INTERNATIONAL TELECOMMUNICATION UNION

ITU-T

TELECOMMUNICATION
STANDARDIZATION SECTOR
OF ITU

G.728

Annex I

(05/99)

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 I: Frame or packet loss concealment for the LD-CELP decoder

ITU-T Recommendation G.728 – Annex I

(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.728

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

ANNEX I

Frame or packet loss concealment for the LD-CELP decoder

Summary

This annex defines an extension for robust operation of the G.728 algorithm at 9.6, 12.8 and 16 kbit/s in the presence of frame erasure or packet loss in mobile or packet environments.

Source

Annex I to ITU-T Recommendation G.728 was prepared by ITU-T Study Group 16 (1997-2000) and was approved under the WTSC Resolution No. 1 procedure on 27 May 1999.

FOREWORD

ITU (International Telecommunication Union) is the United Nations Specialized Agency in the field of telecommunications. The ITU Telecommunication Standardization Sector (ITU-T) is a permanent organ of the ITU. The ITU-T is responsible for studying technical, operating and tariff questions and issuing Recommendations on them with a view to standardizing telecommunications on a worldwide basis.

The World Telecommunication Standardization Conference (WTSC), which meets every four years, establishes the topics for study by the ITU-T Study Groups which, in their turn, produce Recommendations on these topics.

The approval of Recommendations by the Members of the ITU-T is covered by the procedure laid down in WTSC Resolution No. 1.

In some areas of information technology which fall within ITU-T's purview, the necessary standards are prepared on a collaborative basis with ISO and IEC.

NOTE

In this Recommendation the term *recognized operating agency (ROA)* includes any individual, company, corporation or governmental organization that operates a public correspondence service. The terms *Administration*, *ROA* and *public correspondence* are defined in the *Constitution of the ITU (Geneva, 1992)*.

INTELLECTUAL PROPERTY RIGHTS

The ITU draws attention to the possibility that the practice or implementation of this Recommendation may involve the use of a claimed Intellectual Property Right. The ITU takes no position concerning the evidence, validity or applicability of claimed Intellectual Property Rights, whether asserted by ITU members or others outside of the Recommendation development process.

As of the date of approval of this Recommendation, the ITU had received notice of intellectual property, protected by patents, which may be required to implement this Recommendation. However, implementors are cautioned that this may not represent the latest information and are therefore strongly urged to consult the TSB patent database.

CONTENTS

Recommendation G.728

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

ANNEX I

Frame or packet loss concealment for the LD-CELP decoder

(Geneva, 1999)

I.1 Scope

This annex is concerned with how to conceal the loss of bit-stream information due to frame or packet loss in the communications channel. During normal operation, the decoder performs in a manner identical with Recommendation G.728 (main body) or G.728 Annex G at 16 kbit/s or with G.728 Annex H when operating at 12.8 or 9.6 kbit/s. The modification proposed in this annex only involves changing the decoder during times when the bit-stream is unavailable. It is presumed that this loss of the bit stream is indicated to the decoder by some external means. This annex is not essential for normal operation of G.728.

I.2 Normative references

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

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

– ITU-T Recommendation G.728 Annex H (1997), *Variable bit rate LD-CELP operation mainly for DCME at rates less than 16 kbit/s.*

I.3 Introduction

This annex describes the modifications of the G.728 decoder to make it suitable for handling frame erasures in the received bit stream. These modifications include excitation extrapolation, bandwidth expansion for the LPC filter, continuation of part of backward adaptation using the extrapolated excitation, continuation of post-filtering and post-filter adapter using the extrapolated excitation, and limiting the rate of increase for the backward-adaptive gain after frame erasures. Such decoder changes maintain the bit-stream compatibility to Recommendation G.728 and the additional complexity is negligible.

To maintain the bit-stream compatibility to Recommendation G.728, all the modifications are confined to the decoder. They take effect only during erased frames and the first few "good" frames after frame erasures. The proposed system otherwise operates exactly the same way as in G.728 during all other "good" frames.

This annex assumes that the readers are already familiar with the Recommendation G.728 specification document. The G.728 algorithm will not be discussed here. Only modifications to G.728 will be described. To avoid confusion, from here on, we will reserve the word *frame* for the block size associated with frame erasure (typically a multiple of 2.5 ms). We will use *adaptation cycle* to denote the 2.5 ms block for the adaptation of G.728 filter parameters.

Based on the input from ITU-R (formerly CCIR), with a frame erasure channel a received frame of bits can be considered to be either totally good (zero bit error), or totally bad (bit error rate within the frame approaches 50%). It is also assumed that an error detection code has been used so that the

receiver can reliably determine which frames are good and bad, and make such information available to the G.728 speech decoder.

This annex is divided as follows. Subclause I.4, "Principles of operation", contains subclauses which describe the changes in operation of the decoder during frame erasure concealment. Subclause I.5, "Pseudo-code for frame erasure concealment", contains all of the pseudo-code needed for the decoder modifications. For completeness, pseudo-code is given for the floating-point as well as fixed-point specifications. Finally, subclause I.6 contains tables of the additional coder parameters and variables that have been added for frame erasure concealment operations.

I.4 Principles of operation

The modifications to Recommendation G.728 for erased frames include extrapolation of the excitation and the LPC filter coefficients, continuation of certain backward adaptation operations for LPC and gain predictors, and limiting the rate of growth for the gain after frame erasures. These are described separately in the following subclauses.

I.4.1 Excitation extrapolation

First refer to Figure 3/G.728. We do not extrapolate the output of the excitation VQ codebook (Block 29). Instead, we extrapolate the output of the gain scaling unit (Block 31), or the *gain-scaled excitation*, which is denoted by the ET() array in the pseudo-code in 5.14/G.728. In the original G.728 decoder, we keep only 5 samples of ET() in the memory. To be able to extrapolate ET(), now we need to maintain an additional array ETPAST() which stores the present and past ET() vectors.

Refer to the pseudo-code in 5.14/G.728. In the G.728 decoder, the pitch period extraction module (Block 82) gives us a pitch period KP. The pitch predictor tap calculator (Block 83) calculates the optimal tap weight of a single-tap pitch predictor for the decoded speech. This tap weight PTAP is used as a rough indicator of whether the current adaptation cycle is voiced or not. If PTAP is greater than PPFTH, where PPFTH = 0.6, then the current adaptation cycle is considered to be voiced, and the long-term post-filter is applied; otherwise, the long-term post-filter is disabled. Since we already have the pitch period and a voicing indicator available in the original G.728 decoder, we can directly use them in the excitation extrapolation without having to recompute them again.

At the beginning of the first erased frame in an erasure, we use the value of the voicing indicator PTAP obtained in the last adaptation cycle of the last good frame to determine whether the speech is voiced or unvoiced. During erasures we extrapolate the excitation differently for voiced and unvoiced frames. For extrapolation purposes we use a voicing threshold, VTH = PPFTH/1.4, that is lower than PPFTH.

If PTAP > VTH, we treat the last frame as voiced. Then, the extrapolation of ET() is done by periodically repeating a scaled-down version of the last KP samples of the ETPAST() array. Alternatively, this operation can be explained in the following way: Consider a 5-sample rectangular window which has unit sample values and is located at the position corresponding to the current vector of ETPAST(). Now, slide this rectangular window back in time by KP samples, extract the corresponding 5 samples of ETPAST(), and scale them down by multiplying by a scaling factor. The result is the extrapolated ET() vector that we desire. This extrapolated ET() vector is then copied to the proper location of the ETPAST() array to update the current vector of ETPAST(). This process is repeated for all vectors within the erased frame. During erased frames this operation produces an ET() sequence with periodic shapes but decaying magnitudes. The period is KP samples. Such periodic extrapolation of the ETPAST() array is continued until a good frame is received. The scaling factor, FESCALE, is 0.8 for the first 20 ms of an erasure and then decreases by 0.2 for each additional 10 ms. These scaling factors are stored in VOICEDFEGAIN(). If the erasure is longer than 50 ms, FESCALE is set to 0 to avoid introducing long, unnatural, periodic artifacts in the output speech signal.

If PTAP ≤ VTH, then we treat the last adaptation cycle as "unvoiced". Strictly speaking, it is anything other than "voiced", as it could include silence, transition, etc. In this case, we do not perform periodic extrapolation because we do not want to introduce artificial periodicity that is not in the original speech signal. We avoid periodicity by randomly repeating 5 consecutive samples from the last 140 samples in ETPAST(). For each vector in a bad frame, we use a random number generator to generate a random integer between 5 and 140. This random number determines how many samples the sliding rectangular window should slide back in time to extract the corresponding 5 samples of ETPAST() as the extrapolated ET(). Of course, if desired, such a sequence of random numbers can be precomputed and stored.

To insure that the level of the signal generated during an unvoiced frame erasure resembles its recent history, we match the average magnitude in each extrapolated excitation vector to the average excitation magnitude that occurred in the last 5 ms (40 samples) before the erasure. This is done in the following way: At the beginning of the first erased frame, if we determine the speech is unvoiced we calculate the average magnitude of the last 8 vectors of ETPAST(). Let AVMAG be the resulting average magnitude. Then, when we extrapolate each excitation vector, we first calculate the magnitude (MAG) of the vector [which is 5 consecutive samples randomly selected from the last 140 samples of ETPAST()]. Next, we divide AVMAG by MAG to obtain a scaling factor. This scaling factor is then used to multiply each of the 5 samples in the current extrapolated excitation vector. This operation ensures that each of the extrapolated and gain-scaled excitation vector has an average magnitude of AVMAG. To avoid artifacts caused by very long erasures AVMAG is attenuated by 20% for every 10 ms the erasure lasts beyond 20 ms. The attenuation factors are stored in the array UNVOICEDFEGAIN().

I.4.2 LPC filter

During erased frames, we extrapolate the gain-scaled excitation ET() as described above, but we still need an LPC synthesis filter to produce the output speech. We use a somewhat "softened" version of the last good set of LPC coefficients in the last good frame. Such "softening" is achieved by bandwidth expansion, as is done in the bandwidth expansion module of Block 51 (5.6/G.728), except that the bandwidth expansion factor FAC (5.1/G.728) is 0.97 rather than 253/256 ≈ 0.9883. In the first erased frame following a good frame, this bandwidth expansion operation is performed at the 3rd vector of the 1st adaptation cycle, just when the LPC coefficients would have normally been updated were it not for the frame erasure.

Let a_i be the i^{th} LPC coefficients used in the last adaptation cycle of the last good frame. Then, the new set of softened LPC coefficients is obtained as $a'_i = (0.97)^i a_i$, $i = 1, 2, \ldots, 50$. This new set of $\{a'_i\}$ is used during the first 10 ms of the erasure (if it lasts that long), even though there may be subsequent adaptation cycles in the same frame. If the erasure extend beyond 10 ms, this set of $\{a'_i\}$ is further bandwidth expanded by a factor of 0.97. In other words if the erasure lasts more than 10 ms, the LPC coefficients are updated again at the third vector in the 5th adaptation cycle and the i^{th} LPC coefficient is $a''_i = (0.97)^i a'_i = (0.97)^{2i} a_i$. Again, this set of $\{a''_i\}$ is used for the entire next 10 ms of the erasure. Similarly, in the erasure extends to *(k * 10 ms)*, the LPC coefficients are bandwidth expanded version of the last good set of LPC coefficients, where the bandwidth expansion factor is $(0.97)^k$. The farther away from the last good frame, the more bandwidth expansion we apply, and this continues until the next good frame is received. Then, next time there is a bad frame again, the process starts from a bandwidth expansion factor of 0.97 again.

In the first frame of an erasure we do something special to get the most recent LPC coefficients for bandwidth expansion. Specifically, if we directly perform bandwidth expansion on the A() array produced in the last good frame, that A() array corresponds to the autocorrelation function calculated based on the synthesized speech *before* the last good frame. Therefore, it is obsolete for the first erased frame. To get a more recent LPC predictor coefficient array A() for bandwidth expansion, the hybrid window and Durbin's recursion for the 50th-order LPC analysis are performed as usual in the

first erased frame. The resulting A() array is then bandwidth expanded by a factor of 0.97. The resulting LPC coefficients are more up to date because they are based on the synthesized speech up to the last vector of the last good frame. If the erasure extends beyond 10 ms, the special handling stops and the bandwidth expansion is based on the last set of A() array in the memory, which had been bandwidth expanded before as described in the last paragraph.

Note that no such bandwidth expansion of the gain predictor is necessary during erased frames. Since we already extrapolated the gain-scaled excitation ET(), we directly feed this extrapolated ET() to the LPC synthesis filter to produce synthesized speech. Therefore, during erased frames there is no need to produce the backward-adapted excitation gain or to update the gain predictor coefficients.

I.4.3 Backward adaptation of LPC and gain predictors

During erased frames, we do not need the outputs of the backward synthesis filter adapter and backward vector gain adapter (Blocks 33 and 30 in Figure 3/G.728). However, we still want to continue vital operations of the backward adaptation process in these two blocks to reduce discontinuities in the internal states when the next good frame comes. By "vital operations", we mean those parts of the backward adaptation operations that will affect the outputs of Blocks 30 and 33 in the future frames.

Refer to Block 30 (backward vector gain adapter), which is identical to Block 20 (5.7/G.728 and Figure 6/G.728). For Block 30 (backward vector gain adapter), the vital operations include Blocks 67, 39, 40, 41, 42 (which together compute the offset-removed logarithmic gain), and parts of Block 43 (hybrid window module) and Block 46 (log-gain linear predictor). Blocks 67 and 39 through 42 use the extrapolated ET() to compute the offset-removed log-gain, whose value is used to update GTMP(), which is the input to Block 43, and GSTATE(), which is the filter memory of the log-gain linear predictor. Inside Block 43 (hybrid window module), we need to use GTMP() to update the SBLG() buffer, and we also need to update REXPLG(), the recursive component of the autocorrelation function. These operations mentioned above maintain the internal states that will affect the future outputs of Block 30. Any other operations, such as Levinson-Durbin recursion (Block 44), bandwidth expansion (Block 45), inverse logarithm calculator (Block 48), etc., need not be performed during frame erasure, since their outputs are discarded anyway.

Note that since the gain predictor coefficients GP() are not updated during a frame erasure, the first vector after the frame erasure will still use the old GP() calculated in the last good frame before the frame erasure. This is because the GP() array is not updated until the second vector of a G.728 adaptation cycle. We do not attempt to make the GP() array more up to date for the first vector, because it will unnecessarily complicate the algorithm flow control. Also, the gain clamping described below in I.4.5 will guard against a large gain excursion for this first vector. Therefore, the use of a slightly obsolete gain predictor for only one vector is not likely to make a perceptually noticeable difference in speech quality.

Next, refer to Block 33 (decoder backward synthesis filter adapter in 5.14/G.728), which is identical to Block 23 (5.6/G.728 and Figure 5/G.728). For Block 33, the vital operations include:

1) updating the SB() buffer in the hybrid windowing module (Block 49) using the synthesized speech (which is obtained by passing extrapolated ET() through bandwidth expanded version of the last good LPC filter); and

2) computing REXP() in the normal manner using SB().

Most other parts of Block 33 need not be done during erased frames. This is because the LPC coefficients are updated not by the outputs of Blocks 50 and 51 but by a bandwidth expanded version of the LPC coefficients in the last good frame. However, an additional part of Blocks 49 and a small part of Block 50 still need to be performed for the post-filter. This will be discussed in I.4.4.

As just explained, many computationally intensive operations in Blocks 30 and 33 need not be done during erased frames. We can use the saved computation (or DSP processor time) to perform excitation extrapolation and bandwidth expansion of the last good LPC filter, both of which take very little computation to do. This is the reason why the modifications to the G.728 decoder do not increase the overall computational complexity.

I.4.4 Post-filter

For flow control and program simplicity, the operation of the post-filter (5.14/G.728) and post-filter adapter (Blocks 34 and 35 in Figure 3/G.728) is the same in both good and erased frames. This has several implications.

First, during erased frames the post-filter adapter still needs the 10th-order LPC predictor coefficients and the first reflection coefficient from the backward synthesis filter adapter (Block 33 in Figure 3/G.728). Hence, in Block 49 (hybrid windowing module, Figure 5/G.728), we still need to calculate the values of RTMP(1) through RTMP(11); we do not need to calculate RTMP(12) through RTMP(51), however. Similarly, in Block 50, we still need to perform Durbin's recursion from order 1 to order 10, although we do not need to continue from order 11 to order 50. Note that during erased frames the 10th-order LPC coefficients and the first reflection coefficient are actually obtained from the synthesized speech which is ultimately driven by the extrapolated excitation ET().

Second, during erased frames the post-filter updates all its coefficients based on the speech synthesized in the erased frames and then uses the updated coefficients to filter such synthesized speech to produce the final output speech. We do not attempt to "freeze" the post-filter coefficients or use a bandwidth expanded version of the post-filter in the last good frame as we did for the LPC filter. We intentionally let the post-filter "float" with the speech synthesized from the extrapolated excitation. This ensures that the spectral peaks and valleys of the post-filter frequency response match those of the synthesized speech all the time. This avoids the potential cancellation or de-emphasis of the spectral peaks of the synthesized speech by the spectral valleys in the post-filter frequency response due to a mismatch of the two.

While the post-filter continues to update the pitch period KP and the pitch predictor tap PTAP during an erasure, we do not use the updated values to change the voiced/unvoiced decision VOICED and stored pitch period FEDELAY while an erasure is in progress. These values are set once at the beginning of an erasure and do not change for the duration of the erasure.

I.4.5 Limitation of gain growth after frame erasure

It was observed that after a very long frame erasure (up to 100 ms), or when a frame erasure was preceded by a low-amplitude region and followed by a high-amplitude region in the speech, the decoded speech after the frame erasure occasionally had a big "pop", or gain overshoot. This was because the log-gain buffer SBG() in the hybrid window contained very low level gains at the end of the frame erasure, and these low gains were followed by much higher gains in the first few good frames after the frame erasure. Thus, the LPC analysis produced a gain predictor which attempted to quickly catch up with the abrupt jump in the gain level. The predicted (i.e. backward adapted) gain therefore grew too fast and too much, causing a gain overshoot and the resulting "pop".

This problem is avoided by limiting the rate of growth of the backward adapted gain for the first few good frames after a frame erasure. After a frame erasure we limit the gain growth to no more than FEGAINMAX = +2 dB per 5-sample vector. If the increase in the predicted log-gain is more than +2 dB/vector we clip it to 2 dB above the last log-gain. The duration of this "gain clamping" depends on the length of the frame erasure. If the frame erasure is AFTERFEMAX = 16 (40 ms) or less, the duration of gain clamping is the length of the frame erasure. If the frame erasure is longer than 40 ms long, the gain clamping is limited to the first 40 ms after the erasure. The two constants FEGAINMAX and AFTERFEMAX are tunable coder parameters.

I.5 Pseudo-code for frame erasure concealment

First, the pseudo-code for the decoder main program is given below. Only the block execution sequence is shown and no low-level detail of parameter passing is described. In the subclauses which follow, the pseudo-codes for the new routines are given.

I.5.1 New main program loop for decoder

The main program loop for the decoder for either fixed-point or floating-point code only differs in regard to the backward vector gain adapter, which was changed in the fixed-point specification. Since these changes are relatively small in the main loop, we only give one version of the pseudo-code here. The C preprocessor constructs **#ifdef**, **#else**, and **#endif**, will be used to specify which segment belongs to the fixed-point specification and which belongs to the floating-point specification. The pseudo-code between a **#ifdef FIXPT** and the next **#else** is for fixed-point specification, while the pseudo-code between that **#else** and the next **#endif** is for floating-point specification.

```
Initialize all decoder variables to their initial values.
ILLCOND=.FALSE.
ILLCONDG=.FALSE.
FERROR=.FALSE.  | TRUE if current frame is erased
FESIZE=4        | Number of 2.5 msec adaptation cycles in a frame
                | erasure(FE)
                | e.g.: 1 for 2.5 msec FEs, 4 for 10 msec FEs
                | FEs may be any multiple of 2.5 msec. Set by user.
ICOUNT=4        | Vector counter, 1,2,3,4,1,2,.., .625 ms each
ADCOUNT=FESIZE  | Adaptation cycle counter, 2.5 msec each
                | Counts from 1,2,...,FESIZE,1,2,...,FESIZE,...;
FECOUNT=0       | Number of consecutively erased 2.5 msec adaptation
                | cycles
AFTERFE=0       | Number of 2.5 msec adaptations cycles after a FE to
                | limit the gain. Counts down.
OGAINDB=-32     | Last predicted gain in dB

VEC_LOOP:
   If ICOUNT = 4, do the next 14 lines
      ICOUNT = 0                        | reset vector counter
                | If the last adaptation cycle was not erased, and the
                | gain was clamped decrease the number of adaptation cycles
                | left to clamp the gain.
      If FERROR = .FALSE. and AFTERFE > 0, do AFTERFE = AFTERFE - 1
                | Check if the next frame is erased
      If ADCOUNT = FESIZE, do the next 7 lines
         ADCOUNT = 0                    | reset adaptation cycle counter
         read FERROR                    | read in new ferror
                | At the first good frame after an erasure set AFTERFE
                | so the gain will be clamped for the next few good frames.
         If FERROR = .FALSE. and FECOUNT > 0, do the next 4 lines
            AFTERFE = AFTERFE + FECOUNT |add the length of the FE to AFTERFE
            If AFTERFE > AFTERFEMAX, do the next line
               AFTERFE = AFTERFEMAX     | clamp gain for 40 msec max
            FECOUNT = 0                 | reset FECOUNT
      If FERROR = .TRUE., do the next 2 lines
         FECOUNT = FECOUNT + 1          | Update FECOUNT
                | At the start of a FE, and every 10 msec thereafter,
                | call block31SF to initialize the FE flags and update the
                | excitation FE gains. e.g. when FECOUNT is equal to 1, 5, 9, 13, ...
         If (FECOUNT & 3) = 1, do block31SF
      ADCOUNT = ADCOUNT + 1             | update adaptation cycle counter
   ICOUNT = ICOUNT + 1                  | update vector counter

   Get ICHAN of the current vector from the input buffer.
   Obtain the shape index IS and gain index IG from ICHAN

                                        | check whether to update LPC coeff.
```

```
   If ICOUNT = 3, do the next 2 lines
      If FERROR = .FALSE. and ILLCOND = .FALSE., do block 51.
      If FERROR = .TRUE. and (FECOUNT & 3) = 1, do block 51FE
   If ICOUNT = 2 and ILLCONDG=.FALSE. and FERROR = .FALSE., do block 45.

   do block 46                          | GSTATE(1:9) shifted down 1 position

#ifdef FIXPT
   If FERROR = .FALSE., do the next 3 lines
      do blocks 98AF, 99 and 48         | get backward-adapted gain
      OGAINDB = output of block 98AF    | for limiting gain growth after FE
      do blocks 29 and 31               | scale selected excitation codevector
#else
   If FERROR = .FALSE., do the next 3 lines
      do blocks 47AF and 48             | get backward-adapted gain
      OGAINDB = output of block 47AF    | for limiting gain growth after FE
      do blocks 29 and 31               | scale selected excitation codevector
#endif
   If FERROR = .TRUE., do the next line
      do block 31FE                     | get extrapolated excitation

   do block 31E                         | update ETPAST()
   do block 32                          | get ST()
   If ICOUNT = 1, do block 85           | update short-term postfilter coeff.
   do block 81                          | 10th-order LPC inverse filtering
   If ICOUNT = 3, do the next 3 lines
       do block 82                      | pitch period extraction
       do block 83                      | compute pitch predictor tap
       do block 84                      | update long-term postfilter coeff.
   do block 71                          | long-term postfilter
   do block 72                          | short-term postfilter
   do blocks 73 and 74                  | calculate sums of absolute values
   do block 75                          | ratio of sums of absolute values
   do block 76                          | low-pass filter of scaling factor
   do block 77                          | gain control of postfilter output

#ifdef FIXPT
   If FERROR = .FALSE., do the next 2 lines
     do blocks 93, 94, 96, and 97       | update log-gain
     GSTATE(1) = output of block 97     | update gain predictor memory
#else
   If FERROR = .FALSE., do the next 2 lines
     do blocks 39, 40 and 42            | update log-gain
     GSTATE(1) = output of block 42     | update gain predictor memory
#endif

   If FERROR = .TRUE., do the next 3 lines
     do block 97FE                      | compute log gain of extrapolated ET()
     OGAINDB = output of 97FE           | save for limiting gain growth after FE
     GSTATE(1) = output of block 97FE
                                        | then update gain predictor memory

   I=(ICOUNT-1)*IDIM                    | I=starting address of STTMP()
   copy ST(1:5) to STTMP(I+1:I+5)       | update STTMP()
   NLSSTTMP(ICOUNT)=NLSST               | update NLSSTTMP()
                                        | end of once-per-vector processing

                                        | start once-per-cycle processing
   If ICOUNT = 4, do the next 6 lines
       do block 49FE                    | output ill-condition flag = ILLCOND
       do block 50, order 1 to 10       | output pred coef= ATMP() with NLSATMP
                                        | output ill-condition flag =     ILLCONDP
       NLSAPF=NLSATMP                   | save the 10th-order predictor for
       copy ATMP(2:11) to APF(2:11)     | postfilter use later
       If FERROR = .FALSE., do the next line
           continue block 50,           | continue to finish block 50
           order 11 to 50               | output pred coef= ATMP() with
                                        | NLSATMP output ill-condition flag = ILLCOND
```

```
   If ICOUNT = 1, do the next 7 lines
       GTMP(1)=GSTATE(4)               | update GTMP() in one shot
       GTMP(2)=GSTATE(3)
       GTMP(3)=GSTATE(2)
       GTMP(4)=GSTATE(1)
       do block 43FE                   | output ill-condition flag = ILLCONDG
       If FERROR = .FALSE., do the next line
           do block 44                 | output pred. coeff. = GPTMP()
                                       | output ill-condition flag = ILLCONDG
                                       | end of once-per-frame processing
Go to VEC_LOOP
```

I.5.2 Block 31SF – Set flags and scaling factor for frame erasure

This block is new. It is called at the beginning of a frame erasure, and every 10 ms thereafter during an erasure. At the beginning of an erasure it sets the voiced/unvoiced flag based on the value of PTAP. If the last good frame was voiced, it saves the pitch period KP in FEDELAY for periodic extrapolation. If unvoiced, it calculates the sum of magnitudes of the last 40 samples of ETPAST() in the last good frame, and then divides the result by 8. These values are computed once at the beginning of an erasure and are used for the whole erasure. If the erasure lasts more than 10 ms Block 31SF is called again (every 10 ms) and the excitation extrapolation scaling factor FESCALE is attenuated based on the length of the erasure and whether it is voiced or unvoiced. With voiced speech if the erasure lasts 60 ms or more, FESCALE is set to 0 to stop "beeping" artifacts. If the erasure is unvoiced FESCALE is set to 0 at 70 ms. The floating-point code is given first:

```
   VOICEDFEGAIN(0:4) = .8, .8, .6, .4, .2
   UNVOICEDFEGAIN(0:5) = 1., 1., .8, .6, .4, .2

   N10MSEC = FECOUNT >> 2
   If N10MSEC == 0, do the following indented lines
      If PTAP > VTH, do the next 2 indented lines
         FEDELAY = KP
         VOICED = .TRUE.
      Otherwise, do the following indented lines
         VOICED = .FALSE.
         AVMAG = 0.
         For I = -39, -38, ..., 0, do the next line
            AVMAG = AVMAG + |ETPAST(I)|          | sum of ETPAST() magnitudes
         AVMAG = 0.125*AVMAG                     | divide by 8
   If VOICED = .TRUE., do the following indented lines
      If N10MSEC < 5, do FESCALE = VOICEDFEGAIN(N10MSEC)
      Otherwise, do FESCALE = 0.
   Otherwise, do the following indented lines    | not voiced
      If N10MSEC < 6, do FESCALE = AVMAG * UNVOICEDFEGAIN(N10MSEC)
      Otherwise, do FESCALE = 0.
```

The fixed-point code is the following:

```
                                               | Constants below are in Q15
   VOICEDFEGAIN(0:4) = 26214, 26214, 19661, 13107, 6554
   UNVOICEDFEGAIN(0:5) = 32767, 32767, 26214, 19661, 13107, 6554

   N10MSEC = FECOUNT >> 2
   If N10MSEC == 0, do the following indented lines
      If PTAP > VTH, do the next 2 indented lines
         FEDELAY = KP
         VOICED = .TRUE.
      Otherwise, do the following indented lines
         VOICED = .FALSE.
         AA0 = 0
         For I = -39, -38, ..., 0, do the next line
            AA0 = AA0 + |ETPAST(I)|              | sum of 40 ETPAST magnitudes
         Call VSCALE(AA0,1,1,30,AA0,NLS)         | normalize numerator
         AVMAG = RND(AA0)
         NLSAVMAG = NLS - 16 + 3                 | "+3" is for "divided by 8"
   If VOICED = .TRUE., do the following indented lines
      If N10MSEC < 5, do FESCALE = VOICEDFEGAIN(N10MSEC)
      Otherwise, do FESCALE = 0
```

```
Otherwise, do the following indented lines    | not voiced
   If N10MSEC < 6, do the following indented lines
      FESCALE = UNVOICEDFEGAIN(N10MSEC)       | scaling factor in Q15
      AA0 = FESCALE * AVMAG                   | scale AVMAG by FESCALE
      Call VSCALE(AA0,1,1,30,AA0,NLSFESCALE)  | normalize FESCALE
      FESCALE = RND(AA0)                      | save result in FESCALE
      NLSFESCALE = NLSFESCALE + NLSAVMAG - 1  | NLS for FESCALE
                                              | -1 is from (Q15 - 16)
   Otherwise, do FESCALE = 0
```

I.5.3 Block 31FE – Excitation signal extrapolation

This block is new and is used only during erased frames. It extrapolates the excitation signal in two possible ways, depending on whether the last good frame is considered voiced or not. For voiced speech the signal is extrapolated periodically with a decayed value. The scaling factor FESCALE starts at 0.8 and decreases every 10 ms during the erasure in Block 31SF.

For anything else (unvoiced, silence, etc.), a random segment from the previous 140 samples is used. This random segment is scaled so that it has the same average magnitude as the previous 40 samples. To calculate the average magnitudes of the 5-sample random segment and of the previous 40 samples, the sums of magnitudes should have been divided by 5 and 40, respectively. However, all we need is the ratio of the two average magnitudes. Thus, in practice, we calculate MAG as the sum of magnitudes of the 5-sample random segment, and AVMAG as the sum of magnitudes of the previous 40 samples divided by 8. The ratio of the two resulting numbers will still be the correct ratio we want. The value of AVMAG was calculated in Block 31SF above during the first adaptation cycle of the first erased frame. AVMAG is attenuated in Block 31SF based on the duration of the erasure and placed in FESCALE. Note that since interoperability with the G.728 encoder is not an issue during frame erasures there is no need to have a pre-defined sequence of random numbers. The numbers also do not have to be truly random. Any sequence of integers between 5 and 140 can be used here, as long as the sequence is not periodic or close to periodic. Therefore, no pseudo-code is given below for the random number generator.

We begin with the floating-point code.

```
                                          |  Voiced case
 If VOICED = .TRUE., do the next 2 indented lines
     For I = 1, 2, ..., IDIM, do the next line
       ET(I) = FESCALE * ETPAST(I-FEDELAY) | KP is pitch period
                                          | from 82

                                          | Unvoiced case
 If VOICED = .FALSE., do the following indented lines
     set FEDELAY = a random number between 5 and 140
     MAG = 0.
     For I = 1, 2,  ..., IDIM, do the next 2 lines
         ET(I) = ETPAST(I-FEDELAY)
         MAG = MAG + |ET(I)|              | sum of ET() magnitudes
     If MAG = 0 do MAG = 1.               | avoid divide by zero
     RATIO = FESCALE / MAG

     For I = 1, 2,  ..., IDIM, do the next line
         ET(I) = RATIO*ET(I)              | scale to have same magnitude
```

For the fixed-point pseudo-code, ET and ETPAST are using different representations. As specified in Annex G/G.728, ET is represented in 15-bit block floating-point with one leading zero. On the other hand, to simplify the operation of this block, we use a fixed Q2 format to represent the long ETPAST array. There is no point to use block floating-point for ETPAST, because even the numerical error of the fixed Q2 representation is much smaller than the difference between the extrapolated excitation and the true excitation in the encoder. In the extremely rare case of ETPAST overflow, the value of ETPAST is clipped at the saturation level of +32767 or −32768 to avoid

"wrap-around." The following pseudo-code first writes the extrapolated excitation vector (in Q2) into ETPAST(1) through ETPAST(5), then converts the extrapolated 5 ETPAST samples into 15-bit block floating-point ET array.

```
If VOICED = .TRUE., do the next 4 lines       | Voiced case
 TEMP = FESCALE
 NLSTEMP = 15                                 | FESCALE is Q15
   For I = 1, 2,  ..., IDIM, do the next line
    ETPAST(I) = ETPAST(I-FEDELAY)| FEDELAY is pitch period

If VOICED = .FALSE., do the next 13 lines     | Unvoiced case
    set FEDELAY = a random number between 5 and 140
    AA1 = 0
    For I = 1, 2,  ..., IDIM, do the next 2 lines
        ETPAST(I) = ETPAST(I-FEDELAY)
        AA1 = AA1 + |ETPAST(I)|               | sum of ETPAST(1:5) magnitudes
    If AA1 = 0 or FESCALE = 0, do the next 2 lines
        TEMP = 0
        NLSTEMP = 15
    Otherwise, do the next 4 lines
           Call VSCALE(AA1,1,1,30,AA1,NLSDEN) | normalize
                                              | denominator
           DEN = RND(AA1)
           NLSDEN = NLSDEN - 16
           Call DIVIDE(FESCALE,NLSFESCALE,DEN,NLSDEN,TEMP,NLSTEMP)

    For I = 1, 2,  ..., IDIM, do the next 5 lines
        AA0 = TEMP * ETPAST(I)                | AA0 in Q(2+NLSTEMP) format
        AA0 = AA0 >> NLSTEMP                  | shift AA0 to make Q2
        if AA0 > 32767, set AA0 = 32767       | clip if necessary
        if AA0 < -32768, set AA0 = -32768
        ETPAST(I) = AA0                       | scaled ETPAST in Q2
    Call VSCALE(ETPAST,IDIM,IDIM,13,ET,NLS)   | convert to ET in 15-bit
                                              | BFL
    NLSET = NLS + 2                           | update NLSET
```

I.5.4 Block 31E – Excitation signal update

This block updates the excitation signal buffer. Regardless of whether there is a frame erasure currently being concealed, this block must be completed so that the buffer will be ready when it is needed. We begin with the floating-point code.

```
For I = -KPMAX+1, -KPMAX+2, ..., -IDIM, do the next line
    ETPAST(I) = ETPAST(I+IDIM)                | shift ETPAST by IDIM samples
For I = 1, 2, ..., IDIM, do the next line
    ETPAST(I-IDIM) = ET(I)                    | copy ET to ETPAST
```

For the fixed-point pseudo-code, ETPAST is stored in Q2 format, and ET is stored in 15-bit block floating-point with Q format NLSET. Therefore, a format conversion is needed when copying ET to ETPAST. Here is the fixed-point code.

```
For I = -KPMAX+1, -KPMAX+2, ..., -IDIM, do the next line
    ETPAST(I) = ETPAST(I+IDIM)                | shift Q2 ETPAST by 5 samples
NRS = NLSET - 2                               | # of right shifts to make Q2
For I = 1, 2, ..., IDIM, do the next 6 lines
    AA0 = ET(I)                               | get 15-bit mantissa of ET
    if NRS > 0, AA0 >> NRS                    | perform the shifting
    if NRS < 0, AA0 << -NRS
    if AA0 > 32767, set AA0 = 32767           | clip if necessary
    if AA0 < -32768, set AA0 = -32768
    ETPAST(I-IDIM) = AA0                      | ETPAST is now Q2
```

I.5.5 Block 43FE – Hybrid windowing module

In this subclause, both the floating-point and fixed-point pseudo-code for Block 43FE are given. This code has been modified for the case of frame erasure concealment and should be substituted for the original Block 43 described in 5.7/G.728. The only modification in the floating-point code involves only one more line of pseudo-code which skips the final autocorrelation function calculation in the event of a frame erasure. First, the floating-point pseudo-code is presented.

```
N1=LPCLG+NUPDATE                        | compute some constants (can be
N2=LPCLG+NONRLG                         | precomputed and stored in memory)
N3=LPCLG+NUPDATE+NONRLG

For N=1,2,...,N2, do the next line
    SBLG(N)=SBLG(N+NUPDATE)             | shift the old signal buffer;
For N=1,2,...,NUPDATE, do the next line
    SBLG(N2+N)=GTMP(N)                  | shift in the new signal;
                                        | SBLG(N3) is the newest sample
K=1
For N=N3,N3-1,...,3,2,1, do the next 2 lines
    WS(N)=SBLG(N)*WNRLG(K)              | multiply the window function
    K=K+1

For I=1,2,...,LPCLG+1, do the next 4 lines
    TMP=0.
    For N=LPCLG+1,LPCLG+2,...,N1, do the next line
        TMP=TMP+WS(N)*WS(N+1-I)
    REXPLG(I)=(3/4)*REXPLG(I)+TMP       | update the recursive
                                        | component

If FERROR = .TRUE., skip all the lines below
    For I=1,2,...,LPCLG+1, do the next 3 lines
        R(I)=REXPLG(I)
        For N=N1+1,N1+2,...,N3, do the next line
            R(I)=R(I)+WS(N)*WS(N+1-I)   | add the non-recursive
                                        | component

    R(1)=R(1)*WNCF                      | white noise correction
```

Now we give the fixed-point version of the same module. The only modification for the fixed-point code actually occurs in the module HWMCOREFE, which is substituted for HWMCORE. This fixed-point pseudo-code replaces the original Block 43 fixed-point code in G.3.14/G.728. The subroutine HWMCOREFE can be found with Block 49FE in I.5.7.

```
N1=LPCLG+NUPDATE (=10+4)                | compute some constants (can be
N2=LPCLG+NONRLG (=10+20)                | precomputed and stored in memory)
N3=LPCLG+NUPDATE+NONRLG (=10+4+20)

For N=1,2,...,N2, do the next line
    SBLG(N)=SBLG(N+NUPDATE)             | shift the old signal buffer
For N=1,2,..,NUPDATE, do the next line
    SBLG(N2+N)=GTMP(N)                  | SBLG(N3) is the newest sample
                                        | all SBLG are Q9 and represented
                                        | in 16 bits precision
Call FINDNLS(SBLG,N3,N3,14,NLS)         | find the amount of left shifts
NLSTMP=NLS-1                            | needed in the next loop for 2
                                        | bits of headroom later

K=1
For N=34,33,...,1, do the next 5 lines
    P=SBLG(N)*WNRLG(K)                  | WNRLG is Q15
    If NLSTMP = -1, set AA0=P >> 1
    If NLSTMP > -1, set AA0=P << NLSTMP
    WS(N)=RND(AA0)                      | WS(N) is 14 bits or less
    K=K+1

NLSATTLG = 14
Call HWMCOREFE(LPCLG,N1,N3,NLSATTLG,WS,NLSTMP,REXPLG,NLSREXPLG,R,ILLCOND)
```

I.5.6 Block 47AF – Log-gain limiter after frame erasure

This new block should replace Block 47 in 5.7/G.728. This is the floating-point pseudo-code.

```
If GAIN < 0., set GAIN = 0.    | Correspond to linear gain 1.
If GAIN > 60., set GAIN = 60.  | Correspond to linear gain 1000.
TMP = GAIN - OGAINDB
If AFTERFE > 0 and TMP > FEGAINMAX, do the next line
  GAIN = OGAINDB + FEGAINMAX
```

I.5.7 Block 49FE – Hybrid window module for synthesis filter

This is the block for the hybrid window module for the synthesis filter as modified for the case of frame erasures. This block should be substituted for the previous Block 49 described in 5.6/G.728. We begin with the floating-point pseudo-code for the hybrid windowing module.

```
N1=LPC+NFRSZ                                | compute some constants (can be
N2=LPC+NONR                                 | precomputed and stored in memory)
N3=LPC+NFRSZ+NONR

For N=1,2,...,N2, do the next line
    SB(N)=SB(N+NFRSZ)                       | shift the old signal buffer;
For N=1,2,...,NFRSZ, do the next line
    SB(N2+N)=STTMP(N)                       | shift in the new signal;
                                            | SB(N3) is the newest sample
K=1
For N=N3,N3-1,...,3,2,1, do the next 2 lines
    WS(N)=SB(N)*WNR(K)                      | multiply the window function
    K=K+1

For I=1,2,...,LPC+1, do the next 4 lines
    TMP=0.
    For N=LPC+1,LPC+2,...,N1, do the next line
        TMP=TMP+WS(N)*WS(N+1-I)
    REXP(I)=(3/4)*REXP(I)+TMP               | update the recursive component

If FERROR = .TRUE., do the next 4 lines
    For I=1,2,...,11, do the next 3 lines
        RTMP(I)=REXP(I)
        For N=N1+1,N1+2,...,N3, do the next line
           RTMP(I)=RTMP(I)+WS(N)*WS(N+1-I) | add non-recursive
                                            | component

If FERROR = .FALSE., do the next 4 lines
    For I=1,2,...,LPC+1, do the next 3 lines
        RTMP(I)=REXP(I)
        For N=N1+1,N1+2,...,N3, do the next line
           RTMP(I)=RTMP(I)+WS(N)*WS(N+1-I) | add non-recursive
                                            | component

RTMP(1)=RTMP(1)*WNCF                        | white noise correction
```

The fixed-point pseudo-code for the hybrid windowing module (Block 49FE) is modified in a similar fashion. This time the change is actually found in HWMCOREFE. This fixed-point pseudo-code replaces the original Block 49 fixed-point code in G.3.17/G.728.

```
N1=LPC+NFRSZ(=70)                   | compute some constants (can be
N2=LPC+NONR(=85)                    | precomputed and stored in memory)
N3=LPC+NFRSZ+NONR(=105)
N4=N3/IDIM(=21)
N5=NFRSZ/IDIM(=4)
N6=N4-N5(=17)

For N=1,2,...,N2, do the next line
    SB(N)=SB(N+NFRSZ)               | shift old part of buffer SB
For N=1,2,...,N6, do the next line
    NLSSB(N)=NLSSB(N+N5)            | shift old NLSSB
```

```
For N=1,2,...,NFRSZ, do the next line
    SB(N2+N)=STTMP(N)              | shift in new part of SB
For N=1,2,...,N5, do the next line
    NLSSB(N6+N)=NLSSTTMP(N)        | shift in new NLSSB

          | Now find the minimum NLSSB - this determines NLSTMP
NLSTMP=Min{NLSSB(1),NLSSB(2),...,NLSSB(N4)}

K=1                                | now multiply SB by the
N=N3                               | hybrid window function
For J=1,2,...,N4, do the next 8 lines
    NRSH=NLSSB(J)-NLSTMP-1         | -1 to compensate for Q15 mult
    For M=1,2,...,IDIM, do the next 5 lines
       P=SB(K)*WNR(N)              | WNR is Q15 multiplication
       If NRSH = -1, set AA0=P << 1
       If NRSH > -1, set AA0=P >> NRSH
       WS(K)=RND(AA0)              | round upper word & store in WS
       N=N-1
    K=K+1

NLSATT50 = 14
Call HWMCOREFE(LPC,N1,N3,NLSATT50,WS,NLSTMP,REXP,NLSREXP,RTMP,ILLCOND)
```

The fixed-point code for HWMCOREFE has been modified so that in the event of a frame erasure, only the values of RTMP(1), ..., RTMP(11) are calculated, although REXP(1), ..., REXP(51) are updated.

```
    SUBROUTINE HWMCOREFE(LPO,N1,N3,NLSATT,WS,NLSTMP,
      RREC,NLSRREC,R,ILLCOND)
    NLSAA0=2*NLSTMP
    AA0=0                                | compute recursive part of RREC(1)
    For N=LPO+1,...,N1, do the next 2 lines
       P=WS(N)*WS(N)                     | WS has 2 bits of headroom
       AA0=AA0+P                         | AA0 will have 5 bits of headroom
                                         | for energy calculation

                                         | Case 1: NLSRREC > NLSAA0
    If NLSRREC > NLSAA0, do the next 22 lines
       AA0=AA0 >> 1
       IR=NLSRREC-NLSAA0+1
       AA1=RREC(1) << NLSATT             | this can be done by multiplication
       AA1=-AA1+(RREC(1) << 16)          | scale RREC by attenuation factor
       AA1=AA1 >> IR                     | align AA0 & AA1
       AA0=AA0+AA1
       Call VSCALE(AA0,1,1,30,AA0,NLSRE) | Find NLS for RREC
       RREC(1)=RND(AA0)                  | upper 16 bits of AA1 saved
       NLSRREC=NLSAA0-1+NLSRE
       For I=1,2,...,LPO, do the next 11 lines
           AA0=0                         | compute recursive part of RREC(I+1)
           For N=LPO+1,...,N1, do the next 2 lines
               P=WS(N)*WS(N-I)
               AA0=AA0+P
           AA0=AA0 >> 1
           AA1=RREC(I+1) << NLSATT       | scale RREC by 3/4 or 1/2
           AA1=-AA1+(RREC(I+1) << 16)    |
           AA1=AA1 >> IR
           AA0=AA0+AA1
           AA0=AA0 << NLSRE
           RREC(I+1)=RND(AA0)            | upper 16 bits of AA0 saved
       Go to FIN_RECUR

                                         | Case 2: NLSRREC = NLSAA0
    If NLSRREC = NLSAA0, do the next 21 lines
       AA1=RREC(1) << NLSATT             | scale RREC by 3/4 or 1/2
       AA1=-AA1+(RREC(1) << 16)          |
       AA0=AA0 >> 1
       AA1=AA1 >> 1
       AA0=AA0+AA1
       Call VSCALE(AA0,1,1,30,AA0,NLSRE) | Find NLS for RREC
       RREC(1)=RND(AA0)                  | upper 16 bits of AA1 saved
```

```
       NLSRREC=NLSRREC-1+NLSRE
       For I=1,2,...,LPO, do the next 11 lines
           AA0=0                          | compute recursive part of RREC(I+1)
           For N=LPO+1,...,N1, do the next 2 lines
               P=WS(N)*WS(N-I)
               AA0=AA0+P
           AA0=AA0 >> 1
           AA1=RREC(I+1) << NLSATT        | scale RREC by 3/4 or 1/2
           AA1=-AA1+(RREC(I+1) << 16)     |
           AA1=AA1 >> 1
           AA0=AA0+AA1
           AA0=AA0 << NLSRE
           RREC(I+1)=RND(AA0)             | upper 16 bits of AA0 saved
       Go to FIN_RECUR

                                          | Case 3: NLSRREC < NLSAA0
    If NLSRREC < NLSAA0, do the next 21 lines
       IR=NLSAA0-NLSRREC+1
       AA0=AA0 >> IR
       AA1=RREC(1) << NLSATT              | scale RREC by 3/4 or 1/2
       AA1=-AA1+(RREC(1) << 16)           |
       AA1=AA1 >> 1
       AA0=AA0+AA1
       Call VSCALE(AA0,1,1,30,AA0,NLSRE)
       RREC(1)=RND(AA0)                   | upper 16 bits of AA1 saved
       NLSRREC=NLSRREC-1+NLSRE
       For I=1,2,...,LPO, do the next 11 lines
           AA0=0                          | compute recursive part of RREC(I+1)
           For N=LPO+1,...,N1, do the next 2 lines
               P=WS(N)*WS(N-I)
               AA0=AA0+P
           AA0=AA0 >> IR
           AA1=RREC(I+1) << NLSATT        | scale RREC by 3/4 or 1/2
           AA1=-AA1+(RREC(I+1) << 16)     |
           AA1=AA1 >> 1
           AA0=AA0+AA1
           AA0=AA0 << NLSRE
           RREC(I+1)=RND(AA0)             | upper 16 bits of AA0 saved

FIN_RECUR:                                | when you reach this point the
                                          | recursive component has been computed

    AA0=0                                 | compute non-recursive part of R(1)
    For N=N1+1,...,N3, do the next 2 lines
       P=WS(N)*WS(N)
       AA0=AA0+P

    LPFE = LPO
    If FERROR .=. TRUE, set LPFE = 10
                                          | Case 1: NLSRREC > NLSAA0
    If NLSRREC > NLSAA0, do the next 21 lines
       IR=NLSRREC-NLSAA0+1
       AA1=RREC(1) << 16
       AA1=AA1 >> IR
       AA0=AA0 >> 1
       AA1=AA0+AA1
       AA0=AA1 >> 8                       | apply white noise correction factor
       AA1=AA1+AA0
       Call VSCALE(AA1,1,1,30,AA1,NLSRR)
       R(1)=RND(AA1)                      | upper 16 bits of AA1 saved
       For I=1,2,...,LPFE, do the next 10 lines
           AA0=0                          | compute non-recursive part of R(I+1)
           For N=N1+1,...,N3, do the next 2 lines
               P=WS(N)*WS(N-I)
               AA0=AA0+P
           AA0=AA0 >> 1
           AA1=RREC(I+1) << 16
           AA1=AA1 >> IR
           AA1=AA0+AA1
           AA1=AA1 << NLSRR
```

```
        R(I+1)=RND(AA1)                | save upper 16 bits
     Go to END

                                       | Case 2: NLSRREC = NLSAA0
  If NLSRREC = NLSAA0, do the next 18 lines
     AA0=AA0 >> 1
     AA1=RREC(1) << 15                 | this can be done by multiplication
     AA1=AA0+AA1
     AA0=AA1 >> 8                      | apply white noise correction factor
     AA1=AA1+AA0
     Call VSCALE(AA1,1,1,30,AA1,NLSRR)
     R(1)=RND(AA1)                     | upper 16 bits of AA1 saved
     For I=1,2,...,LPFE, do the next 9 lines
         AA0=0                         | compute non-recursive part of R(I+1)
         For N=N1+1,...,N3, do the next 2 lines
             P=WS(N)*WS(N-I)
             AA0=AA0+P
         AA0=AA0 >> 1
         AA1=RREC(I+1) << 15
         AA1=AA0+AA1
         AA1=AA1 << NLSRR
         R(I+1)=RND(AA1)               | save upper 16 bits
     Go to END

                                       | Case 3: NLSRREC < NLSAA0
  If NLSRREC < NLSAA0, do the next 18 lines
     IR=NLSAA0-NLSRREC+1
     AA0=AA0 >> IR
     AA1=RREC(1) << 15                 | this can be done by multiplication
     AA1=AA0+AA1
     AA0=AA1 >> 8                      | apply white noise correction factor
     AA1=AA1+AA0
     Call VSCALE(AA1,1,1,30,AA1,NLSRR)
     R(1)=RND(AA1)                     | upper 16 bits of AA1 saved
     For I=1,2,...,LPFE, do the next 9 lines
         AA0=0                         | compute non-recursive part of R(I+1)
         For N=N1+1,...,N3, do the next 2 lines
             P=WS(N)*WS(N-I)
             AA0=AA0+P
         AA0=AA0 >> IR
         AA1=RREC(I+1) << 15
         AA1=AA0+AA1
         AA1=AA1 << NLSRR
         R(I+1)=RND(AA1)               | save upper 16 bits

END:                                   | one last job, check for ill-conditioning
  ILLCOND=.FALSE.
  If AA1 = 0, set ILLCOND=.TRUE.       | AA1 still contains 32 bit R(LPFE+1)
```

I.5.8 Block 51FE – Bandwidth expansion module for frame erasures

This is the floating-point pseudo-code for Block 51FE, the bandwidth expansion module for the LPC synthesis filter for frame erasure conditions. This module is similar to Block 51 in 5.6/G.728, except that the bandwidth broadening coefficients are different. Also, the input values are the A(I) array.

```
     If FECOUNT = 1 and ILLCOND = .FALSE., do the next 2 lines
         For I=2,3,...,LPC+1, do the next line
             A(I)=FACVFE(I) * ATMP(I)          | scale coeff.
     Otherwise, do the next 2 lines
         For I=2,3,...,LPC+1, do the next line
             A(I)=FACVFE(I) * A(I)             | scale coeff.
```

The tables for FACVFE are given in Q14 format. The values for the input A array are always in Q14 format. The final values of A can always be represented in Q14 format. Thus, a shift is necessary after the multiplication. This is the fixed-point pseudo-code for Block 51.

```
If FECOUNT = 1 and ILLCOND = .FALSE., do the next 4 lines
    For I=2,3,4,...,LPC+1, do the next 3 lines
        AA0 = FACVFE(I) * ATMP(I)  | AA0 is Q28
        AA0 = AA0 << 2             | make AA0 Q30
        A(I) = RND(AA0)            | round to high word and get Q14
Otherwise, do the next 4 lines
    For I=2,3,4,...,LPC+1, do the next 3 lines
        AA0 = FACVFE(I) * A(I)     | AA0 is Q28
        AA0 = AA0 << 2             | make AA0 Q30
        A(I) = RND(AA0)            | round to high word and get Q14
```

I.5.9 Block 97FE – Update GSTATE during frame erasures

We begin with the floating-point version of this new block. This is the same code as Blocks 67, 39, 40 and 42 described in 5.7/G.728.

```
ETRMS = 0.
For K = 1,...,IDIM, do the next line   |  compute energy
    ETRMS = ETRMS + ET(K)*ET(K)        |  of excitation
ETRMS = ETRMS * DIMINV
If ETRMS < 1., set ETRMS = 1.          |  check lower limit
ETRMS = 10.* log10 (ETRMS)             |  compute dB value

GSTATE(1) = ETRMS - GOFF               | subtract dB gain offset
```

The fixed-point version of this new block performs the equivalent operations. The $\log_{10}$ function was eliminated from the original fixed-point specification of G.728. Now it must be re-introduced in order to perform the above calculation. Because we are not trying to exactly match the output of the floating-point specification, we do not need the greatest possible precision for the output of $\log_{10}$. Here is the fixed-point code. Also, because bit exact compatibility is not required for frame erasures, we have not included a bit exact description of the LIN2DB function for fixed-point implementation. This LIN2DB convert a scalar floating-point representation of ETRMS to the dB value represented in Q9. Any fixed-point implementation of LIN2DB with reasonable accuracy can be used here. An example of such a LIN2DB fixed-point code is included here.

```
AA0 = 0
For K = 1,...,IDIM, do the next line   |  compute energy of excitation
    AA0 = AA0 + ET(K)*ET(K)            |  ET is Q2, so AA0 is Q4

Call VSCALE(AA0,1,1,30,AA0,NLS)        |  scale AA0
ETRMS = AA0 >> 16                      |  take high word of AA0
NLSETRMS = (4 + NLS) - 16              |  NLS of ETRMS
AA0 = ETRMS * DIMINV                   |  DIMINV = 0.2 in Q16
Call VSCALE(AA0,1,1,30,AA0,NLS)        |  scale AA0
ETRMS = AA0 >> 16                      |  take high word of AA0
NLSETRMS = NLSETRMS + NLS              |  NLS of ETRMS
If NLSETRMS > 14, do the next 2 lines  |  if ETRMS is Q15 or
                                       |  smaller (<1)
    ETRMS = 16384                      |  set ETRMS to 1 in Q16
    NLSETRMS = 14
Call LIN2DB(ETRMS,NLSETRMS,GSTATE(1))  | convert linear to dB (fixed
                                       | Q9)
                                       | gain offset subtraction is done
                                       | inside LIN2DB
```

The constants needed for the LIN2DB fixed-point code is listed below.

c(0)	=	−3 400	=	−0.4150374993 in Q13
c(1)	=	15 758	=	1.9235933879 in Q13
c(2)	=	−10 505	=	−1.2823955919 in Q13
c(3)	=	9 338	=	1.1399071928 in Q13
c(4)	=	−9 338	=	−1.1399071928 in Q13
c(5)	=	9 961	=	1.2159010057 in Q13
FAC1	=	24 660	=	3.0102999566 in Q13
GOFF	=	−16 384	=	−32 in Q9
THRQTR	=	24 576	=	0.75 in Q15

The LIN2DB fixed-point code is given below. It calculates a truncated Taylor series expansion of $\log_2$ function around 0.75. It assumes the input X is a positive normalized 16-bit mantissa with NLSX as the Q format. It performs the necessary scaling to get the dB value and subtracts the gain offset GOFF from the resulting 32-bit number before converting to the Q9 fixed Q format output.

```
LIN2DB(X, NLSX, RET)
AA0 = x                  | normalized .5 <= x <= 1, Q15
AA0 = AA0 - THRQTR       | subtract .75, now -.25 <= x .25, Q15
AA0 = AA0 << 1           | convert to Q16
AA1 = 0
For I=5,4...,1 do the next 3 lines
        AA1 = AA1 + C(I) | Q13
        P = AA1 * AA0    | Q13 * Q16 = Q29
        AA1 = P >> 16    | Q29 -> Q13
AA1 = AA1 + C(0)         | Q13
AA1 = AA1 >> 3           | convert to Q10
AA0 = 15 - NLSX          | exponent in Q0
AA0 = AA0 << 10          | convert exponent to Q10
AA1 = AA0 + AA1          | Add exp to log of mantissa, AA1 now log2(x)
AA1 = AA1 * FAC1         | Q10 * Q13 = Q23, convert to 10*log10(x)
AA1 = AA1 >> 14          | Q23 -> Q9
AA1 = AA1 - GOFF         | Subtract GOFF
RET = AA1                | return Q9 value
```

I.5.10 Block 98AF – Log-gain limiter after frame erasure

This new block should replace Block 98 in G.3.16/G.728. This is the fixed-point pseudo-code for Block 98AF.

```
If LOGGAIN > 14336, set LOGGAIN=14336
If LOGGAIN < -16384, set LOGGAIN=-16384
TMP = LOGGAIN - OGAINDB
If AFTERFE > 0 and TMP > FEGAINMAX, do the next line
    LOGGAIN = OGAINDB + FEGAINMAX
```

I.6 Additional coder parameters and variables

This clause contains a table with the new variables that have been added to the G.728 decoder for frame erasure concealment. Under the format column, we list the fixed-point format. For a floating-point implementation, all block floating-point or fixed Q format variables can be represented in floating-point.

Table I.1/G.728 – Additional coder parameters needed for frame erasure concealment

Name	Floating-point value	Fixed-point value	Q format	Description
AFTERFEMAX	16	16	Q0	Maximum 2.5 ms frames for gain clipping
DIMINV	0.2	13 107	Q16	Inverse of vector dimension
FEGAINMAX	2	1 024	Q9	Maximum gain DB increase after frame erasure
VTH	0.4285714	7 022	Q14	Voicing threshold for frame erasures
FESCALE	0.8	26 214	Q15	ETPAST scaling factor for frame erasures

Table I.2/G.728 – Additional LD-CELP internal processing variables needed for frame erasure concealment

Name	Array index range	Fixed-point format	Description
MAG	1	a)	Average magnitude of current excitation vector
ADCOUNT	1	Integer	G.728 Adaptation cycle counter (2.5 ms)
AFTERFE	1	Integer	Counter for gain clipping after FE (2.5 ms)
AVMAG	1	SFL	Average magnitude of past 8 excitation vectors
ETPAST	–139 to IDIM	Q2	Past excitation (initially zero)
ETRMS	1	SFL	Energy in dB of current vector
FECOUNT	1	Integer	Length of current frame erasure (2.5 ms)
FEDELAY	1	Integer	Pitch (KP) stored at beginning of FE
FERROR	1	Logical	Frame erasure indicator
FESCALE	1	Integer	ETPAST scaling factor for frame erasures
FESIZE	1	Integer	Number of 2.5 ms Adaptation cycles in a FE
NLSFESCALE	1	Integer	NLS for FESCALE
NLSAVMAG	1	Integer	NLS for AVGMAG
NLSETRMS	1	Integer	NLS for ETRMS
OGAINDB	1	Q9	Old prediction gain in dB
VOICED	1	Logical	Voiced/Unvoiced flag

a) MAG is not used in the fixed-point pseudo-code.

APPENDIX I.I

Values used for scaling ETPAST during frame erasures

Tables I.I.1 and I.I.2 give the integer and floating-point values used to scale ETPAST for both voiced (VOICEDFEGAIN) and non-voiced (UNVOICEDFEGAIN) signals during frame erasures. For the fixed-point these are in Q15 format.

Table I.I.1/G.728 – Values used to scale ETPAST with voiced signals (VOICEDFEGAIN)

VOICEDFEGAIN index	Floating-point value	Fixed-point value Q15
0	0.8	26 214
1	0.8	26 214
2	0.6	19 661
3	0.4	13 107
4	0.2	6 554

Table I.I.2/G.728 – Values used to scale ETPAST with non-voiced signals (UNVOICEDFEGAIN)

VOICEDFEGAIN index	Floating-point value	Fixed-point value Q15
0	1.0	32 767
1	1.0	32 767
2	0.8	26 214
3	0.6	19 661
4	0.4	13 107
5	0.2	6 554

Values used for bandwidth expansion of LPC predictor during frame erasures

The following table gives the integer values for the bandwidth expansion coefficient array FACVFE used to soften the LPC predictor during frame erasures. These values are in Q14 format. To obtain the equivalent floating-point values, divide the integer values by 16384. Starting at FACVFE(1) and ending at FACVFE(51), this table should be read from left to right, and then from top to bottom.

```
16384   15892   15416   14953   14505
14069   13647   13238   12841   12456
12082   11719   11368   11027   10696
10375   10064    9762    9469    9185
 8910    8642    8383    8131    7888
 7651    7421    7199    6983    6773
 6570    6373    6182    5996    5816
 5642    5473    5309    5149    4995
 4845    4700    4559    4422    4289
 4161    4036    3915    3797    3683
 3573
```

ITU-T RECOMMENDATIONS SERIES

Series A	Organization of the work of the ITU-T
Series B	Means of expression: definitions, symbols, classification
Series C	General telecommunication statistics
Series D	General tariff principles
Series E	Overall network operation, telephone service, service operation and human factors
Series F	Non-telephone telecommunication services
Series G	**Transmission systems and media, digital systems and networks**
Series H	Audiovisual and multimedia systems
Series I	Integrated services digital network
Series J	Transmission of television, sound programme and other multimedia signals
Series K	Protection against interference
Series L	Construction, installation and protection of cables and other elements of outside plant
Series M	TMN and network maintenance: international transmission systems, telephone circuits, telegraphy, facsimile and leased circuits
Series N	Maintenance: international sound programme and television transmission circuits
Series O	Specifications of measuring equipment
Series P	Telephone transmission quality, telephone installations, local line networks
Series Q	Switching and signalling
Series R	Telegraph transmission
Series S	Telegraph services terminal equipment
Series T	Terminals for telematic services
Series U	Telegraph switching
Series V	Data communication over the telephone network
Series X	Data networks and open system communications
Series Y	Global information infrastructure and Internet protocol aspects
Series Z	Languages and general software aspects for telecommunication systems