SCTE · ISBE

STANDARDS

Data Standards Subcommittee

SCTE STANDARD

SCTE 24-22 2018

iLBCv2.0 Speech Codec Specification for Voice over IP Applications in Cable Telephony

Documents
Document Sharing Hub

Full document details: SCTE 24-22 2018

NOTICE

The Society of Cable Telecommunications Engineers (SCTE) / International Society of Broadband Experts (ISBE) Standards and Operational Practices (hereafter called "documents") are intended to serve the public interest by providing specifications, test methods and procedures that promote uniformity of product, interchangeability, best practices and ultimately the long-term reliability of broadband communications facilities. These documents shall not in any way preclude any member or non-member of SCTE•ISBE from manufacturing or selling products not conforming to such documents, nor shall the existence of such standards preclude their voluntary use by those other than SCTE•ISBE members.

SCTE•ISBE assumes no obligations or liability whatsoever to any party who may adopt the documents. Such adopting party assumes all risks associated with adoption of these documents, and accepts full responsibility for any damage and/or claims arising from the adoption of such documents.

Attention is called to the possibility that implementation of this document may require the use of subject matter covered by patent rights. By publication of this document, no position is taken with respect to the existence or validity of any patent rights in connection therewith. SCTE•ISBE shall not be responsible for identifying patents for which a license may be required or for conducting inquiries into the legal validity or scope of those patents that are brought to its attention.

Patent holders who believe that they hold patents which are essential to the implementation of this document have been requested to provide information about those patents and any related licensing terms and conditions. Any such declarations made before or after publication of this document are available on the SCTE•ISBE web site at http://www.scte.org.

All Rights Reserved

© Society of Cable Telecommunications Engineers, Inc. 2018
140 Philips Road
Exton, PA 19341

Documents
Document Sharing Hub

Full document details: SCTE 24-22 2018

Abstract

This document specifies a speech codec suitable for robust voice communication over IP. It is designed for narrow band speech and results in a payload bit rate of 13.33 kbit/s for 30 ms frames and 15.20 kbit/s for 20 ms frames. The codec enables graceful speech quality degradation in the case of lost frames, which occurs in connection with lost or delayed IP packets.

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

Table of Contents

1 INTRODUCTION 8

2 REFERENCES 8

THE FOLLOWING DOCUMENTS CONTAIN PROVISIONS, WHICH, THROUGH REFERENCE IN THIS TEXT, CONSTITUTE PROVISIONS OF THE STANDARD. AT THE TIME OF SUBCOMMITTEE APPROVAL, THE EDITIONS INDICATED WERE VALID. ALL STANDARDS ARE SUBJECT TO REVISION; AND WHILE PARTIES TO ANY AGREEMENT BASED ON THIS STANDARD ARE ENCOURAGED TO INVESTIGATE THE POSSIBILITY OF APPLYING THE MOST RECENT EDITIONS OF THE DOCUMENTS LISTED BELOW, THEY ARE REMINDED THAT NEWER EDITIONS OF THOSE DOCUMENTS MAY NOT BE COMPATIBLE WITH THE REFERENCED VERSION. 8

2.1 NORMATIVE REFERENCES 8

2.2 INFORMATIVE REFERENCES 8

3 ABBREVIATIONS AND ACRONYMS 8

4 CONVENTIONS 9

5 OUTLINE OF THE CODEC 9

5.1 ENCODER 9

5.2 DECODER 10

6 ENCODER PRINCIPLES 10

6.1 PRE-PROCESSING 11

6.2 LPC ANALYSIS AND QUANTIZATION 11

6.2.1 Computation of Autocorrelation Coefficients 12

6.2.2 Computation of LPC Coefficients 13

6.2.3 Computation of LSF Coefficients from LPC Coefficients 13

6.2.4 Quantization of LSF Coefficients 13

6.2.5 Stability Check of LSF Coefficients 14

6.2.6 Interpolation of LSF Coefficients 14

6.2.7 LPC Analysis and Quantization for 20 ms Frames 14

6.3 CALCULATION OF THE RESIDUAL 15

6.4 PERCEPTUAL WEIGHTING FILTER 15

6.5 START STATE ENCODER 15

6.5.1 Start State Estimation 15

6.5.2 All-Pass Filtering and Scale Quantization 16

6.5.3 Scalar Quantization 17

6.6 ENCODING THE REMAINING SAMPLES 17

6.6.1 Codebook Memory 18

6.6.2 Perceptual Weighting of Codebook Memory and Target 19

6.6.3 Codebook Creation 20

6.6.4 Codebook Search 22

6.7 GAIN CORRECTION ENCODING 23

6.8 BITSTREAM DEFINITION 24

7 DECODER PRINCIPLES 26

7.1 LPC FILTER RECONSTRUCTION 27

7.2 START STATE RECONSTRUCTION 27

7.3 EXCITATION DECODING LOOP 28

7.4 MULTISTAGE ADAPTIVE CODEBOOK DECODING 28

7.4.1 Construction of the Decoded Excitation Signal 28

7.5 PACKET LOSS CONCEALMENT 28

7.5.1 Block Received Correctly and Previous Block Also Received 28

7.5.2 Block Not Received 28

7.5.3 Block Received Correctly When Previous Block Not Received 29

7.6 ENHANCEMENT 29

Documents Document Sharing Hub

Full document details: SCTE 24-22 2018

7.6.1 Estimating the Pitch *30*
7.6.2 Determination of the Pitch-Synchronous Sequences *30*
7.6.3 Calculation of the Smoothed Excitation *31*
7.6.4 Enhancer Criterion *32*
7.6.5 Enhancing the excitation *32*
7.7 SYNTHESIS FILTERING 33
7.8 POST FILTERING 33

8 SECURITY CONSIDERATIONS 33

APPENDIX A REFERENCE IMPLEMENTATION 34

A.1 iLBC_TEST.C 35
A.2 iLBC_ENCODE.H 39
A.3 iLBC_ENCODE.C 40
A.4 iLBC_DECODE.H 47
A.5 iLBC_DECODE.C 48
A.6 iLBC_DEFINE.H 57
A.7 CONSTANTS.H 60
A.8 CONSTANTS.C 62
A.9 ANAFILTER.H 73
A.10 ANAFILTER.C 73
A.11 CREATECB.H 74
A.12 CREATECB.C 75
A.13 DOCPLC.H 78
A.14 DOCPLC.C 79
A.15 ENHANCER.H 82
A.16 ENHANCER.C 83
A.17 FILTER.H 93
A.18 FILTER.C 94
A.19 FRAMECLASSIFY.H 97
A.20 FRAMECLASSIFY.C 97
A.21 GAINQUANT.H 99
A.22 GAINQUANT.C 99
A.23 GETCBVEC.H 101
A.24 GETCBVEC.C 101
A.25 HELPFUN.H 104
A.26 HELPFUN.C 105
A.27 HPINPUT.H 110
A.28 HPINPUT.C 110
A.29 HPOUTPUT.H 111
A.30 HPOUTPUT.C 112
A.31 ICBCONSTRUCT.H 112
A.32 ICBCONSTRUCT.C 113
A.33 ICBSEARCH.H 115
A.34 ICBSEARCH.C 115
A.35 LPCDECODE.H 122
A.36 LPCDECODE.C 123
A.37 LPCENCODE.H 125
A.38 LPCENCODE.C 126
A.39 LSF.H 129
A.40 LSF.C 129
A.41 PACKING.H 133
A.42 PACKING.C 134
A.43 STATECONSTRUCTW.H 137
A.44 STATECONSTRUCTW.C 137
A.45 STATESEARCHW.H 138
A.46 STATESEARCHW.C 139
A.47 SYNTFILTER.H 142

AVA Documents Document Sharing Hub
Full document details: SCTE 24-22 2018

A.48 SYNTFILTER.C 142

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

Figures

Figure 6.1. Flow chart of the iLBC encoder 11
Figure 6.2. One input block to the encoder for 20 ms (with four sub-frames) and 30 ms (with six sub-frames) 11
Figure 6.3. Quantization of start state samples by DPCM in the weighted speech domain....17
Figure 6.4. Flow chart of the codebook search in the iLBC encoder 18
Figure 6.5. The order from 1 to 5 in which the sub-blocks are encoded. The slashed area is the start state. 18
Figure 6.6. The codebook memory, length lMem=85 samples, and the target vector 1, length 22 samples 18
Figure 6.7. The codebook memory, length lMem=147 samples, and the target vector 2, length 40 samples 19
Figure 6.8. The codebook memory, length lMem=147 samples, and the target vector 3, length 40 samples 19
Figure 6.9. The codebook memory, length lMem=147 samples, and the target vector 4, length 40 samples 19
Figure 6.10. The codebook memory, length lMem=147 samples, and the target vector 5, length 40 samples 19
Figure 6.11. Generation of the first augmented codebook....21
Figure 6.12. Generation of the second augmented codebook....21
Figure 7.1. Flow chart of the iLBC decoder....26
Figure 7.2. Flow chart of the enhancer....30
Figure 7.3. Enhancement for 20 ms frame size 31
Figure 7.4. Enhancement for 30 ms frame size 31

Tables

Table 6.1. Codebook sizes for the 30 ms mode 20
Table 6.2. The bitstream definition for iLBC for both the 20 ms frame size mode and the 30 ms frame size mode 25

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

1 INTRODUCTION

This document contains the description of an algorithm for coding of speech signals sampled at 8 kHz. The algorithm, called iLBC, uses a block-independent linear-predictive coding (LPC) algorithm and has support for two basic frame lengths: 20 ms at 15.2 kbit/s and 30 ms at 13.33 kbit/s. When the codec operates at block lengths of 20 ms, it produces 304 bits per block, which SHOULD be packetized as in RFC 3952. Similarly, for block lengths of 30 ms it produces 400 bits per block, which SHOULD be packetized as in RFC 3952. The two modes for the different frame sizes operate in a very similar way. When they differ it is explicitly stated in the text, usually with the notation x/y, where x refers to the 20 ms mode and y refers to the 30 ms mode.

The described algorithm results in a speech coding system with a controlled response to packet losses similar to what is known from pulse code modulation (PCM) with packet loss concealment (PLC), such as the ITU-T G.711 standard, which operates at a fixed bit rate of 64 kbit/s. At the same time, the described algorithm enables fixed bit rate coding with a quality-versus-bit rate tradeoff close to state-of-the-art. A suitable RTP payload format for the iLBC codec is specified in RFC 3952.

Some of the applications for which this coder is suitable are real time communications such as telephony and videoconferencing, streaming audio, archival, and messaging.

This document is organized as follows. Section 5 gives a brief outline of the codec. The specific encoder and decoder algorithms are explained in sections 6 and 7, respectively. Appendix A provides a c-code reference implementation.

2 REFERENCES

The following documents contain provisions, which, through reference in this text, constitute provisions of the standard. At the time of Subcommittee approval, the editions indicated were valid. All standards are subject to revision; and while parties to any agreement based on this standard are encouraged to investigate the possibility of applying the most recent editions of the documents listed below, they are reminded that newer editions of those documents may not be compatible with the referenced version.

2.1 Normative References

This document does not contain any normative references.

2.2 Informative References

The following documents may provide valuable information to the reader but are not required when complying with this standard.

G.711	ITU-T Rec. G.711, Pulse code modulation (PCM) of voice frequencies
RFC 3952	IETF RFC 3952, Duric, A. and S. Andersen, "Real-time Transport Protocol (RTP) Payload Format for internet Low Bit Rate Codec (iLBC) Speech", December 2004.
RFC 3711	IETF RFC 3711, Baugher, M., McGrew, D., Naslund, M., Carrara, E., and K. Norman, "The Secure Real Time Transport Protocol (SRTP)", March 2004

3 ABBREVIATIONS AND ACRONYMS

This document uses the following abbreviations or acronyms.

LPC Linear Predictive Coding

LSF Line Spectral Frequencies

PLC Packet Loss Concealment

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

VQ Vector Quantization

4 CONVENTIONS

Throughout this document words that are used to define the significance of particular requirements are capitalized. These words are:

"MUST"	This word or the adjective "REQUIRED" means that the item is an absolute requirement of this specification.
"MUST NOT"	This phrase means that the item is an absolute prohibition of this specification.
"SHOULD"	This word or the adjective "RECOMMENDED" means that there may exist valid reasons in particular circumstances to ignore this item, but the full implications should be understood and the case carefully weighed before choosing a different course.
"SHOULD NOT"	This phrase means that there may exist valid reasons in particular circumstances when the listed behavior is acceptable or even useful, but the full implications should be understood and the case carefully weighed before implementing any behavior described with this label.
"MAY"	This word or the adjective "OPTIONAL" means that this item is truly optional. One vendor may choose to include the item because a particular marketplace requires it or because it enhances the product, for example; another vendor may omit the same item.

5 OUTLINE OF THE CODEC

The codec consists of an encoder and a decoder as described in sections 5.1 and 5.2, respectively.

The essence of the codec is LPC and block-based coding of the LPC residual signal. For each 160/240 (20 ms/30 ms) sample block, the following major steps are performed: A set of LPC filters are computed, and the speech signal is filtered through them to produce the residual signal. The codec uses scalar quantization of the dominant part, in terms of energy, of the residual signal for the block. The dominant state is of length 57/58 (20 ms/30 ms) samples and forms a start state for dynamic codebooks constructed from the already coded parts of the residual signal. These dynamic codebooks are used to code the remaining parts of the residual signal. By this method, coding independence between blocks is achieved, resulting in elimination of propagation of perceptual degradations due to packet loss. The method facilitates high-quality packet loss concealment (PLC).

5.1 Encoder

The input to the encoder SHOULD be 16 bit uniform PCM sampled at 8 kHz. It SHOULD be partitioned into blocks of BLOCKL=160/240 samples for the 20/30 ms frame size. Each block is divided into NSUB=4/6 consecutive sub-blocks of SUBL=40 samples each. For 30 ms frame size, the encoder performs two LPC_FILTERORDER=10 linear-predictive coding (LPC) analyses. The first analysis applies a smooth window centered over the second sub-block and extending to the middle of the fifth sub-block. The second LPC analysis applies a smooth asymmetric window centered over the fifth sub-block and extending to the end of the sixth sub-block. For 20 ms frame size, one LPC_FILTERORDER=10 linear-predictive coding (LPC) analysis is performed with a smooth window centered over the third sub-frame.

For each of the LPC analyses, a set of Line Spectral Frequencies (LSFs) are obtained, quantized, and interpolated to obtain LSF coefficients for each sub-block. Subsequently, the LPC residual is computed by using the quantized and interpolated LPC analysis filters.

The two consecutive sub-blocks of the residual exhibiting the maximal weighted energy are identified. Within these two sub-blocks, the start state (segment) is selected from two choices: the first 57/58 samples or the last 57/58 samples of the two consecutive sub-blocks. The selected segment is the one of higher energy. The start state is encoded with scalar quantization.

Documents Document Sharing Hub
Full document details: SCTE 24-22 2018

A dynamic codebook encoding procedure is used to encode 1) the 23/22 (20 ms/30 ms) remaining samples in the two sub-blocks containing the start state; 2) the sub-blocks after the start state in time; and 3) the sub-blocks before the start state in time. Thus, the encoding target can be either the 23/22 samples remaining of the two sub- blocks containing the start state or a 40-sample sub-block. This target can consist of samples indexed forward in time or backward in time, depending on the location of the start state.

The codebook coding is based on an adaptive codebook built from a codebook memory that contains decoded LPC excitation samples from the already encoded part of the block. These samples are indexed in the same time direction as the target vector, ending at the sample instant prior to the first sample instant represented in the target vector. The codebook is used in CB_NSTAGES=3 stages in a successive refinement approach, and the resulting three code vector gains are encoded with 5-, 4-, and 3-bit scalar quantization, respectively.

The codebook search method employs noise shaping derived from the LPC filters, and the main decision criterion is to minimize the squared error between the target vector and the code vectors. Each code vector in this codebook comes from one of CB_EXPAND=2 codebook sections. The first section is filled with delayed, already encoded residual vectors. The code vectors of the second codebook section are constructed by predefined linear combinations of vectors in the first section of the codebook.

As codebook encoding with squared-error matching is known to produce a coded signal of less power than does the scalar quantized start state signal, a gain re-scaling method is implemented by a refined search for a better set of codebook gains in terms of power matching after encoding. This is done by searching for a higher value of the gain factor for the first stage codebook, as the subsequent stage codebook gains are scaled by the first stage gain.

5.2 Decoder

Typically for packet communications, a jitter buffer placed at the receiving end decides whether the packet containing an encoded signal block has been received or lost. This logic is not part of the codec described here. For each encoded signal block received the decoder performs a decoding. For each lost signal block, the decoder performs a PLC operation.

The decoding for each block starts by decoding and interpolating the LPC coefficients. Subsequently the start state is decoded.

For codebook-encoded segments, each segment is decoded by constructing the three code vectors given by the received codebook indices in the same way that the code vectors were constructed in the encoder. The three gain factors are also decoded and the resulting decoded signal is given by the sum of the three codebook vectors scaled with respective gain.

An enhancement algorithm is applied to the reconstructed excitation signal. This enhancement augments the periodicity of voiced speech regions. The enhancement is optimized under the constraint that the modification signal (defined as the difference between the enhanced excitation and the excitation signal prior to enhancement) has a short-time energy that does not exceed a preset fraction of the short-time energy of the excitation signal prior to enhancement.

A packet loss concealment (PLC) operation is easily embedded in the decoder. The PLC operation can, e.g., be based on repeating LPC filters and obtaining the LPC residual signal by using a long-term prediction estimate from previous residual blocks.

6 ENCODER PRINCIPLES

The following block diagram is an overview of all the components of the iLBC encoding procedure. The description of the blocks contains references to the section where that particular procedure is further described.

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
          +-----------+    +---------+    +---------+
speech -> | 1. Pre P  | -> | 2. LPC  | -> | 3. Ana  | ->
          +-----------+    +---------+    +---------+

          +---------------+   +--------------+
       -> | 4. Start Sel  | ->| 5. Scalar Qu | ->
          +---------------+   +--------------+

          +--------------+    +---------------+
       -> |6. CB Search  | -> | 7. Packetize  | -> payload
       |  +--------------+ |  +---------------+
       ----<---------<------
    sub-frame 0..2/4 (20 ms/30 ms)
```

Figure 6.1. Flow chart of the iLBC encoder

1. Pre-process speech with a high-pass filter, if needed (section 6.1).

2. Compute LPC parameters, quantize, and interpolate (section 6.2).

3. Use analysis filters on speech to compute residual (section 6.3).

4. Select position of 57/58-sample start state (section 6.5).

5. Quantize the 57/58-sample start state with scalar quantization (section 6.5).

6. Search the codebook for each sub-frame. Start with 23/22 sample block, then encode sub-blocks forward in time, and then encode sub-blocks backward in time. For each block, the steps in Figure 6.4 are performed (section 6.6).

7. Packetize the bits into the payload specified in Table 6.2.

The input to the encoder SHOULD be 16-bit uniform PCM sampled at 8 kHz. Also it SHOULD be partitioned into blocks of BLOCKL=160/240 samples. Each block input to the encoder is divided into NSUB=4/6 consecutive sub-blocks of SUBL=40 samples each.

```
            0        39        79       119       159
            +----------------------------------------+
            |    1    |    2    |    3    |    4    |
            +----------------------------------------+
                           20 ms frame

0        39        79       119       159       199       239
+------------------------------------------------------------+
|    1    |    2    |    3    |    4    |    5    |    6    |
+------------------------------------------------------------+
                                30 ms frame
```

Figure 6.2. One input block to the encoder for 20 ms (with four sub-frames) and 30 ms (with six sub-frames)

6.1 Pre-processing

In some applications, the recorded speech signal contains DC level and/or 50/60 Hz noise. If these components have not been removed prior to the encoder call, they should be removed by a high-pass filter. A reference implementation of this, using a filter with a cutoff frequency of 90 Hz, can be found in Appendix A.28.

6.2 LPC Analysis and Quantization

The input to the LPC analysis module is a possibly high-pass filtered speech buffer, speech_hp, that contains 240/300 (LPC_LOOKBACK + BLOCKL = 80/60 + 160/240 = 240/300) speech samples, where samples 0 through 79/59 are from the previous block and samples 80/60 through 239/299 are from the current block. No look-ahead into the next block is used. For the very first block processed, the look-back samples are assumed to be zeros.

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

For each input block, the LPC analysis calculates one/two set(s) of LPC_FILTERORDER=10 LPC filter coefficients using the autocorrelation method and the Levinson-Durbin recursion. These coefficients are converted to the Line Spectrum Frequency representation. In the 20 ms case, the single lsf set represents the spectral characteristics as measured at the center of the third sub-block. For 30 ms frames, the first set, lsf1, represents the spectral properties of the input signal at the center of the second sub-block, and the other set, lsf2, represents the spectral characteristics as measured at the center of the fifth sub-block. The details of the computation for 30 ms frames are described in sections 6.2.1 through 6.2.6. Section 6.2.7 explains how the LPC Analysis and Quantization differs for 20 ms frames.

6.2.1 Computation of Autocorrelation Coefficients

The first step in the LPC analysis procedure is to calculate autocorrelation coefficients by using windowed speech samples. This windowing is the only difference in the LPC analysis procedure for the two sets of coefficients. For the first set, a 240-sample-long standard symmetric Hanning window is applied to samples 0 through 239 of the input data. The first window, lpc_winTbl, is defined as

```
lpc_winTbl[i]= 0.5 * (1.0 - cos((2*PI*(i+1))/(BLOCKL+1)));
         i=0,...,119
lpc_winTbl[i] = winTbl[BLOCKL - i - 1]; i=120,...,239
```

The windowed speech speech_hp_win1 is then obtained by multiplying the first 240 samples of the input speech buffer with the window coefficients:

```
speech_hp_win1[i] = speech_hp[i] * lpc_winTbl[i];
         i=0,...,BLOCKL-1
```

From these 240 windowed speech samples, 11 (LPC_FILTERORDER + 1) autocorrelation coefficients, acf1, are calculated:

```
acf1[lag] += speech_hp_win1[n] * speech_hp_win1[n + lag];
         lag=0,...,LPC_FILTERORDER; n=0,...,BLOCKL-lag-1
```

In order to make the analysis more robust against numerical precision problems, a spectral smoothing procedure is applied by windowing the autocorrelation coefficients before the LPC coefficients are computed. Also, a white noise floor is added to the autocorrelation function by multiplying coefficient zero by 1.0001 (40dB below the energy of the windowed speech signal). These two steps are implemented by multiplying the autocorrelation coefficients with the following window:

```
lpc_lagwinTbl[0] = 1.0001;
lpc_lagwinTbl[i] = exp(-0.5 * ((2 * PI * 60.0 * i) /FS)^2);
         i=1,...,LPC_FILTERORDER
         where FS=8000 is the sampling frequency
```

Then, the windowed acf function acf1_win is obtained by

```
acf1_win[i] = acf1[i] * lpc_lagwinTbl[i];
         i=0,...,LPC_FILTERORDER
```

The second set of autocorrelation coefficients, acf2_win, are obtained in a similar manner. The window, lpc_asymwinTbl, is applied to samples 60 through 299, i.e., the entire current block. The window consists of two segments, the first (samples 0 to 219) being half a Hanning window with length 440 and the second a quarter of a cycle of a cosine wave. By using this asymmetric window, an LPC analysis centered in the fifth sub-block is obtained without the need for any look-ahead, which would add delay. The asymmetric window is defined as

```
lpc_asymwinTbl[i] = (sin(PI * (i + 1) / 441))^2; i=0,...,219

lpc_asymwinTbl[i] = cos((i - 220) * PI / 40); i=220,...,239
```

and the windowed speech is computed by

```
speech_hp_win2[i] = speech_hp[i + LPC_LOOKBACK] *
         lpc_asymwinTbl[i];  i=0,....BLOCKL-1
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

The windowed autocorrelation coefficients are then obtained in exactly the same way as for the first analysis instance.

The generation of the windows lpc_winTbl, lpc_asymwinTbl, and lpc_lagwinTbl are typically done in advance, and the arrays are stored in ROM rather than repeating the calculation for every block.

6.2.2 Computation of LPC Coefficients

From the 2 x 11 smoothed autocorrelation coefficients, acf1_win and acf2_win, the 2 x 11 LPC coefficients, lp1 and lp2, are calculated in the same way for both analysis locations by using the well known Levinson-Durbin recursion. The first LPC coefficient is always 1.0, resulting in ten unique coefficients.

After determining the LPC coefficients, a bandwidth expansion procedure is applied to smooth the spectral peaks in the short-term spectrum. The bandwidth addition is obtained by the following modification of the LPC coefficients:

```
lp1_bw[i] = lp1[i] * chirp^i; i=0,...,LPC_FILTERORDER
lp2_bw[i] = lp2[i] * chirp^i; i=0,...,LPC_FILTERORDER
```

where "chirp" is a real number between 0 and 1. It is RECOMMENDED to use a value of 0.9.

6.2.3 Computation of LSF Coefficients from LPC Coefficients

Thus far, two sets of LPC coefficients that represent the short-term spectral characteristics of the speech signal for two different time locations within the current block have been determined. These coefficients SHOULD be quantized and interpolated. Before this is done, it is advantageous to convert the LPC parameters into another type of representation called Line Spectral Frequencies (LSF). The LSF parameters are used because they are better suited for quantization and interpolation than the regular LPC coefficients. Many computationally efficient methods for calculating the LSFs from the LPC coefficients have been proposed in the literature. The detailed implementation of one applicable method can be found in Appendix A.26. The two arrays of LSF coefficients obtained, lsf1 and lsf2, are of dimension 10 (LPC_FILTERORDER).

6.2.4 Quantization of LSF Coefficients

Because the LPC filters defined by the two sets of LSFs are also needed in the decoder, the LSF parameters need to be quantized and transmitted as side information. The total number of bits required to represent the quantization of the two LSF representations for one block of speech is 40, with 20 bits used for each of lsf1 and lsf2.

For computational and storage reasons, the LSF vectors are quantized using three-split vector quantization (VQ). That is, the LSF vectors are split into three sub-vectors that are each quantized with a regular VQ. The quantized versions of lsf1 and lsf2, qlsf1 and qlsf2, are obtained by using the same memoryless split VQ. The length of each of these two LSF vectors is 10, and they are split into three sub-vectors containing 3, 3, and 4 values, respectively.

For each of the sub-vectors, a separate codebook of quantized values has been designed with a conventional VQ training method for a large database containing speech from a large number of speakers recorded under various conditions. The size of each of the three codebooks associated with the split definitions above is

```
int size_lsfCbTbl[LSF_NSPLIT] = {64,128,128};
```

The actual values of the vector quantization codebook that must be used can be found in the reference code of Appendix A. Both sets of LSF coefficients, lsf1 and lsf2, are quantized with a memoryless split vector quantization structure using the squared error criterion in the LSF domain. The split VQ consists of the following steps:

1) Quantize the first three LSF coefficients (1 - 3) with a VQ codebook of size 64.

2) Quantize the next three LSF coefficients 4 - 6 with a VQ codebook of size 128.

3) Quantize the last four LSF coefficients (7 - 10) with a VQ codebook of size 128.

This procedure, repeated for lsf1 and lsf2, gives six quantization indices and the quantized sets of LSF coefficients qlsf1 and qlsf2. Each set of three indices is encoded with 6 + 7 + 7 = 20 bits. The total number of bits used for LSF quantization in a block is thus 40 bits.

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

6.2.5 Stability Check of LSF Coefficients

The LSF representation of the LPC filter has the convenient property that the coefficients are ordered by increasing value, i.e., lsf(n-1) < lsf(n), 0 < n < 10, if the corresponding synthesis filter is stable. As we are employing a split VQ scheme, it is possible that at the split boundaries the LSF coefficients are not ordered correctly and hence that the corresponding LPC filter is unstable. To ensure that the filter used is stable, a stability check is performed for the quantized LSF vectors. If it turns out that the coefficients are not ordered appropriately (with a safety margin of 50 Hz to ensure that formant peaks are not too narrow), they will be moved apart. The detailed method for this can be found in Appendix A.40. The same procedure is performed in the decoder. This ensures that exactly the same LSF representations are used in both encoder and decoder.

6.2.6 Interpolation of LSF Coefficients

From the two sets of LSF coefficients that are computed for each block of speech, different LSFs are obtained for each sub-block by means of interpolation. This procedure is performed for the original LSFs (lsf1 and lsf2), as well as the quantized versions qlsf1 and qlsf2, as both versions are used in the encoder. Here follows a brief summary of the interpolation scheme; the details are found in the c-code of Appendix A. In the first sub-block, the average of the second LSF vector from the previous block and the first LSF vector in the current block is used. For sub-blocks two through five, the LSFs used are obtained by linear interpolation from lsf1 (and qlsf1) to lsf2 (and qlsf2), with lsf1 used in sub-block two and lsf2 in sub- block five. In the last sub-block, lsf2 is used. For the very first block it is assumed that the last LSF vector of the previous block is equal to a predefined vector, lsfmeanTbl, obtained by calculating the mean LSF vector of the LSF design database.

```
lsfmeanTbl[LPC_FILTERORDER] = {0.281738, 0.445801, 0.663330,
                0.962524, 1.251831, 1.533081, 1.850586, 2.137817,
                2.481445, 2.777344}
```

The interpolation method is standard linear interpolation in the LSF domain. The interpolated LSF values are converted to LPC coefficients for each sub-block. The unquantized and quantized LPC coefficients form two sets of filters respectively. The unquantized analysis filter for sub-block k is defined as follows

```
            ___
            \
Ak(z)= 1 + > ak(i)*z^(-i)
            /__
        i=1...LPC_FILTERORDER
```

The quantized analysis filter for sub-block k is defined as follows

```
            ___
            \
A~k(z)= 1 + > a~k(i)*z^(-i)
            /__
        i=1...LPC_FILTERORDER
```

A reference implementation of the lsf encoding is given in Appendix A.38. A reference implementation of the corresponding decoding can be found in Appendix A.36.

6.2.7 LPC Analysis and Quantization for 20 ms Frames

As previously stated, the codec only calculates one set of LPC parameters for the 20 ms frame size as opposed to two sets for 30 ms frames. A single set of autocorrelation coefficients is calculated on the LPC_LOOKBACK + BLOCKL = 80 + 160 = 240 samples. These samples are windowed with the asymmetric window lpc_asymwinTbl, centered over the third sub-frame, to form speech_hp_win. Autocorrelation coefficients, acf, are calculated on the 240 samples in speech_hp_win and then windowed exactly as in section 6.2.1 (resulting in acf_win).

This single set of windowed autocorrelation coefficients is used to calculate LPC coefficients, LSF coefficients, and quantized LSF coefficients in exactly the same manner as in sections 6.2.3 through 6.2.4. As for the 30 ms frame size, the ten LSF coefficients are divided into three sub-vectors of size 3, 3, and 4 and quantized by using the same scheme

Documents Document Sharing Hub
Full document details: SCTE 24-22 2018

and codebook as in section 6.2.4 to finally get 3 quantization indices. The quantized LSF coefficients are stabilized with the algorithm described in section 6.2.5.

From the set of LSF coefficients computed for this block and those from the previous block, different LSFs are obtained for each sub- block by means of interpolation. The interpolation is done linearly in the LSF domain over the four sub-blocks, so that the n-th sub- frame uses the weight (4-n)/4 for the LSF from old frame and the weight n/4 of the LSF from the current frame. For the very first block the mean LSF, lsfmeanTbl, is used as the LSF from the previous block. Similarly as seen in section 6.2.6, both unquantized, A(z), and quantized, A~(z), analysis filters are calculated for each of the four sub-blocks.

6.3 Calculation of the Residual

The block of speech samples is filtered by the quantized and interpolated LPC analysis filters to yield the residual signal. In particular, the corresponding LPC analysis filter for each 40 sample sub-block is used to filter the speech samples for the same sub- block. The filter memory at the end of each sub-block is carried over to the LPC filter of the next sub-block. The signal at the output of each LPC analysis filter constitutes the residual signal for the corresponding sub-block.

A reference implementation of the LPC analysis filters is given in Appendix A.10.

6.4 Perceptual Weighting Filter

In principle any good design of a perceptual weighting filter can be applied in the encoder without compromising this codec definition. However, it is RECOMMENDED to use the perceptual weighting filter Wk for sub-block k specified below:

```
Wk(z)=1/Ak(z/LPC_CHIRP_WEIGHTDENUM), where
                          LPC_CHIRP_WEIGHTDENUM = 0.4222
```

This is a simple design with low complexity that is applied in the LPC residual domain. Here Ak(z) is the filter obtained for sub-block k from unquantized but interpolated LSF coefficients.

6.5 Start State Encoder

The start state is quantized by using a common 6-bit scalar quantizer for the block magnitude and a 3-bit scalar quantizer operating on scaled samples in the weighted speech domain. In the following we describe the state encoding in greater detail.

6.5.1 Start State Estimation

The two sub-blocks containing the start state are determined by finding the two consecutive sub-blocks in the block having the highest power. Advantageously, down-weighting is used in the beginning and end of the sub-frames, i.e., the following measure is computed (NSUB=4/6 for 20/30 ms frame size):

```
nsub=1,...,NSUB-1
ssqn[nsub] = 0.0;
for (i=(nsub-1)*SUBL; i<(nsub-1)*SUBL+5; i++)
         ssqn[nsub] += sampEn_win[i-(nsub-1)*SUBL]*
                           residual[i]*residual[i];
for (i=(nsub-1)*SUBL+5; i<(nsub+1)*SUBL-5; i++)
         ssqn[nsub] += residual[i]*residual[i];
for (i=(nsub+1)*SUBL-5; i<(nsub+1)*SUBL; i++)
         ssqn[nsub] += sampEn_win[(nsub+1)*SUBL-i-1]*
                           residual[i]*residual[i];
```

where sampEn_win[5]={1/6, 2/6, 3/6, 4/6, 5/6}; MAY be used. The sub-frame number corresponding to the maximum value of ssqEn_win[nsub-1]*ssqn[nsub] is selected as the start state indicator. A weighting of ssqEn_win[]={0.8,0.9,1.0,0.9,0.8} for 30 ms frames and ssqEn_win[]={0.9,1.0,0.9} for 20 ms frames; MAY advantageously be used to bias the start state towards the middle of the frame.

For 20 ms frames there are three possible positions for the two-sub- block length maximum power segment; the start state position is encoded with 2 bits. The start state position, start, MUST be encoded as

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
start=1: start state in sub-frame 0 and 1
start=2: start state in sub-frame 1 and 2
start=3: start state in sub-frame 2 and 3
```

For 30 ms frames there are five possible positions of the two-sub- block length maximum power segment, the start state position is encoded with 3 bits. The start state position, start, MUST be encoded as

```
start=1: start state in sub-frame 0 and 1
start=2: start state in sub-frame 1 and 2
start=3: start state in sub-frame 2 and 3
start=4: start state in sub-frame 3 and 4
start=5: start state in sub-frame 4 and 5
```

Hence, in both cases, index 0 is not used. In order to shorten the start state for bit rate efficiency, the start state is brought down to STATE_SHORT_LEN=57 samples for 20 ms frames and STATE_SHORT_LEN=58 samples for 30 ms frames. The power of the first 23/22 and last 23/22 samples of the two sub-frame blocks identified above is computed as the sum of the squared signal sample values, and the 23/22-sample segment with the lowest power is excluded from the start state. One bit is transmitted to indicate which of the two possible 57/58 sample segments is used. The start state position within the two sub-frames determined above, state_first, MUST be encoded as

```
state_first=1: start state is first STATE_SHORT_LEN samples
state_first=0: start state is last STATE_SHORT_LEN samples
```

6.5.2 All-Pass Filtering and Scale Quantization

The block of residual samples in the start state is first filtered by an all-pass filter with the quantized LPC coefficients as denominator and reversed quantized LPC coefficients as numerator. The purpose of this phase-dispersion filter is to get a more even distribution of the sample values in the residual signal. The filtering is performed by circular convolution, where the initial filter memory is set to zero.

```
res(0..(STATE_SHORT_LEN-1))   = uncoded start state residual
res((STATE_SHORT_LEN)..(2*STATE_SHORT_LEN-1)) = 0

Pk(z) = A~rk(z)/A~k(z), where

                                ___
                                \
A~rk(z)= z^(-LPC_FILTERORDER)+>a~k(i+1)*z^(i-(LPC_FILTERORDER-1))
                                /__
                           i=0...(LPC_FILTERORDER-1)

and A~k(z) is taken from the block where the start state begins

res -> Pk(z) -> filtered

ccres(k) = filtered(k) + filtered(k+STATE_SHORT_LEN),
                                  k=0..(STATE_SHORT_LEN-1)
```

The all-pass filtered block is searched for its largest magnitude sample. The 10-logarithm of this magnitude is quantized with a 6-bit quantizer, state_frgqTbl, by finding the nearest representation.

This results in an index, idxForMax, corresponding to a quantized value, qmax. The all-pass filtered residual samples in the block are then multiplied with a scaling factor scal=4.5/(10^qmax) to yield normalized samples.

```
state_frgqTbl[64] = {1.000085, 1.071695, 1.140395, 1.206868,
                1.277188, 1.351503, 1.429380, 1.500727, 1.569049,
                1.639599, 1.707071, 1.781531, 1.840799, 1.901550,
                1.956695, 2.006750, 2.055474, 2.102787, 2.142819,
                2.183592, 2.217962, 2.257177, 2.295739, 2.332967,
                2.369248, 2.402792, 2.435080, 2.468598, 2.503394,
                2.539284, 2.572944, 2.605036, 2.636331, 2.668939,
                2.698780, 2.729101, 2.759786, 2.789834, 2.818679,
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
2.848074, 2.877470, 2.906899, 2.936655, 2.967804,
3.000115, 3.033367, 3.066355, 3.104231, 3.141499,
3.183012, 3.222952, 3.265433, 3.308441, 3.350823,
3.395275, 3.442793, 3.490801, 3.542514, 3.604064,
3.666050, 3.740994, 3.830749, 3.938770, 4.101764}
```

6.5.3 Scalar Quantization

The normalized samples are quantized in the perceptually weighted speech domain by a sample-by-sample scalar DPCM quantization as depicted in Figure 6.3. Each sample in the block is filtered by a weighting filter Wk(z), specified in section 6.4, to form a weighted speech sample x[n]. The target sample d[n] is formed by subtracting a predicted sample y[n], where the prediction filter is given by

```
Pk(z) = 1 - 1 / Wk(z).
```

```
            +-------+  x[n] +    d[n] +-----------+ u[n]
residual -->| Wk(z) |-------->(+)---->| Quantizer |------> quantized
            +-------+       - /|\     +-----------+    |   residual
                              |                       \|/
                         y[n] +---------------------->(+)
                              |                        |
                              |        +------+        |
                              +--------| Pk(z)|<------+
                                       +------+
```

Figure 6.3. Quantization of start state samples by DPCM in the weighted speech domain

The coded state sample u[n] is obtained by quantizing d[n] with a 3- bit quantizer with quantization table state_sq3Tbl.

```
state_sq3Tbl[8] = {-3.719849, -2.177490, -1.130005, -0.309692,
                   0.444214, 1.329712, 2.436279, 3.983887}
```

The quantized samples are transformed back to the residual domain by 1) scaling with 1/scal; 2) time-reversing the scaled samples; 3) filtering the time-reversed samples by the same all-pass filter, as in section 6.5.2, by using circular convolution; and 4) time- reversing the filtered samples. (More detail is in section 7.2.)

A reference implementation of the start-state encoding can be found in Appendix A.46.

6.6 Encoding the Remaining Samples

A dynamic codebook is used to encode 1) the 23/22 remaining samples in the two sub-blocks containing the start state; 2) the sub-blocks after the start state in time; and 3) the sub-blocks before the start state in time. Thus, the encoding target can be either the 23/22 samples remaining of the 2 sub-blocks containing the start state, or a 40-sample sub-block. This target can consist of samples that are indexed forward in time or backward in time, depending on the location of the start state. The length of the target is denoted by lTarget.

The coding is based on an adaptive codebook that is built from a codebook memory that contains decoded LPC excitation samples from the already encoded part of the block. These samples are indexed in the same time direction as is the target vector and end at the sample instant prior to the first sample instant represented in the target vector. The codebook memory has length lMem, which is equal to CB_MEML=147 for the two/four 40-sample sub-blocks and 85 for the 23/22-sample sub-block.

The following figure shows an overview of the encoding procedure.

Documents Document Sharing Hub
Full document details: SCTE 24-22 2018

```
   +------------+    +---------------+    +-------------+
-> | 1. Decode  | -> | 2. Mem setup  | -> | 3. Perc. W. | ->
   +------------+    +---------------+    +-------------+

   +------------+    +-----------------+
-> | 4. Search  | -> | 5. Upd. Target  | ------------------>
 | +------------+    +------------------ |
 ----<-------------<-----------<----------
               stage=0..2

   +----------------+
-> | 6. Recalc G[0] | ---------------> gains and CB indices
   +----------------+
```

Figure 6.4. Flow chart of the codebook search in the iLBC encoder

1. Decode the part of the residual that has been encoded so far, using the codebook without perceptual weighting.

2. Set up the memory by taking data from the decoded residual. This memory is used to construct codebooks. For blocks preceding the start state, both the decoded residual and the target are time reversed (section 6.6.1).

3. Filter the memory + target with the perceptual weighting filter (section 6.6.2).

4. Search for the best match between the target and the codebook vector. Compute the optimal gain for this match and quantize that gain (section 6.6.4).

5. Update the perceptually weighted target by subtracting the contribution from the selected codebook vector from the perceptually weighted memory (quantized gain times selected vector). Repeat 4 and 5 for the two additional stages.

6. Calculate the energy loss due to encoding of the residual. If needed, compensate for this loss by an upscaling and requantization of the gain for the first stage (section 6.7).

The following sections provide an in-depth description of the different blocks of Figure 6.4.

6.6.1 Codebook Memory

The codebook memory is based on the already encoded sub-blocks, so the available data for encoding increases for each new sub-block that has been encoded. Until enough sub-blocks have been encoded to fill the codebook memory with data, it is padded with zeros. The following figure shows an example of the order in which the sub- blocks are encoded for the 30 ms frame size if the start state is located in the last 58 samples of sub-block 2 and 3.

```
+-------------------------------------------------------+
|  5     | 1  |///|////////|    2   |    3   |    4   |
+-------------------------------------------------------+
```

Figure 6.5. The order from 1 to 5 in which the sub-blocks are encoded. The slashed area is the start state.

The first target sub-block to be encoded is number 1, and the corresponding codebook memory is shown in the following figure. As the target vector comes before the start state in time, the codebook memory and target vector are time reversed; thus, after the block has been time reversed the search algorithm can be reused. As only the start state has been encoded so far, the last samples of the codebook memory are padded with zeros.

```
+-------------------------
|zeros|\\\\\\\\|\\\\|  1 |
+-------------------------
```

Figure 6.6. The codebook memory, length lMem=85 samples, and the target vector 1, length 22 samples

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

The next step is to encode sub-block 2 by using the memory that now has increased since sub-block 1 has been encoded. The following figure shows the codebook memory for encoding of sub-block 2.

```
+-----------------------------------
| zeros | 1  |///|////////|    2   |
+-----------------------------------
```

Figure 6.7. The codebook memory, length lMem=147 samples, and the target vector 2, length 40 samples

The next step is to encode sub-block 3 by using the memory which has been increased yet again since sub-blocks 1 and 2 have been encoded, but the sub-block still has to be padded with a few zeros. The following figure shows the codebook memory for encoding of sub-block 3.

```
+-----------------------------------------
|zeros| 1  |///|////////|    2   |   3    |
+-----------------------------------------
```

Figure 6.8. The codebook memory, length lMem=147 samples, and the target vector 3, length 40 samples

The next step is to encode sub-block 4 by using the memory which now has increased yet again since sub-blocks 1, 2, and 3 have been encoded. This time, the memory does not have to be padded with zeros. The following figure shows the codebook memory for encoding of sub-block 4.

```
+-----------------------------------------
|1|///|////////|    2   |   3    |   4    |
+-----------------------------------------
```

Figure 6.9. The codebook memory, length lMem=147 samples, and the target vector 4, length 40 samples

The final target sub-block to be encoded is number 5, and the following figure shows the corresponding codebook memory. As the target vector comes before the start state in time, the codebook memory and target vector are time reversed.

```
+-------------------------------------------
|  3  |   2    |\\\\\\\\|\\\\|  1 |   5    |
+-------------------------------------------
```

Figure 6.10. The codebook memory, length lMem=147 samples, and the target vector 5, length 40 samples

For the case of 20 ms frames, the encoding procedure looks almost exactly the same. The only difference is that the size of the start state is 57 samples and that there are only three sub-blocks to be encoded. The encoding order is the same as above, starting with the 23-sample target and then encoding the two remaining 40-sample sub- blocks, first going forward in time and then going backward in time relative to the start state.

6.6.2 Perceptual Weighting of Codebook Memory and Target

To provide a perceptual weighting of the coding error, a concatenation of the codebook memory and the target to be coded is all-pole filtered with the perceptual weighting filter specified in section 6.4. The filter state of the weighting filter is set to zero.

```
in(0..(lMem-1))            = unweighted codebook memory
in(lMem..(lMem+lTarget-1)) = unweighted target signal

in -> Wk(z) -> filtered,
    where Wk(z) is taken from the sub-block of the target

weighted codebook memory = filtered(0..(lMem-1))
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
weighted target signal = filtered(lMem..(lMem+lTarget-1))
```

The codebook search is done with the weighted codebook memory and the weighted target, whereas the decoding and the codebook memory update uses the unweighted codebook memory.

6.6.3 Codebook Creation

The codebook for the search is created from the perceptually weighted codebook memory. It consists of two sections, where the first is referred to as the base codebook and the second as the expanded codebook, as it is created by linear combinations of the first. Each of these two sections also has a subsection referred to as the augmented codebook. The augmented codebook is only created and used for the coding of the 40-sample sub-blocks and not for the 23/22- sample sub-block case. The codebook size used for the different sub-blocks and different stages are summarized in the table below.

Table 6.1. Codebook sizes for the 30 ms mode

		Stage 1		Stage 2 & 3	
	22	128	(64+0)*2	128	(64+0)*2
Sub-	1:st 40	256	(108+20)*2	128	(44+20)*2
Blocks	2:nd 40	256	(108+20)*2	256	(108+20)*2
	3:rd 40	256	(108+20)*2	256	(108+20)*2
	4:th 40	256	(108+20)*2	256	(108+20)*2

Table 6.1 shows the codebook size for the different sub-blocks and stages for 30 ms frames. Inside the parentheses it shows how the number of codebook vectors is distributed, within the two sections, between the base/expanded codebook and the augmented base/expanded codebook. It should be interpreted in the following way: (base/expanded cb + augmented base/expanded cb). The total number of codebook vectors for a specific sub-block and stage is given by the following formula:

```
Tot. cb vectors = base cb + aug. base cb + exp. cb + aug. exp. cb
```

The corresponding values to Figure 6.1 for 20 ms frames are only slightly modified. The short sub-block is 23 instead of 22 samples, and the 3:rd and 4:th sub-frame are not present.

6.6.3.1 Creation of a Base Codebook

The base codebook is given by the perceptually weighted codebook memory that is mentioned in section 6.5.3. The different codebook vectors are given by sliding a window of length 23/22 or 40, given by variable lTarget, over the lMem-long perceptually weighted codebook memory. The indices are ordered so that the codebook vector containing sample (lMem-lTarget-n) to (lMem-n-1) of the codebook memory vector has index n, where n=0..lMem-lTarget. Thus the total number of base codebook vectors is lMem-lTarget+1, and the indices are ordered from sample delay lTarget (23/22 or 40) to lMem+1 (86 or 148).

6.6.3.2 Codebook Expansion

The base codebook is expanded by a factor of 2, creating an additional section in the codebook. This new section is obtained by filtering the base codebook, base_cb, with an FIR filter with filter length CB_FILTERLEN=8. The construction of the expanded codebook compensates for the delay of four samples introduced by the FIR filter.

```
cbfiltersTbl[CB_FILTERLEN]={-0.033691, 0.083740, -0.144043,
               0.713379, 0.806152, -0.184326,
               0.108887, -0.034180};

                ___
                \
   exp_cb(k)=  + > cbfiltersTbl(i)*x(k-i+4)
                /__
          i=0...(LPC_FILTERORDER-1)
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
where x(j) = base_cb(j) for j=0..lMem-1 and 0 otherwise
```

The individual codebook vectors of the new filtered codebook, exp_cb, and their indices are obtained in the same fashion as described above for the base codebook.

6.6.3.3 Codebook Augmentation

For cases where encoding entire sub-blocks, i.e., cbveclen=40, the base and expanded codebooks are augmented to increase codebook richness. The codebooks are augmented by vectors produced by interpolation of segments. The base and expanded codebook, constructed above, consists of vectors corresponding to sample delays in the range from cbveclen to lMem. The codebook augmentation attempts to augment these codebooks with vectors corresponding to sample delays from 20 to 39. However, not all of these samples are present in the base codebook and expanded codebook, respectively. Therefore, the augmentation vectors are constructed as linear combinations between samples corresponding to sample delays in the range 20 to 39. The general idea of this procedure is presented in the following figures and text. The procedure is performed for both the base codebook and the expanded codebook.

```
   - - ------------------------|
codebook memory                |
   - - ------------------------|
             |-5-|---15---|-5-|
             pi  pp       po

                 |        |                      Codebook vector
                 |---15---|-5-|-----20-----|   <- corresponding to
                     i     ii      iii            sample delay 20
```

Figure 6.11. Generation of the first augmented codebook

Figure 6.11 shows the codebook memory with pointers pi, pp, and po, where pi points to sample 25, pp to sample 20, and po to sample 5. Below the codebook memory, the augmented codebook vector corresponding to sample delay 20 is drawn. Segment i consists of fifteen samples from pointer pp and forward in time. Segment ii consists of five interpolated samples from pi and forward and from po and forward. The samples are linearly interpolated with weights [0.0, 0.2, 0.4, 0.6, 0.8] for pi and weights [1.0, 0.8, 0.6, 0.4, 0.2] for po. Segment iii consists of twenty samples from pp and forward. The augmented codebook vector corresponding to sample delay 21 is produced by moving pointers pp and pi one sample backward in time. This gives us the following figure.

```
   - - ------------------------|
codebook memory                |
   - - ------------------------|
             |-5-|---16---|-5-|
             pi  pp       po

                 |        |                      Codebook vector
                 |---16---|-5-|-----19-----|   <- corresponding to
                     i     ii      iii            sample delay 21
```

Figure 6.12. Generation of the second augmented codebook

Figure 6.12 shows the codebook memory with pointers pi, pp and po where pi points to sample 26, pp to sample 21, and po to sample 5. Below the codebook memory, the augmented codebook vector corresponding to sample delay 21 is drawn. Segment i now consists of sixteen samples from pp and forward. Segment ii consists of five interpolated samples from pi and forward and from po and forward, and the interpolation weights are the same throughout the procedure. Segment iii consists of nineteen samples from pp and forward. The same procedure of moving the two pointers is continued until the last augmented vector corresponding to sample delay 39 has been created. This gives a total of twenty new codebook vectors to each of the two sections. Thus the total number of codebook vectors for each of the two sections, when including the augmented codebook, becomes lMem- SUBL+1+SUBL/2. This is provided that augmentation is evoked, i.e., that lTarget=SUBL.

Documents Document Sharing Hub
Full document details: SCTE 24-22 2018

6.6.4 Codebook Search

The codebook search uses the codebooks described in the sections above to find the best match of the perceptually weighted target, see section 6.6.2. The search method is a multi-stage gain-shape matching performed as follows. At each stage the best shape vector is identified, then the gain is calculated and quantized, and finally the target is updated in preparation for the next codebook search stage. The number of stages is CB_NSTAGES=3.

If the target is the 23/22-sample vector the codebooks are indexed so that the base codebook is followed by the expanded codebook. If the target is 40 samples the order is as follows: base codebook, augmented base codebook, expanded codebook, and augmented expanded codebook. The size of each codebook section and its corresponding augmented section is given by Table 6.1 in section 6.6.3.

For example, when the second 40-sample sub-block is coded, indices 0 - 107 correspond to the base codebook, 108 - 127 correspond to the augmented base codebook, 128 - 235 correspond to the expanded codebook, and indices 236 - 255 correspond to the augmented expanded codebook. The indices are divided in the same fashion for all stages in the example. Only in the case of coding the first 40-sample sub- block is there a difference between stages (see Table 6.1).

6.6.4.1 Codebook Search at Each Stage

The codebooks are searched to find the best match to the target at each stage. When the best match is found, the target is updated and the next-stage search is started. The three chosen codebook vectors and their corresponding gains constitute the encoded sub-block. The best match is decided by the following three criteria:

```
1. Compute the measure

   (target*cbvec)^2 / ||cbvec||^2

for all codebook vectors, cbvec, and choose the codebook vector
maximizing the measure.  The expression (target*cbvec) is the dot
product between the target vector to be coded and the codebook vector
for which we compute the measure.  The norm, ||x||, is defined as the
square root of (x*x).

2. The absolute value of the gain, corresponding to the chosen
   codebook vector, cbvec, must be smaller than a fixed limit,
   CB_MAXGAIN=1.3:

         |gain| < CB_MAXGAIN

   where the gain is computed in the following way:

         gain = (target*cbvec) / ||cbvec||^2

3. For the first stage, the dot product of the chosen codebook vector
   and target must be positive:

   target*cbvec > 0
```

In practice the above criteria are used in a sequential search through all codebook vectors. The best match is found by registering a new max measure and index whenever the previously registered max measure is surpassed and all other criteria are fulfilled. If none of the codebook vectors fulfill (2) and (3), the first codebook vector is selected.

6.6.4.2 Gain Quantization at Each Stage

The gain follows as a result of the computation

```
   gain = (target*cbvec) / ||cbvec||^2

for the optimal codebook vector found by the procedure in section 6.6.4.1.
```

The three stages quantize the gain, using 5, 4, and 3 bits, respectively. In the first stage, the gain is limited to positive values. This gain is quantized by finding the nearest value in the quantization table gain_sq5Tbl.

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
gain_sq5Tbl[32]={0.037476, 0.075012, 0.112488, 0.150024, 0.187500,
                0.224976, 0.262512, 0.299988, 0.337524, 0.375000,
                0.412476, 0.450012, 0.487488, 0.525024, 0.562500,
                0.599976, 0.637512, 0.674988, 0.712524, 0.750000,
                0.787476, 0.825012, 0.862488, 0.900024, 0.937500,
                0.974976, 1.012512, 1.049988, 1.087524, 1.125000,
                1.162476, 1.200012}
```

The gains of the subsequent two stages can be either positive or negative. The gains are quantized by using a quantization table times a scale factor. The second stage uses the table gain_sq4Tbl, and the third stage uses gain_sq3Tbl. The scale factor equates 0.1 or the absolute value of the quantized gain representation value obtained in the previous stage, whichever is larger. Again, the resulting gain index is the index to the nearest value of the quantization table times the scale factor.

```
     gainQ = scaleFact * gain_sqXTbl[index]

gain_sq4Tbl[16]={-1.049988, -0.900024, -0.750000, -0.599976,
                -0.450012, -0.299988, -0.150024, 0.000000, 0.150024,
                0.299988, 0.450012, 0.599976, 0.750000, 0.900024,
                1.049988, 1.200012}

gain_sq3Tbl[8]={-1.000000, -0.659973, -0.330017,0.000000,
                0.250000, 0.500000, 0.750000, 1.00000}
```

6.6.4.3 Preparation of Target for Next Stage

Before performing the search for the next stage, the perceptually weighted target vector is updated by subtracting from it the selected codebook vector (from the perceptually weighted codebook) times the corresponding quantized gain.

```
     target[i] = target[i] - gainQ * selected_vec[i];
```

A reference implementation of the codebook encoding is found in Appendix A.34.

6.7 Gain Correction Encoding

The start state is quantized in a relatively model independent manner using 3 bits per sample. In contrast, the remaining parts of the block are encoded by using an adaptive codebook. This codebook will produce high matching accuracy whenever there is a high correlation between the target and the best codebook vector. For unvoiced speech segments and background noises, this is not necessarily so, which, due to the nature of the squared error criterion, results in a coded signal with less power than the target signal. As the coded start state has good power matching to the target, the result is a power fluctuation within the encoded frame. Perceptually, the main problem with this is that the time envelope of the signal energy becomes unsteady. To overcome this problem, the gains for the codebooks are re-scaled after the codebook encoding by searching for a new gain factor for the first stage codebook that provides better power matching.

First, the energy for the target signal, tene, is computed along with the energy for the coded signal, cene, given by the addition of the three gain scaled codebook vectors. Because the gains of the second and third stage scale with the gain of the first stage, when the first stage gain is changed from gain[0] to gain_sq5Tbl[i] the energy of the coded signal changes from cene to

```
     cene*(gain_sq5Tbl[i]*gain_sq5Tbl[i])/(gain[0]*gain[0])
```

where gain[0] is the gain for the first stage found in the original codebook search. A refined search is performed by testing the gain indices i=0 to 31, and as long as the new codebook energy as given above is less than tene, the gain index for stage 1 is increased. A restriction is applied so that the new gain value for stage 1 cannot be more than two times higher than the original value found in the codebook search. Note that by using this method we do not change the shape of the encoded vector, only the gain or amplitude.

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

6.8 Bitstream Definition

The total number of bits used to describe one frame of 20 ms speech is 304, which fits in 38 bytes and results in a bit rate of 15.20 kbit/s. For the case of a frame length of 30 ms speech, the total number of bits used is 400, which fits in 50 bytes and results in a bit rate of 13.33 kbit/s. In the bitstream definition, the bits are distributed into three classes according to their bit error or loss sensitivity. The most sensitive bits (class 1) are placed first in the bitstream for each frame. The less sensitive bits (class 2) are placed after the class 1 bits. The least sensitive bits (class 3) are placed at the end of the bitstream for each frame.

In the 20/30 ms frame length cases for each class, the following hold true: The class 1 bits occupy a total of 6/8 bytes (48/64 bits), the class 2 bits occupy 8/12 bytes (64/96 bits), and the class 3 bits occupy 24/30 bytes (191/239 bits). This distribution of the bits enables the use of uneven level protection (ULP) as is exploited in the payload format definition for iLBC contained in RFC 3952. The detailed bit allocation is shown in the table below. When a quantization index is distributed between more classes, the more significant bits belong to the lowest class.

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

Table 6.2. The bitstream definition for iLBC for both the 20 ms frame size mode and the 30 ms frame size mode

Bitstream structure:

Parameter			Bits Class <1,2,3> 20 ms frame	Bits Class <1,2,3> 30 ms frame
LSF	LSF 1	Split 1	6 <6,0,0>	6 <6,0,0>
		Split 2	7 <7,0,0>	7 <7,0,0>
		Split 3	7 <7,0,0>	7 <7,0,0>
	LSF 2	Split 1	NA (Not Appl.)	6 <6,0,0>
		Split 2	NA	7 <7,0,0>
		Split 3	NA	7 <7,0,0>
	Sum		20 <20,0,0>	40 <40,0,0>
Block Class			2 <2,0,0>	3 <3,0,0>
Position 22 sample segment			1 <1,0,0>	1 <1,0,0>
Scale Factor State Coder			6 <6,0,0>	6 <6,0,0>
Quantized Residual State Samples	Sample 0		3 <0,1,2>	3 <0,1,2>
	Sample 1		3 <0,1,2>	3 <0,1,2>
	:		: :	: :
	:		: :	: :
	:		: :	: :
	Sample 56		3 <0,1,2>	3 <0,1,2>
	Sample 57		NA	3 <0,1,2>
	Sum		171 <0,57,114>	174 <0,58,116>
CB for 22/23 sample block		Stage 1	7 <6,0,1>	7 <4,2,1>
		Stage 2	7 <0,0,7>	7 <0,0,7>
		Stage 3	7 <0,0,7>	7 <0,0,7>
	Sum		21 <6,0,15>	21 <4,2,15>
Gain for 22/23 sample block		Stage 1	5 <2,0,3>	5 <1,1,3>
		Stage 2	4 <1,1,2>	4 <1,1,2>
		Stage 3	3 <0,0,3>	3 <0,0,3>
	Sum		12 <3,1,8>	12 <2,2,8>
Indices for CB sub-blocks	sub-block 1	Stage 1	8 <7,0,1>	8 <6,1,1>
		Stage 2	7 <0,0,7>	7 <0,0,7>
		Stage 3	7 <0,0,7>	7 <0,0,7>
	sub-block 2	Stage 1	8 <0,0,8>	8 <0,7,1>
		Stage 2	8 <0,0,8>	8 <0,0,8>
		Stage 3	8 <0,0,8>	8 <0,0,8>
	sub-block 3	Stage 1	NA	8 <0,7,1>
		Stage 2	NA	8 <0,0,8>
		Stage 3	NA	8 <0,0,8>
	sub-block 4	Stage 1	NA	8 <0,7,1>
		Stage 2	NA	8 <0,0,8>
		Stage 3	NA	8 <0,0,8>

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
                Sum                |  46 <7,0,39>  |  94 <6,22,66> |
-----------------------------------+---------------+---------------+
                          Stage 1  |   5 <1,2,2>   |   5 <1,2,2>   |
            sub-block 1   Stage 2  |   4 <1,1,2>   |   4 <1,2,1>   |
                          Stage 3  |   3 <0,0,3>   |   3 <0,0,3>   |
                ------------------+---------------+---------------+
                          Stage 1  |   5 <1,1,3>   |   5 <0,2,3>   |
            sub-block 2   Stage 2  |   4 <0,2,2>   |   4 <0,2,2>   |
                          Stage 3  |   3 <0,0,3>   |   3 <0,0,3>   |
Gains for       ------------------+---------------+---------------+
sub-blocks                Stage 1  |      NA       |   5 <0,1,4>   |
            sub-block 3   Stage 2  |      NA       |   4 <0,1,3>   |
                          Stage 3  |      NA       |   3 <0,0,3>   |
                ------------------+---------------+---------------+
                          Stage 1  |      NA       |   5 <0,1,4>   |
            sub-block 4   Stage 2  |      NA       |   4 <0,1,3>   |
                          Stage 3  |      NA       |   3 <0,0,3>   |
                ------------------+---------------+---------------+
                Sum                |  24 <3,6,15>  |  48 <2,12,34> |
-----------------------------------+---------------+---------------+
Empty frame indicator              |   1 <0,0,1>   |   1 <0,0,1>   |
-----------------------------------------------------------------------
SUM                                  304 <48,64,192> 400 <64,96,240>
```

When packetized into the payload, the bits MUST be sorted as follows:

All the class 1 bits in the order (from top to bottom) as specified in the table, all the class 2 bits (from top to bottom), and all the class 3 bits in the same sequential order. The last bit, the empty frame indicator, SHOULD be set to zero by the encoder. If this bit is set to 1 the decoder SHOULD treat the data as a lost frame. For example, this bit can be set to 1 to indicate lost frame for file storage format, as in RFC 3952.

7 DECODER PRINCIPLES

This section describes the principles of each component of the decoder algorithm.

```
           +-------------+    +--------+    +---------------+
payload -> | 1. Get para | -> | 2. LPC | -> | 3. Sc Dequant | ->
           +-------------+    +--------+    +---------------+

           +-------------+    +------------------+
        -> | 4. Mem setup| -> | 5. Construct res |------->
        |  +-------------+    +------------------   |
        ---------<-----------<-----------<------------
                  Sub-frame 0...2/4 (20 ms/30 ms)

           +----------------+    +----------+
        -> | 6. Enhance res | -> | 7. Synth | ------------>
           +----------------+    +----------+

           +-----------------+
        -> | 8. Post Process | ----------------> decoded speech
           +-----------------+
```

Figure 7.1. Flow chart of the iLBC decoder

If a frame was lost, steps 1 to 5 SHOULD be replaced by a PLC algorithm.

1. Extract the parameters from the bitstream.

2. Decode the LPC and interpolate (section 7.1).

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

3. Construct the 57/58-sample start state (section 7.2).

4. Set up the memory by using data from the decoded residual. This memory is used for codebook construction. For blocks preceding the start state, both the decoded residual and the target are time reversed. Sub-frames are decoded in the same order as they were encoded.

5. Construct the residuals of this sub-frame (gain[0]*cbvec[0] + gain[1]*cbvec[1] + gain[2]*cbvec[2]). Repeat 4 and 5 until the residual of all sub-blocks has been constructed.

6. Enhance the residual with the post filter (section 7.6).

7. Synthesis of the residual (section 7.7).

8. Post process with HP filter, if desired (section 7.8).

7.1 LPC Filter Reconstruction

The decoding of the LPC filter parameters is very straightforward. For a set of three/six indices, the corresponding LSF vector(s) are found by simple table lookup. For each of the LSF vectors, the three split vectors are concatenated to obtain qlsf1 and qlsf2, respectively (in the 20 ms mode only one LSF vector, qlsf, is constructed). The next step is the stability check described in section 6.2.5 followed by the interpolation scheme described in section 6.2.6 (6.2.7 for 20 ms frames). The only difference is that only the quantized LSFs are known at the decoder, and hence the unquantized LSFs are not processed.

A reference implementation of the LPC filter reconstruction is given in Appendix A.36.

7.2 Start State Reconstruction

The scalar encoded STATE_SHORT_LEN=58 (STATE_SHORT_LEN=57 in the 20 ms mode) state samples are reconstructed by 1) forming a set of samples (by table lookup) from the index stream idxVec[n], 2) multiplying the set with 1/scal=(10^qmax)/4.5, 3) time reversing the 57/58 samples, 4) filtering the time reversed block with the dispersion (all-pass) filter used in the encoder (as described in section 6.5.2); this compensates for the phase distortion of the earlier filter operation, and 5 reversing the 57/58 samples from the previous step.

```
in(0..(STATE_SHORT_LEN-1)) = time reversed samples from table
                             look-up,
                             idxVecDec((STATE_SHORT_LEN-1)..0)

in(STATE_SHORT_LEN..(2*STATE_SHORT_LEN-1)) = 0

Pk(z) = A~rk(z)/A~k(z), where
                                ___
                                \
A~rk(z)= z^(-LPC_FILTERORDER) + > a~ki*z^(i-(LPC_FILTERORDER-1))
                                /__
                           i=0...(LPC_FILTERORDER-1)

and A~k(z) is taken from the block where the start state begins

in -> Pk(z) -> filtered

out(k) = filtered(STATE_SHORT_LEN-1-k) +
                        filtered(2*STATE_SHORT_LEN-1-k),
                                      k=0..(STATE_SHORT_LEN-1)
```

The remaining 23/22 samples in the state are reconstructed by the same adaptive codebook technique described in section 7.3. The location bit determines whether these are the first or the last 23/22 samples of the 80-sample state vector. If the remaining 23/22 samples are the first samples, then the scalar encoded STATE_SHORT_LEN state samples are time-reversed before initialization of the adaptive codebook memory vector.

A reference implementation of the start state reconstruction is given in Appendix A.44.

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

7.3 Excitation Decoding Loop

The decoding of the LPC excitation vector proceeds in the same order in which the residual was encoded at the encoder. That is, after the decoding of the entire 80-sample state vector, the forward sub-blocks (corresponding to samples occurring after the state vector samples) are decoded, and then the backward sub-blocks (corresponding to samples occurring before the state vector) are decoded, resulting in a fully decoded block of excitation signal samples.

In particular, each sub-block is decoded by using the multistage adaptive codebook decoding module described in section 7.4. This module relies upon an adaptive codebook memory constructed before each run of the adaptive codebook decoding. The construction of the adaptive codebook memory in the decoder is identical to the method outlined in section 6.6.3, except that it is done on the codebook memory without perceptual weighting.

For the initial forward sub-block, the last STATE_LEN=80 samples of the length CB_LMEM=147 adaptive codebook memory are filled with the samples of the state vector. For subsequent forward sub-blocks, the first SUBL=40 samples of the adaptive codebook memory are discarded, the remaining samples are shifted by SUBL samples toward the beginning of the vector, and the newly decoded SUBL=40 samples are placed at the end of the adaptive codebook memory. For backward sub-blocks, the construction is similar, except that every vector of samples involved is first time reversed.

A reference implementation of the excitation decoding loop is found in Appendix A.5.

7.4 Multistage Adaptive Codebook Decoding

The Multistage Adaptive Codebook Decoding module is used at both the sender (encoder) and the receiver (decoder) ends to produce a synthetic signal in the residual domain that is eventually used to produce synthetic speech. The module takes the index values used to construct vectors that are scaled and summed together to produce a synthetic signal that is the output of the module.

7.4.1 Construction of the Decoded Excitation Signal

The unpacked index values provided at the input to the module are references to extended codebooks, which are constructed as described in section 6.6.3, except that they are based on the codebook memory without the perceptual weighting. The unpacked three indices are used to look up three codebook vectors. The unpacked three gain indices are used to decode the corresponding 3 gains. In this decoding, the successive rescaling, as described in section 6.6.4.2, is applied.

A reference implementation of the adaptive codebook decoding is listed in Appendix A.32.

7.5 Packet Loss Concealment

If packet loss occurs, the decoder receives a signal saying that information regarding a block is lost. For such blocks it is RECOMMENDED to use a Packet Loss Concealment (PLC) unit to create a decoded signal that masks the effect of that packet loss. In the following we will describe an example of a PLC unit that can be used with the iLBC codec. As the PLC unit is used only at the decoder, the PLC unit does not affect interoperability between implementations. Other PLC implementations MAY therefore be used.

The PLC described operates on the LPC filters and the excitation signals and is based on the following principles:

7.5.1 Block Received Correctly and Previous Block Also Received

If the block is received correctly, the PLC only records state information of the current block that can be used in case the next block is lost. The LPC filter coefficients for each sub-block and the entire decoded excitation signal are all saved in the decoder state structure. All of this information will be needed if the following block is lost.

7.5.2 Block Not Received

If the block is not received, the block substitution is based on a pitch-synchronous repetition of the excitation signal, which is filtered by the last LPC filter of the previous block. The previous block's information is stored in the decoder state structure.

A correlation analysis is performed on the previous block's excitation signal in order to detect the amount of pitch periodicity and a pitch value. The correlation measure is also used to decide on the voicing level (the degree to which

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

the previous block's excitation was a voiced or roughly periodic signal). The excitation in the previous block is used to create an excitation for the block to be substituted, such that the pitch of the previous block is maintained. Therefore, the new excitation is constructed in a pitch-synchronous manner. In order to avoid a buzzy-sounding substituted block, a random excitation is mixed with the new pitch periodic excitation, and the relative use of the two components is computed from the correlation measure (voicing level).

For the block to be substituted, the newly constructed excitation signal is then passed through the LPC filter to produce the speech that will be substituted for the lost block.

For several consecutive lost blocks, the packet loss concealment continues in a similar manner. The correlation measure of the last block received is still used along with the same pitch value. The LPC filters of the last block received are also used again. The energy of the substituted excitation for consecutive lost blocks is decreased, leading to a dampened excitation, and therefore to dampened speech.

7.5.3 Block Received Correctly When Previous Block Not Received

For the case in which a block is received correctly when the previous block was not, the correctly received block's directly decoded speech (based solely on the received block) is not used as the actual output. The reason for this is that the directly decoded speech does not necessarily smoothly merge into the synthetic speech generated for the previous lost block. If the two signals are not smoothly merged, an audible discontinuity is accidentally produced. Therefore, a correlation analysis between the two blocks of excitation signal (the excitation of the previous concealed block and that of the current received block) is performed to find the best phase match. Then a simple overlap-add procedure is performed to merge the previous excitation smoothly into the current block's excitation.

The exact implementation of the packet loss concealment does not influence interoperability of the codec.

A reference implementation of the packet loss concealment is suggested in Appendix A.14. Exact compliance with this suggested algorithm is not needed for a reference implementation to be fully compatible with the overall codec specification.

7.6 Enhancement

The decoder contains an enhancement unit that operates on the reconstructed excitation signal. The enhancement unit increases the perceptual quality of the reconstructed signal by reducing the speech-correlated noise in the voiced speech segments. Compared to traditional postfilters, the enhancer has an advantage in that it can only modify the excitation signal slightly. This means that there is no risk of over enhancement. The enhancer works very similarly for both the 20 ms frame size mode and the 30 ms frame size mode.

For the mode with 20 ms frame size, the enhancer uses a memory of six 80-sample excitation blocks prior in time plus the two new 80-sample excitation blocks. For each block of 160 new unenhanced excitation samples, 160 enhanced excitation samples are produced. The enhanced excitation is 40-sample delayed compared to the unenhanced excitation, as the enhancer algorithm uses lookahead.

For the mode with 30 ms frame size, the enhancer uses a memory of five 80-sample excitation blocks prior in time plus the three new 80-sample excitation blocks. For each block of 240 new unenhanced excitation samples, 240 enhanced excitation samples are produced. The enhanced excitation is 80-sample delayed compared to the unenhanced excitation, as the enhancer algorithm uses lookahead.

Outline of Enhancer

The speech enhancement unit operates on sub-blocks of 80 samples, which means that there are two/three 80 sample sub-blocks per frame. Each of these two/three sub-blocks is enhanced separately, but in an analogous manner.

```
unenhanced residual
        |
        |   +---------------+    +--------------+
        +-> | 1. Pitch Est  | -> | 2. Find PSSQ | -------->
            +---------------+  | +--------------+
                               +-----<-------<------<--+
            +------------+         enh block 0..1/2    |
         -> | 3. Smooth  |                             |
            +------------+                             |
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
          \                                        |
           /\                                      |
          /  \   Already                           |
         / 4. \----------->----------->-----------+ |
         \Crit/ Fulfilled                         | |
          \? /                                    v |
           \/                                     | |
            \  +-----------------+    +---------+ | |
      Not +->| 5. Use Constr.  | -> | 6. Mix  | ----->
   Fulfilled +-----------------+    +---------+

    ---------------> enhanced residual
```

Figure 7.2. Flow chart of the enhancer

1. Pitch estimation of each of the two/three new 80-sample blocks.

2. Find the pitch-period-synchronous sequence n (for block k) by a search around the estimated pitch value. Do this for n=1,2,3, -1,-2,-3.

3. Calculate the smoothed residual generated by the six pitch- period-synchronous sequences from prior step.

4. Check if the smoothed residual satisfies the criterion (section 7.6.4).

5. Use constraint to calculate mixing factor (section 7.6.5).

6. Mix smoothed signal with unenhanced residual (pssq(n) n=0).

The main idea of the enhancer is to find three 80 sample blocks before and three 80-sample blocks after the analyzed unenhanced sub- block and to use these to improve the quality of the excitation in that sub-block. The six blocks are chosen so that they have the highest possible correlation with the unenhanced sub-block that is being enhanced. In other words, the six blocks are pitch-period- synchronous sequences to the unenhanced sub-block.

A linear combination of the six pitch-period-synchronous sequences is calculated that approximates the sub-block. If the squared error between the approximation and the unenhanced sub-block is small enough, the enhanced residual is set equal to this approximation. For the cases when the squared error criterion is not fulfilled, a linear combination of the approximation and the unenhanced residual forms the enhanced residual.

7.6.1 Estimating the Pitch

Pitch estimates are needed to determine the locations of the pitch- period-synchronous sequences in a complexity-efficient way. For each of the new two/three sub-blocks, a pitch estimate is calculated by finding the maximum correlation in the range from lag 20 to lag 120. These pitch estimates are used to narrow down the search for the best possible pitch-period-synchronous sequences.

7.6.2 Determination of the Pitch-Synchronous Sequences

Upon receiving the pitch estimates from the prior step, the enhancer analyzes and enhances one 80-sample sub-block at a time. The pitch- period-synchronous-sequences pssq(n) can be viewed as vectors of length 80 samples each shifted n*lag samples from the current sub- block. The six pitch-period-synchronous-sequences, pssq(-3) to pssq(-1) and pssq(1) to pssq(3), are found one at a time by the steps below:

1) Calculate the estimate of the position of the pssq(n). For pssq(n) in front of pssq(0) (n > 0), the location of the pssq(n) is estimated by moving one pitch estimate forward in time from the exact location of pssq(n-1). Similarly, pssq(n) behind pssq(0) (n < 0) is estimated by moving one pitch estimate backward in time from the exact location of pssq(n+1). If the estimated pssq(n) vector location is totally within the enhancer memory (Figure 7.3), steps 2, 3, and 4 are performed, otherwise the pssq(n) is set to zeros.

2) Compute the correlation between the unenhanced excitation and vectors around the estimated location interval of pssq(n). The correlation is calculated in the interval estimated location +/- 2 samples. This results in five correlation values.

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

3) The five correlation values are upsampled by a factor of 4, by using four simple upsampling filters (MA filters with coefficients upsFilter1.. upsFilter4). Within these the maximum value is found, which specifies the best pitch-period with a resolution of a quarter of a sample.

```
upsFilter1[7]={0.000000 0.000000 0.000000 1.000000
       0.000000 0.000000 0.000000}
upsFilter2[7]={0.015625 -0.076904 0.288330 0.862061
      -0.106445 0.018799 -0.015625}
upsFilter3[7]={0.023682 -0.124268 0.601563 0.601563
      -0.124268 0.023682 -0.023682}
upsFilter4[7]={0.018799 -0.106445 0.862061 0.288330
      -0.076904 0.015625 -0.018799}
```

4) Generate the pssq(n) vector by upsampling of the excitation memory and extracting the sequence that corresponds to the lag delay that was calculated in prior step.

With the steps above, all the pssq(n) can be found in an iterative manner, first moving backward in time from pssq(0) and then forward in time from pssq(0).

```
0              159             319             479             639
+-------------------------------------------------------------+
|  -5   |  -4   |  -3   |  -2   |  -1   |   0   |   1   |   2   |
+-------------------------------------------------------------+
                                          |pssq 0 |
                                     |pssq -1| |pssq 1 |
                                |pssq -2|       |pssq 2 |
                           |pssq -3|             |pssq 3 |
```

Figure 7.3. Enhancement for 20 ms frame size

Figure 7.3 depicts pitch-period-synchronous sequences in the enhancement of the first 80 sample block in the 20 ms frame size mode. The unenhanced signal input is stored in the last two sub- blocks (1 - 2), and the six other sub-blocks contain unenhanced residual prior-in-time. We perform the enhancement algorithm on two blocks of 80 samples, where the first of the two blocks consists of the last 40 samples of sub-block 0 and the first 40 samples of sub- block 1. The second 80-sample block consists of the last 40 samples of sub-block 1 and the first 40 samples of sub-block 2.

```
0              159             319             479             639
+-------------------------------------------------------------+
|  -4   |  -3   |  -2   |  -1   |   0   |   1   |   2   |   3   |
+-------------------------------------------------------------+
                                |pssq 0 |
                           |pssq -1| |pssq 1 |
                      |pssq -2|       |pssq 2 |
                 |pssq -3|             |pssq 3 |
```

Figure 7.4. Enhancement for 30 ms frame size

Figure 7.4 depicts pitch-period-synchronous sequences in the enhancement of the first 80-sample block in the 30 ms frame size mode. The unenhanced signal input is stored in the last three sub- blocks (1 - 3). The five other sub-blocks contain unenhanced residual prior-in-time. The enhancement algorithm is performed on the three 80 sample sub-blocks 0, 1, and 2.

7.6.3 Calculation of the Smoothed Excitation

A linear combination of the six pssq(n) (n!=0) form a smoothed approximation, z, of pssq(0). Most of the weight is put on the sequences that are close to pssq(0), as these are likely to be most similar to pssq(0). The smoothed vector is also rescaled so that the energy of z is the same as the energy of pssq(0).

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
   ___
   \
y = > pssq(i) * pssq_weight(i)
   /__
i=-3,-2,-1,1,2,3

pssq_weight(i) = 0.5*(1-cos(2*pi*(i+4)/(2*3+2)))

z = C * y, where C = ||pssq(0)||/||y||
```

7.6.4 Enhancer Criterion

The criterion of the enhancer is that the enhanced excitation is not allowed to differ much from the unenhanced excitation. This criterion is checked for each 80-sample sub-block.

```
e < (b * ||pssq(0)||^2), where b=0.05 and   (Constraint 1)

e = (pssq(0)-z)*(pssq(0)-z), and "*" means the dot product
```

7.6.5 Enhancing the excitation

From the criterion in the previous section, it is clear that the excitation is not allowed to change much. The purpose of this constraint is to prevent the creation of an enhanced signal significantly different from the original signal. This also means that the constraint limits the numerical size of the errors that the enhancement procedure can make. That is especially important in unvoiced segments and background noise segments for which increased periodicity could lead to lower perceived quality.

When the constraint in the prior section is not met, the enhanced residual is instead calculated through a constrained optimization by using the Lagrange multiplier technique. The new constraint is that

```
e = (b * ||pssq(0)||^2)                    (Constraint 2)
```

We distinguish two solution regions for the optimization: 1) the region where the first constraint is fulfilled and 2) the region where the first constraint is not fulfilled and the second constraint must be used.

In the first case, where the second constraint is not needed, the optimized re-estimated vector is simply z, the energy-scaled version of y.

In the second case, where the second constraint is activated and becomes an equality constraint, we have

```
z= A*y + B*pssq(0)

where

A = sqrt((b-b^2/4)*(w00*w00)/ (w11*w00 + w10*w10)) and

w11 = pssq(0)*pssq(0)
w00 = y*y
w10 = y*pssq(0)    (* symbolizes the dot product)

and

B = 1 - b/2 - A * w10/w00
```

Appendix A.16 contains a listing of a reference implementation for the enhancement method.

Documents Document Sharing Hub
Full document details: SCTE 24-22 2018

7.7 Synthesis Filtering

Upon decoding or PLC of the LPC excitation block, the decoded speech block is obtained by running the decoded LPC synthesis filter, 1/A~k(z), over the block. The synthesis filters have to be shifted to compensate for the delay in the enhancer. For 20 ms frame size mode, they SHOULD be shifted one 40-sample sub-block, and for 30 ms frame size mode, they SHOULD be shifted two 40-sample sub-blocks. The LPC coefficients SHOULD be changed at the first sample of every sub-block while keeping the filter state. For PLC blocks, one solution is to apply the last LPC coefficients of the last decoded speech block for all sub-blocks.

The reference implementation for the synthesis filtering can be found in Appendix A.48.

7.8 Post Filtering

If desired, the decoded block can be filtered by a high-pass filter. This removes the low frequencies of the decoded signal. A reference implementation of this, with cutoff at 65 Hz, is shown in Appendix A.30.

8 SECURITY CONSIDERATIONS

This algorithm for the coding of speech signals is not subject to any known security consideration; however, its RTP payload format (RFC 3952) is subject to several considerations, which are addressed there. Confidentiality of the media streams is achieved by encryption; therefore external mechanisms, such as SRTP (RFC 3711), MAY be used for that purpose.

Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

Appendix A Reference Implementation

This appendix contains the complete c-code for a reference implementation of encoder and decoder for the specified codec.

```
The c-code consists of the following files with highest-level functions:

         iLBC_test.c: main function for evaluation purpose
         iLBC_encode.h: encoder header
         iLBC_encode.c: encoder function
         iLBC_decode.h: decoder header
         iLBC_decode.c: decoder function

The following files contain global defines and constants:

         iLBC_define.h: global defines
         constants.h: global constants header
         constants.c: global constants memory allocations

The following files contain subroutines:

         anaFilter.h: lpc analysis filter header
         anaFilter.c: lpc analysis filter function
         createCB.h: codebook construction header
         createCB.c: codebook construction function
         doCPLC.h: packet loss concealment header
         doCPLC.c: packet loss concealment function
         enhancer.h: signal enhancement header
         enhancer.c: signal enhancement function
         filter.h: general filter header
         filter.c: general filter functions
         FrameClassify.h: start state classification header
         FrameClassify.c: start state classification function
         gainquant.h: gain quantization header
         gainquant.c: gain quantization function
         getCBvec.h: codebook vector construction header
         getCBvec.c: codebook vector construction function
         helpfun.h: general purpose header
         helpfun.c: general purpose functions
         hpInput.h: input high pass filter header
         hpInput.c: input high pass filter function
         hpOutput.h: output high pass filter header
         hpOutput.c: output high pass filter function
         iCBConstruct.h: excitation decoding header
         iCBConstruct.c: excitation decoding function
         iCBSearch.h: excitation encoding header
         iCBSearch.c: excitation encoding function
         LPCdecode.h: lpc decoding header
         LPCdecode.c: lpc decoding function
         LPCencode.h: lpc encoding header
         LPCencode.c: lpc encoding function
         lsf.h: line spectral frequencies header
         lsf.c: line spectral frequencies functions
         packing.h: bitstream packetization header
         packing.c: bitstream packetization functions
         StateConstructW.h: state decoding header
         StateConstructW.c: state decoding functions
         StateSearchW.h: state encoding header
         StateSearchW.c: state encoding function
         syntFilter.h: lpc synthesis filter header
         syntFilter.c: lpc synthesis filter function
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

The implementation is portable and should work on many different platforms. However, it is not difficult to optimize the implementation on particular platforms, an exercise left to the reader.

A.1 iLBC_test.c

```
/******************************************************************

    iLBC Speech Coder ANSI-C Source Code

    iLBC_test.c

******************************************************************/

#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include "iLBC_define.h"
#include "iLBC_encode.h"
#include "iLBC_decode.h"

/* Runtime statistics */
#include <time.h>

#define ILBCNOOFWORDS_MAX   (NO_OF_BYTES_30MS/2)

/*----------------------------------------------------------------*
 *  Encoder interface function


 *----------------------------------------------------------------*/

short encode(   /* (o) Number of bytes encoded */
    iLBC_Enc_Inst_t *iLBCenc_inst,
                                /* (i/o) Encoder instance */
    short *encoded_data,    /* (o) The encoded bytes */
    short *data                 /* (i) The signal block to encode*/
){
    float block[BLOCKL_MAX];
    int k;

    /* convert signal to float */

    for (k=0; k<iLBCenc_inst->blockl; k++)
        block[k] = (float)data[k];

    /* do the actual encoding */

    iLBC_encode((unsigned char *)encoded_data, block, iLBCenc_inst);


    return (iLBCenc_inst->no_of_bytes);
}

/*----------------------------------------------------------------*
 *  Decoder interface function
 *----------------------------------------------------------------*/

short decode(       /* (o) Number of decoded samples */
    iLBC_Dec_Inst_t *iLBCdec_inst,  /* (i/o) Decoder instance */
    short *decoded_data,        /* (o) Decoded signal block*/
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
    short *encoded_data,        /* (i) Encoded bytes */
    short mode                       /* (i) 0=PL, 1=Normal */
){
    int k;
    float decblock[BLOCKL_MAX], dtmp;

    /* check if mode is valid */

    if (mode<0 || mode>1) {
        printf("\nERROR - Wrong mode - 0, 1 allowed\n"); exit(3);}

    /* do actual decoding of block */

    iLBC_decode(decblock, (unsigned char *)encoded_data,
        iLBCdec_inst, mode);

    /* convert to short */

    for (k=0; k<iLBCdec_inst->blockl; k++){
        dtmp=decblock[k];

        if (dtmp<MIN_SAMPLE)
            dtmp=MIN_SAMPLE;
        else if (dtmp>MAX_SAMPLE)
            dtmp=MAX_SAMPLE;
        decoded_data[k] = (short) dtmp;
    }

    return (iLBCdec_inst->blockl);
}

/*----------------------------------------------------------------*
 *  Main program to test iLBC encoding and decoding
 *
 *  Usage:
 *    exefile_name.exe <infile> <bytefile> <outfile> <channel>
 *
 *    <infile>   : Input file, speech for encoder (16-bit pcm file)
 *    <bytefile> : Bit stream output from the encoder
 *    <outfile>  : Output file, decoded speech (16-bit pcm file)
 *    <channel>  : Bit error file, optional (16-bit)
 *                     1 - Packet received correctly
 *                     0 - Packet Lost
 *
 *----------------------------------------------------------------*/

int main(int argc, char* argv[])
{

    /* Runtime statistics */

    float starttime;
    float runtime;
    float outtime;

    FILE *ifileid,*efileid,*ofileid, *cfileid;
    short data[BLOCKL_MAX];
    short encoded_data[ILBCNOOFWORDS_MAX], decoded_data[BLOCKL_MAX];
    int len;
    short pli, mode;
    int blockcount = 0;
    int packetlosscount = 0;

    /* Create structs */
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
iLBC_Enc_Inst_t Enc_Inst;
iLBC_Dec_Inst_t Dec_Inst;


/* get arguments and open files */

if ((argc!=5) && (argc!=6)) {
    fprintf(stderr,
    "\n*-----------------------------------------------*\n");
    fprintf(stderr,
    "   %s <20,30> input encoded decoded (channel)\n\n",
        argv[0]);
    fprintf(stderr,
    "   mode    : Frame size for the encoding/decoding\n");
    fprintf(stderr,
    "                 20 - 20 ms\n");
    fprintf(stderr,
    "                 30 - 30 ms\n");
    fprintf(stderr,
    "   input   : Speech for encoder (16-bit pcm file)\n");
    fprintf(stderr,
    "   encoded : Encoded bit stream\n");
    fprintf(stderr,
    "   decoded : Decoded speech (16-bit pcm file)\n");
    fprintf(stderr,
    "   channel : Packet loss pattern, optional (16-bit)\n");
    fprintf(stderr,
    "                  1 - Packet received correctly\n");
    fprintf(stderr,
    "                  0 - Packet Lost\n");
    fprintf(stderr,
    "*-----------------------------------------------*\n\n");
    exit(1);
}
mode=atoi(argv[1]);
if (mode != 20 && mode != 30) {
    fprintf(stderr,"Wrong mode %s, must be 20, or 30\n",
        argv[1]);
    exit(2);
}
if ( (ifileid=fopen(argv[2],"rb")) == NULL) {
    fprintf(stderr,"Cannot open input file %s\n", argv[2]);
    exit(2);}
if ( (efileid=fopen(argv[3],"wb")) == NULL) {
    fprintf(stderr, "Cannot open encoded file %s\n",
        argv[3]); exit(1);}
if ( (ofileid=fopen(argv[4],"wb")) == NULL) {
    fprintf(stderr, "Cannot open decoded file %s\n",
        argv[4]); exit(1);}
if (argc==6) {
    if( (cfileid=fopen(argv[5],"rb")) == NULL) {
        fprintf(stderr, "Cannot open channel file %s\n",
            argv[5]);
        exit(1);
    }
} else {
    cfileid=NULL;
}

/* print info */

fprintf(stderr, "\n");
fprintf(stderr,
    "*-----------------------------------------------------*\n");
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
fprintf(stderr,
    "*                                                    *\n");
fprintf(stderr,
    "*      iLBC test program                             *\n");
fprintf(stderr,
    "*                                                    *\n");
fprintf(stderr,
    "*                                                    *\n");
fprintf(stderr,
    "*----------------------------------------------------*\n");
fprintf(stderr,"\nMode           : %2d ms\n", mode);
fprintf(stderr,"Input file     : %s\n", argv[2]);
fprintf(stderr,"Encoded file   : %s\n", argv[3]);
fprintf(stderr,"Output file    : %s\n", argv[4]);
if (argc==6) {
    fprintf(stderr,"Channel file   : %s\n", argv[5]);
}
fprintf(stderr,"\n");

/* Initialization */

initEncode(&Enc_Inst, mode);
initDecode(&Dec_Inst, mode, 1);

/* Runtime statistics */

starttime=clock()/(float)CLOCKS_PER_SEC;

/* loop over input blocks */

while (fread(data,sizeof(short),Enc_Inst.blockl,ifileid)==
        Enc_Inst.blockl) {

    blockcount++;

    /* encoding */

    fprintf(stderr, "--- Encoding block %i --- ",blockcount);
    len=encode(&Enc_Inst, encoded_data, data);
    fprintf(stderr, "\r");

    /* write byte file */

    fwrite(encoded_data, sizeof(unsigned char), len, efileid);

    /* get channel data if provided */
    if (argc==6) {
        if (fread(&pli, sizeof(short), 1, cfileid)) {
            if ((pli!=0)&&(pli!=1)) {
                fprintf(stderr, "Error in channel file\n");
                exit(0);
            }
            if (pli==0) {
                /* Packet loss -> remove info from frame */
                memset(encoded_data, 0,
                    sizeof(short)*ILBCNOOFWORDS_MAX);
                packetlosscount++;
            }
        } else {
            fprintf(stderr, "Error. Channel file too short\n");
            exit(0);
        }
    } else {
        pli=1;
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
        }

        /* decoding */

        fprintf(stderr, "--- Decoding block %i --- ",blockcount);

        len=decode(&Dec_Inst, decoded_data, encoded_data, pli);
        fprintf(stderr, "\r");

        /* write output file */

        fwrite(decoded_data,sizeof(short),len,ofileid);
    }

    /* Runtime statistics */

    runtime = (float)(clock()/(float)CLOCKS_PER_SEC-starttime);
    outtime = (float)((float)blockcount*(float)mode/1000.0);
    printf("\n\nLength of speech file: %.1f s\n", outtime);
    printf("Packet loss          : %.1f%%\n",
        100.0*(float)packetlosscount/(float)blockcount);


    printf("Time to run iLBC     :");
    printf(" %.1f s (%.1f %% of realtime)\n\n", runtime,
        (100*runtime/outtime));

    /* close files */

    fclose(ifileid);  fclose(efileid); fclose(ofileid);
    if (argc==6) {
        fclose(cfileid);
    }
    return(0);
}
```

A.2 iLBC_encode.h

```
/*****************************************************************

    iLBC Speech Coder ANSI-C Source Code

    iLBC_encode.h

*****************************************************************/

#ifndef __iLBC_ILBCENCODE_H
#define __iLBC_ILBCENCODE_H

#include "iLBC_define.h"

short initEncode(                   /* (o) Number of bytes
                                           encoded */
    iLBC_Enc_Inst_t *iLBCenc_inst,  /* (i/o) Encoder instance */
    int mode                     /* (i) frame size mode */
);

void iLBC_encode(

    unsigned char *bytes,           /* (o) encoded data bits iLBC */
    float *block,                   /* (o) speech vector to
                                           encode */
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
    iLBC_Enc_Inst_t *iLBCenc_inst   /* (i/o) the general encoder
                                           state */
);

#endif
```

A.3 iLBC_encode.c

```
/******************************************************************

    iLBC Speech Coder ANSI-C Source Code

    iLBC_encode.c

******************************************************************/

#include <math.h>
#include <stdlib.h>
#include <string.h>

#include "iLBC_define.h"
#include "LPCencode.h"
#include "FrameClassify.h"
#include "StateSearchW.h"
#include "StateConstructW.h"
#include "helpfun.h"
#include "constants.h"
#include "packing.h"
#include "iCBSearch.h"
#include "iCBConstruct.h"
#include "hpInput.h"
#include "anaFilter.h"
#include "syntFilter.h"

/*----------------------------------------------------------------*
 *  Initiation of encoder instance.
 *----------------------------------------------------------------*/

short initEncode(                   /* (o) Number of bytes
                                           encoded */
    iLBC_Enc_Inst_t *iLBCenc_inst,  /* (i/o) Encoder instance */
    int mode                    /* (i) frame size mode */
){
    iLBCenc_inst->mode = mode;
    if (mode==30) {
        iLBCenc_inst->blockl = BLOCKL_30MS;
        iLBCenc_inst->nsub = NSUB_30MS;
        iLBCenc_inst->nasub = NASUB_30MS;
        iLBCenc_inst->lpc_n = LPC_N_30MS;
        iLBCenc_inst->no_of_bytes = NO_OF_BYTES_30MS;
        iLBCenc_inst->no_of_words = NO_OF_WORDS_30MS;
        iLBCenc_inst->state_short_len=STATE_SHORT_LEN_30MS;
        /* ULP init */
        iLBCenc_inst->ULP_inst=&ULP_30msTbl;
    }
    else if (mode==20) {
        iLBCenc_inst->blockl = BLOCKL_20MS;
        iLBCenc_inst->nsub = NSUB_20MS;
        iLBCenc_inst->nasub = NASUB_20MS;
        iLBCenc_inst->lpc_n = LPC_N_20MS;
        iLBCenc_inst->no_of_bytes = NO_OF_BYTES_20MS;
```


Documents Document Sharing Hub
Full document details: SCTE 24-22 2018

```
        iLBCenc_inst->no_of_words = NO_OF_WORDS_20MS;
        iLBCenc_inst->state_short_len=STATE_SHORT_LEN_20MS;
        /* ULP init */
        iLBCenc_inst->ULP_inst=&ULP_20msTbl;
    }
    else {
        exit(2);
    }

    memset((*iLBCenc_inst).anaMem, 0,
        LPC_FILTERORDER*sizeof(float));
    memcpy((*iLBCenc_inst).lsfold, lsfmeanTbl,
        LPC_FILTERORDER*sizeof(float));
    memcpy((*iLBCenc_inst).lsfdeqold, lsfmeanTbl,
        LPC_FILTERORDER*sizeof(float));
    memset((*iLBCenc_inst).lpc_buffer, 0,
        (LPC_LOOKBACK+BLOCKL_MAX)*sizeof(float));
    memset((*iLBCenc_inst).hpimem, 0, 4*sizeof(float));

    return (iLBCenc_inst->no_of_bytes);
}

/*----------------------------------------------------------------*
 *  main encoder function
 *----------------------------------------------------------------*/

void iLBC_encode(
    unsigned char *bytes,           /* (o) encoded data bits iLBC */
    float *block,                   /* (o) speech vector to
                                           encode */
    iLBC_Enc_Inst_t *iLBCenc_inst   /* (i/o) the general encoder
                                           state */
){

    float data[BLOCKL_MAX];
    float residual[BLOCKL_MAX], reverseResidual[BLOCKL_MAX];

    int start, idxForMax, idxVec[STATE_LEN];
    float reverseDecresidual[BLOCKL_MAX], mem[CB_MEML];
    int n, k, meml_gotten, Nfor, Nback, i, pos;
    int gain_index[CB_NSTAGES*NASUB_MAX],
        extra_gain_index[CB_NSTAGES];
    int cb_index[CB_NSTAGES*NASUB_MAX],extra_cb_index[CB_NSTAGES];
    int lsf_i[LSF_NSPLIT*LPC_N_MAX];
    unsigned char *pbytes;
    int diff, start_pos, state_first;
    float en1, en2;
    int index, ulp, firstpart;
    int subcount, subframe;
    float weightState[LPC_FILTERORDER];
    float syntdenum[NSUB_MAX*(LPC_FILTERORDER+1)];
    float weightdenum[NSUB_MAX*(LPC_FILTERORDER+1)];
    float decresidual[BLOCKL_MAX];

    /* high pass filtering of input signal if such is not done
           prior to calling this function */

    hpInput(block, iLBCenc_inst->blockl,
                data, (*iLBCenc_inst).hpimem);

    /* otherwise simply copy */

    /*memcpy(data,block,iLBCenc_inst->blockl*sizeof(float));*/
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
/* LPC of hp filtered input data */

LPCencode(syntdenum, weightdenum, lsf_i, data, iLBCenc_inst);


/* inverse filter to get residual */

for (n=0; n<iLBCenc_inst->nsub; n++) {
    anaFilter(&data[n*SUBL], &syntdenum[n*(LPC_FILTERORDER+1)],
        SUBL, &residual[n*SUBL], iLBCenc_inst->anaMem);
}

/* find state location */

start = FrameClassify(iLBCenc_inst, residual);

/* check if state should be in first or last part of the
two subframes */

diff = STATE_LEN - iLBCenc_inst->state_short_len;
en1 = 0;
index = (start-1)*SUBL;


for (i = 0; i < iLBCenc_inst->state_short_len; i++) {
    en1 += residual[index+i]*residual[index+i];
}
en2 = 0;
index = (start-1)*SUBL+diff;
for (i = 0; i < iLBCenc_inst->state_short_len; i++) {
    en2 += residual[index+i]*residual[index+i];
}

if (en1 > en2) {
    state_first = 1;
    start_pos = (start-1)*SUBL;
} else {
    state_first = 0;
    start_pos = (start-1)*SUBL + diff;
}

/* scalar quantization of state */

StateSearchW(iLBCenc_inst, &residual[start_pos],
    &syntdenum[(start-1)*(LPC_FILTERORDER+1)],
    &weightdenum[(start-1)*(LPC_FILTERORDER+1)], &idxForMax,
    idxVec, iLBCenc_inst->state_short_len, state_first);

StateConstructW(idxForMax, idxVec,
    &syntdenum[(start-1)*(LPC_FILTERORDER+1)],
    &decresidual[start_pos], iLBCenc_inst->state_short_len);

/* predictive quantization in state */

if (state_first) { /* put adaptive part in the end */

    /* setup memory */

    memset(mem, 0,
        (CB_MEML-iLBCenc_inst->state_short_len)*sizeof(float));
    memcpy(mem+CB_MEML-iLBCenc_inst->state_short_len,
        decresidual+start_pos,
        iLBCenc_inst->state_short_len*sizeof(float));
    memset(weightState, 0, LPC_FILTERORDER*sizeof(float));
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018

```
        /* encode sub-frames */

        iCBSearch(iLBCenc_inst, extra_cb_index, extra_gain_index,
            &residual[start_pos+iLBCenc_inst->state_short_len],
            mem+CB_MEML-stMemLTbl,
            stMemLTbl, diff, CB_NSTAGES,
            &weightdenum[start*(LPC_FILTERORDER+1)],
            weightState, 0);

        /* construct decoded vector */

        iCBConstruct(
            &decresidual[start_pos+iLBCenc_inst->state_short_len],
            extra_cb_index, extra_gain_index,
            mem+CB_MEML-stMemLTbl,
            stMemLTbl, diff, CB_NSTAGES);

    }
    else { /* put adaptive part in the beginning */

        /* create reversed vectors for prediction */

        for (k=0; k<diff; k++) {
            reverseResidual[k] = residual[(start+1)*SUBL-1
                -(k+iLBCenc_inst->state_short_len)];
        }

        /* setup memory */

        meml_gotten = iLBCenc_inst->state_short_len;
        for (k=0; k<meml_gotten; k++) {
            mem[CB_MEML-1-k] = decresidual[start_pos + k];
        }
        memset(mem, 0, (CB_MEML-k)*sizeof(float));
        memset(weightState, 0, LPC_FILTERORDER*sizeof(float));

        /* encode sub-frames */

        iCBSearch(iLBCenc_inst, extra_cb_index, extra_gain_index,
            reverseResidual, mem+CB_MEML-stMemLTbl, stMemLTbl,
            diff, CB_NSTAGES,
            &weightdenum[(start-1)*(LPC_FILTERORDER+1)],
            weightState, 0);

        /* construct decoded vector */

        iCBConstruct(reverseDecresidual, extra_cb_index,
            extra_gain_index, mem+CB_MEML-stMemLTbl, stMemLTbl,
            diff, CB_NSTAGES);

        /* get decoded residual from reversed vector */

        for (k=0; k<diff; k++) {
            decresidual[start_pos-1-k] = reverseDecresidual[k];


        }
    }

    /* counter for predicted sub-frames */

    subcount=0;
```


Documents
Document Sharing Hub
Full document details: SCTE 24-22 2018