INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

G.728

Annex J

(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 of speech at 16 kbit/s using low-delay code excited linear prediction

Annex J: Variable bit-rate operation of LD-CELP mainly for voiceband-data applications in DCME

ITU-T Recommendation G.728 – Annex J

(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	
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.728

CODING OF SPEECH AT 16 kbit/s USING LOW-DELAY CODE EXCITED LINEAR PREDICTION

ANNEX J

Variable bit-rate operation of LD-CELP mainly for voiceband-data applications in DCME

Summary

Annex J to Recommendation G.728 defines a 40 kbit/s extension optimized for voiceband data signals of the existing Annex G/G.728 – 16 kbit/s fixed point specification. The main difference between the codec described hereby and the codec described in Annex G/G.728 is the application of a Trellis-Coded Quantization (TCQ) approach to codebook search. The TCQ approach replaces the analysis-by-synthesis approach to codebook search of Recommendation G.728 only in voiceband data (VBD) mode.

The backward adaptation of the predictor achieved in VBD mode is almost identical to the backward adaptation achieved in speech mode (Recommendation G.728). Additionally, the same adaptation cycle is used for both speech mode (Recommendation G.728) and VBD mode. In speech mode, the 40 kbit/s reverts to the LD-CELP of Recommendation G.728.

This annex includes an electronic attachement containing test vectors for implementation verification of Annex J/G.728.

Source

Annex J to Recommendation G.728, 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.728

CODING OF SPEECH AT 16 kbit/s USING LOW-DELAY CODE EXCITED LINEAR PREDICTION

ANNEX J

Variable bit-rate operation of LD-CELP mainly for voiceband-data applications in DCME[1]

(Geneva, 1999)

J.1 Scope

This annex provides the information for the implementation of the codec, and the modifications required in Annex G/G.728 to enable a mode-switch, on a fixed-point arithmetic device.

Annex J operates at transmission rate of 40 kbit/s. The algorithmic delay is five samples long (0.625 ms), which is exactly the same as Annex G and all LD-CELP algorithms recommended in Recommendation G.728. The 40 kbit/s VBD algorithm in Annex J is intended for VBD signal compression transmission, in applications such as DCME. The algorithm allows soft transition to and from the LD-CELP (Recommendation G.728) algorithm, and is also designed to maintain toll quality speech. Annex J is targeted to replace the 40 kbit/s ADPCM mode (Recommendation G.726) in DCME systems incorporating LD-CELP (Recommendation G.728).

J.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.728 (1992), *Coding of speech at 16 kbit/s using low-delay code excited linear prediction*.

[2] ITU-T Recommendation G.728 Annex G (1994), *16 kbit/s fixed point specification*.

[3] ITU-T Recommendation G.763 (1998), *Digital circuit multiplication equipment using ADPCM (Recommendation G.726) and digital speech interpolation*.

J.3 Overview

The codec uses a transmission rate of 40 kbit/s. The algorithmic delay is five samples long, totalling 0.625 ms. The codec can perform a mode-switch every "adaptation-cycle" (2.5 ms).

The main difference between the codec described hereby and the codec described in Annex G/G.728, is the application of a Trellis-Coded Quantization (TCQ) approach to codebook search. The TCQ approach replaces the analysis-by-synthesis approach to codebook search of Recommendation G.728 only in VBD mode.

The backward adaptation of the predictor achieved in VBD mode is almost identical to the backward adaptation achieved in speech mode (Recommendation G.728). Additionally, the same adaptation

[1] This annex includes an electronic attachement containing test vectors for implementation verification.

cycle is used for both speech mode (Recommendation G.728) and VBD mode. In speech mode, the 40 kbit/s reverts to the LD-CELP of Recommendation G.728.

Clause J.4, Algorithm description, provides the full description of the codec's operation. A detailed description of the implementation begins in J.4.3 – Encoder; subclause J.4.4 provides additional details concerning the decoder, and J.4.5 provides the mode-switch details.

J.4 Algorithm description

J.4.1 Codec Structure

The algorithm is based on a RELP structure and is divided into the following blocks (Figure J.1):

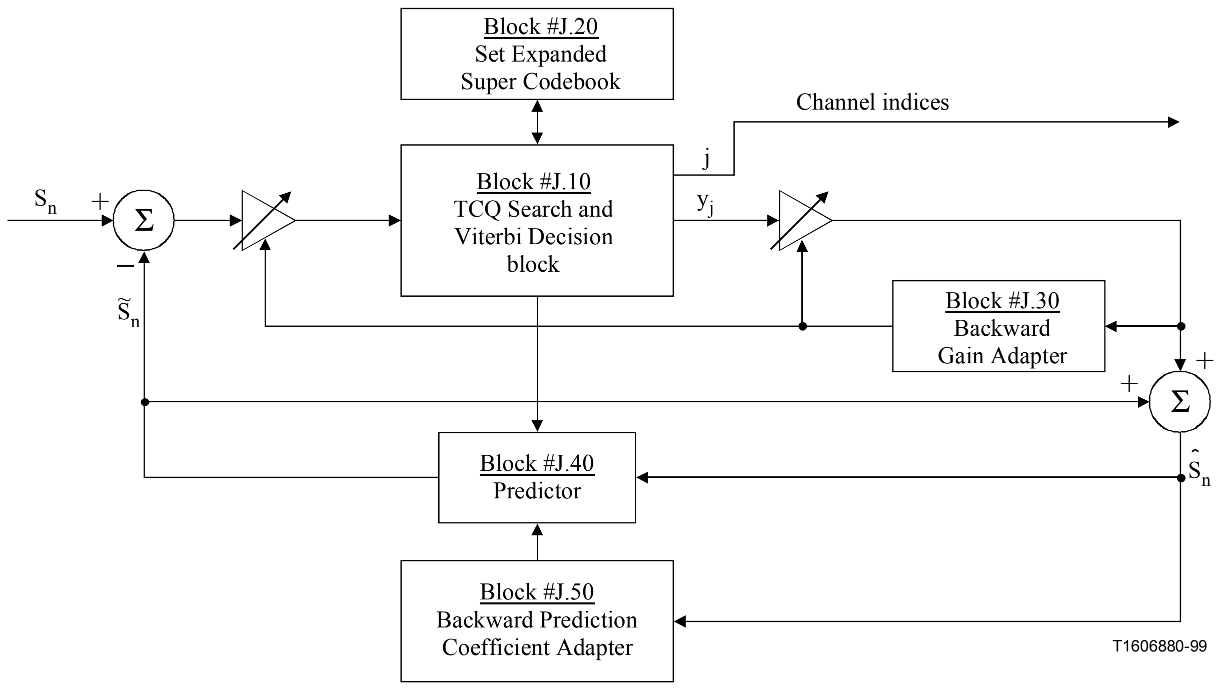

Figure J.1/G.728 – Coder structure

J.4.1.1 Block #J.10 – TCQ Search and Viterbi Decision block

This block performs all the operations required for the TCQ (Trellis-Coded Quantization) algorithm, such as management of the trellis survivors and of the specified reproduction values, calculation and comparison of matrices, and determination of the Viterbi decisions. The encoder releases five channel symbols, as selected by the Viterbi algorithm, from the best survivor for the five source samples.

Figure J.2 shows the state machine that generates the trellis diagram and Figure J.3 shows the trellis diagram. The tables in J.4.6.1, J.4.6.2, J.4.6.3 and J.4.6.4 are derived from the above-mentioned figures, as described below.

Subclause J.4.6.1 provides the allowed path to the **previous** nodes through the trellis, for every node. For example, the allowed previous nodes for the first node (s[0]) are node 0 under branch 0 (b[0]) and node 2 under branch 1 (b[1]).

Subclause J.4.6.2 provides the allowed path to the **next** nodes through the trellis, for every node. For example, the allowed next nodes for the first node (s[0]) are node 0 under branch 0 (b[0]) and node 2 under branch 1 (b[1]).

Subclause J.4.6.3 provides the quantization subset {D0, D1, D2, D3} that is associated with every trellis path. For example, the transition from s[0] to s[0] is associated with subset D0. Transition from s[0] to s[1] is associated with subset D2, and transitions to s[2] and s[3] are not allowed and are, therefore, marked with X.

Subclause J.4.6.4 provides the index bit that labels each transition, and identifies the two branches that emanate from each node. For example, transition from s[0] to s[0] is associated with 0. Transition from s[0] to s[1] is associated with 1 (note that bit 5 is used, and 0x10 is 10h in C), and transitions to s[2] and s[3] are not allowed and are, therefore, marked with X.

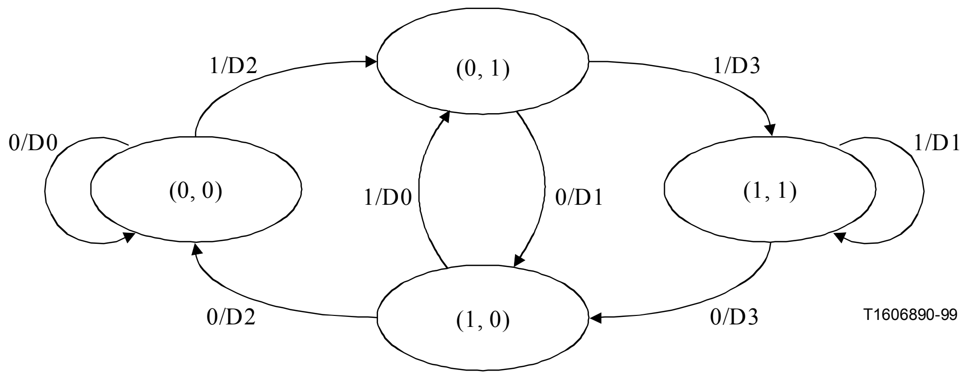

Figure J.2/G.728 – TCQ state machine

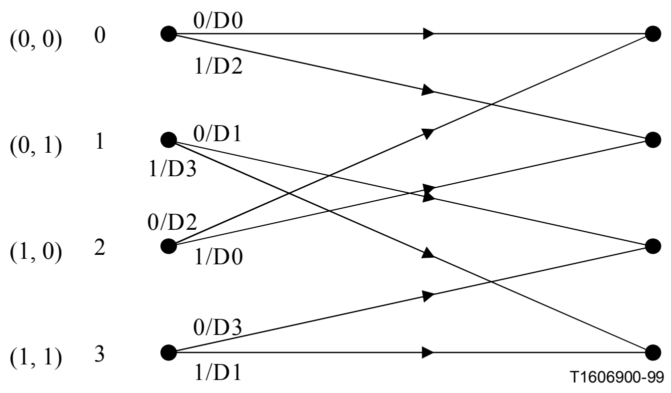

Figure J.3/G.728 – TCQ trellis diagram

J.4.1.2 Block #J.20 – Set Expanded Super Codebook

The Super Codebook is a set-expanded scalar Lloyd-Max quantizer. The 64 output levels are partitioned into four subsets, starting with the most negative point and proceeding towards the most positive point, labelling consecutive points as {D0, D1, D2, D3, ..., D0, D1, D2, D3}. The quantization levels are given in J.4.6.6 and the interval limits are given in J.4.6.5. The levels that belong to subset D0 are shown in the column marked s[0]. D1 levels are shown below s[1], D2 levels are shown below s[2] and D3 are shown below s[3].

J.4.1.3 Block #J.30 – Backward Gain Adapter

The Gain Adaptation scheme is almost identical for both VBD and G.728 speech. The main differences are:

1) In VBD mode, the RMS value of the codebook output values is calculated over a sequence of output levels (quantized residuals) that are specified by the survivor path. The RMS is calculated over a sequence of eight samples. However, unlike the situation in Annex G, where precomputed tables store the log RMS in VBD mode it is necessary to calculate the logarithmic value of the RMS. Equation J.4-1 provides the logarithmic approximation. The coefficients d_0,d_1,d_2,d_3,d_4 are provided in J.4.7 and the detailed description of the logarithmic calculator is provided in J.4.3.17.

$$2 * \log_{10}(x) = d_0 * (x-1) + d_1 * (x-1)^2 + d_2 * (x-1)^3 + d_3 * (x-1)^4 + d_4 * (x-1)^5 \qquad \text{(J.4-1)}$$

   for $1 \leq x < 2$

   The log RMS value replaces the output of the shape and gain codebook, log-gain tables blocks #G.93 and #G.94 (the last two terms in Equation G-14).

2) A smoothing filter is introduced in the log gain loop, to reduce the steady-state oscillation for signals with stationary variance, such as voiceband data waveforms. To overcome both speech and data signals, a Dynamic Locking Quantizer (5) algorithm generates a variable speed adaptation. The DLQ is similar to the DLQ block of Recommendation G.726.

   The input to the DLQ, is the offset removed log-gain d(n). This input is averaged by the weighting filter (J.4.3.18, block #J.14) to produce the locked gain G_L.

   The quantizer is in purely locked state if $a_l = 0$, and in the purely unlocked state if $a_l = 1$. a_l is calculated by comparing between the long-term and the short-term energy of the quantized residuals ET(n) (J.4.3.10, block #J.12). The comparison provides a measure of the constancy of the variance of quantized residuals.

$$G = G_U * \alpha_1 + G_L * (1 - \alpha_1) \qquad \text{(J.4-2)}$$

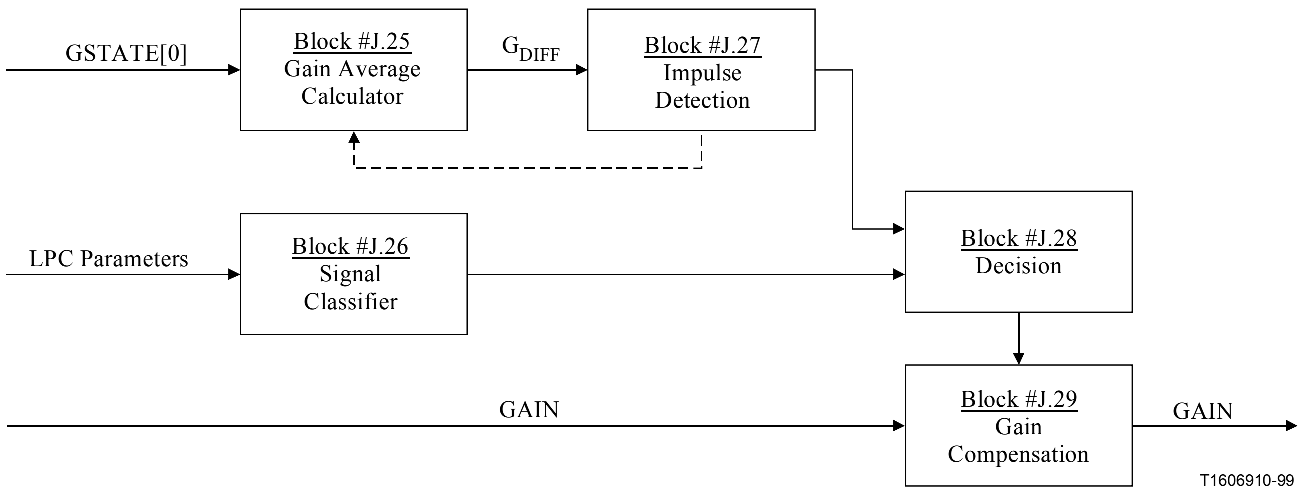

Figure J.4/G.728 – Gain compensation

3) Prediction error impulses might cause quantizer saturation. To avoid this, an additional group of five blocks (see Figure J.4) produces temporal change in quantization gain. These blocks are:

Block #J.25 – Gain Average Calculator

A smoothing filter calculates the average of the gain estimation, G_{ave}, using the most recent vector gain value, GSTATE[0] (J.4.3.12, block #J.25 and Equation J.4-3). The difference between GSTATE[0] and G_{ave} is calculated (G_{diff}) and passed to the Impulse Detection block.

Block #J.27 – Impulse Detection block

This block detects sudden changes in the gain after a predetermined period of steady gain (see J.4.3.14, block #J.27). G_{diff} is compared against a fixed threshold. If G_{diff} is smaller than the threshold for a period greater than a predefined period of time, then the signal is considered "steady". An error impulse is detected if G_{diff} is greater than the threshold and the preceding signal has been "steady".

Block #J.26 – Signal Classifier

During certain VBD transmissions, error impulses are more likely to occur. Thus, upon their detection, gain compensation is maximized. The Signal Classifier block detects these transmissions using the LP coefficient (see J.4.3.13, block #J.26).

Block #J.28 – Decision

The Decision block accepts both the signal classifier block output and the Impulse Detection block output, and activates the Gain Compensation block (see J.4.3.15, block #J.28).

Block #J.29 – Gain Compensation

This block increases the gain factor for a fixed period of time (unless a certain gain factor peak is reached, in which case the period is extended).

$$G_{ave} = G_{const} \times G_{ave} + (1 - G_{const}) \times GSTATE[0] \qquad \text{(J.4-3)}$$

(See J.4.3.16, block #J.29).

J.4.1.4 Block #J.40 – The Predictor Block

The Predictor is a shorter version of the G.728 synthesis filter (block #G.22). The LPC order is 10 taps, instead of the usual 50 taps used in the synthesis filter. The prediction is based on the survivor path (see J.4.3.4, block #J.7) in the following manner: At time n, a prediction of the current sample is formed for each node (see J.4.3.5, block #J.8), using the sequence of reproductions specified by the survivor selected at time n – 1. Using this method, only a one-step scalar prediction is performed, and the prediction does not have to be extended far into the future. This makes the prediction more "localized" than in many other predictive VQ schemes.

J.4.1.5 Block #J.50 – Backward Prediction Coefficient Adapter

The Backward Prediction Coefficient Adapter is almost identical to the Backward Synthesis Filter Adapter (block #G.23). The main differences are as follows:

- Only 10 LPC parameters are calculated. The hybrid windowing module (block #G.49) constantly calculates 51 auto-correlation coefficients, enhancing the performance of data to-voice transitions.
- The Bandwidth Expansion factor of the synthesis filter is now 240/256. The bandwidth expansion coefficients are provided in J.4.8.

J.4.2 Structure of the Decoder

Figure J.5 displays the decoder block diagram. Figure J.6 displays the signal mapping module block #J.60. The sequence of 5 bits-per-sample, received from the decoder, is combined from two types of bit sequences. The signal mapping module block #J.60 separates the combined sequence into two sequences j_{4n}. A sequence of 1 bit per sample is expanded into 2 bits using a convolution coder. The output from the convolution coder specifies the trellis path and selects the proper subset. The subset selection is according to the tables in J.4.6.2 and J.4.6.3. The selection is detailed in J.4.4.2. The remaining 4 bits per sample $j_{0n} \ldots j_{3n}$ are used to select the output level in the selected subset. The output level is scaled in the Backward gain adaptation. The gain scaled output level is then fed to the predictor block #J.40 and produces the reproduction values (see Figure J.5).

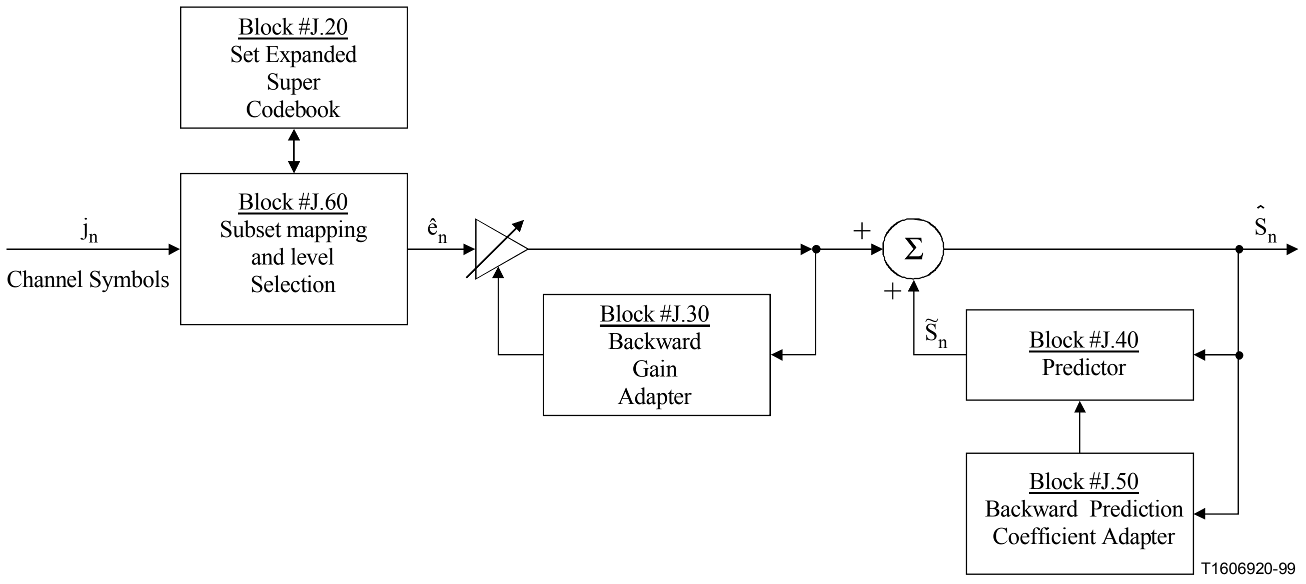

Figure J.5/G.728 – Decoder structure

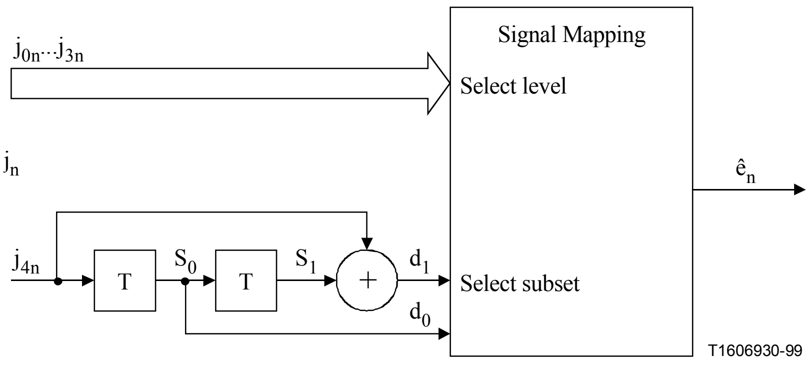

Figure J.6/G.728 – Mapping module (Block #J.60)

J.4.3 Detailed description on the Encoder

This subclause provides a detailed description of the encoder's internal blocks. The description is provided as a top-down design, starting from Figure J.7 which provides a schematic description of the operation of G.728 at various coding rates. The two blocks – block #J.1 and block #J.2 – are derived from Figure 2/G.728 [1]. The two blocks labelled Annex H and Annex G represent the two annexes for Recommendation G.728 (Fixed Point Specification and VBR mode of operation). Block #J.1 represents the mode-switch speech-to-data and data-to-speech. Block #J.2 represents the Voiceband Data mode.

Several C operators (such as: &, *, >>, <<) are used in the description. It is assumed that the reader is familiar with these operators' definitions (Note that the word "**OR**" replaces the C bitwise OR operator "|").

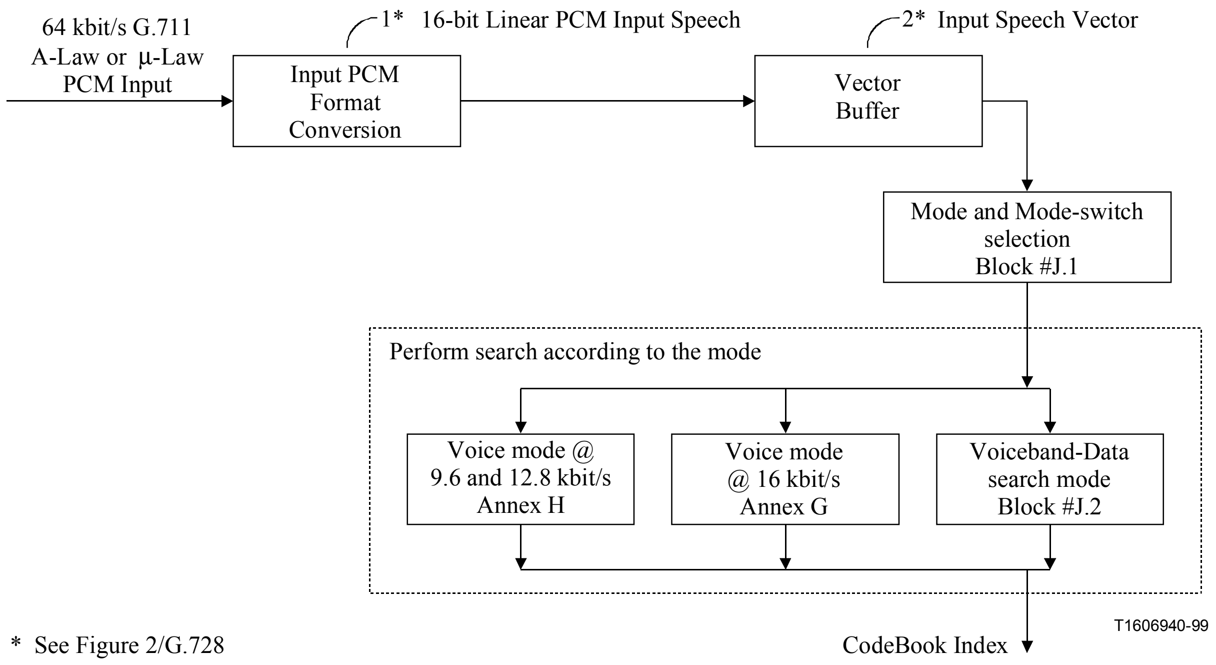
64 kbit/s G.711 A-Law or μ-Law PCM Input
1* 16-bit Linear PCM Input Speech
2* Input Speech Vector
Input PCM Format Conversion
Vector Buffer
Mode and Mode-switch selection Block #J.1
Perform search according to the mode
Voice mode @ 9.6 and 12.8 kbit/s Annex H
Voice mode @ 16 kbit/s Annex G
Voiceband-Data search mode Block #J.2
T1606940-99
* See Figure 2/G.728
CodeBook Index

Figure J.7/G.728 – Schematic Block Diagram of various modes of Recommendation G.728

J.4.3.1 Block #J.2 – Voiceband-Data search mode

Input: S(n)

Output: TCQ_channel_symbols

Operation: Encoding of input vector.

```
| For every Vector (5 samples) perform the following blocks.

CALL BLOCK #J.3  | Trellis search per vector module.
CALL BLOCK #J.4  | Select Best Survivor module.
CALL BLOCK #J.5  | Adaptation module.

CALL BLOCK #J.17 | Initialize next search module.
```

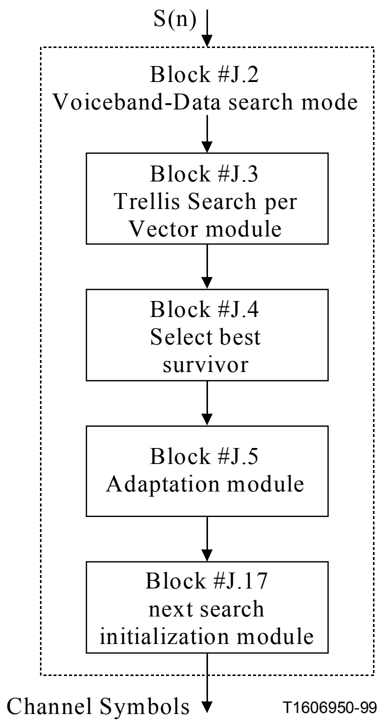

Figure J.8/G.728 – Block #J.2 Voiceband-Data search mode

J.4.3.2 Block #J.3 – Trellis Search per Vector module

Input: block_depth, next_stage

Output: See output of blocks #J.8, #J.9, #J.10

Operation: A sample-by-sample stepping through the trellis diagram.

```
FOR I = 1, 2, ..., IDIM | Do the next line
   CALL BLOCK #J.6 | TCQ transition.

block_depth = next_stage
next_stage = next_stage+1
IF (BLOCK_LEN <= next_stage)
   next_stage = 0
```

J.4.3.3 Block #J.6 – TCQ_transition

Input: S(n), block_depth, next_stage

Output: See output of blocks #J.8, #J.9, #J.10

Operation: A single step through the trellis diagram.

```
| For every sample perform the following blocks.

CALL BLOCK #J.7 | Select Predictor state.
CALL BLOCK #J.8 | Calculate residuals.
CALL BLOCK #J.9 | Select "new" survivor.
CALL BLOCK #J.10 | Calculate reconstructed Signal.
```

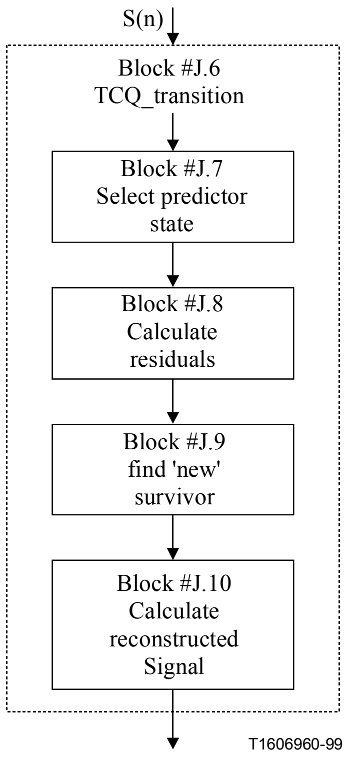

Figure J.9/G.728 – Block #J.6 TCQ_transition

J.4.3.4 Block #J.7 – Select predictor state

Input: block_depth, TCQ_reconstructed_q2

Output: TCQ_predictor_state_q2

Operation: For every node 0, 1, 2, 3 (of Figure J.3), form the predictor state, by tracing back through the reconstructed-levels trellis path.

NOTE – The prediction filter state is composed of two sequences:

Sequence 1: The previously selected reconstructed levels (from the index: block_depth+1, block_depth+2, ..., up to 10), that were selected during the previous search cycle, are valid for all the trellis states (nodes) and are calculated once per sample.

Sequence 2: Each node is associated with a currently reconstructed level sequence (from index: 1, 2, ..., up to block_depth), that is being currently formed in the reconstructed-levels trellis.

This module forms sequence 2 of each node's predictor state.

```
FOR d = 0, 1, ..., (N_STATES - 1) | Do next 6 lines
    IF (0 < block_depth)
        I=0
        src = d
        FOR k = block_depth, block_depth - 1, ..., 1 | Do the next 3 lines
            TCQ_predictor_state_q2[d][i] = TCQ_reconstructed_q2[k][src]
            src = prev_node [k] [src]
            i=i+1
```

J.4.3.5 Block #J.8 – Calculate residuals

Input: block_depth, A, TCQ_STTMP_q2, DLQ_inv_GAIN, DLQ_inv_nls_m, sample

Output: TCQ_predict_sample_q2, TCQ_resid

Operation: For every node 0, 1, 2, 3 (of Figure J.3), calculate the associated predicted value and residuals.

NOTE – Each current state node is associated with a single prediction, while next state nodes are associated with two prediction levels (one for each incoming branch). It is, therefore, preferred to associate the predictions with the current state node (shorter processing time) and to reference them by block_depth.

```
FOR d = 0, 1, ..., (N_STATES - 1) | Do lines till END #J.8

| Calculate the part of the predicted value that is valid (common)
| for all trellis states (nodes). See note of J.4.3.4.

    IF (d == 0) | Do next FOR. Calculate only once, during 1st node.
       TCQ_common_part_pred_q2 = 0
       FOR i = 0, 1, ... (PREDICTOR_ORDER - block_depth-1)
          TCQ_common_part_pred_q2=TCQ_common_part_pred_q2-TCQ_STTMP_q2
[(NFRSZ - 1) - i] * A[i+1+block_depth]
    | End of IF (d == 0).
| FOR each trellis state (node), predict the current sample.
AA0 = TCQ_common_part_pred_q2
FOR i = 0, 1, .., (block_depth - 1) | Do the next line
    AA0 = AA0 - TCQ_predictor_state_q2[d][i] * A[i+1]

AA0 = rnd_int (AA0<<2) | Q2 representation.

TCQ_predict_sample_q2[d] = AA0  | save predicted value.
TCQ_resid[d] = sample - AA0

| Normalize the residuals.

AA0 = TCQ_resid[d] * DLQ_inv_GAIN
AA1 = -1*(DLQ_inv_nls_m-1)
IF (0 <= AA1)
    AA0 = AA0 >> AA1
ELSE
    AA1 = -1*AA1
    AA0 = AA0 << AA1

| Force saturation.

IF (32767L < AA0)
    AA0 = 32767L

IF (AA0 < -32768L)
    AA0 = -32768L
TCQ_resid [d] = AA0 | Save the normalized residual.

END#J.8:
```

J.4.3.6 Block #J.9 – find 'new' survivor

Input: block_depth, next_stage, TCQ_resid

Output: distortion_metric, Q_resid, prev_node, TCQ_channel_indices

Operation: For every next_stage trellis state (node) 0, 1, 2, 3 (Figure J.3), select one of the two incoming branches and mark it as 'new' survivor for that trellis state (node).

NOTE – Each next state node is associated with a single survivor (one of the two incoming branches). Therefore, it is preferred to associate the 'new' survivor with a next state node and to reference it by next_stage pointer.

The complexity (MIPS) consumption of this block is substantial.

```
| For each trellis state, quantize the associated residuals, using the
| subset that labels the two emanating branches (of Figure J.3).

| Perform 8 Quantizations. CALL BLOCK #J.11 8 times.
TCQ_quantize_resid ( TCQ_resid [0], 0, &ch_index[0], &qerror[0] )
TCQ_quantize_resid ( TCQ_resid [0], 2, &ch_index[1], &qerror[1] )
TCQ_quantize_resid ( TCQ_resid [1], 1, &ch_index[2], &qerror[2] )
TCQ_quantize_resid ( TCQ_resid [1], 3, &ch_index[3], &qerror[3] )
TCQ_quantize_resid ( TCQ_resid [2], 2, &ch_index[4], &qerror[4] )
TCQ_quantize_resid ( TCQ_resid [2], 0, &ch_index[5], &qerror[5] )
TCQ_quantize_resid ( TCQ_resid [3], 3, &ch_index[6], &qerror[6] )
TCQ_quantize_resid ( TCQ_resid [3], 1, &ch_index[7], &qerror[7] )

| Calculate the distortion metric associated with each node.

AA0 = (TCQ_resid[0] - qerror[0]) * (TCQ_resid[0] - qerror[0])
AA0 = AA0 >> 2
temp_metric[0] = distortion_metric[0] + AA0

AA0 = (TCQ_resid[0] - qerror[1]) * (TCQ_resid [0] - qerror[1])
AA0 = AA0 >> 2
temp_metric[1] = distortion_metric[0] + AA0

AA0 = (TCQ_resid[1] - qerror[2]) * (TCQ_resid[1] - qerror[2])
AA0 = AA0 >> 2
temp_metric[2] = distortion_metric[1] + AA0

AA0 = (TCQ_resid[1] - qerror[3]) * (TCQ_resid[1] - qerror[3])
AA0 = AA0 >> 2
temp_metric[3] = distortion_metric[1] + AA0

AA0 = (TCQ_resid[2] - qerror[4]) * (TCQ_resid[2] - qerror[4])
AA0 = AA0 >> 2
temp_metric[4] = distortion_metric[2] + AA0

AA0 = (TCQ_resid[2] - qerror[5]) * (TCQ_resid[2] - qerror[5])
AA0 = AA0 >> 2
temp_metric[5] = distortion_metric[2] + AA0

AA0 = (TCQ_resid[3] - qerror[6]) * (TCQ_resid[3] - qerror[6])
AA0 = AA0 >> 2
temp_metric[6] = distortion_metric[3] + AA0

AA0 = (TCQ_resid[3] - qerror[7]) * (TCQ_resid[3] - qerror[7])
AA0 = AA0 >> 2
temp_metric[7] = distortion_metric[3] + AA0

| Select the new survivor for each node.

| node 0
```

```
IF (temp_metric[0] < temp_metric[4])
  distortion_metric[0] = temp_metric[0]
  Q_resid[0] = qerror[0]
  prev_node [next_stage][0] = 0
  TCQ_channel_indices [next_stage] [0] = ch_index[0]
ELSE
  distortion_metric[0] = temp_metric[4]
  Q_resid[0] = qerror[4]
  prev_node [next_stage][0] = 2
  TCQ_channel_indices [next_stage] [0] = ch_index[4]

| node 1

IF (temp_metric[1] < temp_metric[5])
  distortion_metric[1] = temp_metric[1]
  Q_resid[1] = qerror[1]
  prev_node [next_stage][1] = 0
  TCQ_channel_indices [next_stage] [1] = ch_index[1] OR 0x10
ELSE
  distortion_metric[1] = temp_metric[5]
  Q_resid[1] = qerror[5]
  prev_node [next_stage][1] = 2
  TCQ_channel_indices [next_stage] [1] = ch_index[5] OR 0x10

| node 2

IF (temp_metric[2] < temp_metric[6])
  distortion_metric[2] = temp_metric[2]
  Q_resid[2] = qerror[2]
  prev_node [next_stage][2] = 1
  TCQ_channel_indices [next_stage] [2] = ch_index[2]
ELSE
  distortion_metric[2] = temp_metric[6]
  Q_resid[2] = qerror[6]
  prev_node [next_stage][2] = 3
  TCQ_channel_indices [next_stage] [2] = ch_index[6]

| node 3

IF (temp_metric[3] < temp_metric[7])
  distortion_metric[3] = temp_metric[3]
  Q_resid[3] = qerror[3]
  prev_node [next_stage][3] = 1
  TCQ_channel_indices [next_stage] [3] = ch_index[3] OR 0x10
ELSE
  distortion_metric[3] = temp_metric[7]
  Q_resid[3] = qerror[7]
  prev_node [next_stage][3] = 3
  TCQ_channel_indices [next_stage] [3] = ch_index[7] OR 0x10
```

J.4.3.7 Block #J.11 – TCQ_quantize_resid

Input: resid, state

Output: index, qerror

Operation: Quantization of single residual using a single subset.

NOTE – This routine is performed eight times per sample; each quantization is a 4 bit, 16-levels quantization.

```
index = 0
while ( (Xk[state][index] < resid) && (index < Q_CELLS) )| Do the next line
   index = index+1
*qerror = Yk[state][index] | return quantized residual.
```

J.4.3.8 Block #J.10 – Calculate reconstructed Signal

Input: next_stage DLQ_GAIN, DLQ_nls_p_cb_q_m_18, Q_resid, prev_node

Output: TCQ_reconstructed

Operation: For every node 0, 1, 2, 3 (Figure J.3), calculate the reconstructed level.

NOTE – The reconstructed level is referenced by next_stage. See Note to J.4.3.6.

Internal Variables (C-definition)

- int src;
- long int AA0;
- short int nls;

```
       FOR d = 0, 1, ..., (N_STATES - 1)
| For each trellis state (node) presented by the variable d.
| Do the next blocks till END #J.10

src = prev_node [next_stage] [d] | find the previous node.

TCQ_ET [ next_stage ] [ d ] = Q_resid [ d ]

AA0 = Q_resid[d] * DLQ_GAIN

IF (0 <= DLQ_nls_p_cb_q_m_18 )
       AA0 <<= DLQ_nls_p_cb_q_m_18
    ELSE
       AA0 >>= abs (DLQ_nls_p_cb_q_m_18)
    AA0 = AA0 + (TCQ_predict_sample_q2 [src] << 16L)
    TCQ_reconstructed_q2 [next_stage] [d] = AA0 >> 16L

END #J.10:
```

J.4.3.9 Block #J.4 – Select best survivor

Input: distortion_metric, prev_node, TCQ_ET, TCQ_reconstructed_q2, TCQ_channel_indices

Output: TCQ_channel_symbols, ET, TCQ_STTMP_q2

Operation: Select the survivor-node and the associated sequence of channel symbols.

NOTE – This module is performed after the completion of five steps through the trellis path. A sequence of five channel symbols (5-bits-per-sample) should be selected (survivor-node).

Internal Variables (C-definition)

- int src;
- int survivor_node;
- long int min_dist;

```
| shift TCQ_STTMP_q2 Kr samples backward.
    FOR i = TCQ_Kr.. (NFRSZ-1)
       TCQ_STTMP_q2 [i - TCQ_Kr] = TCQ_STTMP_q2[i]
| shift ET backward.
FOR i = TCQ_Kr..(RMS_BUF_LEN-1)
    ET [i - TCQ_Kr] = ET [i]

| Select the node with a minimum quantization distortion, and the
| associated sequence of reconstructed samples.
```

```
min_dist = MAX_NUMBER
FOR d= 0..(N_STATES-1)
    IF (distortion_metric[d] <= min_dist) | Do the next 2 lines
       min_dist = distortion_metric[d]
       survivor_node= d

| Copy the last Kr samples from the
| survivor sequence into TCQ_STTMP_q2

k = 0 | TCQ_Kr == TCQ_Kd as in Kd/Kr=5/5.
src = survivor_node

FOR i = TCQ_DEPTH..(TCQ_Kd - TCQ_Kr)  | Do the next lines
| i is decremented.
| Arrange reproductions in ascending order.

    TCQ_STTMP_q2 [i + (NFRSZ-(TCQ_DEPTH+1))]=TCQ_reconstructed_q2 [k] [src]
    ET[ i - (TCQ_Kd - TCQ_Kr) + (RMS_BUF_LEN - TCQ_Kr) ] = TCQ_ET [k] [src]
    TCQ_channel_symbols [ i - (TCQ_Kd - TCQ_Kr) ] =TCQ_channel_indices [ k ]
    [ src ]
    src = prev_node [k] [src]

    k = k -1
    IF (k < 0)
       k = TCQ_DEPTH
| End of FOR.

| Estimate the gain for the next 5 samples.

CALL BLOCK #J.12 | TCQ Backward Gain Adapter.
```

J.4.3.10 Block #J.12 – TCQ Backward Gain Adapter

Input: next_stage

Output: DLQ_GAIN, DLQ_NLSGAIN

Operation: A once-per-vector calculation of the scaling gain s(n).

NOTE – This block replaces block #20 (of Annex G/G.728) in data mode.

Internal Variables (C-definition)

- int src;
- int survivor_node;
- long int min_dist;

```
| Calculate the log gain RMS value for the last selected path segment.

CALL BLOCK #J.13 | vbd_log_calc_and_lim97

| Calculate the dms and dml DLQ decision parameters.

dms = (dms << 5) - dms
dml = (dml << 7) - dml
dms += RMS_Q11
dml += RMS_Q11
dms >>= 5
dml >>= 7
diff = labs(dms-dml)

IF (diff >= (dml >> 3L))
    | Average Power is variable, non stationary signals such
    | as speech. ap -> 2, and unlock the quantizer.
```

```
   ap_q11 = (ap_q11 * 15 ) >> 4 | ap = ap * 15.0 /16.0
   ap_q11 += (1 Q11 ) >> 3
ELSE
   IF (DLQ_GAIN >> DLQ_NLSGAIN < 10)
      | Idle Channel Conditions, ap -> 2, and unlock the quantizer.

      ap_q11 = (ap_q11 * 15 ) >> 4
      ap_q11 += (1 Q11 ) >> 3
   ELSE
      | Average Power is constant, stationary signals, such as VBD
      | ap -> 2, and lock the quantizer.

      ap_q11 = (ap_q11 * 15 ) >> 4

IF (1 Q11 < ap_q11)
   al_q11 = 1 Q11
ELSE
   al_q11 = ap_q11

FOR i = 0..NUPDATE | save GSTATE as in Annex G/G.728.
   GTMP[i] = GSTATE[i]
| Predict the gain.

| BLOCK #46, P. 39 of Annex G of Recommendation G.728 is the
| LOG-GAIN LINEAR PREDICTOR.

CALL BLOCK #G.46 | Log-gain linear prediction.
CALL BLOCK #G.98 | log gain limiter 98, part of blocks 46 AND 47, P. 39.
CALL BLOCK #J.14 | log_gain_weighting

GAIN = after_limiter_98 | GAIN is the averaged log-gain.

CALL BLOCK #G.99 | add_log_offset, part of blocks 46 AND 47, P. 39.
CALL BLOCK #G.48 | Inverse logarithmic Calculator.

DLQ_GAIN = GAIN
DLQ_NLSGAIN = NLSGAIN
CALL BLOCK #J.15 | GAIN_inverse
```

J.4.3.11 Block #J.13 – vbd_log_calc_and_lim97

Input: ET

Output: GSTATE[0], RMS_Q11

Operation: To Calculate the log gain RMS value.

NOTE – This block replaces blocks 67, 39, and 40 (1-Vector delay, RMS calculator and logarithmic calculator) in the floating point version, and blocks 93, 94, 96, 97 of Annex G/G.728, in the fix point version.

Internal Variables (C-definition)

- long int AA0;
- long int AA1;

```
AA0 = 0
FOR i=(RMS_BUF_LEN - 1), (RMS_BUF_LEN - 2),..., 1
   AA0 += ET [i] * ET [i]
AA0 += ET[0] * ET[0]

| Divide by RMS_BUF_LEN, and scale AA0 from Q22 to Q11.
AA0 = (AA0 + 1 Q13) >> 14

RMS_Q11 = AA0 | 32 bit operation.
```

```
AA0 = calc_10log10(AA0) | BLOCK #J.16 is implemented as a function.

AA0 = AA0 + (after_limiter_98 << 2) | 32 bit operation.

AA0 = AA0 >> 2 | scale to Q9 format.

| block 97 limit GSTATE to -32dB.

IF (AA0 < -16384)
   AA0 = -16384

| If immediately after transition, than use the GSTATE of the speech mode.
IF (speech_to_vbd_transition == 1)
   speech_to_vbd_transition = 0
ELSE
   GSTATE[0]=AA0
```

J.4.3.12 Block #J.25 – Gain Average Calculator

Input: GSTATE[0], UNSTEADY

Output: G_DIFF, G_CNT

Operation: Calculation of a quasi-average value of the gain.

Internal Variables (C-definition)

- long int G_AVE

```
GDIFF=GSTATE[0]-G_AVE;
IF UNSTEADY=1           | Do the next 3 lines
   G_AVE=GSTATE[0]
   G_CNT=0
   UNSTEADY=0
ELSE                    | Do the next 3 lines
   IF GDIFF<G_TRS | Do the next 2 lines
      G_AVE=((G_AVE<<G_CONST-G_AVE+GSTATE[0])>>G_CONST
      G_CNT++
```

J.4.3.13 Block #J.26 – Signal Classifier

Inputs: ATMP

Outputs: GC_SC_FLAG

Operation: Detection of narrow bandwidth signal.

Internal Variables (C-definition)

- int GC_ATMP_SUM
- int GC_ATMP1

```
GC_ATMP_SUM=0
GC_ATMP1=ATMP[1]
FOR I=2,3,...LPC+1,          |Do the next line
   GC_ATMP_SUM=GC_ATMP_SUM+ABS(ATMP[I])

IF ((GC_ATMP_SUM*ATMP_CONST)>>3)<ABS(GC_ATMP1)
    GC_SC_FLAG=1
ELSE
    GC_SC_FLAG=0
```

J.4.3.14 Block #J.27 – Impulse Detection

Inputs: GDIFF, GC_LEN, GC_SC_FLAG

Outputs: GC_ID_FLAG

Operation: Search for a sharp gain increase after a predefined period of steady gain.

```
IF GDIFF > G_TRS              | Do the next 4 lines
   IF G_CNT>GC_LEN
      GC_ID_FLAG=1
   ELSE
      GC_ID_FLAG=0
```

J.4.3.15 Block #J.28 – Gain Compensation Decision

Inputs: GC_ID_FLAG, GC_SC_FLAG

Outputs: GC_FLAG, GC_CNT, UNSTEADY, GC_NLS_LIMIT, GC_COMPENSATION

Operation: Decision logic for the Gain Compensation block.

```
GC_LEN=0
GC_NLS_LIMIT=16383
GC_COMPENSATION=0
IF GC_SC_FLAG=1         | Do the next 3 lines
   GC_LEN=GC_CNT_INIT
   GC_NLS_LIMIT=GC_NLS_LIMIT_INIT
   GC_COMPENSATION=GC_COMPENSATION_INIT
IF GC_ID_FLAG=1         | Do the next 3 lines
   UNSTEADY=1
   GC_FLAG=1            | GAIN COMPENSATION FLAG
   GC_CNT=GC_LEN
```

J.4.3.16 Block #J.29 – Gain Compensation

Inputs: GC_FLAG, DLQ_NLSGAIN, GC_CNT, GC_COMPENSATION, GC_NLS_LIMIT

Outputs: GC_FLAG

Operation: Decrease DLQ_NLSGAIN by a fixed value for a predefined period of time.

```
IF GC_FLAG=1            | Do the next 7 lines
   IF DLQ_NLSGAIN>GC_NLS_LIMIT     | Do the next 6 lines
      GC_CNT=GC_CNT-1
      DLQ_NLSGAIN=DLQ_NLSGAIN-GC_COMPENSATION
      IF DLQ_NLSGAIN< GC_NLS_LIMIT
         DLQ_NLSGAIN= GC_NLS_LIMIT
      IF GC_CNT=0
         GC_FLAG=0
```

J.4.3.17 Block #J.16 – logarithmic calculator

Input: AA0. Q11 format. (C-definition: long int AA0)

Output: logarithmic value

Operation: Approximation of the logarithmic function.

NOTE – The approximation is valid for $1 \leq x < 2$.

Internal Variables (C- definition)

- long int T;
- short int exp;

```
IF (AA0 < 1)
AA0 = 0 | illegal input number clipping to 0dB.
ELSE
| Scale the input number to the range 1Q14..2Q14
exp = 3 | Q3 is the difference Q14 - Q11
WHILE ( AA0 < 1 Q14 ) | Do the next two lines
AA0 = (AA0 << 1)
exp -= 1

WHILE (2 Q14 <= AA0) | Do the next two lines
AA0 = (AA0 >> 1L)
exp += 1

T = AA0 - 1 Q14
T <<= 1 | Translate T from Q14 to Q15
AA0 = log_pol[LOG_POL_ORDER-1]

FOR i = (LOG_POL_ORDER-1), (LOG_POL_ORDER-2).., 1
AA0 = ((T * AA0) * 2 + (log_pol[i-1] << 16) + 1 Q15) >> 16

AA0 = ( (AA0 * T) * 2 + 1 Q15) >> 16

AA0 = AA0 >> 4 | Translate AA0 to Q11.

AA0 = AA0 * 5 | Divide AA0 by 2 and multiply by 10 to get dB units.

| Add the mantisa, and calculate the 10 log dB value of the input

AA0 = AA0 + (log_2 * exp) >> 2

return(AA0)
```

J.4.3.18 Block #J.14 – log_gain_weighting

Input: after_limiter_98, GAIN_state, al_q11

Output: after_limiter_98

Operation: To average the log-gain and produce the variable adaptation speed.

Internal Variables (C-definition)

- long int AA0, AA1 | 32 bit accumulators.
- short int locked, unlocked;

```
AA0 = (long)GAIN_state
AA1 = AA0 << 6L | 63/64 iir.
AA1 = AA1 - AA0

| Add AA1 (Q9 format) to after_limiter_98 and round the result to 16 bit Q9.

AA0 = after_limiter_98
AA1 = AA1 + AA0

AA0 = AA1 >> 6L | 63/64 iir.

locked = AA0
unlocked = after_limiter_98

AA0 = locked * ( 1 Q11 - al_q11)
AA0 = AA0 + unlocked * al_q11
AA0 >>= 11

after_limiter_98 = AA0
GAIN_state = AA0
```

J.4.3.19 Block #J.15 – GAIN_inverse

Input: DLQ_GAIN, DLQ_NLSGAIN

Output: DLQ_inv_GAIN, DLQ_nls_p_cb_q_m_18, DLQ_inv_nls_m

Operation: Inversion of the gain.

Internal Variables (C-definition)

- long int NUM | constant 16384: 1 in the format Q14
- long int NUMNLS | constant 14

```
DLQ_nls_p_cb_q_m_18 = 18 - CODEBOOK_Q - DLQ_NLSGAIN |

| Initialize the numerator for inversion.

NUM  = 16384
NUMNLS = 14

divide ( NUM, NUMNLS, DLQ_GAIN, DLQ_NLSGAIN, &DLQ_inv_GAIN, &inv_nls) | The
divide is function Annex G/G.728.

DLQ_inv_nls_m = - 2 - inv_nls + CODEBOOK_Q + 1
```

J.4.3.20 Block #J.5 – Adaptation Module

Operation: To perform the adaptation cycle.

NOTE – The same adaptation cycle is used for data and speech, and only the translation of a few variables to Annex G/G.728 input adaptation format is required before each adaptation phase.

Internal Variables (C-definition)

- long int nls
- short int min_nls;

```
ICOUNT = ICOUNT + 1
IF (ICOUNT > NUPDATE)
    ICOUNT = 1

IF (ICOUNT == 4)
   FOR k = 0, 1, .., (NUPDATE - 1)
      | Prepare STTMP & NLSSTTMP for autocore routine,-HW_s.

      VSCALE( &TCQ_STTMP_q2[k * IDIM], IDIM,
               IDIM, 12, &STTMP[k * IDIM], &NLSSTTMP[k])
      NLSSTTMP[k] = NLSSTTMP[k] + 2 | Q2 correction.

   CALL BLOCK #G.49 | HW_s

IF (ICOUNT == 2)
   IF (ILLCOND == 0)
      durbin (RTMP, ATMP, 10) | BLOCK #G.50
      IF (DurbinFaultFlag == 0)
         CALL BLOCK #J.26 | signal classifier
         CALL BLOCK #J.51 | bandexpand51_vbd()
      ELSE
         DurbinFaultFlag = 0
         FOR I=1, 2,..LPC
            ATMP[i]=A[i]
```

```
IF (ICOUNT == 1)
   CALL BLOCK #G.43 | HW_gain
   IF ( ILLCOND == 0 )
      durbin(R, GPTMP, LPCLG) | BLOCK #G.44
   IF (DurbinFaultFlag == 0)
      CALL BLOCK #G.45 | bandexpand45
   ELSE
      DurbinFaultFlag = 0

IF (ICOUNT == 3)
   FOR i = 1, 2.., PREDICTOR_ORDER
      A[i] = ATMP[i]
```

J.4.3.21 Block #J.51 – Bandwidth Expansion module

Input: See block #G.51

Output: See block #G.51

Operation: See block #G.51.

NOTE – This block is identical to block #G.51, with the exception that the array FACV_vbd replaces FACV, which has 10 non-zero elements.

J.4.3.22 Block #J.17 – next search initialization module

Input: survivor_node

Output: distortion_metric

Operation: Set metrics for a new search.

```
FOR i = 0, 1, ..,(N_STATES-1)
    distortion_metric[i] = (MAX_NUMBER >> 2)

| Force the next selected path to pass through the survivor node.
distortion_metric[survivor_node] = 0L

block_depth = 0
next_stage = 1
```

J.4.4 Detailed description on the Decoder

This section provides a detailed description of the decoder.

J.4.4.1 Block #J.20 – Decoder module

Input: channel_symbol

Output: reconstructed_sample_q2

Operation: Perform the per-vector decoding operations.

```
For I = 1, 2, ..., IDIM | Do the next line
    CALL BLOCK #J.21 | Decoder transition module.

CALL BLOCK #J.12 | TCQ Backward Gain Adapter
CALL BLOCK #J.5 | Adaptation Module.
```

J.4.4.2 Block #J.21 – Decoder transition module

Input: channel_symbol

Output: reconstructed_sample_q2

Operation: Perform the per sample decoder operations.

NOTE – The Prediction is performed over the Encoder's predicted value for node 0.

Internal Variables (C-definition)

- short int next_state_label;
- short int level_selector;
- short int subset;
- short int next_state;
- short int codebook_level;
- long int AA0;

```
TCQ_predict_sample_q2[0] = 0 | Use the Encoder's node 0 space.

FOR i = 0, 1,..,(PREDICTOR_ORDER-1)
    TCQ_predict_sample_q2[0] -= TCQ_STTMP_q2 [(NFRSZ - 1) - i] * A[i+1]

FOR i = 0, 1,..,(NFRSZ-1)
    TCQ_STTMP_q2[i] = TCQ_STTMP_q2[i + 1]

TCQ_predict_sample_q2[0] =
    rnd_int ( TCQ_predict_sample_q2[0] << 2)

| Separate the index into 2 sets the next state label and
| the level selector.

next_state_label = (channel_symbol >> BITS_PER_LEVEL)
level_selector = (channel_symbol & LEVEL_MASK)
next_state = TCQ_next_state [TCQ_decoder_state] [next_state_label]
subset = TCQ_trans_from_src_to_dst [TCQ_decoder_state] [next_state]
TCQ_decoder_state = next_state

codebook_level = Yk[subset][level_selector] | Get the codebook level

FOR i = 1, 2, ..,(RMS_BUF_LEN - 1)
    ET[i-1] = ET[i]

ET[RMS_BUF_LEN - 1] = codebook_level

AA0 = codebook_level * DLQ_GAIN

IF (0 <= DLQ_nls_p_cb_q_m_18)
    AA0 <<= DLQ_nls_p_cb_q_m_18
ELSE
    AA0 >>= abs (DLQ_nls_p_cb_q_m_18)

AA0 += (TCQ_predict_sample_q2[0] << 16L)
TCQ_STTMP_q2 [(NFRSZ-1)] = AA0 >> 16L
reconstructed_sample_q2 = AA0 >> 16L
```

J.4.5 Detailed description of the mode-switch modules

This section provides a detailed description of the mode-switch modules.

J.4.5.1 Block #J.18 – speech to data transition module

Input: NLSSTATE, STATELPC, GAIN, NLSGAIN, IAQ_for_VBD, ATMP_for_VBD

Output: TCQ_STTMP_q2, DLQ_GAIN, DLQ_NLSGAIN, IAQ_for_VBD, ATMP_for_VBD

Operation: To perform the speech to data transition.

```
| Translate 20 elements in SBFL format of STATELPC and NLSSTATE
| to the Q2 format of TCQ_STTMP_q2.
| Note that the opposite ordering of STATELPC, NLSSTATE and
| TCQ_STTMP_q2.

I = 0
FOR J = 0, 1, 2, 3
    FOR L = 0, 1,..,4
        k = NLSSTATE [9] - 2
        IF (k<0)
            k = -k
            TCQ_STTMP_q2[NFRSZ - 1 - I] = STATELPC [I] << k
        ELSE
            TCQ_STTMP_q2[NFRSZ - 1 - I] = STATELPC [I] >> k
        I = I + 1

| GAIN VARIABLES.

speech_to_vbd_transition = 1

GAIN_state = after_limiter_98

dms = 0
dml = 0
ap_q11 = 0

DLQ_GAIN = GAIN
DLQ_NLSGAIN = NLSGAIN

CALL BLOCK #J.15 | GAIN_inverse()

| Use the previously saved 10 LPC parameters (saved during last
| adaptation cycle of speech-mode).

FOR I = 1,2, .., PREDICTOR_ORDER
    ATMP[i] = ATMP_for_VBD[i]

IAQ = IAQ_for_VBD
CALL BLOCK #J.51 | bandexpand51_vbd

FOR I = 1, 2,.., PREDICTOR_ORDER
    A[i] = ATMP[i]

GC_FLAG=0;    | Gain Compensation
G_CNT=0;      | Gain Compensation
```

J.4.5.2 Block #J.19 – data to speech transition module

Input: See description below

Output: See description below

Operation: To perform the data to speech transition.

Internal Variables (C-definition)

- long int temp[NFRSZ]

```
| Reinitialize the perceptual weighting internal variables.
| (Only in the Encoder).

AWP[0]=16384
FOR I=1, 2, ..,LPCW
    AWP[i]=0
AWZ[0]=16384
FOR I = 1, 2, ..,LPCW
    AWZ[i]=0
FOR I = 0, 1, 2, ..(NFRSZ-1)
    STMP[i]=0
FOR I = 0, 1,..,(N3weight-1)
    SBW[i]=0

FOR I = 0, 1, ..,LPCW
    REXPW[i]=0
NLSREXPW= 31

| Reinitialize the post filter internal variables.
| (Only in the decoder).

ILLCONDP = 1
AP[0] = 16384
FOR I =1, 2, ..,10
    AP[i]=0
AZ[0] = 16384
FOR I =1, 2,..,10
    AZ[i]=0
FOR I = 0,1,..,59
    DEC[i]=0
FOR I = 0, 1,..,239
    D[i]=0
FOR I = 0, 1,..,9
    STLPCI[i]=0
FOR I = 0, 1, ..,9
    STPFFIR[i]=STPFIIR[i]=0
FOR I =0, 1, 2, 3
    LPFFIR[i]=0
    LPFIIR[i]=0
FOR I = 0, 1, .., 244
    SST[i]=0
SCALEFIL= 16384
B=0
GL=16384
GLB = 0
TILTZ=0
APF[0] = 16384
FOR I = 1, 2, ..,10
    APF[i]=0
PF_delay = 100
KP=KP1=50
```

```
FOR I =11, 12, .., LPC
    A[i]=0
    ATMP[i]=0

FOR I = 0, 1, ..,(NFRSZ-1)
    temp[i] = TCQ_STTMP_q2[(NFRSZ-1) - i]

FOR I = 0, 1, 2, 3
    VSCALE( &temp[i * 5], 5, 5, 12,&STATELPC[i * 5], &NLSSTATE[9-i] )
    NLSSTATE[9-i] += 2 | Q2 Correction.

FOR I = NFRSZ, NFRSZ+1, .., (LPC - 1)
    STATELPC [i] = 0
FOR I = 0, 1, .., 5
    NLSSTATE [i] = 16

| Update several LD-CELP internal blocks.
| These blocks are usually updated during the third adaptation phase
| ( ICOUNT == 3), but they are needed from phase 1.

CALL BLOCK #G.12 | Impulse response vector calculation.
CALL BLOCK #G.14 | Shape codevector convolution and
                 | energy calculation.

| GAIN VARIABLES.

CALL BLOCK #J.13 | vbd_log_calc_and_lim97()
vbd_to_speech_transition = 1
```

J.4.5.3 Block #J.22 – Modifications in the backward vector gain adapter of Annex G/G.728

Operation: The following lines replaces blocks #G.93, #G.94, #G.96 and #G.97.

NOTE – During data-to-speech transition the one index delay values of Gain and Shape codebook index are not available. therefore, GSTATE[0] is calculated during the transition.

```
IF ( vbd_to_speech_transition == 1)
    vbd_to_speech_transition = 0
ELSE
    | default operation of Annex G/G.728.
    | Perform blocks #G.93, #G.94, #G.96 and #G.97.
```

J.4.5.4 Block #J.23 – Modifications in the application of the post filter of Annex G/G.728

Operation: To bypass the post-filter for the first 500 samples (62.5 ms) after the transition, and to use the reconstructed signal as the decoder output for that period. After 62.5 ms, when the post-filter is ready for use, its output is used as the decoder's output.

```
IF (PF_delay == 0)
  FOR I =0, 1, .., (IDIM-1)
    ST[i]=SPF[i]<<1
ELSE
  PF_delay--
  FOR I = 0, 1, ..,(IDIM-1)
    ST[i]=ST[i]<<(16-NLSST+3)
    ST[i]=rnd_int(ST[i])
```

J.4.5.5 Block #J.24 – Modifications in Annex G/G.728 required for saving of interim LPC parameters

Operation: To save the interim 10 LPC parameters. The LPC parameters are required for speech-to-data transition.

NOTE – The 10 interim LPC parameters should be saved during the execution of block #G.23. The execution of the durbin block should be halted (as described, in Annex G/G.728 for the Post filter coefficients, APF). The coefficients should be saved, and then the durbin block should be resumed.

```
IF (DurbinFaultFlag == 0)
    FOR I =1, 2,.., 10
        ATMP_for_VBD[I]=ATMP[I];
    IAQ_for_VBD = IAQ;
```

J.4.6 Trellis transition tables

This section provides the trellis transition tables. The tables describe a realization of the minimal feedback free convolution encoder.

J.4.6.1 TCQ_prev_state

This table provides the previous (source) trellis state (node), per trellis branch, for every trellis state (node), as shown in Figures J.2 and J.3.

C-definition: int TCQ_prev_state[N_STATES][N_BRANCHES]

TCQ_prev_state		
Trellis State (Node)	**prev_state**	
	b[0]	b[1]
s[0]	0	2
s[1]	2	0
s[2]	1	3
s[3]	3	1

J.4.6.2 TCQ_next_state

This table provides the next (target) trellis state (node) per trellis branch, for every trellis state (node), as shown in Figures J.2 and J.3.

C-definition: int TCQ_next_state[N_STATES][N_BRANCHES];

TCQ_next_state		
Trellis State (Node)	**next_state**	
	b[0]	b[1]
s[0]	0	1
s[1]	2	3
s[2]	0	1
s[3]	2	3

J.4.6.3 TCQ_trans_from_src_to_dst

This table provides the codebook subset associated with a transition, as presented in Figures J.2 and J.3.

C-definition: int TCQ_trans_from_src_to_dst [N_STATES][N_STATES];

TCQ_trans_from_src_to_dst				
Trellis State (Node)	**transition label**			
	s[0]	s[1]	s[2]	s[3]
s[0]	0	2	X	X
s[1]	X	X	1	3
s[2]	2	0	X	X
s[3]	X	X	3	1

J.4.6.4 TCQ_index_from_src_to_dst

This table provides the channel index bit that identifies the transition, as presented in Figures J.2 and J.3, to the decoder.

C-definition: int TCQ_index_from_src_to_dst [N_STATES][N_STATES];

TCQ_index_from_src_to_dst				
Trellis State (Node)	**channel index bit**			
	s[0]	s[1]	s[2]	s[3]
s[0]	0	0x10	X	X
s[1]	X	X	0	0x10
s[2]	0	0x10	X	X
s[3]	X	X	0	0x10

J.4.6.5 Xk – Quantizer limits

This table provides the interval limits of the super codebook in Q11 format. The first column displays the cell index; the following columns display the levels for each subset.

C-definition: int Xk [N_STATES][Q_CELLS];

Xk				
index	**Codebook Limits**			
	s[0]	s[1]	s[2]	s[3]
0	–9 547	–8 509	–7 779	–7 191
1	–6 690	–6 246	–5 845	–5 477
2	–5 134	–4 812	–4 507	–4 217
3	–3 939	–3 672	–3 414	–3 164
4	–2 921	–2 684	–2 453	–2 226
5	–2 003	–1 783	–1 567	–1 353
6	–1 141	–931	–723	–515

Xk				
index	Codebook Limits			
7	–309	–103	103	309
8	515	723	931	1 141
9	1 353	1 567	1 783	2 003
10	2 226	2 453	2 684	2 921
11	3 164	3 414	3 672	3 939
12	4 217	4 507	4 812	5 134
13	5 477	5 845	6 246	6 690
14	7 191	7 779	8 509	9 547
15	32 767	32 767	32 767	32 767

J.4.6.6 Yk – Quantizer levels

This table provides the quantization levels of the super codebook in Q11 format. The first column displays the cell index; the following columns display the levels for each subset.

C-definition: int Yk [N_STATES][Q_CELLS];

Yk				
index	Codebook Levels			
	s[0]	s[1]	s[2]	s[3]
0	–11 502	–9 955	–8 962	–8 209
1	–7 592	–7 062	–6 595	–6 174
2	–5 788	–5 430	–5 095	–4 780
3	–4 480	–4 194	–3 919	–3 655
4	–3 399	–3 151	–2 910	–2 674
5	–2 444	–2 218	–1 996	–1 777
6	–1 562	–1 348	–1 138	–928
7	–721	–514	–308	–103
8	103	308	514	721
9	928	1 138	1 348	1 562
10	1 777	1 996	2 218	2 444
11	2 674	2 910	3 151	3 399
12	3 655	3 919	4 194	4 480
13	4 780	5 095	5 430	5 788
14	6 174	6 595	7 062	7 592
15	8 209	8 962	9 955	11 502

J.4.7 The Coefficients of the logarithmic calculator polynomial

This table provides the coefficients of the polynomial used in the logarithmic calculator.

C-definition: int log_poly[LOG_POL_ORDER];

log_poly		
Index	**Floating Point presentation**	**Fix Point presentation Q15**
0	0.8678284	28 437
1	–0.4255677	–13 945
2	0.2481384	8 131
3	–0.1155701	–3 787
4	0.0272522	893

J.4.8 Bandwidth Expansion Coefficients for data mode

This table provides the bandwidth expansion coefficients for the data mode.

NOTE – There are only 10 non-zero elements.

C-definition: int FACV_vbd[LPC];

Table J.1/G.728 – Bandwidth expansion coefficients

FACV_vbd	
index	**Coefficient**
1	15 360
2	14 400
3	13 500
4	12 656
5	11 865
6	11 124
7	10 428
8	9 777
9	9 166
10	8 593
11-50	0

J.4.9 Internal Blocks

Table J.2 contains a list of the internal blocks.

Table J.2/G.728 – Internal Blocks

Block ID	Block Name
Block #J.1	Mode and Mode-switch selection
Block #J.2	Voiceband-Data search mode
Block #J.3	Trellis Search per Vector module
Block #J.4	Select best survivor
Block #J.5	Adaptation Module
Block #J.6	TCQ_transition
Block #J.7	Select predictor state
Block #J.8	Calculate residuals
Block #J.9	find 'new' survivor
Block #J.10	Calculate reconstructed Signal
Block #J.11	TCQ_quantize_resid
Block #J.12	TCQ Backward Gain Adapter
Block #J.13	vbd_log_calc_and_lim97
Block #J.14	log_gain_weighting
Block #J.15	GAIN_inverse
Block #J.16	calc_10log10 (logarithmic calculator)
Block #J.17	next search initialization module
Block #J.18	speech to data transition module
Block #J.19	data to speech transition module
Block #J.20	Decoder Module
Block #J.21	Decoder transition module
Block #J.22	Modifications in the backward vector gain adapter of Annex G/G.728
Block #J.23	Modifications in the application of the post filter of Annex G/G.728
Block #J.24	Modifications in Annex G/G.728 required for saving of interim LPC parameters
Block #J.25	Gain Average Calculator for the Gain Compensation Module
Block #J.26	Signal Classifier for the Gain Compensation Module
Block #J.27	Impulse Detection for the Gain Compensation Module
Block #J.28	Decision block for the Gain Compensation Module
Block #J.29	Gain Compensation for the Gain Compensation Module
Block #J.51	Bandwidth Expansion module

J.4.10 Internal Processing Variables and Constants

Tables J.3 and J.4 provide a list and description on the internal variables and constants.

Table J.3/G.728 – Internal Processing Variables

Name	Array Index Range	Fixed Point Format	Description
after_limiter_98			Q9 input of adder block #G.99
al_q11	1	Q11	Quantizer locking factor
ap_q11	1	Q11	Quantizer locking factor
ATMP_for_VBD	1..11	Q13/Q14/Q15	Temporary buffer for LPC parameters
block_depth	1	Q0	Points at current Trellis step (time n)
ch_index	0..7	Q0	Temporary memory of the selected index
distortion_metric	[0..(N_STATES−1)]	32 bit Q20	Accumulated distortion metric, per trellis state (node)
DLQ_GAIN	1	SFL	Linear Excitation gain, equivalent to GAIN in Annex G
DLQ_inv_GAIN	1	SBFL	Inverted GAIN for VBD
DLQ_inv_nls_m	1	SBFL	NLS for the inverted VBD GAIN
DLQ_nls_p_cb_q_m_18	1	Q0	NLS for linear gain (+ Q format correction offset), equivalent to GAIN in Annex G
dml	1	Q11	Quantized residuals' long term energy
dms	1	Q11	Quantized residuals' short term energy
ET	[0..(RMS_BUF_LEN−1)]		Memory of quantized residuals
FACV_vbd	---	---	See FACV of Annex G
GAIN_state	1	Q9	Weighting filter memory
IAQ_for_VBD	1	Q0	Durbin's recursion precision flag for ATMP
next_stage	1	Q0	Points at the next Trellis step (time n+1)
prev_node	[0..(BLOCK_LEN−1)]*[0..(N_STATES−1)]	Q0	Trellis memory of previous states (nodes)
Q_resid	[0..(N_STATES−1)]	Q11	Temporary memory of the quantized residuals
qerror	1	Q11	Temporary memory of quantized residuals
RMS_Q11	1	Q11	RMS of ET
sample	1	Q2	Input sample, single element of S
speech_to_vbd_transition	1	Q0	Speech-to-VBD transition flag
survivor_node	1	Q0	Survivor node index
TCQ_channel_indices	[0..(BLOCK_LEN−1)] * [0..(N_STATES−1)]	Q0	Trellis memory of channel indices
TCQ_channel_symbols	[0..(IDIM−1)]	Q0	Memory of compressed signal

Table J.3/G.728 – Internal Processing Variables *(concluded)*

Name	Array Index Range	Fixed Point Format	Description
TCQ_common_part_pred_q2	1	Q2	Common part of the predicted value
TCQ_ET	[0..(BLOCK_LEN−1)] [0..(N_STATES−1)]		Trellis memory of residuals
TCQ_predict_sample_q2	[0..(N_STATES−1)	Q2	Memory of predicted value for each trellis state (node)
TCQ_predictor_state_q2	[0..(N_STATES−1)][0..(PREDICTOR_ORDER−1)]	Q2	Trellis memory of the predictor state
TCQ_reconstructed_q2	[0..(BLOCK_LEN−1)] [0..(N_STATES−1)]	Q2	Trellis memory of reconstructed levels
TCQ_reconstructed_q2	[0..(BLOCK_LEN−1)]*[0..(N_STATES−1)]	Q2	Trellis memory of reconstructed samples
TCQ_resid	[0..(N_STATES−1)]	Q11	Memory of nodes' (trellis state) residuals
TCQ_STTMP_q2	[0..NFRSZ]	Q2	Buffer for synthesis filter hybrid window
temp_metric	[0..7]	32 bit Q20	Temporary memory of distortion metrics
vbd_to_speech_transition	1	Q0	VBD-to-Speech transition flag
Xk	[0..(N_STATES−1)] [0..(Q_CELLS−1)]	Q11	Quantization intervals limits
Yk	[0..(N_STATES−1)] [0..(Q_CELLS−1)]	Q11	Quantization levels
GDIFF	1	32bit Q9	The difference between the current gain and its average
G_AVE	1	32bit Q9	Gain average
GC_ATMP_SUM	1		Sum of absolute LPC Parameters of block #G.50
GC_ATMP1	1		Second LPC Parameter of block #G.50
GC_NLS_LIMIT	1	Q0	Gain NLS threshold for use in the Gain Compensation block
GC_COMPENSATION	1	Q0	The compensation factor

Table J.4/G.728 – Internal Processing Constants

Name	Value	Symbol	Description
LOG_POL_ORDER	5		Order of logarithmic approximation polynomial
BITS_PER_LEVEL	4		Number of bits, that identify quantization levels
LEVEL_MASK	0x0F		Mask for level bits
PREDICTOR_ORDER	10		Order of data-mode predictor
CODEBOOK_Q	1 Q11		The Q presentation of the codebook
Log_2	24 660		10*log10(2) in Q13
RMS_BUF_LEN	8		Length of RMS calculation
BLOCK_LEN	5	K_d	Trellis block length
MAX_NUMBER	0x7fffffff		Maximum positive number
N_BRANCHES	2		Number of branches emanating from or incoming to each trellis state
N_STATES	4		Number of Trellis states
Q_CELLS	16		Number of Quantization levels in each subset
TCQ_Kd	5	K_d	Trellis delay length
TCQ_Kr	5	K_r	Trellis release role
TCQ_DEPTH	BLOCK_LEN−1		Trellis block length – 1
G_CONST	5		Used for the calculation of G_{ave} in the Gain Compensation module
ATMP_CONST	3		A threshold for detection of narrow bandwidth signal in the Signal Classifier
G_TRS	1 800		Gain Compensation G_{diff} threshold
GC_NLS_LIMIT_INIT	7		Gain Compensation limiter
GC_COMPENSATION_INIT	3		The value subtracted from the gain NLS when a Gain Compensation occurs
GC_CNT_INIT	11		The period of time in which the Gain Compensation is active

J.4.11 Initial Values

Table J.5/G.728 – Initial Values

NAME	Initial Value
ATMP_for_VBD	16 384, 0, ..., 0
IAQ_for_VBD	14
vbd_to_speech_transition	0
block_depth	0
next_stage	1
TCQ_decoder_state	0
ET	0..0
GAIN	512
NLSGAIN	0
DLQ_GAIN	16 384
DLQ_NLSGAIN	14
DLQ_nls_p_cb_q_m_18	–7
DLQ_inv_GAIN	16 384
DLQ_inv_nls_m	–4
GAIN_state_fx	0L
after_limiter_98	–16 384
TCQ_STTMP_q2	0..0
TCQ_reconstructed_q2	0, ..., 0
distortion_metric	0, (MAX_NUMBER>>2), …, (MAX_NUMBER>>2)
dms	0
dml	0
RMS_Q11	0
ap_q11	0
al_q11	0
GAVE	0
UNSTEADY	1
G_CNT	0
GC_SC_FLAG	0
GC_ID_FLAG	0
GC_CNT	0
GC_FLAG	0

J.5 Bibliography

(1) GERSHO (A.) and GRAY (R.M.), Vector quantization and signal compression.

(2) MARCELLIN (M.W.) and FISCHER (T.R.), Trellis-Coded Quantization of Memoryless and Gauss-Markov Sources, IEEE transactions on communications Vol. 38, No. 1, January 1990.

(3) TRUSHKIN (A.V.), On the design of an optimal Quantizer, IEEE transactions on information theory Vol. 39, No. 4, July 1993.

(4) JAYANT (N.S.) and NOLL (P.), Digital coding of waveforms.

(5) PETR (D.W.), 32 Kbps ADPCM-DLQ coding for network applications, IEEE 1982.

(6) MOC/Israel, Test results for non-voiced performance assessment of LD-CELP algorithm operating at 40 kbit/s mainly for DCME applications, SG 16 COM-D.278, Santiago, Chile, 18-27 May 1999.