl     b                               ; times 2
                asr     b                               ; limited to -16384 and +16383
                move    b,x:xl_cod                      ; for input of lsbcod
                move    a,x:xh_cod                      ; for input of hsbcod
                move    x:ptr_q_tx,m0                   ; recall m0 value
                move    r0,x:ptr_q_tx                   ; save modulo pointer
;;
;;      end of qmf_tx  subroutine
;;      ==================
;;
                rts
;;
```

## A.2. QMF Receive Filter Code on the DSP56156

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      Subroutine for RX QMF filter
;;      Compute two values from yl_dec and yh_dec (outputs of lsbdec and
;;      hsbdec);  the outputs are xout1(n) and xout2 (n+1)
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
qmf_rx          move    x:ptr_q_rx,r0                   ; recall saved pointer
                move    m0,x:ptr_q_rx                   ; save old value of m0
                move    #23,m0                          ; modulo 24 pointer
                move    #q_coef,r3                      ; address of coefficients
;;
;;      Compute XS and XD (RECB and RECA) G.722 spec.
;;      ========================================
;;
                move    x:yl_dec,a                      ; recall yl = rl G.722
                move    x:yh_dec,b                      ; recall yh = rh G.722
                add     a,b         b,y0                ; xs in b, x0 = rh
                sub     y0,a        b,x:(r0)+           ; xd in a, save xs
                move    a,x:(r0)+                       ; save xd in delay line
                                                        ; now r0 points on xs11
;;
;;      Begin mac computation
;;      ==================
;;
                clr a   x:(r0)+,y0                      ; read xs11
                clr b   x:(r3)+,x0                      ; read H[23]
;;
                do      #12,end_q_rx                    ; for 12 values
                mac     x0,y0,b  x:(r0)+,y1  x:(r3)+,x1 ; mac and read next values
                mac     x1,y1,a  x:(r0)+,y0  x:(r3)+,x0 ; mac and read next values
                end_q_rx                                ; end of do loop
;;
;;      Scaling for output
;;      =================
;;
                asl     a           (r0)-               ; times 2
                asl     b                               ; times 2
                asl     a                               ; for coefficient scaling
                asl     a                               ; same
                asl     b                               ; for coefficient scaling
                asl     b                               ; same
                move    b,x:xout2                       ; xout(n)
                move    a,x:xout1                       ; xout(n-1)
                move    x:ptr_q_rx,m0                   ; recall m0 value
                move    r0,x:ptr_q_rx                   ; save modulo pointer
;;
;;      End of subroutine qmf_rx
;;      =====================
;;
        rts
```

## A.3. Encoder Quantisation Routines

Higher & Lower Bands

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;       Encoder: G.722 encoder                                                 ;
;        Compute the output, is, from inputs, xl_cod and xh_cod                ;
;        First compute il_cod from xl_cod (lsbcod procedure)                   ;
;        Then compute ih_cod from xh_cod (hsbcod procedure)                    ;
;       Finally compute is from il_cod and ih_cod                              ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;       Lsbcod : lower sub band coder                                          ;
;        Compute the output code il_cod from input xl_cod                      ;
;        First compute el then quantise on 6 bits                              ;
;                                                                              ;
;       NOTE: entry point of encoder = lsbcod                                  ;
;        ============================                                          ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
encoder         move    x:xl_cod,a                      ; read xl_cod in a
                move    x:(dat_lsbcod+sl),b             ; read prediction
                sub     b,a                             ; compute el in a
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;       Quantl : lower sub band 6 bits quantizer                               ;
;        el in a                                                               ;
;          This procedure uses a mixed tree and direct search                  ;
;          to minimize speed and size of code.                                 ;
;        A full binary search procedure would save 20 cycles                   ;
;        (10 instructions) but at the expense of 100 program words.            ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
quantl          move    #cod_6_mi,b                     ; select table for el <0
                move    #cod_6_pl,x0                    ; select table for el >0
                move    #level_0,r2                     ; offset of table level in ram
                tst     a                               ; test if sign of el <0
                tpl     x0,b                            ; select table >0
                move    b,r0                            ; save table in r0
                tfr     a,b         x:(r2+14),x0        ; level 14 in x0
                move    x:dat_lsbcod,y0                 ; y0 = detl
                inc24   b                               ; to compute lell of G.722
                abs     b           x:(r2+6),x1         ; level 6 in x1
                tst     a                               ; test if a >=0
                tmi     b,a                             ; a = lell = wd
;;
;;      Beginning of the tree search
;;      =====================
;;
test_14         mpy     y0,x0,b x:(r2+22),x0            ; level 22 in x0
                move    b,b                             ; set lsp of b to 0
                cmp     b,a                             ; test wd with level 14
                bpl     <test_22                        ; if >0 go test_22
;
```

```
test_6      mpy     y0,x1,b                         ; level 6 * detl
            move    b,b                             ; set lsp of b to 0
            cmp     b,a                             ; test wd with level 6
            bpl     <init_7                         ; if >0 go to init_7
;
init__1     move    #-1,n0                          ; set init of r0 index to -1
            move    #level__1,r3                    ; set r3 to level__1
            move    x:(r0)+n0,b                     ; dummy read to update r0 to r0-1
            bra     <end_q6                         ; direct branch to end of procedure
;
init_7      move    #7,n0                           ; set init of r0 index to 7
            move    #level_7,r3                     ; set r3 to level_7
            move    x:(r0)+n0,b                     ; dummy read to update r0 to r0+7
            bra     <end_q6                         ; direct branch to end of procedure
            ;
test_22     mpy     y0,x0,b                         ; level 22 * detl
            move    b,b                             ; set lsp to b 0
            cmp     b,a                             ; test wd with level 22
            bpl     <init_23                        ; if >0 go ro init_23
            ;
init_15     move    #15,n0                          ; set init of r0 index to 15
            move    #level_15,r3                    ; set r3 to level_15
            move    x:(r0)+n0,b                     ; dummy read to update r0 to r0+15
            bra     <end_q6                         ; direct branch to end of procedure
            ;
init_23     move    #23,n0                          ; set init of r0 index to 23
            move    #level_23,r3                    ; set r3 to level_23
            move    x:(r0)+n0,b                     ; dummy read to update r0 to r0+23
            bra     <end_q6                         ; direct branch to end of procedure
;
;;
;;      Beginning of direct search for 7 values of index
;;      ================================================
;;
end_q6      move    r0,r1                           ; set r1 to init of r0
            move    x:(r3)+,x0                      ; read level -1,7,15,23
            mpy     y0,x0,b  x:(r3)+,x0             ; read level  0,8,16,24
            move    b,y1                            ; set lsp of b to 0 (x1)
;
            mpy     y0,x0,b x:(r0)+,x1 x:(r3)+,x0   ; r0++, read level 1,9,17,25
            cmp     y1,a        b,y1                ; compare level -1,7,15,23
            tpl     b,b         r0,r1               ; increment r1 if >0
;
            mpy     y0,x0,b x:(r0)+,x1 x:(r3)+,x0   ; r0++, read level 2,10,18,26
            cmp     y1,a        b,y1                ; compare level 0,8,16,24
            tpl     b,b         r0,r1               ; increment r1 if >0
;
            mpy     y0,x0,b x:(r0)+,x1 x:(r3)+,x0   ; r0++, read level 3,11,19,27
            cmp     y1,a        b,y1                ; compare level 1,9,17,25
            tpl     b,b         r0,r1               ; increment r1 if >0
;
            mpy     y0,x0,b x:(r0)+,x1 x:(r3)+,x0   ; r0++, read level 4,12,20,28
            cmp     y1,a        b,y1                ; compare level 2,10,18,26
            tpl     b,b         r0,r1               ; increment r1 if >0
;
            mpy     y0,x0,b x:(r0)+,x1 x:(r3)+,x0   ; r0++, read level 5,13,21,29
            cmp     y1,a        b,y1                ; compare level 3,11,19,27
            tpl     b,b         r0,r1               ; increment r1 if >0
;
```

```
                mpy     y0,x0,b x:(r0)+,x1 x:(r3)+,x0       ; r0++, read level 6,14,22,30
                cmp     y1,a        b,y1                    ; compare level 4,12,20,28
                tpl     b,b         r0,r1                   ; increment r1 if >0
;
                cmp     y1,a        x:(r0)+,x1              ; compare level 5,13,21,29
                tpl     b,b         r0,r1                   ; increment r1 if >0
;
                move    #dat_lsbcod,r2                      ; set offset for lsbcod
                move    p:(r1),a                            ; code il_cod in a
                move    a,x:il_cod                          ; save code for lower sub_band
;;
;;      We must call subroutine pred_l
;;      ============================
;;
                move    #const_pr_l,r3                      ; set constant table for low band
                jsr     pred_l                              ; call subroutine pred_l
                                                            ; with il_cod in a
                move     a,x:(r2+sl)                        ; save sl in lsbcod
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                        ;
;       Hsbcod : higher sub band coder                                   ;
;        Compute the output code ih_cod from input xh_cod                ;
;        First compute eh then quantize on 2 bits                        ;
;                                                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
hsbcod          move    #dat_hsbcod,r2                      ; set offset of data hsbcod
                move    x:xh_cod,b                          ; read xh_cod in b
                move    x:(r2+sl),a                         ; read prediction
                sub     a,b  x:(r2+delt),y0                 ; compute eh in a
;;                                                          ; read deth in y0
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                        ;
;       Quanth : higher sub band 2 bits quantizer                        ;
;           eh in a                                                      ;
;                                                                        ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
quanth          move    #4512,x0                            ; level of quantization
                mpy     y0,x0,a                             ; compute wd,save eh
                tst     b  a,x1                             ; test for sign of eh
                bmi     <cod_hi_mi                          ; if neg bra cod_hi_mi
                move    #3,x0                               ; lower limit
                move    #2,y0                               ; upper limit
                cmp     x1,b  x0,a                          ; set a with lower limit
                tpl     y0,a                                ; if plus =>upper limit
                bra     <end_q2                             ; end of quant_h
;                                                           ; with ih in a
                cod_hi_mi inc24     b
                abs     b                                   ; compute lehl
                move    #1,x0                               ; lower limit
                move    #0,y0                               ; upper limit
                cmp     x1,b  x0,a                          ; set a with lower limit
                tpl     y0,a                                ; ih in a
;
end_q2          move    a,x:ih_cod                          ; save code for higher sub_band
;;
```

```
;;      We must call subroutine pred_h
;;      ==============================
;;
                move    #const_pr_h,r3          ; constant table for high band
jsr             pred_h                          ; call subrouinte pred_h
                                                ; with ih_cod in a
                move    a,x:(r2+sl)             ; save sh in hsbcod
;;
;;      Computation of is code from il_cod and ih_cod
;;      ============================================
;;
                move    x:il_cod,a              ; read il in RAM
                move    x:ih_cod,x0             ; read ih in RAM
                move    #64,y0                  ; for << 6
                imac    y0,x0,a                 ; to compute cod
                move    a,x:is                  ; save is code in RAM
                rts                             ; return of encoder
;;
;;      End of encoder procedure
;;      ========================
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

## A.4. Decoder Routine (Higher & Lower Bands)

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                                                  ;;
;;      Subroutine decoder : compute yl_dec and yh_dec from ir                                       ;;
;;                      First compute ilr_dec and ihr_dec                                            ;;
;;                      Then execute lsbdec and hsbdec                                               ;;
;;                                                                                                  ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
decoder         move    x:ir,a                  ; read receive code ir
                move    #63,x0                  ; set mask for ilr_dec
                move    #3,y0                   ; set mask for ihr_dec
                asr     a       a,b             ; shift a save a in b
                asr4    a                       ; to compute ihr_dec
                asr     a                       ; final shift of 6 shifts
                and     y0,a                    ; mask ihr_dec
                and     x0,b                    ; mask for ilr_dec
                move    b,x:ilr_dec             ; save ilr_dec
                move    a,x:ihr_dec             ; save ihr_dec
;;
;;      lsbdec
;;      ====
;;
;;      Select mode of operation of lower sub band decoder
;;      ==================================
;;
                move    #dat_lsbdec,r2          ; set data ram
                move    #sel_mode,r0            ; load table sel_mode in r0
                move    x:mode,a                ; read mode of decoder
                and     y0,a    a0,x0           ; mask mode bits, x0 == 0 (#3 still in y0)
                dec24   a       b,y1            ; compute modified mode, ilr_dec in y1
                tmi     x0,a                    ; select default mode ==1
;;
;;      read table sel_mode
;;      =============
;;
                rep     a1                      ; repeat 0 1 or 2 times
                asr     b  x:(r0)+,x1           ; shift and dummy read
                move    b,n1                    ; offset for table QQ6,QQ5 or QQ4
                move    x:(r0),r1               ; selected table in r1
                move    x:(r2+sl),b             ; read prediction in b
                move    x:(r1)+n1,x0            ; dummy read to compute r1+n1
                move    x:(r2),y0               ; read detl in ram
                move    p:(r1),x0               ; read table of inverse quantizer
                mac     y0,x0,b  y1,a           ; compute yl, a = ilr_dec
                asl     b                       ; limit yl to 16384
                asr     b                       ; end of limiting
                move    #const_pr_l,r3          ; for lower predictor
                move    b,x:yl_dec              ; save reconstructed signal
;;
;;      call pred_l
;;      ========
;;
jsr             pred_l                          ; lower predictor
                move    a,x:(r2+sl)             ; save next prediction
;;
```

```
;;      hsbdec
;;      ======
;;
                move    #dat_hsbdec,r2          ; select ram
                move    #const_pr_h,r3          ; select higher constant
                move    x:ihr_dec,a             ; read ih in a
;;
;;      call pred_h
;;      ===========
;;
                jsr     pred_h                  ; higher predictor
                move    x:(r2+dlt0),b           ; reconstructed signal
                move    x:(r2+sl),y1            ; last prediction
                add     y1,b  a,x:(r2+sl)       ; compute yh, save new sl
                asl     b                       ; limit yh
                asr     b                       ; end of limiting
                move    b,x:yh_dec              ; save yh_dec
;;
;;      end of decoder
;;      ==============
;;
                rts
;;
```

## A.5. Structure of Variables for G.722 Signal Predictor

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;      structure of variables for predictor in lower sub band coder         ;;
;;                              in higher sub band coder                     ;;
;;                              in lower sub band decoder                    ;;
;;                              in higher sub band decoder                   ;;
;;      the address of this structure is passed to the subroutine predictor;;
;;      in the r2 address register                                           ;;
;;      this structure need 23 words of ram that must be initialized for     ;;
;;      correct opration of the G.722 algorithm (digital test sequences)     ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
delt    equ     0                               ;
sl      equ     1                               ; signal predicted
szl     equ     2                               ; output of the zero predictor
p_nbl   equ     3                               ; nabla of the predictor
al1     equ     4                               ; first pole predictor coefficient
al2     equ     5                               ; first pole predictor coefficient
bl1     equ     6                               ; zero predictor coefficient
bl2     equ     7                               ; zero predictor coefficient
bl3     equ     8                               ; zero predictor coefficient
bl4     equ     9                               ; zero predictor coefficient
bl5     equ     10                              ; zero predictor coefficient
bl6     equ     11                              ; zero predictor coefficient
rlt0    equ     12                              ; pole signal predictor
rlt1    equ     13                              ; pole signal predictor
dlt0    equ     14                              ; zero signal predictor
dlt1    equ     15                              ; zero signal predictor
dlt2    equ     16                              ; zero signal predictor
dlt3    equ     17                              ; zero signal predictor
dlt4    equ     18                              ; zero signal predictor
dlt5    equ     19                              ; zero signal predictor
dlt6    equ     20                              ; zero signal predictor
plt0    equ     21                              ; pole partial signal predictor
plt1    equ     22                              ; pole partial signal predictor
;;
```

## A.6. G.722 Signal Prediction Routine

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                            ;;
;;      Subroutine pred_l: compute invqal, logscl, scalel                     ;;
;;                  then compute the adaptive predictor                       ;;
;;      Input :     il in a (lower subband code)                              ;;
;;                  r3 must point on const_pr_l (constant for lower band)     ;;
;;                  r2 must point on data ram for lower band                  ;;
;;                                                                            ;;
;;      Subroutine pred_h: compute invqah, logsch, scaleh                     ;;
;;                                                                            ;;
;;      Input :     ih in a (higher subband code)                             ;;
;;                  r3 must point on const_pr_h (constant for higher band)    ;;
;;                  r2 must point on data ram for higher band                 ;;
;;                                                                            ;;
;;      NOTE:       pred_l and pred_h are the same but the entry point of     ;;
;;                  pred_h skip the >> 2 of input code                        ;;
;;                                                                            ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      Invaqal/h: inverse quantizer on 4/2 bits
;;      ========================================
;;
;;      assume a = il/ih (level of quantizer)
;;
pred_l          lsr     a
                lsr     a                               ; to compute ilr = il >>2
pred_h          move    a1,n0                           ; offset for table QQ4/QQ2
                move    x:(r3)+,r0                      ; to address table QQ4/QQ2
                move    n0,n1                           ; for table W4/W2
                move    x:(r3)+,r1                      ; to address table W4/W2
                move    x:(r0)+n0,x1                    ; table QQ4/QQ2 (dummy read)
                move    x:(r2),x0                       ; detl =first data in structure
                move    p:(r0),x1                       ; read inverse quantizer output
                mpy     x1,x0,b x:(r1)+n1,x0            ; b=detl*IQ4/IQ2, dummy read->r1+n1
                clr     b   b,x:(r2+dlt0)               ; b=0, save new dlt0 in ram
;;
;;      Begin Logscl/h
;;      ==============
;;
                move    x:(r3)+,x0                      ; x0 =32512
                move    x:(r2+p_nbl),y1                 ; read old p_nbl
                mpy     y1,x0,a  x:(r3)+,y0             ; a= p_nbl*32512; y0=18432/22528)
                move    p:(r1),y1                       ; read table W4/W2
                add     y1,a                            ; compute p_nbl*32512 + wl in a
                tmi     b,a                             ; limit to 0 if < 0
                cmp     y0,a                            ; test if > 18432/22528
                tpl     y0,a                            ; limit to 18432/22528
;;
```

```
;;      Begin Scalel/h
;;      ==============
;;
                asr     a a,x:(r2+p_nbl)            ; save new p_nbl
                asr     a x:(r3)+,x0                ; to compute 9/11-wd2 = 1 +(8/10-wd2)
                asr4    a                           ; a = p_nbl >> 6
                move    x:(r3)+,r0                  ; to address the ILB table
                move    #31,y1                      ; for mask
                and     y1,a  a,b                   ; b = p_nbl >> 6
                move    a1,n0                       ; offset of table ILB
                asr4    b
                asr     b   x:(r0)+n0,y1            ; b = p_nbl>>11, dummy read,r0->
                tfr     x0,b  b,y1                  ; b= 9/11, y1 = wd2 (lsp set to 0)
                sub     y1,b                        ; b1 = 9/11 - wd2 (always >=0)
                move    p:(r0),a                    ; read table ILB*2 (ie 9-wd2)
                rep     b1                          ; b1 must be >=0
                asr     a                           ; a = a >> (9/11-wd2)
                move    a,a                         ; set lsp of a  to 0
                asl     a
                asl     a                           ; a = a << 2
                move    a,x:(r2)                    ; save new detl
;;

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;                                                                    ;;
;;      Predictor :   compute the following equations of the         ;;
;;      ========= :  G.722 predictor (see detailed                    ;;
;;                :  recommendation and 'C' program).                 ;;
;;                :       upzero(dlt,bl);                             ;;
;;                :       plt[0]=parrec(dlt[0],szl);                  ;;
;;                :       rlt[0]=recons(sl,dlt[0]);                   ;;
;;                :       uppol2(al,plt);                             ;;
;;                :       uppol1(al,plt);                             ;;
;;                :       szl=filtez(dlt,bl);                         ;;
;;                :       spl=filtep(rlt,al);                         ;;
;;                :       sl=predic(spl,szl);                         ;;
;;                                                                    ;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
predictor       clr      b  x:(r2+dlt0),a           ; a = dlt0, b=0
                move    #64,x0                      ; x0 = 128/2
                move    #-64,y0                     ; y0 = -128/2
                tst     a                           ; set flag
                tgt     x0,b                        ; if >0 b =64
                tlt     y0,b                        ; if < 0 b=-64 (else b=0)
                asl     b  b0,y1                    ; b=2*b; y1 = 0
                move    b,y0                        ; sign suppressed in y
;;
;;      address computation
;;      ===================
;;
                move    #dlt6,n2                    ; for address computation
                move    #-2,n0                      ; for updating the delay line
                lea     (r2)+n2,r0                  ; r0 = address of dlt6
                move    #bl6,n2                     ; for adress computation
                move    n0,n3                       ; idem
                lea     (r2)+n2,r3                  ; r3 = address of bl6
;;
```

```
;;      upzero
;;      ======
;;
                move    #32640,x1                       ; fixed coefficient for bli
                move    x:(r0)+,a x:(r3)+,x0            ; a= dlt6, x0 = bl6
                add     y,a         (r0)+n0             ; a= dlt6+wd1, r0 = &dlt5
                abs     a           (r3)+n3             ; set sign of a0, r3 =&bl5
                mpy     x1,x0,b     a0,x0               ; x0 = wd2, b= 32640*bl6
                add     x0,b x:(r0)+,a x:(r3)+,x0       ; b = new bl6, a= dlt5,x0= bl5
;;
;;      loop for the following bli
;;      ========================
;;
                do      #5,end_upzero
                add     y,a         a,x:(r0)+n0         ; save dlti in dlti-1,r0 =&dlti+1
                abs     a           b,x:(r3)+n3         ; save bli
                mpy     x1,x0,b     a0,x0               ; x0 = wd2, b= 32640*bli-1
                add     x0,b        x:(r0)+,a x:(r3)+,x0 ; b =new bli-1
end_upzero
;;
;;      We must compute the new plt0 and the rlt0 and save 2*dlt0 in dlt1
;;      for the filtez computation
;;      Also we must save the new bl1 coefficients in ram
;;      ==============================================
;;
                tfr     a,x0  b,x:(r3)+                 ; x0=a=dlt0, save bl1 in ram
                asl     a  x:(r2+szl),b                 ; a=2*dlt0, b= szl
                add     x0,b x:(r2+plt1),x1             ; b= plt0, x1=plt1 (ie plt2)
                tfr     b,y1  a,x:(r0)+                 ; save 2*dlt0 in dlt1
                eor     x1,b  x:(r2+sl),a               ; sg0^sg2=b, a= sl
                add     x0,a  x:(r2+plt0),x0            ; a= rlt0, x0 = plt0 (ie plt1)
                asl     a    x0,x:(r2+plt1)             ; a=2*rlt0, save new plt1
                tfr     y1,a  a,x:(r2+rlt0)             ; a= plt0, save 2*rlt0 in ram
                eor     x0,a  y1,x:(r2+plt0)            ; sg0^sg1=a, save new plt0
;;      uppol2 and uppol1
;;      ================
;;

;;      uppol2
;;      ======
;;
                move    a1,x1                           ; x1 = sg0 ^ sg1
                move    b1,y1                           ; y1 = sg0 ^ sg2
                move    x:(r2+al1),a                    ; a= al1
                move    #-192,b                         ; b=-192
                neg     b           b,y0                ; b =192, y0 = -192
                asl     a                               ; to compute wd1
                asl     a                               ; for limiting and fixe a0 to 0
                neg     a           a,x0                ; a= -wd1 , x0 = wd1 (4*al1)
                tst     x1                              ; test if sg0 ^ sg1 == 1 or 0
                tlt     x0,a                            ; if 1 a= wd1 ==>wd2
                tlt     y0,b                            ; if 1 b = -192 (wd1 of uppol1)
                asr4    a                               ; wd2 >>4
                asr4    a                               ; wd2 >>4
                asl     a           b,y0                ; wd2 <<1
                ; y0 = wd1_uppol1
                move    #128,b                          ; for wd3
                move    #-128,x0                        ; for -wd3
                tst     y1                              ; test sg0 ^ sg2
```

```
            tlt     x0,b                          ; set b to wd3
            add     a,b       x:(r2+al2),x0       ; b= wd4, read al2 in x0
            move    #32512,y1                     ; set 32512 in y1
            move    b,b                           ; limit wd4
            mac     y1,x0,b                       ; b= apl2
            move    #-12288,a                     ; set lower limit in a
            move    b,b                           ; limit apl2
            neg     a         a,x0                ; a= 12288, x0=-12288
            cmp     a,b                           ; compare apl2 with +12288
            tpl     a,b                           ; set b to 12288 if gt
            cmp     x0,b                          ; compare apl2 with -12288
            tmi     x0,b                          ; set b to -12288 if lt
            tfr     y0,a b,x:(r2+al2)             ; y0 = wd1, save new al2
;;
;;    uppol1
;;    ======
;;
            move    #15360,x0                     ; to compute wd3
            sub     x0,b      x:(r2+al1),x0       ; b = -wd3, x0 = al1
            neg     b         b,y0                ; b = wd3 , y0 = -wd3
            move    #32640,x1                     ; factor of al1
            mac     x0,x1,a                       ; a= apl1
            move    a,a                           ; limit apl1
            cmp     b,a                           ; test if a > wd3
            tpl     b,a                           ; set a to wd3 if gt
            cmp     y0,a                          ; test if a < wd3
            tmi     y0,a                          ; set to -wd3 if lt
;;
;;    filtez
;;    ======
;;
            move    #dlt6,n2                      ; for computation updating
            move    #-1,n0                        ; n0 = -1
            lea     (r2)+n2,r0                    ; r0 = address of dlt6
            move    #bl6,n2                       ;
            move    n0,n3                         ; n3 =-1
            lea     (r2)+n2,r3                    ; r3 = address of bl6
;;
            move    a,x:(r2+al1)                  ; save new al1
;;
            move    x:(r0)+n0,y1 x:(r3)+n3,x1     ; y1 =dlt6, x1 = bl6

            mpy     x1,y1,a x:(r0)+n0,y1 x:(r3)+n3,x1   ; y1 =dlt5, x1 = bl5
            move    a,a                           ; limit partial product
            mac     x1,y1,a x:(r0)+n0,y1 x:(r3)+n3,x1   ; y1 =dlt4, x1 = bl4
            move    a,a                           ; limit partial product
            mac     x1,y1,a x:(r0)+n0,y1 x:(r3)+n3,x1   ; y1 =dlt3, x1 = bl3
            move    a,a                           ; limit partial product
            mac     x1,y1,a x:(r0)+n0,y1 x:(r3)+n3,x1   ; y1 =dlt2, x1 = bl2
            move    a,a                           ; limit partial product
            mac     x1,y1,a x:(r0)+n0,y1 x:(r3)+n3,x1   ; y1 =dlt1, x1 = bl1
            move    a,a                           ; limit partial product
            mac     x1,y1,a x:(r0)+n0,y1 x:(r3)+n3,x1   ; y1 =dlt0, x1 = al2
;;                                                ; a = szl then limit in x0
;;
```

```
;;      filtep
;;      ======
;;
                tfr     a,x0        x:(r0)+n0,y1            ; y1 =rlt1, x0 =szl
                mpy     x1,y1,a x:(r0)+,y1 x:(r3)+n3,x1     ; y1 =rlt0, x1 = al1
                move    a,a                                 ; limit al2 * rlt2
                mac     x1,y1,a x0,x:(r2+szl)               ; save szl
                add     x0,a    y1,x:(r0)+                  ; rlt0 in rlt1
                                                            ; prediction in accu a
;;
;;      return of subroutine pred_l or pred_h
;;
;;      ================================================================
;;      WARNING: the prediction sl or sh is in accu A and must be saved
;;              in the calling procedure
;;      ================================================================
;;
                rts
```

## A.7. (a) : G.722 Variable Initialisation Structure

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      ============================================
;;      This data ram area is initialized at reset by the init_const
;;      procedure
;;      ============================================
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      Table for selection of decoder (mode)
;;      ======================================
;;
sel_mode        ds      1               ; for mode 1 = 64 kbit/s
                ds      1               ; for mode 2 = 56 kbit/s
                ds      1               ; for mode 3 = 48 kbit/s
;;
;;      constant area for lower sub band predictor
;;      ============================================
;;
const_pr_l      ds      1               ; inverse 4 bits quantizer
                ds      1               ; log adaptation (4 bits)
                ds      1               ; multiplicand factor
                ds      1               ; upper limit of p_nbl (low sub band)
                ds      1               ; to compute shift right
                ds      1               ; table of 32 values
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      constant area for higher sub band predictor
;;      ============================================
;;
const_pr_h      ds      1               ; inverse 2 bits quantizer
                ds      1               ; log adaptation (2 bits)
                ds      1               ; multiplicand factor
                ds      1               ; upper limit of p_nbl (high sub band)
                ds      1               ; to compute shift right
                ds      1               ; table of 32 values
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;       quantizer thresholds (Q6) for lower sub_band encoder
;       ====================================================
;
level__1        ds      1               ; Q6(-1)
level_0         ds      1               ; Q6( 0)
level_1         ds      1               ; Q6( 1)
level_2         ds      1               ; Q6( 2)
level_3         ds      1               ; Q6( 3)
level_4         ds      1               ; Q6( 4)
level_5         ds      1               ; Q6( 5)
level_6         ds      1               ; Q6( 6)
level_7         ds      1               ; Q6( 7)
level_8         ds      1               ; Q6( 8)
level_9         ds      1               ; Q6( 9)
level_10        ds      1               ; Q6(10)
level_11        ds      1               ; Q6(11)
```

```
level_12        ds      1                       ; Q6(19)
level_13        ds      1                       ; Q6(13)
level_14        ds      1                       ; Q6(14)
level_15        ds      1                       ; Q6(15)
level_16        ds      1                       ; Q6(16)
level_17        ds      1                       ; Q6(17)
level_18        ds      1                       ; Q6(18)
level_19        ds      1                       ; Q6(19)
level_20        ds      1                       ; Q6(20)
level_21        ds      1                       ; Q6(21)
level_22        ds      1                       ; Q6(22)
level_23        ds      1                       ; Q6(23)
level_24        ds      1                       ; Q6(24)
level_25        ds      1                       ; Q6(25)
level_26        ds      1                       ; Q6(26)
level_27        ds      1                       ; Q6(27)
level_28        ds      1                       ; Q6(28)
level_29        ds      1                       ; Q6(29)
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      Coefficients for QMF TX and RX filters
;;      ======================================
;;      Note: they must be seen as H[23],H[22] and so on
;;      ________________________________________________
;;
q_coef ds       24                      ; for 24 coeffiecents
;;
;;=======================================================
;;      QMF sections
;;      ============
;;=======================================================
;;
;;      address for modulo arithmetic of delay TX line
;;      ==============================================
        org     x:$100
;;
dat_q_tx        ds      24                      ; delay line for QMF tx filter
ptr_q_tx        ds      1                       ; modulo pointer to dat_q_tx
;;
;;
;;      address for modulo arithmetic of delay RX line
;;      ==============================================
org     x:$200
;;
dat_q_rxds      24                              ; delay line for QMF rx filter
ptr_q_rxds      1                               ; modulo pointer to dat_q_rx
;;
```

## A.7. (b) : Variable Initialisation Subroutine calls

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      Beginning of test program
;;      =======================
;;
                org     p:$40
;;
prog            reset                   ; reset on chip peripherals
                nop                     ; for the pipeline
                nop
;;
                ori     #$30,omr        ; saturation 32 bits,rounding 2s
                nop                     ; for the pipeline
                nop
;;
;;      Reset section
;;      =============
;;
                jsr init_const          ; init const in ram
                jsr reset_cod           ; encoder reset
                jsr reset_dec           ; decoder reset
                jsr init_q_tx           ; reset qmf tx
                jsr init_q_rx           ; reset qmf rx
;;
```

## A.7. (c) : Variable Initialisation Subroutines

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
; reset_cod: subroutine to reset the encoder (lower and higher)              ;
;         state variables                                                    ;
;         We must call this subroutine in order to pass the                  ;
;         digital test sequences of the CCITT G.722                          ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
reset_cod       move    #dat_lsbcod,r0          ; pointer to data of l_coder
                move    #32,x0                  ; set detl for reset
                move    x0,x:(r0)+              ; save in memory
                clr     a                       ; set a to 0
                rep     #22                     ; set 22 state variables to 0
                move    a,x:(r0)+               ; end for coder_low
;
                move    #dat_hsbcod,r0          ; pointer to data of h_coder
                move    #8,x0                   ; set deth for reset
                move    x0,x:(r0)+              ; save in memory
                rep     #22                     ; set 22 state variables to 0
                move    a,x:(r0)+               ; end for coder_high
;
                rts                             ; return of subroutine
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                            ;
; reset_dec: subroutine to reset the decoder (lower and higher)              ;
;         states variables                                                   ;
;         We must call this subroutine in order to pass the                  ;
;         digital test sequences of CCITT G.722                              ;
;                                                                            ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
reset_dec       move    #dat_lsbdec,r0          ; pointer to data of l_decoder
                move    #32,x0                  ; set detl for reset
                move    x0,x:(r0)+              ; save in memory
                clr     a                       ; set a to 0
                rep     #22                     ; set 22 state variables to 0
                move    a,x:(r0)+               ; end for decoder_low
;
                move    #dat_hsbdec,r0          ; pointer to data of h_decoder
                move    #8,x0                   ; set deth for reset
                move    x0,x:(r0)+              ; save in memory
                rep     #22                     ; set 22 state variables to 0
                move    a,x:(r0)+               ; end for decoder_low
;
                rts                             ; return of subroutine
;
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      Subroutine to initialise qmf TX filter
;;      ========================================
;;      Set to 0 all the delay line and initialize the pointer
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
init_q_tx       move    #dat_q_tx,r0            ; address of delay line
                clr     a                       ; a to 0
                move    r0,x:ptr_q_tx           ; save pointer value
                rep     #24                     ; for 24 elements
                move    a,x:(r0)+               ; set all the line to 0
                rts                             ; end of subprogram
;;
;;      end of init_q_tx
;;      ================
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      Subroutine to initialise qmf RX filter
;;      ========================================
;;      Set to 0 all the delay line and initialize the pointer
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
init_q_rx       move    #dat_q_rx,r0            ; address of delay line
                clr     a                       ; a to 0
                move    r0,x:ptr_q_rx           ; save pointer value
                rep     #24                     ; for 24 elements
                move    a,x:(r0)+               ; set all the line to 0
                rts                             ; end of subprogram
;;
;;      end of init_q_rx
;;      ================
```

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      Subroutine to initialise constants in data ram
;;      ==================================
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
init_const      move    #pr_sel_mode,r0         ; start of sel_mode
                move    #sel_mode,r3            ; in ram
                rep     #3                      ; for 3 values
                move    p:(r0)+,x:(r3)+         ; prom_ram
;;
                move    #pr_const_pr_l,r0       ; start of const_pr_l
                move    #const_pr_l,r3          ; in ram
                rep     #6                      ; for 3 values
                move    p:(r0)+,x:(r3)+         ; prom_ram
;;
                move    #pr_const_pr_h,r0       ; start of const_pr_h
                move    #const_pr_h,r3          ; in ram
                rep     #6                      ; for 3 values
                move    p:(r0)+,x:(r3)+         ; prom_ram
;;
                move    #pr_level__1,r0         ; start of level__1
                move    #level__1,r3            ; in ram
                rep     #31                     ; for 3 values
                move    p:(r0)+,x:(r3)+         ; prom_ram
;;
                move    #pr_q_coef,r0           ; start of q_ceof
                move    #q_coef,r3              ; in ram
                rep     #24                     ; for 3 values
                move    p:(r0)+,x:(r3)+         ; prom_ram
;;
                rts                             ; end of subprogram
;;
;;      end of init_const
;;      ================
;;
```

## A.7. (d) : Initialisation Variable Structure

Required in P memory

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      Tables of constants that must be loaded in data RAM at reset
;;      ============================================================
;;
;;      Note: they have the same name but with a pr_ prefix (prom)
;;      ==========================================================
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      Table for selection of decoder (mode)
;;      =====================================
;;
pr_sel_mode     dc      QQ6                     ; for mode 1 = 64 kbit/s
                dc      QQ5                     ; for mode 2 = 56 kbit/s
                dc      QQ4                     ; for mode 3 = 48 kbit/s
;;
;;      constant area for lower sub band predictor
;;      ==========================================
;;
pr_const_pr_l dc  QQ4                           ; inverse 4 bits quantizer
                dc      W4                      ; log adaptation (4 bits)
                dc      32512                   ; multiplicand factor
                dc      18432                   ; upper limit of p_nbl (low sub band)
                dc      9                       ; to compute shift right
                dc      ILB                     ; table of 32 values
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      constant area for higher sub band predictor
;;      ===========================================
;;
pr_const_pr_h   dc      QQ2                     ; inverse 2 bits quantizer
                dc      W2                      ; log adaptation (2 bits)
                dc      32512                   ; multiplicand factor
                dc      22528                   ; upper limit of p_nbl (high sub band)
                dc      11                      ; to compute shift right
                dc      ILB                     ; table of 32 values
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
;       quantiser thresholds (Q6) for lower sub_band encoder
;       ====================================================
;
pr_level__1     dc       0*8                    ; Q6(-1)
pr_level_0      dc       0*8                    ; Q6( 0)
pr_level_1      dc       35*8                   ; Q6( 1)
pr_level_2      dc       72*8                   ; Q6( 2)
pr_level_3      dc      110*8                   ; Q6( 3)
pr_level_4      dc      150*8                   ; Q6( 4)
pr_level_5      dc      190*8                   ; Q6( 5)
pr_level_6      dc      233*8                   ; Q6( 6)
pr_level_7      dc      276*8                   ; Q6( 7)
```

```
pr_level_8      dc      323*8                           ; Q6( 8)
pr_level_9      dc      370*8                           ; Q6( 9)
pr_level_10     dc      422*8                           ; Q6(10)
pr_level_11     dc      473*8                           ; Q6(11)
pr_level_12     dc      530*8                           ; Q6(19)
pr_level_13     dc      587*8                           ; Q6(13)
pr_level_14     dc      650*8                           ; Q6(14)
pr_level_15     dc      714*8                           ; Q6(15)
pr_level_16     dc      786*8                           ; Q6(16)
pr_level_17     dc      858*8                           ; Q6(17)
pr_level_18     dc      940*8                           ; Q6(18)
pr_level_19     dc      1023*8                          ; Q6(19)
pr_level_20     dc      1121*8                          ; Q6(20)
pr_level_21     dc      1219*8                          ; Q6(21)
pr_level_22     dc      1339*8                          ; Q6(22)
pr_level_23     dc      1458*8                          ; Q6(23)
pr_level_24     dc      1612*8                          ; Q6(24)
pr_level_25     dc      1765*8                          ; Q6(25)
pr_level_26     dc      1980*8                          ; Q6(26)
pr_level_27     dc      2195*8                          ; Q6(27)
pr_level_28     dc      2557*8                          ; Q6(28)
pr_level_29     dc      2919*8                          ; Q6(29)
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;
;;      Coefficients for QMF TX and RX filters
;;      ====================================
;;      Note: they must be seen as H[23],H[22] and so on
;;      ------------------------------------------------
;;
pr_q_coef       dc      3*2,-11*2,-11*2,53*2,12*2,-156*2
                dc      32*2,362*2,-210*2,-805*2,951*2,3876*2
                dc      3876*2,951*2,-805*2,-210*2,362*2,32*2
                dc      -156*2,12*2,53*2,-11*2,-11*2,3*2
;;
;;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
end
```

## A.8. Variable Structure Required in X memory

```
;;
;;      Area of X RAM
;;      =============
;;
org x:$00
;;              ;;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;;
;;      data for test
;;      =============
;;
mode            ds      1               ; mode of decoder from MODCOD.TST file
x1_in           ds      1               ; input of qmf
x0_in           ds      1               ; input of qmf
xout1           ds      1               ; output of qmf
xout2           ds      1               ; output of qmf
adc_in_1        ds      1               ; input #1 for adc at 16 kHz
adc_in_0        ds      1               ; input #0 for adc at 16 kHz
dac_out_1       ds      1               ; output #1 for dac at 16 kHz
dac_out_2       ds      1               ; output #2 for dac at 16 kHz
flag_in         ds      1               ; flag to check 2nd input at 16 kHz
sav_x0          ds      1               ; save for x0 in interrupt SSI
;;
;;+++++++++++++++++++++++++++++++++++++++++++++++++++
;;
;;      data for predictor in lower sub band coder
;;      ==========================================
;;
dat_lsbcod      ds      23              ; data ram for the lower sub band predictor
xl_cod          ds      1               ; input of lsbcod
il_cod          ds      1               ; output of lsbcod
;;
;;
;;      data for predictor in higher sub band coder
;;      ===========================================
;;
dat_hsbcod      ds      23              ; data ram for the higher sub band predictor
xh_cod          ds      1               ; input of hsbcod
ih_cod          ds      1               ; output of hsbcod
;;
;;      data for predictor in lower sub band decoder
;;      ============================================
;;
dat_lsbdec      ds      23              ; data ram for the lower sub band predictor
ilr_dec         ds      1               ; input of lsbdec
yl_dec          ds      1               ; output of lsbdec
;;
;;;     data for predictor in higher sub band decoder
;;      =============================================
;;
dat_hsbdec      ds      23              ; data ram for the higher sub band predictor
ihr_dec         ds      1               ; input of hsbdec
yh_dec          ds      1               ; output of hsbdec
;;
```

```
;;      output of encoder
;;      =================
;;
is              ds      1               ; output code of encoder
;;
;;      input of decoder
;;      ================
;;
ir              ds      1               ; receive code for input of decoder
;;
```

**Literature Distribution Centers:**
USA: Motorola Literature Distribution; P.O. Box 20912; Phoenix, Arizona 85036.
EUROPE: Motorola Ltd.; European Literature Centre; 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
JAPAN: Nippon Motorola Ltd.; 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141 Japan.
ASIA-PACIFIC: Motorola Semiconductors H.K. Ltd.; Silicon Harbour Center, No. 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.

MOTOROLA

1PHX33213–0 PRINTED IN USA 4/92 IMPERIAL LITHO 200

APR404/D