INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

G.722.1
(09/99)

SERIES G: TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS

Digital transmission systems – Terminal equipments – Coding of analogue signals by methods other than PCM

Coding at 24 and 32 kbit/s for hands-free operation in systems with low frame loss

ITU-T Recommendation G.722.1

(Previously CCITT Recommendation)

ITU-T G-SERIES RECOMMENDATIONS

TRANSMISSION SYSTEMS AND MEDIA, DIGITAL SYSTEMS AND NETWORKS

INTERNATIONAL TELEPHONE CONNECTIONS AND CIRCUITS	G.100–G.199
INTERNATIONAL ANALOGUE CARRIER SYSTEM	
GENERAL CHARACTERISTICS COMMON TO ALL ANALOGUE CARRIER-TRANSMISSION SYSTEMS	G.200–G.299
INDIVIDUAL CHARACTERISTICS OF INTERNATIONAL CARRIER TELEPHONE SYSTEMS ON METALLIC LINES	G.300–G.399
GENERAL CHARACTERISTICS OF INTERNATIONAL CARRIER TELEPHONE SYSTEMS ON RADIO-RELAY OR SATELLITE LINKS AND INTERCONNECTION WITH METALLIC LINES	G.400–G.449
COORDINATION OF RADIOTELEPHONY AND LINE TELEPHONY	G.450–G.499
TESTING EQUIPMENTS	
TRANSMISSION MEDIA CHARACTERISTICS	G.600–G.699
DIGITAL TRANSMISSION SYSTEMS	
TERMINAL EQUIPMENTS	G.700–G.799
General	G.700–G.709
Coding of analogue signals by pulse code modulation	G.710–G.719
Coding of analogue signals by methods other than PCM	**G.720–G.729**
Principal characteristics of primary multiplex equipment	G.730–G.739
Principal characteristics of second order multiplex equipment	G.740–G.749
Principal characteristics of higher order multiplex equipment	G.750–G.759
Principal characteristics of transcoder and digital multiplication equipment	G.760–G.769
Operations, administration and maintenance features of transmission equipment	G.770–G.779
Principal characteristics of multiplexing equipment for the synchronous digital hierarchy	G.780–G.789
Other terminal equipment	G.790–G.799
DIGITAL NETWORKS	G.800–G.899
DIGITAL SECTIONS AND DIGITAL LINE SYSTEM	G.900–G.999

For further details, please refer to ITU-T List of Recommendations.

ITU-T RECOMMENDATION G.722.1

CODING AT 24 AND 32 kbit/s FOR HANDS-FREE OPERATION IN SYSTEMS WITH LOW FRAME LOSS

Summary

This Recommendation describes a low complexity encoder and decoder that may be used for 7 kHz bandwidth audio signals working at 24 kbit/s or 32 kbit/s. Further, this algorithm is recommended for use in hands-free applications such as conferencing where there is a low probability of frame loss. It may be used with speech or music inputs. The bit rate may be changed at any 20 ms frame boundary.

This Recommendation includes a software package which contains the encoder and decoder source code and a set of test vectors for developers. These vectors are a tool providing an indication of success in implementing this code.

Source

ITU-T Recommendation G.722.1 was prepared by ITU-T Study Group 16 (1997-2000) and was approved under the WTSC Resolution No. 1 procedure on 30 September 1999.

CONTENTS

Recommendation G.722.1

CODING AT 24 AND 32 kbit/s FOR HANDS-FREE OPERATION IN SYSTEMS WITH LOW FRAME LOSS[1]

(Geneva, 1999)

1 Scope

This Recommendation describes a digital wideband coder algorithm that provides an audio bandwidth of 50 Hz to 7 kHz, operating at a bit rate of 24 kbit/s or 32 kbit/s. The digital input to the coder may be 14, 15 or 16 bit 2's complement format at a sample rate of 16 kHz (handled in the same way as in Recommendation G.722). The analogue and digital interface circuitry at the encoder input and decoder output should conform to the same specifications described in Recommendation G.722.

The algorithm is based on transform technology, using a Modulated Lapped Transform (MLT). It operates on 20 ms frames (320 samples) of audio. Because the transform window (basis function length) is 640 samples and a 50 per cent (320 samples) overlap is used between frames, the effective look-ahead buffer size is 20 ms. Hence the total algorithmic delay of 40 ms is the sum of the frame size plus look-ahead. All other delays are due to computational and network transmission delays.

The description of the coding algorithm of this Recommendation is made in terms of bit-exact, fixed-point mathematical operations. The C code indicated in clause 5, which constitutes an integral part of this Recommendation, reflects this bit-exact, fixed-point descriptive approach, and shall take precedence over the mathematical descriptions of clauses 3 and 4 whenever discrepancies are found.

The mathematical descriptions of the encoder (clause 3), and decoder (clause 4), could have been implemented in several other fashions, but the C code of clause 5 has been provided as reference purposes. Thus, to comply with this Recommendation, any implementation must produce for any input signal the same output results as the C code of clause 5.

Note that to ensure that this goal is achieved, implementations should follow the computational details, tables of constants, sequencing of variable adaptation and use given by the C code of clause 5. However, it is recognized that there are many parts of the algorithm critical to maintaining correct bit-exact operation. For these parts, implementations shall reproduce the computational details, tables of constants, sequencing of variables adaptation and use written in the C code of clause 5.

It is recognized that the C code provided is for reference, and has not been optimized (in terms of memory, complexity, etc.) for a specific implementation platform. The C code may require optimization for a particular implementation.

A non-exhaustive set of test signals is provided as part of this Recommendation, as a tool to assist implementors to verify their implementations of the encoder and decoder comply with this Recommendation.

In practice, purchasers of wideband equipment or software implementations will expect them to be compliant with this standard to ensure interworking capability. Implementors may choose to optimize the C code, or otherwise modify the reference C code. In such cases the implementor shall verify that his implementation produces the same resultant output for any given input as would be expected using the C code expressed in clause 5.

[1] This Recommendation includes a software package which contains the encoder and decoder source code and a set of test vectors for developers.

2 Normative references

The following ITU-T Recommendations and other references contain provisions which, through reference in this text, constitute provisions of this Recommendation. At the time of publication, the editions indicated were valid. All Recommendations and other references are subject to revision; all users of this Recommendation are therefore encouraged to investigate the possibility of applying the most recent edition of the Recommendations and other references listed below. A list of the currently valid ITU-T Recommendations is regularly published.

[1] CCITT Recommendation G.722 (1988), *7 kHz audio-coding within 64 kbit/s*.

[2] ITU-T Recommendation G.192 (1996), *A common digital parallel interface for speech standardization activities*.

[3] ISO/IEC 9899:1999, *Programming languages – C*.

3 The encoder

Figure 1 presents a block diagram of the encoder.

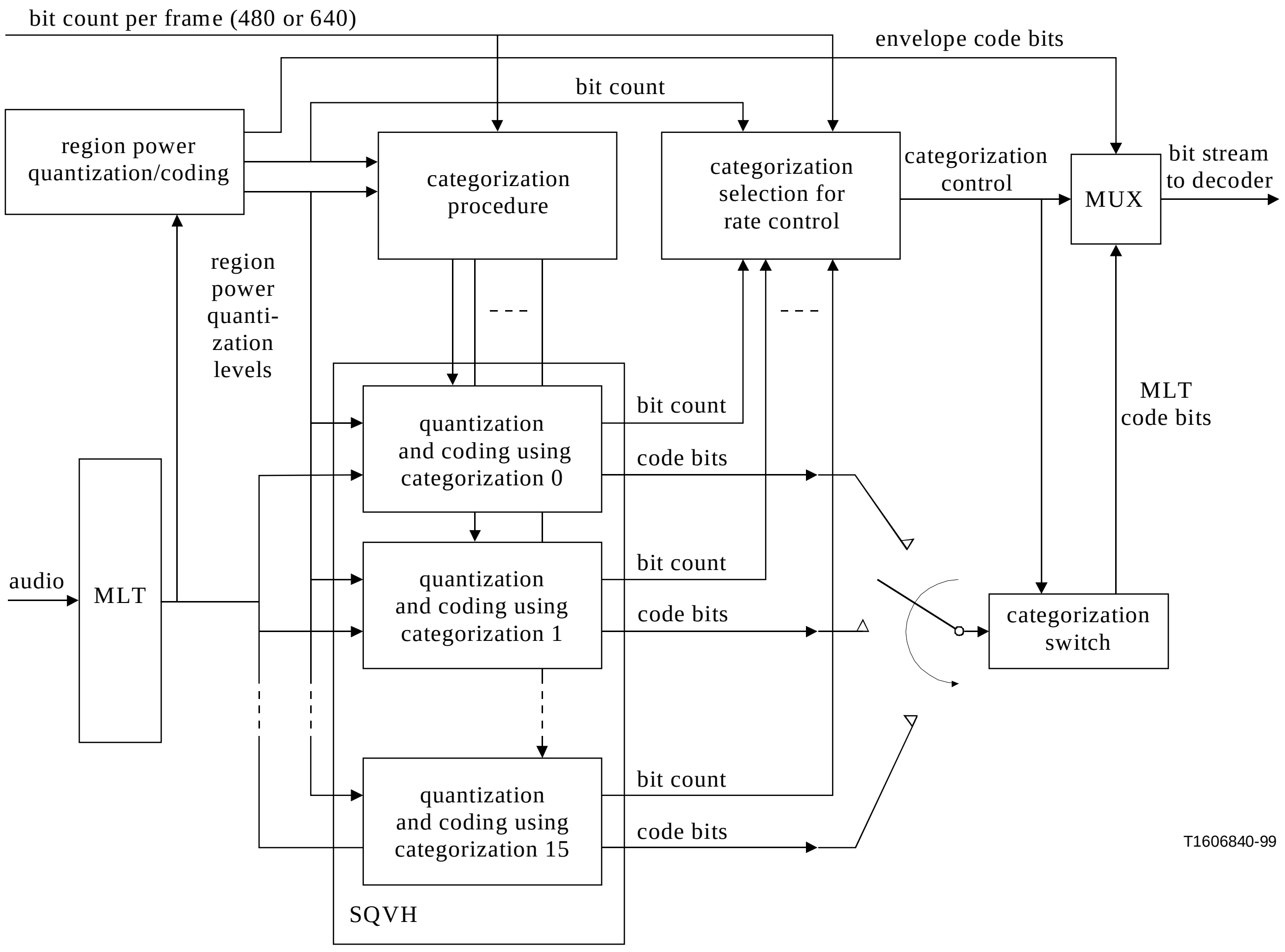
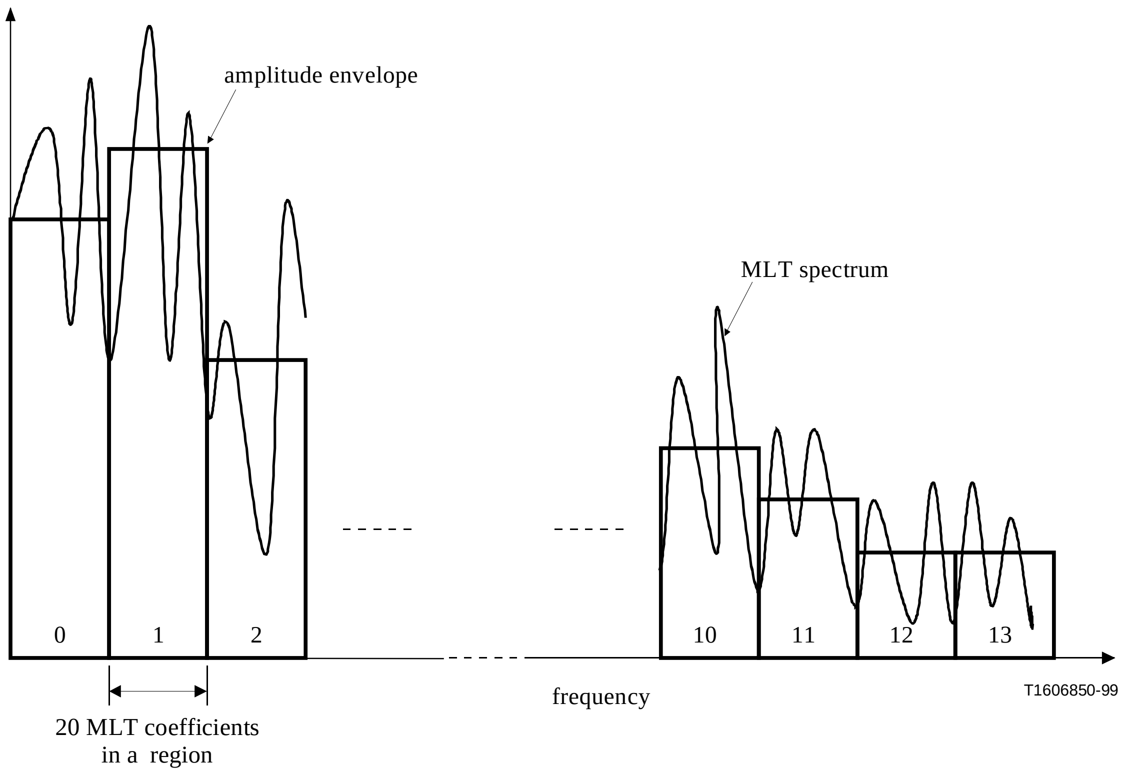

bit count per frame (480 or 640)
envelope code bits
bit count
region power quantization/coding
categorization procedure
categorization selection for rate control
categorization control
MUX
bit stream to decoder
region power quanti-zation levels
quantization and coding using categorization 0
bit count
code bits
MLT code bits
audio
MLT
quantization and coding using categorization 1
bit count
code bits
categorization switch
quantization and coding using categorization 15
bit count
code bits
SQVH
T1606840-99

Figure 1/G.722.1 – Block diagram of the encoder

Every 20 milliseconds (320 samples) the most recent 640 time domain audio samples are fed to a Modulated Lapped Transform (MLT). Each transform produces a frame of 320 MLT coefficients, and each frame of MLT coefficients is coded independently, i.e. there is no state information left over from the previous frame. For 24 kbit/s and 32 kbit/s operation the allotment of bits per frame is 480 and 640, respectively.

The transform coefficients generated by the MLT transform are first applied to a module which computes the *amplitude envelope* and quantizes it; see Figure 2. The amplitude envelope is a coarse representation of the MLT spectrum. The spectrum is divided into blocks of 20 MLT coefficients called *regions*. Each region represents a bandwidth of 500 Hz. As the bandwidth is 7 kHz, the *number_of_regions* is set at fourteen. MLT coefficients representing frequencies above 7 kHz are ignored. The code bits representing the amplitude envelope are sent to the MUX (Multiplexer) for transmission to the decoder. The bits remaining after quantization and coding of the amplitude envelope are used to encode the MLT coefficients in the *categorization* process.



NOTE – Each value of the amplitude envelope represents the RMS (root-mean-square) value of the MLT coefficients in that region.

Figure 2/G.722.1 – An illustration of how the spectrum is divided into fourteen regions, each containing 20 MLT coefficients

Using the quantized amplitude envelope and the number of bits remaining in the frame after amplitude envelope encoding (and provision for four categorization control bits), the categorization procedure generates sixteen sets of *categorizations* (*categorization 0* to *categorization 15*). Different categorizations require different numbers of bits to encode the same MLT coefficients.

Each categorization consists of a set of fourteen *category assignments,* one assignment for each of the fourteen regions. A *category* defines a set of predetermined quantization and coding parameters for a region. Associated with each category is an expected number of bits required to encode a region. Because this coder uses variable length Huffman coding, the final number of bits used will vary depending on the particular sequence of MLT coefficients in the region.

Next, the MLT coefficients are quantized and coded differently for each one of the sixteen computed categorizations. For each categorization the actual number of code bits required is determined.

The quantization and encoding proceeds region by region. A categorization determines the category assignment for all the fourteen regions, and the category assignment together with the amplitude envelope for each region determine all of the quantization and coding parameters which will be used for all twenty MLT coefficients in the region.

The MLT coefficients in a region are first normalized by the quantized amplitude envelope in the region and then scalar quantized. The resulting scalar quantization indices are combined into vector indices. The vector indices are then Huffman coded, i.e. they are coded with a variable number of bits. The most frequent vector indices require fewer bits than the less frequent vector indices.

Because this codec uses variable length Huffman coding and a constant transmitted bit rate is required, a method of constraining the bit rate to the channel rate is required. Four *categorization control* bits identify to the decoder which categorization was selected. The categorization switch directs the code bits (representing the quantized MLT coefficients produced using the selected categorization) to the MUX for transmission. The categorization that results in providing the number of bits closest to the channel rate is selected for transmission.

3.1 The Modulated Lapped Transform (MLT)

The MLT is a critically sampled, perfect reconstruction, linear transform with a 50 per cent overlap between the basis functions of adjacent MLT frames. The inputs to each MLT are the most recent 640 audio samples, *x(n)*,

where:

$$x\,(0) \text{ is the oldest sample,}$$

and:

$$0 \le n < 640$$

The MLT outputs 320 transform coefficients, *mlt(m)*,

where:

$$0 \le m < 320$$

The MLT is given by:

$$mlt(m) = \sum_{n=0}^{639} \sqrt{\frac{2}{320}} \sin\left(\frac{\pi}{640}(n+0.5)\right)\cos\left(\frac{\pi}{320}(n-159.5)(m+0.5)\right)x(n)$$

The MLT can be decomposed into a window, overlap and add operation followed by a type IV Discrete Cosine Transform (DCT). The window, overlap and add operation is given by:

$$v(n) = w(159-n)x(159-n) + w(160+n)x(160+n) \quad \text{for } 0 \le n \ge 159$$

$$v(n+160) = w(319-n)x(320+n) - w(n)x(639-n) \quad \text{for } 0 \le n \ge 159$$

where:

$$w(n) = \sin\left(\frac{\pi}{640}(n+0.5)\right) \quad \text{for } 0 \le n < 320$$

Combining $v(n)$ with a type IV DCT, the resulting expression for the MLT is:

$$mlt(m) = \sum_{n=0}^{319} \sqrt{\frac{2}{320}} \cos\left(\frac{\pi}{320}(n+0.5)(m+0.5)\right)v(n)$$

Note that fast transform techniques are used to significantly reduce the complexity of the DCT.

3.2 Computing and quantizing the amplitude envelope

The MLT coefficients are divided into regions of twenty coefficients. Thus, the total *number_of_regions* = 14. Region r includes MLT coefficients $20r$ through $20r + 19$,

where:

$$0 \le r < number\_of\_regions$$

The forty highest frequency MLT coefficients, representing frequencies above 7 kHz, are not used because they are outside the bandwidth of interest.

The *amplitude envelope* in the region r is defined as the RMS (Root-Mean-Square) value of the MLT coefficients in the region, and is computed as

$$rms(r) = \sqrt{\frac{1}{20}\sum_{n=0}^{19} mlt(20r+n)mlt(20r+n)}$$

It is then quantized. The quantizer output index is *rms_index*(r). The allowed set of quantization reconstruction values are:

$$2^{\left(\frac{i+2}{2}\right)} \quad \text{for integer values of } i,\ -8 \le i \le 31$$

and *rms_index*(0) is further constrained so that

$$1 \le rms\_index(0) \le 31$$

A log domain metric is used so that the values which get quantized to $2^{\left(\frac{i+2}{2}\right)}$ range from $2^{\left(\frac{i-0.5+2}{2}\right)}$ to $2^{\left(\frac{i+0.5+2}{2}\right)}$.

For example, if *rms*(r) = 310, then the corresponding quantization level is $2^{\left(\frac{15+2}{2}\right)}$ or 362.04, and *rms_index*(r) = 15, because $2^{\left(\frac{15-0.5+2}{2}\right)}$ = 304.43.

3.3 Coding the amplitude envelope

rms_index(0) is the first value transmitted in each frame. Five bits are used. The most significant bit of the index is transmitted first. The value, *rms_index*(0) = 0, is reserved and not used.

The indices of the remaining thirteen regions are differentially coded and then Huffman coded for transmission. The largest differences which may be coded are +11 and −12. To contain the differences within this range, the valleys are first adjusted upwards to allow the peaks which follow them to be accurately represented. This is described in the following in pseudo C code:

```
for (r = number_of_regions - 2; r >= 0; r--)
{
    if (rms_index[r] < rms_index[r + 1] - 11)
        rms_index[r] = rms_index[r + 1] - 11;
}
```

```
for (r = 1; r < number_of_regions; r + +)
{
        j = rms_index[r] - rms_index[r - 1];
        if (j < - 12)
        {
              j = 12
              rms_index[r] = rms_index[r - 1] + j;
        }
        differential_rms_index[r] = j;
}
```

The differences, *differential_rms_index*[*r*], are transmitted in order of region. They are coded in accordance with the variable length Huffman codes defined in table differential_region_power_codes[*r*][*j* + 12], and the table differential_region_power_bits[*r*][*j* + 12] which defines the number of bits for each Huffman code. These arrays are contained in the C code part of this Recommendation. Each region is associated with a unique set of Huffman codes. The leftmost (or most significant) bit is always transmitted first.

3.4 Categorization procedure

The *categorization procedure* determines the step-sizes (and other related quantization and coding parameters) used to quantize the MLT coefficients.

The process of *categorization* assigns a category to each of the regions. There are eight categories: 0-7. Sixteen different sets of categorizations are computed, and only one is selected for transmission.

The same categorization procedure is employed in the decoder. Hence, it is important for interoperability that when provided with the same inputs, different implementations of this procedure should produce identical categorizations. The inputs to this procedure are:

- *number_of_available_bits*: the actual number of bits in the frame still unused after accounting for the amplitude envelope and categorization control bits.
- *rms_index*(): the set of quantized values of *rms*(*r*) for all regions.

The category assigned to a region determines the quantization and coding parameters for that region, and the expected total number of bits required to represent the region's quantized MLT coefficients. Because variable length Huffman coding is used, the actual number of bits will vary depending on the statistics of a region's MLT coefficients. Hence, of the sixteen possible categorization sets computed, according to criteria described later, the best fitting categorization will be selected for transmission.

The expected number of bits for each category (0-7) is predefined in Table 1.

Table 1/G.722.1 – Expected number of bits for each category

category	**code bits per region as a function of category** (refer to expected_bits_table[] in the C code)
0	52
1	47
2	43
3	37
4	29
5	22
6	16
7	0

3.4.1 Adjusting the number of available bits

Based on the actual number of available bits, the following computes an estimation of the number of available bits:

if:

$$\mathit{number\_of\_available\_bits} > 320,$$

then:

$$\mathit{estimated\_number\_of\_available\_bits} = 320 + ((\mathit{number\_of\_available\_bits} - 320) * 5/8)$$

estimated_number_of_available_bits is always less than the actual number of bits to provide head room in the categorization process.

3.4.2 Calculating the initial categorization

For any integer *offset* in the range –32 to 31, the assignment of categories is given by:

$$\mathit{category}(r) = \mathrm{MAX}\ \{0, \mathrm{MIN}\ \{7, (\mathit{offset} - \mathit{rms\_index}(r)\ )/2\ \}\}$$

where:

$$0 \le r < \mathit{number\_of\_regions}.$$

The same *offset* is used for all regions. The total expected number of MLT code bits is:

$$\mathit{expected\_number\_of\_code\_bits} = \sum_{r=0}^{13} \mathit{expected\_bits\_table}(\mathit{category}(r))$$

The *offset* value is then adjusted until the largest *offset* found satisfies.

$$\mathit{expected\_number\_of\_code\_bits} \ge \mathit{estimated\_number\_of\_available\_bits} - 32$$

3.4.3 Generating the other fifteen categories

Once the initial categorization has been computed, the fifteen other categorizations must then be calculated. For each new categorization the category is adjusted in only one region relative to the previous categorization. The method for determining the remaining categorizations now follows:

$$\mathit{initial\_categorization}(r) = \mathrm{MAX}\ \{0, \mathrm{MIN}\ \{7, (\mathit{offset} - \mathit{rms\_index}(r))/2\}\}$$

where:

$$0 \le r < \mathit{number\_of\_regions}$$

create the temporary variables:

> *max_category*(*r*)
> *max_bits*
>
> *min_category*(*r*)
> *min_bits*

$max\_category(r) = initial\_categorization\ (r)$
$min\_category(r) = initial\_categorization\ (r)$

$max\_bits = expected\_number\_of\_code\_bits$
$min\_bits = expected\_number\_of\_code\_bits.$

Then for each of the remaining fifteen categorizations, the following comparison is performed,

if:

$$\text{max\_bits} + \text{min\_bits} \le 2 * \text{estimated\_number\_of\_available\_bits}$$

then:

> a new categorization is required with a larger expected number of bits.
>
> For the regions, *r*, for which
>
> $$max\_category(r) > 0$$
>
> find the region which minimizes the function
>
> $$offset - rms\_index(r) - 2 * max\_category(r)$$
>
> If there are several regions for which this function is equally small, then set *r* equal to the smallest (i.e. lowest frequency) such region.
>
> The category for this region in *max_category*(*r*) is then decreased by one; the expected number of bits for this new categorization is re-computed and *max_bits* is set equal to it.

Otherwise:

> a categorization with a smaller expected number of bits is required.
>
> For the regions, *r*, for which
>
> $$min\_category(r) < 7$$
>
> find the region which maximizes the function
>
> $$offset - rms\_index[r] - 2 * min\_category(r)$$
>
> If there are several regions for which this function is equally large, then set *r* equal to the largest (i.e. highest frequency) such region.
>
> The category for this region in *min_category*(*r*) is then increased by one; the expected number of bits for this new categorization is re-computed and *min_bits* is set equal to it.

In this way, sixteen unique categorizations are produced. They are ordered according to their expected number of bits as detailed in clause 6. Categorization 0 has the largest expected number of bits and categorization 15 the smallest. Each categorization is the same as its neighbouring categorization, except in one region where the category entry will differ by one, e.g. region 5 of categorization 7 may have a category of 2, and in categorization 8 region 5 may have category 3, while being equal to categorization 7 in the other regions.

A detailed flow chart of the categorization procedure is provided in clause 6.

3.5 Scalar Quantized Vector Huffman Coding (SQVH)

For regions assigned category values 0-6, the MLT coefficients are separated into sign and magnitude parts. The magnitude parts are normalized by the quantized value of *rms*(*r*), then scalar quantized with dead zone expansion, combined into vectors, and Huffman coded. Regions that are assigned a category 7 are not processed in this way at all, and are not allocated any bits for transmission.

For each region, *r*, the encoder first normalizes and quantizes the absolute value of each MLT coefficient, *mlt*(*i*), to produce quantization index, *k*(*i*):

$k(i)$ = MIN {whole number part of (x * absolute value of (mlt($20r + i$)) + *deadzone_rounding*),*kmax*}

where the index within a particular region is:

$$0 \le i < 20$$

and:

$$x = 1/(stepsize * (\text{quantized value of } rms(r))$$

and:

stepsize, deadzone_rounding, and kmax are given in Table 2.

Table 2/G.722.1 – Table of constants used by the SQVH procedure

category	stepsize	deadzone_rounding	kmax
0	$2^{-1.5}$	0.3	13
1	$2^{-1.0}$	0.33	9
2	$2^{-0.5}$	0.36	6
3	$2^{0.0}$	0.39	4
4	$2^{0.5}$	0.42	3
5	$2^{1.0}$	0.45	2
6	$2^{1.5}$	0.5	1

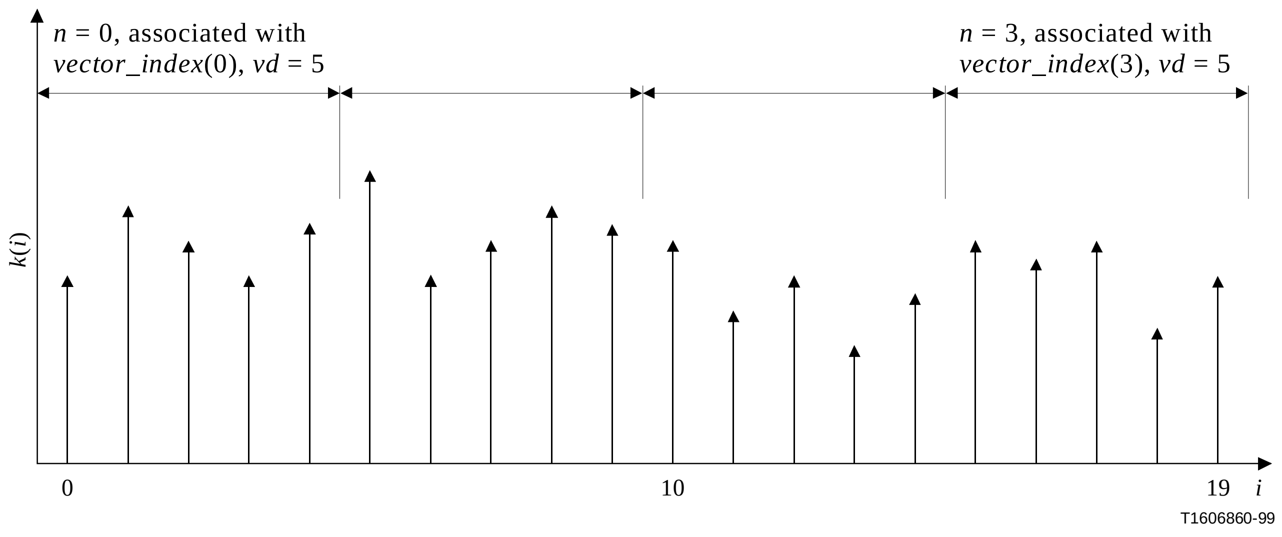
The indices, *k*(), are combined into vector indices; the properties of the vectors differ for each category. In each region there are *vpr* predefined vectors with dimension *vd* as defined in Table 3, and illustrated in Figure 3. The set of scalar *k*() values correspond to a unique vector identified by an index as follows:

$$vector\_index(n) = \sum_{j=0}^{vd-1} k(n \times vd + j)(kmax+1)^{(vd-(j+1))}$$

where: $0 \le n \le vpr - 1$, represents the n^{th} vector in the region r

and:

j =	index to j^{th} value of *k*() in a given vector, in a given region
vd =	vector dimension for given category
vpr =	number of vectors per region for a given category
$kmax$ =	maximum value of *k*() for a given category as shown in Table 2.



NOTE – Each vector represents *vd* quantized MLT coefficients.

Figure 3/G.722.1 – An example illustrating how a region, assigned a category of 6, is split into a series of five dimensional vectors (*vd* = 5) with four vectors per region (*vpr* = 4 and 0 ≤ *n* < 4)

Table 3 provides the values for *vd*, *vpr* and *u*, where $u = (kmax + 1)^{vd}$ represents the number of distinct values a vector may have in any given category.

Table 3/G.722.1 – Definition of constants *vd*, *vpr* and *u*

category	***vd***	***vpr***	***u***
0	2	10	196
1	2	10	100
2	2	10	49
3	4	5	625
4	4	5	256
5	5	4	243
6	5	4	32

The number of bits required to represent a vector, *vector_index*(*n*), for a given category is provided by tables mlt_sqvh_bitcount_category_0[] to mlt_sqvh_bitcount_category_6[]. These tables also provide the number of bits required by the corresponding code word entries in the tables mlt_svqh_code_category_0[] to mlt_svqh_code_category_6[]. These bit counts do not include the sign bits. The values of *k*() = 0 do not require sign bits:

The number of bits actually required (including sign bits) to represent the MLT coefficients for a region, *r*, of a given category, *y*, is given by:

$$number\_of\_region\_bits(r) = \sum_{n=0}^{vpr-1} mlt\_svqh\_bitcount\_category\_y(vector\_index(n)) \\ + (\text{number of sign bits in } n^{th} \text{ vector})$$

3.6 Rate control

The total number of bits actually required to represent the frame is computed for each categorization. This includes the bits used to represent the amplitude envelope, the four categorization control bits, and the bits required for the MLT coefficients. It then remains to select the best categorization for transmission and indicate this selection using the categorization control bits.

First, categorizations with bit totals in excess of the allotment are ruled out. Of the remaining categorizations the one with the lowest index is selected, e.g. when categorizations 0 through 3 use too many bits and categorization 4 fits within the bit allotment, categorization 4 is selected.

If no categorization yields a bit total that fits within the allotment, the categorization that comes closest (normally 15) is selected. Then, code bits are transmitted until the allotment for the frame is exhausted.

It may happen that the number of bits required by the encoder to represent one 20 ms frame of audio is less than the allowed number of bits per frame (480 or 640 bits). In this case the remaining unused bits at the end of the bit stream sequence are all set to one.

3.7 Transmission of the MLT vector indices

The vector indices are transmitted in frequency order – low to high frequency. They are coded in accordance with the variable length codes defined by the C code array mlt_svqh_bitcount_category_x[] and mlt_svqh_code_category_x[] (where _x represents the category value, $0 \le x \le 6$). The leftmost (or most significant) bits are transmitted first. The sign bits, relating to the non-zero MLT coefficients of each vector, are transmitted immediately following the respective vector index variable length code. The sign bits are also transmitted in frequency order. The sign bit is set to "1" for positive numbers.

3.8 Bit stream

The total number of bits in a frame is either 480 or 640 bits, for the bit rates of 24 kbit/s and 32 kbit/s respectively. While the number of bits in a frame is fixed, except for the categorization control bits parameter, all other parameters are represented by variable length codes – or a variable number of bits. Figure 4 illustrates this point, and the order of the transmitted parameter fields. All variable length codes, and the categorization control bits, are transmitted in order from the left most (most significant) bit to the right most (least significant) bit.

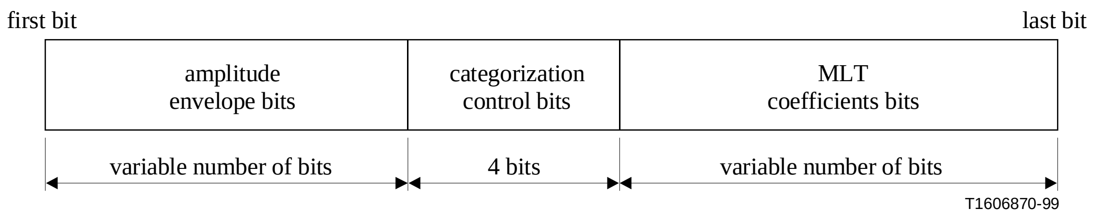


Figure 4/G.722.1 – Major bit stream fields and their order in transmission

4 The decoder

First for every frame, the first five bits, representing the amplitude index for region zero, are decoded. Then the remaining regions are Huffman decoded and reconstructed. The four categorization control bits can then be decoded to determine which of the sixteen possible categorizations was selected and transmitted by the encoder. The remaining code bits in the frame represent the quantized MLT coefficients and they are decoded according to the category information for each region. Just like in the encoder, the categorization procedure in the decoder uses the amplitude envelope together with the number of bits remaining to be decoded (in the current frame) and computes the set of sixteen possible categorizations.

Some regions may have been assigned a category of 7 by the encoder. This means that no MLT coefficients were transmitted to represent these regions. The category 7 regions are reconstructed using a technique called *noise-fill*. The average MLT magnitude for these regions is available from the amplitude envelope. Instead of setting the category 7 MLT coefficients to zero, the decoder sets the value of their amplitude proportional to the average MLT coefficient magnitude for the region, and the sign of each coefficient is set randomly. Determining the coefficient signs may be done by one of a number of methods; a simple pseudo-random number generator is sufficient.

Noise-fill is also applied to categories 5 and 6, because for these categories many of the MLT coefficients may be quantized to zero. The values which were transmitted as zero are set to small fractions of the average magnitude for the region. Again, the signs are determined randomly.

For those coefficients that were scalar quantized to non-zero values, a predetermined table contains reconstruction values of the normalized coefficients. The reconstructed values are then scaled using the appropriate value of $rms(r)$. The forty MLT coefficients representing frequencies above 7 kHz are set to zero. After reconstruction of the MLT coefficients, the Inverse Modulated Lapped Transform (IMLT) generates 320 new time domain samples.

Except for the final overlap and add operation of the IMLT, the information received in each frame by the decoder is independent of the information in the previous frame.

4.1 Decoding the amplitude envelope

The first five bits of the frame represent $rms\_index(0)$. Then, for the remaining regions, the variable length codes for $differential\_rms\_index(r)$ are decoded according to the arrays differential_region_power_bits[][] and differential_region_power_codes[][] referenced in the C code; the quantizer indices for these regions are reconstructed as follows:

$$rms\_index(r) = rms\_index(r-1) + differential\_rms\_index(r),$$

where:

$$1 \le r < number\_of\_regions.$$

4.2 Determining categorization

After decoding the amplitude envelope, the decoder determines the number of bits remaining to represent the MLT coefficients, this is done as follows:

bits available = bits per frame – amplitude envelope bits – four(categorization control bits)

Using the same categorization procedure as the encoder, the same set of sixteen possible categorizations is computed. The four categorization control bits indicate which categorization was used to encode the MLT coefficients, and consequently should also be used by the decoder.

4.3 Decoding MLT coefficients

For each region, the variable length codes representing the MLT vectors are decoded according to the appropriate category tables. The arrays mlt_svqh_bitcount_category_x[] and mlt_svqh_code_category_x[] are used for this purpose in the C code. (Where _x represents the category value, $0 \le x \le 6$.) The individual MLT coefficient quantization indices, $k(i)$, in a region are recovered from the vector index as follows:

$$k(i) = \left\lfloor \frac{vector\_index(n)}{(kmax+1)^j} \right\rfloor MOD(kmax+1)$$

where:

$\lfloor z \rfloor$ indicates taking the greatest integer value less than or equal to z

$i = (n + 1)vd - j - 1$

$0 \le j \le vd - 1$

$0 \le n \le vpr-1$, represents the n^{th} vector in the region r,

and:

vd = vector dimension for a given category

$kmax$ = maximum value of $k()$ for a given category as shown in Table 2.

Reconstruction of the MLT coefficients uses the centroid tables in the C code array mlt_quant_centroid[][]. The MLT coefficient amplitudes are reconstructed by computing the product of *rms*(*r*), in the region of interest, and the centroid specified by the decoded vector index. Non-zero values have their signs set according to the sign bit.

4.4 Noise-fill

No MLT coefficient amplitudes are encoded for regions assigned category 7. For categories 5 and 6 the large quantizer step sizes result in most MLT coefficients being coded as zero; these zeroes are replaced with coefficient values of random sign and amplitude proportional to *rms*(*r*). The proportionality constants are defined in Table 4.

Table 4/G.722.1 – Noise-fill proportionality constants

category	default noise-fill proportionality constant
5	0.176777
6	0.25
7	0.707107

4.5 Insufficient bits

There may be frames for which the encoder ran out of bits before it finished coding the last non-category 7 region. The decoder action in these cases is to process that region and all remaining regions with a category 7 assignment.

4.6 Frame erasure

If the decoder is informed (by means of an external signalling mechanism not defined in this Recommendation) that a frame has been lost or corrupted, it repeats the previous frame's decoded MLT coefficients. It proceeds by transforming them to the time domain, and performing the overlap and add operation with the previous and next frame's decoded information. If the previous frame was also lost or corrupted, then the decoder sets all the current frames MLT coefficients to zero.

4.7 The Inverse MLT (IMLT)

Each IMLT operation operates on 320 coefficients to produce 320 time domain audio samples. The IMLT can be decomposed into a type IV DCT followed by a window, overlap and add operation.

The type IV DCT is:

$$u(n) = \sum_{m=0}^{319} \sqrt{\frac{2}{320}} \cos\left(\frac{\pi}{320}(m+0.5)(n+0.5)\right) mlt(m)$$

The window, overlap and add operation uses half of the samples from the current frame's DCT output with half of those from the previous frame's DCT output:

$$y(n) = w(n)u(159 - n) + w(319 - n)u\_old(n) \quad \text{for } 0 \le n \le 159$$

$$y(n + 160) = w(160 + n)u(n) - w(159 - n)u\_old(159 - n) \quad \text{for } 0 \le n \le 159,$$

where:

$$w(n) = \sin\left(\frac{\pi}{640}(n+0.5)\right) \quad \text{for } 0 \le n \le 319$$

The unused half of $u(\,)$ is stored as $u\_old(\,)$ for use in the next frame:

$$u\_old(n) = u(n + 160) \text{ for } 0 \le n \le 159$$

5 C code

The attached C code, which is an integral part of this Recommendation, is divided into a number of files. Those files are now listed.

basic_op.c
coef2sam.c
common.c
count.c
dct4_a.c
dct4_s.c
decode.c
decoder.c
dct4_a.h
encode.c
encoder.c
huff_tab.c
sam2coef.c
tables.c
basic_op.h
count.h
dct4_s.h
defs.h
huff_def.h
huff_tab.h
tables.h
typedefs.h

Once the stand-alone program is compiled into the encoder file, *encode*, and the decoder file, *decode*, then the command line format for using the coder is as follows:

```
encode  bit-stream-type input-audio-file output-bit-stream-file  bit-rate
decode  bit-stream-type input-bit-stream-file output-audio-file  bit-rate
```

where:

bit-stream-type =	0, specifies using the compacted bit stream
bit-stream-type =	1, specifies using the Recommendation G.192 bit stream format for test purposes
input-audio-file =	name of 16 bit PCM audio file to read samples from
output-audio-file =	name of 16 bit PCM audio file to save decoded output
input-bit-stream-file =	name of file to read the input bit stream from
output-bit-stream-file =	name of file to save the encoded bit stream output
bit-rate =	24 000 or 32 000, for 24 kbit/s and 32 kbit/s operation respectively.

6 Flow chart of categorization procedure

This clause describes in detail the procedure for the categorization computation used by both the encoder and decoder. The procedure is divided into twenty-five steps which may be represented as a flow chart.

The following variable is defined and referred to in this clause:

$$category.X[r]$$

is the category assigned to region r for categorization X,

where:

$0 \le X < 16$

$0 \le r < 14$ (*number_of_regions*)

$0 \le category.X[r] \le 7$

STEP (0)

Compute the number of available bits by starting with the allotted number of bits per frame (e.g. 480 for 24 kbit/s operation). Then subtract the number of bits used to represent the amplitude envelope, and subtract the four categorization control bits used to represent the categorization selection. Then modify this number as follows:

if:

$$number\_of\_available\_bits > 320,$$

then:

$$estimated\_number\_of\_available\_bits = 320 + ((number\_of\_available\_bits - 320) * 5/8)$$

(This compensates for differences in the statistics associated with category assignments at different bit rates.)

STEP (1)

Allocate the temporary arrays:

initial_categorization[*number_of_regions*]
max_category[*number_of_regions*]
min_category[*number_of_regions*]
temp_category_balances[32]

Allocate the temporary variables:

offset
delta
expected_bits
max_expected_bits
min_expected_bits
max_pointer
min_pointer
categorization_count

STEP (2)

Initialize:

$$offset = -32$$
$$delta = 32$$

STEP (3)

Compute unbounded category assignments for each region:

$$initial\_categorization[r] = (offset + delta - rms\_index[r])/2$$

STEP (4)

Bound *initial_categorization*[] for each region:

```
if
    $initial\_categorization[r] < 0$
then
    $initial\_categorization[r] = 0$
if
    $initial\_categorization[r] > 7$
then
    $initial\_categorization[r] = 7$
```

STEP (5)

expected_bits_table[8] is a predetermined table containing an average bit count for each category. It is used to compute the expected total bit count for this categorization:

$$expected\_bits = \sum_{r=0}^{number\_of\_regions-1} expected\_bits\_table[initial\_categorization[r]]$$

STEP (6)

```
if
    $expected\_bits \geq estimated\_number\_of\_available\_bits - 32,$
then
    $offset = offset + delta.$
```

STEP (7)

$$delta = delta/2$$

STEP (8)

if

$delta > 0$,

then

go to STEP (3),

otherwise

continue on to STEP (9).

STEP (9)

$initial\_categorization[r] = (offset - rms\_index[\mathrm{r}])/2$

STEP (10)

Bound *initial_categorization*[*r*] for each region as done in STEP (4).

STEP (11)

Compute *expected_bits* for *inital_categorization*[] as in STEP (5).

STEP (12)

Initialize:

$max\_category[r] = initial\_categorization[r]$
$min\_category[r] = initial\_categorization[r]$

$max\_bits = expected\_bits$
$min\_bits = expected\_bits$

$max\_pointer = 16$
$min\_pointer = 16$
$categorization\_count = 1$

STEP (13)

if

$max\_bits + min\_bits \leq 2 * estimated\_number\_of\_available\_bits$

then

go to STEP (14)

otherwise

go to STEP (16)

STEP (14)

For the regions, *r*, for which

$max\_category[r] > 0$

find the region which minimizes the function

$offset - rms\_index[r] - 2 * max\_category[r]$

If there are several regions for which this function is equally small, then set *r* equal to the smallest (i.e. lowest frequency) such region.

STEP (15)

```
$max\_pointer = max\_pointer - 1$
$temp\_category\_balances[max\_pointer] = r$
$max\_bits = max\_bits - expected\_bits\_table[max\_category[r]]$
$max\_category[r] = max\_category[r] - 1$
$max\_bits = max\_bits + expected\_bits\_table[max\_category[r]]$

go to STEP (18)
```

STEP (16)

For the regions, *r*, for which

$$min\_category[r] < 7$$

find the region which maximizes the function

$$offset - rms\_index[r] - 2 * min\_category[r]$$

If there are several regions for which this function is equally large, then set *r* equal to the largest (i.e. highest frequency) such region.

STEP (17)

```
$temp\_category\_balances[min\_pointer] = r$
$min\_pointer = min\_pointer + 1$
$min\_bits = min\_bits - expected\_bits\_table[min\_category[r]]$
$min\_category[r] = min\_category[r] + 1$
$min\_bits = min\_bits + expected\_bits\_table[min\_category[r]]$
```

STEP (18)

$$categorization\_count = categorization\_count + 1$$

if:

$$categorization\_count < 16$$

then:

go to STEP (13)

otherwise:

go to STEP (19)

STEP (19)

Copy *max_category*[] to *category.0* [].

For all regions

$$Category.0\ [r] = max\_category[r]$$

STEP (20)

$$n = 1$$

STEP (21)

Copy the contents of the $(n - 1)$th categorization to the nth categorization:

for all regions

$$category.\,n[r] = category.n - 1[r]$$

STEP (22)

Increment the category assignment for the one region indicated by the array *temp_category_balances*[*max_pointer*]:

```
    $category.n\ [temp\_category\_balances[max\_pointer]] =$
            $category.n\ [temp\_category\_balances\ [max\_pointer]] + 1$
```

STEP (23)

```
    $max\_pointer = max\_pointer + 1$
    $n = n + 1$
```

STEP (24)

```
if:
    $n < 16$
then:
    go to STEP (21)
otherwise:
        END
```

Printed in Switzerland

Geneva, 2000