Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 13, 2011

K. Vos
S. Jensen
K. Soerensen
Skype Technologies S.A.

September 9, 2010

SILK Speech Codec
draft-vos-silk-02

Abstract

This document describes SILK, a speech codec for real-time, packet-based voice communications. Targeting a diverse range of operating environments, SILK provides scalability in several dimensions. Four different sampling frequencies are supported for encoding the audio input signal. Adaptation to network characteristics is provided through control of bitrate, packet rate, packet loss resilience and use of discontinuous transmission (DTX). And several different complexity levels let SILK take advantage of available processing power without relying on it. Each of these properties can be adjusted during operation of the codec on a frame-by-frame basis.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on March 13, 2011.

Copyright Notice

Table of Contents

1. Introduction

A central component in voice communications is the speech codec, which compresses the audio signal for efficient transmission over a network. A good speech codec achieves high coding efficiency, meaning that it delivers high audio quality at a given bitrate. However, for a good user experience in a broad range of applications and environments, a speech codec should also be able adapt its operating point to the characteristics and limitations of network, hardware and audio signal. SILK [silk-website] is a speech codec for real-time voice communications designed and developed by Skype [skype-website] to offer this kind of scalability. This document describes the technical details of SILK.

The source code for the reference implementation of the SILK codec is provided in Appendix A. This fixed-point source code is the normative specification of the codec. The corresponding text description in this document is provided for informative purposes. For more information about SILK, the patent licensing terms, and to download the source code in a convenient format, please visit [silk-website]. For any additional questions, please send an email to SILKSupport@skype.net.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

2. Outline of the Codec

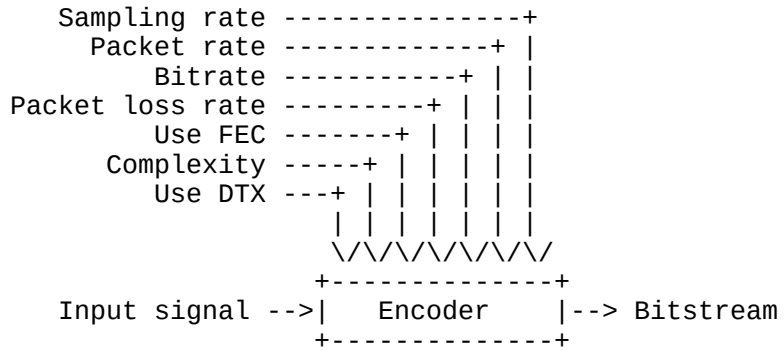
The SILK codec consist of an encoder and a decoder as described in Section 2.1 and Section 2.2, respectively.

2.1. Encoder

We start the description of the encoder by listing the parameters that controls the operating point of the encoder. Afterwards, we describe the encoder components in detail.

2.1.1. Control Parameters

The encoder with control parameters specifying the operating point is depicted in Figure 1. All control parameters can be changed during regular operation of the codec, when inputting a frame of audio data, without interrupting the audio stream from encoder to decoder. The codec control parameters are described in Section 2.1.1.1 to Section 2.1.1.7.



Block diagram illustrating the control parameters that specifies the operating point of the SILK encoder.

Figure 1

2.1.1.1. Sampling Rate

As described in [silk-payload], SILK negotiates one of four modes during call setup:

- Narrowband (NB): 8 kHz sampling rate
- Mediumband (MB): 8 or 12 kHz sampling rate

- Wideband (WB): 8, 12 or 16 kHz sampling rate
- Super Wideband (SWB): 8, 12, 16 or 24 kHz sampling rate

The purpose of these modes is to allow the decoder to limit the highest sampling rate used by the encoder.

Regardless of the mode it operates in, the encoder accepts input signals sampled at 8, 12, 16 or 24 kHz and allows the input sampling rate to be changed in real-time. Internally, the encoder resamples the input signal to a lower sampling rate if the mode does not support the input sampling rate or if the bitrate is so low that reducing the sampling rate improves overall fidelity.

2.1.1.2. Packet Rate

SILK encodes frames of 20 milliseconds at a time and can combine 1, 2, 3, 4 or 5 frames in one payload, thus creating one packet every 20, 40, 60, 80 or 100 milliseconds. Because of the overhead from IP/UDP/RTP headers, sending fewer packets per second reduces the bitrate, but increases latency and sensitivity to packet losses as losing one packet constitutes a loss of a bigger chunk of audio signal. Reducing the packet rate also slightly improves coding efficiency because some parameters are conditionally encoded in all but the first frame in a packet.

2.1.1.3. Bitrate

The bitrate can be set between 6 and 40 kbps. A higher bitrate improves audio quality by lowering the amount of quantization noise in the decoded signal. The required bitrate for a given level of quantization noise is approximately linear with the sampling rate. Good quality is achieved at around 1 bit/sample, and at 1.5 bits/sample the quality becomes transparent for most material.

2.1.1.4. Packet Loss Resilience

Speech codecs often exploit inter-frame correlations to reduce the bitrate at a cost in error propagation: After losing one packet several packets need to be received before the decoder is able to accurately reconstruct the speech signal. The extent to which SILK exploits inter-frame dependencies can be adjusted on the fly to choose a trade-off between bitrate and amount of error propagation.

2.1.1.5. Use FEC

Another mechanism providing robustness against packet loss is the in-band Forward Error Correction (FEC). Packets that are determined to contain perceptually important speech information, such as onsets or transients, are encoded again at a lower bitrate and this re-encoded

information is added to a subsequent packet.

2.1.1.6. Complexity

SILK has several optional optimizations that can be enabled to reduce the CPU load by a few times, at the cost of increasing the bitrate by a few percent. The most important algorithmic parts controlled by the three complexity settings (that is, high, medium, and low) are:

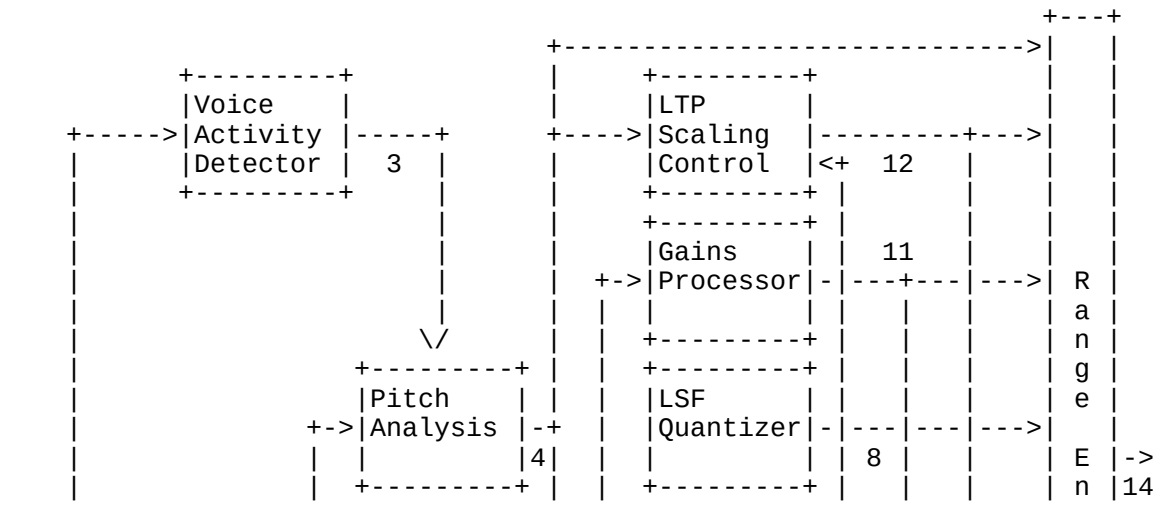
- The filter order of the whitening filter and the downsampling quality in the pitch analysis.
- The filter order of the short-term noise shaping filter used in the prefilter and noise shaping quantizer.
- Adjustment of the number of survivors that are carried over between stages in the multi-stage LSF vector quantization.
- The number of states in delayed decision quantization of the residual signal.

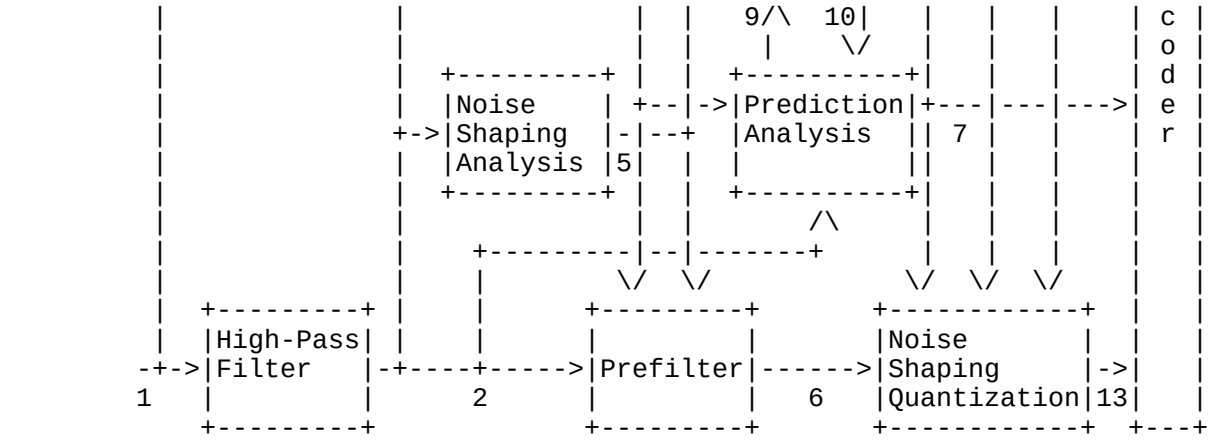
2.1.1.7. Use DTX

Discontinuous Transmission (DTX) reduces the bitrate during silence or background noise. When DTX is enabled, only one frame is encoded every 400 milliseconds.

2.1.2. Encoder Modules

In the following, we focus on the core encoder and describe its components. For simplicity, we will refer to the core encoder simply as the encoder in the remainder of this document. An overview of the encoder is given in Figure 2.





1: Input speech signal
2: High passed input signal
3: Voice activity estimate
4: Pitch lags (per 5 ms) and voicing decision (per 20 ms)
5: Noise shaping quantization coefficients
- Short term synthesis and analysis noise shaping coefficients (per 5 ms)
- Long term synthesis and analysis noise shaping coefficients (per 5 ms and for voiced speech only)
- Noise shaping tilt (per 5 ms)
- Quantizer gain/step size (per 5 ms)

6: Input signal filtered with analysis noise shaping filters
7: Short and long term prediction coefficients LTP (per 5 ms) and LPC (per 20 ms)
8: LSF quantization indices
9: LSF coefficients
10: Quantized LSF coefficients
11: Processed gains, and synthesis noise shape coefficients
12: LTP state scaling coefficient. Controlling error propagation / prediction gain trade-off
13: Quantized signal
14: Range encoded bitstream

Encoder block diagram.

Figure 2

2.1.2.1. Voice Activity Detection

The input signal is processed by a VAD (Voice Activity Detector) to produce a measure of voice activity, and also spectral tilt and signal-to-noise estimates, for each frame. The VAD uses a sequence of half-band filterbanks to split the signal in four subbands: 0 - Fs/16, Fs/16 - Fs/8, Fs/8 - Fs/4, and Fs/4 - Fs/2, where Fs is the sampling frequency, that is, 8, 12, 16 or 24 kHz. The lowest subband, from 0 - Fs/16 is high-pass filtered with a first-order MA (Moving Average) filter (with transfer function H(z) = 1-z^(-1)) to reduce the energy at the lowest frequencies. For each frame, the signal energy per subband is computed. In each subband, a noise level estimator tracks the background noise level and an SNR (Signal-to-Noise Ratio) value is computed as the logarithm of the ratio of energy to noise level. Using these intermediate variables, the following parameters are calculated for use in other SILK modules:

- Average SNR. The average of the subband SNR values.
- Smoothed subband SNRs. Temporally smoothed subband SNR values.
- Speech activity level. Based on the average SNR and a weighted average of the subband energies.
- Spectral tilt. A weighted average of the subband SNRs, with positive weights for the low subbands and negative weights for the high subbands.

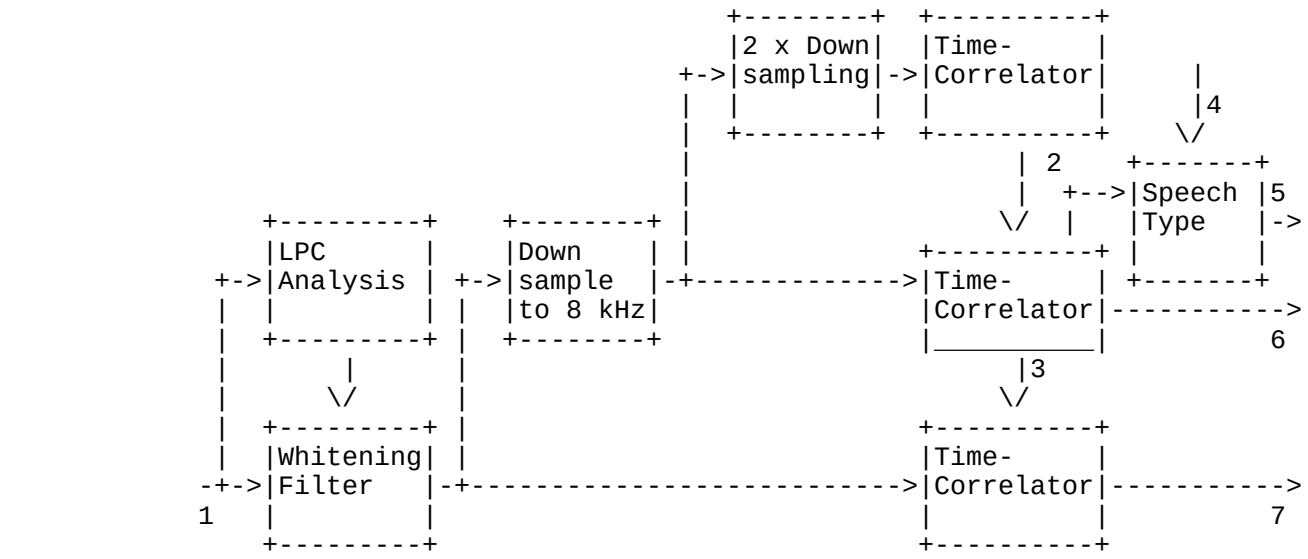
2.1.2.2. High-Pass Filter

The input signal is filtered by a high-pass filter to remove the lowest part of the spectrum that contains little speech energy and may contain background noise. This is a second order ARMA (Auto Regressive Moving Average) filter with a cut-off frequency around 70 Hz.

In the future, a music detector may also be used to lower the cut-off frequency when the input signal is detected to be music rather than speech.

2.1.2.3. Pitch Analysis

The high-passed input signal is processed by the open loop pitch estimator shown in Figure 3.



Block diagram of the pitch estimator.

Figure 3

The pitch analysis finds a binary voiced/unvoiced classification, and, for frames classified as voiced, four pitch lags per frame - one for each 5 ms subframe - and a pitch correlation indicating the periodicity of the signal. The input is first whitened using a Linear Prediction (LP) whitening filter, where the coefficients are computed through standard Linear Prediction Coding (LPC) analysis. The order of the whitening filter is 16 for best results, but is reduced to 12 for medium complexity and 8 for low complexity modes. The whitened signal is analyzed to find pitch lags for which the time correlation is high. The analysis consists of three stages for reducing the complexity:

o In the first stage, the whitened signal is downsampled to 4 kHz (from 8 kHz) and the current frame is correlated to a signal delayed by a range of lags, starting from a shortest lag corresponding to 500 Hz, to a longest lag corresponding to 56 Hz.

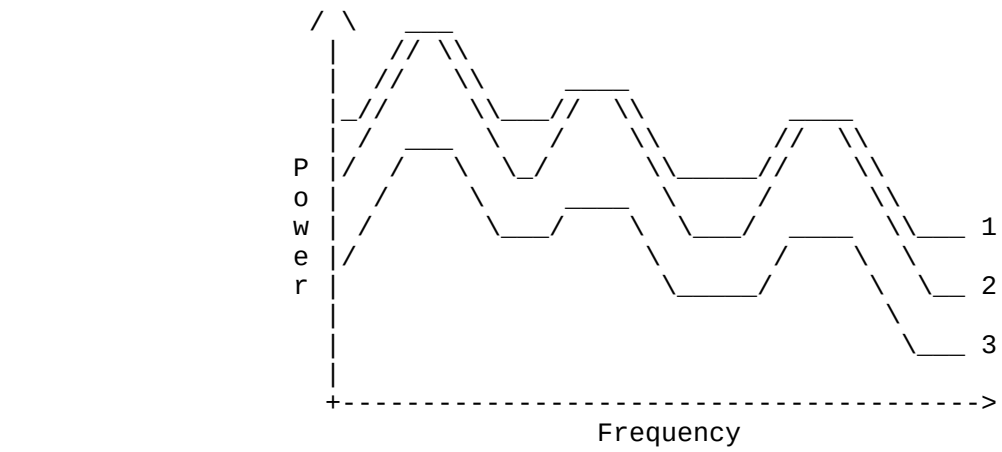
- The second stage operates on a 8 kHz signal (downsampled from 12, 16 or 24 kHz) and measures time correlations only near the lags corresponding to those that had sufficiently high correlations in the first stage. The resulting correlations are adjusted for a small bias towards short lags to avoid ending up with a multiple of the true pitch lag. The highest adjusted correlation is compared to a threshold depending on:
  - Whether the previous frame was classified as voiced
  - The speech activity level
  - The spectral tilt.

  If the threshold is exceeded, the current frame is classified as voiced and the lag with the highest adjusted correlation is stored for a final pitch analysis of the highest precision in the third stage.
- The last stage operates directly on the whitened input signal to compute time correlations for each of the four subframes independently in a narrow range around the lag with highest correlation from the second stage.

2.1.2.4. Noise Shaping Analysis

The noise shaping analysis finds gains and filter coefficients used in the prefilter and noise shaping quantizer. These parameters are chosen such that they will fulfil several requirements:

- Balancing quantization noise and bitrate. The quantization gains determine the step size between reconstruction levels of the excitation signal. Therefore, increasing the quantization gain amplifies quantization noise, but also reduces the bitrate by lowering the entropy of the quantization indices.
- Spectral shaping of the quantization noise; the noise shaping quantizer is capable of reducing quantization noise in some parts of the spectrum at the cost of increased noise in other parts without substantially changing the bitrate. By shaping the noise such that it follows the signal spectrum, it becomes less audible. In practice, best results are obtained by making the shape of the noise spectrum slightly flatter than the signal spectrum.
- Deemphasizing spectral valleys; by using different coefficients in the analysis and synthesis part of the prefilter and noise shaping quantizer, the levels of the spectral valleys can be decreased relative to the levels of the spectral peaks such as speech formants and harmonics. This reduces the entropy of the signal, which is the difference between the coded signal and the quantization noise, thus lowering the bitrate.
- Matching the levels of the decoded speech formants to the levels of the original speech formants; an adjustment gain and a first order tilt coefficient are computed to compensate for the effect of the noise shaping quantization on the level and spectral tilt.



1: Input signal spectrum
2: Deemphasized and level matched spectrum
3: Quantization noise spectrum

Noise shaping and spectral de-emphasis illustration.

Figure 4

Figure 4 shows an example of an input signal spectrum (1). After de-emphasis and level matching, the spectrum has deeper valleys (2). The quantization noise spectrum (3) more or less follows the input signal spectrum, while having slightly less pronounced peaks. The entropy, which provides a lower bound on the bitrate for encoding the excitation signal, is proportional to the area between the deemphasized spectrum (2) and the quantization noise spectrum (3). Without de-emphasis, the entropy is proportional to the area between input spectrum (1) and quantization noise (3) - clearly higher.

The transformation from input signal to deemphasized signal can be described as a filtering operation with a filter

$$H(z) = G * (1 - c\_tilt * z^{-1}) * \frac{Wana(z)}{Wsyn(z)},$$

having an adjustment gain G, a first order tilt adjustment filter with tilt coefficient c_tilt, and where

$$Wana(z) = (1 - \sum_{k=1}^{16} (a\_ana(k) * z^{(-k)})*(1 - \sum_{k=-d}^{d} b\_ana(k)*z^{(-k-L)}),$$

is the analysis part of the de-emphasis filter, consisting of the short-term shaping filter with coefficients a_ana(k), and the long-term shaping filter with coefficients b_ana(k) and pitch lag L. The parameter d determines the number of long-term shaping filter taps.

Similarly, but without the tilt adjustment, the synthesis part can be written as

$$Wsyn(z) = (1 - \sum_{k=1}^{16} (a\_syn(k) * z^{(-k)})*(1 - \sum_{k=-d}^{d} b\_syn(k)*z^{(-k-L)}).$$

All noise shaping parameters are computed and applied per subframe of 5 milliseconds. First, an LPC analysis is performed on a windowed signal block of 15 milliseconds. The signal block has a look-ahead of 5 milliseconds relative to the current subframe, and the window is an asymmetric sine window. The LPC analysis is done with the autocorrelation method, with an order of 16 for best quality or 12 in low complexity operation. The quantization gain is found as the square-root of the residual energy from the LPC analysis, multiplied by a value inversely proportional to the coding quality control parameter and the pitch correlation.

Next we find the two sets of short-term noise shaping coefficients a_ana(k) and a_syn(k), by applying different amounts of bandwidth expansion to the coefficients found in the LPC analysis. This bandwidth expansion moves the roots of the LPC polynomial towards the origin, using the formulas

$$a\_ana(k) = a(k)*g\_ana^k, \text{ and}$$
$$a\_syn(k) = a(k)*g\_syn^k,$$

where a(k) is the k'th LPC coefficient and the bandwidth expansion factors g_ana and g_syn are calculated as

$$g\_ana = 0.94 - 0.02*C, \text{ and}$$
$$g\_syn = 0.94 + 0.02*C,$$

where C is the coding quality control parameter between 0 and 1. Applying more bandwidth expansion to the analysis part than to the synthesis part gives the desired de-emphasis of spectral valleys in between formants.

The long-term shaping is applied only during voiced frames. It uses three filter taps, described by

$$b\_ana = F\_ana * [0.25, 0.5, 0.25], \text{ and}$$
$$b\_syn = F\_syn * [0.25, 0.5, 0.25].$$

For unvoiced frames these coefficients are set to 0. The multiplication factors F_ana and F_syn are chosen between 0 and 1, depending on the coding quality control parameter, as well as the calculated pitch correlation and smoothed subband SNR of the lowest subband. By having F_ana less than F_syn, the pitch harmonics are emphasized relative to the valleys in between the harmonics.

The tilt coefficient c_tilt is for unvoiced frames chosen as

$$c\_tilt = 0.4, \text{ and as}$$
$$c\_tilt = 0.04 + 0.06 * C$$

for voiced frames, where C again is the coding quality control parameter and is between 0 and 1.

The adjustment gain G serves to correct any level mismatch between original and decoded signal that might arise from the noise shaping and de-emphasis. This gain is computed as the ratio of the prediction gain of the short-term analysis and synthesis filter coefficients. The prediction gain of an LPC synthesis filter is the square-root of the output energy when the filter is excited by a unit-energy impulse on the input. An efficient way to compute the prediction gain is by first computing the reflection coefficients from the LPC coefficients through the step-down algorithm, and extracting the prediction gain from the reflection coefficients as

$$\sum_{K}$$

$$predGain = \left(\prod_{k=1} 1 - (r_k)^2 \right)^{(-0.5)},$$

where r_k is the k'th reflection coefficient.

Initial values for the quantization gains are computed as the square-root of the residual energy of the LPC analysis, adjusted by the coding quality control parameter. These quantization gains are later adjusted based on the results of the prediction analysis.

2.1.2.5. Prefilter

In the prefilter the input signal is filtered using the spectral valley de-emphasis filter coefficients from the noise shaping analysis, see Section 2.1.2.4. By applying only a filter with the noise shaping analysis coefficients to the input signal, it provides the input to the noise shaping quantizer.

2.1.2.6. Prediction Analysis

The prediction analysis is performed in one of two ways depending on how the pitch estimator classified the frame. The processing for voiced and unvoiced speech are described in Section 2.1.2.6.1 and Section 2.1.2.6.2, respectively. Inputs to this function include the pre-whitened signal from the pitch estimator, see Section 2.1.2.3.

2.1.2.6.1. Voiced Speech

For a frame of voiced speech the pitch pulses will remain dominant in the pre-whitened input signal. Further whitening is desirable as it leads to higher quality at the same available bit-rate. To achieve this, a Long-Term Prediction (LTP) analysis is carried out to estimate the coefficients of a fifth order LTP filter for each of four sub-frames. The LTP coefficients are used to find an LTP residual signal with the simulated output signal as input to obtain better modelling of the output signal. This LTP residual signal is the input to an LPC analysis where the LPCs are estimated using Burgs method, such that the residual energy is minimized. The estimated LPCs are converted to a Line Spectral Frequency (LSF) vector, and quantized as described in Section 2.1.2.7. After quantization, the quantized LSF vector is converted to LPC coefficients and hence by using these quantized coefficients the encoder remains fully synchronized with the decoder. The LTP coefficients are quantized using a method described in Section 2.1.2.8. The quantized LPC and LTP coefficients are now used to filter the high-pass filtered input signal and measure a residual energy for each of the four subframes.

2.1.2.6.2. Unvoiced Speech

For a speech signal that has been classified as unvoiced there is no need for LTP filtering as it has already been determined that the pre-whitened input signal is not periodic enough within the allowed pitch period range for an LTP analysis to be worth-while the cost in terms of complexity and rate. Therefore, the pre-whitened input signal is discarded and instead the high-pass filtered input signal is used for LPC analysis using Burgs method. The resulting LPC coefficients are converted to an LSF vector, quantized as described in the following section and transformed back to obtain quantized LPC coefficients. The quantized LPC coefficients are used to filter the high-pass filtered input signal and measure a residual energy for each of the four subframes.

2.1.2.7. LSF Quantization

The purpose of quantization in general is to significantly lower the bit rate at the cost of some introduced distortion. A higher rate should always result in lower distortion, and lowering the rate will generally lead to higher distortion. A commonly used but generally sub-optimal approach is to use a quantization method with a constant rate where only the error is minimized when quantizing.

2.1.2.7.1. Rate-Distortion Optimization

Instead, we minimize an objective function that consists of a weighted sum of rate and distortion, and use a codebook with an associated non-uniform rate table. Thus, we take into account that the probability mass function for selecting the codebook entries are by no means guaranteed to be uniform in our scenario. The advantage of this approach is that it ensures that rarely used codebook vector centroids, which are modelling statistical outliers in the training set can be quantized with a low error but with a relatively high cost in terms of a high rate. At the same time this approach also provides the advantage that frequently used centroids are modelled with low error and a relatively low rate. This approach will lead to equal or lower distortion than the fixed rate codebook at any given average rate, provided that the data is similar to the data used for training the codebook.

2.1.2.7.2. Error Mapping

Instead of minimizing the error in the LSF domain, we map the errors to better approximate spectral distortion by applying an individual weight to each element in the error vector. The weight vectors are calculated for each input vector using the Inverse Harmonic Mean Weighting (IHMW) function proposed by Laroia et al., see

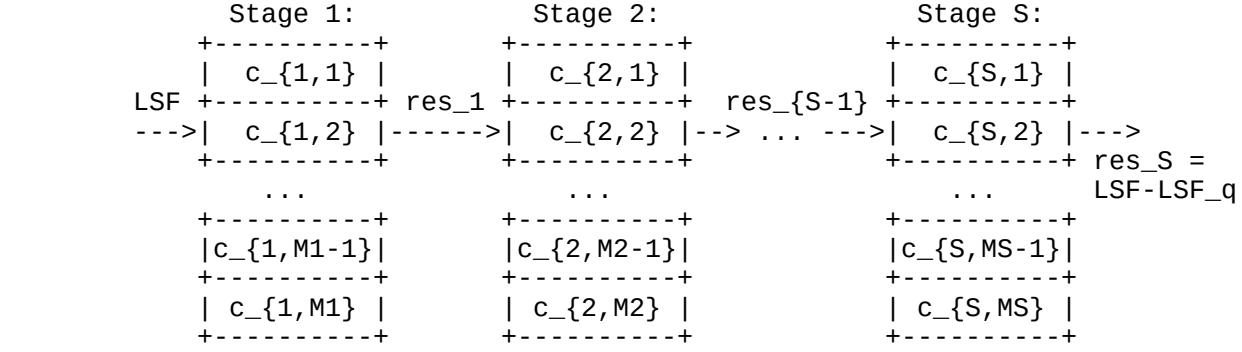
[laroia-icassp]. Consequently, we solve the following minimization problem, i.e.,

$$LSF\_q = \underset{c \in C}{\operatorname{argmin}} \{ (LSF - c)' * W * (LSF - c) + mu * rate \},$$

where LSF_q is the quantized vector, LSF is the input vector to be quantized, and c is the quantized LSF vector candidate taken from the set C of all possible outcomes of the codebook.

2.1.2.7.3. Multi-Stage Vector Codebook

We arrange the codebook in a multiple stage structure to achieve a quantizer that is both memory efficient and highly scalable in terms of computational complexity, see e.g. [sinervo-norsig]. In the first stage the input is the LSF vector to be quantized, and in any other stage s > 1, the input is the quantization error from the previous stage, see Figure 5.



Multi-Stage LSF Vector Codebook Structure.

Figure 5

By storing total of M codebook vectors, i.e.,

$$M = \sum_{s=1}^{S} Ms,$$

where M_s is the number of vectors in stage s, we obtain a total of

$$T = \prod_{s=1}^{S} Ms$$

possible combinations for generating the quantized vector. It is for example possible to represent 2^36 uniquely combined vectors using only 216 vectors in memory, as done in SILK for voiced speech at all sample frequencies above 8 kHz.

2.1.2.7.4. Survivor Based Codebook Search

This number of possible combinations is far too high for a full search to be carried out for each frame so for all stages but the last, i.e., s smaller than S, only the best min(L, Ms) centroids are carried over to stage s+1. In each stage the objective function, i.e., the weighted sum of accumulated bit-rate and distortion, is evaluated for each codebook vector entry and the results are sorted. Only the best paths and the corresponding quantization errors are considered in the next stage. In the last stage S the single best path through the multistage codebook is determined. By varying the maximum number of survivors from each stage to the next L, the complexity can be adjusted in real-time at the cost of a potential increase when evaluating the objective function for the resulting quantized vector. This approach scales all the way between the two extremes, L=1 being a greedy search, and the desirable but infeasible full search, L=T/MS. In fact, a performance almost as good as what can be achieved with the infeasible full search can be obtained at a substantially lower complexity by using this approach, see e.g. [leblanc-tsap].

2.1.2.7.5. LSF Stabilization

If the input is stable, finding the best candidate will usually result in the quantized vector also being stable, but due to the multi-stage approach it could in theory happen that the best quantization candidate is unstable and because of this there is a need to explicitly ensure that the quantized vectors are stable. Therefore we apply a LSF stabilization method which ensures that the LSF parameters are within valid range, increasingly sorted, and have minimum distances between each other and the border values that have been pre-determined as the 0.01 percentile distance values from a large training set.

2.1.2.7.6. Off-Line Codebook Training

The vectors and rate tables for the multi-stage codebook have been trained by minimizing the average of the objective function for LSF vectors from a large training set.

2.1.2.8. LTP Quantization

For voiced frames, the prediction analysis described in Section 2.1.2.6.1 resulted in four sets (one set per subframe) of five LTP coefficients, plus four weighting matrices. Also, the LTP coefficients for each subframe are quantized using entropy constrained vector quantization. A total of three vector codebooks are available for quantization, with different rate-distortion trade-offs. The three codebooks have 10, 20 and 40 vectors and average rates of about 3, 4, and 5 bits per vector, respectively. Consequently, the first codebook has larger average quantization distortion at a lower rate, whereas the last codebook has smaller average quantization distortion at a higher rate. Given the weighting matrix W_ltp and LTP vector b, the weighted rate-distortion measure for a codebook vector cb_i with rate r_i is give by

$$RD = u * (b - cb\_i)' * W\_ltp * (b - cb\_i) + r\_i,$$

where u is a fixed, heuristically-determined parameter balancing the distortion and rate. Which codebook gives the best performance for a given LTP vector depends on the weighting matrix for that LTP vector. For example, for a low valued W_ltp, it is advantageous to use the codebook with 10 vectors as it has a lower average rate. For a large W_ltp, on the other hand, it is often better to use the codebook with 40 vectors, as it is more likely to contain the best codebook vector. The weighting matrix W_ltp depends mostly on two aspects of the input signal. The first is the periodicity of the signal; the more periodic the larger W_ltp. The second is the change in signal energy in the current subframe, relative to the signal one pitch lag earlier. A decaying energy leads to a larger W_ltp than an increasing energy. Both aspects do not fluctuate very fast which causes the W_ltp matrices for different subframes of one frame often to be similar. As a result, one of the three codebooks typically gives good performance for all subframes. Therefore the codebook search for the subframe LTP vectors is constrained to only allow codebook vectors to be chosen from the same codebook, resulting in a rate reduction.

To find the best codebook, each of the three vector codebooks is used to quantize all subframe LTP vectors and produce a combined weighted

rate-distortion measure for each vector codebook and the vector codebook with the lowest combined rate-distortion over all subframes is chosen. The quantized LTP vectors are used in the noise shaping quantizer, and the index of the codebook plus the four indices for the four subframe codebook vectors are passed on to the range encoder.

2.1.2.9. Noise Shaping Quantizer

The noise shaping quantizer independently shapes the signal and coding noise spectra to obtain a perceptually higher quality at the same bitrate.

The prefilter output signal is multiplied with a compensation gain G computed in the noise shaping analysis. Then the output of a synthesis shaping filter is added, and the output of a prediction filter is subtracted to create a residual signal. The residual signal is multiplied by the inverse quantized quantization gain from the noise shaping analysis, and input to a scalar quantizer. The quantization indices of the scalar quantizer represent a signal of pulses that is input to the pyramid range encoder. The scalar quantizer also outputs a quantization signal, which is multiplied by the quantized quantization gain from the noise shaping analysis to create an excitation signal. The output of the prediction filter is added to the excitation signal to form the quantized output signal y(n). The quantized output signal y(n) is input to the synthesis shaping and prediction filters.

2.1.2.10. Range Encoder

Range encoding is a well known method for entropy coding in which a bitstream sequence is continually updated with every new symbol, based on the probability for that symbol. It is similar to arithmetic coding but rather than being restricted to generating binary output symbols, it can generate symbols in any chosen number base. In SILK all side information is range encoded. Each quantized parameter has its own cumulative density function based on histograms for the quantization indices obtained by running a training database.

2.1.2.10.1. Bitstream Encoding Details

TBD.

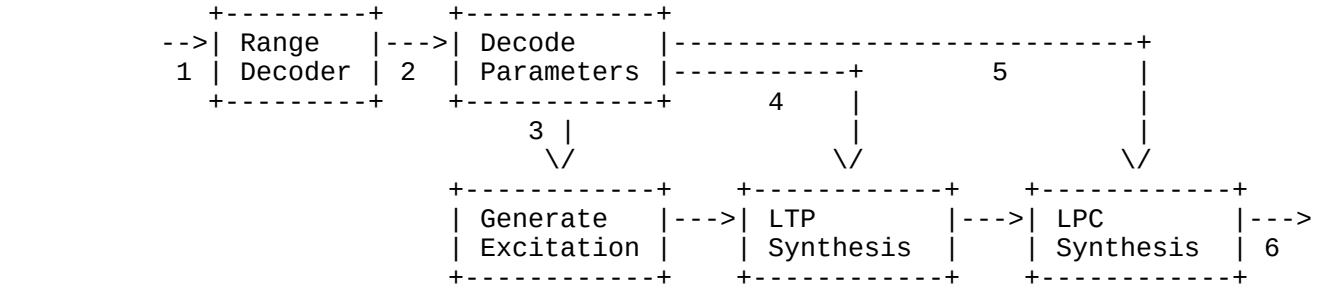
2.2. Decoder

At the receiving end, the received packets are by the range decoder split into a number of frames contained in the packet. Each of which contains the necessary information to reconstruct a 20 ms frame of

the output signal.

2.2.1. Decoder Modules

An overview of the decoder is given in Figure 6.



1: Range encoded bitstream
2: Coded parameters
3: Pulses and gains
4: Pitch lags and LTP coefficients
5: LPC coefficients
6: Decoded signal

Decoder block diagram.

Figure 6

2.2.1.1. Range Decoder

The range decoder decodes the encoded parameters from the received bitstream. Output from this function includes the pulses and gains for the excitation signal generation, as well as LTP and LSF codebook indices, which are needed for decoding LTP and LPC coefficients needed for LTP and LPC synthesis filtering the excitation signal, respectively.

2.2.1.2. Decode Parameters

Pulses and gains are decoded from the parameters that was decoded by the range decoder.

When a voiced frame is decoded and LTP codebook selection and indices are received, LTP coefficients are decoded using the selected codebook by choosing the vector that corresponds to the given

codebook index in that codebook. This is done for each of the four subframes. The LPC coefficients are decoded from the LSF codebook by first adding the chosen vectors, one vector from each stage of the codebook. The resulting LSF vector is stabilized using the same method that was used in the encoder, see Section 2.1.2.7.5. The LSF coefficients are then converted to LPC coefficients, and passed on to the LPC synthesis filter.

2.2.1.3. Generate Excitation

The pulses signal is multiplied with the quantization gain to create the excitation signal.

2.2.1.4. LTP Synthesis

For voiced speech, the excitation signal e(n) is input to an LTP synthesis filter that will recreate the long term correlation that was removed in the LTP analysis filter and generate an LPC excitation signal e_LPC(n), according to

$$e\_LPC(n) = e(n) + \sum_{i=-d}^{d} e(n - L - i) * b\_i,$$

using the pitch lag L, and the decoded LTP coefficients b_i. For unvoiced speech, the output signal is simply a copy of the excitation signal, i.e., e_LPC(n) = e(n).

2.2.1.5. LPC Synthesis

In a similar manner, the short-term correlation that was removed in the LPC analysis filter is recreated in the LPC synthesis filter. The LPC excitation signal e_LPC(n) is filtered using the LTP coefficients a_i, according to

$$y(n) = e\_LPC(n) + \sum_{i=1}^{d\_LPC} e\_LPC(n - i) * a\_i,$$

where d_LPC is the LPC synthesis filter order, and y(n) is the

decoded output signal.

3. Security Considerations

To Be Defined.

4. Informative References

[laroia-icassp]
Laroia, R., Phamdo, N., and N. Farvardin, "Robust and Efficient Quantization of Speech LSP Parameters Using Structured Vector Quantization", ICASSP-1991, Proc. IEEE Int. Conf. Acoust., Speech, Signal Processing, pp. 641-644, October 1991.

[leblanc-tsap]
LeBlanc, W., Bhattacharya, B., Mahmoud, S., and V. Cuperman, "Efficient Search and Design Procedures for Robust Multi-Stage VQ of LPC Parameters for 4 kb/s Speech Coding", IEEE Transactions on Speech and Audio Processing, Vol. 1, No. 4, October 1993.

[silk-payload]
Spittka, J., Astrom, H., and K. Vos, "RTP Payload Format and File Storage Format for SILK Speech and Audio Codec", http://tools.ietf.org/id/draft-spittka-silk-payload-format-01.txt.

[silk-website]
"SILK", SILK website https://developer.skype.com/silk/.

[sinervo-norsig]
Sinervo, U., Nurminen, J., Heikkinen, A., and J. Saarinen, "Evaluation of Split and Multistage Techniques in LSF Quantization", NORSIG-2001, Norsk symposium i signalbehandling, Trondheim, Norge, October 2001.

[skype-website]
"Skype", Skype website http://www.skype.com/.

Appendix A. Reference Implementation

This appendix contains the source code of a fixed-point C reference implementation of SILK.

A.1. Makefile

```
#
# Makefile for Silk SDK
#
# Copyright (c) 2010, Skype Limited
# All rights reserved.
#

#Platform detection and settings

BUILD_OS := $(shell uname | sed -e 's/^.*Darwin.*/MacOS-X/ ; s/^.*CYGWIN.*/Window
s/')

BUILD_ARCHITECTURE := $(shell uname -m | sed -e 's/i686/i386/')

EXESUFFIX =
LIBPREFIX = lib
LIBSUFFIX = .a
OBJSUFFIX = .o

CC     = $(TOOLCHAIN_PREFIX)gcc$(TOOLCHAIN_SUFFIX)
AR     = $(TOOLCHAIN_PREFIX)ar
RANLIB = $(TOOLCHAIN_PREFIX)ranlib
CP     = $(TOOLCHAIN_PREFIX)cp

cflags-from-defines    = $(addprefix -D,$(1))
cflags-from-includes   = $(addprefix -I,$(1))
ldflags-from-ldlibdirs = $(addprefix -L,$(1))
ldlibs-from-libs       = $(addprefix -l,$(1))

CFLAGS  += -Wall -enable-threads -O3

CFLAGS  += $(call cflags-from-defines,$(CDEFINES))
CFLAGS  += $(call cflags-from-includes,$(CINCLUDES))
LDFLAGS += $(call ldflags-from-ldlibdirs,$(LDLIBDIRS))
LDLIBS  += $(call ldlibs-from-libs,$(LIBS))

COMPILE.c.cmdline   = $(CC) -c $(CFLAGS) -o $@ $<
LINK.o.cmdline      = $(LINK.o) -lm $^ $(LDLIBS) -o $@$(EXESUFFIX)
ARCHIVE.cmdline     = $(AR) $(ARFLAGS) $@ $^ && $(RANLIB) $@

%$(OBJSUFFIX):%.c
```

```
        $(COMPILE.c.cmdline)

# Directives

CINCLUDES += interface src test

# VPATH e.g. VPATH = src:../headers
VPATH = ./ \
        interface \
        src \
        test

# Variable definitions
LIB_NAME = SKP_SILK_SDK
TARGET = $(LIBPREFIX)$(LIB_NAME)$(LIBSUFFIX)

SRCS_C = $(wildcard src/*.c)

OBJS := $(patsubst %.c,%$(OBJSUFFIX),$(SRCS_C))

ENCODER_SRCS_C = test/Encoder.c
ENCODER_OBJS := $(patsubst %.c,%$(OBJSUFFIX),$(ENCODER_SRCS_C))

DECODER_SRCS_C = test/Decoder.c
DECODER_OBJS := $(patsubst %.c,%$(OBJSUFFIX),$(DECODER_SRCS_C))

SIGNALCMP_SRCS_C = test/signalCompare.c
SIGNALCMP_OBJS := $(patsubst %.c,%$(OBJSUFFIX),$(SIGNALCMP_SRCS_C))

LIBS = \
        $(LIB_NAME)

LDLIBDIRS = ./

# Rules
default: all

all: $(TARGET) encoder decoder signalcompare

lib: $(TARGET)

$(TARGET): $(OBJS)
        $(ARCHIVE.cmdline)

encoder$(EXESUFFIX): $(ENCODER_OBJS)
        $(LINK.o.cmdline)

decoder$(EXESUFFIX): $(DECODER_OBJS)
```

```
        $(LINK.o.cmdline)

signalcompare$(EXESUFFIX): $(SIGNALCMP_OBJS)
        $(LINK.o.cmdline)

clean:
        $(RM) $(TARGET)* $(OBJS) $(ENCODER_OBJS) $(DECODER_OBJS) \
                  $(SIGNALCMP_OBJS) $(TEST_OBJS) \
                  encoder$(EXESUFFIX) decoder$(EXESUFFIX) signalcompare$(EXESUFFI
X)
```

A.2. interface/SKP_Silk_control.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_CONTROL_H
#define SKP_SILK_CONTROL_H

#include "SKP_Silk_typedef.h"

#ifdef __cplusplus
```

```
extern "C"
{
#endif

/****************************************/
/* Structure for controlling encoder operation */
/****************************************/
typedef struct {
    /* I:   Sampling rate in Hertz; 8000/12000/16000/24000
           */
    SKP_int32 sampleRate;

    /* I:   Number of samples per packet; must be equivalent of 20, 40, 60, 80 or
 100 ms    */
    SKP_int packetSize;

    /* I:   Bitrate during active speech in bits/second; internally limited
           */
    SKP_int32 bitRate;

    /* I:   Uplink Packet loss in pct (0...100)
           */
    SKP_int packetLossPercentage;

    /* I:   Complexity mode; 0 is lowest; 1 is medium and 2 is highest complexity
           */
    SKP_int complexity;

    /* I:   Flag to enable in-band Forward Error Correction (FEC); 0/1
           */
    SKP_int useInBandFEC;

    /* I:   Flag to enable Discontinous Transmission; 0/1
           */
    SKP_int useDTX;
} SKP_SILK_SDK_EncControlStruct;

/****************************************************************************/
/* Structure for controlling decoder operation and reading decoder status */
/****************************************************************************/
typedef struct {
    /* I:   Sampling rate in Hertz; 8000/12000/16000/24000
           */
    SKP_int32 sampleRate;

    /* O:   Number of samples per frame
           */
    SKP_int frameSize;

    /* O:   Frames per packet 1, 2, 3, 4, 5
           */
    SKP_int framesPerPacket;

    /* O:   Flag to indicate that the decoder has remaining payloads internally
           */
    SKP_int moreInternalDecoderFrames;

    /* O:   Distance between main payload and redundant payload in packets
           */
    SKP_int inBandFECOffset;
```

```
} SKP_SILK_SDK_DecControlStruct;

#ifdef __cplusplus
}
#endif

#endif
```

A.3. interface/SKP_Silk_errors.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_ERRORS_H
#define SKP_SILK_ERRORS_H

#ifdef __cplusplus
extern "C"
{
#endif

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

```
/* Error messages */
/******************/
#define SKP_SILK_NO_ERROR                                   0

/**************************/
/* Encoder error messages */
/**************************/

/* Input length is not a multiplum of 10 ms,
   or length is longer than the packet length */
#define SKP_SILK_ENC_INPUT_INVALID_NO_OF_SAMPLES           -1

/* Sampling frequency not 8000, 12000, 16000
   or 24000 Hertz */
#define SKP_SILK_ENC_FS_NOT_SUPPORTED                      -2

/* Packet size not 20, 40, 60, 80 or 100 ms */
#define SKP_SILK_ENC_PACKET_SIZE_NOT_SUPPORTED             -3

/* Allocated payload buffer too short */
#define SKP_SILK_ENC_PAYLOAD_BUF_TOO_SHORT                 -4

/* Loss rate not between 0 and 100 percent */
#define SKP_SILK_ENC_WRONG_LOSS_RATE                       -5

/* Complexity setting not valid, use 0, 1 or 2 */
#define SKP_SILK_ENC_WRONG_COMPLEXITY_SETTING              -6

/* Inband FEC setting not valid, use 0 or 1 */
#define SKP_SILK_ENC_WRONG_INBAND_FEC_SETTING              -7

/* DTX setting not valid, use 0 or 1 */
#define SKP_SILK_ENC_WRONG_DTX_SETTING                     -8

/* Internal encoder error */
#define SKP_SILK_ENC_INTERNAL_ERROR                        -9

/**************************/
/* Decoder error messages */
/**************************/

/* Output sampling frequency lower than internal
   decoded sampling frequency */
#define SKP_SILK_DEC_WRONG_SAMPLING_FREQUENCY             -10

/* Payload size exceeded the maximum allowed 1024 bytes */
#define SKP_SILK_DEC_PAYLOAD_TOO_LARGE                    -11
```

```
/* Payload has bit errors */
#define SKP_SILK_DEC_PAYLOAD_ERROR                      -12


#ifdef __cplusplus
}
#endif

#endif
```

A.4. interface/SKP_Silk_SDK_API.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_SDK_API_H
#define SKP_SILK_SDK_API_H

#include "SKP_Silk_control.h"
#include "SKP_Silk_typedef.h"
#include "SKP_Silk_errors.h"
```

```
#ifdef __cplusplus
extern "C"
{
#endif

#define SILK_MAX_FRAMES_PER_PACKET  5

/* Struct for TOC (Table of Contents) */
typedef struct {
    SKP_int     framesInPacket;                             /* Number of 20 ms fr
ames in packet     */
    SKP_int     fs_kHz;                                     /* Sampling frequency
 in packet         */
    SKP_int     inbandLBRR;                                 /* Does packet contai
n LBRR information */
    SKP_int     corrupt;                                    /* Packet is corrupt
                   */
    SKP_int     vadFlags[     SILK_MAX_FRAMES_PER_PACKET ]; /* VAD flag for each
frame in packet    */
    SKP_int     sigtypeFlags[ SILK_MAX_FRAMES_PER_PACKET ]; /* Signal type for ea
ch frame in packet */
} SKP_Silk_TOC_struct;

/***************************************/
/* Encoder functions                   */
/***************************************/

/************************************************/
/* Get size in bytes of the Silk encoder state */
/************************************************/
SKP_int SKP_Silk_SDK_Get_Encoder_Size(
    SKP_int32                           *encSizeBytes   /* O:   Number of bytes i
n SILK encoder state           */
);

/*************************/
/* Init or reset encoder */
/*************************/
SKP_int SKP_Silk_SDK_InitEncoder(
    void                                *encState,      /* I/O: State
                                */
    SKP_SILK_SDK_EncControlStruct       *encStatus      /* O:   Encoder Status
                                */
);

/*************************************/
/* Read control structure from encoder */
/*************************************/
SKP_int SKP_Silk_SDK_QueryEncoder(
    const void                          *encState,      /* I:   State
                                */
    SKP_SILK_SDK_EncControlStruct       *encStatus      /* O:   Encoder Status
                                */
);

/*************************/
/* Encode frame with Silk */
/*************************/
SKP_int SKP_Silk_SDK_Encode(
```

```
    void                                *encState,      /* I/O: State
                                */
    const SKP_SILK_SDK_EncControlStruct *encControl,    /* I:   Control status
                                */
    const SKP_int16                     *samplesIn,     /* I:   Speech sample inp
ut vector                       */
    SKP_int                             nSamplesIn,     /* I:   Number of samples
 in input vector                */
    SKP_uint8                           *outData,       /* O:   Encoded output ve
ctor                            */
    SKP_int16                           *nBytesOut      /* I/O: Number of Bytes i
n outData (input: Max Bytes)    */
);

/****************************************/
/* Decoder functions                    */
/****************************************/

/************************************************/
/* Get size in bytes of the Silk decoder state */
/************************************************/
SKP_int SKP_Silk_SDK_Get_Decoder_Size(
    SKP_int32                           *decSizeBytes   /* O:   Number of bytes i
n SILK decoder state            */
);

/*************************/
/* Init or Reset decoder */
/*************************/
SKP_int SKP_Silk_SDK_InitDecoder(
    void                                *decState       /* I/O: State
                                */
);

/******************/
/* Decode a frame */
/******************/
SKP_int SKP_Silk_SDK_Decode(
    void*                               decState,       /* I/O: State
                                */
    SKP_SILK_SDK_DecControlStruct*      decControl,     /* I/O: Control Structure
                                */
    SKP_int                             lostFlag,       /* I:   0: no loss, 1 los
s                               */
    const SKP_uint8                     *inData,        /* I:   Encoded input vec
tor                             */
    const SKP_int                       nBytesIn,       /* I:   Number of input B
ytes                            */
    SKP_int16                           *samplesOut,    /* O:   Decoded output sp
eech vector                     */
    SKP_int16                           *nSamplesOut    /* I/O: Number of samples
 (vector/decoded)               */
);

/***************************************************************/
/* Find Low Bit Rate Redundancy (LBRR) information in a packet */
/***************************************************************/
void SKP_Silk_SDK_search_for_LBRR(
    void                                *decState,      /* I:   Decoder state, to
 select bitstream version only */
    const SKP_uint8                     *inData,        /* I:   Encoded input vec
tor                             */
```

```
    const SKP_int16                 nBytesIn,       /* I:   Number of input B
ytes                                */
    SKP_int                         lost_offset,    /* I:   Offset from lost
packet                              */
    SKP_uint8                       *LBRRData,      /* O:   LBRR payload
                                    */
```

```
    SKP_int16                       *nLBRRBytes     /* O:   Number of LBRR By
tes                             */
);

/************************************/
/* Get type of content for a packet */
/************************************/
void SKP_Silk_SDK_get_TOC(
    void                            *decState,      /* I:   Decoder state, to
 select bitstream version only */
    const SKP_uint8                 *inData,        /* I:   Encoded input vec
tor                             */
    const SKP_int16                 nBytesIn,       /* I:   Number of input b
ytes                            */
    SKP_Silk_TOC_struct             *Silk_TOC       /* O:   Type of content
                                */
);

/**************************/
/* Get the version number */
/**************************/
/* Return a pointer to string specifying the version */
const char *SKP_Silk_SDK_get_version();

#ifdef __cplusplus
}
#endif

#endif
```

A.5. interface/SKP_Silk_typedef.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
```

```
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef _SKP_SILK_API_TYPDEF_H_
#define _SKP_SILK_API_TYPDEF_H_

#ifndef SKP_USE_DOUBLE_PRECISION_FLOATS
#define SKP_USE_DOUBLE_PRECISION_FLOATS     0
#endif

#include <float.h>
#if defined( __GNUC__ )
#include <stdint.h>
#endif

#define SKP_int         int                     /* used for counters etc; at leas
t 16 bits */
#define SKP_int64       long long
#define SKP_int32       int
#define SKP_int16       short
#define SKP_int8        signed char

#define SKP_uint        unsigned int            /* used for counters etc; at leas
t 16 bits */
#define SKP_uint64      unsigned long long
#define SKP_uint32      unsigned int
#define SKP_uint16      unsigned short
#define SKP_uint8       unsigned char

#define SKP_int_ptr_size intptr_t

#if SKP_USE_DOUBLE_PRECISION_FLOATS
# define SKP_float      double
# define SKP_float_MAX  DBL_MAX
#else
# define SKP_float      float
# define SKP_float_MAX  FLT_MAX
#endif

#define SKP_INLINE      static __inline

#ifdef _WIN32
# define SKP_STR_CASEINSENSITIVE_COMPARE(x, y) _stricmp(x, y)
#else
# define SKP_STR_CASEINSENSITIVE_COMPARE(x, y) strcasecmp(x, y)
```

```
#endif

#define SKP_int64_MAX   ((SKP_int64)0x7FFFFFFFFFFFFFFFLL)   //  2^63 - 1
#define SKP_int64_MIN   ((SKP_int64)0x8000000000000000LL)   // -2^63
#define SKP_int32_MAX   0x7FFFFFFF                          //  2^31 - 1 =  21474
83647
#define SKP_int32_MIN   ((SKP_int32)0x80000000)             // -2^31     = -21474
83648
#define SKP_int16_MAX   0x7FFF                              //  2^15 - 1 =  32767
#define SKP_int16_MIN   ((SKP_int16)0x8000)                 // -2^15     = -32768
#define SKP_int8_MAX    0x7F                                //  2^7 - 1  =  127
#define SKP_int8_MIN    ((SKP_int8)0x80)                    // -2^7      = -128

#define SKP_uint32_MAX  0xFFFFFFFF  // 2^32 - 1 = 4294967295
#define SKP_uint32_MIN  0x00000000
#define SKP_uint16_MAX  0xFFFF      // 2^16 - 1 = 65535
#define SKP_uint16_MIN  0x0000
#define SKP_uint8_MAX   0xFF        //  2^8 - 1 = 255
#define SKP_uint8_MIN   0x00

#define SKP_TRUE        1
#define SKP_FALSE       0

/* assertions */
#if (defined _WIN32 && !defined _WINCE && !defined(__GNUC__) && !defined(NO_ASSER
TS))
# ifndef SKP_assert
#  include <crtdbg.h>      /* ASSERTE() */
#  define SKP_assert(COND)   _ASSERTE(COND)
# endif
#else
# define SKP_assert(COND)
#endif

#endif
```

A.6. src/SKP_Silk_A2NLSF.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
```

```
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/* Conversion between prediction filter coefficients and NLSFs  */
/* Requires the order to be an even number                      */
/* A piecewise linear approximation maps LSF <-> cos(LSF)       */
/* Therefore the result is not accurate NLSFs, but the two      */
/* function are accurate inverses of each other                 */

#include "SKP_Silk_SigProc_FIX.h"

/* Number of binary divisions, when not in low complexity mode */
#define BIN_DIV_STEPS_A2NLSF_FIX      2 /* must be no higher than 16 - log2( LSF_
COS_TAB_SZ_FIX ) */
#define QPoly                         16
#define MAX_ITERATIONS_A2NLSF_FIX     50

/* Flag for using 2x as many cosine sampling points, reduces the risk of missing
a root */
#define OVERSAMPLE_COSINE_TABLE       0

/* Helper function for A2NLSF(..)                    */
/* Transforms polynomials from cos(n*f) to cos(f)^n  */
SKP_INLINE void SKP_Silk_A2NLSF_trans_poly(
    SKP_int32        *p,    /* I/O    Polynomial                                *
/
    const SKP_int    dd     /* I      Polynomial order (= filter order / 2 )    *
/
)
{
    SKP_int k, n;

    for( k = 2; k <= dd; k++ ) {
        for( n = dd; n > k; n-- ) {
            p[ n - 2 ] -= p[ n ];
        }
        p[ k - 2 ] -= SKP_LSHIFT( p[ k ], 1 );
    }
}
```

```
/* Helper function for A2NLSF(..)                    */
/* Polynomial evaluation                             */
SKP_INLINE SKP_int32 SKP_Silk_A2NLSF_eval_poly(    /* return the polynomial evalu
ation, in QPoly */
    SKP_int32       *p,    /* I    Polynomial, QPoly        */
    const SKP_int32   x,   /* I    Evaluation point, Q12    */
    const SKP_int    dd    /* I    Order                    */
)
{
    SKP_int   n;
    SKP_int32 x_Q16, y32;

    y32 = p[ dd ];                                    /* QPoly */
    x_Q16 = SKP_LSHIFT( x, 4 );
    for( n = dd - 1; n >= 0; n-- ) {
        y32 = SKP_SMLAWW( p[ n ], y32, x_Q16 );       /* QPoly */
    }
    return y32;
}

SKP_INLINE void SKP_Silk_A2NLSF_init(
     const SKP_int32    *a_Q16,
     SKP_int32           *P,
     SKP_int32           *Q,
     const SKP_int       dd
)
{
    SKP_int k;

    /* Convert filter coefs to even and odd polynomials */
    P[dd] = SKP_LSHIFT( 1, QPoly );
    Q[dd] = SKP_LSHIFT( 1, QPoly );
    for( k = 0; k < dd; k++ ) {
#if( QPoly < 16 )
        P[ k ] = SKP_RSHIFT_ROUND( -a_Q16[ dd - k - 1 ] - a_Q16[ dd + k ], 16 - Q
Poly ); /* QPoly */
        Q[ k ] = SKP_RSHIFT_ROUND( -a_Q16[ dd - k - 1 ] + a_Q16[ dd + k ], 16 - Q
Poly ); /* QPoly */
#elif( Qpoly == 16 )
        P[ k ] = -a_Q16[ dd - k - 1 ] - a_Q16[ dd + k ]; // QPoly
        Q[ k ] = -a_Q16[ dd - k - 1 ] + a_Q16[ dd + k ]; // QPoly
#else
        P[ k ] = SKP_LSHIFT( -a_Q16[ dd - k - 1 ] - a_Q16[ dd + k ], QPoly - 16 )
; /* QPoly */
        Q[ k ] = SKP_LSHIFT( -a_Q16[ dd - k - 1 ] + a_Q16[ dd + k ], QPoly - 16 )
; /* QPoly */
#endif
    }

    /* Divide out zeros as we have that for even filter orders, */
    /* z =  1 is always a root in Q, and                        */
    /* z = -1 is always a root in P                             */
    for( k = dd; k > 0; k-- ) {
```

```
        P[ k - 1 ] -= P[ k ];
        Q[ k - 1 ] += Q[ k ];
    }

    /* Transform polynomials from cos(n*f) to cos(f)^n */
    SKP_Silk_A2NLSF_trans_poly( P, dd );
    SKP_Silk_A2NLSF_trans_poly( Q, dd );
}

/* Compute Normalized Line Spectral Frequencies (NLSFs) from whitening filter coe
fficients        */
/* If not all roots are found, the a_Q16 coefficients are bandwidth expanded unti
l convergence.    */
void SKP_Silk_A2NLSF(
    SKP_int          *NLSF,                 /* O    Normalized Line Spectral Freq
uencies, Q15 (0 - (2^15-1)), [d]    */
    SKP_int32        *a_Q16,                /* I/O  Monic whitening filter coeffi
cients in Q16 [d]                   */
    const SKP_int    d                      /* I    Filter order (must be even)
                                    */
)
{
    SKP_int      i, k, m, dd, root_ix, ffrac;
    SKP_int32 xlo, xhi, xmid;
    SKP_int32 ylo, yhi, ymid;
    SKP_int32 nom, den;
    SKP_int32 P[ SigProc_MAX_ORDER_LPC / 2 + 1 ];
    SKP_int32 Q[ SigProc_MAX_ORDER_LPC / 2 + 1 ];
    SKP_int32 *PQ[ 2 ];
    SKP_int32 *p;

    /* Store pointers to array */
    PQ[ 0 ] = P;
    PQ[ 1 ] = Q;

    dd = SKP_RSHIFT( d, 1 );

    SKP_Silk_A2NLSF_init( a_Q16, P, Q, dd );

    /* Find roots, alternating between P and Q */
    p = P;    /* Pointer to polynomial */

    xlo = SKP_Silk_LSFCosTab_FIX_Q12[ 0 ]; // Q12
    ylo = SKP_Silk_A2NLSF_eval_poly( p, xlo, dd );

    if( ylo < 0 ) {
        /* Set the first NLSF to zero and move on to the next */
        NLSF[ 0 ] = 0;
        p = Q;                      /* Pointer to polynomial */
        ylo = SKP_Silk_A2NLSF_eval_poly( p, xlo, dd );
        root_ix = 1;                /* Index of current root */
    } else {
        root_ix = 0;                /* Index of current root */
```

```
    }
    k = 1;                          /* Loop counter */
    i = 0;                          /* Counter for bandwidth expansions applied *
/
    while( 1 ) {
        /* Evaluate polynomial */
#if OVERSAMPLE_COSINE_TABLE
        xhi = SKP_Silk_LSFCosTab_FIX_Q12[   k       >> 1 ] +
          ( ( SKP_Silk_LSFCosTab_FIX_Q12[ ( k + 1 ) >> 1 ] -
              SKP_Silk_LSFCosTab_FIX_Q12[   k       >> 1 ] ) >> 1 );    /* Q12 */
#else
        xhi = SKP_Silk_LSFCosTab_FIX_Q12[ k ]; /* Q12 */
#endif
        yhi = SKP_Silk_A2NLSF_eval_poly( p, xhi, dd );

        /* Detect zero crossing */
        if( ( ylo <= 0 && yhi >= 0 ) || ( ylo >= 0 && yhi <= 0 ) ) {
            /* Binary division */
#if OVERSAMPLE_COSINE_TABLE
            ffrac = -128;
#else
            ffrac = -256;
#endif
            for( m = 0; m < BIN_DIV_STEPS_A2NLSF_FIX; m++ ) {
                /* Evaluate polynomial */
                xmid = SKP_RSHIFT_ROUND( xlo + xhi, 1 );
                ymid = SKP_Silk_A2NLSF_eval_poly( p, xmid, dd );

                /* Detect zero crossing */
                if( ( ylo <= 0 && ymid >= 0 ) || ( ylo >= 0 && ymid <= 0 ) ) {
                    /* Reduce frequency */
                    xhi = xmid;
                    yhi = ymid;
                } else {
                    /* Increase frequency */
                    xlo = xmid;
                    ylo = ymid;
#if OVERSAMPLE_COSINE_TABLE
                    ffrac = SKP_ADD_RSHIFT( ffrac,  64, m );
#else
                    ffrac = SKP_ADD_RSHIFT( ffrac, 128, m );
#endif
                }
            }

            /* Interpolate */
            if( SKP_abs( ylo ) < 65536 ) {
                /* Avoid dividing by zero */
                den = ylo - yhi;
```

```
                nom = SKP_LSHIFT( ylo, 8 - BIN_DIV_STEPS_A2NLSF_FIX ) + SKP_RSHIF
T( den, 1 );
                if( den != 0 ) {
                    ffrac += SKP_DIV32( nom, den );
                }
            } else {
                /* No risk of dividing by zero because abs(ylo - yhi) >= abs(ylo)
 >= 65536 */
                ffrac += SKP_DIV32( ylo, SKP_RSHIFT( ylo - yhi, 8 - BIN_DIV_STEPS
_A2NLSF_FIX ) );
            }
#if OVERSAMPLE_COSINE_TABLE
            NLSF[ root_ix ] = (SKP_int)SKP_min_32( SKP_LSHIFT( (SKP_int32)k, 7 )
+ ffrac, SKP_int16_MAX );
#else
            NLSF[ root_ix ] = (SKP_int)SKP_min_32( SKP_LSHIFT( (SKP_int32)k, 8 )
+ ffrac, SKP_int16_MAX );
#endif

            SKP_assert( NLSF[ root_ix ] >=     0 );
            SKP_assert( NLSF[ root_ix ] <= 32767 );

            root_ix++;        /* Next root */
            if( root_ix >= d ) {
                /* Found all roots */
                break;
            }
            /* Alternate pointer to polynomial */
            p = PQ[ root_ix & 1 ];

            /* Evaluate polynomial */
#if OVERSAMPLE_COSINE_TABLE
            xlo = SKP_Silk_LSFCosTab_FIX_Q12[ ( k - 1 ) >> 1 ] +
              ( ( SKP_Silk_LSFCosTab_FIX_Q12[   k       >> 1 ] -
                  SKP_Silk_LSFCosTab_FIX_Q12[ ( k - 1 ) >> 1 ] ) >> 1 ); // Q12
#else
            xlo = SKP_Silk_LSFCosTab_FIX_Q12[ k - 1 ]; // Q12
#endif
            ylo = SKP_LSHIFT( 1 - ( root_ix & 2 ), 12 );
        } else {
            /* Increment loop counter */
            k++;
            xlo    = xhi;
            ylo    = yhi;

#if OVERSAMPLE_COSINE_TABLE
            if( k > 2 * LSF_COS_TAB_SZ_FIX ) {
#else
            if( k > LSF_COS_TAB_SZ_FIX ) {
#endif
                i++;
                if( i > MAX_ITERATIONS_A2NLSF_FIX ) {
                    /* Set NLSFs to white spectrum and exit */
```

```
                    NLSF[ 0 ] = SKP_DIV32_16( 1 << 15, d + 1 );
                    for( k = 1; k < d; k++ ) {
                        NLSF[ k ] = SKP_SMULBB( k + 1, NLSF[ 0 ] );
                    }
                    return;
                }

                /* Error: Apply progressively more bandwidth expansion and run ag
ain */
                SKP_Silk_bwexpander_32( a_Q16, d, 65536 - SKP_SMULBB( 66, i ) );
// 66_Q16 = 0.001

                SKP_Silk_A2NLSF_init( a_Q16, P, Q, dd );
                p = P;                              /* Pointer to polynomial */
                xlo = SKP_Silk_LSFCosTab_FIX_Q12[ 0 ]; // Q12
                ylo = SKP_Silk_A2NLSF_eval_poly( p, xlo, dd );
                if( ylo < 0 ) {
                    /* Set the first NLSF to zero and move on to the next */
                    NLSF[ 0 ] = 0;
                    p = Q;                          /* Pointer to polynomial */
                    ylo = SKP_Silk_A2NLSF_eval_poly( p, xlo, dd );
                    root_ix = 1;                /* Index of current root */
                } else {
                    root_ix = 0;                /* Index of current root */
                }
                k = 1;                            /* Reset loop counter */
            }
        }
    }
}
```

A.7. src/SKP_Silk_allpass_int.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
```

```
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_allpass_int.c                                             *
 *                                                                     *
 * First-order allpass filter with                                     *
 * transfer function:                                                  *
 *                                                                     *
 *         A + Z^(-1)                                                  *
 * H(z) = ------------                                                 *
 *        1 + A*Z^(-1)                                                 *
 *                                                                     *
 * Implemented using minimum multiplier filter design.                 *
 *                                                                     *
 * Reference: http://www.univ.trieste.it/~ramponi/teaching/            *
 * DSP/materiale/Ch6(2).pdf                                            *
 *                                                                     *
 * Copyright 2007 (c), Skype Limited                                   *
 * Date: 070525                                                        *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"


/* First-order allpass filter */
void SKP_Silk_allpass_int(
    const SKP_int32      *in,    /* I:    Q25 input signal [len]               */
    SKP_int32            *S,     /* I/O: Q25 state [1]                         */
    SKP_int              A,      /* I:    Q15 coefficient    (0 <= A < 32768)  */
    SKP_int32            *out,   /* O:    Q25 output signal [len]              */
    const SKP_int32      len     /* I:    Number of samples                    */
)
{
    SKP_int32    Y2, X2, S0;
    SKP_int        k;

    S0 = S[ 0 ];
    for( k = len - 1; k >= 0; k-- ) {
        Y2         = *in - S0;
```

```
        X2          = ( Y2 >> 15 ) * A + ( ( ( Y2 & 0x00007FFF ) * A ) >> 15 );
        ( *out++ ) = S0 + X2;
        S0          = ( *in++ ) + X2;
    }
    S[ 0 ] = S0;
}
```

A.8. src/SKP_Silk_ana_filt_bank_1.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_ana_filt_bank_1.c                                                *
 *                                                                      *
 * Split signal into two decimated bands using first-order allpass filters  *
 *                                                                      *
 * Copyright 2006 (c), Skype Limited                                    *
 * Date: 060221                                                         *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"
```

```
/* Coefficients for 2-band filter bank based on first-order allpass filters */
static SKP_int16 A_fb1_20[ 1 ] = {  5394 };
static SKP_int16 A_fb1_21[ 1 ] = { 20623 };

/* Split signal into two decimated bands using first-order allpass filters */
void SKP_Silk_ana_filt_bank_1(
    const SKP_int16      *in,        /* I:    Input signal [N]       */
    SKP_int32            *S,         /* I/O: State vector [2]        */
    SKP_int16            *outL,      /* O:    Low band [N/2]         */
    SKP_int16            *outH,      /* O:    High band [N/2]        */
    SKP_int32            *scratch,   /* I:    Scratch memory [3*N/2] */
    const SKP_int32      N           /* I:   Number of input samples */
)
{
    SKP_int        k, N2 = SKP_RSHIFT( N, 1 );
    SKP_int32    out_tmp;

    /* De-interleave three allpass inputs, and convert Q15 -> Q25 */
    for( k = 0; k < N2; k++ ) {
        scratch[ k + N  ] = SKP_LSHIFT( (SKP_int32)in[ 2 * k     ], 10 );
        scratch[ k + N2 ] = SKP_LSHIFT( (SKP_int32)in[ 2 * k + 1 ], 10 );
    }

    /* Allpass filters */
    SKP_Silk_allpass_int( scratch + N2, S+0, A_fb1_20[ 0 ], scratch,      N2 );
    SKP_Silk_allpass_int( scratch + N,  S+1, A_fb1_21[ 0 ], scratch + N2, N2 );

    /* Add and subtract two allpass outputs to create bands */
    for( k = 0; k < N2; k++ ) {
        out_tmp   = scratch[ k ] + scratch[ k + N2 ];
        outL[ k ] = (SKP_int16)SKP_SAT16( SKP_RSHIFT_ROUND( out_tmp, 11 ) );

        out_tmp   = scratch[ k ] - scratch[ k + N2 ];
        outH[ k ] = (SKP_int16)SKP_SAT16( SKP_RSHIFT_ROUND( out_tmp, 11 ) );
    }
}
```

A.9. src/SKP_Silk_apply_sine_window.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
```

```
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_SigProc_FIX.h"

/* Apply sine window to signal vector.                                  */
/* Window types:                                                        */
/*    0 -> sine window from 0 to pi                                     */
/*    1 -> sine window from 0 to pi/2                                   */
/*    2 -> sine window from pi/2 to pi                                  */
/* every other sample of window is linearly interpolated, for speed     */
void SKP_Silk_apply_sine_window(
    SKP_int16                       px_win[],           /* O    Pointer to wind
owed signal                  */
    const SKP_int16                 px[],               /* I    Pointer to inpu
t signal                     */
    const SKP_int                   win_type,           /* I    Selects a windo
w type                       */
    const SKP_int                   length              /* I    Window length,
multiple of 4                */
)
{
    SKP_int   k;
    SKP_int32 px32, f_Q16, c_Q20, S0_Q16, S1_Q16;

    /* Length must be multiple of 4 */
    SKP_assert( ( length & 3 ) == 0 );

    /* Input pointer must be 4-byte aligned */
    SKP_assert( ( (SKP_int64)px & 3 ) == 0 );

    if( win_type == 0 ) {
        f_Q16 = SKP_DIV32_16( 411775, length + 1 );        // 411775 = 2 * 65536
* pi
    } else {
        f_Q16 = SKP_DIV32_16( 205887, length + 1 );        // 205887 = 65536 * pi
```

```
    }

    /* factor used for cosine approximation */
    c_Q20 = -SKP_RSHIFT( SKP_MUL( f_Q16, f_Q16 ), 12 );

    /* c_Q20 becomes too large if length is too small */
    SKP_assert( c_Q20 >= -32768 );

    /* initialize state */
    if( win_type < 2 ) {
        /* start from 0 */
        S0_Q16 = 0;
        /* approximation of sin(f) */
        S1_Q16 = f_Q16;
    } else {
        /* start from 1 */
        S0_Q16 = ( 1 << 16 );
        /* approximation of cos(f) */
        S1_Q16 = ( 1 << 16 ) + SKP_RSHIFT( c_Q20, 5 );
    }

    /* Uses the recursive equation:   sin(n*f) = 2 * cos(f) * sin((n-1)*f) - sin(
(n-2)*f)    */
    /* 4 samples at a time */
    for( k = 0; k < length; k += 4 ) {
        px32 = *( (SKP_int32 *)&px[ k ] );                      /* load two val
ues at once */
        px_win[ k ]     = (SKP_int16)SKP_SMULWB( SKP_RSHIFT( S0_Q16 + S1_Q16, 1 )
, px32 );
        px_win[ k + 1 ] = (SKP_int16)SKP_SMULWT( S1_Q16, px32 );
        S0_Q16 = SKP_RSHIFT( SKP_MUL( c_Q20, S1_Q16 ), 20 ) + SKP_LSHIFT( S1_Q16,
 1 ) - S0_Q16 + 1;
        S0_Q16 = SKP_min( S0_Q16, ( 1 << 16 ) );

        px32 = *( (SKP_int32 *)&px[k + 2] );                    /* load two value
s at once */
        px_win[ k + 2 ] = (SKP_int16)SKP_SMULWB( SKP_RSHIFT( S0_Q16 + S1_Q16, 1 )
, px32 );
        px_win[ k + 3 ] = (SKP_int16)SKP_SMULWT( S0_Q16, px32 );
        S1_Q16 = SKP_RSHIFT( SKP_MUL( c_Q20, S0_Q16 ), 20 ) + SKP_LSHIFT( S0_Q16,
 1 ) - S1_Q16;
        S1_Q16 = SKP_min( S1_Q16, ( 1 << 16 ) );
    }
}
```

A.10. src/SKP_Silk_array_maxabs.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
```

```
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_int16_array_maxabs.c                                     *
 *                                                                     *
 * Function that returns the maximum absolut value of                 *
 * the input vector                                                    *
 *                                                                     *
 * Copyright 2006 (c), Skype Limited                                   *
 * Date: 060221                                                        *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"

/* Function that returns the maximum absolut value of the input vector */
SKP_int16 SKP_Silk_int16_array_maxabs(    /* O    Maximum absolute value, max: 2^
15-1   */
    const SKP_int16        *vec,            /* I    Input vector  [len]
         */
    const SKP_int32        len              /* I    Length of input vector
         */
)
{
    SKP_int32 max = 0, i, lvl = 0, ind;

    ind = len - 1;
    max = SKP_SMULBB( vec[ ind ], vec[ ind ] );
    for( i = len - 2; i >= 0; i-- ) {
        lvl = SKP_SMULBB( vec[ i ], vec[ i ] );
        if( lvl > max ) {
            max = lvl;
            ind = i;
        }
```

```
    }

    /* Do not return 32768, as it will not fit in an int16 so may lead to problem
s later on */
    lvl = SKP_abs( vec[ ind ] );
    if( lvl > SKP_int16_MAX ) {
        return( SKP_int16_MAX );
    } else {
        return( (SKP_int16)lvl );
    }
}
```

A.11. src/SKP_Silk_autocorr.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_Silk_autocorr.c                                                  *
 *                                                                      *
 * Calculates the autocorrelation                                       *
 * The result has 29 non-zero bits for the first correlation, to leave  *
 * some    room for adding white noise fractions etc.                   *
```

```
 *                                                                     *
 * Copyright 2008 (c), Skype Limited                                   *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"

/* Compute autocorrelation */
void SKP_Silk_autocorr(
    SKP_int32        *results,                  /* O    Result (length correlati
onCount)            */
    SKP_int32        *scale,                    /* O    Scaling of the correlati
on vector           */
    const SKP_int16  *inputData,                /* I    Input data to correlate
                    */
    const SKP_int    inputDataSize,             /* I    Length of input
                    */
    const SKP_int    correlationCount           /* I    Number of correlation ta
ps to compute       */
)
{
    SKP_int   i, lz, nRightShifts, corrCount;
    SKP_int64 corr64;

    corrCount = SKP_min_int( inputDataSize, correlationCount );

    /* compute energy (zero-lag correlation) */
    corr64 = SKP_Silk_inner_prod16_aligned_64( inputData, inputData, inputDataSiz
e );

    /* deal with all-zero input data */
    corr64 += 1;

    /* number of leading zeros */
    lz = SKP_Silk_CLZ64( corr64 );

    /* scaling: number of right shifts applied to correlations */
    nRightShifts = 35 - lz;
    *scale = nRightShifts;

    if( nRightShifts <= 0 ) {
        results[ 0 ] = SKP_LSHIFT( (SKP_int32)SKP_CHECK_FIT32( corr64 ), -nRightS
hifts );

        /* compute remaining correlations based on int32 inner product */
          for( i = 1; i < corrCount; i++ ) {
            results[ i ] = SKP_LSHIFT( SKP_Silk_inner_prod_aligned( inputData, in
putData + i, inputDataSize - i ), -nRightShifts );
        }
    } else {
        results[ 0 ] = (SKP_int32)SKP_CHECK_FIT32( SKP_RSHIFT64( corr64, nRightSh
ifts ) );

        /* compute remaining correlations based on int64 inner product */
          for( i = 1; i < corrCount; i++ ) {
            results[ i ] =  (SKP_int32)SKP_CHECK_FIT32( SKP_RSHIFT64( SKP_Silk_in
ner_prod16_aligned_64( inputData, inputData + i, inputDataSize - i ), nRightShift
s ) );
        }
    }
}
```

A.12. src/SKP_Silk_biquad.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_Silk_biquad.c                                                  *
 *                                                                      *
 * Second order ARMA filter                                             *
 * Can handle slowly varying filter coefficients                        *
 *                                                                      *
 * Copyright 2006 (c), Skype Limited                                    *
 * Date: 060221                                                         *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"

/* Second order ARMA filter */
/* Can handle slowly varying filter coefficients */
void SKP_Silk_biquad(
    const SKP_int16      *in,        /* I:    input signal               */
    const SKP_int16      *B,         /* I:    MA coefficients, Q13 [3]   */
    const SKP_int16      *A,         /* I:    AR coefficients, Q13 [2]   */
    SKP_int32            *S,         /* I/O: state vector [2]            */
```

```
    SKP_int16           *out,       /* O:    output signal               */
    const SKP_int32     len         /* I:    signal length               */
)
{
    SKP_int   k, in16;
    SKP_int32 A0_neg, A1_neg, S0, S1, out32, tmp32;

    S0 = S[ 0 ];
    S1 = S[ 1 ];
    A0_neg = -A[ 0 ];
    A1_neg = -A[ 1 ];
    for( k = 0; k < len; k++ ) {
        /* S[ 0 ], S[ 1 ]: Q13 */
        in16  = in[ k ];
        out32 = SKP_SMLABB( S0, in16, B[ 0 ] );

        S0 = SKP_SMLABB( S1, in16, B[ 1 ] );
        S0 += SKP_LSHIFT( SKP_SMULWB( out32, A0_neg ), 3 );

        S1 = SKP_LSHIFT( SKP_SMULWB( out32, A1_neg ), 3 );
        S1 = SKP_SMLABB( S1, in16, B[ 2 ] );
        tmp32    = SKP_RSHIFT_ROUND( out32, 13 ) + 1;
        out[ k ] = (SKP_int16)SKP_SAT16( tmp32 );
    }
    S[ 0 ] = S0;
    S[ 1 ] = S1;
}
```

A.13. src/SKP_Silk_biquad_alt.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
```

```
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_biquad_alt.c                                              *
 *                                                                     *
 * Second order ARMA filter                                            *
 * Can handle slowly varying filter coefficients                       *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"


/* Second order ARMA filter, alternative implementation */
void SKP_Silk_biquad_alt(
    const SKP_int16      *in,            /* I:    Input signal
*/
    const SKP_int32      *B_Q28,         /* I:    MA coefficients [3]
*/
    const SKP_int32      *A_Q28,         /* I:    AR coefficients [2]
*/
    SKP_int32            *S,             /* I/O: State vector [2]
*/
    SKP_int16            *out,           /* O:    Output signal
*/
    const SKP_int32      len             /* I:    Signal length (must be even)
*/
)
{
    /* DIRECT FORM II TRANSPOSED (uses 2 element state vector) */
    SKP_int   k;
    SKP_int32 inval, A0_U_Q28, A0_L_Q28, A1_U_Q28, A1_L_Q28, out32_Q14;

    /* Negate A_Q28 values and split in two parts */
    A0_L_Q28 = ( -A_Q28[ 0 ] ) & 0x00003FFF;        /* lower part */
    A0_U_Q28 = SKP_RSHIFT( -A_Q28[ 0 ], 14 );       /* upper part */
    A1_L_Q28 = ( -A_Q28[ 1 ] ) & 0x00003FFF;        /* lower part */
    A1_U_Q28 = SKP_RSHIFT( -A_Q28[ 1 ], 14 );       /* upper part */

    for( k = 0; k < len; k++ ) {
        /* S[ 0 ], S[ 1 ]: Q12 */
        inval = in[ k ];
        out32_Q14 = SKP_LSHIFT( SKP_SMLAWB( S[ 0 ], B_Q28[ 0 ], inval ), 2 );

        S[ 0 ] = S[1] + SKP_RSHIFT( SKP_SMULWB( out32_Q14, A0_L_Q28 ), 14 );
        S[ 0 ] = SKP_SMLAWB( S[ 0 ], out32_Q14, A0_U_Q28 );
        S[ 0 ] = SKP_SMLAWB( S[ 0 ], B_Q28[ 1 ], inval);
```

```
        S[ 1 ] = SKP_RSHIFT( SKP_SMULWB( out32_Q14, A1_L_Q28 ), 14 );
        S[ 1 ] = SKP_SMLAWB( S[ 1 ], out32_Q14, A1_U_Q28 );
        S[ 1 ] = SKP_SMLAWB( S[ 1 ], B_Q28[ 2 ], inval );

        /* Scale back to Q0 and saturate */
        out[ k ] = (SKP_int16)SKP_SAT16( SKP_RSHIFT( out32_Q14, 14 ) + 2 );
    }
}
```

A.14. src/SKP_Silk_bwexpander.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_SigProc_FIX.h"

/* Chirp (bandwidth expand) LP AR filter */
void SKP_Silk_bwexpander(
    SKP_int16           *ar,        /* I/O  AR filter to be expanded (without le
ading 1)    */
    const SKP_int       d,          /* I    Length of ar
            */
    SKP_int32           chirp_Q16   /* I    Chirp factor (typically in the range
 0 to 1)    */
)
```

```
{
    SKP_int   i;
    SKP_int32 chirp_minus_one_Q16;

    chirp_minus_one_Q16 = chirp_Q16 - 65536;

    /* NB: Dont use SKP_SMULWB, instead of SKP_RSHIFT_ROUND( SKP_MUL() , 16 ), be
low. */
    /* Bias in SKP_SMULWB can lead to unstable filters
     */
    for( i = 0; i < d - 1; i++ ) {
        ar[ i ]    = (SKP_int16)SKP_RSHIFT_ROUND( SKP_MUL( chirp_Q16, ar[ i ]
         ), 16 );
        chirp_Q16 +=            SKP_RSHIFT_ROUND( SKP_MUL( chirp_Q16, chirp_minus
_one_Q16 ), 16 );
    }
    ar[ d - 1 ] = (SKP_int16)SKP_RSHIFT_ROUND( SKP_MUL( chirp_Q16, ar[ d - 1 ] ),
 16 );
}
```

A.15. src/SKP_Silk_bwexpander_32.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_SigProc_FIX.h"

/* Chirp (bandwidth expand) LP AR filter */
void SKP_Silk_bwexpander_32(
    SKP_int32        *ar,      /* I/O    AR filter to be expanded (without leadin
g 1)    */
    const SKP_int    d,        /* I    Length of ar
        */
    SKP_int32        chirp_Q16 /* I    Chirp factor in Q16
        */
)
{
    SKP_int   i;
    SKP_int32 tmp_chirp_Q16;

    tmp_chirp_Q16 = chirp_Q16;
    for( i = 0; i < d - 1; i++ ) {
        ar[ i ]       = SKP_SMULWW( ar[ i ],   tmp_chirp_Q16 );
        tmp_chirp_Q16 = SKP_SMULWW( chirp_Q16, tmp_chirp_Q16 );
    }
    ar[ d - 1 ] = SKP_SMULWW( ar[ d - 1 ], tmp_chirp_Q16 );
}
```

A.16. src/SKP_Silk_burg_modified.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_Silk_burg_modified.c                                           *
 *                                                                      *
 * Calculates the reflection coefficients from the input vector         *
 * Input vector contains nb_subfr sub vectors of length L_sub + D       *
 *                                                                      *
 * Copyright 2009 (c), Skype Limited                                    *
 * Date: 100105                                                         *
 */

#include "SKP_Silk_SigProc_FIX.h"

#define MAX_FRAME_SIZE              544 // subfr_length * nb_subfr = ( 0.005 * 24
000 + 16 ) * 4 = 544
#define MAX_NB_SUBFR                4

#define QA                          24
#define N_BITS_HEAD_ROOM            2
#define MIN_RSHIFTS                 -16
```

```
#define MAX_RSHIFTS                 (32 - QA)

/* Compute reflection coefficients from input signal */
void SKP_Silk_burg_modified(
    SKP_int32       *res_nrg,           /* O    residual energy
                                        */
    SKP_int         *res_nrg_Q,         /* O    residual energy Q value
                                        */
    SKP_int32       A_Q16[],            /* O    prediction coefficients (length o
rder)                                   */
    const SKP_int16 x[],                /* I    input signal, length: nb_subfr *
( D + subfr_length )                    */
    const SKP_int   subfr_length,       /* I    input signal subframe length (inc
luding D preceeding samples)            */
    const SKP_int   nb_subfr,           /* I    number of subframes stacked in x
                                        */
    const SKP_int32 WhiteNoiseFrac_Q32, /* I    fraction added to zero-lag autoco
rrelation                               */
    const SKP_int   D                   /* I    order
                                        */
)
{
    SKP_int         k, n, s, lz, rshifts, rshifts_extra;
    SKP_int32       C0, num, nrg, rc_Q31, Atmp_QA, Atmp1, tmp1, tmp2, x1, x2;
    const SKP_int16 *x_ptr;

    SKP_int32       C_first_row[ SigProc_MAX_ORDER_LPC ];
    SKP_int32       C_last_row[  SigProc_MAX_ORDER_LPC ];
    SKP_int32       Af_QA[       SigProc_MAX_ORDER_LPC ];

    SKP_int32       CAf[ SigProc_MAX_ORDER_LPC + 1 ];
    SKP_int32       CAb[ SigProc_MAX_ORDER_LPC + 1 ];

    SKP_assert( subfr_length * nb_subfr <= MAX_FRAME_SIZE );
    SKP_assert( nb_subfr <= MAX_NB_SUBFR );


    /* Compute autocorrelations, added over subframes */
    SKP_Silk_sum_sqr_shift( &C0, &rshifts, x, nb_subfr * subfr_length );
    if( rshifts > MAX_RSHIFTS ) {
        C0 = SKP_LSHIFT32( C0, rshifts - MAX_RSHIFTS );
        SKP_assert( C0 > 0 );
        rshifts = MAX_RSHIFTS;
    } else {
        lz = SKP_Silk_CLZ32( C0 ) - 1;
        rshifts_extra = N_BITS_HEAD_ROOM - lz;
        if( rshifts_extra > 0 ) {
            rshifts_extra = SKP_min( rshifts_extra, MAX_RSHIFTS - rshifts );
            C0 = SKP_RSHIFT32( C0, rshifts_extra );
        } else {
            rshifts_extra = SKP_max( rshifts_extra, MIN_RSHIFTS - rshifts );
            C0 = SKP_LSHIFT32( C0, -rshifts_extra );
        }
        rshifts += rshifts_extra;
    }
    SKP_memset( C_first_row, 0, SigProc_MAX_ORDER_LPC * sizeof( SKP_int32 ) );
```

```
    if( rshifts > 0 ) {
        for( s = 0; s < nb_subfr; s++ ) {
            x_ptr = x + s * subfr_length;
            for( n = 1; n < D + 1; n++ ) {
                C_first_row[ n - 1 ] += (SKP_int32)SKP_RSHIFT64(
                    SKP_Silk_inner_prod16_aligned_64( x_ptr, x_ptr + n, subfr_len
gth - n ), rshifts );
            }
        }
    } else {
        for( s = 0; s < nb_subfr; s++ ) {
            x_ptr = x + s * subfr_length;
            for( n = 1; n < D + 1; n++ ) {
                C_first_row[ n - 1 ] += SKP_LSHIFT32(
                    SKP_Silk_inner_prod_aligned( x_ptr, x_ptr + n, subfr_length -
 n ), -rshifts );
            }
        }
    }
    SKP_memcpy( C_last_row, C_first_row, SigProc_MAX_ORDER_LPC * sizeof( SKP_int3
2 ) );

    /* Initialize */
    CAb[ 0 ] = CAf[ 0 ] = C0 + SKP_SMMUL( WhiteNoiseFrac_Q32, C0 ) + 1;         /
/ Q(-rshifts)

    for( n = 0; n < D; n++ ) {
        /* Update first row of correlation matrix (without first element) */
        /* Update last row of correlation matrix (without last element, stored in
 reversed order) */
        /* Update C * Af */
        /* Update C * flipud(Af) (stored in reversed order) */
        if( rshifts > -2 ) {
            for( s = 0; s < nb_subfr; s++ ) {
                x_ptr = x + s * subfr_length;
                x1  = -SKP_LSHIFT32( (SKP_int32)x_ptr[ n ],                    16
 - rshifts );      // Q(16-rshifts)
                x2  = -SKP_LSHIFT32( (SKP_int32)x_ptr[ subfr_length - n - 1 ], 16
 - rshifts );      // Q(16-rshifts)
                tmp1 = SKP_LSHIFT32( (SKP_int32)x_ptr[ n ],                    QA
 - 16 );           // Q(QA-16)
                tmp2 = SKP_LSHIFT32( (SKP_int32)x_ptr[ subfr_length - n - 1 ], QA
 - 16 );           // Q(QA-16)
                for( k = 0; k < n; k++ ) {
                    C_first_row[ k ] = SKP_SMLAWB( C_first_row[ k ], x1, x_ptr[ n
 - k - 1 ]           ); // Q( -rshifts )
                    C_last_row[ k ]  = SKP_SMLAWB( C_last_row[ k ],  x2, x_ptr[ s
ubfr_length - n + k ] ); // Q( -rshifts )
                    Atmp_QA = Af_QA[ k ];
                    tmp1 = SKP_SMLAWB( tmp1, Atmp_QA, x_ptr[ n - k - 1 ]
   );              // Q(QA-16)
                    tmp2 = SKP_SMLAWB( tmp2, Atmp_QA, x_ptr[ subfr_length - n + k
 ] );              // Q(QA-16)
                }
                tmp1 = SKP_LSHIFT32( -tmp1, 32 - QA - rshifts );
                   // Q(16-rshifts)
                tmp2 = SKP_LSHIFT32( -tmp2, 32 - QA - rshifts );
                   // Q(16-rshifts)
                for( k = 0; k <= n; k++ ) {
                    CAf[ k ] = SKP_SMLAWB( CAf[ k ], tmp1, x_ptr[ n - k ]
            );     // Q( -rshift )
                    CAb[ k ] = SKP_SMLAWB( CAb[ k ], tmp2, x_ptr[ subfr_length -
```

```
n + k - 1 ] );     // Q( -rshift )
                }
            }
```

```
        } else {
            for( s = 0; s < nb_subfr; s++ ) {
                x_ptr = x + s * subfr_length;
                x1  = -SKP_LSHIFT32( (SKP_int32)x_ptr[ n ],                    -r
shifts );          // Q( -rshifts )
                x2  = -SKP_LSHIFT32( (SKP_int32)x_ptr[ subfr_length - n - 1 ], -r
shifts );          // Q( -rshifts )
                tmp1 = SKP_LSHIFT32( (SKP_int32)x_ptr[ n ],                    17
 );                // Q17
                tmp2 = SKP_LSHIFT32( (SKP_int32)x_ptr[ subfr_length - n - 1 ], 17
 );                // Q17
                for( k = 0; k < n; k++ ) {
                    C_first_row[ k ] = SKP_MLA( C_first_row[ k ], x1, x_ptr[ n -
k - 1 ]             ); // Q( -rshifts )
                    C_last_row[ k ]  = SKP_MLA( C_last_row[ k ],  x2, x_ptr[ subf
r_length - n + k ] ); // Q( -rshifts )
                    Atmp1 = SKP_RSHIFT_ROUND( Af_QA[ k ], QA - 17 );
                   // Q17
                    tmp1 = SKP_MLA( tmp1, x_ptr[ n - k - 1 ],            Atmp1 );
                   // Q17
                    tmp2 = SKP_MLA( tmp2, x_ptr[ subfr_length - n + k ], Atmp1 );
                   // Q17
                }
                tmp1 = -tmp1;
                   // Q17
                tmp2 = -tmp2;
                   // Q17
                for( k = 0; k <= n; k++ ) {
                    CAf[ k ] = SKP_SMLAWW( CAf[ k ], tmp1,
                        SKP_LSHIFT32( (SKP_int32)x_ptr[ n - k ], -rshifts - 1 ) )
;                  // Q( -rshift )
                    CAb[ k ] = SKP_SMLAWW( CAb[ k ], tmp2,
                        SKP_LSHIFT32( (SKP_int32)x_ptr[ subfr_length - n + k - 1
], -rshifts - 1 ) );// Q( -rshift )
                }
            }
        }

        /* Calculate nominator and denominator for the next order reflection (par
cor) coefficient */
        tmp1 = C_first_row[ n ];
           // Q( -rshifts )
        tmp2 = C_last_row[ n ];
           // Q( -rshifts )
        num  = 0;
           // Q( -rshifts )
        nrg  = SKP_ADD32( CAb[ 0 ], CAf[ 0 ] );
           // Q( 1-rshifts )
        for( k = 0; k < n; k++ ) {
            Atmp_QA = Af_QA[ k ];
            lz = SKP_Silk_CLZ32( SKP_abs( Atmp_QA ) ) - 1;
            lz = SKP_min( 32 - QA, lz );
            Atmp1 = SKP_LSHIFT32( Atmp_QA, lz );
           // Q( QA + lz )

            tmp1 = SKP_ADD_LSHIFT32( tmp1, SKP_SMMUL( C_last_row[  n - k - 1 ], A
tmp1 ), 32 - QA - lz );    // Q( -rshifts )
            tmp2 = SKP_ADD_LSHIFT32( tmp2, SKP_SMMUL( C_first_row[ n - k - 1 ], A
tmp1 ), 32 - QA - lz );    // Q( -rshifts )
            num  = SKP_ADD_LSHIFT32( num,  SKP_SMMUL( CAb[ n - k ],             A
tmp1 ), 32 - QA - lz );    // Q( -rshifts )
            nrg  = SKP_ADD_LSHIFT32( nrg,  SKP_SMMUL( SKP_ADD32( CAb[ k + 1 ], CA
```

```
f[ k + 1 ] ),
                                                                        A
tmp1 ), 32 - QA - lz );    // Q( 1-rshifts )
        }
        CAf[ n + 1 ] = tmp1;
           // Q( -rshifts )
        CAb[ n + 1 ] = tmp2;
           // Q( -rshifts )
        num = SKP_ADD32( num, tmp2 );
           // Q( -rshifts )
        num = SKP_LSHIFT32( -num, 1 );
           // Q( 1-rshifts )

        /* Calculate the next order reflection (parcor) coefficient */
```

```
        if( SKP_abs( num ) < nrg ) {
            rc_Q31 = SKP_DIV32_varQ( num, nrg, 31 );
        } else {
            /* Negative energy or ratio too high; set remaining coefficients to z
ero and exit loop */
            SKP_memset( &Af_QA[ n ], 0, ( D - n ) * sizeof( SKP_int32 ) );
            SKP_assert( 0 );
            break;
        }

        /* Update the AR coefficients */
        for( k = 0; k < (n + 1) >> 1; k++ ) {
            tmp1 = Af_QA[ k ];
           // QA
            tmp2 = Af_QA[ n - k - 1 ];
           // QA
            Af_QA[ k ]         = SKP_ADD_LSHIFT32( tmp1, SKP_SMMUL( tmp2, rc_Q31
), 1 );    // QA
            Af_QA[ n - k - 1 ] = SKP_ADD_LSHIFT32( tmp2, SKP_SMMUL( tmp1, rc_Q31
), 1 );    // QA
        }
        Af_QA[ n ] = SKP_RSHIFT32( rc_Q31, 31 - QA );
           // QA

        /* Update C * Af and C * Ab */
        for( k = 0; k <= n + 1; k++ ) {
            tmp1 = CAf[ k ];
           // Q( -rshifts )
            tmp2 = CAb[ n - k + 1 ];
           // Q( -rshifts )
            CAf[ k ]         = SKP_ADD_LSHIFT32( tmp1, SKP_SMMUL( tmp2, rc_Q31 ),
 1 );      // Q( -rshifts )
            CAb[ n - k + 1 ] = SKP_ADD_LSHIFT32( tmp2, SKP_SMMUL( tmp1, rc_Q31 ),
 1 );      // Q( -rshifts )
        }
    }

    /* Return residual energy */
    nrg  = CAf[ 0 ];
           // Q( -rshifts )
    tmp1 = 1 << 16;
           // Q16
    for( k = 0; k < D; k++ ) {
        Atmp1 = SKP_RSHIFT_ROUND( Af_QA[ k ], QA - 16 );
           // Q16
        nrg  = SKP_SMLAWW( nrg, CAf[ k + 1 ], Atmp1 );
           // Q( -rshifts )
        tmp1 = SKP_SMLAWW( tmp1, Atmp1, Atmp1 );
           // Q16
        A_Q16[ k ] = -Atmp1;
    }
    *res_nrg = SKP_SMLAWW( nrg, SKP_SMMUL( WhiteNoiseFrac_Q32, C0 ), -tmp1 );
           // Q( -rshifts )
    *res_nrg_Q = -rshifts;
}
```

A.17. src/SKP_Silk_CNG.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
```

Redistribution and use in source and binary forms, with or without modification, (subject to the limitations in the disclaimer below)

```
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/* Generates excitation for CNG LPC synthesis */
SKP_INLINE void SKP_Silk_CNG_exc(
    SKP_int16                       residual[],         /* O    CNG residual sign
al Q0                       */
    SKP_int32                       exc_buf_Q10[],      /* I    Random samples bu
ffer Q10                    */
    SKP_int32                       Gain_Q16,           /* I    Gain to apply
                            */
    SKP_int                         length,             /* I    Length
                            */
    SKP_int32                       *rand_seed          /* I/O  Seed to random in
dex generator               */
)
{
    SKP_int32 seed;
    SKP_int   i, idx, exc_mask;

    exc_mask = CNG_BUF_MASK_MAX;
    while( exc_mask > length ) {
        exc_mask = SKP_RSHIFT( exc_mask, 1 );
    }

    seed = *rand_seed;
    for( i = 0; i < length; i++ ) {
        seed = SKP_RAND( seed );
        idx = ( SKP_int )( SKP_RSHIFT( seed, 24 ) & exc_mask );
        SKP_assert( idx >= 0 );
        SKP_assert( idx <= CNG_BUF_MASK_MAX );
```

```
        residual[ i ] = ( SKP_int16 )SKP_SAT16( SKP_RSHIFT_ROUND( SKP_SMULWW( exc
_buf_Q10[ idx ], Gain_Q16 ), 10 ) );
    }
    *rand_seed = seed;
}

void SKP_Silk_CNG_Reset(
    SKP_Silk_decoder_state      *psDec              /* I/O  Decoder state
                        */
)
{
    SKP_int i, NLSF_step_Q15, NLSF_acc_Q15;

    NLSF_step_Q15 = SKP_DIV32_16( SKP_int16_MAX, psDec->LPC_order + 1 );
    NLSF_acc_Q15 = 0;
    for( i = 0; i < psDec->LPC_order; i++ ) {
        NLSF_acc_Q15 += NLSF_step_Q15;
        psDec->sCNG.CNG_smth_NLSF_Q15[ i ] = NLSF_acc_Q15;
    }
    psDec->sCNG.CNG_smth_Gain_Q16 = 0;
    psDec->sCNG.rand_seed = 3176576;
}

/* Updates CNG estimate, and applies the CNG when packet was lost   */
void SKP_Silk_CNG(
    SKP_Silk_decoder_state      *psDec,             /* I/O  Decoder state
                        */
    SKP_Silk_decoder_control    *psDecCtrl,         /* I/O  Decoder control
                        */
    SKP_int16                   signal[],           /* I/O  Signal
                        */
    SKP_int                     length              /* I    Length of residual
                        */
)
{
    SKP_int   i, subfr;
    SKP_int32 tmp_32, Gain_Q26, max_Gain_Q16;
    SKP_int16 LPC_buf[ MAX_LPC_ORDER ];
    SKP_int16 CNG_sig[ MAX_FRAME_LENGTH ];
    SKP_Silk_CNG_struct *psCNG;
    psCNG = &psDec->sCNG;

    if( psDec->fs_kHz != psCNG->fs_kHz ) {
        /* Reset state */
        SKP_Silk_CNG_Reset( psDec );

        psCNG->fs_kHz = psDec->fs_kHz;
    }
    if( psDec->lossCnt == 0 && psDec->vadFlag == NO_VOICE_ACTIVITY ) {
        /* Update CNG parameters */

        /* Smoothing of LSF's  */
        for( i = 0; i < psDec->LPC_order; i++ ) {
            psCNG->CNG_smth_NLSF_Q15[ i ] += SKP_SMULWB( psDec->prevNLSF_Q15[ i ]
 - psCNG->CNG_smth_NLSF_Q15[ i ], CNG_NLSF_SMTH_Q16 );
```

```
        }
        /* Find the subframe with the highest gain */
        max_Gain_Q16 = 0;
        subfr        = 0;
        for( i = 0; i < NB_SUBFR; i++ ) {
            if( psDecCtrl->Gains_Q16[ i ] > max_Gain_Q16 ) {
                max_Gain_Q16 = psDecCtrl->Gains_Q16[ i ];
                subfr        = i;
            }
        }
        /* Update CNG excitation buffer with excitation from this subframe */
        SKP_memmove( &psCNG->CNG_exc_buf_Q10[ psDec->subfr_length ], psCNG->CNG_e
xc_buf_Q10, ( NB_SUBFR - 1 ) * psDec->subfr_length * sizeof( SKP_int32 ) );
        SKP_memcpy(   psCNG->CNG_exc_buf_Q10, &psDec->exc_Q10[ subfr * psDec->sub
fr_length ], psDec->subfr_length * sizeof( SKP_int32 ) );

        /* Smooth gains */
        for( i = 0; i < NB_SUBFR; i++ ) {
            psCNG->CNG_smth_Gain_Q16 += SKP_SMULWB( psDecCtrl->Gains_Q16[ i ] - p
sCNG->CNG_smth_Gain_Q16, CNG_GAIN_SMTH_Q16 );
        }
    }

    /* Add CNG when packet is lost and / or when low speech activity */
    if( psDec->lossCnt ) {//|| psDec->vadFlag == NO_VOICE_ACTIVITY ) {

        /* Generate CNG excitation */
        SKP_Silk_CNG_exc( CNG_sig, psCNG->CNG_exc_buf_Q10,
                psCNG->CNG_smth_Gain_Q16, length, &psCNG->rand_seed );

        /* Convert CNG NLSF to filter representation */
        SKP_Silk_NLSF2A_stable( LPC_buf, psCNG->CNG_smth_NLSF_Q15, psDec->LPC_ord
er );

        Gain_Q26 = ( SKP_int32 )1 << 26; /* 1.0 */

        /* Generate CNG signal, by synthesis filtering */
        if( psDec->LPC_order == 16 ) {
            SKP_Silk_LPC_synthesis_order16( CNG_sig, LPC_buf,
                Gain_Q26, psCNG->CNG_synth_state, CNG_sig, length );
        } else {
            SKP_Silk_LPC_synthesis_filter( CNG_sig, LPC_buf,
                Gain_Q26, psCNG->CNG_synth_state, CNG_sig, length, psDec->LPC_ord
er );
        }
        /* Mix with signal */
        for( i = 0; i < length; i++ ) {
            tmp_32 = signal[ i ] + CNG_sig[ i ];
            signal[ i ] = SKP_SAT16( tmp_32 );
        }
    } else {
        SKP_memset( psCNG->CNG_synth_state, 0, psDec->LPC_order *  sizeof( SKP_in
t32 ) );
    }
```

```
}
```

A.18. src/SKP_Silk_code_signs.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

//#define SKP_enc_map(a)                ((a) > 0 ? 1 : 0)
//#define SKP_dec_map(a)                ((a) > 0 ? 1 : -1)
/* shifting avoids if-statement */
#define SKP_enc_map(a)                  ( SKP_RSHIFT( (a), 15 ) + 1 )
#define SKP_dec_map(a)                  ( SKP_LSHIFT( (a),  1 ) - 1 )

/* Encodes signs of excitation */
void SKP_Silk_encode_signs(
    SKP_Silk_range_coder_state      *sRC,               /* I/O  Range coder state
                    */
    const SKP_int                   q[],                /* I    Pulse signal
                    */
    const SKP_int                   length,             /* I    Length of input
                    */
    const SKP_int                   sigtype,            /* I    Signal type
                    */
    const SKP_int                   QuantOffsetType,    /* I    Quantization offs
et type             */
```

```
    const SKP_int                   RateLevelIndex      /* I    Rate level index
                        */
)
{
    SKP_int i;
    SKP_int inData;
    const SKP_uint16 *cdf;

    i = SKP_SMULBB( N_RATE_LEVELS - 1, SKP_LSHIFT( sigtype, 1 ) + QuantOffsetType
 ) + RateLevelIndex;
    cdf = SKP_Silk_sign_CDF[ i ];

    for( i = 0; i < length; i++ ) {
        if( q[ i ] != 0 ) {
            inData = SKP_enc_map( q[ i ] ); /* - = 0, + = 1 */
            SKP_Silk_range_encoder( sRC, inData, cdf );
        }
    }
}

/* Decodes signs of excitation */
void SKP_Silk_decode_signs(
    SKP_Silk_range_coder_state      *sRC,               /* I/O  Range coder state
                            */
    SKP_int                         q[],                /* I/O  pulse signal
                            */
    const SKP_int                   length,             /* I    length of output
                            */
    const SKP_int                   sigtype,            /* I    Signal type
                            */
    const SKP_int                   QuantOffsetType,    /* I    Quantization offs
et type                     */
    const SKP_int                   RateLevelIndex      /* I    Rate Level Index
                            */
)
{
    SKP_int i;
    SKP_int data;
    const SKP_uint16 *cdf;

    i = SKP_SMULBB( N_RATE_LEVELS - 1, SKP_LSHIFT( sigtype, 1 ) + QuantOffsetType
 ) + RateLevelIndex;
    cdf = SKP_Silk_sign_CDF[ i ];

    for( i = 0; i < length; i++ ) {
        if( q[ i ] > 0 ) {
            SKP_Silk_range_decoder( &data, sRC, cdf, 1 );
            /* attach sign */
            /* implementation with shift, subtraction, multiplication */
            q[ i ] *= SKP_dec_map( data );
        }
    }
}
```

A.19. src/SKP_Silk_common_pitch_est_defines.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SIGPROC_COMMON_PITCH_EST_DEFINES_H
#define SIGPROC_COMMON_PITCH_EST_DEFINES_H

#include "SKP_Silk_SigProc_FIX.h"

/*********************************************************/
/* Definitions For Fix pitch estimator                   */
/*********************************************************/

#define PITCH_EST_MAX_FS_KHZ                24 /* Maximum sampling frequency used
 */

#define PITCH_EST_FRAME_LENGTH_MS           40 /* 40 ms */

#define PITCH_EST_MAX_FRAME_LENGTH          (PITCH_EST_FRAME_LENGTH_MS * PITCH_ES
T_MAX_FS_KHZ)
#define PITCH_EST_MAX_FRAME_LENGTH_ST_1     (PITCH_EST_MAX_FRAME_LENGTH >> 2)
#define PITCH_EST_MAX_FRAME_LENGTH_ST_2     (PITCH_EST_MAX_FRAME_LENGTH >> 1)
#define PITCH_EST_MAX_SF_FRAME_LENGTH       (PITCH_EST_SUB_FRAME * PITCH_EST_MAX_
FS_KHZ)

#define PITCH_EST_MAX_LAG_MS                18            /* 18 ms -> 56 Hz */
#define PITCH_EST_MIN_LAG_MS                2            /* 2 ms -> 500 Hz */
#define PITCH_EST_MAX_LAG                   (PITCH_EST_MAX_LAG_MS * PITCH_EST_MAX
_FS_KHZ)
```

```
#define PITCH_EST_MIN_LAG                   (PITCH_EST_MIN_LAG_MS * PITCH_EST_MAX
_FS_KHZ)

#define PITCH_EST_NB_SUBFR                  4

#define PITCH_EST_D_SRCH_LENGTH             24

#define PITCH_EST_MAX_DECIMATE_STATE_LENGTH 7

#define PITCH_EST_NB_STAGE3_LAGS            5

#define PITCH_EST_NB_CBKS_STAGE2            3
#define PITCH_EST_NB_CBKS_STAGE2_EXT        11

#define PITCH_EST_CB_mn2                    1
#define PITCH_EST_CB_mx2                    2

#define PITCH_EST_NB_CBKS_STAGE3_MAX        34
#define PITCH_EST_NB_CBKS_STAGE3_MID        24
#define PITCH_EST_NB_CBKS_STAGE3_MIN        16

extern const SKP_int16 SKP_Silk_CB_lags_stage2[PITCH_EST_NB_SUBFR][PITCH_EST_NB_C
BKS_STAGE2_EXT];
extern const SKP_int16 SKP_Silk_CB_lags_stage3[PITCH_EST_NB_SUBFR][PITCH_EST_NB_C
BKS_STAGE3_MAX];
extern const SKP_int16 SKP_Silk_Lag_range_stage3[ SigProc_PITCH_EST_MAX_COMPLEX +
 1 ] [ PITCH_EST_NB_SUBFR ][ 2 ];
extern const SKP_int16 SKP_Silk_cbk_sizes_stage3[ SigProc_PITCH_EST_MAX_COMPLEX +
 1 ];
extern const SKP_int16 SKP_Silk_cbk_offsets_stage3[ SigProc_PITCH_EST_MAX_COMPLEX
 + 1 ];

#endif
```

A.20. src/SKP_Silk_control_codec_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
```

```
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/* Control encoder SNR */
SKP_int SKP_Silk_control_encoder_FIX(
    SKP_Silk_encoder_state_FIX  *psEnc,             /* I/O  Pointer to Silk encod
er state                    */
    const SKP_int               API_fs_kHz,         /* I    External (API) sampli
ng rate (kHz)               */
    const SKP_int               PacketSize_ms,      /* I    Packet length (ms)
                            */
    SKP_int32                   TargetRate_bps,     /* I    Target max bitrate (b
ps) (used if SNR_dB == 0)   */
    const SKP_int               PacketLoss_perc,    /* I    Packet loss rate (in
percent)                    */
    const SKP_int               INBandFec_enabled,  /* I    Enable (1) / disable
(0) inband FEC              */
    const SKP_int               DTX_enabled,        /* I    Enable / disable DTX
                            */
    const SKP_int               InputFramesize_ms,  /* I    Inputframe in ms
                            */
    const SKP_int               Complexity          /* I    Complexity (0->low; 1
->medium; 2->high)          */
)
{
    SKP_int32 LBRRRate_thres_bps;
    SKP_int   k, fs_kHz, ret = 0;
    SKP_int32 frac_Q6;
    const SKP_int32 *rateTable;

    /* State machine for the SWB/WB switching */
    fs_kHz = psEnc->sCmn.fs_kHz;

    /* Only switch during low speech activity, when no frames are sitting in the
payload buffer */
    if( API_fs_kHz == 8 || fs_kHz == 0 || API_fs_kHz < fs_kHz ) {
        // Switching is not possible, encoder just initialized, or internal mode
higher than external
        fs_kHz = API_fs_kHz;
    } else {

        /* Resample all valid data in x_buf. Resampling the last part gets rid of
 a click, 5ms after switching  */
        /* this is because the same state is used when downsampling in API.c and
is then up to date             */
        /* the click immidiatly after switching is most of the time still there
                              */

        if( psEnc->sCmn.fs_kHz == 24 ) {
            /* Accumulate the difference between the target rate and limit */
            psEnc->sCmn.bitrateDiff += SKP_MUL( InputFramesize_ms, TargetRate_bps
```

```
 - SWB2WB_BITRATE_BPS );
```

```
            psEnc->sCmn.bitrateDiff = SKP_min( psEnc->sCmn.bitrateDiff, 0 );

            /* Check if we should switch from 24 to 16 kHz */
#if SWITCH_TRANSITION_FILTERING
            if( ( psEnc->sCmn.sLP.transition_frame_no == 0 ) && /* Transition pha
se not active */
                ( psEnc->sCmn.bitrateDiff <= -ACCUM_BITS_DIFF_THRESHOLD || psEnc-
>sCmn.sSWBdetect.WB_detected == 1 ) &&
                ( psEnc->speech_activity_Q8 < 128 && psEnc->sCmn.nFramesInPayload
Buf == 0 ) ) {
                psEnc->sCmn.sLP.transition_frame_no = 1; /* Begin transition phas
e */
                psEnc->sCmn.sLP.mode = 0; /* Switch down */
            }

            if( ( psEnc->sCmn.sLP.transition_frame_no >= TRANSITION_FRAMES_DOWN )
 && ( psEnc->sCmn.sLP.mode == 0 ) && /* Transition phase complete, ready to switc
h */
#else
            if( ( psEnc->sCmn.bitrateDiff <= -ACCUM_BITS_DIFF_THRESHOLD || psEnc-
>sCmn.sSWBdetect.WB_detected == 1 ) &&
#endif
                ( psEnc->speech_activity_Q8 < 128 && psEnc->sCmn.nFramesInPayload
Buf == 0 ) ) {

                    SKP_int16 x_buf[    2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX ];
                    SKP_int16 x_bufout[ 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX ];

                    psEnc->sCmn.bitrateDiff = 0;
                    fs_kHz = 16;

                    SKP_memcpy( x_buf, psEnc->x_buf, ( 2 * MAX_FRAME_LENGTH + LA_
SHAPE_MAX ) * sizeof( SKP_int16 ) );

                    SKP_memset( psEnc->sCmn.resample24To16state, 0, sizeof( psEnc
->sCmn.resample24To16state ) );

#if LOW_COMPLEXITY_ONLY
                    {
                        SKP_int16 scratch[ ( 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX
) + SigProc_Resample_2_3_coarse_NUM_FIR_COEFS - 1 ];
                        SKP_Silk_resample_2_3_coarse( &x_bufout[ 0 ], psEnc->sCmn
.resample24To16state, &x_buf[ 0 ], SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + ps
Enc->sCmn.la_shape, (SKP_int16*)scratch );
                    }
#else
                    SKP_Silk_resample_2_3( &x_bufout[ 0 ], psEnc->sCmn.resample24
To16state, &x_buf[ 0 ], SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + psEnc->sCmn.l
a_shape );
#endif

                    /* set the first frame to zero, no performance difference was
 noticed though */
                    SKP_memset( x_bufout, 0, 320 * sizeof( SKP_int16 ) );
                    SKP_memcpy( psEnc->x_buf, x_bufout, ( 2 * MAX_FRAME_LENGTH +
LA_SHAPE_MAX ) * sizeof( SKP_int16 ) );

#if SWITCH_TRANSITION_FILTERING
                    psEnc->sCmn.sLP.transition_frame_no = 0; /* Transition phase
complete */
#endif
            }
```

```
        } else if( psEnc->sCmn.fs_kHz == 16 ) {

            /* Check if we should switch from 16 to 24 kHz */
#if SWITCH_TRANSITION_FILTERING
```

```
            if( ( psEnc->sCmn.sLP.transition_frame_no == 0 ) && /* No transition
phase running, ready to switch */
#else
            if(
#endif
                ( API_fs_kHz > psEnc->sCmn.fs_kHz && TargetRate_bps >= WB2SWB_BIT
RATE_BPS && psEnc->sCmn.sSWBdetect.WB_detected == 0 ) &&
                ( psEnc->speech_activity_Q8 < 128 && psEnc->sCmn.nFramesInPayload
Buf == 0 ) ) {

                SKP_int16 x_buf[          2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX ];
                SKP_int16 x_bufout[ 3 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX ) /
 2 ];
                SKP_int32 resample16To24state[ 11 ];

                psEnc->sCmn.bitrateDiff = 0;
                fs_kHz = 24;

                SKP_memcpy( x_buf, psEnc->x_buf, ( 2 * MAX_FRAME_LENGTH + LA_SHAP
E_MAX ) * sizeof( SKP_int16 ) );

                SKP_memset( resample16To24state, 0, sizeof(resample16To24state) )
;

                SKP_Silk_resample_3_2( &x_bufout[ 0 ], resample16To24state, &x_bu
f[ 0 ], SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + psEnc->sCmn.la_shape );

                /* set the first frame to zero, no performance difference was not
iced though */
                SKP_memset( x_bufout, 0, 480 * sizeof( SKP_int16 ) );
                SKP_memcpy( psEnc->x_buf, x_bufout, ( 2 * MAX_FRAME_LENGTH + LA_S
HAPE_MAX ) * sizeof( SKP_int16 ) );
#if SWITCH_TRANSITION_FILTERING
                psEnc->sCmn.sLP.mode = 1; /* Switch up */
#endif
            } else {
                /* accumulate the difference between the target rate and limit */
                psEnc->sCmn.bitrateDiff += SKP_MUL( InputFramesize_ms, TargetRate
_bps - WB2MB_BITRATE_BPS );
                psEnc->sCmn.bitrateDiff = SKP_min( psEnc->sCmn.bitrateDiff, 0 );

                /* Check if we should switch from 16 to 12 kHz */
#if SWITCH_TRANSITION_FILTERING
                if( ( psEnc->sCmn.sLP.transition_frame_no == 0 ) && /* Transition
 phase not active */
                    ( psEnc->sCmn.bitrateDiff <= -ACCUM_BITS_DIFF_THRESHOLD ) &&
                    ( psEnc->speech_activity_Q8 < 128 && psEnc->sCmn.nFramesInPay
loadBuf == 0 ) ) {
                    psEnc->sCmn.sLP.transition_frame_no = 1; /* Begin transition
phase */
                    psEnc->sCmn.sLP.mode = 0; /* Switch down */
                }

                if( ( psEnc->sCmn.sLP.transition_frame_no >= TRANSITION_FRAMES_DO
WN ) && ( psEnc->sCmn.sLP.mode == 0 ) && /* Transition phase complete, ready to s
witch */
#else
                if( ( psEnc->sCmn.bitrateDiff <= -ACCUM_BITS_DIFF_THRESHOLD ) &&
#endif
                    ( psEnc->speech_activity_Q8 < 128 && psEnc->sCmn.nFramesInPay
loadBuf == 0 ) ) {
```

```
                    SKP_int16 x_buf[ 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX ];
```

```
                    SKP_memcpy( x_buf, psEnc->x_buf, ( 2 * MAX_FRAME_LENGTH + LA_
SHAPE_MAX ) * sizeof( SKP_int16 ) );

                    psEnc->sCmn.bitrateDiff = 0;
                    fs_kHz = 12;

                    if( API_fs_kHz == 24 ) {

                        /* Intermediate upsampling of x_bufFIX from 16 to 24 kHz
*/
                        SKP_int16 x_buf24[ 3 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE_
MAX ) / 2 ];
                        SKP_int32 scratch[    3 * ( 2 * MAX_FRAME_LENGTH + LA_SHA
PE_MAX ) ];
                        SKP_int32 resample16To24state[ 11 ];

                        SKP_memset( resample16To24state, 0, sizeof( resample16To2
4state ) );
                        SKP_Silk_resample_3_2( &x_buf24[ 0 ], resample16To24state
, &x_buf[ 0 ], SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + psEnc->sCmn.la_shape )
;

                        /* Update the state of the resampler used in API.c, from
24 to 12 kHz */
                        SKP_memset( psEnc->sCmn.resample24To12state, 0, sizeof( p
sEnc->sCmn.resample24To12state ) );
                        SKP_Silk_resample_1_2_coarse( &x_buf24[ 0 ], psEnc->sCmn.
resample24To12state, &x_buf[ 0 ], scratch, SKP_RSHIFT( SKP_SMULBB( 3, SKP_LSHIFT(
 psEnc->sCmn.frame_length, 1 ) + psEnc->sCmn.la_shape ), 2 ) );

                        /* set the first frame to zero, no performance difference
 was noticed though */
                        SKP_memset( x_buf, 0, 240 * sizeof( SKP_int16 ) );
                        SKP_memcpy( psEnc->x_buf, x_buf, ( 2 * MAX_FRAME_LENGTH +
 LA_SHAPE_MAX ) * sizeof( SKP_int16 ) );

                    } else if( API_fs_kHz == 16 ) {
                        SKP_int16 x_bufout[ 3 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE
_MAX ) / 4 ];
                        SKP_memset( psEnc->sCmn.resample16To12state, 0, sizeof( p
sEnc->sCmn.resample16To12state ) );

                        SKP_Silk_resample_3_4( &x_bufout[ 0 ], psEnc->sCmn.resamp
le16To12state, &x_buf[ 0 ], SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + psEnc->sC
mn.la_shape );

                        /* set the first frame to zero, no performance difference
 was noticed though */
                        SKP_memset( x_bufout, 0, 240 * sizeof( SKP_int16 ) );
                        SKP_memcpy( psEnc->x_buf, x_bufout, ( 2 * MAX_FRAME_LENGT
H + LA_SHAPE_MAX ) * sizeof( SKP_int16 ) );
                    }
#if SWITCH_TRANSITION_FILTERING
                    psEnc->sCmn.sLP.transition_frame_no = 0; /* Transition phase
complete */
#endif
                }
            }
        } else if( psEnc->sCmn.fs_kHz == 12 ) {

            /* Check if we should switch from 12 to 16 kHz */
#if SWITCH_TRANSITION_FILTERING
```

```
            if( ( psEnc->sCmn.sLP.transition_frame_no == 0 ) && /* No transition
phase running, ready to switch */
#else
            if(
#endif
                ( API_fs_kHz > psEnc->sCmn.fs_kHz && TargetRate_bps >= MB2WB_BITR
ATE_BPS ) &&
                ( psEnc->speech_activity_Q8 < 128 && psEnc->sCmn.nFramesInPayload
Buf == 0 ) ) {
```

```
                SKP_int16 x_buf[ 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX ];

                SKP_memcpy( x_buf, psEnc->x_buf, ( 2 * MAX_FRAME_LENGTH + LA_SHAP
E_MAX ) * sizeof( SKP_int16 ) );

                psEnc->sCmn.bitrateDiff = 0;
                fs_kHz = 16;

                /* Reset state of the resampler to be used */
                if( API_fs_kHz == 24 ) {

                    SKP_int16 x_bufout[ 2 * 2 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE
_MAX ) / 3 ];

                    /* Intermediate upsampling of x_bufFIX from 12 to 24 kHz */
                    SKP_int16 x_buf24[ 2 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX
) ];
                    SKP_int32 scratch[    3 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE_M
AX ) ];
                    SKP_int32 resample12To24state[6];

                    SKP_memset( resample12To24state, 0, sizeof( resample12To24sta
te ) );
                    SKP_Silk_resample_2_1_coarse( &x_buf[ 0 ], resample12To24stat
e, &x_buf24[ 0 ], scratch, SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + psEnc->sCm
n.la_shape );

                    SKP_memset( psEnc->sCmn.resample24To16state, 0, sizeof( psEnc
->sCmn.resample24To16state ) );

#if LOW_COMPLEXITY_ONLY
                    SKP_assert( sizeof( SKP_int16 ) * ( 2 * ( 2 * MAX_FRAME_LENGT
H + LA_SHAPE_MAX ) + SigProc_Resample_2_3_coarse_NUM_FIR_COEFS - 1 ) <= sizeof( s
cratch ) );
                    SKP_Silk_resample_2_3_coarse( &x_bufout[ 0 ], psEnc->sCmn.res
ample24To16state, &x_buf24[ 0 ], SKP_LSHIFT( SKP_LSHIFT( psEnc->sCmn.frame_length
, 1 ) + psEnc->sCmn.la_shape, 1 ), (SKP_int16*)scratch );
#else
                    SKP_Silk_resample_2_3( &x_bufout[ 0 ], psEnc->sCmn.resample24
To16state, &x_buf24[ 0 ], SKP_LSHIFT( SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) +
 psEnc->sCmn.la_shape, 1 ) );
#endif

                    /* set the first frame to zero, no performance difference was
 noticed though */
                    SKP_memset( x_bufout, 0, 320 * sizeof( SKP_int16 ) );
                    SKP_memcpy( psEnc->x_buf, x_bufout, ( 2 * MAX_FRAME_LENGTH +
LA_SHAPE_MAX ) * sizeof( SKP_int16 ) );
                }
#if SWITCH_TRANSITION_FILTERING
                psEnc->sCmn.sLP.mode = 1; /* Switch up */
#endif
            } else {
                /* accumulate the difference between the target rate and limit */
                psEnc->sCmn.bitrateDiff += SKP_MUL( InputFramesize_ms, TargetRate
_bps - MB2NB_BITRATE_BPS );
                psEnc->sCmn.bitrateDiff  = SKP_min( psEnc->sCmn.bitrateDiff, 0 );

                /* Check if we should switch from 12 to 8 kHz */
#if SWITCH_TRANSITION_FILTERING
                if( ( psEnc->sCmn.sLP.transition_frame_no == 0 ) && /* Transition
 phase not active */
```

```
                    ( psEnc->sCmn.bitrateDiff <= -ACCUM_BITS_DIFF_THRESHOLD ) &&
                    ( psEnc->speech_activity_Q8 < 128 && psEnc->sCmn.nFramesInPay
loadBuf == 0 ) ) {
                    psEnc->sCmn.sLP.transition_frame_no = 1; /* Begin transition
phase */
                    psEnc->sCmn.sLP.mode = 0; /* Switch down */
```

```
                }

                if( ( psEnc->sCmn.sLP.transition_frame_no >= TRANSITION_FRAMES_DO
WN ) && ( psEnc->sCmn.sLP.mode == 0 ) &&
#else
                if( ( psEnc->sCmn.bitrateDiff <= -ACCUM_BITS_DIFF_THRESHOLD ) &&
#endif
                    ( psEnc->speech_activity_Q8 < 128 && psEnc->sCmn.nFramesInPay
loadBuf == 0 ) ) {

                    SKP_int16 x_buf[ 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX ];

                    SKP_memcpy( x_buf, psEnc->x_buf, ( 2 * MAX_FRAME_LENGTH + LA_
SHAPE_MAX ) * sizeof( SKP_int16 ) );

                    psEnc->sCmn.bitrateDiff = 0;
                    fs_kHz = 8;

                    if( API_fs_kHz == 24 ) {

                        SKP_int32 scratch[    3 * ( 2 * MAX_FRAME_LENGTH + LA_SHA
PE_MAX ) ];
                        /* Intermediate upsampling of x_buf from 12 to 24 kHz */
                        SKP_int16 x_buf24[ 2 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE_
MAX ) ];
                        SKP_int32 resample12To24state[ 6 ];

                        SKP_memset( resample12To24state, 0, sizeof( resample12To2
4state ) );
                        SKP_Silk_resample_2_1_coarse( &x_buf[ 0 ], resample12To24
state, &x_buf24[ 0 ], scratch, SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + psEnc-
>sCmn.la_shape );

                        /* Update the state of the resampler used in API.c, from
24 to 8 kHz */
                        SKP_memset( psEnc->sCmn.resample24To8state, 0, sizeof( ps
Enc->sCmn.resample24To8state ) );
                        SKP_Silk_resample_1_3( &x_buf[ 0 ], psEnc->sCmn.resample2
4To8state, &x_buf24[ 0 ], SKP_LSHIFT( SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) +
 psEnc->sCmn.la_shape, 1 ) );

                        /* set the first frame to zero, no performance difference
 was noticed though */
                        SKP_memset( x_buf, 0, 160 * sizeof( SKP_int16 ) );
                        SKP_memcpy( psEnc->x_buf, x_buf, ( 2 * MAX_FRAME_LENGTH +
 LA_SHAPE_MAX ) * sizeof( SKP_int16 ) );

                    } else if( API_fs_kHz == 16 ) {
                        /* Intermediate upsampling of x_bufFIX from 12 to 16 kHz
*/
                        SKP_int16 x_buf16[  3 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE
_MAX ) / 2 ];
                        SKP_int32 resample12To16state[11];

                        SKP_memset( resample12To16state, 0, sizeof( resample12To1
6state ) );
                        SKP_Silk_resample_3_2( &x_buf16[ 0 ], resample12To16state
, &x_buf[ 0 ], SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + psEnc->sCmn.la_shape )
;

                        /* set the first frame to zero, no performance difference
 was noticed though */
```

```
                        SKP_memset( x_buf, 0, 160 * sizeof( SKP_int16 ) );
                        SKP_memcpy( psEnc->x_buf, x_buf, ( 2 * MAX_FRAME_LENGTH +
 LA_SHAPE_MAX ) * sizeof( SKP_int16 ) );

                    } else if( API_fs_kHz == 12 ) {
                        SKP_int16 x_bufout[ 2 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE
_MAX ) / 3 ];
                        SKP_memset( psEnc->sCmn.resample12To8state, 0, sizeof( ps
Enc->sCmn.resample12To8state ) );
```

```
#if LOW_COMPLEXITY_ONLY
                        {
                            SKP_int16 scratch[ ( 2 * MAX_FRAME_LENGTH + LA_SHAPE_
MAX ) + SigProc_Resample_2_3_coarse_NUM_FIR_COEFS - 1 ];
                            SKP_Silk_resample_2_3_coarse( &x_bufout[ 0 ], psEnc->
sCmn.resample12To8state, &x_buf[ 0 ],
                                SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + psEnc
->sCmn.la_shape, scratch );
                        }
#else
                        SKP_Silk_resample_2_3( &x_bufout[ 0 ], psEnc->sCmn.resamp
le12To8state, &x_buf[ 0 ], SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + psEnc->sCm
n.la_shape );
#endif
                        /* set the first frame to zero, no performance difference
 was noticed though */
                        SKP_memset( x_bufout, 0, 160 * sizeof( SKP_int16 ) );
                        SKP_memcpy( psEnc->x_buf, x_bufout, ( 2 * MAX_FRAME_LENGT
H + LA_SHAPE_MAX ) * sizeof( SKP_int16 ) );
                    }
#if SWITCH_TRANSITION_FILTERING
                    psEnc->sCmn.sLP.transition_frame_no = 0; /* Transition phase
complete */
#endif
                }
            }
        } else if( psEnc->sCmn.fs_kHz == 8 ) {

            /* Check if we should switch from 8 to 12 kHz */
#if SWITCH_TRANSITION_FILTERING
            if( ( psEnc->sCmn.sLP.transition_frame_no == 0 ) && /* No transition
phase running, ready to switch */
#else
            if(
#endif
                ( API_fs_kHz > psEnc->sCmn.fs_kHz && TargetRate_bps >= NB2MB_BITR
ATE_BPS ) &&
                ( psEnc->speech_activity_Q8 < 128 && psEnc->sCmn.nFramesInPayload
Buf == 0 ) ) {

                SKP_int16 x_buf[ 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX ];

                SKP_memcpy( x_buf, psEnc->x_buf, ( 2 * MAX_FRAME_LENGTH + LA_SHAP
E_MAX ) * sizeof( SKP_int16 ) );

                psEnc->sCmn.bitrateDiff = 0;
                fs_kHz = 12;

                /* Reset state of the resampler to be used */
                if( API_fs_kHz == 24 ) {
                    SKP_int16 x_buf24[  3 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX
 ) ];
                    SKP_int32 scratch[ 3 * 3 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE_
MAX ) / 2 ];
                    SKP_int32 resample8To24state[ 7 ];

                    /* Intermediate upsampling of x_bufFIX from 8 to 24 kHz */
                    SKP_memset( resample8To24state, 0, sizeof( resample8To24state
 ) );
                    SKP_Silk_resample_3_1( &x_buf24[ 0 ], resample8To24state, &x_
buf[ 0 ], SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + psEnc->sCmn.la_shape );
```

```
                    SKP_memset( psEnc->sCmn.resample24To12state, 0, sizeof( psEnc
->sCmn.resample24To12state ) );
```

```
                    SKP_Silk_resample_1_2_coarse( &x_buf24[ 0 ], psEnc->sCmn.resa
mple24To12state, &x_buf[ 0 ], scratch, SKP_RSHIFT( SKP_SMULBB( 3, SKP_LSHIFT( psE
nc->sCmn.frame_length, 1 ) + psEnc->sCmn.la_shape ), 1 ) );

                    /* set the first frame to zero, no performance difference was
 noticed though */
                    SKP_memset( x_buf, 0, 240 * sizeof( SKP_int16 ) );
                    SKP_memcpy( psEnc->x_buf, x_buf, ( 2 * MAX_FRAME_LENGTH + LA_
SHAPE_MAX ) * sizeof( SKP_int16 ) );

                } else if( API_fs_kHz == 16 ) {
                    SKP_int16 x_buf16[ 2 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX
) ];
                    SKP_int32 scratch[ 3 * ( 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX
) ];
                    SKP_int32 resample8To16state[ 6 ];

                    /* Intermediate upsampling of x_bufFIX from 8 to 16 kHz */
                    SKP_memset( resample8To16state, 0, sizeof( resample8To16state
 ) );
                    SKP_Silk_resample_2_1_coarse( &x_buf[ 0 ], resample8To16state
, &x_buf16[ 0 ], scratch, SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + psEnc->sCmn
.la_shape );

                    SKP_memset( psEnc->sCmn.resample16To12state, 0, sizeof( psEnc
->sCmn.resample16To12state ) );

                    SKP_Silk_resample_3_4( &x_buf[ 0 ], psEnc->sCmn.resample16To1
2state, &x_buf16[ 0 ], SKP_LSHIFT( SKP_LSHIFT( psEnc->sCmn.frame_length, 1 ) + ps
Enc->sCmn.la_shape, 1 ) );

                    /* set the first frame to zero, no performance difference was
 noticed though */
                    SKP_memset( x_buf, 0, 240 * sizeof( SKP_int16 ) );
                    SKP_memcpy( psEnc->x_buf, x_buf, ( 2 * MAX_FRAME_LENGTH + LA_
SHAPE_MAX ) * sizeof( SKP_int16 ) );
                }
#if SWITCH_TRANSITION_FILTERING
                psEnc->sCmn.sLP.mode = 1; /* Switch up */
#endif
            }
        } else {
            // Internal sample frequency not supported!
            SKP_assert( 0 );
        }
    }

#if SWITCH_TRANSITION_FILTERING
    /* After switching up, stop transition filter during speech inactivity */
    if( ( psEnc->sCmn.sLP.mode == 1 ) &&
        ( psEnc->sCmn.sLP.transition_frame_no >= TRANSITION_FRAMES_UP ) &&
        ( psEnc->speech_activity_Q8 < 128 ) &&
        ( psEnc->sCmn.nFramesInPayloadBuf == 0 ) ) {

        psEnc->sCmn.sLP.transition_frame_no = 0;

        /* Reset transition filter state */
        SKP_memset( psEnc->sCmn.sLP.In_LP_State, 0, 2 * sizeof( SKP_int32 ) );
    }
#endif
```

```
    /* Set internal sampling frequency */
    if( psEnc->sCmn.fs_kHz != fs_kHz ) {
        /* reset part of the state */
        SKP_memset( &psEnc->sShape,          0, sizeof( SKP_Silk_shape_state_FIX
) );
        SKP_memset( &psEnc->sPrefilt,        0, sizeof( SKP_Silk_prefilter_state_
FIX ) );
        SKP_memset( &psEnc->sNSQ,            0, sizeof( SKP_Silk_nsq_state ) );
        SKP_memset( &psEnc->sPred,           0, sizeof( SKP_Silk_predict_state_FI
X ) );
        SKP_memset( psEnc->sNSQ.xq,          0, ( 2 * MAX_FRAME_LENGTH ) * sizeof
( SKP_int16 ) );
        SKP_memset( psEnc->sNSQ_LBRR.xq,     0, ( 2 * MAX_FRAME_LENGTH ) * sizeof
( SKP_int16 ) );
        SKP_memset( psEnc->sCmn.LBRR_buffer, 0, MAX_LBRR_DELAY * sizeof( SKP_SILK
_LBRR_struct ) );
#if SWITCH_TRANSITION_FILTERING
        SKP_memset( psEnc->sCmn.sLP.In_LP_State, 0, 2 * sizeof( SKP_int32 ) );
        if( psEnc->sCmn.sLP.mode == 1 ) {
            /* Begin transition phase */
            psEnc->sCmn.sLP.transition_frame_no = 1;
        } else {
            /* End transition phase */
            psEnc->sCmn.sLP.transition_frame_no = 0;
        }
#endif
        psEnc->sCmn.inputBufIx          = 0;
        psEnc->sCmn.nFramesInPayloadBuf = 0;
        psEnc->sCmn.nBytesInPayloadBuf  = 0;
        psEnc->sCmn.oldest_LBRR_idx     = 0;
        psEnc->sCmn.TargetRate_bps      = 0; /* ensures that psEnc->SNR_dB is rec
omputed */

        SKP_memset( psEnc->sPred.prev_NLSFq_Q15, 0, MAX_LPC_ORDER * sizeof( SKP_i
nt ) );

        /* Initialize non-zero parameters */
        psEnc->sCmn.prevLag                 = 100;
        psEnc->sCmn.prev_sigtype            = SIG_TYPE_UNVOICED;
        psEnc->sCmn.first_frame_after_reset = 1;
        psEnc->sPrefilt.lagPrev             = 100;
        psEnc->sShape.LastGainIndex        = 1;
        psEnc->sNSQ.lagPrev                = 100;
        psEnc->sNSQ.prev_inv_gain_Q16      = 65536;
        psEnc->sNSQ_LBRR.prev_inv_gain_Q16 = 65536;
        psEnc->sCmn.fs_kHz = fs_kHz;
        if( psEnc->sCmn.fs_kHz == 8 ) {
            psEnc->sCmn.predictLPCOrder = MIN_LPC_ORDER;
            psEnc->sCmn.psNLSF_CB[ 0 ]  = &SKP_Silk_NLSF_CB0_10;
            psEnc->sCmn.psNLSF_CB[ 1 ]  = &SKP_Silk_NLSF_CB1_10;
        } else {
            psEnc->sCmn.predictLPCOrder = MAX_LPC_ORDER;
            psEnc->sCmn.psNLSF_CB[ 0 ]  = &SKP_Silk_NLSF_CB0_16;
            psEnc->sCmn.psNLSF_CB[ 1 ]  = &SKP_Silk_NLSF_CB1_16;
        }
        psEnc->sCmn.frame_length   = SKP_SMULBB( FRAME_LENGTH_MS, fs_kHz );
```

```
        psEnc->sCmn.subfr_length   = SKP_DIV32_16( psEnc->sCmn.frame_length, NB_S
UBFR );
        psEnc->sCmn.la_pitch       = SKP_SMULBB( LA_PITCH_MS, fs_kHz );
        psEnc->sCmn.la_shape       = SKP_SMULBB( LA_SHAPE_MS, fs_kHz );
        psEnc->sPred.min_pitch_lag = SKP_SMULBB(  3, fs_kHz );
        psEnc->sPred.max_pitch_lag = SKP_SMULBB( 18, fs_kHz );
        psEnc->sPred.pitch_LPC_win_length = SKP_SMULBB( FIND_PITCH_LPC_WIN_MS, fs
_kHz );
        if( psEnc->sCmn.fs_kHz == 24 ) {
            psEnc->mu_LTP_Q8 = MU_LTP_QUANT_SWB_Q8;
        } else if( psEnc->sCmn.fs_kHz == 16 ) {
            psEnc->mu_LTP_Q8 = MU_LTP_QUANT_WB_Q8;
        } else if( psEnc->sCmn.fs_kHz == 12 ) {
            psEnc->mu_LTP_Q8 = MU_LTP_QUANT_MB_Q8;
        } else {
            psEnc->mu_LTP_Q8 = MU_LTP_QUANT_NB_Q8;
        }
        psEnc->sCmn.fs_kHz_changed = 1;

        /* Check that settings are valid */
        SKP_assert( ( psEnc->sCmn.subfr_length * NB_SUBFR ) == psEnc->sCmn.frame_
length );
    }

    /* Set encoding complexity */
    if( Complexity == 0 || LOW_COMPLEXITY_ONLY ) {
        /* Low complexity */
        psEnc->sCmn.Complexity                  = 0;
        psEnc->sCmn.pitchEstimationComplexity   = PITCH_EST_COMPLEXITY_LC_MODE;
        psEnc->pitchEstimationThreshold_Q16     = FIND_PITCH_CORRELATION_THRESHOL
D_Q16_LC_MODE;
        psEnc->sCmn.pitchEstimationLPCOrder     = 8;
        psEnc->sCmn.shapingLPCOrder             = 12;
        psEnc->sCmn.nStatesDelayedDecision      = 1;
        psEnc->NoiseShapingQuantizer            = SKP_Silk_NSQ;
        psEnc->sCmn.useInterpolatedNLSFs        = 0;
        psEnc->sCmn.LTPQuantLowComplexity       = 1;
        psEnc->sCmn.NLSF_MSVQ_Survivors         = MAX_NLSF_MSVQ_SURVIVORS_LC_MODE
;
    } else if( Complexity == 1 ) {
        /* Medium complexity */
        psEnc->sCmn.Complexity                  = 1;
        psEnc->sCmn.pitchEstimationComplexity   = PITCH_EST_COMPLEXITY_MC_MODE;
        psEnc->pitchEstimationThreshold_Q16     = FIND_PITCH_CORRELATION_THRESHOL
D_Q16_MC_MODE;
        psEnc->sCmn.pitchEstimationLPCOrder     = 12;
        psEnc->sCmn.shapingLPCOrder             = 16;
        psEnc->sCmn.nStatesDelayedDecision      = 2;
        psEnc->NoiseShapingQuantizer            = SKP_Silk_NSQ_del_dec;
        psEnc->sCmn.useInterpolatedNLSFs        = 0;
        psEnc->sCmn.LTPQuantLowComplexity       = 0;
        psEnc->sCmn.NLSF_MSVQ_Survivors         = MAX_NLSF_MSVQ_SURVIVORS_MC_MODE
;
    } else if( Complexity == 2 ) {
        /* High complexity */
```

```
        psEnc->sCmn.Complexity                  = 2;
        psEnc->sCmn.pitchEstimationComplexity   = PITCH_EST_COMPLEXITY_HC_MODE;
        psEnc->pitchEstimationThreshold_Q16     = FIND_PITCH_CORRELATION_THRESHOL
D_Q16_HC_MODE;
        psEnc->sCmn.pitchEstimationLPCOrder     = 16;
        psEnc->sCmn.shapingLPCOrder             = 16;
        psEnc->sCmn.nStatesDelayedDecision      = 4;
        psEnc->NoiseShapingQuantizer            = SKP_Silk_NSQ_del_dec;
        psEnc->sCmn.useInterpolatedNLSFs        = 1;
        psEnc->sCmn.LTPQuantLowComplexity       = 0;
        psEnc->sCmn.NLSF_MSVQ_Survivors         = MAX_NLSF_MSVQ_SURVIVORS;
    } else {
        ret = SKP_SILK_ENC_WRONG_COMPLEXITY_SETTING;
    }

    /* Dont have higher Pitch estimation LPC order than predict LPC order */
    psEnc->sCmn.pitchEstimationLPCOrder = SKP_min_int( psEnc->sCmn.pitchEstimatio
nLPCOrder, psEnc->sCmn.predictLPCOrder );

    SKP_assert( psEnc->sCmn.pitchEstimationLPCOrder <= FIND_PITCH_LPC_ORDER_MAX )
;
    SKP_assert( psEnc->sCmn.shapingLPCOrder         <= SHAPE_LPC_ORDER_MAX );
    SKP_assert( psEnc->sCmn.nStatesDelayedDecision  <= DEL_DEC_STATES_MAX );

    /* Set bitrate/coding quality */
    TargetRate_bps = SKP_min( TargetRate_bps, 100000 );
    if( psEnc->sCmn.fs_kHz == 8 ) {
        TargetRate_bps = SKP_max( TargetRate_bps, MIN_TARGET_RATE_NB_BPS );
    } else if( psEnc->sCmn.fs_kHz == 12 ) {
        TargetRate_bps = SKP_max( TargetRate_bps, MIN_TARGET_RATE_MB_BPS );
    } else if( psEnc->sCmn.fs_kHz == 16 ) {
        TargetRate_bps = SKP_max( TargetRate_bps, MIN_TARGET_RATE_WB_BPS );
    } else {
        TargetRate_bps = SKP_max( TargetRate_bps, MIN_TARGET_RATE_SWB_BPS );
    }
    if( TargetRate_bps != psEnc->sCmn.TargetRate_bps ) {
        psEnc->sCmn.TargetRate_bps = TargetRate_bps;

        /* if new TargetRate_bps, translate to SNR_dB value */
        if( psEnc->sCmn.fs_kHz == 8 ) {
            rateTable = TargetRate_table_NB;
        } else if( psEnc->sCmn.fs_kHz == 12 ) {
            rateTable = TargetRate_table_MB;
        } else if( psEnc->sCmn.fs_kHz == 16 ) {
            rateTable = TargetRate_table_WB;
        } else {
            rateTable = TargetRate_table_SWB;
        }
        for( k = 1; k < TARGET_RATE_TAB_SZ; k++ ) {
            /* find bitrate interval in table and interpolate */
            if( TargetRate_bps < rateTable[ k ] ) {
```

```
                frac_Q6 = SKP_DIV32( SKP_LSHIFT( TargetRate_bps - rateTable[ k -
1 ], 6 ), rateTable[ k ] - rateTable[ k - 1 ] );
                psEnc->SNR_dB_Q7 = SKP_LSHIFT( SNR_table_Q1[ k - 1 ], 6 ) + SKP_M
UL( frac_Q6, SNR_table_Q1[ k ] - SNR_table_Q1[ k - 1 ] );
                break;
            }
        }
    }

    /* Set packet size */
    if( ( PacketSize_ms !=  20 ) &&
        ( PacketSize_ms !=  40 ) &&
        ( PacketSize_ms !=  60 ) &&
        ( PacketSize_ms !=  80 ) &&
        ( PacketSize_ms != 100 ) ) {
        ret = SKP_SILK_ENC_PACKET_SIZE_NOT_SUPPORTED;
    } else {
        if( PacketSize_ms != psEnc->sCmn.PacketSize_ms ) {
            psEnc->sCmn.PacketSize_ms = PacketSize_ms;

            /* Packet length changes. Reset LBRR buffer */
            SKP_Silk_LBRR_reset( &psEnc->sCmn );
        }
    }

    /* Set packet loss rate measured by farend */
    if( ( PacketLoss_perc < 0 ) || ( PacketLoss_perc > 100 ) ) {
        ret = SKP_SILK_ENC_WRONG_LOSS_RATE;
    }
    psEnc->sCmn.PacketLoss_perc = PacketLoss_perc;

#if USE_LBRR
    if( INBandFec_enabled < 0 || INBandFec_enabled > 1 ) {
        ret = SKP_SILK_ENC_WRONG_INBAND_FEC_SETTING;
    }

    /* Only change settings if first frame in packet */
    if( psEnc->sCmn.nFramesInPayloadBuf == 0 ) {

        psEnc->sCmn.LBRR_enabled = INBandFec_enabled;
        if( psEnc->sCmn.fs_kHz == 8 ) {
            LBRRRate_thres_bps = INBAND_FEC_MIN_RATE_BPS - 9000;
        } else if( psEnc->sCmn.fs_kHz == 12 ) {
            LBRRRate_thres_bps = INBAND_FEC_MIN_RATE_BPS - 6000;;
        } else if( psEnc->sCmn.fs_kHz == 16 ) {
            LBRRRate_thres_bps = INBAND_FEC_MIN_RATE_BPS - 3000;
        } else {
            LBRRRate_thres_bps = INBAND_FEC_MIN_RATE_BPS;
        }
```

```
        if( psEnc->sCmn.TargetRate_bps >= LBRRRate_thres_bps ) {
            /* Set gain increase / rate reduction for LBRR usage */
            /* Coarse tuned with pesq for now. */
            /* Linear regression coefs G = 8 - 0.5 * loss */
            /* Meaning that at 16% loss main rate and redundant rate is the same,
 -> G = 0 */
            psEnc->sCmn.LBRR_GainIncreases = SKP_max_int( 8 - SKP_RSHIFT( psEnc->
sCmn.PacketLoss_perc, 1 ), 0 );

            /* Set main stream rate compensation */
            if( psEnc->sCmn.LBRR_enabled && psEnc->sCmn.PacketLoss_perc > LBRR_LO
SS_THRES ) {
                /* Tuned to give aprox same mean / weighted bitrate as no inband
FEC */
                psEnc->inBandFEC_SNR_comp_Q8 = ( 6 << 8 ) - SKP_LSHIFT( psEnc->sC
mn.LBRR_GainIncreases, 7 );
            } else {
                psEnc->inBandFEC_SNR_comp_Q8 = 0;
                psEnc->sCmn.LBRR_enabled     = 0;
            }
        } else {
            psEnc->inBandFEC_SNR_comp_Q8     = 0;
            psEnc->sCmn.LBRR_enabled         = 0;
        }
    }
#else
    psEnc->sCmn.LBRR_enabled = 0;
#endif

    /* Set DTX mode */
    if( DTX_enabled < 0 || DTX_enabled > 1 ) {
        ret = SKP_SILK_ENC_WRONG_DTX_SETTING;
    }
    psEnc->sCmn.useDTX = DTX_enabled;

    return ret;
}

/* Control low bitrate redundancy usage */
void SKP_Silk_LBRR_ctrl_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,     /* I/O  encoder state
                    */
    SKP_Silk_encoder_control_FIX    *psEncCtrl  /* I/O  encoder control
                    */
)
{
    SKP_int LBRR_usage;

    if( psEnc->sCmn.LBRR_enabled ) {
        /* Control LBRR */

        /* Usage Control based on sensitivity and packet loss caracteristics */
        /* For now only enable adding to next for active frames. Make more comple
x later */
        LBRR_usage = SKP_SILK_NO_LBRR;
        if( psEnc->speech_activity_Q8 > LBRR_SPEECH_ACTIVITY_THRES_Q8 && psEnc->s
Cmn.PacketLoss_perc > LBRR_LOSS_THRES ) { // nb! maybe multiply loss prob and spe
ech activity
```

```
            //if( psEnc->PacketLoss_burst > BURST_THRES )
            //  psEncCtrl->LBRR_usage = SKP_SILK_ADD_LBRR_TO_PLUS2;
            //} else {
                LBRR_usage = SKP_SILK_ADD_LBRR_TO_PLUS1;//SKP_SILK_NO_LBRR
            //}
        }
        psEncCtrl->sCmn.LBRR_usage = LBRR_usage;
    } else {
        psEncCtrl->sCmn.LBRR_usage = SKP_SILK_NO_LBRR;
    }
}
```

A.21. src/SKP_Silk_corrMatrix_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/***********************************************************************
 * Correlation Matrix Computations for LS estimate.
 ***********************************************************************/

#include "SKP_Silk_main_FIX.h"
```

```
/* Calculates correlation vector X'*t */
void SKP_Silk_corrVector_FIX(
    const SKP_int16                 *x,         /* I    x vector [L + order - 1]
used to form data matrix X */
    const SKP_int16                 *t,         /* I    target vector [L]
                        */
    const SKP_int                   L,          /* I    Length of vectors
                        */
    const SKP_int                   order,      /* I    Max lag for correlation
                        */
    SKP_int32                       *Xt,        /* O    Pointer to X'*t correlati
on vector [order]       */
    const SKP_int                   rshifts     /* I    Right shifts of correlati
ons                     */
)
{
    SKP_int         lag, i;
    const SKP_int16 *ptr1, *ptr2;
    SKP_int32       inner_prod;

    ptr1 = &x[ order - 1 ]; /* Points to first sample of column 0 of X: X[:,0] */
    ptr2 = t;
    /* Calculate X'*t */
    if( rshifts > 0 ) {
        /* Right shifting used */
        for( lag = 0; lag < order; lag++ ) {
            inner_prod = 0;
            for( i = 0; i < L; i++ ) {
                inner_prod += SKP_RSHIFT32( SKP_SMULBB( ptr1[ i ], ptr2[i] ), rsh
ifts );
            }
            Xt[ lag ] = inner_prod; /* X[:,lag]'*t */
            ptr1--; /* Go to next column of X */
        }
    } else {
        SKP_assert( rshifts == 0 );
        for( lag = 0; lag < order; lag++ ) {
            Xt[ lag ] = SKP_Silk_inner_prod_aligned( ptr1, ptr2, L ); /* X[:,lag]
'*t */
            ptr1--; /* Go to next column of X */
        }
    }
}

/* Calculates correlation matrix X'*X */
void SKP_Silk_corrMatrix_FIX(
    const SKP_int16                 *x,         /* I    x vector [L + order - 1]
used to form data matrix X */
    const SKP_int                   L,          /* I    Length of vectors
                        */
    const SKP_int                   order,      /* I    Max lag for correlation
                        */
    SKP_int32                       *XX,        /* O    Pointer to X'*X correlati
on matrix [ order x order ]*/
    SKP_int                         *rshifts    /* I/O  Right shifts of correlati
ons                     */
)
{
    SKP_int         i, j, lag, rshifts_local, head_room_rshifts;
    SKP_int32       energy;
    const SKP_int16 *ptr1, *ptr2;
```

```
    /* Calculate energy to find shift used to fit in 32 bits */
    SKP_Silk_sum_sqr_shift( &energy, &rshifts_local, x, L + order - 1 );
    /* Add shifts to get the wanted head room */

    head_room_rshifts = SKP_max( LTP_CORRS_HEAD_ROOM - SKP_Silk_CLZ32( energy ),
0 );

    energy = SKP_RSHIFT32( energy, head_room_rshifts );
    rshifts_local += head_room_rshifts;

    /* Calculate energy of first column (0) of X: X[:,0]'*X[:,0] */
    /* Remove contribution of first order - 1 samples */
    for( i = 0; i < order - 1; i++ ) {
        energy -= SKP_RSHIFT32( SKP_SMULBB( x[ i ], x[ i ] ), rshifts_local );
    }
    if( rshifts_local < *rshifts ) {
        /* Adjust energy */
        energy = SKP_RSHIFT32( energy, *rshifts - rshifts_local );
        rshifts_local = *rshifts;
    }

    /* Calculate energy of remaining columns of X: X[:,j]'*X[:,j] */
    /* Fill out the diagonal of the correlation matrix */
    matrix_ptr( XX, 0, 0, order ) = energy;
    ptr1 = &x[ order - 1 ]; /* First sample of column 0 of X */
    for( j = 1; j < order; j++ ) {
        energy = SKP_SUB32( energy, SKP_RSHIFT32( SKP_SMULBB( ptr1[ L - j ], ptr1
[ L - j ] ), rshifts_local ) );
        energy = SKP_ADD32( energy, SKP_RSHIFT32( SKP_SMULBB( ptr1[ -j ], ptr1[ -
j ] ), rshifts_local ) );
        matrix_ptr( XX, j, j, order ) = energy;
    }

    ptr2 = &x[ order - 2 ]; /* First sample of column 1 of X */
    /* Calculate the remaining elements of the correlation matrix */
    if( rshifts_local > 0 ) {
        /* Right shifting used */
        for( lag = 1; lag < order; lag++ ) {
            /* Inner product of column 0 and column lag: X[:,0]'*X[:,lag] */
            energy = 0;
            for( i = 0; i < L; i++ ) {
                energy += SKP_RSHIFT32( SKP_SMULBB( ptr1[ i ], ptr2[i] ), rshifts
_local );
            }
            /* Calculate remaining off diagonal: X[:,j]'*X[:,j + lag] */
            matrix_ptr( XX, lag, 0, order ) = energy;
            matrix_ptr( XX, 0, lag, order ) = energy;
            for( j = 1; j < ( order - lag ); j++ ) {
                energy = SKP_SUB32( energy, SKP_RSHIFT32( SKP_SMULBB( ptr1[ L - j
 ], ptr2[ L - j ] ), rshifts_local ) );
                energy = SKP_ADD32( energy, SKP_RSHIFT32( SKP_SMULBB( ptr1[ -j ],
 ptr2[ -j ] ), rshifts_local ) );
                matrix_ptr( XX, lag + j, j, order ) = energy;
                matrix_ptr( XX, j, lag + j, order ) = energy;
```

```
            }
            ptr2--; /* Update pointer to first sample of next column (lag) in X *
/
        }
    } else {
        for( lag = 1; lag < order; lag++ ) {
            /* Inner product of column 0 and column lag: X[:,0]'*X[:,lag] */
            energy = SKP_Silk_inner_prod_aligned( ptr1, ptr2, L );
            matrix_ptr( XX, lag, 0, order ) = energy;
            matrix_ptr( XX, 0, lag, order ) = energy;
            /* Calculate remaining off diagonal: X[:,j]'*X[:,j + lag] */
            for( j = 1; j < ( order - lag ); j++ ) {
                energy = SKP_SUB32( energy, SKP_SMULBB( ptr1[ L - j ], ptr2[ L -
j ] ) );
                energy = SKP_SMLABB( energy, ptr1[ -j ], ptr2[ -j ] );
                matrix_ptr( XX, lag + j, j, order ) = energy;
                matrix_ptr( XX, j, lag + j, order ) = energy;
            }
            ptr2--;/* Update pointer to first sample of next column (lag) in X */
        }
    }
    *rshifts = rshifts_local;
}
```

A.22. src/SKP_Silk_create_init_destroy.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
```

```
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"


/************************/
/* Init Decoder State   */
/************************/
SKP_int SKP_Silk_init_decoder(
    SKP_Silk_decoder_state      *psDec              /* I/O  Decoder state pointer
                        */
)
{
    SKP_memset( psDec, 0, sizeof( SKP_Silk_decoder_state ) );
    /* Set sampling rate to 24 kHz, and init non-zero values */
    SKP_Silk_decoder_set_fs( psDec, 24 );

    /* Used to deactivate e.g. LSF interpolation and fluctuation reduction */
    psDec->first_frame_after_reset = 1;
    psDec->prev_inv_gain_Q16 = 65536;

    /* Reset CNG state */
    SKP_Silk_CNG_Reset( psDec );

    SKP_Silk_PLC_Reset( psDec );

    psDec->bitstream_v = USE_BIT_STREAM_V;

    return(0);
}
```

A.23. src/SKP_Silk_dec_API.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
```

```
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_SDK_API.h"
#include "SKP_Silk_main_FIX.h"

/*********************/
/* Decoder functions */
/*********************/

SKP_int SKP_Silk_SDK_Get_Decoder_Size( SKP_int32 *decSizeBytes )
{
    SKP_int ret = 0;

    *decSizeBytes = sizeof( SKP_Silk_decoder_state );

    return ret;
}

/* Reset decoder state */
SKP_int SKP_Silk_SDK_InitDecoder(
    void* decState                                      /* I/O: State
                                */
)
{
    SKP_int ret = 0;
    SKP_Silk_decoder_state *struc;

    struc = (SKP_Silk_decoder_state *)decState;

    ret  = SKP_Silk_init_decoder( struc );

    return ret;
}
```

```
/* Decode a frame */
SKP_int SKP_Silk_SDK_Decode(
    void*                               decState,       /* I/O: State
                                */
    SKP_SILK_SDK_DecControlStruct*      decControl,     /* I/O: Control structure
                                */
    SKP_int                             lostFlag,       /* I:   0: no loss, 1 los
s                               */
    const SKP_uint8                     *inData,        /* I:   Encoded input vec
tor                             */
    const SKP_int                       nBytesIn,       /* I:   Number of input B
ytes                            */
    SKP_int16                           *samplesOut,    /* O:   Decoded output sp
eech vector                     */
    SKP_int16                           *nSamplesOut    /* I/O: Number of samples
 (vector/decoded)               */
)
{
    SKP_int ret = 0, used_bytes, prev_fs_kHz;
    SKP_Silk_decoder_state *psDec;

    psDec = (SKP_Silk_decoder_state *)decState;

    /**********************************/
    /* Test if first frame in payload */
    /**********************************/
    if( psDec->moreInternalDecoderFrames == 0 ) {
        /* First Frame in Payload */
        psDec->nFramesDecoded = 0;  /* Used to count frames in packet */
    }

    if( psDec->moreInternalDecoderFrames == 0 &&    /* First frame in packet    *
/
        lostFlag == 0 &&                            /* Not packet loss          *
/
        nBytesIn > MAX_ARITHM_BYTES ) {             /* Too long payload         *
/
            /* Avoid trying to decode a too large packet */
            lostFlag = 1;
            ret = SKP_SILK_DEC_PAYLOAD_TOO_LARGE;
    }

    /* Save previous sample frequency */
    prev_fs_kHz = psDec->fs_kHz;

    /* Call decoder for one frame */
    ret += SKP_Silk_decode_frame( psDec, samplesOut, nSamplesOut, inData, nBytesI
n,
            lostFlag, &used_bytes );

    if( used_bytes ) { /* Only Call if not a packet loss */
        if( psDec->nBytesLeft > 0 && psDec->FrameTermination == SKP_SILK_MORE_FRA
MES && psDec->nFramesDecoded < 5 ) {
            /* We have more frames in the Payload */
            psDec->moreInternalDecoderFrames = 1;
        } else {
            /* Last frame in Payload */
            psDec->moreInternalDecoderFrames = 0;
            psDec->nFramesInPacket = psDec->nFramesDecoded;
```

```
            /* Track inband FEC usage */
            if( psDec->vadFlag == VOICE_ACTIVITY ) {
                if( psDec->FrameTermination == SKP_SILK_LAST_FRAME ) {
                    psDec->no_FEC_counter++;
                    if( psDec->no_FEC_counter > NO_LBRR_THRES ) {
                        psDec->inband_FEC_offset = 0;
                    }
                } else if( psDec->FrameTermination == SKP_SILK_LBRR_VER1 ) {
                    psDec->inband_FEC_offset = 1; /* FEC info with 1 packet delay
 */
                    psDec->no_FEC_counter    = 0;
                } else if( psDec->FrameTermination == SKP_SILK_LBRR_VER2 ) {
                    psDec->inband_FEC_offset = 2; /* FEC info with 2 packets dela
y */
                    psDec->no_FEC_counter    = 0;
                }
            }
        }
    }

    if( psDec->fs_kHz * 1000 > decControl->sampleRate ) {
        ret = SKP_SILK_DEC_WRONG_SAMPLING_FREQUENCY;
    }

    /* Do any resampling if needed */
    if( psDec->fs_kHz * 1000 != decControl->sampleRate ) {
        SKP_int16 samplesOut_tmp[ 2 * MAX_FRAME_LENGTH ];
        SKP_int32 scratch[        3 * MAX_FRAME_LENGTH ];

        /* Copy to a tmpbuffer as the resampling writes to samplesOut */
        memcpy( samplesOut_tmp, samplesOut, *nSamplesOut * sizeof( SKP_int16 ) );

        /* Clear resampler state when switching internal sampling frequency */
        if( prev_fs_kHz != psDec->fs_kHz ) {
            SKP_memset( psDec->resampleState, 0, sizeof( psDec->resampleState ) )
;
        }

        if( psDec->fs_kHz == 16 && decControl->sampleRate == 24000 ) {
            /* Resample from 16 kHz to 24 kHz */
            SKP_Silk_resample_3_2( samplesOut, psDec->resampleState, samplesOut_t
mp, *nSamplesOut );
        } else if( psDec->fs_kHz == 12 && decControl->sampleRate == 24000 ) {
            /* Resample from 12 kHz to 24 kHz */
            SKP_Silk_resample_2_1_coarse( samplesOut_tmp, psDec->resampleState, s
amplesOut, scratch, *nSamplesOut );
        } else if( psDec->fs_kHz == 8 && decControl->sampleRate == 24000 ) {
            /* Resample from 8 kHz to 24 kHz */
            SKP_Silk_resample_3_1( samplesOut, psDec->resampleState, samplesOut_t
mp, *nSamplesOut );
        } else if( psDec->fs_kHz == 12 && decControl->sampleRate == 16000 ) {
            /* Resample from 12 kHz to 16 kHz */
            SKP_Silk_resample_4_3( samplesOut, psDec->resampleState, samplesOut_t
mp, *nSamplesOut );
        } else if( psDec->fs_kHz == 8 && decControl->sampleRate == 16000 ) {
```

```
            /* Resample from 8 kHz to 16 kHz */
            SKP_Silk_resample_2_1_coarse( samplesOut_tmp, psDec->resampleState, s
amplesOut, scratch, *nSamplesOut );
        } else if( psDec->fs_kHz == 8 && decControl->sampleRate == 12000 ) {
            /* Resample from 8 kHz to 12 kHz */
            SKP_Silk_resample_3_2( samplesOut, psDec->resampleState, samplesOut_t
mp, *nSamplesOut );
        }

        *nSamplesOut = SKP_DIV32( ( SKP_int32 )*nSamplesOut * decControl->sampleR
ate, psDec->fs_kHz * 1000 );
    }

    /* Copy all parameters that are needed out of internal structure to the contr
ol stucture */
    decControl->frameSize                = ( SKP_int )psDec->frame_length;
    decControl->framesPerPacket          = ( SKP_int )psDec->nFramesInPacket;
    decControl->inBandFECOffset          = ( SKP_int )psDec->inband_FEC_offset;
    decControl->moreInternalDecoderFrames = ( SKP_int )psDec->moreInternalDecoder
Frames;

    return ret;
}

/* Function to find LBRR information in a packet */
void SKP_Silk_SDK_search_for_LBRR(
    void                                *decState,      /* I:   Decoder state, to
 select bitstream version only */
    const SKP_uint8                     *inData,        /* I:   Encoded input vec
tor                           */
    const SKP_int16                     nBytesIn,       /* I:   Number of input B
ytes                          */
    SKP_int                             lost_offset,    /* I:   Offset from lost
packet                        */
    SKP_uint8                           *LBRRData,      /* O:   LBRR payload
                              */
    SKP_int16                           *nLBRRBytes     /* O:   Number of LBRR By
tes                           */
)
{
    SKP_Silk_decoder_state   *psDec;
    SKP_Silk_decoder_state   sDec; // Local decoder state to avoid interfering wi
th running decoder */
    SKP_Silk_decoder_control sDecCtrl;
    SKP_int i, TempQ[ MAX_FRAME_LENGTH ];

    psDec = ( SKP_Silk_decoder_state * )decState;

    if( lost_offset < 1 || lost_offset > MAX_LBRR_DELAY ) {
        /* No useful FEC in this packet */
        *nLBRRBytes = 0;
        return;
    }

    sDec.nFramesDecoded = 0;
    sDec.fs_kHz         = 0; /* Force update parameters LPC_order etc */
    SKP_memset( sDec.prevNLSF_Q15, 0, MAX_LPC_ORDER * sizeof( SKP_int ) );
    SKP_Silk_range_dec_init( &sDec.sRC, inData, ( SKP_int32 )nBytesIn );

    if( psDec->bitstream_v == BIT_STREAM_V4 ) { /* Silk_v4 payload */
```

```
        /* Decode all parameter indices for the whole packet*/
        SKP_Silk_decode_indices_v4( &sDec );

        /* Is there usable LBRR in this packet */
        *nLBRRBytes = 0;
        if( ( sDec.FrameTermination - 1 ) & lost_offset && sDec.FrameTermination
> 0 && sDec.nBytesLeft >= 0 ) {
            /* The wanted FEC is present in the packet */
            for( i = 0; i < sDec.nFramesInPacket; i++ ) {
                SKP_Silk_decode_parameters_v4( &sDec, &sDecCtrl, TempQ, 0 );

                if( sDec.nBytesLeft <= 0 || sDec.sRC.error ) {
                    /* Corrupt stream */
                    LBRRData = NULL;
                    *nLBRRBytes = 0;
                    break;
                } else {
                    sDec.nFramesDecoded++;
                }
            }

            if( LBRRData != NULL ) {
                /* The wanted FEC is present in the packet */
                *nLBRRBytes = sDec.nBytesLeft;
                SKP_memcpy( LBRRData, &inData[ nBytesIn - sDec.nBytesLeft ], sDec
.nBytesLeft * sizeof( SKP_uint8 ) );
            }
        }
    } else { /* Silk_v3 payload */
        while(1) {
            SKP_Silk_decode_parameters( &sDec, &sDecCtrl, TempQ, 0 );

            if( sDec.sRC.error ) {
                /* Corrupt stream */
                *nLBRRBytes = 0;
                return;
            };

            if( ( sDec.FrameTermination - 1 ) & lost_offset && sDec.FrameTerminat
ion > 0 && sDec.nBytesLeft >= 0 ) {
                /* The wanted FEC is present in the packet */
                *nLBRRBytes = sDec.nBytesLeft;
                SKP_memcpy( LBRRData, &inData[ nBytesIn - sDec.nBytesLeft ], sDec
.nBytesLeft * sizeof( SKP_uint8 ) );
                break;
            }
            if( sDec.nBytesLeft > 0 && sDec.FrameTermination == SKP_SILK_MORE_FRA
MES ) {
                sDec.nFramesDecoded++;
            } else {
                LBRRData = NULL;
                *nLBRRBytes = 0;
                break;
```

```
            }
        }
    }
}

/* Getting type of content for a packet */
void SKP_Silk_SDK_get_TOC(
    void                                *decState,      /* I/O: Decoder state, to
 select bitstream version only */
    const SKP_uint8                     *inData,        /* I:   Encoded input vec
tor                                 */
    const SKP_int16                     nBytesIn,       /* I:   Number of input b
ytes                                */
    SKP_Silk_TOC_struct                 *Silk_TOC       /* O:   Type of content
                                */
)
{
    SKP_Silk_decoder_state      *psDec;
    SKP_Silk_decoder_state      sDec; // Local Decoder state to avoid interfering
 with running decoder */
    SKP_Silk_decoder_control    sDecCtrl;
    SKP_int i, TempQ[ MAX_FRAME_LENGTH ];

    psDec = (SKP_Silk_decoder_state *)decState;

    sDec.nFramesDecoded = 0;
    sDec.fs_kHz         = 0; /* Force update parameters LPC_order etc */
    SKP_Silk_range_dec_init( &sDec.sRC, inData, ( SKP_int32 )nBytesIn );

    if( psDec->bitstream_v == BIT_STREAM_V4 ) { /* Silk_v4 payload */
        /* Decode all parameter indices for the whole packet*/
        SKP_Silk_decode_indices_v4( &sDec );

        if( sDec.nFramesInPacket > SILK_MAX_FRAMES_PER_PACKET || sDec.sRC.error )
 {
            /* Corrupt packet */
            SKP_memset( Silk_TOC, 0, sizeof( SKP_Silk_TOC_struct ) );
            Silk_TOC->corrupt = 1;
        } else {
            Silk_TOC->corrupt = 0;
            Silk_TOC->framesInPacket = sDec.nFramesInPacket;
            Silk_TOC->fs_kHz         = sDec.fs_kHz;
            if( sDec.FrameTermination == SKP_SILK_LAST_FRAME ) {
                Silk_TOC->inbandLBRR = sDec.FrameTermination;
            } else {
                Silk_TOC->inbandLBRR = sDec.FrameTermination - 1;
            }
            /* Copy data */
            for( i = 0; i < sDec.nFramesInPacket; i++ ) {
                Silk_TOC->vadFlags[ i ]     = sDec.vadFlagBuf[ i ];
                Silk_TOC->sigtypeFlags[ i ] = sDec.sigtype[ i ];
            }
        }
    } else { /* Silk_v3 payload */
```

```
        Silk_TOC->corrupt = 0;
        while( 1 ) {
            SKP_Silk_decode_parameters( &sDec, &sDecCtrl, TempQ, 0 );

            Silk_TOC->vadFlags[     sDec.nFramesDecoded ] = sDec.vadFlag;
            Silk_TOC->sigtypeFlags[ sDec.nFramesDecoded ] = sDecCtrl.sigtype;

            if( sDec.sRC.error ) {
                /* Corrupt stream */
                Silk_TOC->corrupt = 1;
                break;
            };

            if( sDec.nBytesLeft > 0 && sDec.FrameTermination == SKP_SILK_MORE_FRA
MES ) {
                sDec.nFramesDecoded++;
            } else {
                break;
            }
        }
        if( Silk_TOC->corrupt || sDec.FrameTermination == SKP_SILK_MORE_FRAMES ||
            sDec.nFramesInPacket > SILK_MAX_FRAMES_PER_PACKET ) {
            /* Corrupt packet */
            SKP_memset( Silk_TOC, 0, sizeof( SKP_Silk_TOC_struct ) );
            Silk_TOC->corrupt = 1;
        } else {
            Silk_TOC->framesInPacket = sDec.nFramesDecoded;
            Silk_TOC->fs_kHz         = sDec.fs_kHz;
            if( sDec.FrameTermination == SKP_SILK_LAST_FRAME ) {
                Silk_TOC->inbandLBRR = sDec.FrameTermination;
            } else {
                Silk_TOC->inbandLBRR = sDec.FrameTermination - 1;
            }
        }
    }
}

/*************************/
/* Get the version number */
/*************************/
/* Return a pointer to string specifying the version */
const char *SKP_Silk_SDK_get_version()
{
    static const char version[] = "1.0.2";
    return version;
}
```

A.24. src/SKP_Silk_decode_core.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

/*******************************************************/
/* Core decoder. Performs inverse NSQ operation LTP + LPC */
/*******************************************************/
void SKP_Silk_decode_core(
    SKP_Silk_decoder_state      *psDec,                             /* I/O  Decod
er state                */
    SKP_Silk_decoder_control    *psDecCtrl,                         /* I    Decod
er control              */
    SKP_int16                   xq[],                               /* O    Decod
ed speech               */
    const SKP_int               q[ MAX_FRAME_LENGTH ]               /* I    Pulse
 signal                 */
)
{
    SKP_int     i, k, lag = 0, start_idx, NLSF_interpolation_flag, sigtype, LTP_s
cale_Q14;
    SKP_int16   *A_Q12, *B_Q14, *pxq, A_Q12_tmp[ MAX_LPC_ORDER ];
    SKP_int16   sLTP[ MAX_FRAME_LENGTH ];
    SKP_int32   Gain_Q16, *pred_lag_ptr, *pexc_Q10, *pres_Q10, LTP_pred_Q14, LPC_
pred_Q10;
    SKP_int32   rand_seed, offset_Q10, dither;
    SKP_int32   vec_Q10[ MAX_FRAME_LENGTH / NB_SUBFR ], Atmp;
    SKP_int32   inv_gain_Q16, inv_gain_Q32, gain_adj_Q16, FiltState[ MAX_LPC_ORDE
R ];
    SKP_assert( psDec->prev_inv_gain_Q16 != 0 );
```

```
    offset_Q10 = SKP_Silk_Quantization_Offsets_Q10[ psDecCtrl->sigtype ][ psDecCt
rl->QuantOffsetType ];

    if( psDecCtrl->NLSFInterpCoef_Q2 < ( 1 << 2 ) ) {
        NLSF_interpolation_flag = 1;
    } else {
        NLSF_interpolation_flag = 0;
    }


    /* Decode excitation */
    rand_seed = psDecCtrl->Seed;
    for( i = 0; i < psDec->frame_length; i++ ) {
        rand_seed = SKP_RAND( rand_seed );
        /* dither = rand_seed < 0 ? 0xFFFFFFFF : 0; */
        dither = SKP_RSHIFT( rand_seed, 31 );

        psDec->exc_Q10[ i ] = SKP_LSHIFT( ( SKP_int32 )q[ i ], 10 ) + offset_Q10;
        psDec->exc_Q10[ i ] = ( psDec->exc_Q10[ i ] ^ dither ) - dither;

        rand_seed += q[ i ];
    }


    pexc_Q10 = psDec->exc_Q10;
    pres_Q10 = psDec->res_Q10;
    pxq      = &psDec->outBuf[ psDec->frame_length ];
    psDec->sLTP_buf_idx = psDec->frame_length;
    /* Loop over subframes */
    for( k = 0; k < NB_SUBFR; k++ ) {
        A_Q12 = psDecCtrl->PredCoef_Q12[ k >> 1 ];

        /* Preload LPC coeficients to array on stack. Gives small performance gai
n */
        SKP_memcpy( A_Q12_tmp, A_Q12, psDec->LPC_order * sizeof( SKP_int16 ) );
        B_Q14         = &psDecCtrl->LTPCoef_Q14[ k * LTP_ORDER ];
        Gain_Q16      = psDecCtrl->Gains_Q16[ k ];
        LTP_scale_Q14 = psDecCtrl->LTP_scale_Q14;
        sigtype       = psDecCtrl->sigtype;

        inv_gain_Q16  = SKP_DIV32( SKP_int32_MAX, SKP_RSHIFT( Gain_Q16, 1 ) );
        inv_gain_Q16  = SKP_min( inv_gain_Q16, SKP_int16_MAX );

        /* Calculate Gain adjustment factor */
        gain_adj_Q16 = ( SKP_int32 )1 << 16;
        if( inv_gain_Q16 != psDec->prev_inv_gain_Q16 ) {
            gain_adj_Q16 =  SKP_DIV32_varQ( inv_gain_Q16, psDec->prev_inv_gain_Q1
6, 16 );
        }

        /* Avoid abrupt transition from voiced PLC to unvoiced normal decoding */
```

```
        if( psDec->lossCnt && psDec->prev_sigtype == SIG_TYPE_VOICED &&
            psDecCtrl->sigtype == SIG_TYPE_UNVOICED && k < ( NB_SUBFR >> 1 ) ) {

            SKP_memset( B_Q14, 0, LTP_ORDER * sizeof( SKP_int16 ) );
            B_Q14[ LTP_ORDER/2 ] = ( SKP_int16 )1 << 12; /* 0.25 */

            sigtype = SIG_TYPE_VOICED;
            psDecCtrl->pitchL[ k ] = psDec->lagPrev;
            LTP_scale_Q14 = ( SKP_int )1 << 14;
        }
        if( sigtype == SIG_TYPE_VOICED ) {
            /* Voiced */

            lag = psDecCtrl->pitchL[ k ];
            /* Re-whitening */
            if( ( k & ( 3 - SKP_LSHIFT( NLSF_interpolation_flag, 1 ) ) ) == 0 ) {
                /* Rewhiten with new A coefs */
                start_idx = psDec->frame_length - lag - psDec->LPC_order - LTP_OR
DER / 2;
                start_idx = SKP_LIMIT( start_idx, 0, psDec->frame_length - psDec-
>LPC_order );

                SKP_Silk_MA_Prediction( &psDec->outBuf[ start_idx + k * ( psDec->
frame_length >> 2 ) ],
                    A_Q12, FiltState, sLTP + start_idx, psDec->frame_length - sta
rt_idx, psDec->LPC_order );

                /* After rewhitening the LTP state is unscaled */
                inv_gain_Q32 = SKP_LSHIFT( inv_gain_Q16, 16 );
                if( k == 0 ) {
                    /* Do LTP downscaling */
                    inv_gain_Q32 = SKP_LSHIFT( SKP_SMULWB( inv_gain_Q32, psDecCtr
l->LTP_scale_Q14 ), 2 );
                }
                for( i = 0; i < (lag + LTP_ORDER/2); i++ ) {
                    psDec->sLTP_Q16[ psDec->sLTP_buf_idx - i - 1 ] = SKP_SMULWB(
inv_gain_Q32, sLTP[ psDec->frame_length - i - 1 ] );
                }
            } else {
                /* Update LTP state when Gain changes */
                if( gain_adj_Q16 != ( SKP_int32 )1 << 16 ) {
                    for( i = 0; i < ( lag + LTP_ORDER / 2 ); i++ ) {
                        psDec->sLTP_Q16[ psDec->sLTP_buf_idx - i - 1 ] = SKP_SMUL
WW( gain_adj_Q16, psDec->sLTP_Q16[ psDec->sLTP_buf_idx - i - 1 ] );
                    }
                }
            }
        }

        /* Scale short term state */
        for( i = 0; i < MAX_LPC_ORDER; i++ ) {
            psDec->sLPC_Q14[ i ] = SKP_SMULWW( gain_adj_Q16, psDec->sLPC_Q14[ i ]
 );
        }

        /* Save inv_gain */
```

```
        SKP_assert( inv_gain_Q16 != 0 );
        psDec->prev_inv_gain_Q16 = inv_gain_Q16;


        /* Long-term prediction */
        if( sigtype == SIG_TYPE_VOICED ) {
            /* Setup pointer */
            pred_lag_ptr = &psDec->sLTP_Q16[ psDec->sLTP_buf_idx - lag + LTP_ORDE
R / 2 ];
            for( i = 0; i < psDec->subfr_length; i++ ) {
                /* Unrolled loop */
                LTP_pred_Q14 = SKP_SMULWB(               pred_lag_ptr[  0 ], B_Q1
4[ 0 ] );
                LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -1 ], B_Q1
4[ 1 ] );
                LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -2 ], B_Q1
4[ 2 ] );
                LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -3 ], B_Q1
4[ 3 ] );
                LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -4 ], B_Q1
4[ 4 ] );
                pred_lag_ptr++;

                /* Generate LPC residual */
                pres_Q10[ i ] = SKP_ADD32( pexc_Q10[ i ], SKP_RSHIFT_ROUND( LTP_p
red_Q14, 4 ) );

                /* Update states */
                psDec->sLTP_Q16[ psDec->sLTP_buf_idx ] = SKP_LSHIFT( pres_Q10[ i
], 6 );
                psDec->sLTP_buf_idx++;
            }
        } else {
            SKP_memcpy( pres_Q10, pexc_Q10, psDec->subfr_length * sizeof( SKP_int
32 ) );
        }


        /* Short term prediction */
        /* NOTE: the code below loads two int16 values in an int32, and multiplie
s each using the   */
        /* SMLAWB and SMLAWT instructions. On a big-endian CPU the two int16 vari
ables would be     */
        /* loaded in reverse order and the code will give the wrong result. In th
at case swapping   */
        /* the SMLAWB and SMLAWT instructions should solve the problem.
                   */
        if( psDec->LPC_order == 16 ) {
            for( i = 0; i < psDec->subfr_length; i++ ) {
                /* unrolled */
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 0 ] );    /* read two coeffic
ients at once */
                LPC_pred_Q10 = SKP_SMULWB(               psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  1 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  2 ], Atmp );
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 2 ] );
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  3 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  4 ], Atmp );
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 4 ] );
```

```
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  5 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  6 ], Atmp );
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 6 ] );
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  7 ], Atmp );
```

```
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  8 ], Atmp );
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 8 ] );
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  9 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i - 10 ], Atmp );
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 10 ] );
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i - 11 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i - 12 ], Atmp );
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 12 ] );
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i - 13 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i - 14 ], Atmp );
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 14 ] );
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i - 15 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i - 16 ], Atmp );

                /* Add prediction to LPC residual */
                vec_Q10[ i ] = SKP_ADD32( pres_Q10[ i ], LPC_pred_Q10 );

                /* Update states */
                psDec->sLPC_Q14[ MAX_LPC_ORDER + i ] = SKP_LSHIFT( vec_Q10[ i ],
4 );
            }
        } else {
            SKP_assert( psDec->LPC_order == 10 );
            for( i = 0; i < psDec->subfr_length; i++ ) {
                /* unrolled */
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 0 ] );    /* read two coeffic
ients at once */
                LPC_pred_Q10 = SKP_SMULWB(               psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  1 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  2 ], Atmp );
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 2 ] );
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  3 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  4 ], Atmp );
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 4 ] );
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  5 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  6 ], Atmp );
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 6 ] );
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  7 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  8 ], Atmp );
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 8 ] );
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  9 ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i - 10 ], Atmp );

                /* Add prediction to LPC residual */
```

```
                vec_Q10[ i ] = SKP_ADD32( pres_Q10[ i ], LPC_pred_Q10 );

                /* Update states */
                psDec->sLPC_Q14[ MAX_LPC_ORDER + i ] = SKP_LSHIFT( vec_Q10[ i ],
4 );
            }
        }
```

```
        /* Scale with Gain */
        for( i = 0; i < psDec->subfr_length; i++ ) {
            pxq[ i ] = ( SKP_int16 )SKP_SAT16( SKP_RSHIFT_ROUND( SKP_SMULWW( vec_
Q10[ i ], Gain_Q16 ), 10 ) );
        }

        /* Update LPC filter state */
        SKP_memcpy( psDec->sLPC_Q14, &psDec->sLPC_Q14[ psDec->subfr_length ], MAX
_LPC_ORDER * sizeof( SKP_int32 ) );
        pexc_Q10 += psDec->subfr_length;
        pres_Q10 += psDec->subfr_length;
        pxq      += psDec->subfr_length;
    }

    /* Copy to output */
    SKP_memcpy( xq, &psDec->outBuf[ psDec->frame_length ], psDec->frame_length *
sizeof( SKP_int16 ) );

}
```

A.25. src/SKP_Silk_decode_frame.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/
```

```
#include "SKP_Silk_main.h"
#include "SKP_Silk_PLC.h"

/****************/
/* Decode frame */
/****************/
SKP_int SKP_Silk_decode_frame(
    SKP_Silk_decoder_state          *psDec,             /* I/O  Pointer to Silk d
ecoder state                */
    SKP_int16                       pOut[],             /* O    Pointer to output
 speech frame               */
    SKP_int16                       *pN,                /* O    Pointer to size o
f output frame              */
    const SKP_uint8                 pCode[],            /* I    Pointer to payloa
d                           */
    const SKP_int                   nBytes,             /* I    Payload length
                            */
    SKP_int                         action,             /* I    Action from Jitte
r Buffer                    */
    SKP_int                         *decBytes           /* O    Used bytes to dec
ode this frame              */
)
{
    SKP_Silk_decoder_control sDecCtrl;
    SKP_int         L, fs_Khz_old, LPC_order_old, ret = 0;
    SKP_int         Pulses[ MAX_FRAME_LENGTH ];


    L = psDec->frame_length;
    sDecCtrl.LTP_scale_Q14 = 0;

    /* Safety checks */
    SKP_assert( L > 0 && L <= MAX_FRAME_LENGTH );

    /*****************************************/
    /* Decode Frame if packet is not lost  */
    /*****************************************/
    *decBytes = 0;
    if( action == 0 ) {
        /*****************************************/
        /* Initialize arithmetic coder           */
        /*****************************************/
        fs_Khz_old    = psDec->fs_kHz;
        LPC_order_old = psDec->LPC_order;
        if( psDec->nFramesDecoded == 0 ) {
            /* Initialize range decoder state */
            SKP_Silk_range_dec_init( &psDec->sRC, pCode, nBytes );

            if( psDec->bitstream_v == BIT_STREAM_V4 ) {
                SKP_Silk_decode_indices_v4( psDec );
            }
        }

        /*****************************************/
        /* Decode parameters and pulse signal    */
```

```
        /****************************************/
        if( psDec->bitstream_v == BIT_STREAM_V4 ) {
            SKP_Silk_decode_parameters_v4( psDec, &sDecCtrl, Pulses, 1 );
        } else {
            SKP_Silk_decode_parameters( psDec, &sDecCtrl, Pulses, 1 );
        }


        if( psDec->sRC.error ) {
            psDec->nBytesLeft = 0;

            action              = 1; /* PLC operation */
            psDec->fs_kHz       = fs_Khz_old;    /* revert fs if changed in decod
e_parameters */
            psDec->LPC_order    = LPC_order_old; /* revert lpc_order if changed i
n decode_parameters */
            psDec->frame_length = fs_Khz_old * FRAME_LENGTH_MS;
            psDec->subfr_length = fs_Khz_old * FRAME_LENGTH_MS / NB_SUBFR;

            /* Avoid crashing */
            *decBytes = psDec->sRC.bufferLength;

            if( psDec->sRC.error == RANGE_CODER_DEC_PAYLOAD_TOO_LONG ) {
                ret = SKP_SILK_DEC_PAYLOAD_TOO_LARGE;
            } else {
                ret = SKP_SILK_DEC_PAYLOAD_ERROR;
            }
        } else {
            *decBytes = psDec->sRC.bufferLength - psDec->nBytesLeft;
            psDec->nFramesDecoded++;

            /* Update lengths. Sampling frequency could have changed */
            L = psDec->frame_length;

            /*******************************************************/
            /* Run inverse NSQ                                     */
            /*******************************************************/
            SKP_Silk_decode_core( psDec, &sDecCtrl, pOut, Pulses );

            /*******************************************************/
            /* Update PLC state                                    */
            /*******************************************************/
            SKP_Silk_PLC( psDec, &sDecCtrl, pOut, L, action );

            psDec->lossCnt = 0;
            psDec->prev_sigtype = sDecCtrl.sigtype;

            /* A frame has been decoded without errors */
            psDec->first_frame_after_reset = 0;
        }
```

```
    }
    /************************************************************/
    /* Generate Concealment Frame if packet is lost, or corrupt  */
    /************************************************************/
    if( action == 1 ) {
        /* Handle packet loss by extrapolation */
        SKP_Silk_PLC( psDec, &sDecCtrl, pOut, L, action );
        psDec->lossCnt++;

    }

    /*************************/
    /* Update output buffer. */
    /*************************/
    SKP_memcpy( psDec->outBuf, pOut, L * sizeof( SKP_int16 ) );

    /************************************************************/
    /* Ensure smooth connection of extrapolated and good frames     */
    /************************************************************/
    SKP_Silk_PLC_glue_frames( psDec, &sDecCtrl, pOut, L );

    /*********************************************/
    /* Comfort noise generation / estimation        */
    /*********************************************/
    SKP_Silk_CNG( psDec, &sDecCtrl, pOut , L );

    /*****************************************/
    /* HP filter output                         */
    /*****************************************/
    SKP_assert( ( ( psDec->fs_kHz == 12 ) && ( L % 3 ) == 0 ) ||
                ( ( psDec->fs_kHz != 12 ) && ( L % 2 ) == 0 ) );
    SKP_Silk_biquad( pOut, psDec->HP_B, psDec->HP_A, psDec->HPState, pOut, L );

    /*****************************************/
    /* set output frame length                  */
    /*****************************************/
    *pN = ( SKP_int16 )L;

    /* Update some decoder state variables */
    psDec->lagPrev = sDecCtrl.pitchL[ NB_SUBFR - 1 ];

    return ret;
}
```

A.26. src/SKP_Silk_decode_indices_v4.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

/* Decode indices from payload */
void SKP_Silk_decode_indices_v4(
    SKP_Silk_decoder_state      *psDec            /* I/O    State
              */
)
{
    SKP_int   i, k, Ix, fs_kHz_dec, FrameIndex = 0, FrameTermination;
    SKP_int   sigtype, QuantOffsetType, seed_int, nBytesUsed;
    SKP_int   decode_absolute_lagIndex, delta_lagIndex, prev_lagIndex = 0;
    const SKP_Silk_NLSF_CB_struct *psNLSF_CB = NULL;
    SKP_Silk_range_coder_state  *psRC = &psDec->sRC;
    /*************************/
    /* Decode sampling rate */
    /*************************/
    /* only done for first frame of packet */
    if( psDec->nFramesDecoded == 0 ) {
        SKP_Silk_range_decoder( &Ix, psRC, SKP_Silk_SamplingRates_CDF, SKP_Silk_S
amplingRates_offset );

        /* check that sampling rate is supported */
        if( Ix < 0 || Ix > 3 ) {
```

```
            psRC->error = RANGE_CODER_ILLEGAL_SAMPLING_RATE;
            return;
        }
        fs_kHz_dec = SKP_Silk_SamplingRates_table[ Ix ];
        SKP_Silk_decoder_set_fs( psDec, fs_kHz_dec );

        FrameIndex       = 0;
        FrameTermination = SKP_SILK_MORE_FRAMES;
    }

    while( FrameTermination == SKP_SILK_MORE_FRAMES ) {
        /*******************/
        /* Decode VAD flag */
        /*******************/
        SKP_Silk_range_decoder( &psDec->vadFlagBuf[ FrameIndex ], psRC, SKP_Silk_
vadflag_CDF, SKP_Silk_vadflag_offset );

        /*****************************************/
        /* Decode signal type and quantizer offset */
        /*****************************************/
        if( FrameIndex == 0 ) {
            /* first frame in packet: independent coding */
            SKP_Silk_range_decoder( &Ix, psRC, SKP_Silk_type_offset_CDF, SKP_Silk
_type_offset_CDF_offset );
        } else {
            /* condidtional coding */
            SKP_Silk_range_decoder( &Ix, psRC, SKP_Silk_type_offset_joint_CDF[ ps
Dec->typeOffsetPrev ],
                    SKP_Silk_type_offset_CDF_offset );
        }
        sigtype               = SKP_RSHIFT( Ix, 1 );
        QuantOffsetType       = Ix & 1;
        psDec->typeOffsetPrev = Ix;

        /****************/
        /* Decode gains */
        /****************/
        /* first subframe */
        if( FrameIndex == 0 ) {
            /* first frame in packet: independent coding */
            SKP_Silk_range_decoder( &psDec->GainsIndices[ FrameIndex ][ 0 ], psRC
, SKP_Silk_gain_CDF[ sigtype ], SKP_Silk_gain_CDF_offset );
        } else {
            /* condidtional coding */
            SKP_Silk_range_decoder( &psDec->GainsIndices[ FrameIndex ][ 0 ], psRC
, SKP_Silk_delta_gain_CDF, SKP_Silk_delta_gain_CDF_offset );
        }

        /* remaining subframes */
        for( i = 1; i < NB_SUBFR; i++ ) {
            SKP_Silk_range_decoder( &psDec->GainsIndices[ FrameIndex ][ i ], psRC
, SKP_Silk_delta_gain_CDF, SKP_Silk_delta_gain_CDF_offset );
        }
```

```
        /**********************/
        /* Decode LSF Indices */
        /**********************/

        /* Set pointer to LSF VQ CB for the current signal type */
        psNLSF_CB = psDec->psNLSF_CB[ sigtype ];

        /* Arithmetically decode NLSF path */
        SKP_Silk_range_decoder_multi( psDec->NLSFIndices[ FrameIndex ], psRC, psN
LSF_CB->StartPtr, psNLSF_CB->MiddleIx, psNLSF_CB->nStages );

        /***********************************/
        /* Decode LSF interpolation factor */
        /***********************************/
        SKP_Silk_range_decoder( &psDec->NLSFInterpCoef_Q2[ FrameIndex ], psRC, SK
P_Silk_NLSF_interpolation_factor_CDF,
            SKP_Silk_NLSF_interpolation_factor_offset );

        if( sigtype == SIG_TYPE_VOICED ) {
            /**********************/
            /* Decode pitch lags */
            /**********************/
            /* Get lag index */
            decode_absolute_lagIndex = 1;
            if( FrameIndex > 0 && psDec->sigtype[ FrameIndex - 1 ] == SIG_TYPE_VO
ICED ) {
                /* Decode Delta index */
                SKP_Silk_range_decoder( &delta_lagIndex,psRC, SKP_Silk_pitch_delt
a_CDF,  SKP_Silk_pitch_delta_CDF_offset );
                if( delta_lagIndex < ( MAX_DELTA_LAG << 1 ) + 1 ) {
                    delta_lagIndex = delta_lagIndex - MAX_DELTA_LAG;
                    psDec->lagIndex[ FrameIndex ] = prev_lagIndex + delta_lagInde
x;
                    decode_absolute_lagIndex = 0;
                }
            }
            if( decode_absolute_lagIndex ) {
                /* Absolute decoding */
                if( psDec->fs_kHz == 8 ) {
                    SKP_Silk_range_decoder( &psDec->lagIndex[ FrameIndex ], psRC,
 SKP_Silk_pitch_lag_NB_CDF,  SKP_Silk_pitch_lag_NB_CDF_offset );
                } else if( psDec->fs_kHz == 12 ) {
                    SKP_Silk_range_decoder( &psDec->lagIndex[ FrameIndex ], psRC,
 SKP_Silk_pitch_lag_MB_CDF,  SKP_Silk_pitch_lag_MB_CDF_offset );
                } else if( psDec->fs_kHz == 16 ) {
                    SKP_Silk_range_decoder( &psDec->lagIndex[ FrameIndex ], psRC,
 SKP_Silk_pitch_lag_WB_CDF,  SKP_Silk_pitch_lag_WB_CDF_offset );
                } else {
                    SKP_Silk_range_decoder( &psDec->lagIndex[ FrameIndex ], psRC,
 SKP_Silk_pitch_lag_SWB_CDF, SKP_Silk_pitch_lag_SWB_CDF_offset );
                }
            }
            prev_lagIndex = psDec->lagIndex[ FrameIndex ];

            /* Get countour index */
            if( psDec->fs_kHz == 8 ) {
                /* Less codevectors used in 8 khz mode */
```

```
                SKP_Silk_range_decoder( &psDec->contourIndex[ FrameIndex ], psRC,
 SKP_Silk_pitch_contour_NB_CDF, SKP_Silk_pitch_contour_NB_CDF_offset );
            } else {
                /* Joint for 12, 16, and 24 khz */
                SKP_Silk_range_decoder( &psDec->contourIndex[ FrameIndex ], psRC,
 SKP_Silk_pitch_contour_CDF, SKP_Silk_pitch_contour_CDF_offset );
            }

            /*********************/
            /* Decode LTP gains */
            /*********************/
            /* Decode PERIndex value */
            SKP_Silk_range_decoder( &psDec->PERIndex[ FrameIndex ], psRC, SKP_Sil
k_LTP_per_index_CDF, SKP_Silk_LTP_per_index_CDF_offset );

            for( k = 0; k < NB_SUBFR; k++ ) {
                SKP_Silk_range_decoder( &psDec->LTPIndex[ FrameIndex ][ k ], psRC
, SKP_Silk_LTP_gain_CDF_ptrs[ psDec->PERIndex[ FrameIndex ] ],
                    SKP_Silk_LTP_gain_CDF_offsets[ psDec->PERIndex[ FrameIndex ]
] );
            }

            /***********************/
            /* Decode LTP scaling */
            /***********************/
            SKP_Silk_range_decoder( &psDec->LTP_scaleIndex[ FrameIndex ], psRC, S
KP_Silk_LTPscale_CDF, SKP_Silk_LTPscale_offset );
        }

        /***************/
        /* Decode seed */
        /***************/
        SKP_Silk_range_decoder( &seed_int, psRC, SKP_Silk_Seed_CDF, SKP_Silk_Seed
_offset );
        psDec->Seed[ FrameIndex ] = ( SKP_int32 )seed_int;
        /***********************************/
        /* Decode Frame termination indicator */
        /***********************************/
        SKP_Silk_range_decoder( &FrameTermination, psRC, SKP_Silk_FrameTerminatio
n_v4_CDF, SKP_Silk_FrameTermination_v4_offset );

        psDec->sigtype[ FrameIndex ]         = sigtype;
        psDec->QuantOffsetType[ FrameIndex ] = QuantOffsetType;

        FrameIndex++;
    }

    /*************************************/
    /* get number of bytes used so far      */
    /*************************************/
    SKP_Silk_range_coder_get_length( psRC, &nBytesUsed );
    psDec->nBytesLeft = psRC->bufferLength - nBytesUsed;
    if( psDec->nBytesLeft < 0 ) {
        psRC->error = RANGE_CODER_READ_BEYOND_BUFFER;
    }
```

```
    psDec->nFramesInPacket  = FrameIndex;
    psDec->FrameTermination = FrameTermination;
}
```

A.27. src/SKP_Silk_decode_parameters.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

/* Decode parameters from payload */
void SKP_Silk_decode_parameters(
    SKP_Silk_decoder_state      *psDec,             /* I/O  State
                        */
    SKP_Silk_decoder_control    *psDecCtrl,         /* I/O  Decoder control
                        */
    SKP_int                     q[],                /* O    Excitation signal
                        */
    const SKP_int               fullDecoding        /* I    Flag to tell if only
arithmetic decoding    */
)
{
    SKP_int   i, k, Ix, fs_kHz_dec, nBytesUsed;
    SKP_int   Ixs[ NB_SUBFR ];
    SKP_int   GainsIndices[ NB_SUBFR ];
```

```
    SKP_int   NLSFIndices[ NLSF_MSVQ_MAX_CB_STAGES ];
    SKP_int   pNLSF_Q15[ MAX_LPC_ORDER ], pNLSF0_Q15[ MAX_LPC_ORDER ];
    const SKP_int16 *cbk_ptr_Q14;
    const SKP_Silk_NLSF_CB_struct *psNLSF_CB = NULL;
    SKP_Silk_range_coder_state  *psRC = &psDec->sRC;

    /************************/
    /* Decode sampling rate */
    /************************/
    /* only done for first frame of packet */
    if( psDec->nFramesDecoded == 0 ) {
        SKP_Silk_range_decoder( &Ix, psRC, SKP_Silk_SamplingRates_CDF, SKP_Silk_S
amplingRates_offset );

        /* check that sampling rate is supported */
        if( Ix < 0 || Ix > 3 ) {
            psRC->error = RANGE_CODER_ILLEGAL_SAMPLING_RATE;
            return;
        }
        fs_kHz_dec = SKP_Silk_SamplingRates_table[ Ix ];
        SKP_Silk_decoder_set_fs( psDec, fs_kHz_dec );
    }

    /*****************************************/
    /* Decode signal type and quantizer offset */
    /*****************************************/
    if( psDec->nFramesDecoded == 0 ) {
        /* first frame in packet: independent coding */
        SKP_Silk_range_decoder( &Ix, psRC, SKP_Silk_type_offset_CDF, SKP_Silk_typ
e_offset_CDF_offset );
    } else {
        /* condidtional coding */
        SKP_Silk_range_decoder( &Ix, psRC, SKP_Silk_type_offset_joint_CDF[ psDec-
>typeOffsetPrev ],
                SKP_Silk_type_offset_CDF_offset );
    }
    psDecCtrl->sigtype         = SKP_RSHIFT( Ix, 1 );
    psDecCtrl->QuantOffsetType = Ix & 1;
    psDec->typeOffsetPrev      = Ix;

    /****************/
    /* Decode gains */
    /****************/
    /* first subframe */
    if( psDec->nFramesDecoded == 0 ) {
        /* first frame in packet: independent coding */
        SKP_Silk_range_decoder( &GainsIndices[ 0 ], psRC, SKP_Silk_gain_CDF[ psDe
cCtrl->sigtype ], SKP_Silk_gain_CDF_offset );
    } else {
        /* condidtional coding */
        SKP_Silk_range_decoder( &GainsIndices[ 0 ], psRC, SKP_Silk_delta_gain_CDF
, SKP_Silk_delta_gain_CDF_offset );
    }
```

```
    /* remaining subframes */
    for( i = 1; i < NB_SUBFR; i++ ) {
        SKP_Silk_range_decoder( &GainsIndices[ i ], psRC, SKP_Silk_delta_gain_CDF
, SKP_Silk_delta_gain_CDF_offset );
    }

    /* Dequant Gains */
    SKP_Silk_gains_dequant( psDecCtrl->Gains_Q16, GainsIndices, &psDec->LastGainI
ndex, psDec->nFramesDecoded );
    /****************/
    /* Decode NLSFs */
    /****************/
    /* Set pointer to NLSF VQ CB for the current signal type */
    psNLSF_CB = psDec->psNLSF_CB[ psDecCtrl->sigtype ];

    /* Arithmetically decode NLSF path */
    SKP_Silk_range_decoder_multi( NLSFIndices, psRC, psNLSF_CB->StartPtr, psNLSF_
CB->MiddleIx, psNLSF_CB->nStages );

    /* From the NLSF path, decode an NLSF vector */
    SKP_Silk_NLSF_MSVQ_decode( pNLSF_Q15, psNLSF_CB, NLSFIndices, psDec->LPC_orde
r );

    /***********************************/
    /* Decode NLSF interpolation factor */
    /***********************************/
    SKP_Silk_range_decoder( &psDecCtrl->NLSFInterpCoef_Q2, psRC, SKP_Silk_NLSF_in
terpolation_factor_CDF,
        SKP_Silk_NLSF_interpolation_factor_offset );

    /* If just reset, e.g., because internal Fs changed, do not allow interpolati
on */
    /* improves the case of packet loss in the first frame after a switch
   */
    if( psDec->first_frame_after_reset == 1 ) {
        psDecCtrl->NLSFInterpCoef_Q2 = 4;
    }

    if( fullDecoding ) {
        /* Convert NLSF parameters to AR prediction filter coefficients */
        SKP_Silk_NLSF2A_stable( psDecCtrl->PredCoef_Q12[ 1 ], pNLSF_Q15, psDec->L
PC_order );

        if( psDecCtrl->NLSFInterpCoef_Q2 < 4 ) {
            /* Calculation of the interpolated NLSF0 vector from the interpolatio
n factor, */
            /* the previous NLSF1, and the current NLSF1
          */
            for( i = 0; i < psDec->LPC_order; i++ ) {
                pNLSF0_Q15[ i ] = psDec->prevNLSF_Q15[ i ] + SKP_RSHIFT( SKP_MUL(
 psDecCtrl->NLSFInterpCoef_Q2,
                    ( pNLSF_Q15[ i ] - psDec->prevNLSF_Q15[ i ] ) ), 2 );
            }

            /* Convert NLSF parameters to AR prediction filter coefficients */
            SKP_Silk_NLSF2A_stable( psDecCtrl->PredCoef_Q12[ 0 ], pNLSF0_Q15, psD
ec->LPC_order );
        } else {
            /* Copy LPC coefficients for first half from second half */
            SKP_memcpy( psDecCtrl->PredCoef_Q12[ 0 ], psDecCtrl->PredCoef_Q12[ 1
],
```

```
                psDec->LPC_order * sizeof( SKP_int16 ) );
        }
    }

    SKP_memcpy( psDec->prevNLSF_Q15, pNLSF_Q15, psDec->LPC_order * sizeof( SKP_in
t ) );

    /* After a packet loss do BWE of LPC coefs */
    if( psDec->lossCnt ) {
        SKP_Silk_bwexpander( psDecCtrl->PredCoef_Q12[ 0 ], psDec->LPC_order, BWE_
AFTER_LOSS_Q16 );
        SKP_Silk_bwexpander( psDecCtrl->PredCoef_Q12[ 1 ], psDec->LPC_order, BWE_
AFTER_LOSS_Q16 );
    }

    if( psDecCtrl->sigtype == SIG_TYPE_VOICED ) {
        /**********************/
        /* Decode pitch lags */
        /**********************/
        /* Get lag index */
        if( psDec->fs_kHz == 8 ) {
            SKP_Silk_range_decoder( &Ixs[ 0 ], psRC, SKP_Silk_pitch_lag_NB_CDF,
SKP_Silk_pitch_lag_NB_CDF_offset );
        } else if( psDec->fs_kHz == 12 ) {
            SKP_Silk_range_decoder( &Ixs[ 0 ], psRC, SKP_Silk_pitch_lag_MB_CDF,
SKP_Silk_pitch_lag_MB_CDF_offset );
        } else if( psDec->fs_kHz == 16 ) {
            SKP_Silk_range_decoder( &Ixs[ 0 ], psRC, SKP_Silk_pitch_lag_WB_CDF,
SKP_Silk_pitch_lag_WB_CDF_offset );
        } else {
            SKP_Silk_range_decoder( &Ixs[ 0 ], psRC, SKP_Silk_pitch_lag_SWB_CDF,
SKP_Silk_pitch_lag_SWB_CDF_offset );
        }

        /* Get countour index */
        if( psDec->fs_kHz == 8 ) {
            /* Less codevectors used in 8 khz mode */
            SKP_Silk_range_decoder( &Ixs[ 1 ], psRC, SKP_Silk_pitch_contour_NB_CD
F, SKP_Silk_pitch_contour_NB_CDF_offset );
        } else {
            /* Joint for 12, 16, and 24 khz */
            SKP_Silk_range_decoder( &Ixs[ 1 ], psRC, SKP_Silk_pitch_contour_CDF,
SKP_Silk_pitch_contour_CDF_offset );
        }

        /* Decode pitch values */
        SKP_Silk_decode_pitch( Ixs[ 0 ], Ixs[ 1 ], psDecCtrl->pitchL, psDec->fs_k
Hz );

        /********************/
        /* Decode LTP gains */
        /********************/
        /* Decode PERIndex value */
        SKP_Silk_range_decoder( &psDecCtrl->PERIndex, psRC, SKP_Silk_LTP_per_inde
x_CDF,
                SKP_Silk_LTP_per_index_CDF_offset );

        /* Decode Codebook Index */
        cbk_ptr_Q14 = SKP_Silk_LTP_vq_ptrs_Q14[ psDecCtrl->PERIndex ]; // set poi
nter to start of codebook
```

```
        for( k = 0; k < NB_SUBFR; k++ ) {
            SKP_Silk_range_decoder( &Ix, psRC, SKP_Silk_LTP_gain_CDF_ptrs[ psDecC
trl->PERIndex ],
                SKP_Silk_LTP_gain_CDF_offsets[ psDecCtrl->PERIndex ] );

            for( i = 0; i < LTP_ORDER; i++ ) {
                psDecCtrl->LTPCoef_Q14[ SKP_SMULBB( k, LTP_ORDER ) + i ] = cbk_pt
r_Q14[ SKP_SMULBB( Ix, LTP_ORDER ) + i ];
            }
        }

        /**********************/
        /* Decode LTP scaling */
        /**********************/
        SKP_Silk_range_decoder( &Ix, psRC, SKP_Silk_LTPscale_CDF, SKP_Silk_LTPsca
le_offset );
        psDecCtrl->LTP_scale_Q14 = SKP_Silk_LTPScales_table_Q14[ Ix ];
    } else {
        SKP_memset( psDecCtrl->pitchL,      0, NB_SUBFR * sizeof( SKP_int ) );
        SKP_memset( psDecCtrl->LTPCoef_Q14, 0, NB_SUBFR * LTP_ORDER * sizeof( SKP
_int16 ) );
        psDecCtrl->PERIndex      = 0;
        psDecCtrl->LTP_scale_Q14 = 0;
    }

    /***************/
    /* Decode seed */
    /***************/
    SKP_Silk_range_decoder( &Ix, psRC, SKP_Silk_Seed_CDF, SKP_Silk_Seed_offset );
    psDecCtrl->Seed = ( SKP_int32 )Ix;
    /*****************************************/
    /* Decode quantization indices of excitation */
    /*****************************************/
    SKP_Silk_decode_pulses( psRC, psDecCtrl, q, psDec->frame_length );

    /*****************************************/
    /* Decode VAD flag                       */
    /*****************************************/
    SKP_Silk_range_decoder( &psDec->vadFlag, psRC, SKP_Silk_vadflag_CDF, SKP_Silk
_vadflag_offset );

    /*************************************/
    /* Decode Frame termination indicator */
    /*************************************/
    SKP_Silk_range_decoder( &psDec->FrameTermination, psRC, SKP_Silk_FrameTermina
tion_CDF, SKP_Silk_FrameTermination_offset );

    /*************************************/
    /* get number of bytes used so far      */
    /*************************************/
    SKP_Silk_range_coder_get_length( psRC, &nBytesUsed );
    psDec->nBytesLeft = psRC->bufferLength - nBytesUsed;
    if( psDec->nBytesLeft < 0 ) {
        psRC->error = RANGE_CODER_READ_BEYOND_BUFFER;
```

```
    }

    /****************************************/
    /* check remaining bits in last byte    */
    /****************************************/
    if( psDec->nBytesLeft == 0 ) {
        SKP_Silk_range_coder_check_after_decoding( psRC );
    }
}
```

A.28. src/SKP_Silk_decode_parameters_v4.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

/* Decode parameters from payload */
void SKP_Silk_decode_parameters_v4(
    SKP_Silk_decoder_state      *psDec,                                 /* I/O  S
tate                                 */
    SKP_Silk_decoder_control    *psDecCtrl,                             /* I/O  D
ecoder control                       */
    SKP_int                     q[ MAX_FRAME_LENGTH ],                  /* O    E
xcitation signal                     */
```

```
    const SKP_int               fullDecoding                            /* I    F
lag to tell if only arithmetic decoding */
)
{
    SKP_int   i, k, Ix, nBytesUsed;
    SKP_int   pNLSF_Q15[ MAX_LPC_ORDER ], pNLSF0_Q15[ MAX_LPC_ORDER ];
    const SKP_int16 *cbk_ptr_Q14;
    const SKP_Silk_NLSF_CB_struct *psNLSF_CB = NULL;
    SKP_Silk_range_coder_state  *psRC = &psDec->sRC;

    psDec->FrameTermination       = SKP_SILK_MORE_FRAMES;
    psDecCtrl->sigtype            = psDec->sigtype[ psDec->nFramesDecoded ];
    psDecCtrl->QuantOffsetType    = psDec->QuantOffsetType[ psDec->nFramesDecoded
 ];
    psDec->vadFlag                = psDec->vadFlagBuf[ psDec->nFramesDecoded ];
    psDecCtrl->NLSFInterpCoef_Q2  = psDec->NLSFInterpCoef_Q2[ psDec->nFramesDecod
ed ];
    psDecCtrl->Seed               = psDec->Seed[ psDec->nFramesDecoded ];

    /* Dequant Gains */
    SKP_Silk_gains_dequant( psDecCtrl->Gains_Q16, psDec->GainsIndices[ psDec->nFr
amesDecoded ], &psDec->LastGainIndex, psDec->nFramesDecoded );
    /****************/
    /* Decode NLSFs */
    /****************/

    /* Set pointer to NLSF VQ CB for the current signal type */
    psNLSF_CB = psDec->psNLSF_CB[ psDecCtrl->sigtype ];

    /* From the NLSF path, decode an NLSF vector */
    SKP_Silk_NLSF_MSVQ_decode( pNLSF_Q15, psNLSF_CB, psDec->NLSFIndices[ psDec->n
FramesDecoded ], psDec->LPC_order );

    /* Convert NLSF parameters to AR prediction filter coefficients */
    SKP_Silk_NLSF2A_stable( psDecCtrl->PredCoef_Q12[ 1 ], pNLSF_Q15, psDec->LPC_o
rder );

    /* If just reset, e.g., because internal Fs changed, do not allow interpolati
on */
    /* improves the case of packet loss in the first frame after a switch
   */
    if( psDec->first_frame_after_reset == 1 ) {
        psDecCtrl->NLSFInterpCoef_Q2 = 4;
    }

    if( psDecCtrl->NLSFInterpCoef_Q2 < 4 ) {
        /* Calculation of the interpolated NLSF0 vector from the interpolation fa
ctor, */
        /* the previous NLSF1, and the current NLSF1
      */
        for( i = 0; i < psDec->LPC_order; i++ ) {
            pNLSF0_Q15[ i ] = psDec->prevNLSF_Q15[ i ] + SKP_RSHIFT( SKP_MUL( psD
ecCtrl->NLSFInterpCoef_Q2,
                ( pNLSF_Q15[ i ] - psDec->prevNLSF_Q15[ i ] ) ), 2 );
        }

        /* Convert NLSF parameters to AR prediction filter coefficients */
        SKP_Silk_NLSF2A_stable( psDecCtrl->PredCoef_Q12[ 0 ], pNLSF0_Q15, psDec->
LPC_order );
    } else {
```

```
        /* Copy LPC coefficients for first half from second half */
        SKP_memcpy( psDecCtrl->PredCoef_Q12[ 0 ], psDecCtrl->PredCoef_Q12[ 1 ],
            psDec->LPC_order * sizeof( SKP_int16 ) );
    }

    SKP_memcpy( psDec->prevNLSF_Q15, pNLSF_Q15, psDec->LPC_order * sizeof( SKP_in
t ) );

    /* After a packet loss do BWE of LPC coefs */
    if( psDec->lossCnt ) {
        SKP_Silk_bwexpander( psDecCtrl->PredCoef_Q12[ 0 ], psDec->LPC_order, BWE_
AFTER_LOSS_Q16 );
        SKP_Silk_bwexpander( psDecCtrl->PredCoef_Q12[ 1 ], psDec->LPC_order, BWE_
AFTER_LOSS_Q16 );
    }

    if( psDecCtrl->sigtype == SIG_TYPE_VOICED ) {
        /**********************/
        /* Decode pitch lags */
        /**********************/

        /* Decode pitch values */
        SKP_Silk_decode_pitch( psDec->lagIndex[ psDec->nFramesDecoded ],
            psDec->contourIndex[ psDec->nFramesDecoded ], psDecCtrl->pitchL, psDe
c->fs_kHz );

        /*********************/
        /* Decode LTP gains */
        /*********************/
        psDecCtrl->PERIndex = psDec->PERIndex[ psDec->nFramesDecoded ];

        /* Decode Codebook Index */
        cbk_ptr_Q14 = SKP_Silk_LTP_vq_ptrs_Q14[ psDecCtrl->PERIndex ]; /* set poi
nter to start of codebook */

        for( k = 0; k < NB_SUBFR; k++ ) {
            Ix = psDec->LTPIndex[ psDec->nFramesDecoded ][ k ];
            for( i = 0; i < LTP_ORDER; i++ ) {
                psDecCtrl->LTPCoef_Q14[ SKP_SMULBB( k, LTP_ORDER ) + i ] = cbk_pt
r_Q14[ SKP_SMULBB( Ix, LTP_ORDER ) + i ];
            }
        }

        /**********************/
        /* Decode LTP scaling */
        /**********************/
        Ix = psDec->LTP_scaleIndex[ psDec->nFramesDecoded ];
        psDecCtrl->LTP_scale_Q14 = SKP_Silk_LTPScales_table_Q14[ Ix ];
    } else {
        SKP_memset( psDecCtrl->pitchL,      0, NB_SUBFR * sizeof( SKP_int ) );
        SKP_memset( psDecCtrl->LTPCoef_Q14, 0, NB_SUBFR * LTP_ORDER * sizeof( SKP
_int16 ) );
        psDecCtrl->PERIndex      = 0;
        psDecCtrl->LTP_scale_Q14 = 0;
    }
```

```
    /*****************************************/
    /* Decode quantization indices of excitation */
    /*****************************************/
    SKP_Silk_decode_pulses( psRC, psDecCtrl, q, psDec->frame_length );

    /*************************************/
    /* get number of bytes used so far      */
    /*************************************/
    SKP_Silk_range_coder_get_length( psRC, &nBytesUsed );
    psDec->nBytesLeft = psRC->bufferLength - nBytesUsed;
    if( psDec->nBytesLeft < 0 ) {
        psRC->error = RANGE_CODER_READ_BEYOND_BUFFER;
    }

    /*************************************/
    /* check remaining bits in last byte    */
    /*************************************/
    if( psDec->nBytesLeft == 0 ) {
        SKP_Silk_range_coder_check_after_decoding( psRC );
    }

    if( psDec->nFramesInPacket == (psDec->nFramesDecoded + 1)) {
        /* To indicate the packet has been fully decoded */
        psDec->FrameTermination = SKP_SILK_LAST_FRAME;
    }
}
```

A.29. src/SKP_Silk_decode_pulses.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
```

```
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

/*****************************************/
/* Decode quantization indices of excitation */
/*****************************************/
void SKP_Silk_decode_pulses(
    SKP_Silk_range_coder_state      *psRC,              /* I/O  Range coder state
                            */
    SKP_Silk_decoder_control        *psDecCtrl,         /* I/O  Decoder control
                            */
    SKP_int                         q[],                /* O    Excitation signal
                            */
    const SKP_int                   frame_length        /* I    Frame length (pre
liminary)                   */
)
{
    SKP_int   i, j, k, iter, abs_q, nLS, bit;
    SKP_int   sum_pulses[ MAX_NB_SHELL_BLOCKS ], nLshifts[ MAX_NB_SHELL_BLOCKS ];
    SKP_int   *pulses_ptr;
    const SKP_uint16 *cdf_ptr;

    /*********************/
    /* Decode rate level */
    /*********************/
    SKP_Silk_range_decoder( &psDecCtrl->RateLevelIndex, psRC,
            SKP_Silk_rate_levels_CDF[ psDecCtrl->sigtype ], SKP_Silk_rate_levels_
CDF_offset );

    /* Calculate number of shell blocks */
    iter = frame_length / SHELL_CODEC_FRAME_LENGTH;

    /*************************************************/
    /* Sum-Weighted-Pulses Decoding                  */
    /*************************************************/
    cdf_ptr = SKP_Silk_pulses_per_block_CDF[ psDecCtrl->RateLevelIndex ];
    for( i = 0; i < iter; i++ ) {
        nLshifts[ i ] = 0;
        SKP_Silk_range_decoder( &sum_pulses[ i ], psRC, cdf_ptr, SKP_Silk_pulses_
per_block_CDF_offset );

        /* LSB indication */
        while( sum_pulses[ i ] == ( MAX_PULSES + 1 ) ) {
            nLshifts[ i ]++;
            SKP_Silk_range_decoder( &sum_pulses[ i ], psRC,
```

```
                    SKP_Silk_pulses_per_block_CDF[ N_RATE_LEVELS - 1 ], SKP_Silk_
pulses_per_block_CDF_offset );
        }
    }

    /***************************************************/
    /* Shell decoding                                  */
    /***************************************************/
    for( i = 0; i < iter; i++ ) {
        if( sum_pulses[ i ] > 0 ) {
            SKP_Silk_shell_decoder( &q[ SKP_SMULBB( i, SHELL_CODEC_FRAME_LENGTH )
 ], psRC, sum_pulses[ i ] );
        } else {
            SKP_memset( &q[ SKP_SMULBB( i, SHELL_CODEC_FRAME_LENGTH ) ], 0, SHELL
_CODEC_FRAME_LENGTH * sizeof( SKP_int ) );
        }
    }

    /***************************************************/
    /* LSB Decoding                                    */
    /***************************************************/
    for( i = 0; i < iter; i++ ) {
        if( nLshifts[ i ] > 0 ) {
            nLS = nLshifts[ i ];
            pulses_ptr = &q[ SKP_SMULBB( i, SHELL_CODEC_FRAME_LENGTH ) ];
            for( k = 0; k < SHELL_CODEC_FRAME_LENGTH; k++ ) {
                abs_q = pulses_ptr[ k ];
                for( j = 0; j < nLS; j++ ) {
                    abs_q = SKP_LSHIFT( abs_q, 1 );
                    SKP_Silk_range_decoder( &bit, psRC, SKP_Silk_lsb_CDF, 1 );
                    abs_q += bit;
                }
                pulses_ptr[ k ] = abs_q;
            }
        }
    }

    /***************************************/
    /* Decode and add signs to pulse signal */
    /***************************************/
    SKP_Silk_decode_signs( psRC, q, frame_length, psDecCtrl->sigtype,
        psDecCtrl->QuantOffsetType, psDecCtrl->RateLevelIndex);
}
```

A.30. src/SKP_Silk_decoder_set_fs.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
```

```
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

/* Set decoder sampling rate */
void SKP_Silk_decoder_set_fs(
    SKP_Silk_decoder_state          *psDec,             /* I/O  Decoder state poi
nter                        */
    SKP_int                         fs_kHz              /* I    Sampling frequenc
y (kHz)                     */
)
{
    if( psDec->fs_kHz != fs_kHz ) {
        psDec->fs_kHz  = fs_kHz;
        psDec->frame_length = SKP_SMULBB( FRAME_LENGTH_MS, fs_kHz );
        psDec->subfr_length = SKP_SMULBB( FRAME_LENGTH_MS / NB_SUBFR, fs_kHz );
        if( psDec->fs_kHz == 8 ) {
            psDec->LPC_order = MIN_LPC_ORDER;
            psDec->psNLSF_CB[ 0 ] = &SKP_Silk_NLSF_CB0_10;
            psDec->psNLSF_CB[ 1 ] = &SKP_Silk_NLSF_CB1_10;
        } else {
            psDec->LPC_order = MAX_LPC_ORDER;
            psDec->psNLSF_CB[ 0 ] = &SKP_Silk_NLSF_CB0_16;
            psDec->psNLSF_CB[ 1 ] = &SKP_Silk_NLSF_CB1_16;
        }
        /* Reset part of the decoder state */
        SKP_memset( psDec->sLPC_Q14,     0, MAX_LPC_ORDER      * sizeof( SKP_int3
2 ) );
        SKP_memset( psDec->outBuf,       0, MAX_FRAME_LENGTH   * sizeof( SKP_int1
6 ) );
```

```
        SKP_memset( psDec->prevNLSF_Q15, 0, MAX_LPC_ORDER      * sizeof( SKP_int
)   );

        psDec->sLTP_buf_idx            = 0;
        psDec->lagPrev                 = 100;
        psDec->LastGainIndex           = 1;
        psDec->prev_sigtype            = 0;
        psDec->first_frame_after_reset = 1;

        if( fs_kHz == 24 ) {
            psDec->HP_A = SKP_Silk_Dec_A_HP_24;
            psDec->HP_B = SKP_Silk_Dec_B_HP_24;
        } else if( fs_kHz == 16 ) {
            psDec->HP_A = SKP_Silk_Dec_A_HP_16;
            psDec->HP_B = SKP_Silk_Dec_B_HP_16;
        } else if( fs_kHz == 12 ) {
            psDec->HP_A = SKP_Silk_Dec_A_HP_12;
            psDec->HP_B = SKP_Silk_Dec_B_HP_12;
        } else if( fs_kHz == 8 ) {
            psDec->HP_A = SKP_Silk_Dec_A_HP_8;
            psDec->HP_B = SKP_Silk_Dec_B_HP_8;
        } else {
            /* unsupported sampling rate */
            SKP_assert( 0 );
        }
    }

    /* Check that settings are valid */
    SKP_assert( psDec->frame_length > 0 && psDec->frame_length <= MAX_FRAME_LENGT
H );
}
```

A.31. src/SKP_Silk_define.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
```

```
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_DEFINE_H
#define SKP_SILK_DEFINE_H

#include "SKP_Silk_errors.h"
#include "SKP_Silk_typedef.h"

#ifdef __cplusplus
extern "C"
{
#endif


#define MAX_FRAMES_PER_PACKET                   5
#define BIT_STREAM_V3                           3
#define BIT_STREAM_V4                           4
#define USE_BIT_STREAM_V                        BIT_STREAM_V3 // Should be moved
to a API call


/* MAX DELTA LAG used for multiframe packets */
#define MAX_DELTA_LAG                           10

/* Lower limit on bitrate for each mode */
#define MIN_TARGET_RATE_NB_BPS                  5000
#define MIN_TARGET_RATE_MB_BPS                  7000
#define MIN_TARGET_RATE_WB_BPS                  8000
#define MIN_TARGET_RATE_SWB_BPS                 20000

/* Transition bitrates between modes */
#define SWB2WB_BITRATE_BPS                      26000
#define WB2SWB_BITRATE_BPS                      32000
#define WB2MB_BITRATE_BPS                       15000
#define MB2WB_BITRATE_BPS                       20000
#define MB2NB_BITRATE_BPS                       10000
#define NB2MB_BITRATE_BPS                       14000
```

```
/* Integration/hysteresis threshold for lowering internal sample frequency */
/* 30000000 -> 6 sec if bitrate is 5000 bps below limit; 3 sec if bitrate is 1000
0 bps below limit */
#define ACCUM_BITS_DIFF_THRESHOLD               30000000
#define TARGET_RATE_TAB_SZ                      8

/* DTX settings                                 */
#define NO_SPEECH_FRAMES_BEFORE_DTX             5       /* eq 100 ms */
#define MAX_CONSECUTIVE_DTX                     20      /* eq 400 ms */

#define USE_LBRR                                1

/* Amount of concecutive no FEC packets before telling JB */
#define NO_LBRR_THRES                           10

/* Maximum delay between real packet and LBRR packet */
#define MAX_LBRR_DELAY                          2
#define LBRR_IDX_MASK                           1

#define INBAND_FEC_MIN_RATE_BPS                 18000  /* Dont use inband FEC bel
ow this total target rate  */
#define LBRR_LOSS_THRES                         2   /* Start adding LBRR at this
loss rate (needs tuning)   */

/* LBRR usage defines */
#define SKP_SILK_NO_LBRR                        0   /* No LBRR information for th
is packet                  */
#define SKP_SILK_ADD_LBRR_TO_PLUS1              1   /* Add LBRR for this packet t
o packet n + 1             */
#define SKP_SILK_ADD_LBRR_TO_PLUS2              2   /* Add LBRR for this packet t
o packet n + 2             */

/* Frame termination indicator defines */
#define SKP_SILK_LAST_FRAME                     0   /* Last frames in packet
                           */
#define SKP_SILK_MORE_FRAMES                    1   /* More frames to follow this
 one                       */
#define SKP_SILK_LBRR_VER1                      2   /* LBRR information from pack
et n - 1                   */
#define SKP_SILK_LBRR_VER2                      3   /* LBRR information from pack
et n - 2                   */
#define SKP_SILK_EXT_LAYER                      4   /* Extension layers added
                           */

/* Number of Second order Sections for SWB detection HP filter */
#define NB_SOS                                  3
#define HP_8_KHZ_THRES                          10          /* average energy per
 sample, above 8 kHz       */
#define CONCEC_SWB_SMPLS_THRES                  480 * 15    /* 300 ms
                           */
#define WB_DETECT_ACTIVE_SPEECH_MS_THRES        15000       /* ms of active speec
h needed for WB detection  */

/* Low complexity setting */
#ifdef EMBEDDED_OPT
#   define LOW_COMPLEXITY_ONLY                  1
#else
#   define LOW_COMPLEXITY_ONLY                  0
#endif

/* Activate bandwidth transition filtering for mode switching */
#ifdef EMBEDDED_OPT
```

```
#   define SWITCH_TRANSITION_FILTERING         0
#else
#ifndef FORCE_FS_KHZ
#   define SWITCH_TRANSITION_FILTERING         1
#else
#   define SWITCH_TRANSITION_FILTERING         0
#endif
#endif

/* Decoder Parameters */
#define DEC_HP_ORDER                           2

/* Maximum sampling frequency, should be 16 for embedded */
#define MAX_FS_KHZ                             24

/* Signal Types used by silk */
#define SIG_TYPE_VOICED                        0
#define SIG_TYPE_UNVOICED                      1

/* VAD Types used by silk */
#define NO_VOICE_ACTIVITY                      0
#define VOICE_ACTIVITY                         1

/* number of samples per frame */
#define FRAME_LENGTH_MS                        20 /* 20 ms */
#define MAX_FRAME_LENGTH                       (FRAME_LENGTH_MS * MAX_FS_KHZ)

/* number of lookahead samples for pitch analysis */
#define LA_PITCH_MS                            3
#define LA_PITCH_MAX                           (LA_PITCH_MS * MAX_FS_KHZ)

/* number of lookahead samples for noise shape analysis */
#define LA_SHAPE_MS                            5
#define LA_SHAPE_MAX                           (LA_SHAPE_MS * MAX_FS_KHZ)

/* Order of LPC used in find pitch */
#define FIND_PITCH_LPC_ORDER_MAX               16

/* Length of LPC window used in find pitch */
#define FIND_PITCH_LPC_WIN_MS                  (30 + (LA_PITCH_MS << 1))
#define FIND_PITCH_LPC_WIN_MAX                 (FIND_PITCH_LPC_WIN_MS * MAX_FS_K
HZ)

#define PITCH_EST_COMPLEXITY_HC_MODE           SigProc_PITCH_EST_MAX_COMPLEX
#define PITCH_EST_COMPLEXITY_MC_MODE           SigProc_PITCH_EST_MID_COMPLEX
#define PITCH_EST_COMPLEXITY_LC_MODE           SigProc_PITCH_EST_MIN_COMPLEX


/* Max number of bytes in payload output buffer (may contain multiple frames) */
```

```
#define MAX_ARITHM_BYTES                        1024

#define RANGE_CODER_WRITE_BEYOND_BUFFER         -1
#define RANGE_CODER_CDF_OUT_OF_RANGE            -2
#define RANGE_CODER_NORMALIZATION_FAILED        -3
#define RANGE_CODER_ZERO_INTERVAL_WIDTH         -4
#define RANGE_CODER_DECODER_CHECK_FAILED        -5
#define RANGE_CODER_READ_BEYOND_BUFFER          -6
#define RANGE_CODER_ILLEGAL_SAMPLING_RATE       -7
#define RANGE_CODER_DEC_PAYLOAD_TOO_LONG        -8

/* dB level of lowest gain quantization level */
#define MIN_QGAIN_DB                            6
/* dB level of highest gain quantization level */
#define MAX_QGAIN_DB                            86
/* Number of gain quantization levels */
#define N_LEVELS_QGAIN                          64
/* Max increase in gain quantization index */
#define MAX_DELTA_GAIN_QUANT                    40
/* Max decrease in gain quantization index */
#define MIN_DELTA_GAIN_QUANT                    -4

/* Quantization offsets (multiples of 4) */
#define OFFSET_VL_Q10                           32
#define OFFSET_VH_Q10                           100
#define OFFSET_UVL_Q10                          100
#define OFFSET_UVH_Q10                          256

/* Maximum numbers of iterations used to stabilize a LPC vector */
#define MAX_LPC_STABILIZE_ITERATIONS            20

#define MAX_LPC_ORDER                           16
#define MIN_LPC_ORDER                           10

/* Find Pred Coef defines */
#define LTP_ORDER                               5

/* LTP quantization settings */
#define NB_LTP_CBKS                             3

/* Number of subframes */
#define NB_SUBFR                                4

/* Flag to use harmonic noise shaping */
#define USE_HARM_SHAPING                        1

/* Max LPC order of noise shaping filters */
#define SHAPE_LPC_ORDER_MAX                     16
```

```
#define HARM_SHAPE_FIR_TAPS                     3

/* Length of LPC window used in noise shape analysis */
#define SHAPE_LPC_WIN_MS                        15
#define SHAPE_LPC_WIN_16_KHZ                    (SHAPE_LPC_WIN_MS * 16)
#define SHAPE_LPC_WIN_24_KHZ                    (SHAPE_LPC_WIN_MS * 24)
#define SHAPE_LPC_WIN_MAX                       (SHAPE_LPC_WIN_MS * MAX_FS_KHZ)

/* Maximum number of delayed decision states */
#define DEL_DEC_STATES_MAX                      4

#define LTP_BUF_LENGTH                          512
#define LTP_MASK                                (LTP_BUF_LENGTH - 1)

#define DECISION_DELAY                          32
#define DECISION_DELAY_MASK                     (DECISION_DELAY - 1)

/* number of subframes for excitation entropy coding */
#define SHELL_CODEC_FRAME_LENGTH                16
#define MAX_NB_SHELL_BLOCKS                     (MAX_FRAME_LENGTH / SHELL_CODEC_F
RAME_LENGTH)

/* number of rate levels, for entropy coding of excitation */
#define N_RATE_LEVELS                           10

/* maximum sum of pulses per shell coding frame */
#define MAX_PULSES                              18

#define MAX_MATRIX_SIZE                         MAX_LPC_ORDER /* Max of LPC Order
 and LTP order */

#if( MAX_LPC_ORDER > DECISION_DELAY )
# define NSQ_LPC_BUF_LENGTH                     MAX_LPC_ORDER
#else
# define NSQ_LPC_BUF_LENGTH                     DECISION_DELAY
#endif

/***********************/
/* High pass filtering */
/***********************/
#define HIGH_PASS_INPUT                         1

/**************************/
/* Voice activity detector */
/**************************/
#define VAD_N_BANDS                             4       /* 0-1, 1-2, 2-4, and 4-8
 kHz                        */

#define VAD_INTERNAL_SUBFRAMES_LOG2             2
#define VAD_INTERNAL_SUBFRAMES                  (1 << VAD_INTERNAL_SUBFRAMES_LOG2
)
```

```
#define VAD_NOISE_LEVEL_SMOOTH_COEF_Q16         1024    /* Must be <  4096
                            */
#define VAD_NOISE_LEVELS_BIAS                   50

/* Sigmoid settings */
#define VAD_NEGATIVE_OFFSET_Q5                  128     /* sigmoid is 0 at -128
                            */
#define VAD_SNR_FACTOR_Q16                      45000

/* smoothing for SNR measurement */
#define VAD_SNR_SMOOTH_COEF_Q18                 4096

/******************/
/* NLSF quantizer */
/******************/
#   define NLSF_MSVQ_MAX_CB_STAGES                      10  /* Update manually wh
en changing codebooks      */
#   define NLSF_MSVQ_MAX_VECTORS_IN_STAGE               128 /* Update manually wh
en changing codebooks      */
#   define NLSF_MSVQ_MAX_VECTORS_IN_STAGE_TWO_TO_END    16  /* Update manually wh
en changing codebooks      */

#define NLSF_MSVQ_FLUCTUATION_REDUCTION         1
#define MAX_NLSF_MSVQ_SURVIVORS                 16
#define MAX_NLSF_MSVQ_SURVIVORS_LC_MODE         2
#define MAX_NLSF_MSVQ_SURVIVORS_MC_MODE         4

/* Based on above defines, calculate how much memory is necessary to allocate */
#if( NLSF_MSVQ_MAX_VECTORS_IN_STAGE > ( MAX_NLSF_MSVQ_SURVIVORS_LC_MODE * NLSF_MS
VQ_MAX_VECTORS_IN_STAGE_TWO_TO_END ) )
#   define NLSF_MSVQ_TREE_SEARCH_MAX_VECTORS_EVALUATED_LC_MODE  NLSF_MSVQ_MAX_VEC
TORS_IN_STAGE
#else
#   define NLSF_MSVQ_TREE_SEARCH_MAX_VECTORS_EVALUATED_LC_MODE  MAX_NLSF_MSVQ_SUR
VIVORS_LC_MODE * NLSF_MSVQ_MAX_VECTORS_IN_STAGE_TWO_TO_END
#endif

#if( NLSF_MSVQ_MAX_VECTORS_IN_STAGE > ( MAX_NLSF_MSVQ_SURVIVORS * NLSF_MSVQ_MAX_V
ECTORS_IN_STAGE_TWO_TO_END ) )
#   define NLSF_MSVQ_TREE_SEARCH_MAX_VECTORS_EVALUATED  NLSF_MSVQ_MAX_VECTORS_IN_
STAGE
#else
#   define NLSF_MSVQ_TREE_SEARCH_MAX_VECTORS_EVALUATED  MAX_NLSF_MSVQ_SURVIVORS *
 NLSF_MSVQ_MAX_VECTORS_IN_STAGE_TWO_TO_END
#endif

#define NLSF_MSVQ_SURV_MAX_REL_RD               4

/* Transition filtering for mode switching */
#if SWITCH_TRANSITION_FILTERING
#  define TRANSITION_TIME_UP_MS             5120 // 5120 = 64 * FRAME_LENGTH_MS *
 ( TRANSITION_INT_NUM - 1 ) = 64*(20*4)
#  define TRANSITION_TIME_DOWN_MS           2560 // 2560 = 32 * FRAME_LENGTH_MS *
 ( TRANSITION_INT_NUM - 1 ) = 32*(20*4)
#  define TRANSITION_NB                     3 /* Hardcoded in tables */
#  define TRANSITION_NA                     2 /* Hardcoded in tables */
#  define TRANSITION_INT_NUM                5 /* Hardcoded in tables */
#  define TRANSITION_FRAMES_UP          ( TRANSITION_TIME_UP_MS   / FRAME_LENGTH_
MS )
#  define TRANSITION_FRAMES_DOWN        ( TRANSITION_TIME_DOWN_MS / FRAME_LENGTH_
MS )
#  define TRANSITION_INT_STEPS_UP       ( TRANSITION_FRAMES_UP    / ( TRANSITION_
```

```
INT_NUM - 1 )  )
#  define TRANSITION_INT_STEPS_DOWN     ( TRANSITION_FRAMES_DOWN  / ( TRANSITION_
INT_NUM - 1 )  )
```

```
#endif

/* Row based */
#define matrix_ptr(Matrix_base_adr, row, column, N)         *(Matrix_base_adr + (
(row)*(N)+(column)))
#define matrix_adr(Matrix_base_adr, row, column, N)          (Matrix_base_adr + (
(row)*(N)+(column)))

/* Column based */
#ifndef matrix_c_ptr
#   define matrix_c_ptr(Matrix_base_adr, row, column, M)    *(Matrix_base_adr + (
(row)+(M)*(column)))
#endif
#define matrix_c_adr(Matrix_base_adr, row, column, M)        (Matrix_base_adr + (
(row)+(M)*(column)))

/* BWE factors to apply after packet loss */
#define BWE_AFTER_LOSS_Q16                                63570

#ifdef __cplusplus
}
#endif

#endif
```

A.32. src/SKP_Silk_define_FIX.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
```

```
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_DEFINE_FIX_H
#define SKP_SILK_DEFINE_FIX_H

#ifdef __cplusplus
extern "C"
{
#endif

/* Head room for correlations                          */
#define LTP_CORRS_HEAD_ROOM                           2
#define LPC_CORRS_HEAD_ROOM                           10

#define WB_DETECT_ACTIVE_SPEECH_LEVEL_THRES_Q8        179     // 179.2_Q8 = 0.7
f required speech activity for counting frame as active

/* DTX settings */
#define SPEECH_ACTIVITY_DTX_THRES_Q8                  26      // 25.60_Q8 = 0.1
f

#define LBRR_SPEECH_ACTIVITY_THRES_Q8                 128

/* level of noise floor for whitening filter LPC analysis in pitch analysis */
#define FIND_PITCH_WHITE_NOISE_FRACTION_Q16           66

/* bandwdith expansion for whitening filter in pitch analysis */
#define FIND_PITCH_BANDWITH_EXPANSION_Q16             64881

/* Threshold used by pitch estimator for early escape */
#define FIND_PITCH_CORRELATION_THRESHOLD_Q16_HC_MODE   45875       // 0.7
#define FIND_PITCH_CORRELATION_THRESHOLD_Q16_MC_MODE   49152       // 0.75
#define FIND_PITCH_CORRELATION_THRESHOLD_Q16_LC_MODE   52429       // 0.8

/* Regualarization factor for correlation matrix. Equivalent to adding noise at -
50 dB */
#define FIND_LTP_COND_FAC_Q31                         21475
#define FIND_LPC_COND_FAC_Q32                         257698       // 6e-5

/* Find Pred Coef defines */
#define INACTIVE_BWExp_Q16                            64225       // 0.98
#define ACTIVE_BWExp_Q16                              65470       // 0.999
#define LTP_DAMPING_Q16                               66
#define LTP_SMOOTHING_Q26                             6710886

/* LTP quantization settings */
#define MU_LTP_QUANT_NB_Q8                            8
#define MU_LTP_QUANT_MB_Q8                            6
#define MU_LTP_QUANT_WB_Q8                            5
```

```
#define MU_LTP_QUANT_SWB_Q8                             4

/***********************/
/* High pass filtering */
/***********************/
/* Smoothing parameters for low end of pitch frequency range estimation */
#define VARIABLE_HP_SMTH_COEF1_Q16                      6554    // 0.1
#define VARIABLE_HP_SMTH_COEF2_Q16                      983     // 0.015

/* Min and max values for low end of pitch frequency range estimation */
#define VARIABLE_HP_MIN_FREQ_Q0                         80
#define VARIABLE_HP_MAX_FREQ_Q0                         150

/* Max absolute difference between log2 of pitch frequency and smoother state, to
 enter the smoother */
#define VARIABLE_HP_MAX_DELTA_FREQ_Q7                   51      // 0.4 in Q7

/* Defines for CN generation */
#define CNG_BUF_MASK_MAX                                255             /* 2^floo
r(log2(MAX_FRAME_LENGTH))  */
#define CNG_GAIN_SMTH_Q16                               4634            /* 0.25^(
1/4)                        */
#define CNG_NLSF_SMTH_Q16                               16348           /* 0.25
                            */

#ifdef __cplusplus
}
#endif

#endif
```

A.33. src/SKP_Silk_detect_SWB_input.c

```
/**********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
```

```
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*
 * Detect SWB input by measuring energy above 8 kHz.
 */

#include "SKP_Silk_main.h"

void SKP_Silk_detect_SWB_input(
    SKP_Silk_detect_SWB_state   *psSWBdetect,   /* (I/O) encoder state  */
    const SKP_int16             samplesIn[],    /* (I) input to encoder */
    SKP_int                     nSamplesIn      /* (I) length of input */
)
{
    SKP_int     HP_8_kHz_len, i;
    SKP_int16   in_HP_8_kHz[ MAX_FRAME_LENGTH ];
    SKP_int32   energy_32, shift;

    /* High pass filter with cutoff at 8 khz */
    HP_8_kHz_len = SKP_min_int( nSamplesIn, MAX_FRAME_LENGTH );
    HP_8_kHz_len = SKP_max_int( HP_8_kHz_len, 0 );

    /* Cutoff around 9 khz */
    /* A = conv(conv([8192,14613, 6868], [8192,12883, 7337]), [8192,11586, 7911])
; */
    /* B = conv(conv([575, -948, 575], [575, -221, 575]), [575, 104, 575]); */
    SKP_Silk_biquad( samplesIn, SKP_Silk_SWB_detect_B_HP_Q13[ 0 ], SKP_Silk_SWB_d
etect_A_HP_Q13[ 0 ],
        psSWBdetect->S_HP_8_kHz[ 0 ], in_HP_8_kHz, HP_8_kHz_len );
    for( i = 1; i < NB_SOS; i++ ) {
        SKP_Silk_biquad( in_HP_8_kHz, SKP_Silk_SWB_detect_B_HP_Q13[ i ], SKP_Silk
_SWB_detect_A_HP_Q13[ i ],
            psSWBdetect->S_HP_8_kHz[ i ], in_HP_8_kHz, HP_8_kHz_len );
    }

    /* Calculate energy in HP signal */
    SKP_Silk_sum_sqr_shift( &energy_32, &shift, in_HP_8_kHz, HP_8_kHz_len );

    /* Count concecutive samples above threshold, after adjusting threshold for n
umber of input samples and shift */
    if( energy_32 > SKP_RSHIFT( SKP_SMULBB( HP_8_KHZ_THRES, HP_8_kHz_len ), shift
 ) ) {
        psSWBdetect->ConsecSmplsAboveThres += nSamplesIn;
        if( psSWBdetect->ConsecSmplsAboveThres > CONCEC_SWB_SMPLS_THRES ) {
            psSWBdetect->SWB_detected = 1;
```

```
        }
    } else {
        psSWBdetect->ConsecSmplsAboveThres -= nSamplesIn;
        psSWBdetect->ConsecSmplsAboveThres = SKP_max( psSWBdetect->ConsecSmplsAbo
veThres, 0 );
    }

    /* If sufficient speech activity and no SWB detected, we detect the signal as
 being WB */
    if( ( psSWBdetect->ActiveSpeech_ms > WB_DETECT_ACTIVE_SPEECH_MS_THRES ) && (
psSWBdetect->SWB_detected == 0 ) ) {
        psSWBdetect->WB_detected = 1;
    }
}
```

A.34. src/SKP_Silk_enc_API.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/


#include "SKP_Silk_define.h"
#include "SKP_Silk_main_FIX.h"
#include "SKP_Silk_SDK_API.h"
#include "SKP_Silk_control.h"
```

```
#include "SKP_Silk_typedef.h"
#include "SKP_Silk_structs.h"
#define SKP_Silk_EncodeControlStruct SKP_SILK_SDK_EncControlStruct

/***************************************/
/* Encoder functions                   */
/***************************************/

SKP_int SKP_Silk_SDK_Get_Encoder_Size( SKP_int *encSizeBytes )
{
    SKP_int ret = 0;

    *encSizeBytes = sizeof( SKP_Silk_encoder_state_FIX );

    return ret;
}


/***************************************/
/* Read control structure from encoder */
/***************************************/
SKP_int SKP_Silk_SDK_QueryEncoder(
    const void *encState,                       /* I:   State Vector
                        */
    SKP_Silk_EncodeControlStruct *encStatus     /* O:   Control Structure
                        */
)
{
    SKP_Silk_encoder_state_FIX *psEnc;
    SKP_int ret = 0;

    psEnc = ( SKP_Silk_encoder_state_FIX* )encState;

    encStatus->sampleRate = ( unsigned short )SKP_SMULBB( psEnc->sCmn.fs_kHz, 100
0 );                        /* convert kHz -> Hz */
    encStatus->packetSize = ( unsigned short )SKP_SMULBB( psEnc->sCmn.fs_kHz, psE
nc->sCmn.PacketSize_ms );  /* convert samples -> ms */
    encStatus->bitRate    = ( unsigned short )psEnc->sCmn.TargetRate_bps;
    encStatus->packetLossPercentage = psEnc->sCmn.PacketLoss_perc;
    encStatus->complexity           = psEnc->sCmn.Complexity;

    return ret;
}

/*************************/
/* Init or Reset encoder */
/*************************/
SKP_int SKP_Silk_SDK_InitEncoder(
    void                            *encState,          /* I/O: State
                                */
    SKP_Silk_EncodeControlStruct    *encStatus          /* O:   Control structure
                                */
)
{
```

```
    SKP_Silk_encoder_state_FIX *psEnc;
    SKP_int ret = 0;


    psEnc = ( SKP_Silk_encoder_state_FIX* )encState;

    /* Reset Encoder */
    if( ret += SKP_Silk_init_encoder_FIX( psEnc ) ) {
        SKP_assert( 0 );
    }

    /* Read Control structure */
    if( ret += SKP_Silk_SDK_QueryEncoder( encState, encStatus ) ) {
        SKP_assert( 0 );
    }


    return ret;
}

/**************************/
/* Encode frame with Silk */
/**************************/
SKP_int SKP_Silk_SDK_Encode(
    void                                *encState,      /* I/O: State
                                */
    const SKP_Silk_EncodeControlStruct  *encControl,    /* I:   Control structure
                                */
    const SKP_int16                     *samplesIn,     /* I:   Speech sample inp
ut vector                       */
    SKP_int                             nSamplesIn,     /* I:   Number of samples
 in input vector                */
    SKP_uint8                           *outData,       /* O:   Encoded output ve
ctor                            */
    SKP_int16                           *nBytesOut      /* I/O: Number of bytes i
n outData (input: Max bytes)    */
)
{
    SKP_int   API_fs_kHz, PacketSize_ms, PacketLoss_perc, UseInBandFec, UseDTX, r
et = 0;
    SKP_int   nSamplesToBuffer, Complexity, input_ms, nSamplesFromInput = 0;
    SKP_int32 TargetRate_bps;
    SKP_int16 MaxBytesOut;
    SKP_Silk_encoder_state_FIX *psEnc = ( SKP_Silk_encoder_state_FIX* )encState;


    SKP_assert( encControl != NULL );

    /* Check sampling frequency first, to avoid divide by zero later */
    if( ( encControl->sampleRate !=  8000 ) && ( encControl->sampleRate != 12000
) &&
        ( encControl->sampleRate != 16000 ) && ( encControl->sampleRate != 24000
) ) {
        ret = SKP_SILK_ENC_FS_NOT_SUPPORTED;
        SKP_assert( 0 );
        return( ret );
    }
```

```
    /* Set Encoder parameters from Control structure */
    API_fs_kHz      = SKP_DIV32_16( ( SKP_int )encControl->sampleRate, 1000 );
       /* convert Hz -> kHz */
    PacketSize_ms   = SKP_DIV32_16( ( SKP_int )encControl->packetSize, API_fs_kHz
 );    /* convert samples -> ms */
    TargetRate_bps  =             ( SKP_int32 )encControl->bitRate;
    PacketLoss_perc =               ( SKP_int )encControl->packetLossPercentage;
    UseInBandFec    =               ( SKP_int )encControl->useInBandFEC;
    Complexity      =               ( SKP_int )encControl->complexity;
    UseDTX          =               ( SKP_int )encControl->useDTX;

    /* Only accept input lengths that are multiplum of 10 ms */
    input_ms = SKP_DIV32_16( nSamplesIn, API_fs_kHz );
    if( ( input_ms % 10) != 0 || nSamplesIn < 0 ) {
        ret = SKP_SILK_ENC_INPUT_INVALID_NO_OF_SAMPLES;
        SKP_assert( 0 );
        return( ret );
    }

    /* Make sure no more than one packet can be produced */
    if( nSamplesIn > SKP_SMULBB( psEnc->sCmn.PacketSize_ms, API_fs_kHz ) ) {
        ret = SKP_SILK_ENC_INPUT_INVALID_NO_OF_SAMPLES;
        SKP_assert( 0 );
        return( ret );
    }

    if( ( ret = SKP_Silk_control_encoder_FIX( psEnc, API_fs_kHz, PacketSize_ms, T
argetRate_bps,
                    PacketLoss_perc, UseInBandFec, UseDTX, input_ms, Complexity )
 ) != 0 ) {
        SKP_assert( 0 );
        return( ret );
    }

    /* Detect energy above 8 kHz */
    if( encControl->sampleRate == 24000 && psEnc->sCmn.sSWBdetect.SWB_detected ==
 0 && psEnc->sCmn.sSWBdetect.WB_detected == 0 ) {
        SKP_Silk_detect_SWB_input( &psEnc->sCmn.sSWBdetect, samplesIn, ( SKP_int
)nSamplesIn );
    }

    /* Input buffering/resampling and encoding */
    MaxBytesOut = 0;                    /* return 0 output bytes if no encoder ca
lled */
    while( 1 ) {
        /* Resample/buffer */
        nSamplesToBuffer = psEnc->sCmn.frame_length - psEnc->sCmn.inputBufIx;
        if( encControl->sampleRate == SKP_SMULBB( psEnc->sCmn.fs_kHz, 1000 ) ) {
            /* Same sample frequency - copy the data */
            nSamplesToBuffer  = SKP_min_int( nSamplesToBuffer, nSamplesIn );
            nSamplesFromInput = nSamplesToBuffer;
            SKP_memcpy( &psEnc->sCmn.inputBuf[ psEnc->sCmn.inputBufIx ], samplesI
n, SKP_SMULBB( nSamplesToBuffer, sizeof( SKP_int16 ) ) );
        } else if( encControl->sampleRate == 24000 && psEnc->sCmn.fs_kHz == 16 )
{
            /* Resample the data from 24 kHz to 16 kHz */
            nSamplesToBuffer  = SKP_min_int( nSamplesToBuffer, SKP_SMULWB( SKP_LS
HIFT( nSamplesIn, 1 ), 21846 ) ); // 21846 = ceil(2/3)*2^15
```

```
            nSamplesFromInput = SKP_RSHIFT( SKP_SMULBB( nSamplesToBuffer, 3 ), 1
);
#if LOW_COMPLEXITY_ONLY
            {
                SKP_int16 scratch[ MAX_FRAME_LENGTH + SigProc_Resample_2_3_coarse
_NUM_FIR_COEFS - 1 ];
                SKP_assert( nSamplesFromInput <= MAX_FRAME_LENGTH );
                SKP_Silk_resample_2_3_coarse( &psEnc->sCmn.inputBuf[ psEnc->sCmn.
inputBufIx ], psEnc->sCmn.resample24To16state,
                    samplesIn, nSamplesFromInput, scratch );
            }
#else
            SKP_Silk_resample_2_3( &psEnc->sCmn.inputBuf[ psEnc->sCmn.inputBufIx
], psEnc->sCmn.resample24To16state,
                samplesIn, nSamplesFromInput );
#endif
        } else if( encControl->sampleRate == 24000 && psEnc->sCmn.fs_kHz == 12 )
{
            SKP_int32 scratch[ 3 * MAX_FRAME_LENGTH ];
            /* Resample the data from 24 kHz to 12 kHz */
            nSamplesToBuffer  = SKP_min_int( nSamplesToBuffer, SKP_RSHIFT( nSampl
esIn, 1 ) );
            nSamplesFromInput = SKP_LSHIFT16( nSamplesToBuffer, 1 );
            SKP_Silk_resample_1_2_coarse( samplesIn, psEnc->sCmn.resample24To12st
ate,
                &psEnc->sCmn.inputBuf[ psEnc->sCmn.inputBufIx ], scratch, nSample
sToBuffer );
        } else if( encControl->sampleRate == 24000 && psEnc->sCmn.fs_kHz == 8 ) {
            /* Resample the data from 24 kHz to 8 kHz */
            nSamplesToBuffer  = SKP_min_int( nSamplesToBuffer, SKP_DIV32_16( nSam
plesIn, 3 ) );
            nSamplesFromInput = SKP_SMULBB( nSamplesToBuffer, 3 );
            SKP_Silk_resample_1_3( &psEnc->sCmn.inputBuf[ psEnc->sCmn.inputBufIx
], psEnc->sCmn.resample24To8state,
                samplesIn, nSamplesFromInput);
        } else if( encControl->sampleRate == 16000 && psEnc->sCmn.fs_kHz == 12 )
{
            /* Resample the data from 16 kHz to 12 kHz */
            nSamplesToBuffer  = SKP_min_int( nSamplesToBuffer, SKP_RSHIFT( SKP_SM
ULBB( nSamplesIn, 3 ), 2 ) );
            nSamplesFromInput = SKP_SMULWB( SKP_LSHIFT16( nSamplesToBuffer, 2 ),
21846 ); // 21846 = ceil((1/3)*2^16)
            SKP_Silk_resample_3_4( &psEnc->sCmn.inputBuf[ psEnc->sCmn.inputBufIx
], psEnc->sCmn.resample16To12state,
                samplesIn, nSamplesFromInput );
        } else if( encControl->sampleRate == 16000 && psEnc->sCmn.fs_kHz == 8 ) {
            SKP_int32 scratch[ 3 * MAX_FRAME_LENGTH ];
            /* Resample the data from 16 kHz to 8 kHz */
            nSamplesToBuffer  = SKP_min_int( nSamplesToBuffer, SKP_RSHIFT( nSampl
esIn, 1 ) );
            nSamplesFromInput = SKP_LSHIFT16( nSamplesToBuffer, 1 );
            SKP_Silk_resample_1_2_coarse( samplesIn, psEnc->sCmn.resample16To8sta
te,
                &psEnc->sCmn.inputBuf[ psEnc->sCmn.inputBufIx ], scratch, nSample
sToBuffer );
        } else if( encControl->sampleRate == 12000 && psEnc->sCmn.fs_kHz == 8 ) {
            /* Resample the data from 12 kHz to 8 kHz */
            nSamplesToBuffer  = SKP_min_int( nSamplesToBuffer, SKP_SMULWB( SKP_LS
HIFT( nSamplesIn, 1 ), 21846 ) );
            nSamplesFromInput = SKP_RSHIFT( SKP_SMULBB( nSamplesToBuffer, 3 ), 1
);
#if LOW_COMPLEXITY_ONLY
```

```
            {
                SKP_int16 scratch[ MAX_FRAME_LENGTH + SigProc_Resample_2_3_coarse
_NUM_FIR_COEFS - 1 ];
                SKP_assert( nSamplesFromInput <= MAX_FRAME_LENGTH );
                SKP_Silk_resample_2_3_coarse( &psEnc->sCmn.inputBuf[ psEnc->sCmn.
inputBufIx ], psEnc->sCmn.resample12To8state,
                    samplesIn, nSamplesFromInput, scratch );
```

```
            }
#else
            SKP_Silk_resample_2_3( &psEnc->sCmn.inputBuf[ psEnc->sCmn.inputBufIx
], psEnc->sCmn.resample12To8state,
                samplesIn, nSamplesFromInput );
#endif
        }
        samplesIn  += nSamplesFromInput;
        nSamplesIn -= nSamplesFromInput;
        psEnc->sCmn.inputBufIx += nSamplesToBuffer;

        /* Silk encoder */
        if( psEnc->sCmn.inputBufIx >= psEnc->sCmn.frame_length ) {
            /* Enough data in input buffer, so encode */
            if( MaxBytesOut == 0 ) {
                /* No payload obtained so far */
                MaxBytesOut = *nBytesOut;
                if( ( ret = SKP_Silk_encode_frame_FIX( psEnc, outData, &MaxBytesO
ut, psEnc->sCmn.inputBuf ) ) != 0 ) {
                    SKP_assert( 0 );
                }
            } else {
                /* outData already contains a payload */
                if( ( ret = SKP_Silk_encode_frame_FIX( psEnc, outData, nBytesOut,
 psEnc->sCmn.inputBuf ) ) != 0 ) {
                    SKP_assert( 0 );
                }
                /* Check that no second payload was created */
                SKP_assert( *nBytesOut == 0 );
            }
            psEnc->sCmn.inputBufIx = 0;
        } else {
            break;
        }
    }

    *nBytesOut = MaxBytesOut;
    if( psEnc->sCmn.useDTX && psEnc->sCmn.inDTX ) {
        /* Dtx simulation */
        *nBytesOut = 0;
    }


    return ret;
}
```

A.35. src/SKP_Silk_encode_frame_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"
/*****************/
/* Encode frame */
/*****************/
SKP_int SKP_Silk_encode_frame_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,             /* I/O  Pointer to Silk F
IX encoder state           */
    SKP_uint8                       *pCode,             /* O    Pointer to payloa
d                          */
    SKP_int16                       *pnBytesOut,        /* I/O  Pointer to number
 of payload bytes          */
                                                        /*      input: max length
; output: used             */
    const SKP_int16                 *pIn                /* I    Pointer to input
speech frame               */
)
{
    SKP_Silk_encoder_control_FIX sEncCtrl;
    SKP_int     i, nBytes, ret = 0;
    SKP_int16   *x_frame, *res_pitch_frame;
    SKP_int16   xfw[ MAX_FRAME_LENGTH ];
    SKP_int16   pIn_HP[ MAX_FRAME_LENGTH ];
    SKP_int16   res_pitch[ 2 * MAX_FRAME_LENGTH + LA_PITCH_MAX ];
    SKP_int     LBRR_idx, frame_terminator, SNR_dB_Q7;
    const SKP_uint16 *FrameTermination_CDF;
```

```
    /* Low bitrate redundancy parameters */
    SKP_uint8   LBRRpayload[ MAX_ARITHM_BYTES ];
    SKP_int16   nBytesLBRR;

    //SKP_int32   Seed[ MAX_LAYERS ];
    sEncCtrl.sCmn.Seed = psEnc->sCmn.frameCounter++ & 3;


    /*****************************************************************/
    /* Setup Input Pointers, and insert frame in input buffer    */
    /*****************************************************************/
    x_frame         = psEnc->x_buf + psEnc->sCmn.frame_length; /* start of frame
to encode */
    res_pitch_frame = res_pitch    + psEnc->sCmn.frame_length; /* start of pitch
LPC residual frame */

    /*****************************/
    /* Voice Activity Detection */
    /*****************************/
    ret = SKP_Silk_VAD_GetSA_Q8( &psEnc->sCmn.sVAD, &psEnc->speech_activity_Q8, &
SNR_dB_Q7,
                                 sEncCtrl.input_quality_bands_Q15, &sEncCtrl.inpu
t_tilt_Q15,
                                 pIn,psEnc->sCmn.frame_length );

    /*******************************************/
    /* High-pass filtering of the input signal */
    /*******************************************/
#if HIGH_PASS_INPUT
    /* Variable high-pass filter */
    SKP_Silk_HP_variable_cutoff_FIX( psEnc, &sEncCtrl, pIn_HP, pIn );
#else
    SKP_memcpy( pIn_HP, pIn,psEnc->sCmn.frame_length * sizeof( SKP_int16 ) );
#endif

#if SWITCH_TRANSITION_FILTERING
    /* Ensure smooth bandwidth transitions */
    SKP_Silk_LP_variable_cutoff( &psEnc->sCmn.sLP, x_frame + psEnc->sCmn.la_shape
, pIn_HP, psEnc->sCmn.frame_length );
#else
    SKP_memcpy( x_frame + psEnc->sCmn.la_shape, pIn_HP,psEnc->sCmn.frame_length *
 sizeof( SKP_int16 ) );
#endif

    /*****************************************/
    /* Find pitch lags, initial LPC analysis */
    /*****************************************/
    SKP_Silk_find_pitch_lags_FIX( psEnc, &sEncCtrl, res_pitch, x_frame );

    /************************/
    /* Noise shape analysis */
    /************************/
    SKP_Silk_noise_shape_analysis_FIX( psEnc, &sEncCtrl, res_pitch_frame, x_frame
 );
```

```
    /*****************************************/
    /* Prefiltering for noise shaper         */
    /*****************************************/
    SKP_Silk_prefilter_FIX( psEnc, &sEncCtrl, xfw, x_frame );


    /*************************************************/
    /* Find linear prediction coefficients (LPC + LTP) */
    /*************************************************/
    SKP_Silk_find_pred_coefs_FIX( psEnc, &sEncCtrl, res_pitch );

    /*************************************/
    /* Process gains                     */
    /*************************************/
    SKP_Silk_process_gains_FIX( psEnc, &sEncCtrl );

    psEnc->sCmn.sigtype[         psEnc->sCmn.nFramesInPayloadBuf ] = sEncCtrl.sCm
n.sigtype;
    psEnc->sCmn.QuantOffsetType[ psEnc->sCmn.nFramesInPayloadBuf ] = sEncCtrl.sCm
n.QuantOffsetType;

    /*************************************/
    /* Low Bitrate Redundant Encoding    */
    /*************************************/
    nBytesLBRR = MAX_ARITHM_BYTES;
    SKP_Silk_LBRR_encode_FIX( psEnc, &sEncCtrl, LBRRpayload, &nBytesLBRR, xfw );

    /*************************************/
    /* Noise shaping quantization        */
    /*************************************/
    psEnc->NoiseShapingQuantizer( &psEnc->sCmn, &sEncCtrl.sCmn, &psEnc->sNSQ, xfw
,
        &psEnc->sCmn.q[ psEnc->sCmn.nFramesInPayloadBuf *psEnc->sCmn.frame_length
 ], sEncCtrl.sCmn.NLSFInterpCoef_Q2,
        sEncCtrl.PredCoef_Q12[ 0 ], sEncCtrl.LTPCoef_Q14, sEncCtrl.AR2_Q13, sEncC
trl.HarmShapeGain_Q14,
        sEncCtrl.Tilt_Q14, sEncCtrl.LF_shp_Q14, sEncCtrl.Gains_Q16, sEncCtrl.Lamb
da_Q10,
        sEncCtrl.LTP_scale_Q14 );

    /************************************************/
    /* Convert speech activity into VAD and DTX flags */
    /************************************************/
    if( psEnc->speech_activity_Q8 < SPEECH_ACTIVITY_DTX_THRES_Q8 ) {
        psEnc->sCmn.vadFlag = NO_VOICE_ACTIVITY;
        psEnc->sCmn.noSpeechCounter++;
        if( psEnc->sCmn.noSpeechCounter > NO_SPEECH_FRAMES_BEFORE_DTX ) {
            psEnc->sCmn.inDTX = 1;
        }
        if( psEnc->sCmn.noSpeechCounter > MAX_CONSECUTIVE_DTX ) {
            psEnc->sCmn.noSpeechCounter = 0;
            psEnc->sCmn.inDTX           = 0;
        }
    } else {
```

```
        psEnc->sCmn.noSpeechCounter = 0;
        psEnc->sCmn.inDTX           = 0;
        psEnc->sCmn.vadFlag         = VOICE_ACTIVITY;
    }

    /*****************************************/
    /* Initialize arithmetic coder           */
    /*****************************************/
    if( psEnc->sCmn.nFramesInPayloadBuf == 0 ) {
        SKP_Silk_range_enc_init( &psEnc->sCmn.sRC );
        psEnc->sCmn.nBytesInPayloadBuf = 0;
    }

    /*****************************************/
    /* Encode Parameters                     */
    /*****************************************/
    if( psEnc->sCmn.bitstream_v == BIT_STREAM_V4 ) {
        SKP_Silk_encode_parameters_v4( &psEnc->sCmn, &sEncCtrl.sCmn, &psEnc->sCmn
.sRC );
        FrameTermination_CDF = SKP_Silk_FrameTermination_v4_CDF;
    } else {
        SKP_Silk_encode_parameters( &psEnc->sCmn, &sEncCtrl.sCmn, &psEnc->sCmn.sR
C,
            &psEnc->sCmn.q[ psEnc->sCmn.nFramesInPayloadBuf *psEnc->sCmn.frame_le
ngth ] );
        FrameTermination_CDF = SKP_Silk_FrameTermination_CDF;
    }

    /*****************************************/
    /* Update Buffers and State              */
    /*****************************************/
    /* Update Input buffer */
    SKP_memmove( psEnc->x_buf, &psEnc->x_buf[ psEnc->sCmn.frame_length ], ( psEnc
->sCmn.frame_length + psEnc->sCmn.la_shape ) * sizeof( SKP_int16 ) );

    /* parameters needed for next frame */
    psEnc->sCmn.prev_sigtype            = sEncCtrl.sCmn.sigtype;
    psEnc->sCmn.prevLag                 = sEncCtrl.sCmn.pitchL[ NB_SUBFR - 1];
    psEnc->sCmn.first_frame_after_reset = 0;

    if( psEnc->sCmn.sRC.error ) {
        /* encoder returned error: clear payload buffer */
        psEnc->sCmn.nFramesInPayloadBuf = 0;
    } else {
        psEnc->sCmn.nFramesInPayloadBuf++;
    }

    /*****************************************/
    /* finalize payload and copy to output  */
    /*****************************************/
    if( psEnc->sCmn.nFramesInPayloadBuf * FRAME_LENGTH_MS >= psEnc->sCmn.PacketSi
ze_ms ) {
```

```
        LBRR_idx = ( psEnc->sCmn.oldest_LBRR_idx + 1 ) & LBRR_IDX_MASK;

        /* Check if FEC information should be added */
        frame_terminator = SKP_SILK_LAST_FRAME;
        if( psEnc->sCmn.LBRR_buffer[ LBRR_idx ].usage == SKP_SILK_ADD_LBRR_TO_PLU
S1 ) {
            frame_terminator = SKP_SILK_LBRR_VER1;
        }
        if( psEnc->sCmn.LBRR_buffer[ psEnc->sCmn.oldest_LBRR_idx ].usage == SKP_S
ILK_ADD_LBRR_TO_PLUS2 ) {
            frame_terminator = SKP_SILK_LBRR_VER2;
            LBRR_idx = psEnc->sCmn.oldest_LBRR_idx;
        }
        /* Add the frame termination info to stream */
        SKP_Silk_range_encoder( &psEnc->sCmn.sRC, frame_terminator, FrameTerminat
ion_CDF );

        if( psEnc->sCmn.bitstream_v == BIT_STREAM_V4 ) {
            /* Code excitation signal */
            for( i = 0; i <psEnc->sCmn.nFramesInPayloadBuf; i++ ) {
                SKP_Silk_encode_pulses( &psEnc->sCmn.sRC, psEnc->sCmn.sigtype[ i
],psEnc->sCmn.QuantOffsetType[ i ],
                    &psEnc->sCmn.q[ i * psEnc->sCmn.frame_length],psEnc->sCmn.fra
me_length );
            }
        }
        /* payload length so far */
        SKP_Silk_range_coder_get_length( &psEnc->sCmn.sRC, &nBytes );

        /* check that there is enough space in external output buffer, and move d
ata */
        if( *pnBytesOut >= nBytes ) {
            SKP_Silk_range_enc_wrap_up( &psEnc->sCmn.sRC );
            SKP_memcpy( pCode, psEnc->sCmn.sRC.buffer, nBytes * sizeof( SKP_uint8
 ) );

            if( frame_terminator > SKP_SILK_MORE_FRAMES &&
                    *pnBytesOut >= nBytes + psEnc->sCmn.LBRR_buffer[ LBRR_idx ].n
Bytes ) {
                /* Get old packet and add to payload. */
                SKP_memcpy( &pCode[ nBytes ],
                    psEnc->sCmn.LBRR_buffer[ LBRR_idx ].payload,
                    psEnc->sCmn.LBRR_buffer[ LBRR_idx ].nBytes * sizeof( SKP_uint
8 ) );
                nBytes += psEnc->sCmn.LBRR_buffer[ LBRR_idx ].nBytes;
            }

            *pnBytesOut = nBytes;

            /* Update FEC buffer */
            SKP_memcpy( psEnc->sCmn.LBRR_buffer[ psEnc->sCmn.oldest_LBRR_idx ].pa
yload, LBRRpayload,
                nBytesLBRR * sizeof( SKP_uint8 ) );
            psEnc->sCmn.LBRR_buffer[ psEnc->sCmn.oldest_LBRR_idx ].nBytes = nByte
sLBRR;
            /* This line tells describes how FEC should be used */
            psEnc->sCmn.LBRR_buffer[ psEnc->sCmn.oldest_LBRR_idx ].usage = sEncCt
rl.sCmn.LBRR_usage;
            psEnc->sCmn.oldest_LBRR_idx = ( psEnc->sCmn.oldest_LBRR_idx + 1 ) & L
BRR_IDX_MASK;
```

```
            /* Reset number of frames in payload buffer */
            psEnc->sCmn.nFramesInPayloadBuf = 0;
        } else {
            /* Not enough space: Payload will be discarded */
            *pnBytesOut = 0;
            nBytes      = 0;
            psEnc->sCmn.nFramesInPayloadBuf = 0;
            ret = SKP_SILK_ENC_PAYLOAD_BUF_TOO_SHORT;
        }
    } else {
        /* no payload for you this time */
        *pnBytesOut = 0;

        /* Encode that more frames follows */
        frame_terminator = SKP_SILK_MORE_FRAMES;
        SKP_Silk_range_encoder( &psEnc->sCmn.sRC, frame_terminator, FrameTerminat
ion_CDF );

        /* payload length so far */
        SKP_Silk_range_coder_get_length( &psEnc->sCmn.sRC, &nBytes );

        if( psEnc->sCmn.bitstream_v == BIT_STREAM_V4 ) {
            /* Take into account the q signal that isnt in the bitstream yet */
            nBytes +=  SKP_Silk_pulses_to_bytes( &psEnc->sCmn,
                &psEnc->sCmn.q[ (psEnc->sCmn.nFramesInPayloadBuf - 1) * psEnc->sC
mn.frame_length ] );
        }
    }

    /* Check for arithmetic coder errors */
    if( psEnc->sCmn.sRC.error ) {
        ret = SKP_SILK_ENC_INTERNAL_ERROR;
    }

    /* simulate number of ms buffered in channel because of exceeding TargetRate
*/
    SKP_assert(  ( 8 * 1000 * ( (SKP_int64)nBytes - (SKP_int64)psEnc->sCmn.nBytes
InPayloadBuf ) ) ==
        SKP_SAT32( 8 * 1000 * ( (SKP_int64)nBytes - (SKP_int64)psEnc->sCmn.nBytes
InPayloadBuf ) ) );
    SKP_assert( psEnc->sCmn.TargetRate_bps > 0 );
    psEnc->BufferedInChannel_ms   += SKP_DIV32( 8 * 1000 * ( nBytes -psEnc->sCmn.
nBytesInPayloadBuf ),psEnc->sCmn.TargetRate_bps );
    psEnc->BufferedInChannel_ms   -= FRAME_LENGTH_MS;
    psEnc->BufferedInChannel_ms    = SKP_LIMIT( psEnc->BufferedInChannel_ms, 0, 1
00 );
    psEnc->sCmn.nBytesInPayloadBuf = nBytes;

    if( psEnc->speech_activity_Q8 > WB_DETECT_ACTIVE_SPEECH_LEVEL_THRES_Q8 ) {
        psEnc->sCmn.sSWBdetect.ActiveSpeech_ms = SKP_ADD_POS_SAT32( psEnc->sCmn.s
SWBdetect.ActiveSpeech_ms, FRAME_LENGTH_MS );
    }


    return( ret );
}
```

```
/* Low BitRate Redundancy encoding functionality. Reuse all parameters but encode
 residual with lower bitrate */
void SKP_Silk_LBRR_encode_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  Pointer to Silk encod
er state           */
    SKP_Silk_encoder_control_FIX    *psEncCtrl,     /* I/O  Pointer to Silk encod
er control struct  */
    SKP_uint8                       *pCode,         /* O    Pointer to payload
                   */
    SKP_int16                       *pnBytesOut,    /* I/O  Pointer to number of
payload bytes      */
    SKP_int16                       xfw[]           /* I    Input signal
                   */
)
{
    SKP_int     i, TempGainsIndices[ NB_SUBFR ], frame_terminator;
    SKP_int     nBytes, nFramesInPayloadBuf;
    SKP_int32   TempGains_Q16[ NB_SUBFR ];
    SKP_int     typeOffset, LTP_scaleIndex, Rate_only_parameters = 0;
    /*****************************************/
    /* Control use of inband LBRR                */
    /*****************************************/
    SKP_Silk_LBRR_ctrl_FIX( psEnc, psEncCtrl );

    if( psEnc->sCmn.LBRR_enabled ) {
        /* Save original Gains */
        SKP_memcpy( TempGainsIndices, psEncCtrl->sCmn.GainsIndices, NB_SUBFR * si
zeof( SKP_int   ) );
        SKP_memcpy( TempGains_Q16,    psEncCtrl->Gains_Q16,    NB_SUBFR * sizeof(
 SKP_int32 ) );

        typeOffset     = psEnc->sCmn.typeOffsetPrev; // Temp save as cannot be ov
erwritten
        LTP_scaleIndex = psEncCtrl->sCmn.LTP_scaleIndex;

        /* Set max rate where quant signal is encoded */
        if( psEnc->sCmn.fs_kHz == 8 ) {
            Rate_only_parameters = 13500;
        } else if( psEnc->sCmn.fs_kHz == 12 ) {
            Rate_only_parameters = 15500;
        } else if( psEnc->sCmn.fs_kHz == 16 ) {
            Rate_only_parameters = 17500;
        } else if( psEnc->sCmn.fs_kHz == 24 ) {
            Rate_only_parameters = 19500;
        } else {
            SKP_assert( 0 );
        }

        if( psEnc->sCmn.Complexity > 0 && psEnc->sCmn.TargetRate_bps > Rate_only_
parameters ) {
            if( psEnc->sCmn.nFramesInPayloadBuf == 0 ) {
                /* First frame in packet copy Everything */
                SKP_memcpy( &psEnc->sNSQ_LBRR, &psEnc->sNSQ, sizeof( SKP_Silk_nsq
_state ) );

                psEnc->sCmn.LBRRprevLastGainIndex = psEnc->sShape.LastGainIndex;
                /* Increase Gains to get target LBRR rate */
                psEncCtrl->sCmn.GainsIndices[ 0 ] = psEncCtrl->sCmn.GainsIndices[
 0 ] + psEnc->sCmn.LBRR_GainIncreases;
                psEncCtrl->sCmn.GainsIndices[ 0 ] = SKP_LIMIT( psEncCtrl->sCmn.Ga
insIndices[ 0 ], 0, N_LEVELS_QGAIN - 1 );
```

```
            }
            /* Decode to get Gains in sync with decoder         */
            /* Overwrite unquantized gains with quantized gains */
            SKP_Silk_gains_dequant( psEncCtrl->Gains_Q16, psEncCtrl->sCmn.GainsIn
dices,
                &psEnc->sCmn.LBRRprevLastGainIndex, psEnc->sCmn.nFramesInPayloadB
uf );
            /*****************************************/
            /* Noise shaping quantization            */
            /*****************************************/
            psEnc->NoiseShapingQuantizer( &psEnc->sCmn, &psEncCtrl->sCmn,
                &psEnc->sNSQ_LBRR, xfw, &psEnc->sCmn.q_LBRR[ psEnc->sCmn.nFramesI
nPayloadBuf * psEnc->sCmn.frame_length ],
                psEncCtrl->sCmn.NLSFInterpCoef_Q2, psEncCtrl->PredCoef_Q12[ 0 ],
psEncCtrl->LTPCoef_Q14,
                psEncCtrl->AR2_Q13, psEncCtrl->HarmShapeGain_Q14, psEncCtrl->Tilt
_Q14, psEncCtrl->LF_shp_Q14,
                psEncCtrl->Gains_Q16, psEncCtrl->Lambda_Q10, psEncCtrl->LTP_scale
_Q14 );
        } else {
            SKP_memset( &psEnc->sCmn.q_LBRR[ psEnc->sCmn.nFramesInPayloadBuf *psE
nc->sCmn.frame_length ], 0,
                psEnc->sCmn.frame_length * sizeof( SKP_int ) );
            psEncCtrl->sCmn.LTP_scaleIndex = 0;
        }
        /*****************************************/
        /* Initialize arithmetic coder           */
        /*****************************************/
        if( psEnc->sCmn.nFramesInPayloadBuf == 0 ) {
            SKP_Silk_range_enc_init( &psEnc->sCmn.sRC_LBRR );
            psEnc->sCmn.nBytesInPayloadBuf = 0;
        }

        /*****************************************/
        /* Encode Parameters                     */
        /*****************************************/
        if( psEnc->sCmn.bitstream_v == BIT_STREAM_V4 ) {
            SKP_Silk_encode_parameters_v4( &psEnc->sCmn, &psEncCtrl->sCmn, &psEnc
->sCmn.sRC_LBRR );
        } else {
            SKP_Silk_encode_parameters( &psEnc->sCmn, &psEncCtrl->sCmn, &psEnc->s
Cmn.sRC_LBRR,
                &psEnc->sCmn.q_LBRR[ psEnc->sCmn.nFramesInPayloadBuf * psEnc->sCm
n.frame_length] );
        }

        if( psEnc->sCmn.sRC_LBRR.error ) {
            /* encoder returned error: clear payload buffer */
            nFramesInPayloadBuf = 0;
        } else {
            nFramesInPayloadBuf = psEnc->sCmn.nFramesInPayloadBuf + 1;
        }

        /*****************************************/
        /* finalize payload and copy to output  */
        /*****************************************/
        if( SKP_SMULBB( nFramesInPayloadBuf, FRAME_LENGTH_MS ) >= psEnc->sCmn.Pac
ketSize_ms ) {
```

```
            /* Check if FEC information should be added */
            frame_terminator = SKP_SILK_LAST_FRAME;

            /* Add the frame termination info to stream */
            SKP_Silk_range_encoder( &psEnc->sCmn.sRC_LBRR, frame_terminator, SKP_
Silk_FrameTermination_CDF );

            if( psEnc->sCmn.bitstream_v == BIT_STREAM_V4 ) {
                /*****************************************/
                /* Encode quantization indices of excitation */
                /*****************************************/
                for( i = 0; i < nFramesInPayloadBuf; i++ ) {
                    SKP_Silk_encode_pulses( &psEnc->sCmn.sRC_LBRR, psEnc->sCmn.si
gtype[ i ], psEnc->sCmn.QuantOffsetType[ i ],
                        &psEnc->sCmn.q_LBRR[ i * psEnc->sCmn.frame_length ], psEn
c->sCmn.frame_length );
                }
            }
            /* payload length so far */
            SKP_Silk_range_coder_get_length( &psEnc->sCmn.sRC_LBRR, &nBytes );

            /* check that there is enough space in external output buffer, and mo
ve data */
            if( *pnBytesOut >= nBytes ) {
                SKP_Silk_range_enc_wrap_up( &psEnc->sCmn.sRC_LBRR );
                SKP_memcpy( pCode, psEnc->sCmn.sRC_LBRR.buffer, nBytes * sizeof(
SKP_uint8 ) );

                *pnBytesOut = nBytes;
            } else {
                /* not enough space: payload will be discarded */
                *pnBytesOut = 0;
                SKP_assert( 0 );
            }
        } else {
            /* no payload for you this time */
            *pnBytesOut = 0;

            /* Encode that more frames follows */
            frame_terminator = SKP_SILK_MORE_FRAMES;
            SKP_Silk_range_encoder( &psEnc->sCmn.sRC_LBRR, frame_terminator, SKP_
Silk_FrameTermination_CDF );
        }

        /* Restore original Gains */
        SKP_memcpy( psEncCtrl->sCmn.GainsIndices, TempGainsIndices, NB_SUBFR * si
zeof( SKP_int   ) );
        SKP_memcpy( psEncCtrl->Gains_Q16,  TempGains_Q16,    NB_SUBFR * sizeof( S
KP_int32 ) );

        /* Restore LTP scale index and typeoffset */
        psEncCtrl->sCmn.LTP_scaleIndex = LTP_scaleIndex;
        psEnc->sCmn.typeOffsetPrev = typeOffset;
    }
}
```

A.36. src/SKP_Silk_encode_parameters.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

/*****************************************/
/* Encode parameters to create the payload */
/*****************************************/
void SKP_Silk_encode_parameters(
    SKP_Silk_encoder_state          *psEncC,        /* I/O  Encoder state
           */
    SKP_Silk_encoder_control        *psEncCtrlC,    /* I/O  Encoder control
           */
    SKP_Silk_range_coder_state      *psRC,          /* I/O  Range encoder state
           */
    const SKP_int                   *q              /* I    Quantization indices
           */
)
{
    SKP_int   i, k, typeOffset;
    const SKP_Silk_NLSF_CB_struct *psNLSF_CB;


    /************************/
    /* Encode sampling rate */
```

```
    /************************/
    /* only done for first frame in packet */
    if( psEncC->nFramesInPayloadBuf == 0 ) {

        /* Initialize arithmetic coder */
        SKP_Silk_range_enc_init( &psEncC->sRC );
        psEncC->nBytesInPayloadBuf = 0;

        /* get sampling rate index */
        for( i = 0; i < 3; i++ ) {
            if( SKP_Silk_SamplingRates_table[ i ] == psEncC->fs_kHz ) {
                break;
            }
        }
        SKP_Silk_range_encoder( psRC, i, SKP_Silk_SamplingRates_CDF );
    }

    /****************************************/
    /* Encode signal type and quantizer offset */
    /****************************************/
    typeOffset = 2 * psEncCtrlC->sigtype + psEncCtrlC->QuantOffsetType;
    if( psEncC->nFramesInPayloadBuf == 0 ) {
        /* first frame in packet: independent coding */
        SKP_Silk_range_encoder( psRC, typeOffset, SKP_Silk_type_offset_CDF );
    } else {
        /* condidtional coding */
        SKP_Silk_range_encoder( psRC, typeOffset, SKP_Silk_type_offset_joint_CDF[
 psEncC->typeOffsetPrev ] );
    }
    psEncC->typeOffsetPrev = typeOffset;

    /****************/
    /* Encode gains */
    /****************/
    /* first subframe */
    if( psEncC->nFramesInPayloadBuf == 0 ) {
        /* first frame in packet: independent coding */
        SKP_Silk_range_encoder( psRC, psEncCtrlC->GainsIndices[ 0 ], SKP_Silk_gai
n_CDF[ psEncCtrlC->sigtype ] );
    } else {
        /* condidtional coding */
        SKP_Silk_range_encoder( psRC, psEncCtrlC->GainsIndices[ 0 ], SKP_Silk_del
ta_gain_CDF );
    }

    /* remaining subframes */
    for( i = 1; i < NB_SUBFR; i++ ) {
        SKP_Silk_range_encoder( psRC, psEncCtrlC->GainsIndices[ i ], SKP_Silk_del
ta_gain_CDF );
    }
```

```
    /****************/
    /* Encode NLSFs */
    /****************/
    /* Range encoding of the NLSF path */
    psNLSF_CB = psEncC->psNLSF_CB[ psEncCtrlC->sigtype ];
    SKP_Silk_range_encoder_multi( psRC, psEncCtrlC->NLSFIndices, psNLSF_CB->Start
Ptr, psNLSF_CB->nStages );

    /* Encode NLSF interpolation factor */
    SKP_assert( psEncC->useInterpolatedNLSFs == 1 || psEncCtrlC->NLSFInterpCoef_Q
2 == ( 1 << 2 ) );
    SKP_Silk_range_encoder( psRC, psEncCtrlC->NLSFInterpCoef_Q2, SKP_Silk_NLSF_in
terpolation_factor_CDF );


    if( psEncCtrlC->sigtype == SIG_TYPE_VOICED ) {
        /**********************/
        /* Encode pitch lags */
        /**********************/


        /* lag index */
        if( psEncC->fs_kHz == 8 ) {
            SKP_Silk_range_encoder( psRC, psEncCtrlC->lagIndex, SKP_Silk_pitch_la
g_NB_CDF );
        } else if( psEncC->fs_kHz == 12 ) {
            SKP_Silk_range_encoder( psRC, psEncCtrlC->lagIndex, SKP_Silk_pitch_la
g_MB_CDF );
        } else if( psEncC->fs_kHz == 16 ) {
            SKP_Silk_range_encoder( psRC, psEncCtrlC->lagIndex, SKP_Silk_pitch_la
g_WB_CDF );
        } else {
            SKP_Silk_range_encoder( psRC, psEncCtrlC->lagIndex, SKP_Silk_pitch_la
g_SWB_CDF );
        }


        /* countour index */
        if( psEncC->fs_kHz == 8 ) {
            /* Less codevectors used in 8 khz mode */
            SKP_Silk_range_encoder( psRC, psEncCtrlC->contourIndex, SKP_Silk_pitc
h_contour_NB_CDF );
        } else {
            /* Joint for 12, 16, 24 khz */
            SKP_Silk_range_encoder( psRC, psEncCtrlC->contourIndex, SKP_Silk_pitc
h_contour_CDF );
        }

        /**********************/
        /* Encode LTP gains */
        /**********************/

        /* PERIndex value */
        SKP_Silk_range_encoder( psRC, psEncCtrlC->PERIndex, SKP_Silk_LTP_per_inde
x_CDF );

        /* Codebook Indices */
        for( k = 0; k < NB_SUBFR; k++ ) {
```

```
            SKP_Silk_range_encoder( psRC, psEncCtrlC->LTPIndex[ k ], SKP_Silk_LTP
_gain_CDF_ptrs[ psEncCtrlC->PERIndex ] );
        }

        /**********************/
        /* Encode LTP scaling */
        /**********************/
        SKP_Silk_range_encoder( psRC, psEncCtrlC->LTP_scaleIndex, SKP_Silk_LTPsca
le_CDF );
    }


    /***************/
    /* Encode seed */
    /***************/
    SKP_Silk_range_encoder( psRC, psEncCtrlC->Seed, SKP_Silk_Seed_CDF );

    /****************************************/
    /* Encode quantization indices of excitation */
    /****************************************/
    SKP_Silk_encode_pulses( psRC, psEncCtrlC->sigtype, psEncCtrlC->QuantOffsetTyp
e, q, psEncC->frame_length );


    /****************************************/
    /* Encode VAD flag                      */
    /****************************************/
    SKP_Silk_range_encoder( psRC, psEncC->vadFlag, SKP_Silk_vadflag_CDF );
}
```

A.37. src/SKP_Silk_encode_parameters_v4.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
```

```
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

/*****************************************/
/* Encode parameters to create the payload */
/*****************************************/
void SKP_Silk_encode_parameters_v4(
    SKP_Silk_encoder_state          *psEncC,        /* I/O  Encoder state
           */
    SKP_Silk_encoder_control        *psEncCtrlC,    /* I/O  Encoder control
           */
    SKP_Silk_range_coder_state      *psRC           /* I/O  Range encoder state
           */
)
{
    SKP_int   i, k, typeOffset;
    SKP_int   encode_absolute_lagIndex, delta_lagIndex;
    const SKP_Silk_NLSF_CB_struct *psNLSF_CB;


    /*************************/
    /* Encode sampling rate */
    /*************************/
    /* only done for first frame in packet */
    if( psEncC->nFramesInPayloadBuf == 0 ) {

        /* Initialize arithmetic coder */
        SKP_Silk_range_enc_init( &psEncC->sRC );
        psEncC->nBytesInPayloadBuf = 0;

        /* get sampling rate index */
        for( i = 0; i < 3; i++ ) {
            if( SKP_Silk_SamplingRates_table[ i ] == psEncC->fs_kHz ) {
                break;
            }
        }
        SKP_Silk_range_encoder( psRC, i, SKP_Silk_SamplingRates_CDF );
    }

    /*****************************************/
    /* Encode VAD flag                       */
    /*****************************************/
```

```
    SKP_Silk_range_encoder( psRC, psEncC->vadFlag, SKP_Silk_vadflag_CDF );

    /*****************************************/
    /* Encode signal type and quantizer offset */
    /*****************************************/
    typeOffset = 2 * psEncCtrlC->sigtype + psEncCtrlC->QuantOffsetType;
    if( psEncC->nFramesInPayloadBuf == 0 ) {
        /* first frame in packet: independent coding */
        SKP_Silk_range_encoder( psRC, typeOffset, SKP_Silk_type_offset_CDF );
    } else {
        /* condidtional coding */
        SKP_Silk_range_encoder( psRC, typeOffset, SKP_Silk_type_offset_joint_CDF[
 psEncC->typeOffsetPrev ] );
    }
    psEncC->typeOffsetPrev = typeOffset;

    /****************/
    /* Encode gains */
    /****************/
    /* first subframe */
    if( psEncC->nFramesInPayloadBuf == 0 ) {
        /* first frame in packet: independent coding */
        SKP_Silk_range_encoder( psRC, psEncCtrlC->GainsIndices[ 0 ], SKP_Silk_gai
n_CDF[ psEncCtrlC->sigtype ] );
    } else {
        /* condidtional coding */
        SKP_Silk_range_encoder( psRC, psEncCtrlC->GainsIndices[ 0 ], SKP_Silk_del
ta_gain_CDF );
    }

    /* remaining subframes */
    for( i = 1; i < NB_SUBFR; i++ ) {
        SKP_Silk_range_encoder( psRC, psEncCtrlC->GainsIndices[ i ], SKP_Silk_del
ta_gain_CDF );
    }


    /****************/
    /* Encode NLSFs */
    /****************/
    /* Range encoding of the NLSF path */
    psNLSF_CB = psEncC->psNLSF_CB[ psEncCtrlC->sigtype ];
    SKP_Silk_range_encoder_multi( psRC, psEncCtrlC->NLSFIndices, psNLSF_CB->Start
Ptr, psNLSF_CB->nStages );

    /* Encode NLSF interpolation factor */
    SKP_assert( psEncC->useInterpolatedNLSFs == 1 || psEncCtrlC->NLSFInterpCoef_Q
2 == ( 1 << 2 ) );
    SKP_Silk_range_encoder( psRC, psEncCtrlC->NLSFInterpCoef_Q2, SKP_Silk_NLSF_in
terpolation_factor_CDF );


    if( psEncCtrlC->sigtype == SIG_TYPE_VOICED ) {
        /*********************/
        /* Encode pitch lags */
```

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


        /* lag index */
        encode_absolute_lagIndex = 1;
        if( psEncC->nFramesInPayloadBuf > 0 && psEncC->prev_sigtype == SIG_TYPE_V
OICED ) {
            /* Delta Encoding */
            delta_lagIndex = psEncCtrlC->lagIndex - psEncC->prev_lagIndex;
            if( delta_lagIndex > MAX_DELTA_LAG ) {
                delta_lagIndex = ( MAX_DELTA_LAG << 1 ) + 1;
            } else if ( delta_lagIndex < -MAX_DELTA_LAG ) {
                delta_lagIndex = ( MAX_DELTA_LAG << 1 ) + 1;
            } else {
                delta_lagIndex = delta_lagIndex + MAX_DELTA_LAG;
                encode_absolute_lagIndex = 0; /* Only use delta */
            }
            SKP_Silk_range_encoder( psRC, delta_lagIndex, SKP_Silk_pitch_delta_CD
F );
        }
        if( encode_absolute_lagIndex ) {
            /* Absolute encoding */
            if( psEncC->fs_kHz == 8 ) {
                SKP_Silk_range_encoder( psRC, psEncCtrlC->lagIndex, SKP_Silk_pitc
h_lag_NB_CDF );
            } else if( psEncC->fs_kHz == 12 ) {
                SKP_Silk_range_encoder( psRC, psEncCtrlC->lagIndex, SKP_Silk_pitc
h_lag_MB_CDF );
            } else if( psEncC->fs_kHz == 16 ) {
                SKP_Silk_range_encoder( psRC, psEncCtrlC->lagIndex, SKP_Silk_pitc
h_lag_WB_CDF );
            } else {
                SKP_Silk_range_encoder( psRC, psEncCtrlC->lagIndex, SKP_Silk_pitc
h_lag_SWB_CDF );
            }
        }
        psEncC->prev_lagIndex = psEncCtrlC->lagIndex;


        /* countour index */
        if( psEncC->fs_kHz == 8 ) {
            /* Less codevectors used in 8 khz mode */
            SKP_Silk_range_encoder( psRC, psEncCtrlC->contourIndex, SKP_Silk_pitc
h_contour_NB_CDF );
        } else {
            /* Joint for 12, 16, 24 khz */
            SKP_Silk_range_encoder( psRC, psEncCtrlC->contourIndex, SKP_Silk_pitc
h_contour_CDF );
        }

        /********************/
        /* Encode LTP gains */
        /********************/

        /* PERIndex value */
        SKP_Silk_range_encoder( psRC, psEncCtrlC->PERIndex, SKP_Silk_LTP_per_inde
x_CDF );
```

```
        /* Codebook Indices */
        for( k = 0; k < NB_SUBFR; k++ ) {
            SKP_Silk_range_encoder( psRC, psEncCtrlC->LTPIndex[ k ], SKP_Silk_LTP
_gain_CDF_ptrs[ psEncCtrlC->PERIndex ] );
        }

        /**********************/
        /* Encode LTP scaling */
        /**********************/
        SKP_Silk_range_encoder( psRC, psEncCtrlC->LTP_scaleIndex, SKP_Silk_LTPsca
le_CDF );
    }


    /***************/
    /* Encode seed */
    /***************/
    SKP_Silk_range_encoder( psRC, psEncCtrlC->Seed, SKP_Silk_Seed_CDF );
}
```

A.38. src/SKP_Silk_encode_pulses.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/
```

```
#include "SKP_Silk_main.h"

/*****************************************/
/* Encode quantization indices of excitation */
/*****************************************/

SKP_INLINE SKP_int combine_and_check(       /* return ok */
    SKP_int         *pulses_comb,           /* O */
    const SKP_int   *pulses_in,             /* I */
    SKP_int         max_pulses,             /* I    max value for sum of pulses *
/
    SKP_int         len                     /* I    number of output values */
)
{
    SKP_int k, sum;

    for( k = 0; k < len; k++ ) {
        sum = pulses_in[ 2 * k ] + pulses_in[ 2 * k + 1 ];
        if( sum > max_pulses ) {
            return 1;
        }
        pulses_comb[ k ] = sum;
    }

    return 0;
}

/* Encode quantization indices of excitation */
void SKP_Silk_encode_pulses(
    SKP_Silk_range_coder_state      *psRC,          /* I/O  Range coder state
           */
    const SKP_int                   sigtype,        /* I    Sigtype
           */
    const SKP_int                   QuantOffsetType,/* I    QuantOffsetType
           */
    const SKP_int                   q[],            /* I    quantization indices
           */
    const SKP_int                   frame_length    /* I    Frame length
           */
)
{
    SKP_int   i, k, j, iter, bit, nLS, scale_down, RateLevelIndex = 0;
    SKP_int32 abs_q, minSumBits_Q6, sumBits_Q6;
    SKP_int   abs_pulses[ MAX_FRAME_LENGTH ];
    SKP_int   sum_pulses[ MAX_NB_SHELL_BLOCKS ];
    SKP_int   nRshifts[   MAX_NB_SHELL_BLOCKS ];
    SKP_int   pulses_comb[ 8 ];
    SKP_int   *abs_pulses_ptr;
    const SKP_int *pulses_ptr;
    const SKP_uint16 *cdf_ptr;
    const SKP_int16 *nBits_ptr;

    SKP_memset( pulses_comb, 0, 8 * sizeof( SKP_int ) ); // Fixing Valgrind repor
ted problem
```

```
    /****************************/
    /* Prepare for shell coding */
    /****************************/
    /* Calculate number of shell blocks */
    iter = frame_length / SHELL_CODEC_FRAME_LENGTH;

    /* Take the absolute value of the pulses */
    for( i = 0; i < frame_length; i+=4 ) {
        abs_pulses[i+0] = ( SKP_int )SKP_abs( q[ i + 0 ] );
        abs_pulses[i+1] = ( SKP_int )SKP_abs( q[ i + 1 ] );
        abs_pulses[i+2] = ( SKP_int )SKP_abs( q[ i + 2 ] );
        abs_pulses[i+3] = ( SKP_int )SKP_abs( q[ i + 3 ] );
    }

    /* Calc sum pulses per shell code frame */
    abs_pulses_ptr = abs_pulses;
    for( i = 0; i < iter; i++ ) {
        nRshifts[ i ] = 0;

        while( 1 ) {
            /* 1+1 -> 2 */
            scale_down = combine_and_check( pulses_comb, abs_pulses_ptr, SKP_Silk
_max_pulses_table[ 0 ], 8 );

            /* 2+2 -> 4 */
            scale_down += combine_and_check( pulses_comb, pulses_comb, SKP_Silk_m
ax_pulses_table[ 1 ], 4 );

            /* 4+4 -> 8 */
            scale_down += combine_and_check( pulses_comb, pulses_comb, SKP_Silk_m
ax_pulses_table[ 2 ], 2 );

            /* 8+8 -> 16 */
            sum_pulses[ i ] = pulses_comb[ 0 ] + pulses_comb[ 1 ];
            if( sum_pulses[ i ] > SKP_Silk_max_pulses_table[ 3 ] ) {
                scale_down++;
            }

            if( scale_down ) {
                /* We need to down scale the quantization signal */
                nRshifts[ i ]++;
                for( k = 0; k < SHELL_CODEC_FRAME_LENGTH; k++ ) {
                    abs_pulses_ptr[ k ] = SKP_RSHIFT( abs_pulses_ptr[ k ], 1 );
                }
            } else {
                /* Jump out of while(1) loop and go to next shell coding frame */
                break;
            }
        }
        abs_pulses_ptr += SHELL_CODEC_FRAME_LENGTH;
    }
```

```
    /**************/
    /* Rate level */
    /**************/
    /* find rate level that leads to fewest bits for coding of pulses per block i
nfo */
    minSumBits_Q6 = SKP_int32_MAX;
    for( k = 0; k < N_RATE_LEVELS - 1; k++ ) {
        nBits_ptr  = SKP_Silk_pulses_per_block_BITS_Q6[ k ];
        sumBits_Q6 = SKP_Silk_rate_levels_BITS_Q6[sigtype][ k ];
        for( i = 0; i < iter; i++ ) {
            if( nRshifts[ i ] > 0 ) {
                sumBits_Q6 += nBits_ptr[ MAX_PULSES + 1 ];
            } else {
                sumBits_Q6 += nBits_ptr[ sum_pulses[ i ] ];
            }
        }
        if( sumBits_Q6 < minSumBits_Q6 ) {
            minSumBits_Q6 = sumBits_Q6;
            RateLevelIndex = k;
        }
    }
    SKP_Silk_range_encoder( psRC, RateLevelIndex, SKP_Silk_rate_levels_CDF[ sigty
pe ] );

    /***************************************************/
    /* Sum-Weighted-Pulses Encoding                    */
    /***************************************************/
    cdf_ptr = SKP_Silk_pulses_per_block_CDF[ RateLevelIndex ];
    for( i = 0; i < iter; i++ ) {
        if( nRshifts[ i ] == 0 ) {
            SKP_Silk_range_encoder( psRC, sum_pulses[ i ], cdf_ptr );
        } else {
            SKP_Silk_range_encoder( psRC, MAX_PULSES + 1, cdf_ptr );
            for( k = 0; k < nRshifts[ i ] - 1; k++ ) {
                SKP_Silk_range_encoder( psRC, MAX_PULSES + 1, SKP_Silk_pulses_per
_block_CDF[ N_RATE_LEVELS - 1 ] );
            }
            SKP_Silk_range_encoder( psRC, sum_pulses[ i ], SKP_Silk_pulses_per_bl
ock_CDF[ N_RATE_LEVELS - 1 ] );
        }
    }

    /******************/
    /* Shell Encoding */
    /******************/
    for( i = 0; i < iter; i++ ) {
        if( sum_pulses[ i ] > 0 ) {
            SKP_Silk_shell_encoder( psRC, &abs_pulses[ i * SHELL_CODEC_FRAME_LENG
TH ] );
        }
    }

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

```
    /* LSB Encoding */
    /****************/
    for( i = 0; i < iter; i++ ) {
        if( nRshifts[ i ] > 0 ) {
            pulses_ptr = &q[ i * SHELL_CODEC_FRAME_LENGTH ];
            nLS = nRshifts[ i ] - 1;
            for( k = 0; k < SHELL_CODEC_FRAME_LENGTH; k++ ) {
                abs_q = SKP_abs( pulses_ptr[ k ] );
                for( j = nLS; j > 0; j-- ) {
                    bit = SKP_RSHIFT( abs_q, j ) & 1;
                    SKP_Silk_range_encoder( psRC, bit, SKP_Silk_lsb_CDF );
                }
                bit = abs_q & 1;
                SKP_Silk_range_encoder( psRC, bit, SKP_Silk_lsb_CDF );
            }
        }
    }

    /****************/
    /* Encode signs */
    /****************/
    SKP_Silk_encode_signs( psRC, q, frame_length, sigtype, QuantOffsetType, RateL
evelIndex );
}
```

A.39. src/SKP_Silk_find_LPC_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
```

```
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/* Finds LPC vector from correlations, and converts to NLSF */
void SKP_Silk_find_LPC_FIX(
    SKP_int             NLSF_Q15[],             /* O    NLSFs
                                                   */
    SKP_int             *interpIndex,           /* O    NLSF interpolation index,
 only used for NLSF interpolation                  */
    const SKP_int       prev_NLSFq_Q15[],       /* I    previous NLSFs, only used
 for NLSF interpolation                            */
    const SKP_int       useInterpolatedNLSFs,   /* I    Flag
                                                   */
    const SKP_int       LPC_order,              /* I    LPC order
                                                   */
    const SKP_int16     x[],                    /* I    Input signal
                                                   */
    const SKP_int       subfr_length            /* I    Input signal subframe len
gth including preceeding samples                  */
)
{
    SKP_int     k;
    SKP_int32   a_Q16[ MAX_LPC_ORDER ];

    SKP_int     isInterpLower, shift;
    SKP_int16   S[ MAX_LPC_ORDER ];
    SKP_int32   res_nrg0, res_nrg1;
    SKP_int     rshift0, rshift1;

    /* Used only for LSF interpolation */
    SKP_int32   a_tmp_Q16[ MAX_LPC_ORDER ], res_nrg_interp, res_nrg, res_tmp_nrg,
 res_nrg_2nd;
    SKP_int     res_nrg_interp_Q, res_nrg_Q, res_tmp_nrg_Q, res_nrg_2nd_Q;
    SKP_int16   a_tmp_Q12[ MAX_LPC_ORDER ];
    SKP_int     NLSF0_Q15[ MAX_LPC_ORDER ];
    SKP_int16   LPC_res[ ( MAX_FRAME_LENGTH + NB_SUBFR * MAX_LPC_ORDER ) / 2 ];

    /* Default: no interpolation */
    *interpIndex = 4;

    /* Burg AR analysis for the full frame */
    SKP_Silk_burg_modified( &res_nrg, &res_nrg_Q, a_Q16, x, subfr_length, NB_SUBF
R, FIND_LPC_COND_FAC_Q32, LPC_order );

    if( useInterpolatedNLSFs == 1 ) {

        /* Optimal solution for last 10 ms */
        SKP_Silk_burg_modified( &res_tmp_nrg, &res_tmp_nrg_Q, a_tmp_Q16, x + ( NB
_SUBFR >> 1 ) * subfr_length,
            subfr_length, ( NB_SUBFR >> 1 ), FIND_LPC_COND_FAC_Q32, LPC_order );

        /* subtract residual energy here, as that's easier than adding it to the
   */
```

```
        /* residual energy of the first 10 ms in each iteration of the search bel
ow */
        shift = res_tmp_nrg_Q - res_nrg_Q;
        if( shift >= 0 ) {
            if( shift < 32 ) {
                res_nrg = res_nrg - SKP_RSHIFT( res_tmp_nrg, shift );
            }
        } else {
            SKP_assert( shift > -32 );
            res_nrg   = SKP_RSHIFT( res_nrg, -shift ) - res_tmp_nrg;
            res_nrg_Q = res_tmp_nrg_Q;
        }

        /* Convert to NLSFs */
        SKP_Silk_A2NLSF( NLSF_Q15, a_tmp_Q16, LPC_order );

        /* Search over interpolation indices to find the one with lowest residual
 energy */
        res_nrg_2nd = SKP_int32_MAX;
        for( k = 3; k >= 0; k-- ) {
            /* Interpolate NLSFs for first half */
            SKP_Silk_interpolate( NLSF0_Q15, prev_NLSFq_Q15, NLSF_Q15, k, LPC_ord
er );

            /* Convert to LPC for residual energy evaluation */
            SKP_Silk_NLSF2A_stable( a_tmp_Q12, NLSF0_Q15, LPC_order );

            /* Calculate residual energy with NLSF interpolation */
            SKP_memset( S, 0, LPC_order * sizeof( SKP_int16 ) );
            SKP_Silk_LPC_analysis_filter( x, a_tmp_Q12, S, LPC_res, 2 * subfr_len
gth, LPC_order );

            SKP_Silk_sum_sqr_shift( &res_nrg0, &rshift0, LPC_res + LPC_order,
            subfr_length - LPC_order );
            SKP_Silk_sum_sqr_shift( &res_nrg1, &rshift1, LPC_res + LPC_order + su
bfr_length, subfr_length - LPC_order );

            /* Add subframe energies from first half frame */
            shift = rshift0 - rshift1;
            if( shift >= 0 ) {
                res_nrg1         = SKP_RSHIFT( res_nrg1, shift );
                res_nrg_interp_Q = -rshift0;
            } else {
                res_nrg0         = SKP_RSHIFT( res_nrg0, -shift );
                res_nrg_interp_Q = -rshift1;
            }
            res_nrg_interp = SKP_ADD32( res_nrg0, res_nrg1 );

            /* Compare with first half energy without NLSF interpolation, or best
 interpolated value so far */
            shift = res_nrg_interp_Q - res_nrg_Q;
            if( shift >= 0 ) {
                if( SKP_RSHIFT( res_nrg_interp, shift ) < res_nrg ) {
                    isInterpLower = SKP_TRUE;
                } else {
```

```
                    isInterpLower = SKP_FALSE;
                }
            } else {
                if( -shift < 32 ) {
                    if( res_nrg_interp < SKP_RSHIFT( res_nrg, -shift ) ) {
                        isInterpLower = SKP_TRUE;
                    } else {
                        isInterpLower = SKP_FALSE;
                    }
                } else {
                    isInterpLower = SKP_FALSE;
                }
            }

            /* Determine whether current interpolated NLSFs are best so far */
            if( isInterpLower == SKP_TRUE ) {
                /* Interpolation has lower residual energy */
                res_nrg   = res_nrg_interp;
                res_nrg_Q = res_nrg_interp_Q;
                *interpIndex = k;
            }
            res_nrg_2nd   = res_nrg_interp;
            res_nrg_2nd_Q = res_nrg_interp_Q;
        }
    }

    if( *interpIndex == 4 ) {
        /* NLSF interpolation is currently inactive, calculate NLSFs from full fr
ame AR coefficients */
        SKP_Silk_A2NLSF( NLSF_Q15, a_Q16, LPC_order );
    }
}
```

A.40. src/SKP_Silk_find_LTP_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
```

```
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

void SKP_Silk_fit_LTP(
    SKP_int32 LTP_coefs_Q16[ LTP_ORDER ],
    SKP_int16 LTP_coefs_Q14[ LTP_ORDER ]
);

void SKP_Silk_find_LTP_FIX(
    SKP_int16           b_Q14[ NB_SUBFR * LTP_ORDER ],              /* O    LTP c
oefs                                                 */
    SKP_int32           WLTP[ NB_SUBFR * LTP_ORDER * LTP_ORDER ],   /* O    Weigh
t for LTP quantization                               */
    SKP_int             *LTPredCodGain_Q7,                          /* O    LTP c
oding gain                                           */
    const SKP_int16     r_first[],                                  /* I    resid
ual signal after LPC signal + state for first 10 ms  */
    const SKP_int16     r_last[],                                   /* I    resid
ual signal after LPC signal + state for last 10 ms   */
    const SKP_int       lag[ NB_SUBFR ],                            /* I    LTP l
ags                                                  */
    const SKP_int32     Wght_Q15[ NB_SUBFR ],                       /* I    weigh
ts                                                   */
    const SKP_int       subfr_length,                               /* I    subfr
ame length                                           */
    const SKP_int       mem_offset,                                 /* I    numbe
r of samples in LTP memory                           */
    SKP_int             corr_rshifts[ NB_SUBFR ]                    /* O    right
 shifts applied to correlations                      */
)
{
    SKP_int   i, k, lshift;
    const SKP_int16 *r_ptr, *lag_ptr;
    SKP_int16 *b_Q14_ptr;

    SKP_int32 regu;
    SKP_int32 *WLTP_ptr;
    SKP_int32 b_Q16[ LTP_ORDER ], delta_b_Q14[ LTP_ORDER ], d_Q14[ NB_SUBFR ], nr
g[ NB_SUBFR ], g_Q26;
    SKP_int32 w[ NB_SUBFR ], WLTP_max, max_abs_d_Q14, max_w_bits;

    SKP_int32 temp32, denom32;
    SKP_int   extra_shifts;
    SKP_int   rr_shifts, maxRshifts, maxRshifts_wxtra, LZs;
    SKP_int32 LPC_res_nrg, LPC_LTP_res_nrg, div_Q16;
```

```
    SKP_int32 Rr[ LTP_ORDER ], rr[ NB_SUBFR ];
    SKP_int32 wd, m_Q12;

    b_Q14_ptr = b_Q14;
    WLTP_ptr  = WLTP;
    r_ptr     = &r_first[ mem_offset ];
    for( k = 0; k < NB_SUBFR; k++ ) {
        if( k == ( NB_SUBFR >> 1 ) ) { /* shift residual for last 10 ms */
            r_ptr = &r_last[ mem_offset ];
        }
        lag_ptr = r_ptr - ( lag[ k ] + LTP_ORDER / 2 );

        SKP_Silk_sum_sqr_shift( &rr[ k ], &rr_shifts, r_ptr, subfr_length ); /* r
r[ k ] in Q( -rr_shifts ) */
        /* Assure headroom */
        LZs = SKP_Silk_CLZ32( rr[k] );
        if( LZs < LTP_CORRS_HEAD_ROOM ) {
            rr[ k ] = SKP_RSHIFT_ROUND( rr[ k ], LTP_CORRS_HEAD_ROOM - LZs );
            rr_shifts += (LTP_CORRS_HEAD_ROOM - LZs);
        }
        corr_rshifts[ k ] = rr_shifts;
        SKP_Silk_corrMatrix_FIX( lag_ptr, subfr_length, LTP_ORDER, WLTP_ptr, &cor
r_rshifts[ k ] );     /* WLTP_fix_ptr in Q( -corr_rshifts[ k ] ) */
        /* The correlation vector always have lower max abs value than rr and/or
RR so head room is assured */
        SKP_Silk_corrVector_FIX( lag_ptr, r_ptr, subfr_length, LTP_ORDER, Rr, cor
r_rshifts[ k ] ); /* Rr_fix_ptr   in Q( -corr_rshifts[ k ] ) */
        if( corr_rshifts[ k ] > rr_shifts ) {
            rr[ k ] = SKP_RSHIFT( rr[ k ], corr_rshifts[ k ] - rr_shifts ); /* rr
[ k ] in Q( -corr_rshifts[ k ] ) */
        }
        SKP_assert( rr[ k ] >= 0 );

        regu = SKP_SMULWB( rr[ k ] + 1, LTP_DAMPING_Q16 );
        SKP_Silk_regularize_correlations_FIX( WLTP_ptr, &rr[k], regu, LTP_ORDER )
;

        SKP_Silk_solve_LDL_FIX( WLTP_ptr, LTP_ORDER, Rr, b_Q16 ); /* WLTP_fix_ptr
 and Rr_fix_ptr both in Q(-corr_rshifts[k]) */

        /* Limit and store in Q14 */
        SKP_Silk_fit_LTP( b_Q16, b_Q14_ptr );

        /* Calculate residual energy */
        nrg[ k ] = SKP_Silk_residual_energy16_covar_FIX( b_Q14_ptr, WLTP_ptr, Rr,
 rr[ k ], LTP_ORDER, 14 ); /* nrg_fix in Q( -corr_rshifts[ k ] ) */

        /* temp = Wght[ k ] / ( nrg[ k ] * Wght[ k ] + 0.01f * subfr_length ); */
        extra_shifts = SKP_min_int( corr_rshifts[ k ], LTP_CORRS_HEAD_ROOM );
        denom32 = SKP_LSHIFT_SAT32( SKP_SMULWB( nrg[ k ], Wght_Q15[ k ] ), 1 + ex
tra_shifts ) +  /* Q( -corr_rshifts[ k ] + extra_shifts ) */
            SKP_RSHIFT( SKP_SMULWB( subfr_length, 655 ), corr_rshifts[ k ] - extr
a_shifts );     /* Q( -corr_rshifts[ k ] + extra_shifts ) */
        denom32 = SKP_max( denom32, 1 );
        SKP_assert( ((SKP_int64)Wght_Q15[ k ] << 16 ) < SKP_int32_MAX ); /* Wght
always < 0.5 in Q0 */
        temp32 = SKP_DIV32( SKP_LSHIFT( ( SKP_int32 )Wght_Q15[ k ], 16 ), denom32
 );  /* Q( 15 + 16 + corr_rshifts[k] - extra_shifts ) */
        temp32 = SKP_RSHIFT( temp32, 31 + corr_rshifts[ k ] - extra_shifts - 26 )
;  /* Q26 */
```

```
        /* Limit temp such that the below scaling never wraps around */
        WLTP_max = 0;
        for( i = 0; i < LTP_ORDER * LTP_ORDER; i++ ) {
            WLTP_max = SKP_max( WLTP_ptr[ i ], WLTP_max );
        }
        lshift = SKP_Silk_CLZ32( WLTP_max ) - 1 - 3; /* keep 3 bits free for vq_n
earest_neighbor_fix */
        SKP_assert( 26 - 18 + lshift >= 0 );
        if( 26 - 18 + lshift < 31 ) {
            temp32 = SKP_min_32( temp32, SKP_LSHIFT( ( SKP_int32 )1, 26 - 18 + ls
hift ) );
        }

        SKP_Silk_scale_vector32_Q26_lshift_18( WLTP_ptr, temp32, LTP_ORDER * LTP_
ORDER ); /* WLTP_ptr in Q( 18 - corr_rshifts[ k ] ) */

        w[ k ] = matrix_ptr( WLTP_ptr, ( LTP_ORDER >> 1 ), ( LTP_ORDER >> 1 ), LT
P_ORDER ); /* w in Q( 18 - corr_rshifts[ k ] ) */
        SKP_assert( w[k] >= 0 );

        r_ptr     += subfr_length;
        b_Q14_ptr += LTP_ORDER;
        WLTP_ptr  += LTP_ORDER * LTP_ORDER;
    }

    maxRshifts = 0;
    for( k = 0; k < NB_SUBFR; k++ ) {
        maxRshifts = SKP_max_int( corr_rshifts[ k ], maxRshifts );
    }

    /* compute LTP coding gain */
    if( LTPredCodGain_Q7 != NULL ) {
        LPC_LTP_res_nrg = 0;
        LPC_res_nrg     = 0;
        SKP_assert( LTP_CORRS_HEAD_ROOM >= 2 ); /* Check that no overflow will ha
ppen when adding */
        for( k = 0; k < NB_SUBFR; k++ ) {
            LPC_res_nrg     = SKP_ADD32( LPC_res_nrg,     SKP_RSHIFT( SKP_ADD32(
SKP_SMULWB(  rr[ k ], Wght_Q15[ k ] ), 1 ), 1 + ( maxRshifts - corr_rshifts[ k ]
) ) ); /*  Q( -maxRshifts ) */
            LPC_LTP_res_nrg = SKP_ADD32( LPC_LTP_res_nrg, SKP_RSHIFT( SKP_ADD32(
SKP_SMULWB( nrg[ k ], Wght_Q15[ k ] ), 1 ), 1 + ( maxRshifts - corr_rshifts[ k ]
) ) ); /*  Q( -maxRshifts ) */
        }
        LPC_LTP_res_nrg = SKP_max( LPC_LTP_res_nrg, 1 ); /* avoid division by zer
o */

        div_Q16 = SKP_DIV32_varQ( LPC_res_nrg, LPC_LTP_res_nrg, 16 );
        *LTPredCodGain_Q7 = ( SKP_int )SKP_SMULBB( 3, SKP_Silk_lin2log( div_Q16 )
 - ( 16 << 7 ) );

        SKP_assert( *LTPredCodGain_Q7 == ( SKP_int )SKP_SAT16( SKP_MUL( 3, SKP_Si
lk_lin2log( div_Q16 ) - ( 16 << 7 ) ) ) );
    }

    /* smoothing */
    /* d = sum( B, 1 ); */
    b_Q14_ptr = b_Q14;
    for( k = 0; k < NB_SUBFR; k++ ) {
        d_Q14[ k ] = 0;
```

```
        for( i = 0; i < LTP_ORDER; i++ ) {
            d_Q14[ k ] += b_Q14_ptr[ i ];
        }
        b_Q14_ptr += LTP_ORDER;
    }

    /* m = ( w * d' ) / ( sum( w ) + 1e-3 ); */

    /* Find maximum absolute value of d_Q14 and the bits used by w in Q0 */
    max_abs_d_Q14 = 0;
    max_w_bits    = 0;
    for( k = 0; k < NB_SUBFR; k++ ) {
        max_abs_d_Q14 = SKP_max_32( max_abs_d_Q14, SKP_abs( d_Q14[ k ] ) );
        /* w[ k ] is in Q( 18 - corr_rshifts[ k ] ) */
        /* Find bits needed in Q( 18 - maxRshifts ) */
        max_w_bits = SKP_max_32( max_w_bits, 32 - SKP_Silk_CLZ32( w[ k ] ) + corr
_rshifts[ k ] - maxRshifts );
    }

    /* max_abs_d_Q14 = (5 << 15); worst case, i.e. LTP_ORDER * -SKP_int16_MIN */
    SKP_assert( max_abs_d_Q14 <= ( 5 << 15 ) );

    /* How many bits is needed for w*d' in Q( 18 - maxRshifts ) in the worst case
, of all d_Q14's being equal to max_abs_d_Q14 */
    extra_shifts = max_w_bits + 32 - SKP_Silk_CLZ32( max_abs_d_Q14 ) - 14;

    /* Subtract what we got available; bits in output var plus maxRshifts */
    extra_shifts -= ( 32 - 1 - 2 + maxRshifts ); /* Keep sign bit free as well as
 2 bits for accumulation */
    extra_shifts = SKP_max_int( extra_shifts, 0 );

    maxRshifts_wxtra = maxRshifts + extra_shifts;

    temp32 = SKP_RSHIFT( 262, maxRshifts + extra_shifts ) + 1; /* 1e-3f in Q( 18
- (maxRshifts + extra_shifts) ) */
    wd = 0;
    for( k = 0; k < NB_SUBFR; k++ ) {
        /* w has at least 2 bits of headroom so no overflow should happen */
        temp32 = SKP_ADD32( temp32,                    SKP_RSHIFT( w[ k ], maxRs
hifts_wxtra - corr_rshifts[ k ] ) );                   /* Q( 18 - maxRshifts_wxt
ra ) */
        wd     = SKP_ADD32( wd, SKP_LSHIFT( SKP_SMULWW( SKP_RSHIFT( w[ k ], maxRs
hifts_wxtra - corr_rshifts[ k ] ), d_Q14[ k ] ), 2 ) ); /* Q( 18 - maxRshifts_wxt
ra ) */
    }
    m_Q12 = SKP_DIV32_varQ( wd, temp32, 12 );

    b_Q14_ptr = b_Q14;
    for( k = 0; k < NB_SUBFR; k++ ) {
        /* w_fix[ k ] from Q( 18 - corr_rshifts[ k ] ) to Q( 16 ) */
        if( 2 - corr_rshifts[k] > 0 ) {
            temp32 = SKP_RSHIFT( w[ k ], 2 - corr_rshifts[ k ] );
        } else {
            temp32 = SKP_LSHIFT_SAT32( w[ k ], corr_rshifts[ k ] - 2 );
        }
```

```
        g_Q26 = SKP_MUL(
            SKP_DIV32(
                LTP_SMOOTHING_Q26,
                SKP_RSHIFT( LTP_SMOOTHING_Q26, 10 ) + temp32 ),
                    /* Q10 */
            SKP_LSHIFT_SAT32( SKP_SUB_SAT32( ( SKP_int32 )m_Q12, SKP_RSHIFT( d_Q1
4[ k ], 2 ) ), 4 ) );  /* Q16 */

        temp32 = 0;
        for( i = 0; i < LTP_ORDER; i++ ) {
            delta_b_Q14[ i ] = SKP_max_16( b_Q14_ptr[ i ], 1638 );  /* 1638_Q14 =
 0.1_Q0 */
            temp32 += delta_b_Q14[ i ];                             /* Q14 */
        }
        temp32 = SKP_DIV32( g_Q26, temp32 ); /* Q14->Q12 */
        for( i = 0; i < LTP_ORDER; i++ ) {
            b_Q14_ptr[ i ] = SKP_LIMIT( ( SKP_int32 )b_Q14_ptr[ i ] + SKP_SMULWB(
 SKP_LSHIFT_SAT32( temp32, 4 ), delta_b_Q14[ i ] ), -16000, 28000 );
        }
        b_Q14_ptr += LTP_ORDER;
    }
}

void SKP_Silk_fit_LTP(
    SKP_int32 LTP_coefs_Q16[ LTP_ORDER ],
    SKP_int16 LTP_coefs_Q14[ LTP_ORDER ]
)
{
    SKP_int i;

    for( i = 0; i < LTP_ORDER; i++ ) {
        LTP_coefs_Q14[ i ] = ( SKP_int16 )SKP_SAT16( SKP_RSHIFT_ROUND( LTP_coefs_
Q16[ i ], 2 ) );
    }
}
```

A.41. src/SKP_Silk_find_pitch_lags_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
```

```
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/* Find pitch lags */
void SKP_Silk_find_pitch_lags_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  encoder state
                        */
    SKP_Silk_encoder_control_FIX    *psEncCtrl,     /* I/O  encoder control
                        */
    SKP_int16                       res[],          /* O    residual
                        */
    const SKP_int16                 x[]             /* I    Speech signal
                        */
)
{
    SKP_Silk_predict_state_FIX *psPredSt = &psEnc->sPred;
    SKP_int   buf_len, i;
    SKP_int32 scale;
    SKP_int32 thrhld_Q15;
    const SKP_int16 *x_buf, *x_buf_ptr;
    SKP_int16 Wsig[      FIND_PITCH_LPC_WIN_MAX ], *Wsig_ptr;
    SKP_int32 auto_corr[ FIND_PITCH_LPC_ORDER_MAX + 1 ];
    SKP_int16 rc_Q15[    FIND_PITCH_LPC_ORDER_MAX ];
    SKP_int32 A_Q24[     FIND_PITCH_LPC_ORDER_MAX ];
    SKP_int32 FiltState[ FIND_PITCH_LPC_ORDER_MAX ];
    SKP_int16 A_Q12[     FIND_PITCH_LPC_ORDER_MAX ];

    /*****************************************/
    /* Setup buffer lengths etc based of Fs  */
    /*****************************************/
    buf_len = SKP_ADD_LSHIFT( psEnc->sCmn.la_pitch, psEnc->sCmn.frame_length, 1 )
;

    /* Safty check */
    SKP_assert( buf_len >= psPredSt->pitch_LPC_win_length );

    x_buf = x - psEnc->sCmn.frame_length;

    /*************************************/
    /* Estimate LPC AR coeficients */
```

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

    /* Calculate windowed signal */

    /* First LA_LTP samples */
    x_buf_ptr = x_buf + buf_len - psPredSt->pitch_LPC_win_length;
    Wsig_ptr  = Wsig;
    SKP_Silk_apply_sine_window( Wsig_ptr, x_buf_ptr, 1, psEnc->sCmn.la_pitch );

    /* Middle un - windowed samples */
    Wsig_ptr  += psEnc->sCmn.la_pitch;
    x_buf_ptr += psEnc->sCmn.la_pitch;
    SKP_memcpy( Wsig_ptr, x_buf_ptr, ( psPredSt->pitch_LPC_win_length - SKP_LSHIF
T( psEnc->sCmn.la_pitch, 1 ) ) * sizeof( SKP_int16 ) );

    /* Last LA_LTP samples */
    Wsig_ptr  += psPredSt->pitch_LPC_win_length - SKP_LSHIFT( psEnc->sCmn.la_pitc
h, 1 );
    x_buf_ptr += psPredSt->pitch_LPC_win_length - SKP_LSHIFT( psEnc->sCmn.la_pitc
h, 1 );
    SKP_Silk_apply_sine_window( Wsig_ptr, x_buf_ptr, 2, psEnc->sCmn.la_pitch );

    /* Calculate autocorrelation sequence */
    SKP_Silk_autocorr( auto_corr, &scale, Wsig, psPredSt->pitch_LPC_win_length, p
sEnc->sCmn.pitchEstimationLPCOrder + 1 );

    /* add white noise, as fraction of energy */
    auto_corr[ 0 ] = SKP_SMLAWB( auto_corr[ 0 ], auto_corr[ 0 ], FIND_PITCH_WHITE
_NOISE_FRACTION_Q16 );

    /* calculate the reflection coefficients using schur */
    SKP_Silk_schur( rc_Q15, auto_corr, psEnc->sCmn.pitchEstimationLPCOrder );

    /* convert reflection coefficients to prediction coefficients */
    SKP_Silk_k2a( A_Q24, rc_Q15, psEnc->sCmn.pitchEstimationLPCOrder );

    /* Convert From 32 bit Q24 to 16 bit Q12 coefs */
    for( i = 0; i < psEnc->sCmn.pitchEstimationLPCOrder; i++ ) {
        A_Q12[ i ] = ( SKP_int16 )SKP_SAT16( SKP_RSHIFT( A_Q24[ i ], 12 ) );
    }

    /* Do BWE */
    SKP_Silk_bwexpander( A_Q12, psEnc->sCmn.pitchEstimationLPCOrder, FIND_PITCH_B
ANDWITH_EXPANSION_Q16 );

    /***************************************/
    /* LPC analysis filtering              */
    /***************************************/
    SKP_memset( FiltState, 0, psEnc->sCmn.pitchEstimationLPCOrder * sizeof( SKP_i
nt16 ) );
    SKP_Silk_MA_Prediction( x_buf, A_Q12, FiltState, res, buf_len, psEnc->sCmn.pi
tchEstimationLPCOrder );
    SKP_memset( res, 0, psEnc->sCmn.pitchEstimationLPCOrder * sizeof( SKP_int16 )
 );

    /* Threshold for pitch estimator */
    thrhld_Q15 = ( 1 << 14 ); // 0.5f in Q15
```

```
    thrhld_Q15 = SKP_SMLABB( thrhld_Q15, -131, psEnc->sCmn.pitchEstimationLPCOrde
r );
    thrhld_Q15 = SKP_SMLABB( thrhld_Q15,  -13, ( SKP_int16 )SKP_Silk_SQRT_APPROX(
 SKP_LSHIFT( ( SKP_int32 )psEnc->speech_activity_Q8, 8 ) ) );
    thrhld_Q15 = SKP_SMLABB( thrhld_Q15, 4587, psEnc->sCmn.prev_sigtype );
    thrhld_Q15 = SKP_MLA(    thrhld_Q15,  -31, SKP_RSHIFT( psEncCtrl->input_tilt_
Q15, 8 ) );
    thrhld_Q15 = SKP_SAT16(  thrhld_Q15 );

    /*****************************************/
    /* Call Pitch estimator */
    /*****************************************/
    psEncCtrl->sCmn.sigtype = SKP_Silk_pitch_analysis_core( res, psEncCtrl->sCmn.
pitchL, &psEncCtrl->sCmn.lagIndex,
        &psEncCtrl->sCmn.contourIndex, &psEnc->LTPCorr_Q15, psEnc->sCmn.prevLag,
psEnc->pitchEstimationThreshold_Q16,
        ( SKP_int16 )thrhld_Q15, psEnc->sCmn.fs_kHz, psEnc->sCmn.pitchEstimationC
omplexity );
}
```

A.42. src/SKP_Silk_find_pred_coefs_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"
```

```
void SKP_Silk_find_pred_coefs_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  encoder state
                    */
    SKP_Silk_encoder_control_FIX    *psEncCtrl,     /* I/O  encoder control
                    */
    const SKP_int16                 res_pitch[]     /* I    Residual from pitch a
nalysis                */
)
{
    SKP_int         i;
    SKP_int32       WLTP[ NB_SUBFR * LTP_ORDER * LTP_ORDER ];
    SKP_int32       invGains_Q16[ NB_SUBFR ], local_gains[ NB_SUBFR ], Wght_Q15[
NB_SUBFR ];
    SKP_int         NLSF_Q15[ MAX_LPC_ORDER ];
    const SKP_int16 *x_ptr;
    SKP_int16       *x_pre_ptr, LPC_in_pre[ NB_SUBFR * MAX_LPC_ORDER + MAX_FRAME_
LENGTH ];
    SKP_int32       tmp, min_gain_Q16;
    SKP_int         LTP_corrs_rshift[ NB_SUBFR ];


    /* weighting for weighted least squares */
    min_gain_Q16 = SKP_int32_MAX >> 6;
    for( i = 0; i < NB_SUBFR; i++ ) {
        min_gain_Q16 = SKP_min( min_gain_Q16, psEncCtrl->Gains_Q16[ i ] );
    }
    for( i = 0; i < NB_SUBFR; i++ ) {
        /* Divide to Q16 */
        SKP_assert( psEncCtrl->Gains_Q16[ i ] > 0 );
        /* Invert and normalize gains, and ensure that maximum invGains_Q16 is wi
thin range of a 16 bit int */
        invGains_Q16[ i ] = SKP_DIV32_varQ( min_gain_Q16, psEncCtrl->Gains_Q16[ i
 ], 16 - 2 );

        /* Ensure Wght_Q15 a minimum value 1 */
        invGains_Q16[ i ] = SKP_max( invGains_Q16[ i ], 363 );

        /* Square the inverted gains */
        SKP_assert( invGains_Q16[ i ] == SKP_SAT16( invGains_Q16[ i ] ) );
        tmp = SKP_SMULWB( invGains_Q16[ i ], invGains_Q16[ i ] );
        Wght_Q15[ i ] = SKP_RSHIFT( tmp, 1 );

        /* Invert the inverted and normalized gains */
        local_gains[ i ] = SKP_DIV32( ( 1 << 16 ), invGains_Q16[ i ] );
    }

    if( psEncCtrl->sCmn.sigtype == SIG_TYPE_VOICED ) {
        /**********/
        /* VOICED */
        /**********/
        SKP_assert( psEnc->sCmn.frame_length - psEnc->sCmn.predictLPCOrder >= psE
ncCtrl->sCmn.pitchL[ 0 ] + LTP_ORDER / 2 );

        /* LTP analysis */
        SKP_Silk_find_LTP_FIX( psEncCtrl->LTPCoef_Q14, WLTP, &psEncCtrl->LTPredCo
dGain_Q7, res_pitch,
            res_pitch + SKP_RSHIFT( psEnc->sCmn.frame_length, 1 ), psEncCtrl->sCm
n.pitchL, Wght_Q15,
```

```
            psEnc->sCmn.subfr_length, psEnc->sCmn.frame_length, LTP_corrs_rshift
);


        /* Quantize LTP gain parameters */
        SKP_Silk_quant_LTP_gains_FIX( psEncCtrl->LTPCoef_Q14, psEncCtrl->sCmn.LTP
Index, &psEncCtrl->sCmn.PERIndex,
            WLTP, psEnc->mu_LTP_Q8, psEnc->sCmn.LTPQuantLowComplexity );

        /* Control LTP scaling */
        SKP_Silk_LTP_scale_ctrl_FIX( psEnc, psEncCtrl );

        /* Create LTP residual */
        SKP_Silk_LTP_analysis_filter_FIX( LPC_in_pre, psEnc->x_buf + psEnc->sCmn.
frame_length - psEnc->sCmn.predictLPCOrder,
            psEncCtrl->LTPCoef_Q14, psEncCtrl->sCmn.pitchL, invGains_Q16, 16, psE
nc->sCmn.subfr_length, psEnc->sCmn.predictLPCOrder );

    } else {
        /************/
        /* UNVOICED */
        /************/
        /* Create signal with prepended subframes, scaled by inverse gains */
        x_ptr     = psEnc->x_buf + psEnc->sCmn.frame_length - psEnc->sCmn.predict
LPCOrder;
        x_pre_ptr = LPC_in_pre;
        for( i = 0; i < NB_SUBFR; i++ ) {
            SKP_Silk_scale_copy_vector16( x_pre_ptr, x_ptr, invGains_Q16[ i ],
                psEnc->sCmn.subfr_length + psEnc->sCmn.predictLPCOrder );
            x_pre_ptr += psEnc->sCmn.subfr_length + psEnc->sCmn.predictLPCOrder;
            x_ptr     += psEnc->sCmn.subfr_length;
        }

        SKP_memset( psEncCtrl->LTPCoef_Q14, 0, NB_SUBFR * LTP_ORDER * sizeof( SKP
_int16 ) );
        psEncCtrl->LTPredCodGain_Q7 = 0;
    }

    /* LPC_in_pre contains the LTP-filtered input for voiced, and the unfiltered
input for unvoiced */
    TIC(FIND_LPC)
    SKP_Silk_find_LPC_FIX( NLSF_Q15, &psEncCtrl->sCmn.NLSFInterpCoef_Q2, psEnc->s
Pred.prev_NLSFq_Q15,
        psEnc->sCmn.useInterpolatedNLSFs * ( 1 - psEnc->sCmn.first_frame_after_re
set ), psEnc->sCmn.predictLPCOrder,
        LPC_in_pre, psEnc->sCmn.subfr_length + psEnc->sCmn.predictLPCOrder );
    TOC(FIND_LPC)


    /* Quantize LSFs */
    TIC(PROCESS_LSFS)
        SKP_Silk_process_NLSFs_FIX( psEnc, psEncCtrl, NLSF_Q15 );
    TOC(PROCESS_LSFS)

    /* Calculate residual energy using quantized LPC coefficients */
    SKP_Silk_residual_energy_FIX( psEncCtrl->ResNrg, psEncCtrl->ResNrgQ, LPC_in_p
re, psEncCtrl->PredCoef_Q12, local_gains,
        psEnc->sCmn.subfr_length, psEnc->sCmn.predictLPCOrder );
```

```
    /* Copy to prediction struct for use in next frame for fluctuation reduction
*/
    SKP_memcpy( psEnc->sPred.prev_NLSFq_Q15, NLSF_Q15, psEnc->sCmn.predictLPCOrde
r * sizeof( SKP_int ) );

}
```

A.43. src/SKP_Silk_gain_quant.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

#define OFFSET          ( ( MIN_QGAIN_DB * 128 ) / 6 + 16 * 128 )
#define SCALE_Q16       ( ( 65536 * ( N_LEVELS_QGAIN - 1 ) ) / ( ( ( MAX_QGAIN_DB
 - MIN_QGAIN_DB ) * 128 ) / 6 ) )
#define INV_SCALE_Q16   ( ( 65536 * ( ( ( MAX_QGAIN_DB - MIN_QGAIN_DB ) * 128 ) /
 6 ) ) / ( N_LEVELS_QGAIN - 1 ) )

/* Gain scalar quantization with hysteresis, uniform on log scale */
void SKP_Silk_gains_quant(
    SKP_int                         ind[ NB_SUBFR ],        /* O    gain indices
                    */
    SKP_int32                       gain_Q16[ NB_SUBFR ],   /* I/O  gains (quanti
zed out)                    */
    SKP_int                         *prev_ind,              /* I/O  last index in
 previous frame             */
```

```
    const SKP_int                   conditional             /* I    first gain is
 delta coded if 1          */
)
{
    SKP_int k;

    for( k = 0; k < NB_SUBFR; k++ ) {
        /* Add half of previous quantization error, convert to log scale, scale,
floor() */
        ind[ k ] = SKP_SMULWB( SCALE_Q16, SKP_Silk_lin2log( gain_Q16[ k ] ) - OFF
SET );

        /* Round towards previous quantized gain (hysteresis) */
        if( ind[ k ] < *prev_ind ) {
            ind[ k ]++;
        }

        /* Compute delta indices and limit */
        if( k == 0 && conditional == 0 ) {
            /* Full index */
            ind[ k ] = SKP_LIMIT( ind[ k ], 0, N_LEVELS_QGAIN - 1 );
            ind[ k ] = SKP_max_int( ind[ k ], *prev_ind + MIN_DELTA_GAIN_QUANT );
            *prev_ind = ind[ k ];
        } else {
            /* Delta index */
            ind[ k ] = SKP_LIMIT( ind[ k ] - *prev_ind, MIN_DELTA_GAIN_QUANT, MAX
_DELTA_GAIN_QUANT );
            /* Accumulate deltas */
            *prev_ind += ind[ k ];
            /* Shift to make non-negative */
            ind[ k ] -= MIN_DELTA_GAIN_QUANT;
        }

        /* Convert to linear scale and scale */
        gain_Q16[ k ] = SKP_Silk_log2lin( SKP_min_32( SKP_SMULWB( INV_SCALE_Q16,
*prev_ind ) + OFFSET, 3967 ) ); /* 3967 = 31 in Q7 */
    }
}

/* Gains scalar dequantization, uniform on log scale */
void SKP_Silk_gains_dequant(
    SKP_int32                       gain_Q16[ NB_SUBFR ],   /* O    quantized gai
ns                        */
    const SKP_int                   ind[ NB_SUBFR ],        /* I    gain indices
                          */
    SKP_int                         *prev_ind,              /* I/O  last index in
 previous frame           */
    const SKP_int                   conditional             /* I    first gain is
 delta coded if 1          */
)
{
    SKP_int   k;

    for( k = 0; k < NB_SUBFR; k++ ) {
        if( k == 0 && conditional == 0 ) {
            *prev_ind = ind[ k ];
        } else {
```

```
            /* Delta index */
            *prev_ind += ind[ k ] + MIN_DELTA_GAIN_QUANT;
        }

        /* Convert to linear scale and scale */
        gain_Q16[ k ] = SKP_Silk_log2lin( SKP_min_32( SKP_SMULWB( INV_SCALE_Q16,
*prev_ind ) + OFFSET, 3967 ) ); /* 3967 = 31 in Q7 */
    }
}
```

A.44. src/SKP_Silk_HP_variable_cutoff_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

#if HIGH_PASS_INPUT

#define SKP_RADIANS_CONSTANT_Q19            1482    // 0.45f * 2.0f * 3.141592653
59 / 1000
#define SKP_LOG2_VARIABLE_HP_MIN_FREQ_Q7    809     // log(80) in Q7

/* High-pass filter with cutoff frequency adaptation based on pitch lag statistic
s */
```

```
void SKP_Silk_HP_variable_cutoff_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,             /* I/O  Encoder state FIX
                            */
    SKP_Silk_encoder_control_FIX    *psEncCtrl,         /* I/O  Encoder control F
IX                          */
    SKP_int16                       *out,               /* O    high-pass filtere
d output signal             */
    const SKP_int16                 *in                 /* I    input signal
                            */
)
{
    SKP_int   quality_Q15;
    SKP_int32 B_Q28[ 3 ], A_Q28[ 2 ];
    SKP_int32 Fc_Q19, r_Q28, r_Q22;
    SKP_int32 pitch_freq_Hz_Q16, pitch_freq_log_Q7, delta_freq_Q7;

    /*****************************************/
    /* Estimate Low End of Pitch Frequency Range */
    /*****************************************/
    if( psEnc->sCmn.prev_sigtype == SIG_TYPE_VOICED ) {
        /* difference, in log domain */
        pitch_freq_Hz_Q16 = SKP_DIV32_16( SKP_LSHIFT( SKP_MUL( psEnc->sCmn.fs_kHz
, 1000 ), 16 ), psEnc->sCmn.prevLag );
        pitch_freq_log_Q7 = SKP_Silk_lin2log( pitch_freq_Hz_Q16 ) - ( 16 << 7 );
//0x70

        /* adjustment based on quality */
        quality_Q15 = psEncCtrl->input_quality_bands_Q15[ 0 ];
        pitch_freq_log_Q7 = SKP_SUB32( pitch_freq_log_Q7, SKP_SMULWB( SKP_SMULWB(
 SKP_LSHIFT( quality_Q15, 2 ), quality_Q15 ),
            pitch_freq_log_Q7 - SKP_LOG2_VARIABLE_HP_MIN_FREQ_Q7 ) );
        pitch_freq_log_Q7 = SKP_ADD32( pitch_freq_log_Q7, SKP_RSHIFT( 19661 - qua
lity_Q15, 9 ) ); // 19661_Q15 = 0.6_Q0

        //delta_freq = pitch_freq_log - psEnc->variable_HP_smth1;
        delta_freq_Q7 = pitch_freq_log_Q7 - SKP_RSHIFT( psEnc->variable_HP_smth1_
Q15, 8 );
        if( delta_freq_Q7 < 0 ) {
            /* less smoothing for decreasing pitch frequency, to track something
close to the minimum */
            delta_freq_Q7 = SKP_MUL( delta_freq_Q7, 3 );
        }

        /* limit delta, to reduce impact of outliers */
        delta_freq_Q7 = SKP_LIMIT( delta_freq_Q7, -VARIABLE_HP_MAX_DELTA_FREQ_Q7,
 VARIABLE_HP_MAX_DELTA_FREQ_Q7 );

        /* update smoother */
        psEnc->variable_HP_smth1_Q15 = SKP_SMLAWB( psEnc->variable_HP_smth1_Q15,
            SKP_MUL( SKP_LSHIFT( psEnc->speech_activity_Q8, 1 ), delta_freq_Q7 ),
 VARIABLE_HP_SMTH_COEF1_Q16 );
    }
    /* second smoother */
    psEnc->variable_HP_smth2_Q15 = SKP_SMLAWB( psEnc->variable_HP_smth2_Q15,
        psEnc->variable_HP_smth1_Q15 - psEnc->variable_HP_smth2_Q15, VARIABLE_HP_
SMTH_COEF2_Q16 );

    /* convert from log scale to Hertz */
    psEncCtrl->pitch_freq_low_Hz = SKP_Silk_log2lin( SKP_RSHIFT( psEnc->variable_
HP_smth2_Q15, 8 ) ); //pow( 2.0, psEnc->variable_HP_smth2 );

    /* limit frequency range */
```

```
    psEncCtrl->pitch_freq_low_Hz = SKP_LIMIT( psEncCtrl->pitch_freq_low_Hz, VARIA
BLE_HP_MIN_FREQ_Q0, VARIABLE_HP_MAX_FREQ_Q0 );

    /*******************************/
    /* Compute Filter Coefficients  */
    /*******************************/
    /* compute cut-off frequency, in radians */
    //Fc_num   = (SKP_float)( 0.45f * 2.0f * 3.14159265359 * psEncCtrl->pitch_fre
q_low_Hz );
    //Fc_denom = (SKP_float)( 1e3f * psEnc->sCmn.fs_kHz );
    SKP_assert( psEncCtrl->pitch_freq_low_Hz <= SKP_int32_MAX / SKP_RADIANS_CONST
ANT_Q19 );
    Fc_Q19 = SKP_DIV32_16( SKP_SMULBB( SKP_RADIANS_CONSTANT_Q19, psEncCtrl->pitch
_freq_low_Hz ), psEnc->sCmn.fs_kHz ); // range: 3704 - 27787, 11-15 bits
    SKP_assert( Fc_Q19 >=  3704 );
    SKP_assert( Fc_Q19 <= 27787 );

    r_Q28 = ( 1 << 28 ) - SKP_MUL( 471, Fc_Q19 ); // 471_Q9 = 0.92_Q0, range: 255
347779 to 266690872, 27-28 bits
    SKP_assert( r_Q28 >= 255347779 );
    SKP_assert( r_Q28 <= 266690872 );

    /* b = r * [ 1; -2; 1 ]; */
    /* a = [ 1; -2 * r * ( 1 - 0.5 * Fc^2 ); r^2 ]; */
    B_Q28[ 0 ] = r_Q28;
    B_Q28[ 1 ] = SKP_LSHIFT( -r_Q28, 1 );
    B_Q28[ 2 ] = r_Q28;

    // -r * ( 2 - Fc * Fc );
    r_Q22  = SKP_RSHIFT( r_Q28, 6 );
    A_Q28[ 0 ] = SKP_SMULWW( r_Q22, SKP_SMULWW( Fc_Q19, Fc_Q19 ) - ( 2 << 22 ) );
    A_Q28[ 1 ] = SKP_SMULWW( r_Q22, r_Q22 );

    /*******************************/
    /* High-Pass Filter             */
    /*******************************/
    SKP_Silk_biquad_alt( in, B_Q28, A_Q28, psEnc->sCmn.In_HP_State, out, psEnc->s
Cmn.frame_length );
}

#endif // HIGH_PASS_INPUT
```

A.45. src/SKP_Silk_init_encoder_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
```

```
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/*********************************/
/* Initialize Silk Encoder state */
/*********************************/
SKP_int SKP_Silk_init_encoder_FIX(
    SKP_Silk_encoder_state_FIX  *psEnc          /* I/O  Pointer to Silk encoder s
tate               */
)
{
    SKP_int ret = 0;
    /* Clear the entire encoder state */
    SKP_memset( psEnc, 0, sizeof( SKP_Silk_encoder_state_FIX ) );

    /* Initialize to 24 kHz sampling, 20 ms packets, 25 kbps, 0% packet loss, and
 init non-zero values */
    ret = SKP_Silk_control_encoder_FIX( psEnc, 24, 20, 25, 0, 0, 0, 10, 1 );

#if HIGH_PASS_INPUT
    psEnc->variable_HP_smth1_Q15 = 200844; /* = SKP_Silk_log2(70)_Q0; */
    psEnc->variable_HP_smth2_Q15 = 200844; /* = SKP_Silk_log2(70)_Q0; */
#endif

    /* Used to deactivate e.g. LSF interpolation and fluctuation reduction */
    psEnc->sCmn.first_frame_after_reset = 1;
    psEnc->sCmn.fs_kHz_changed          = 0;
    psEnc->sCmn.LBRR_enabled            = 0;

    /* Initialize Silk VAD */
    ret += SKP_Silk_VAD_Init( &psEnc->sCmn.sVAD );
```

```
    /* Initialize NSQ */
    psEnc->sNSQ.prev_inv_gain_Q16      = 65536; /* 1.0 in Q16 */
    psEnc->sNSQ_LBRR.prev_inv_gain_Q16 = 65536; /* 1.0 in Q16 */

    psEnc->sCmn.bitstream_v     = USE_BIT_STREAM_V;

    return( ret );
}
```

A.46. src/SKP_Silk_Inlines.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*! \file SKP_Silk_Inlines.h
 *  \brief SigProcFix_Inlines.h defines inline signal processing functions.
 */

#ifndef _SKP_SILK_FIX_INLINES_H_
#define _SKP_SILK_FIX_INLINES_H_

#ifdef  __cplusplus
```

```
extern "C"
{
#endif

/* count leading zeros of SKP_int64 */
SKP_INLINE SKP_int32 SKP_Silk_CLZ64(SKP_int64 in)
{
    SKP_int32 in_upper;

    in_upper = (SKP_int32)SKP_RSHIFT64(in, 32);
    if (in_upper == 0) {
        /* Search in the lower 32 bits */
        return 32 + SKP_Silk_CLZ32( (SKP_int32) in );
    } else {
        /* Search in the upper 32 bits */
        return SKP_Silk_CLZ32( in_upper );
    }
}

/* get number of leading zeros and fractional part (the bits right after the lead
ing one */
SKP_INLINE void SKP_Silk_CLZ_FRAC(SKP_int32 in,            /* I: input */
                                  SKP_int32 *lz,           /* O: number of lead
ing zeros */
                                  SKP_int32 *frac_Q7)      /* O: the 7 bits rig
ht after the leading one */
{
    SKP_int32 leadingZeros;

    leadingZeros = SKP_Silk_CLZ32(in);
    *lz = leadingZeros;
    if( leadingZeros < 24 ) {
        *frac_Q7 = SKP_RSHIFT(in, 24 - leadingZeros) & 0x7F;
    } else {
        *frac_Q7 = SKP_LSHIFT(in, leadingZeros - 24) & 0x7F;
    }
}

/* Approximation of square root                                          */
/* Accuracy: < +/- 10% for output values > 15                            */
/*             < +/- 2.5% for output values > 120                        */
SKP_INLINE SKP_int32 SKP_Silk_SQRT_APPROX(SKP_int32 x)
{
    SKP_int32 y, lz, frac_Q7;

    if( x <= 0 ) {
        return 0;
    }

    SKP_Silk_CLZ_FRAC(x, &lz, &frac_Q7);
```

```
    if( lz & 1 ) {
        y = 32768;
    } else {
        y = 46214;        /* 46214 = sqrt(2) * 32768 */
    }

    /* get scaling right */
    y >>= SKP_RSHIFT(lz, 1);

    /* increment using fractional part of input */
    y = SKP_SMLAWB(y, y, SKP_SMULBB(213, frac_Q7));

    return y;
}

/* returns the number of left shifts before overflow for a 16 bit number (ITU def
inition with norm(0)=0) */
SKP_INLINE SKP_int32 SKP_Silk_norm16(SKP_int16 a) {

  SKP_int32 a32;

  /* if ((a == 0) || (a == SKP_int16_MIN)) return(0); */
  if ((a << 1) == 0) return(0);

  a32 = a;
  /* if (a32 < 0) a32 = -a32 - 1; */
  a32 ^= SKP_RSHIFT(a32, 31);

  return SKP_Silk_CLZ32(a32) - 17;
}

/* returns the number of left shifts before overflow for a 32 bit number (ITU def
inition with norm(0)=0) */
SKP_INLINE SKP_int32 SKP_Silk_norm32(SKP_int32 a) {

  /* if ((a == 0) || (a == SKP_int32_MIN)) return(0); */
  if ((a << 1) == 0) return(0);

  /* if (a < 0) a = -a - 1; */
  a ^= SKP_RSHIFT(a, 31);

  return SKP_Silk_CLZ32(a) - 1;
}

/* Divide two int32 values and return result as int32 in a given Q-domain */
SKP_INLINE SKP_int32 SKP_DIV32_varQ(    /* O    returns a good approximation of "
(a32 << Qres) / b32" */
    const SKP_int32        a32,         /* I    numerator (Q0)                 *
/
    const SKP_int32        b32,         /* I    denominator (Q0)               *
/
    const SKP_int        Qres           /* I    Q-domain of result (>= 0)      *
/
)
```

```
{
    SKP_int   a_headrm, b_headrm, lshift;
    SKP_int32 b32_inv, a32_nrm, b32_nrm, result;

    SKP_assert( b32 != 0 );
    SKP_assert( Qres >= 0 );

    /* Compute number of bits head room and normalize inputs */
    a_headrm = SKP_Silk_CLZ32( SKP_abs(a32) ) - 1;
    a32_nrm = SKP_LSHIFT(a32, a_headrm);                                    /* Q:
 a_headrm                    */
    b_headrm = SKP_Silk_CLZ32( SKP_abs(b32) ) - 1;
    b32_nrm = SKP_LSHIFT(b32, b_headrm);                                    /* Q:
 b_headrm                    */

    /* Inverse of b32, with 14 bits of precision */
    b32_inv = SKP_DIV32_16( SKP_int32_MAX >> 2, SKP_RSHIFT(b32_nrm, 16) );  /* Q:
 29 + 16 - b_headrm        */

    /* First approximation */
    result = SKP_SMULWB(a32_nrm, b32_inv);                                  /* Q:
 29 + a_headrm - b_headrm    */

    /* Compute residual by subtracting product of denominator and first approxima
tion */
    a32_nrm -= SKP_LSHIFT_ovflw( SKP_SMMUL(b32_nrm, result), 3 );           /* Q:
 a_headrm                    */

    /* Refinement */
    result = SKP_SMLAWB(result, a32_nrm, b32_inv);                          /* Q:
 29 + a_headrm - b_headrm    */

    /* Convert to Qres domain */
    lshift = 29 + a_headrm - b_headrm - Qres;
    if( lshift <= 0 ) {
        return SKP_LSHIFT_SAT32(result, -lshift);
    } else {
        if( lshift < 32){
            return SKP_RSHIFT(result, lshift);
        } else {
            /* Avoid undefined result */
            return 0;
        }
    }
}

/* Invert int32 value and return result as int32 in a given Q-domain */
SKP_INLINE SKP_int32 SKP_INVERSE32_varQ(    /* O    returns a good approximation
of "(1 << Qres) / b32" */
    const SKP_int32        b32,             /* I    denominator (Q0)
   */
    const SKP_int        Qres               /* I    Q-domain of result (> 0)
   */
)
{
    SKP_int   b_headrm, lshift;
    SKP_int32 b32_inv, b32_nrm, err_Q32, result;
```

```
    SKP_assert( b32 != 0 );
    SKP_assert( Qres > 0 );

    /* Compute number of bits head room and normalize input */
    b_headrm = SKP_Silk_CLZ32( SKP_abs(b32) ) - 1;
    b32_nrm = SKP_LSHIFT(b32, b_headrm);                                    /* Q:
 b_headrm                */

    /* Inverse of b32, with 14 bits of precision */
    b32_inv = SKP_DIV32_16( SKP_int32_MAX >> 2, SKP_RSHIFT(b32_nrm, 16) );  /* Q:
 29 + 16 - b_headrm    */

    /* First approximation */
    result = SKP_LSHIFT(b32_inv, 16);                                       /* Q:
 61 - b_headrm            */

    /* Compute residual by subtracting product of denominator and first approxima
tion from one */
    err_Q32 = SKP_LSHIFT_ovflw( -SKP_SMULWB(b32_nrm, b32_inv), 3 );         /* Q3
2                          */

    /* Refinement */
    result = SKP_SMLAWW(result, err_Q32, b32_inv);                          /* Q:
 61 - b_headrm            */

    /* Convert to Qres domain */
    lshift = 61 - b_headrm - Qres;
    if( lshift <= 0 ) {
        return SKP_LSHIFT_SAT32(result, -lshift);
    } else {
        if( lshift < 32){
            return SKP_RSHIFT(result, lshift);
        }else{
            /* Avoid undefined result */
            return 0;
        }
    }
}

#define SKP_SIN_APPROX_CONST0       (1073735400)
#define SKP_SIN_APPROX_CONST1        (-82778932)
#define SKP_SIN_APPROX_CONST2          (1059577)
#define SKP_SIN_APPROX_CONST3            (-5013)

/* Sine approximation; an input of 65536 corresponds to 2 * pi */
/* Uses polynomial expansion of the input to the power 0, 2, 4 and 6 */
/* The relative error is below 1e-5 */
SKP_INLINE SKP_int32 SKP_Silk_SIN_APPROX_Q24(        /* O    returns approximatel
y 2^24 * sin(x * 2 * pi / 65536) */
    SKP_int32        x
)
{
    SKP_int y_Q30;

    /* Keep only bottom 16 bits (the function repeats itself with period 65536) *
/
```

```
    x &= 65535;

    /* Split range in four quadrants */
    if( x <= 32768 ) {
        if( x < 16384 ) {
            /* Return cos(pi/2 - x) */
            x = 16384 - x;
        } else {
            /* Return cos(x - pi/2) */
            x -= 16384;
        }
        if( x < 1100 ) {
            /* Special case: high accuracy */
            return SKP_SMLAWB( 1 << 24, SKP_MUL( x, x ), -5053 );
        }
        x = SKP_SMULWB( SKP_LSHIFT( x, 8 ), x );        /* contains x^2 in Q20 */
        y_Q30 = SKP_SMLAWB( SKP_SIN_APPROX_CONST2, x, SKP_SIN_APPROX_CONST3 );
        y_Q30 = SKP_SMLAWW( SKP_SIN_APPROX_CONST1, x, y_Q30 );
        y_Q30 = SKP_SMLAWW( SKP_SIN_APPROX_CONST0 + 66, x, y_Q30 );
    } else {
        if( x < 49152 ) {
            /* Return -cos(3*pi/2 - x) */
            x = 49152 - x;
        } else {
            /* Return -cos(x - 3*pi/2) */
            x -= 49152;
        }
        if( x < 1100 ) {
            /* Special case: high accuracy */
            return SKP_SMLAWB( -1 << 24, SKP_MUL( x, x ), 5053 );
        }
        x = SKP_SMULWB( SKP_LSHIFT( x, 8 ), x );        /* contains x^2 in Q20 */
        y_Q30 = SKP_SMLAWB( -SKP_SIN_APPROX_CONST2, x, -SKP_SIN_APPROX_CONST3 );
        y_Q30 = SKP_SMLAWW( -SKP_SIN_APPROX_CONST1, x, y_Q30 );
        y_Q30 = SKP_SMLAWW( -SKP_SIN_APPROX_CONST0, x, y_Q30 );
    }
    return SKP_RSHIFT_ROUND( y_Q30, 6 );
}

/* Cosine approximation; an input of 65536 corresponds to 2 * pi */
/* The relative error is below 1e-5 */
SKP_INLINE SKP_int32 SKP_Silk_COS_APPROX_Q24(        /* O    returns approximatel
y 2^24 * cos(x * 2 * pi / 65536) */
    SKP_int32        x
)
{
    return SKP_Silk_SIN_APPROX_Q24( x + 16384 );
}
```

```
#ifdef  __cplusplus
}
#endif

#endif //_SKP_SILK_FIX_INLINES_H_
```

A.47. src/SKP_Silk_inner_prod_aligned.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*
 *
 * SKP_Silk_inner_prod_aligned.c                                        *
 *
 *
 *
 *
 * Copyright 2008 (c), Skype Limited
 *
 * Date: 080601
 *
 *
 */
#include "SKP_Silk_SigProc_FIX.h"

/* sum= for(i=0;i<len;i++)inVec1[i]*inVec2[i];      ---        inner product    *
/
/* Note for ARM asm:                                                        *
/
```

```
/*        * inVec1 and inVec2 should be at least 2 byte aligned.    (Or defined a
s short/int16) */
/*        * len should be positive 16bit integer.                                *
/
/*        * only when len>6, memory access can be reduced by half.               *
/

SKP_int32 SKP_Silk_inner_prod_aligned(
    const SKP_int16* const inVec1,  /*    I input vector 1    */
    const SKP_int16* const inVec2,  /*    I input vector 2    */
    const SKP_int             len   /*    I vector lengths    */
)
{
    SKP_int   i;
    SKP_int32 sum = 0;
    for( i = 0; i < len; i++ ) {
        sum = SKP_SMLABB( sum, inVec1[ i ], inVec2[ i ] );
    }
    return sum;
}

SKP_int64 SKP_Silk_inner_prod_aligned_64(
    const SKP_int32 *inVec1,        /*    I input vector 1    */
    const SKP_int32 *inVec2,        /*    I input vector 2    */
    const SKP_int   len             /*    I vector lengths    */
)
{
    SKP_int   i;
    SKP_int64 sum = 0;
    for( i = 0; i < len; i++ ) {
        sum = SKP_SMLAL( sum, inVec1[ i ], inVec2[ i ] );
    }
    return sum;
}
SKP_int64 SKP_Silk_inner_prod16_aligned_64(
    const SKP_int16 *inVec1,        /*    I input vector 1    */
    const SKP_int16 *inVec2,        /*    I input vector 2    */
    const SKP_int   len             /*    I vector lengths    */
)
{
    SKP_int   i;
    SKP_int64 sum = 0;
    for( i = 0; i < len; i++ ) {
        sum = SKP_SMLALBB( sum, inVec1[ i ], inVec2[ i ] );
    }
    return sum;
}

SKP_int32 SKP_Silk_inner_prod16_aligned_sat(
    const SKP_int16* const inVec1,  /*    I input vector 1  */
    const SKP_int16* const inVec2,  /*    I input vector 2  */
```

```
    const SKP_int    len            /*    I vector lengths  */
)
{
    SKP_int   i;
    SKP_int32 sum = 0;
    for( i = 0; i < len; i++ ) {
        sum = SKP_ADD_SAT32( sum, SKP_SMULBB( inVec1[ i ], inVec2[ i ] ) );
    }
    return sum;
}
```

A.48. src/SKP_Silk_interpolate.c

```
#include "SKP_Silk_main.h"

/* Interpolate two vectors */
void SKP_Silk_interpolate(
    SKP_int                     xi[ MAX_LPC_ORDER ],    /* O    interpolated
vector                      */
    const SKP_int               x0[ MAX_LPC_ORDER ],    /* I    first vector
                            */
    const SKP_int               x1[ MAX_LPC_ORDER ],    /* I    second vector
                            */
    const SKP_int               ifact_Q2,               /* I    interp. facto
r, weight on 2nd vector     */
    const SKP_int               d                       /* I    number of par
ameters                     */
)
{
    SKP_int i;

    SKP_assert( ifact_Q2 >= 0 );
    SKP_assert( ifact_Q2 <= ( 1 << 2 ) );

    for( i = 0; i < d; i++ ) {
        xi[ i ] = ( SKP_int )( ( SKP_int32 )x0[ i ] + SKP_RSHIFT( SKP_MUL( ( SKP_
int32 )x1[ i ] - ( SKP_int32 )x0[ i ], ifact_Q2 ), 2 ) );
    }
}
```

A.49. src/SKP_Silk_k2a.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_Silk_k2a.c                                                     *
 *                                                                      *
 * Step up function, converts reflection coefficients to prediction     *
 * coefficients                                                         *
 *                                                                      *
 * Copyright 2008 (c), Skype Limited                                    *
 * Date: 080103                                                         *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"

/* Step up function, converts reflection coefficients to prediction coefficients
*/
void SKP_Silk_k2a(
    SKP_int32           *A_Q24,                 /* O:    Prediction coefficients
 [order] Q24         */
    const SKP_int16     *rc_Q15,                /* I:    Reflection coefficients
 [order] Q15         */
    const SKP_int32     order                   /* I:    Prediction order
                    */
)
{
```

```
    SKP_int   k, n;
    SKP_int32 Atmp[ SigProc_MAX_ORDER_LPC ];

    for( k = 0; k < order; k++ ) {
        for( n = 0; n < k; n++ ) {
            Atmp[ n ] = A_Q24[ n ];
        }
        for( n = 0; n < k; n++ ) {
            A_Q24[ n ] = SKP_SMLAWB( A_Q24[ n ], SKP_LSHIFT( Atmp[ k - n - 1 ], 1
 ), rc_Q15[ k ] );
        }
        A_Q24[ k ] = -SKP_LSHIFT( (SKP_int32)rc_Q15[ k ], 9 );
    }
}
```

A.50. src/SKP_Silk_k2a_Q16.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_Silk_k2a.c                                                       *
 *                                                                      *
```

```
 * Step up function, converts reflection coefficients to prediction     *
 * coefficients                                                         *
 *                                                                      *
 * Copyright 2008 (c), Skype Limited                                    *
 * Date: 080103                                                         *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"

/* Step up function, converts reflection coefficients to prediction coefficients
*/
void SKP_Silk_k2a_Q16(
    SKP_int32           *A_Q24,                 /* O:    Prediction coefficients
 [order] Q24         */
    const SKP_int32     *rc_Q16,                /* I:    Reflection coefficients
 [order] Q16         */
    const SKP_int32     order                   /* I:    Prediction order
                    */
)
{
    SKP_int   k, n;
    SKP_int32 Atmp[ SigProc_MAX_ORDER_LPC ];

    for( k = 0; k < order; k++ ) {
        for( n = 0; n < k; n++ ) {
            Atmp[ n ] = A_Q24[ n ];
        }
        for( n = 0; n < k; n++ ) {
            A_Q24[ n ] = SKP_SMLAWW( A_Q24[ n ], Atmp[ k - n - 1 ], rc_Q16[ k ] )
;
        }
        A_Q24[ k ] = -SKP_LSHIFT( rc_Q16[ k ], 8 );
    }
}
```

A.51. src/SKP_Silk_LBRR_reset.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

/* Resets LBRR buffer, used if packet size changes */
void SKP_Silk_LBRR_reset(
    SKP_Silk_encoder_state      *psEncC             /* I/O  state
                        */
)
{
    SKP_int i;

    for( i = 0; i < MAX_LBRR_DELAY; i++ ) {
        psEncC->LBRR_buffer[ i ].usage = SKP_SILK_NO_LBRR;
    }
}
```

A.52. src/SKP_Silk_lin2log.c

```
/*                                                                      *
 * SKP_Silk_lin2log.c                                                   *
 *                                                                      *
 * Convert input to a log scale                                         *
 * Approximation of 128 * log2()                                        *
 *                                                                      *
 * Copyright 2006 (c), Skype Limited                                    *
 * Date: 060221                                                         *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"
/* Approximation of 128 * log2() (very close inverse of approx 2^() below) */
/* Convert input to a log scale    */
SKP_int32 SKP_Silk_lin2log( const SKP_int32 inLin )    /* I:    Input in linear s
cale */
{
    SKP_int32 lz, frac_Q7;

    SKP_Silk_CLZ_FRAC( inLin, &lz, &frac_Q7 );

    /* Piece-wise parabolic approximation */
    return( SKP_LSHIFT( 31 - lz, 7 ) + SKP_SMLAWB( frac_Q7, SKP_MUL( frac_Q7, 128
 - frac_Q7 ), 179 ) );
}
```

A.53. src/SKP_Silk_log2lin.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_Silk_log2lin.c                                                   *
 *                                                                      *
 * Convert input to a linear scale                                      *
 *                                                                      *
 * Copyright 2006 (c), Skype Limited                                    *
 * Date: 060221                                                         *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"

/* Approximation of 2^() (very close inverse of SKP_Silk_lin2log()) */
/* Convert input to a linear scale    */
SKP_int32 SKP_Silk_log2lin( const SKP_int32 inLog_Q7 )    /* I:    Input on log s
cale */
{
    SKP_int32 out, frac_Q7;

    if( inLog_Q7 < 0 ) {
        return 0;
```

```
    }

    out = SKP_LSHIFT( 1, SKP_RSHIFT( inLog_Q7, 7 ) );
    frac_Q7 = inLog_Q7 & 0x7F;
    if( inLog_Q7 < 2048 ) {
        /* Piece-wise parabolic approximation */
        out = SKP_ADD_RSHIFT( out, SKP_MUL( out, SKP_SMLAWB( frac_Q7, SKP_MUL( fr
ac_Q7, 128 - frac_Q7 ), -174 ) ), 7 );
    } else {
        /* Piece-wise parabolic approximation */
        out = SKP_MLA( out, SKP_RSHIFT( out, 7 ), SKP_SMLAWB( frac_Q7, SKP_MUL( f
rac_Q7, 128 - frac_Q7 ), -174 ) );
    }
    return out;
}
```

A.54. src/SKP_Silk_lowpass_int.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_lowpass_int.c                                              *
 *                                                                     *
```

```
 * First order low-pass filter, with input as SKP_int32, running at     *
 * 48 kHz                                                               *
 *                                                                      *
 * Copyright 2006 (c), Skype Limited                                    *
 * Date: 060221                                                         *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"

/* First order low-pass filter, with input as SKP_int32, running at 48 kHz
 */
void SKP_Silk_lowpass_int(
    const SKP_int32      *in,            /* I:    Q25 48 kHz signal; length = len
 */
    SKP_int32            *S,             /* I/O: Q25 state; length = 1
 */
    SKP_int32            *out,           /* O:    Q25 48 kHz signal; length = len
 */
    const SKP_int32      len             /* I:    Number of samples
 */
)
{
    SKP_int        k;
    SKP_int32    in_tmp, out_tmp, state;

    state = S[ 0 ];
    for( k = len; k > 0; k-- ) {
        in_tmp  = *in++;
        in_tmp -= SKP_RSHIFT( in_tmp, 2 );              /* multiply by 0.75 */
        out_tmp = state + in_tmp;                       /* zero at nyquist  */
        state   = in_tmp - SKP_RSHIFT( out_tmp, 1 );    /* pole             */
        *out++  = out_tmp;
    }
    S[ 0 ] = state;
}
```

A.55. src/SKP_Silk_lowpass_short.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
```

```
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_lowpass_short.c                                          *
 *                                                                     *
 * First order low-pass filter, with input as SKP_int16, running at    *
 * 48 kHz                                                              *
 *                                                                     *
 * Copyright 2006 (c), Skype Limited                                   *
 * Date: 060221                                                        *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"


/* First order low-pass filter, with input as SKP_int16, running at 48 kHz   */
void SKP_Silk_lowpass_short(
    const SKP_int16          *in,        /* I:   Q15 48 kHz signal; [len]    */
    SKP_int32                *S,         /* I/O: Q25 state; length = 1       */
    SKP_int32                *out,       /* O:   Q25 48 kHz signal; [len]    */
    const SKP_int32          len         /* O:   Signal length               */
)
{
    SKP_int        k;
    SKP_int32    in_tmp, out_tmp, state;

    state = S[ 0 ];
    for( k = 0; k < len; k++ ) {
        in_tmp   = SKP_MUL( 768, (SKP_int32)in[k] );    /* multiply by 0.75, goin
g from Q15 to Q25 */
        out_tmp  = state + in_tmp;                      /* zero at nyquist
                  */
        state    = in_tmp - SKP_RSHIFT( out_tmp, 1 );   /* pole
                  */
        out[ k ] = out_tmp;
    }
    S[ 0 ] = state;
}
```

A.56. src/SKP_Silk_LP_variable_cutoff.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*

    Elliptic/Cauer filters designed with 0.1 dB passband ripple,
        80 dB minimum stopband attenuation, and
        [0.95 : 0.15 : 0.35] normalized cut off frequencies.

*/
#include "SKP_Silk_main.h"

#if SWITCH_TRANSITION_FILTERING

/* Helper function, that interpolates the filter taps */
SKP_INLINE void SKP_Silk_LP_interpolate_filter_taps(
    SKP_int32           B_Q28[ TRANSITION_NB ],
    SKP_int32           A_Q28[ TRANSITION_NA ],
    const SKP_int       ind,
    const SKP_int32     fac_Q16
)
```

```
{
    SKP_int nb, na;

    if( ind < TRANSITION_INT_NUM - 1 ) {
        if( fac_Q16 > 0 ) {
            if( fac_Q16 == SKP_SAT16( fac_Q16 ) ) { /* fac_Q16 is in range of a 1
6-bit int */
                /* Piece-wise linear interpolation of B and A */
                for( nb = 0; nb < TRANSITION_NB; nb++ ) {
                    B_Q28[ nb ] = SKP_SMLAWB(
                        SKP_Silk_Transition_LP_B_Q28[ ind     ][ nb ],
                        SKP_Silk_Transition_LP_B_Q28[ ind + 1 ][ nb ] -
                        SKP_Silk_Transition_LP_B_Q28[ ind     ][ nb ],
                        fac_Q16 );
                }
                for( na = 0; na < TRANSITION_NA; na++ ) {
                    A_Q28[ na ] = SKP_SMLAWB(
                        SKP_Silk_Transition_LP_A_Q28[ ind     ][ na ],
                        SKP_Silk_Transition_LP_A_Q28[ ind + 1 ][ na ] -
                        SKP_Silk_Transition_LP_A_Q28[ ind     ][ na ],
                        fac_Q16 );
                }
            } else if( fac_Q16 == ( 1 << 15 ) ) { /* Neither fac_Q16 nor ( ( 1 <<
 16 ) - fac_Q16 ) is in range of a 16-bit int */

                /* Piece-wise linear interpolation of B and A */
                for( nb = 0; nb < TRANSITION_NB; nb++ ) {
                    B_Q28[ nb ] = SKP_RSHIFT(
                        SKP_Silk_Transition_LP_B_Q28[ ind     ][ nb ] +
                        SKP_Silk_Transition_LP_B_Q28[ ind + 1 ][ nb ],
                        1 );
                }
                for( na = 0; na < TRANSITION_NA; na++ ) {
                    A_Q28[ na ] = SKP_RSHIFT(
                        SKP_Silk_Transition_LP_A_Q28[ ind     ][ na ] +
                        SKP_Silk_Transition_LP_A_Q28[ ind + 1 ][ na ],
                        1 );
                }
            } else { /* ( ( 1 << 16 ) - fac_Q16 ) is in range of a 16-bit int */

                SKP_assert( ( ( 1 << 16 ) - fac_Q16 ) == SKP_SAT16( ( ( 1 << 16 )
 - fac_Q16) ) );
                /* Piece-wise linear interpolation of B and A */
                for( nb = 0; nb < TRANSITION_NB; nb++ ) {
                    B_Q28[ nb ] = SKP_SMLAWB(
                        SKP_Silk_Transition_LP_B_Q28[ ind + 1 ][ nb ],
                        SKP_Silk_Transition_LP_B_Q28[ ind     ][ nb ] -
                        SKP_Silk_Transition_LP_B_Q28[ ind + 1 ][ nb ],
                        ( 1 << 16 ) - fac_Q16 );
                }
                for( na = 0; na < TRANSITION_NA; na++ ) {
```

```
                    A_Q28[ na ] = SKP_SMLAWB(
                        SKP_Silk_Transition_LP_A_Q28[ ind + 1 ][ na ],
                        SKP_Silk_Transition_LP_A_Q28[ ind     ][ na ] -
                        SKP_Silk_Transition_LP_A_Q28[ ind + 1 ][ na ],
                        ( 1 << 16 ) - fac_Q16 );
                }
            }
        } else {
            SKP_memcpy( B_Q28, SKP_Silk_Transition_LP_B_Q28[ ind ], TRANSITION_NB
 * sizeof( SKP_int32 ) );
            SKP_memcpy( A_Q28, SKP_Silk_Transition_LP_A_Q28[ ind ], TRANSITION_NA
 * sizeof( SKP_int32 ) );
        }
    } else {
        SKP_memcpy( B_Q28, SKP_Silk_Transition_LP_B_Q28[ TRANSITION_INT_NUM - 1 ]
, TRANSITION_NB * sizeof( SKP_int32 ) );
        SKP_memcpy( A_Q28, SKP_Silk_Transition_LP_A_Q28[ TRANSITION_INT_NUM - 1 ]
, TRANSITION_NA * sizeof( SKP_int32 ) );
    }
}

/* Low-pass filter with variable cutoff frequency based on  */
/* piece-wise linear interpolation between elliptic filters */
/* Start by setting psEncC->transition_frame_no = 1;            */
/* Deactivate by setting psEncC->transition_frame_no = 0;   */
void SKP_Silk_LP_variable_cutoff(
    SKP_Silk_LP_state               *psLP,          /* I/O  LP filter state
                */
    SKP_int16                       *out,           /* O    Low-pass filtered out
put signal     */
    const SKP_int16                 *in,            /* I    Input signal
                */
    const SKP_int                   frame_length    /* I    Frame length
                */
)
{
    SKP_int32   B_Q28[ TRANSITION_NB ], A_Q28[ TRANSITION_NA ];
    SKP_int     fac_Q16 = 0, ind = 0;

    SKP_assert( psLP->transition_frame_no >= 0 );
    SKP_assert( ( ( ( psLP->transition_frame_no <= TRANSITION_FRAMES_DOWN ) && (
psLP->mode == 0 ) ) ||
                  ( ( psLP->transition_frame_no <= TRANSITION_FRAMES_UP   ) && (
psLP->mode == 1 ) ) ) );

    /* Interpolate filter coefficients if needed */
    if( psLP->transition_frame_no > 0 ) {
        if( psLP->mode == 0 ) {
            if( psLP->transition_frame_no < TRANSITION_FRAMES_DOWN ) {
                /* Calculate index and interpolation factor for interpolation */
#if( TRANSITION_INT_STEPS_DOWN == 32 )
                fac_Q16 = SKP_LSHIFT( psLP->transition_frame_no, 16 - 5 );
#else
                fac_Q16 = SKP_DIV32_16( SKP_LSHIFT( psLP->transition_frame_no, 16
 ), TRANSITION_INT_STEPS_DOWN );
#endif
                ind      = SKP_RSHIFT( fac_Q16, 16 );
                fac_Q16 -= SKP_LSHIFT( ind, 16 );
```

```
                SKP_assert( ind >= 0 );
                SKP_assert( ind < TRANSITION_INT_NUM );

                /* Interpolate filter coefficients */
                SKP_Silk_LP_interpolate_filter_taps( B_Q28, A_Q28, ind, fac_Q16 )
;

                /* Increment transition frame number for next frame */
                psLP->transition_frame_no++;

            } else if( psLP->transition_frame_no == TRANSITION_FRAMES_DOWN ) {
                /* End of transition phase */
                SKP_Silk_LP_interpolate_filter_taps( B_Q28, A_Q28, TRANSITION_INT
_NUM - 1, 0 );
            }
        } else if( psLP->mode == 1 ) {
            if( psLP->transition_frame_no < TRANSITION_FRAMES_UP ) {
                /* Calculate index and interpolation factor for interpolation */
#if( TRANSITION_INT_STEPS_UP == 64 )
                fac_Q16 = SKP_LSHIFT( TRANSITION_FRAMES_UP - psLP->transition_fra
me_no, 16 - 6 );
#else
                fac_Q16 = SKP_DIV32_16( SKP_LSHIFT( TRANSITION_FRAMES_UP - psLP->
transition_frame_no, 16 ), TRANSITION_INT_STEPS_UP );
#endif
                ind      = SKP_RSHIFT( fac_Q16, 16 );
                fac_Q16 -= SKP_LSHIFT( ind, 16 );

                SKP_assert( ind >= 0 );
                SKP_assert( ind < TRANSITION_INT_NUM );

                /* Interpolate filter coefficients */
                SKP_Silk_LP_interpolate_filter_taps( B_Q28, A_Q28, ind, fac_Q16 )
;

                /* Increment transition frame number for next frame */
                psLP->transition_frame_no++;

            } else if( psLP->transition_frame_no == TRANSITION_FRAMES_UP ) {
                /* End of transition phase */
                SKP_Silk_LP_interpolate_filter_taps( B_Q28, A_Q28, 0, 0 );
            }
        }
    }

    if( psLP->transition_frame_no > 0 ) {
        /* ARMA low-pass filtering */
        SKP_assert( TRANSITION_NB == 3 && TRANSITION_NA == 2 );
        SKP_Silk_biquad_alt( in, B_Q28, A_Q28, psLP->In_LP_State, out, frame_leng
th );
    } else {
        /* Instead of using the filter, copy input directly to output */
        SKP_memcpy( out, in, frame_length * sizeof( SKP_int16 ) );
    }
```

```
}
#endif
```

A.57. src/SKP_Silk_LPC_inv_pred_gain.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_LPC_inverse_pred_gain.c                                    *
 *                                                                     *
 * Compute inverse of LPC prediction gain, and                         *
 * test if LPC coefficients are stable (all poles within unit circle)  *
 *                                                                     *
 * Copyright 2008 (c), Skype Limited                                       *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"
#define QA          16
#define A_LIMIT     65520

/* Compute inverse of LPC prediction gain, and                         */
/* test if LPC coefficients are stable (all poles within unit circle)  */
```

```
SKP_int SKP_Silk_LPC_inverse_pred_gain(       /* O:   Returns 1 if unstable, othe
rwise 0          */
    SKP_int32           *invGain_Q30,           /* O:   Inverse prediction gain,
Q30 energy domain  */
    const SKP_int16     *A_Q12,                 /* I:   Prediction coefficients,
Q12 [order]        */
    const SKP_int       order                   /* I:   Prediction order
                */
)
{
    SKP_int   k, n, headrm;
    SKP_int32 rc_Q31, rc_mult1_Q30, rc_mult2_Q16;
    SKP_int32 Atmp_QA[ 2 ][ SigProc_MAX_ORDER_LPC ], tmp_QA;
    SKP_int32 *Aold_QA, *Anew_QA;

    Anew_QA = Atmp_QA[ order & 1 ];
    /* Increase Q domain of the AR coefficients */
    for( k = 0; k < order; k++ ) {
        Anew_QA[ k ] = SKP_LSHIFT( (SKP_int32)A_Q12[ k ], QA - 12 );
    }

    *invGain_Q30 = ( 1 << 30 );
    for( k = order - 1; k > 0; k-- ) {
        /* Check for stability */
        if( ( Anew_QA[ k ] > A_LIMIT ) || ( Anew_QA[ k ] < -A_LIMIT ) ) {
            return 1;
        }

        /* Set RC equal to negated AR coef */
        rc_Q31 = -SKP_LSHIFT( Anew_QA[ k ], 31 - QA );

        /* rc_mult1_Q30 range: [ 1 : 2^30-1 ] */
        rc_mult1_Q30 = ( SKP_int32_MAX >> 1 ) - SKP_SMMUL( rc_Q31, rc_Q31 );
        SKP_assert( rc_mult1_Q30 > ( 1 << 15 ) );                    /* reduce A_L
IMIT if fails */
        SKP_assert( rc_mult1_Q30 < ( 1 << 30 ) );

        /* rc_mult2_Q16 range: [ 2^16 : SKP_int32_MAX ] */
        rc_mult2_Q16 = SKP_INVERSE32_varQ( rc_mult1_Q30, 46 );      /* 16 = 46 -
30 */

        /* Update inverse gain */
        /* invGain_Q30 range: [ 0 : 2^30 ] */
        *invGain_Q30 = SKP_LSHIFT( SKP_SMMUL( *invGain_Q30, rc_mult1_Q30 ), 2 );
        SKP_assert( *invGain_Q30 >= 0           );
        SKP_assert( *invGain_Q30 <= ( 1 << 30 ) );

        /* Swap pointers */
        Aold_QA = Anew_QA;
        Anew_QA = Atmp_QA[ k & 1 ];

        /* Update AR coefficient */
        headrm = SKP_Silk_CLZ32( rc_mult2_Q16 ) - 1;
        rc_mult2_Q16 = SKP_LSHIFT( rc_mult2_Q16, headrm );          /* Q: 16 + he
adrm */
```

```
        for( n = 0; n < k; n++ ) {
            tmp_QA = Aold_QA[ n ] - SKP_LSHIFT( SKP_SMMUL( Aold_QA[ k - n - 1 ],
rc_Q31 ), 1 );
            Anew_QA[ n ] = SKP_LSHIFT( SKP_SMMUL( tmp_QA, rc_mult2_Q16 ), 16 - he
adrm );
        }
    }

    /* Check for stability */
    if( ( Anew_QA[ 0 ] > A_LIMIT ) || ( Anew_QA[ 0 ] < -A_LIMIT ) ) {
        return 1;
    }

    /* Set RC equal to negated AR coef */
    rc_Q31 = -SKP_LSHIFT( Anew_QA[ 0 ], 31 - QA );

    /* Range: [ 1 : 2^30 ] */
    rc_mult1_Q30 = ( SKP_int32_MAX >> 1 ) - SKP_SMMUL( rc_Q31, rc_Q31 );

    /* Update inverse gain */
    /* Range: [ 0 : 2^30 ] */
    *invGain_Q30 = SKP_LSHIFT( SKP_SMMUL( *invGain_Q30, rc_mult1_Q30 ), 2 );
    SKP_assert( *invGain_Q30 >= 0     );
    SKP_assert( *invGain_Q30 <= 1<<30 );

    return 0;
}

/* For input in Q13 domain */
SKP_int SKP_Silk_LPC_inverse_pred_gain_Q13(   /* O:   Returns 1 if unstable, othe
rwise 0          */
    SKP_int32           *invGain_Q30,           /* O:   Inverse prediction gain,
Q30 energy domain  */
    const SKP_int16     *A_Q13,                 /* I:   Prediction coefficients,
Q13 [order]        */
    const SKP_int       order                   /* I:   Prediction order
                */
)
{
    SKP_int   k, n, headrm;
    SKP_int32 rc_Q31, rc_mult1_Q30, rc_mult2_Q16;
    SKP_int32 Atmp_QA[ 2 ][ SigProc_MAX_ORDER_LPC ], tmp_QA;
    SKP_int32 *Aold_QA, *Anew_QA;

    Anew_QA = Atmp_QA[ order & 1 ];
    /* Increase Q domain of the AR coefficients */
    for( k = 0; k < order; k++ ) {
        Anew_QA[ k ] = SKP_LSHIFT( (SKP_int32)A_Q13[ k ], QA - 13 );
    }

    *invGain_Q30 = ( 1 << 30 );
    for( k = order - 1; k > 0; k-- ) {
        /* Check for stability */
        if( ( Anew_QA[ k ] > A_LIMIT ) || ( Anew_QA[ k ] < -A_LIMIT ) ) {
```

```
            return 1;
        }

        /* Set RC equal to negated AR coef */
        rc_Q31 = -SKP_LSHIFT( Anew_QA[ k ], 31 - QA );

        /* rc_mult1_Q30 range: [ 1 : 2^30-1 ] */
        rc_mult1_Q30 = ( SKP_int32_MAX >> 1 ) - SKP_SMMUL( rc_Q31, rc_Q31 );
        SKP_assert( rc_mult1_Q30 > ( 1 << 15 ) );                    /* reduce A_L
IMIT if fails */
        SKP_assert( rc_mult1_Q30 < ( 1 << 30 ) );

        /* rc_mult2_Q16 range: [ 2^16 : SKP_int32_MAX ] */
        rc_mult2_Q16 = SKP_INVERSE32_varQ( rc_mult1_Q30, 46 );      /* 16 = 46 -
30 */

        /* Update inverse gain */
        /* invGain_Q30 range: [ 0 : 2^30 ] */
        *invGain_Q30 = SKP_LSHIFT( SKP_SMMUL( *invGain_Q30, rc_mult1_Q30 ), 2 );
        SKP_assert( *invGain_Q30 >= 0     );
        SKP_assert( *invGain_Q30 <= 1<<30 );

        /* Swap pointers */
        Aold_QA = Anew_QA;
        Anew_QA = Atmp_QA[ k & 1 ];

        /* Update AR coefficient */
        headrm = SKP_Silk_CLZ32( rc_mult2_Q16 ) - 1;
        rc_mult2_Q16 = SKP_LSHIFT( rc_mult2_Q16, headrm );          /* Q: 16 + he
adrm */
        for( n = 0; n < k; n++ ) {
            tmp_QA = Aold_QA[ n ] - SKP_LSHIFT( SKP_SMMUL( Aold_QA[ k - n - 1 ],
rc_Q31 ), 1 );
            Anew_QA[ n ] = SKP_LSHIFT( SKP_SMMUL( tmp_QA, rc_mult2_Q16 ), 16 - he
adrm );
        }
    }

    /* Check for stability */
    if( ( Anew_QA[ 0 ] > A_LIMIT ) || ( Anew_QA[ 0 ] < -A_LIMIT ) ) {
        return 1;
    }

    /* Set RC equal to negated AR coef */
    rc_Q31 = -SKP_LSHIFT( Anew_QA[ 0 ], 31 - QA );

    /* Range: [ 1 : 2^30 ] */
    rc_mult1_Q30 = ( SKP_int32_MAX >> 1 ) - SKP_SMMUL( rc_Q31, rc_Q31 );

    /* Update inverse gain */
    /* Range: [ 0 : 2^30 ] */
    *invGain_Q30 = SKP_LSHIFT( SKP_SMMUL( *invGain_Q30, rc_mult1_Q30 ), 2 );
    SKP_assert( *invGain_Q30 >= 0     );
```

```
    SKP_assert( *invGain_Q30 <= 1<<30 );

    return 0;
}
```

A.58. src/SKP_Silk_LPC_stabilize.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_typedef.h"
#include "SKP_Silk_SigProc_FIX.h"

#define LPC_STABILIZE_LPC_MAX_ABS_VALUE_Q16     ( ( (SKP_int32)SKP_int16_MAX ) <<
 4 )

/* LPC stabilizer, for a single input data vector */
void SKP_Silk_LPC_stabilize(
    SKP_int16       *a_Q12,         /* O    stabilized LPC vector [L]
          */
    SKP_int32       *a_Q16,         /* I    LPC vector [L]
          */
    const SKP_int32  bwe_Q16,       /* I    Bandwidth expansion factor
          */
    const SKP_int    L              /* I    Number of LPC parameters in the input
 vector    */
)
```

```
{
    SKP_int32   maxabs, absval, sc_Q16;
    SKP_int     i, idx = 0;
    SKP_int32   invGain_Q30;

    SKP_Silk_bwexpander_32( a_Q16, L, bwe_Q16 );

    /*************************/
    /* Limit range of the LPCs */
    /*************************/
    /* Limit the maximum absolute value of the prediction coefficients */
    while( SKP_TRUE ) {
        /* Find maximum absolute value and its index */
        maxabs = SKP_int32_MIN;
        for( i = 0; i < L; i++ ) {
            absval = SKP_abs( a_Q16[ i ] );
            if( absval > maxabs ) {
                maxabs = absval;
                idx    = i;
            }
        }

        if( maxabs >= LPC_STABILIZE_LPC_MAX_ABS_VALUE_Q16 ) {
            /* Reduce magnitude of prediction coefficients */
            sc_Q16 = SKP_DIV32( SKP_int32_MAX, SKP_RSHIFT( maxabs, 4 ) );
            sc_Q16 = 65536 - sc_Q16;
            sc_Q16 = SKP_DIV32( sc_Q16, idx + 1 );
            sc_Q16 = 65536 - sc_Q16;
            sc_Q16 = SKP_LSHIFT( SKP_SMULWB( sc_Q16, 32604 ), 1 ); // 0.995 in Q1
6
            SKP_Silk_bwexpander_32( a_Q16, L, sc_Q16 );
        } else {
            break;
        }
    }

    /* Convert to 16 bit Q12 */
    for( i = 0; i < L; i++ ) {
        a_Q12[ i ] = (SKP_int16)SKP_RSHIFT_ROUND( a_Q16[ i ], 4 );
    }

    /***********************/
    /* Ensure stable LPCs */
    /***********************/
    while( SKP_Silk_LPC_inverse_pred_gain( &invGain_Q30, a_Q12, L ) == 1 ) {
        SKP_Silk_bwexpander( a_Q12, L, 65339 ); // 0.997 in Q16
    }
}
```

```
void SKP_Silk_LPC_fit(
    SKP_int16       *a_QQ,          /* O    Stabilized LPC vector, Q(24-rshift) [
L]          */
    SKP_int32       *a_Q24,         /* I    LPC vector [L]
            */
    const SKP_int    QQ,            /* I    Q domain of output LPC vector
            */
    const SKP_int    L              /* I    Number of LPC parameters in the input
 vector    */
)
{
    SKP_int     i, rshift, idx = 0;
    SKP_int32   maxabs, absval, sc_Q16;

    rshift = 24 - QQ;

    /**************************/
    /* Limit range of the LPCs */
    /**************************/
    /* Limit the maximum absolute value of the prediction coefficients */
    while( SKP_TRUE ) {
        /* Find maximum absolute value and its index */
        maxabs = SKP_int32_MIN;
        for( i = 0; i < L; i++ ) {
            absval = SKP_abs( a_Q24[ i ] );
            if( absval > maxabs ) {
                maxabs = absval;
                idx    = i;
            }
        }

        maxabs = SKP_RSHIFT( maxabs, rshift );
        if( maxabs >= SKP_int16_MAX ) {
            /* Reduce magnitude of prediction coefficients */
            sc_Q16 = 65470 - SKP_DIV32( SKP_MUL( 65470 >> 2, maxabs - SKP_int16_M
AX ),
                                        SKP_RSHIFT32( SKP_MUL( maxabs, idx + 1),
2 ) );
            SKP_Silk_bwexpander_32( a_Q24, L, sc_Q16 );
        } else {
            break;
        }
    }

    /* Convert to 16 bit Q(24-rshift) */
    SKP_assert( rshift > 0  );
    SKP_assert( rshift < 31 );
    for( i = 0; i < L; i++ ) {
        a_QQ[ i ] = (SKP_int16)SKP_RSHIFT_ROUND( a_Q24[ i ], rshift );
    }
}
```

A.59. src/SKP_Silk_LPC_synthesis_filter.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_Silk_LPC_synthesis_filter.c                                      *
 * Coefficients are in Q12                                              *
 *                                                                      *
 * even order AR filter                                                 *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"

/* even order AR filter */
void SKP_Silk_LPC_synthesis_filter(
    const SKP_int16 *in,        /* I:   excitation signal */
    const SKP_int16 *A_Q12,     /* I:   AR coefficients [Order], between -8_Q0 an
d 8_Q0 */
    const SKP_int32 Gain_Q26,   /* I:   gain */
    SKP_int32 *S,               /* I/O: state vector [Order] */
    SKP_int16 *out,             /* O:   output signal */
    const SKP_int32 len,        /* I:   signal length */
    const SKP_int Order         /* I:   filter order, must be even */
)
{
    SKP_int   k, j, idx, Order_half = SKP_RSHIFT( Order, 1 );
    SKP_int32 SA, SB, Atmp, A_align_Q12[SigProc_MAX_ORDER_LPC >> 1], out32_Q10, o
ut32;
```

```
    /* Order must be even */
    SKP_assert( 2*Order_half == Order );

    /* combine two A_Q12 values and ensure 32-bit alignment */
    for( k = 0; k < Order_half; k++ ) {
        idx = SKP_SMULBB( 2, k );
        A_align_Q12[k] = (((SKP_int32)A_Q12[idx]) & 0x0000ffff) | SKP_LSHIFT( (SK
P_int32)A_Q12[idx+1], 16 );
    }

    /* S[] values are in Q14 */
    for( k = 0; k < len; k++ ) {
        SA = S[Order-1];
        out32_Q10 = 0;
        for( j=0;j<(Order_half-1); j++ ) {
            idx = SKP_SMULBB( 2, j ) + 1;
            /* multiply-add two prediction coefficients for each loop */
            /* NOTE: the code below loads two int16 values in an int32, and multi
plies each using the   */
            /* SMLAWB and SMLAWT instructions. On a big-endian CPU the two int16
variables would be     */
            /* loaded in reverse order and the code will give the wrong result. I
n that case swapping   */
            /* the SMLAWB and SMLAWT instructions should solve the problem.
                      */
            Atmp = A_align_Q12[j];
            SB = S[Order - 1 - idx];
            S[Order - 1 - idx] = SA;
            out32_Q10 = SKP_SMLAWB( out32_Q10, SA, Atmp );
            out32_Q10 = SKP_SMLAWT( out32_Q10, SB, Atmp );
            SA = S[Order - 2 - idx];
            S[Order - 2 - idx] = SB;
        }

        /* unrolled loop: epilog */
        Atmp = A_align_Q12[Order_half-1];
        SB = S[0];
        S[0] = SA;
        out32_Q10 = SKP_SMLAWB( out32_Q10, SA, Atmp );
        out32_Q10 = SKP_SMLAWT( out32_Q10, SB, Atmp );

        /* apply gain to excitation signal and add to prediction */
        out32_Q10 = SKP_ADD_SAT32( out32_Q10, SKP_SMULWB( Gain_Q26, in[k] ) );

        /* scale to Q0 */
        out32 = SKP_RSHIFT_ROUND( out32_Q10, 10 );

        /* saturate output */
        out[k] = (SKP_int16)SKP_SAT16( out32 );

        /* move result into delay line */
        S[Order - 1] = SKP_LSHIFT_SAT32( out32_Q10, 4 );
    }
```

```
}
```

A.60. src/SKP_Silk_LPC_synthesis_order16.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_Silk_LPC_synthesis_order16.c                                     *
 * Coefficients are in Q12                                              *
 *                                                                      *
 * 16th order AR filter                                                 *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"

/* 16th order AR filter */
void SKP_Silk_LPC_synthesis_order16(const SKP_int16 *in,          /* I:   excitat
ion signal */
                                    const SKP_int16 *A_Q12,       /* I:   AR co
efficients [16], between -8_Q0 and 8_Q0 */
                                    const SKP_int32 Gain_Q26,     /* I:   gain
 */
                                    SKP_int32 *S,                 /* I/O: state
 vector [16] */
                                    SKP_int16 *out,               /* O:   outpu
t signal */
                                    const SKP_int32 len           /* I:   signa
l length, must be multiple of 16 */
```

```
)
{
    SKP_int   k;
    SKP_int32 SA, SB, Atmp, A_align_Q12[8], out32_Q10, out32;

    /* combine two A_Q12 values and ensure 32-bit alignment */
    for( k = 0; k < 8; k++ ) {
        A_align_Q12[k] = (((SKP_int32)A_Q12[ 2*k ]) & 0x0000ffff) | SKP_LSHIFT( (
SKP_int32)A_Q12[ 2*k + 1 ], 16 );
    }

    /* S[] values are in Q14 */
    /* NOTE: the code below loads two int16 values in an int32, and multiplies ea
ch using the   */
    /* SMLAWB and SMLAWT instructions. On a big-endian CPU the two int16 variable
s would be     */
    /* loaded in reverse order and the code will give the wrong result. In that c
ase swapping   */
    /* the SMLAWB and SMLAWT instructions should solve the problem.
               */
    for( k = 0; k < len; k++ ) {
        /* unrolled loop: prolog */
        /* multiply-add two prediction coefficients per iteration */
        SA = S[15];
        Atmp = A_align_Q12[0];
        SB = S[14];
        S[14] = SA;
        out32_Q10 = SKP_SMULWB(                   SA, Atmp );
        out32_Q10 = SKP_SMLAWT_ovflw( out32_Q10, SB, Atmp );
        SA = S[13];
        S[13] = SB;

        /* unrolled loop: main loop */
        Atmp = A_align_Q12[1];
        SB = S[12];
        S[12] = SA;
        out32_Q10 = SKP_SMLAWB_ovflw( out32_Q10, SA, Atmp );
        out32_Q10 = SKP_SMLAWT_ovflw( out32_Q10, SB, Atmp );
        SA = S[11];
        S[11] = SB;

        Atmp = A_align_Q12[2];
        SB = S[10];
        S[10] = SA;
        out32_Q10 = SKP_SMLAWB_ovflw( out32_Q10, SA, Atmp );
        out32_Q10 = SKP_SMLAWT_ovflw( out32_Q10, SB, Atmp );
        SA = S[9];
        S[9] = SB;

        Atmp = A_align_Q12[3];
        SB = S[8];
        S[8] = SA;
        out32_Q10 = SKP_SMLAWB_ovflw( out32_Q10, SA, Atmp );
```

```
        out32_Q10 = SKP_SMLAWT_ovflw( out32_Q10, SB, Atmp );
        SA = S[7];
        S[7] = SB;

        Atmp = A_align_Q12[4];
        SB = S[6];
        S[6] = SA;
        out32_Q10 = SKP_SMLAWB_ovflw( out32_Q10, SA, Atmp );
        out32_Q10 = SKP_SMLAWT_ovflw( out32_Q10, SB, Atmp );
        SA = S[5];
        S[5] = SB;

        Atmp = A_align_Q12[5];
        SB = S[4];
        S[4] = SA;
        out32_Q10 = SKP_SMLAWB_ovflw( out32_Q10, SA, Atmp );
        out32_Q10 = SKP_SMLAWT_ovflw( out32_Q10, SB, Atmp );
        SA = S[3];
        S[3] = SB;

        Atmp = A_align_Q12[6];
        SB = S[2];
        S[2] = SA;
        out32_Q10 = SKP_SMLAWB_ovflw( out32_Q10, SA, Atmp );
        out32_Q10 = SKP_SMLAWT_ovflw( out32_Q10, SB, Atmp );
        SA = S[1];
        S[1] = SB;

        /* unrolled loop: epilog */
        Atmp = A_align_Q12[7];
        SB = S[0];
        S[0] = SA;
        out32_Q10 = SKP_SMLAWB_ovflw( out32_Q10, SA, Atmp );
        out32_Q10 = SKP_SMLAWT_ovflw( out32_Q10, SB, Atmp );

        /* unrolled loop: end */
        /* apply gain to excitation signal and add to prediction */
        out32_Q10 = SKP_ADD_SAT32( out32_Q10, SKP_SMULWB( Gain_Q26, in[k] ) );

        /* scale to Q0 */
        out32 = SKP_RSHIFT_ROUND( out32_Q10, 10 );

        /* saturate output */
        out[k] = (SKP_int16)SKP_SAT16( out32 );

        /* move result into delay line */
        S[15] = SKP_LSHIFT_SAT32( out32_Q10, 4 );
    }
```

```
}
```

A.61. src/SKP_Silk_LSF_cos_table.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_SigProc_FIX.h"

// Q12 values (even)
const SKP_int SKP_Silk_LSFCosTab_FIX_Q12[LSF_COS_TAB_SZ_FIX + 1] = {
            8192,             8190,             8182,             8170,
            8152,             8130,             8104,             8072,
            8034,             7994,             7946,             7896,
            7840,             7778,             7714,             7644,
            7568,             7490,             7406,             7318,
            7226,             7128,             7026,             6922,
            6812,             6698,             6580,             6458,
            6332,             6204,             6070,             5934,
            5792,             5648,             5502,             5352,
```

```
            5198,           5040,           4880,           4718,
            4552,           4382,           4212,           4038,
            3862,           3684,           3502,           3320,
            3136,           2948,           2760,           2570,
            2378,           2186,           1990,           1794,
            1598,           1400,           1202,           1002,
             802,            602,            402,            202,
               0,           -202,           -402,           -602,
            -802,          -1002,          -1202,          -1400,
           -1598,          -1794,          -1990,          -2186,
           -2378,          -2570,          -2760,          -2948,
           -3136,          -3320,          -3502,          -3684,
           -3862,          -4038,          -4212,          -4382,
           -4552,          -4718,          -4880,          -5040,
           -5198,          -5352,          -5502,          -5648,
           -5792,          -5934,          -6070,          -6204,
           -6332,          -6458,          -6580,          -6698,
           -6812,          -6922,          -7026,          -7128,
           -7226,          -7318,          -7406,          -7490,
           -7568,          -7644,          -7714,          -7778,
           -7840,          -7896,          -7946,          -7994,
           -8034,          -8072,          -8104,          -8130,
           -8152,          -8170,          -8182,          -8190,
           -8192
};
```

A.62. src/SKP_Silk_LTP_analysis_filter_FIX.c

```
/**********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
```

```
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

void SKP_Silk_LTP_analysis_filter_FIX(
    SKP_int16       *LTP_res,                           /* O:   LTP residual sign
al of length NB_SUBFR * ( pre_length + subfr_length )  */
    const SKP_int16 *x,                                 /* I:   Pointer to input
signal with at least max( pitchL ) preceeding samples  */
    const SKP_int16 LTPCoef_Q14[ LTP_ORDER * NB_SUBFR ],/* I:   LTP_ORDER LTP coe
fficients for each NB_SUBFR subframe                   */
    const SKP_int   pitchL[ NB_SUBFR ],                 /* I:   Pitch lag, one fo
r each subframe                                        */
    const SKP_int32 invGains_Qxx[ NB_SUBFR ],           /* I:   Inverse quantizat
ion gains, one for each subframe                       */
    const SKP_int   Qxx,                                /* I:   Inverse quantizat
ion gains Q domain                                     */
    const SKP_int   subfr_length,                       /* I:   Length of each su
bframe                                                 */
    const SKP_int   pre_length                          /* I:   Length of the pre
ceeding samples starting at &x[0] for each subframe    */
)
{
    const SKP_int16 *x_ptr, *x_lag_ptr;
    SKP_int16   Btmp_Q14[ LTP_ORDER ];
    SKP_int16   *LTP_res_ptr;
    SKP_int     k, i, j;
    SKP_int32   LTP_est;

    x_ptr = x;
    LTP_res_ptr = LTP_res;
    for( k = 0; k < NB_SUBFR; k++ ) {

        x_lag_ptr = x_ptr - pitchL[ k ];
        for( i = 0; i < LTP_ORDER; i++ ) {
            Btmp_Q14[ i ] = LTPCoef_Q14[ k * LTP_ORDER + i ];
        }

        /* LTP analysis FIR filter */
        for( i = 0; i < subfr_length + pre_length; i++ ) {
            LTP_res_ptr[ i ] = x_ptr[ i ];

            /* Long-term prediction */
            LTP_est = SKP_SMULBB( x_lag_ptr[ LTP_ORDER / 2 ], Btmp_Q14[ 0 ] );
            for( j = 1; j < LTP_ORDER; j++ ) {
                LTP_est = SKP_SMLABB_ovflw( LTP_est, x_lag_ptr[ LTP_ORDER / 2 - j
 ], Btmp_Q14[ j ] );
            }
            LTP_est = SKP_RSHIFT_ROUND( LTP_est, 14 ); // round and -> Q0
```

```
            /* Subtract long-term prediction */
            LTP_res_ptr[ i ] = ( SKP_int16 )SKP_SAT16( ( SKP_int32 )x_ptr[ i ] -
LTP_est );

            /* Scale residual */
            if( Qxx == 16 ) {
                LTP_res_ptr[ i ] = SKP_SMULWB( invGains_Qxx[ k ], LTP_res_ptr[ i
] );
            } else {
                LTP_res_ptr[ i ] = ( SKP_int16 )SKP_CHECK_FIT16( SKP_RSHIFT64( SK
P_SMULL( invGains_Qxx[ k ], LTP_res_ptr[ i ] ), Qxx ) );
            }

            x_lag_ptr++;
        }

        /* Update pointers */
        LTP_res_ptr += subfr_length + pre_length;
        x_ptr       += subfr_length;
    }
}
```

A.63. src/SKP_Silk_LTP_scale_ctrl_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
```

```
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

#define NB_THRESHOLDS           11

/* Table containing trained thresholds for LTP scaling */
static const SKP_int16 LTPScaleThresholds_Q15[ NB_THRESHOLDS ] =
{
    31129, 26214, 16384, 13107, 9830, 6554,
     4915,  3276,  2621,  2458,    0
};

void SKP_Silk_LTP_scale_ctrl_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,     /* I/O  encoder state FIX
                    */
    SKP_Silk_encoder_control_FIX    *psEncCtrl  /* I/O  encoder control FIX
                    */
)
{
    SKP_int round_loss, frames_per_packet;
    SKP_int g_out_Q5, g_limit_Q15, thrld1_Q15, thrld2_Q15;

    /* 1st order high-pass filter */
    psEnc->HPLTPredCodGain_Q7 = SKP_max_int( psEncCtrl->LTPredCodGain_Q7 - psEnc-
>prevLTPredCodGain_Q7, 0 )
        + SKP_RSHIFT_ROUND( psEnc->HPLTPredCodGain_Q7, 1 );

    psEnc->prevLTPredCodGain_Q7 = psEncCtrl->LTPredCodGain_Q7;

    /* combine input and filtered input */
    g_out_Q5    = SKP_RSHIFT_ROUND( SKP_RSHIFT( psEncCtrl->LTPredCodGain_Q7, 1 )
+ SKP_RSHIFT( psEnc->HPLTPredCodGain_Q7, 1 ), 3 );
    g_limit_Q15 = SKP_Silk_sigm_Q15( g_out_Q5 - ( 3 << 5 ) ); /* mulitplid with 0
.5 */

    /* Default is minimum scaling */
    psEncCtrl->sCmn.LTP_scaleIndex = 0;

    /* Round the loss measure to whole pct */
    round_loss = ( SKP_int )psEnc->sCmn.PacketLoss_perc;

    /* Only scale if first frame in packet 0% */
    if( psEnc->sCmn.nFramesInPayloadBuf == 0 ) {

        frames_per_packet = SKP_DIV32_16( psEnc->sCmn.PacketSize_ms, FRAME_LENGTH
_MS );

        round_loss += frames_per_packet - 1;
        thrld1_Q15 = LTPScaleThresholds_Q15[ SKP_min_int( round_loss,     NB_THRE
SHOLDS - 1 ) ];
        thrld2_Q15 = LTPScaleThresholds_Q15[ SKP_min_int( round_loss + 1, NB_THRE
SHOLDS - 1 ) ];

        if( g_limit_Q15 > thrld1_Q15 ) {
```

```
            /* Maximum scaling */
            psEncCtrl->sCmn.LTP_scaleIndex = 2;
        } else if( g_limit_Q15 > thrld2_Q15 ) {
            /* Medium scaling */
            psEncCtrl->sCmn.LTP_scaleIndex = 1;
        }
    }
    psEncCtrl->LTP_scale_Q14 = SKP_Silk_LTPScales_table_Q14[ psEncCtrl->sCmn.LTP_
scaleIndex ];
}
```

A.64. src/SKP_Silk_MA.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_MA.c                                                     *
 *                                                                     *
 * Variable order MA filter                                            *
 *                                                                     *
 * Copyright 2006 (c), Skype Limited                                   *
 * Date: 060221                                                        *
```

```
 *                                                                   */
#include "SKP_Silk_SigProc_FIX.h"

/* Variable order MA filter */
void SKP_Silk_MA(
    const SKP_int16      *in,            /* I:   input signal
            */
    const SKP_int16      *B,             /* I:   MA coefficients, Q13 [order+1]
            */
    SKP_int32            *S,             /* I/O: state vector [order]
            */
    SKP_int16            *out,           /* O:   output signal
            */
    const SKP_int32      len,            /* I:   signal length
            */
    const SKP_int32      order           /* I:   filter order
            */
)
{
    SKP_int   k, d, in16;
    SKP_int32 out32;

    for( k = 0; k < len; k++ ) {
        in16 = in[ k ];
        out32 = SKP_SMLABB( S[ 0 ], in16, B[ 0 ] );
        out32 = SKP_RSHIFT_ROUND( out32, 13 );

        for( d = 1; d < order; d++ ) {
            S[ d - 1 ] = SKP_SMLABB( S[ d ], in16, B[ d ] );
        }
        S[ order - 1 ] = SKP_SMULBB( in16, B[ order ] );

        /* Limit */
        out[ k ] = (SKP_int16)SKP_SAT16( out32 );
    }
}
/* Variable order MA prediction error filter */
void SKP_Silk_MA_Prediction(
    const SKP_int16      *in,            /* I:   Input signal
            */
    const SKP_int16      *B,             /* I:   MA prediction coefficients, Q12
[order]     */
    SKP_int32            *S,             /* I/O: State vector [order]
            */
    SKP_int16            *out,           /* O:   Output signal
            */
    const SKP_int32      len,            /* I:   Signal length
            */
    const SKP_int32      order           /* I:   Filter order
            */
)
{
    SKP_int   k, d, in16;
    SKP_int32 out32;
    SKP_int32 B32;

    if( ( order & 1 ) == 0 && (SKP_int32)( (SKP_int_ptr_size)B & 3 ) == 0 ) {
        /* Even order and 4-byte aligned coefficient array */

        /* NOTE: the code below loads two int16 values in an int32, and multiplie
s each using the    */
```

```
        /* SMLABB and SMLABT instructions. On a big-endian CPU the two int16 vari
ables would be      */
        /* loaded in reverse order and the code will give the wrong result. In th
at case swapping    */
        /* the SMLABB and SMLABT instructions should solve the problem.
                    */
        for( k = 0; k < len; k++ ) {
            in16 = in[ k ];
            out32 = SKP_LSHIFT( in16, 12 ) - S[ 0 ];
            out32 = SKP_RSHIFT_ROUND( out32, 12 );

            for( d = 0; d < order - 2; d += 2 ) {
                B32 = *( (SKP_int32*)&B[ d ] );              /* read two coeffi
cients at once */
                S[ d ]     = SKP_SMLABB_ovflw( S[ d + 1 ], in16, B32 );
                S[ d + 1 ] = SKP_SMLABT_ovflw( S[ d + 2 ], in16, B32 );
            }
            B32 = *( (SKP_int32*)&B[ d ] );                  /* read two coeffi
cients at once */
            S[ order - 2 ] = SKP_SMLABB_ovflw( S[ order - 1 ], in16, B32 );
            S[ order - 1 ] = SKP_SMULBT( in16, B32 );

            /* Limit */
            out[ k ] = (SKP_int16)SKP_SAT16( out32 );
        }
    } else {
        /* Odd order or not 4-byte aligned coefficient array */
        for( k = 0; k < len; k++ ) {
            in16 = in[ k ];
            out32 = SKP_LSHIFT( in16, 12 ) - S[ 0 ];
            out32 = SKP_RSHIFT_ROUND( out32, 12 );

            for( d = 0; d < order - 1; d++ ) {
                S[ d ] = SKP_SMLABB_ovflw( S[ d + 1 ], in16, B[ d ] );
            }
            S[ order - 1 ] = SKP_SMULBB( in16, B[ order - 1 ] );

            /* Limit */
            out[ k ] = (SKP_int16)SKP_SAT16( out32 );
        }
    }
}

void SKP_Silk_MA_Prediction_Q13(
    const SKP_int16      *in,            /* I:   input signal
            */
    const SKP_int16      *B,             /* I:   MA prediction coefficients, Q13
[order]     */
    SKP_int32            *S,             /* I/O: state vector [order]
            */
    SKP_int16            *out,           /* O:   output signal
            */
    SKP_int32            len,            /* I:   signal length
            */
    SKP_int32            order           /* I:   filter order
            */
)
{
    SKP_int   k, d, in16;
```

```
    SKP_int32 out32, B32;

    if( ( order & 1 ) == 0 && (SKP_int32)( (SKP_int_ptr_size)B & 3 ) == 0 ) {
        /* Even order and 4-byte aligned coefficient array */

        /* NOTE: the code below loads two int16 values in an int32, and multiplie
s each using the    */
        /* SMLABB and SMLABT instructions. On a big-endian CPU the two int16 vari
ables would be      */
        /* loaded in reverse order and the code will give the wrong result. In th
at case swapping    */
        /* the SMLABB and SMLABT instructions should solve the problem.
                    */
        for( k = 0; k < len; k++ ) {
            in16 = in[ k ];
            out32 = SKP_LSHIFT( in16, 13 ) - S[ 0 ];
            out32 = SKP_RSHIFT_ROUND( out32, 13 );

            for( d = 0; d < order - 2; d += 2 ) {
                B32 = *( (SKP_int32*)&B[ d ] );               /* read two coeffi
cients at once */
                S[ d ]     = SKP_SMLABB( S[ d + 1 ], in16, B32 );
                S[ d + 1 ] = SKP_SMLABT( S[ d + 2 ], in16, B32 );
            }
            B32 = *( (SKP_int32*)&B[ d ] );                   /* read two coeffi
cients at once */
            S[ order - 2 ] = SKP_SMLABB( S[ order - 1 ], in16, B32 );
            S[ order - 1 ] = SKP_SMULBT( in16, B32 );

            /* Limit */
            out[ k ] = (SKP_int16)SKP_SAT16( out32 );
        }
    } else {
        /* Odd order or not 4-byte aligned coefficient array */
        for( k = 0; k < len; k++ ) {
            in16 = in[ k ];
            out32 = SKP_LSHIFT( in16, 13 ) - S[ 0 ];
            out32 = SKP_RSHIFT_ROUND( out32, 13 );

            for( d = 0; d < order - 1; d++ ) {
                S[ d ] = SKP_SMLABB( S[ d + 1 ], in16, B[ d ] );
            }
            S[ order - 1 ] = SKP_SMULBB( in16, B[ order - 1 ] );

            /* Limit */
            out[ k ] = (SKP_int16)SKP_SAT16( out32 );
        }
    }
}
/* Variable order MA prediction error filter. */
/* Inverse filter of SKP_Silk_LPC_synthesis_filter */
void SKP_Silk_LPC_analysis_filter(
    const SKP_int16      *in,            /* I:   Input signal
            */
    const SKP_int16      *B,             /* I:   MA prediction coefficients, Q12
[order]     */
```

```
    SKP_int16           *S,             /* I/O: State vector [order]
            */
    SKP_int16           *out,           /* O:   Output signal
            */
    const SKP_int32     len,            /* I:   Signal length
            */
    const SKP_int32     Order           /* I:   Filter order
            */
)
{
    SKP_int   k, j, idx, Order_half = SKP_RSHIFT( Order, 1 );
    SKP_int32 Btmp, B_align_Q12[ SigProc_MAX_ORDER_LPC >> 1 ], out32_Q12, out32;
    SKP_int16 SA, SB;
    /* Order must be even */
    SKP_assert( 2 * Order_half == Order );

    /* Combine two A_Q12 values and ensure 32-bit alignment */
    for( k = 0; k < Order_half; k++ ) {
        idx = SKP_SMULBB( 2, k );
        B_align_Q12[ k ] = ( ( (SKP_int32)B[ idx ] ) & 0x0000ffff ) | SKP_LSHIFT(
 (SKP_int32)B[ idx + 1 ], 16 );
    }

    /* S[] values are in Q0 */
    for( k = 0; k < len; k++ ) {
        SA = S[ 0 ];
        out32_Q12 = 0;
        for( j = 0; j < ( Order_half - 1 ); j++ ) {
            idx = SKP_SMULBB( 2, j ) + 1;
            /* Multiply-add two prediction coefficients for each loop */
            Btmp = B_align_Q12[ j ];
            SB = S[ idx ];
            S[ idx ] = SA;
            out32_Q12 = SKP_SMLABB( out32_Q12, SA, Btmp );
            out32_Q12 = SKP_SMLABT( out32_Q12, SB, Btmp );
            SA = S[ idx + 1 ];
            S[ idx + 1 ] = SB;
        }

        /* Unrolled loop: epilog */
        Btmp = B_align_Q12[ Order_half - 1 ];
        SB = S[ Order - 1 ];
        S[ Order - 1 ] = SA;
        out32_Q12 = SKP_SMLABB( out32_Q12, SA, Btmp );
        out32_Q12 = SKP_SMLABT( out32_Q12, SB, Btmp );

        /* Subtract prediction */
        out32_Q12 = SKP_SUB_SAT32( SKP_LSHIFT( (SKP_int32)in[ k ], 12 ), out32_Q1
2 );

        /* Scale to Q0 */
        out32 = SKP_RSHIFT_ROUND( out32_Q12, 12 );

        /* Saturate output */
```

```
        out[ k ] = (SKP_int16)SKP_SAT16( out32 );

        /* Move input line */
        S[ 0 ] = in[ k ];
    }
}
```

A.65. src/SKP_Silk_macros.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef _SKP_SILK_API_C_H_
#define _SKP_SILK_API_C_H_

// This is an inline header file for general platform.

// (a32 * (SKP_int32)((SKP_int16)(b32))) >> 16 output have to be 32bit int
#define SKP_SMULWB(a32, b32)            ((((a32) >> 16) * (SKP_int32)((SKP_int16)
(b32))) + ((((a32) & 0x0000FFFF) * (SKP_int32)((SKP_int16)(b32))) >> 16))

// a32 + (b32 * (SKP_int32)((SKP_int16)(c32))) >> 16 output have to be 32bit int
#define SKP_SMLAWB(a32, b32, c32)       ((a32) + ((((b32) >> 16) * (SKP_int32)((S
KP_int16)(c32))) + ((((b32) & 0x0000FFFF) * (SKP_int32)((SKP_int16)(c32))) >> 16)
))
```

```
// (a32 * (b32 >> 16)) >> 16
#define SKP_SMULWT(a32, b32)            (((a32) >> 16) * ((b32) >> 16) + ((((a32)
 & 0x0000FFFF) * ((b32) >> 16)) >> 16))

// a32 + (b32 * (c32 >> 16)) >> 16
#define SKP_SMLAWT(a32, b32, c32)       ((a32) + (((b32) >> 16) * ((c32) >> 16))
+ ((((b32) & 0x0000FFFF) * ((c32) >> 16)) >> 16))

// (SKP_int32)((SKP_int16)(a3))) * (SKP_int32)((SKP_int16)(b32)) output have to b
e 32bit int
#define SKP_SMULBB(a32, b32)            ((SKP_int32)((SKP_int16)(a32)) * (SKP_int
32)((SKP_int16)(b32)))

// a32 + (SKP_int32)((SKP_int16)(b32)) * (SKP_int32)((SKP_int16)(c32)) output hav
e to be 32bit int
#define SKP_SMLABB(a32, b32, c32)       ((a32) + ((SKP_int32)((SKP_int16)(b32)))
* (SKP_int32)((SKP_int16)(c32)))

// (SKP_int32)((SKP_int16)(a32)) * (b32 >> 16)
#define SKP_SMULBT(a32, b32)            ((SKP_int32)((SKP_int16)(a32)) * ((b32) >
> 16))

// a32 + (SKP_int32)((SKP_int16)(b32)) * (c32 >> 16)
#define SKP_SMLABT(a32, b32, c32)       ((a32) + ((SKP_int32)((SKP_int16)(b32)))
* ((c32) >> 16))

// a64 + (b32 * c32)
#define SKP_SMLAL(a64, b32, c32)        (SKP_ADD64((a64), ((SKP_int64)(b32) * (SK
P_int64)(c32))))

// (a32 * b32) >> 16
#define SKP_SMULWW(a32, b32)            SKP_MLA(SKP_SMULWB((a32), (b32)), (a32),
SKP_RSHIFT_ROUND((b32), 16))

// a32 + ((b32 * c32) >> 16)
#define SKP_SMLAWW(a32, b32, c32)       SKP_MLA(SKP_SMLAWB((a32), (b32), (c32)),
(b32), SKP_RSHIFT_ROUND((c32), 16))

/* add/subtract with output saturated */
#define SKP_ADD_SAT32(a, b)             ((((a) + (b)) & 0x80000000) == 0 ?
                        \
                                        ((((a) & (b)) & 0x80000000) != 0 ? SKP_in
t32_MIN : (a)+(b)) :   \
                                        ((((a) | (b)) & 0x80000000) == 0 ? SKP_in
t32_MAX : (a)+(b)) )

#define SKP_SUB_SAT32(a, b)             ((((a)-(b)) & 0x80000000) == 0 ?
                                \
                                        (( (a) & ((b)^0x80000000) & 0x80000000) ?
 SKP_int32_MIN : (a)-(b)) :    \
                                        ((((a)^0x80000000) & (b)  & 0x80000000) ?
 SKP_int32_MAX : (a)-(b)) )

SKP_INLINE SKP_int32 SKP_Silk_CLZ16(SKP_int16 in16)
{
    SKP_int32 out32 = 0;
    if( in16 == 0 ) {
        return 16;
    }
    /* test nibbles */
    if( in16 & 0xFF00 ) {
        if( in16 & 0xF000 ) {
```

```
            in16 >>= 12;
        } else {
            out32 += 4;
```

```
            in16 >>= 8;
        }
    } else {
        if( in16 & 0xFFF0 ) {
            out32 += 8;
            in16 >>= 4;
        } else {
            out32 += 12;
        }
    }
    /* test bits and return */
    if( in16 & 0xC ) {
        if( in16 & 0x8 )
            return out32 + 0;
        else
            return out32 + 1;
    } else {
        if( in16 & 0xE )
            return out32 + 2;
        else
            return out32 + 3;
    }
}

SKP_INLINE SKP_int32 SKP_Silk_CLZ32(SKP_int32 in32)
{
    /* test highest 16 bits and convert to SKP_int16 */
    if( in32 & 0xFFFF0000 ) {
        return SKP_Silk_CLZ16((SKP_int16)(in32 >> 16));
    } else {
        return SKP_Silk_CLZ16((SKP_int16)in32) + 16;
    }
}

#endif //_SKP_SILK_API_C_H_
```

A.66. src/SKP_Silk_main.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
```

```
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_MAIN_H
#define SKP_SILK_MAIN_H

#include "SKP_Silk_SigProc_FIX.h"
#include "SKP_Silk_define.h"
#include "SKP_Silk_structs.h"
#include "SKP_Silk_tables.h"
#include "SKP_Silk_PLC.h"

#ifdef __cplusplus
extern "C"
{
#endif

/* Encodes signs of excitation */
void SKP_Silk_encode_signs(
    SKP_Silk_range_coder_state  *psRC,              /* I/O  Range coder state
                        */
    const SKP_int               q[],                /* I    pulse signal
                        */
    const SKP_int               length,             /* I    length of input
                        */
    const SKP_int               sigtype,            /* I    Signal type
                        */
    const SKP_int               QuantOffsetType,    /* I    Quantization offset t
ype                     */
    const SKP_int               RateLevelIndex      /* I    Rate Level Index
                        */
);

/* Decodes signs of excitation */
void SKP_Silk_decode_signs(
    SKP_Silk_range_coder_state  *psRC,              /* I/O  Range coder state
                        */
    SKP_int                     q[],                /* I/O  pulse signal
                        */
```

```
    const SKP_int               length,             /* I    length of output
                        */
    const SKP_int               sigtype,            /* I    Signal type
                        */
    const SKP_int               QuantOffsetType,    /* I    Quantization offset t
ype                     */
    const SKP_int               RateLevelIndex      /* I    Rate Level Index
                        */
);

/***************/
/* Shell coder */
/***************/

/* Encode quantization indices of excitation */
void SKP_Silk_encode_pulses(
    SKP_Silk_range_coder_state  *psRC,              /* I/O  Range coder state
                        */
    const SKP_int               sigtype,            /* I    Sigtype
                        */
    const SKP_int               QuantOffsetType,    /* I    QuantOffsetType
                        */
    const SKP_int               q[],                /* I    quantization indices
                        */
    const SKP_int               frame_length        /* I    Frame length
                        */
);

/* Shell encoder, operates on one shell code frame of 16 pulses */
void SKP_Silk_shell_encoder(
    SKP_Silk_range_coder_state  *psRC,              /* I/O  compressor data struc
ture                    */
    const SKP_int               *pulses0            /* I    data: nonnegative pul
se amplitudes           */
);

/* Shell decoder, operates on one shell code frame of 16 pulses */
void SKP_Silk_shell_decoder(
    SKP_int                     *pulses0,           /* O    data: nonnegative pul
se amplitudes           */
    SKP_Silk_range_coder_state  *psRC,              /* I/O  compressor data struc
ture                    */
    const SKP_int               pulses4             /* I    number of pulses per
pulse-subframe          */
);

/***************/
/* Range coder */
/***************/
/* Range encoder for one symbol */
void SKP_Silk_range_encoder(
    SKP_Silk_range_coder_state  *psRC,              /* I/O  compressor data struc
ture                    */
    const SKP_int               data,               /* I    uncompressed data
                        */
    const SKP_uint16            prob[]              /* I    cumulative density fu
nctions                 */
);

/* Range encoder for multiple symbols */
void SKP_Silk_range_encoder_multi(
    SKP_Silk_range_coder_state  *psRC,              /* I/O  compressor data struc
```

```
ture                    */
    const SKP_int               data[],             /* I    uncompressed data
[nSymbols]              */
    const SKP_uint16 * const    prob[],             /* I    cumulative density fu
nctions                 */
    const SKP_int               nSymbols            /* I    number of data symbol
s                       */
```

```
);

/* Range decoder for one symbol */
void SKP_Silk_range_decoder(
    SKP_int                     data[],             /* O    uncompressed data
                    */
    SKP_Silk_range_coder_state  *psRC,              /* I/O  compressor data struc
ture                */
    const SKP_uint16            prob[],             /* I    cumulative density fu
nction              */
    SKP_int                     probIx              /* I    initial (middle) entr
y of cdf            */
);

/* Range decoder for multiple symbols */
void SKP_Silk_range_decoder_multi(
    SKP_int                     data[],             /* O    uncompressed data
             [nSymbols] */
    SKP_Silk_range_coder_state  *psRC,              /* I/O  compressor data struc
ture                */
    const SKP_uint16 * const    prob[],             /* I    cumulative density fu
nctions             */
    const SKP_int               probStartIx[],      /* I    initial (middle) entr
ies of cdfs [nSymbols] */
    const SKP_int               nSymbols            /* I    number of data symbol
s                   */
);

/* Initialize range coder structure for encoder */
void SKP_Silk_range_enc_init(
    SKP_Silk_range_coder_state  *psRC               /* O    compressor data struc
ture                */
);

/* Initialize range coder structure for decoder */
void SKP_Silk_range_dec_init(
    SKP_Silk_range_coder_state  *psRC,              /* O    compressor data struc
ture                */
    const SKP_uint8             buffer[],           /* I    buffer for compressed
 data [bufferLength]   */
    const SKP_int32             bufferLength        /* I    buffer length (in byt
es)                 */
);

/* Determine length of bitstream */
SKP_int SKP_Silk_range_coder_get_length(            /* O    returns number of BIT
S in stream         */
    const SKP_Silk_range_coder_state    *psRC,      /* I    compressed data struc
ture                */
    SKP_int                             *nBytes     /* O    number of BYTES in st
ream                */
);

/* Write decodable stream to buffer, and determine its length */
void SKP_Silk_range_enc_wrap_up(
    SKP_Silk_range_coder_state  *psRC               /* I/O  compressed data struc
ture                */
);

/* Check that any remaining bits in the last byte are set to 1 */
void SKP_Silk_range_coder_check_after_decoding(
    SKP_Silk_range_coder_state  *psRC               /* I/O  compressed data struc
```

```
ture                    */
);

/* Gain scalar quantization with hysteresis, uniform on log scale */
```

```
void SKP_Silk_gains_quant(
    SKP_int                     ind[ NB_SUBFR ],        /* O    gain indices
                        */
    SKP_int32                   gain_Q16[ NB_SUBFR ],   /* I/O  gains (quantized
out)                    */
    SKP_int                     *prev_ind,              /* I/O  last index in pre
vious frame             */
    const SKP_int               conditional             /* I    first gain is del
ta coded if 1           */
);

/* Gains scalar dequantization, uniform on log scale */
void SKP_Silk_gains_dequant(
    SKP_int32                   gain_Q16[ NB_SUBFR ],   /* O    quantized gains
                        */
    const SKP_int               ind[ NB_SUBFR ],        /* I    gain indices
                        */
    SKP_int                     *prev_ind,              /* I/O  last index in pre
vious frame             */
    const SKP_int               conditional             /* I    first gain is del
ta coded if 1           */
);

/* Convert NLSF parameters to stable AR prediction filter coefficients */
void SKP_Silk_NLSF2A_stable(
    SKP_int16                   pAR_Q12[ MAX_LPC_ORDER ],   /* O    Stabilized AR
 coefs [LPC_order]      */
    const SKP_int               pNLSF[ MAX_LPC_ORDER ],     /* I    NLSF vector
       [LPC_order]      */
    const SKP_int               LPC_order                   /* I    LPC/LSF order
                        */
);

/* Interpolate two vectors */
void SKP_Silk_interpolate(
    SKP_int                     xi[ MAX_LPC_ORDER ],    /* O    interpolated vect
or                      */
    const SKP_int               x0[ MAX_LPC_ORDER ],    /* I    first vector
                        */
    const SKP_int               x1[ MAX_LPC_ORDER ],    /* I    second vector
                        */
    const SKP_int               ifact_Q2,               /* I    interp. factor, w
eight on 2nd vector     */
    const SKP_int               d                       /* I    number of paramet
ers                     */
);

/***********************************/
/* Noise shaping quantization (NSQ)*/
/***********************************/
void SKP_Silk_NSQ(
    SKP_Silk_encoder_state          *psEncC,                                    /
* I/O  Encoder State                     */
    SKP_Silk_encoder_control        *psEncCtrlC,                                /
* I    Encoder Control                   */
    SKP_Silk_nsq_state              *NSQ,                                       /
* I/O  NSQ state                         */
    const SKP_int16                 x[],                                        /
* I    prefiltered input signal          */
    SKP_int                         q[],                                        /
* O    quantized qulse signal            */
    const SKP_int                   LSFInterpFactor_Q2,                         /
```

```
* I    LSF interpolation factor in Q2      */
    const SKP_int16                 PredCoef_Q12[ 2 * MAX_LPC_ORDER ],          /
* I    Short term prediction coefficients  */
    const SKP_int16                 LTPCoef_Q14[ LTP_ORDER * NB_SUBFR ],        /
* I    Long term prediction coefficients   */
    const SKP_int16                 AR2_Q13[ NB_SUBFR * SHAPE_LPC_ORDER_MAX ],  /
* I                                        */
    const SKP_int                   HarmShapeGain_Q14[ NB_SUBFR ],              /
* I                                        */
    const SKP_int                   Tilt_Q14[ NB_SUBFR ],                       /
* I    Spectral tilt                       */
    const SKP_int32                 LF_shp_Q14[ NB_SUBFR ],                     /
* I                                        */
    const SKP_int32                 Gains_Q16[ NB_SUBFR ],                      /
* I                                        */
```

```
    const SKP_int                   Lambda_Q10,                             /
* I                                      */
    const SKP_int                   LTP_scale_Q14                           /
* I    LTP state scaling                 */
);

/* Noise shaping using delayed decision */
void SKP_Silk_NSQ_del_dec(
    SKP_Silk_encoder_state          *psEncC,                                /
* I/O  Encoder State                     */
    SKP_Silk_encoder_control        *psEncCtrlC,                            /
* I    Encoder Control                   */
    SKP_Silk_nsq_state              *NSQ,                                   /
* I/O  NSQ state                         */
    const SKP_int16                 x[],                                    /
* I    Prefiltered input signal          */
    SKP_int                         q[],                                    /
* O    Quantized pulse signal            */
    const SKP_int                   LSFInterpFactor_Q2,                     /
* I    LSF interpolation factor in Q2    */
    const SKP_int16                 PredCoef_Q12[ 2 * MAX_LPC_ORDER ],      /
* I    Prediction coefs                  */
    const SKP_int16                 LTPCoef_Q14[ LTP_ORDER * NB_SUBFR ],    /
* I    LT prediction coefs               */
    const SKP_int16                 AR2_Q13[ NB_SUBFR * SHAPE_LPC_ORDER_MAX ],  /
* I                                      */
    const SKP_int                   HarmShapeGain_Q14[ NB_SUBFR ],          /
* I                                      */
    const SKP_int                   Tilt_Q14[ NB_SUBFR ],                   /
* I    Spectral tilt                     */
    const SKP_int32                 LF_shp_Q14[ NB_SUBFR ],                 /
* I                                      */
    const SKP_int32                 Gains_Q16[ NB_SUBFR ],                  /
* I                                      */
    const SKP_int                   Lambda_Q10,                             /
* I                                      */
    const SKP_int                   LTP_scale_Q14                           /
* I    LTP state scaling                 */
);

/*************/
/* Silk VAD */
/*************/
/* Initialize the Silk VAD */
SKP_int SKP_Silk_VAD_Init(                          /* O    Return value, 0 if su
ccess                   */
    SKP_Silk_VAD_state          *psSilk_VAD         /* I/O  Pointer to Silk VAD s
tate                    */
);

/* Silk VAD noise level estimation */
void SKP_Silk_VAD_GetNoiseLevels(
    const SKP_int32             pX[ VAD_N_BANDS ],  /* I    subband energies
                        */
    SKP_Silk_VAD_state          *psSilk_VAD         /* I/O  Pointer to Silk VAD s
tate                    */
);

/* Get speech activity level in Q8 */
SKP_int SKP_Silk_VAD_GetSA_Q8(                                  /* O    Return va
lue, 0 if success       */
    SKP_Silk_VAD_state          *psSilk_VAD,                    /* I/O  Silk VAD
```

```
state                   */
    SKP_int                     *pSA_Q8,                        /* O    Speech ac
tivity level in Q8      */
    SKP_int                     *pSNR_dB_Q7,                    /* O    SNR for c
urrent frame in Q7      */
    SKP_int                     pQuality_Q15[ VAD_N_BANDS ],    /* O    Smoothed
SNR for each band       */
    SKP_int                     *pTilt_Q15,                     /* O    current f
rame's frequency tilt   */
    const SKP_int16             pIn[],                          /* I    PCM input
       [framelength]    */
    const SKP_int               framelength                     /* I    Input fra
me length               */
);
```

```
/* Detect signal in 8 - 12 khz range */
void SKP_Silk_detect_SWB_input(
    SKP_Silk_detect_SWB_state   *psSWBdetect,       /* I/O  Encoder state
                        */
    const SKP_int16             samplesIn[],        /* I    Input to encoder
                        */
    SKP_int                     nSamplesIn          /* I    Length of input
                        */
);

#if SWITCH_TRANSITION_FILTERING
/* Low-pass filter with variable cutoff frequency based on  */
/* piece-wise linear interpolation between elliptic filters */
/* Start by setting transition_frame_no = 1;                */
void SKP_Silk_LP_variable_cutoff(
    SKP_Silk_LP_state           *psLP,              /* I/O  LP filter state
                        */
    SKP_int16                   *out,               /* O    Low-pass filtered out
put signal              */
    const SKP_int16             *in,                /* I    Input signal
                        */
    const SKP_int               frame_length        /* I    Frame length
                        */
);
#endif

/***************************************************/
/* Decoder Functions                               */
/***************************************************/
SKP_int SKP_Silk_create_decoder(
    SKP_Silk_decoder_state      **ppsDec            /* I/O  Decoder state pointer
 pointer                */
);

SKP_int SKP_Silk_free_decoder(
    SKP_Silk_decoder_state      *psDec              /* I/O  Decoder state pointer
                        */
);

SKP_int SKP_Silk_init_decoder(
    SKP_Silk_decoder_state      *psDec              /* I/O  Decoder state pointer
                        */
);

/* Set decoder sampling rate */
void SKP_Silk_decoder_set_fs(
    SKP_Silk_decoder_state      *psDec,             /* I/O  Decoder state pointer
                        */
    SKP_int                     fs_kHz              /* I    Sampling frequency (k
Hz)                     */
);

/****************/
/* Decode frame */
/****************/
SKP_int SKP_Silk_decode_frame(
    SKP_Silk_decoder_state      *psDec,             /* I/O  Pointer to Silk decod
er state                */
    SKP_int16                   pOut[],             /* O    Pointer to output spe
ech frame               */
    SKP_int16                   *pN,                /* O    Pointer to size of ou
tput frame              */
```

```
    const SKP_uint8             pCode[],            /* I    Pointer to payload
                        */
```

```
    const SKP_int               nBytes,             /* I    Payload length
                        */
    SKP_int                     action,             /* I    Action from Jitter Bu
ffer                    */
    SKP_int                     *decBytes           /* O    Used bytes to decode
this frame              */
);

/* Decode parameters from payload */
void SKP_Silk_decode_parameters(
    SKP_Silk_decoder_state      *psDec,             /* I/O  State
                        */
    SKP_Silk_decoder_control    *psDecCtrl,         /* I/O  Decoder control
                        */
    SKP_int                     q[],                /* O    Excitation signal
                        */
    const SKP_int               fullDecoding        /* I    Flag to tell if only
arithmetic decoding     */
);

/* Decode indices from payload v4 Bitstream */
void SKP_Silk_decode_indices_v4(
    SKP_Silk_decoder_state      *psDec              /* I/O  State
                        */
);

/* Decode parameters from payload v4 Bitstream */
void SKP_Silk_decode_parameters_v4(
    SKP_Silk_decoder_state      *psDec,                                 /* I/O  S
tate                                    */
    SKP_Silk_decoder_control    *psDecCtrl,                             /* I/O  D
ecoder control                          */
    SKP_int                     q[ MAX_FRAME_LENGTH ],                  /* O    E
xcitation signal                        */
    const SKP_int               fullDecoding                            /* I    F
lag to tell if only arithmetic decoding */
);

/* Core decoder. Performs inverse NSQ operation LTP + LPC */
void SKP_Silk_decode_core(
    SKP_Silk_decoder_state      *psDec,                             /* I/O  Decod
er state                */
    SKP_Silk_decoder_control    *psDecCtrl,                         /* I    Decod
er control              */
    SKP_int16                   xq[],                               /* O    Decod
ed speech               */
    const SKP_int               q[ MAX_FRAME_LENGTH ]               /* I    Pulse
 signal                 */
);

/* NLSF vector decoder */
void SKP_Silk_NLSF_MSVQ_decode(
    SKP_int                         *pNLSF_Q15,     /* O    Pointer to decoded ou
tput [LPC_ORDER x 1]    */
    const SKP_Silk_NLSF_CB_struct   *psNLSF_CB,     /* I    Pointer to NLSF codeb
ook struct              */
    const SKP_int                   *NLSFIndices,   /* I    Pointer to NLSF indic
es [nStages x 1]        */
    const SKP_int                   LPC_order       /* I    LPC order
                        */
);
```

```
/**********************/
/* Arithmetic coding */
/**********************/

/* Decode quantization indices of excitation (Shell coding) */
void SKP_Silk_decode_pulses(
```

```
    SKP_Silk_range_coder_state  *psRC,              /* I/O  Range coder state
                        */
    SKP_Silk_decoder_control    *psDecCtrl,         /* I/O  Decoder control
                        */
    SKP_int                     q[],                /* O    Excitation signal
                        */
    const SKP_int               frame_length        /* I    Frame length (prelimi
nary)                   */
);

/******************/
/* CNG */
/******************/

/* Reset CNG */
void SKP_Silk_CNG_Reset(
    SKP_Silk_decoder_state      *psDec              /* I/O  Decoder state
                        */
);

/* Updates CNG estimate, and applies the CNG when packet was lost   */
void SKP_Silk_CNG(
    SKP_Silk_decoder_state      *psDec,             /* I/O  Decoder state
                        */
    SKP_Silk_decoder_control    *psDecCtrl,         /* I/O  Decoder control
                        */
    SKP_int16                   signal[],           /* I/O  Signal
                        */
    SKP_int                     length              /* I    Length of residual
                        */
);

/* Encoding of various parameters */
void SKP_Silk_encode_parameters(
    SKP_Silk_encoder_state      *psEncC,            /* I/O  Encoder state
                        */
    SKP_Silk_encoder_control    *psEncCtrlC,        /* I/O  Encoder control
                        */
    SKP_Silk_range_coder_state  *psRC,              /* I/O  Range coder state
                        */
    const SKP_int               *q                  /* I    Quantization indices
                        */
);

/* Encoding of various parameters */
void SKP_Silk_encode_parameters_v4(
    SKP_Silk_encoder_state      *psEncC,            /* I/O  Encoder state
                        */
    SKP_Silk_encoder_control    *psEncCtrlC,        /* I/O  Encoder control
                        */
    SKP_Silk_range_coder_state  *psRC               /* I/O  Range encoder state
                        */
);

/* Extract lowest layer encoding */
void SKP_Silk_get_low_layer_internal(
    const SKP_uint8             *indata,            /* I:   Encoded input vector
                        */
    const SKP_int16             nBytesIn,           /* I:   Number of input Bytes
                        */
    SKP_uint8                   *Layer0data,        /* O:   Layer0 payload
                        */
```

```
    SKP_int16                   *nLayer0Bytes       /* O:   Number of FEC Bytes
                        */
);

/* Resets LBRR buffer, used if packet size changes */
void SKP_Silk_LBRR_reset(
```

```
    SKP_Silk_encoder_state      *psEncC             /* I/O  Pointer to Silk encod
er state               */
);

/* Predict number of bytes used to encode q */
SKP_int SKP_Silk_pulses_to_bytes( /* O  Return value, predicted number of bytes u
sed to encode q */
    SKP_Silk_encoder_state      *psEncC,            /* I/O  Encoder State*/
    SKP_int                     q[]                 /* I     Pulse signal */
);

#ifdef __cplusplus
}
#endif

#endif
```

A.67. src/SKP_Silk_main_FIX.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_MAIN_FIX_H
#define SKP_SILK_MAIN_FIX_H
```

```
#include <stdlib.h>
#include "SKP_Silk_SigProc_FIX.h"
#include "SKP_Silk_structs_FIX.h"
#include "SKP_Silk_main.h"
#include "SKP_Silk_define_FIX.h"
#include "SKP_Silk_PLC.h"
#define TIC(TAG_NAME)
#define TOC(TAG_NAME)

#ifndef FORCE_CPP_BUILD
#ifdef __cplusplus
extern "C"
{
#endif
#endif

/*********************/
/* Encoder Functions */
/*********************/

/* Initializes the Silk encoder state */
SKP_int SKP_Silk_init_encoder_FIX(
    SKP_Silk_encoder_state_FIX  *psEnc              /* I/O  Pointer to Silk FIX e
ncoder state           */
);

/* Control the Silk encoder */
SKP_int SKP_Silk_control_encoder_FIX(
    SKP_Silk_encoder_state_FIX  *psEnc,             /* I/O  Pointer to Silk FIX e
ncoder state                */
    const SKP_int               API_fs_kHz,         /* I    External (API) sampli
ng rate (kHz)               */
    const SKP_int               PacketSize_ms,      /* I    Packet length (ms)
                            */
    SKP_int32                   TargetRate_bps,     /* I    Target max bitrate (b
ps) (used if SNR_dB == 0)   */
    const SKP_int               PacketLoss_perc,    /* I    Packet loss rate (in
percent)                    */
    const SKP_int               INBandFec_enabled,  /* I    Enable (1) / disable
(0) inband FEC              */
    const SKP_int               DTX_enabled,        /* I    Enable / disable DTX
                            */
    const SKP_int               InputFramesize_ms,  /* I    Inputframe in ms
                            */
    const SKP_int               Complexity          /* I    Complexity (0->low; 1
->medium; 2->high)          */
);

/* Encoder main function */
SKP_int SKP_Silk_encode_frame_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,             /* I/O  Pointer to Silk F
IX encoder state            */
    SKP_uint8                       *pCode,             /* O    Pointer to payloa
d                           */
    SKP_int16                       *pnBytesOut,        /* I/O  Pointer to number
 of payload bytes;          */
                                                        /*      input: max length
; output: used              */
    const SKP_int16                 *pIn                /* I    Pointer to input
speech frame                */
);
```

```
/* Low BitRate Redundancy encoding functionality. Reuse all parameters but encode
 with lower bitrate */
```

```
void SKP_Silk_LBRR_encode_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  Pointer to Silk FIX e
ncoder state           */
    SKP_Silk_encoder_control_FIX    *psEncCtrl,     /* I/O  Pointer to Silk FIX e
ncoder control struct  */
    SKP_uint8                       *pCode,         /* O    Pointer to payload
                       */
    SKP_int16                       *pnBytesOut,    /* I/O  Pointer to number of
payload bytes          */
    SKP_int16                       xfw[]           /* I    Input signal
                       */
);

/* High-pass filter with cutoff frequency adaptation based on pitch lag statistic
s */
void SKP_Silk_HP_variable_cutoff_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  Encoder state
                       */
    SKP_Silk_encoder_control_FIX    *psEncCtrl,     /* I/O  Encoder control
                       */
    SKP_int16                       *out,           /* O    high-pass filtered ou
tput signal            */
    const SKP_int16                 *in             /* I    input signal
                       */
);

/****************/
/* Prefiltering */
/****************/
void SKP_Silk_prefilter_FIX(
    SKP_Silk_encoder_state_FIX          *psEnc,         /* I/O  Encoder state
                           */
    const SKP_Silk_encoder_control_FIX  *psEncCtrl,     /* I    Encoder control
                           */
    SKP_int16                           xw[],           /* O    Weighted signal
                           */
    const SKP_int16                     x[]             /* I    Speech signal
                           */
);

/*************************************************************/
/* Compute noise shaping coefficients and initial gain values */
/*************************************************************/
void SKP_Silk_noise_shape_analysis_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  Encoder state
                       */
    SKP_Silk_encoder_control_FIX    *psEncCtrl,     /* I/O  Encoder control
                       */
    const SKP_int16                 *pitch_res,     /* I    LPC residual from pit
ch analysis            */
    const SKP_int16                 *x              /* I    Input signal [ 2 * fr
ame_length + la_shape ]*/
);

/* Processing of gains */
void SKP_Silk_process_gains_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  Encoder state
                       */
    SKP_Silk_encoder_control_FIX    *psEncCtrl      /* I/O  Encoder control
                       */
);
```

```
/* Control low bitrate redundancy usage */
void SKP_Silk_LBRR_ctrl_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  encoder state
                        */
    SKP_Silk_encoder_control_FIX    *psEncCtrl      /* I/O  encoder control
                        */
);
```

```
/* Calculation of LTP state scaling */
void SKP_Silk_LTP_scale_ctrl_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  encoder state
                    */
    SKP_Silk_encoder_control_FIX    *psEncCtrl      /* I/O  encoder control
                    */
);

/*****************************************/
/* Prediction Analysis                   */
/*****************************************/

/* Find pitch lags */
void SKP_Silk_find_pitch_lags_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  encoder state
                    */
    SKP_Silk_encoder_control_FIX    *psEncCtrl,     /* I/O  encoder control
                    */
    SKP_int16                       res[],          /* O    residual
                    */
    const SKP_int16                 x[]             /* I    Speech signal
                    */
);

void SKP_Silk_find_pred_coefs_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  encoder state
                    */
    SKP_Silk_encoder_control_FIX    *psEncCtrl,     /* I/O  encoder control
                    */
    const SKP_int16                 res_pitch[]     /* I    Residual from pitch a
nalysis                */
);

void SKP_Silk_find_LPC_FIX(
    SKP_int             NLSF_Q15[],             /* O    LSFs
                                                   */
    SKP_int             *interpIndex,           /* O    LSF interpolation index,
only used for LSF interpolation                    */
    const SKP_int       prev_NLSFq_Q15[],       /* I    previous LSFs, only used
for LSF interpolation                              */
    const SKP_int       useInterpolatedLSFs,    /* I    Flag
                                                   */
    const SKP_int       LPC_order,              /* I    LPC order
                                                   */
    const SKP_int16     x[],                    /* I    Input signal
                                                   */
    const SKP_int       subfr_length            /* I    Input signal subframe len
gth including preceeding samples                   */
);

void SKP_Silk_LTP_analysis_filter_FIX(
    SKP_int16       *LTP_res,                           /* O:   LTP residual sign
al of length NB_SUBFR * ( pre_length + subfr_length )  */
    const SKP_int16 *x,                                 /* I:   Pointer to input
signal with at least max( pitchL ) preceeding samples  */
    const SKP_int16 LTPCoef_Q14[ LTP_ORDER * NB_SUBFR ],/* I:   LTP_ORDER LTP coe
fficients for each NB_SUBFR subframe                   */
    const SKP_int   pitchL[ NB_SUBFR ],                 /* I:   Pitch lag, one fo
r each subframe                                        */
    const SKP_int32 invGains_Qxx[ NB_SUBFR ],           /* I:   Inverse quantizat
ion gains, one for each subframe                       */
    const SKP_int   Qxx,                                /* I:   Inverse quantizat
```

```
ion gains Q domain                                          */
    const SKP_int   subfr_length,                           /* I:   Length of each su
bframe                                                      */
    const SKP_int   pre_length                              /* I:   Length of the pre
ceeding samples starting at &x[0] for each subframe         */
);

/* Finds LTP vector from correlations */
void SKP_Silk_find_LTP_FIX(
    SKP_int16           b_Q14[ NB_SUBFR * LTP_ORDER ],              /* O    LTP c
oefs                                                        */
```

```
    SKP_int32           WLTP[ NB_SUBFR * LTP_ORDER * LTP_ORDER ],   /* O    Weigh
t for LTP quantization                                              */
    SKP_int             *LTPredCodGain_Q7,                          /* O    LTP c
oding gain                                                          */
    const SKP_int16     r_first[],                                  /* I    resid
ual signal after LPC signal + state for first 10 ms    */
    const SKP_int16     r_last[],                                   /* I    resid
ual signal after LPC signal + state for last 10 ms     */
    const SKP_int       lag[ NB_SUBFR ],                            /* I    LTP l
ags                                                                 */
    const SKP_int32     Wght_Q15[ NB_SUBFR ],                       /* I    weigh
ts                                                                  */
    const SKP_int       subfr_length,                               /* I    subfr
ame length                                                          */
    const SKP_int       mem_offset,                                 /* I    numbe
r of samples in LTP memory                                          */
    SKP_int             corr_rshifts[ NB_SUBFR ]                    /* O    right
 shifts applied to correlations                                     */
);

/* LTP tap quantizer */
void SKP_Silk_quant_LTP_gains_FIX(
    SKP_int16               B_Q14[],                /* I/O  (un)quantized LTP gai
ns     */
    SKP_int                 cbk_index[],            /* O    Codebook Index
       */
    SKP_int                 *periodicity_index,     /* O    Periodicity Index
       */
    const SKP_int32         W_Q18[],                /* I    Error Weights in Q18
       */
    SKP_int                 mu_Q8,                  /* I    Mu value (R/D tradeof
f)     */
    SKP_int                 lowComplexity           /* I    Flag for low complexi
ty     */
);

/******************/
/* NLSF Quantizer */
/******************/

/* Limit, stabilize, convert and quantize NLSFs.    */
void SKP_Silk_process_NLSFs_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,     /* I/O  encoder state
                   */
    SKP_Silk_encoder_control_FIX    *psEncCtrl, /* I/O  encoder control
                   */
    SKP_int                         *pNLSF_Q15  /* I/O  Normalized LSFs (quant ou
t) (0 - (2^15-1))  */
);

/* LSF vector encoder */
void SKP_Silk_NLSF_MSVQ_encode_FIX(
          SKP_int                   *NLSFIndices,           /* O    Codebook path
 vector [ CB_STAGES ]      */
          SKP_int                   *pNLSF_Q15,             /* I/O  Quantized NLS
F vector [ LPC_ORDER ]     */
    const SKP_Silk_NLSF_CB_struct   *psNLSF_CB,             /* I    Codebook obje
ct                         */
    const SKP_int                   *pNLSF_q_Q15_prev,      /* I    Prev. quantiz
ed NLSF vector [LPC_ORDER] */
    const SKP_int                   *pW_Q6,                 /* I    NLSF weight v
ector [ LPC_ORDER ]        */
```

```
    const SKP_int                   NLSF_mu_Q15,            /* I    Rate weight f
or the RD optimization     */
    const SKP_int                   NLSF_mu_fluc_red_Q16,   /* I    Fluctuation r
eduction error weight      */
    const SKP_int                   NLSF_MSVQ_Survivors,    /* I    Max survivors
 from each stage           */
    const SKP_int                   LPC_order,              /* I    LPC order
                           */
    const SKP_int                   deactivate_fluc_red     /* I    Deactivate fl
uctuation reduction        */
);

/* Rate-Distortion calculations for multiple input data vectors */
void SKP_Silk_NLSF_VQ_rate_distortion_FIX(
```

```
    SKP_int32                       *pRD_Q20,           /* O    Rate-distortion v
alues [psNLSF_CBS->nVectors*N] */
    const SKP_Silk_NLSF_CBS         *psNLSF_CBS,        /* I    NLSF codebook sta
ge struct                       */
    const SKP_int                   *in_Q15,            /* I    Input vectors to
be quantized                    */
    const SKP_int                   *w_Q6,              /* I    Weight vector
                                */
    const SKP_int32                 *rate_acc_Q5,       /* I    Accumulated rates
 from previous stage            */
    const SKP_int                   mu_Q15,             /* I    Weight between we
ighted error and rate           */
    const SKP_int                   N,                  /* I    Number of input v
ectors to be quantized          */
    const SKP_int                   LPC_order           /* I    LPC order
                                */
);

/* Compute weighted quantization errors for an LPC_order element input vector, ov
er one codebook stage */
void SKP_Silk_NLSF_VQ_sum_error_FIX(
    SKP_int32                       *err_Q20,           /* O    Weighted quantiza
tion errors  [N*K]          */
    const SKP_int                   *in_Q15,            /* I    Input vectors to
be quantized [N*LPC_order] */
    const SKP_int                   *w_Q6,              /* I    Weighting vectors
             [N*LPC_order] */
    const SKP_int16                 *pCB_Q15,           /* I    Codebook vectors
             [K*LPC_order] */
    const SKP_int                   N,                  /* I    Number of input v
ectors                      */
    const SKP_int                   K,                  /* I    Number of codeboo
k vectors                   */
    const SKP_int                   LPC_order           /* I    Number of LPCs
                            */
);

/* Entropy constrained MATRIX-weighted VQ, for a single input data vector */
void SKP_Silk_VQ_WMat_EC_FIX(
    SKP_int                         *ind,               /* O    index of best cod
ebook vector                */
    SKP_int32                       *rate_dist_Q14,     /* O    best weighted qua
ntization error + mu * rate*/
    const SKP_int16                 *in_Q14,            /* I    input vector to b
e quantized                 */
    const SKP_int32                 *W_Q18,             /* I    weighting matrix
                            */
    const SKP_int16                 *cb_Q14,            /* I    codebook
                            */
    const SKP_int16                 *cl_Q6,             /* I    code length for e
ach codebook vector         */
    const SKP_int                   mu_Q8,              /* I    tradeoff between
weighted error and rate     */
    SKP_int                         L                   /* I    number of vectors
 in codebook                */
);

/******************/
/* Linear Algebra */
/******************/

/* Calculates correlation matrix X'*X */
```

```
void SKP_Silk_corrMatrix_FIX(
    const SKP_int16                 *x,         /* I    x vector [L + order - 1]
used to form data matrix X */
    const SKP_int                   L,          /* I    Length of vectors
                            */
    const SKP_int                   order,      /* I    Max lag for correlation
                            */
    SKP_int32                       *XX,        /* O    Pointer to X'*X correlati
on matrix [ order x order ]*/
    SKP_int                         *rshifts    /* I/O  Right shifts of correlati
ons                         */
);

/* Calculates correlation vector X'*t */
void SKP_Silk_corrVector_FIX(
```

```
    const SKP_int16                 *x,         /* I    x vector [L + order - 1]
used to form data matrix X */
    const SKP_int16                 *t,         /* I    target vector [L]
                        */
    const SKP_int                   L,          /* I    Length of vectors
                        */
    const SKP_int                   order,      /* I    Max lag for correlation
                        */
    SKP_int32                       *Xt,        /* O    Pointer to X'*t correlati
on vector [order]       */
    const SKP_int                   rshifts     /* I    Right shifts of correlati
ons                     */
);

/* Add noise to matrix diagonal */
void SKP_Silk_regularize_correlations_FIX(
    SKP_int32                       *XX,                /* I/O  Correlation matri
ces                     */
    SKP_int32                       *xx,                /* I/O  Correlation value
s                       */
    SKP_int32                       noise,              /* I    Noise to add
                        */
    SKP_int                         D                   /* I    Dimension of XX
                        */
);

/* Solves Ax = b, assuming A is symmetric */
void SKP_Silk_solve_LDL_FIX(
    SKP_int32                       *A,                 /* I    Pointer to symetr
ic square matrix A      */
    SKP_int                         M,                  /* I    Size of matrix
                        */
    const SKP_int32                 *b,                 /* I    Pointer to b vect
or                      */
    SKP_int32                       *x_Q16              /* O    Pointer to x solu
tion vector             */
);

/* Residual energy: nrg = wxx - 2 * wXx * c + c' * wXX * c */
SKP_int32 SKP_Silk_residual_energy16_covar_FIX(
    const SKP_int16                 *c,                 /* I    Prediction vector
                        */
    const SKP_int32                 *wXX,               /* I    Correlation matri
x                       */
    const SKP_int32                 *wXx,               /* I    Correlation vecto
r                       */
    SKP_int32                       wxx,                /* I    Signal energy
                        */
    SKP_int                         D,                  /* I    Dimension
                        */
    SKP_int                         cQ                  /* I    Q value for c vec
tor 0 - 15              */
);

/* Calculates residual energies of input subframes where all subframes have LPC_o
rder   */
/* of preceeding samples
       */
void SKP_Silk_residual_energy_FIX(
          SKP_int32 nrgs[ NB_SUBFR ],           /* O    Residual energy per subfr
ame    */
          SKP_int   nrgsQ[ NB_SUBFR ],          /* O    Q value per subframe
```

```
       */
    const SKP_int16 x[],                        /* I    Input signal
       */
    const SKP_int16 a_Q12[ 2 ][ MAX_LPC_ORDER ],/* I    AR coefs for each frame h
alf    */
    const SKP_int32 gains[ NB_SUBFR ],          /* I    Quantization gains
       */
    const SKP_int   subfr_length,               /* I    Subframe length
       */
    const SKP_int   LPC_order                   /* I    LPC order
       */
);

#ifndef FORCE_CPP_BUILD
#ifdef __cplusplus
```

```
}
#endif /* __cplusplus */
#endif /* FORCE_CPP_BUILD */
#endif /* SKP_SILK_MAIN_FIX_H */
```

A.68. src/SKP_Silk_NLSF2A.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/* conversion between prediction filter coefficients and LSFs   */
/* order should be even                                         */
/* a piecewise linear approximation maps LSF <-> cos(LSF)       */
/* therefore the result is not accurate LSFs, but the two       */
/* function are accurate inverses of each other                 */

#include "SKP_Silk_SigProc_FIX.h"

/* helper function for NLSF2A(..) */
SKP_INLINE void SKP_Silk_NLSF2A_find_poly(
    SKP_int32        *out,        /* o    intermediate polynomial, Q20
 */
    const SKP_int32    *cLSF,     /* i    vector of interleaved 2*cos(LSFs), Q20
 */
```

```
    SKP_int            dd          /* i    polynomial order (= 1/2 * filter order)
 */
)
{
    SKP_int        k, n;
    SKP_int32    ftmp;

    out[0] = SKP_LSHIFT( 1, 20 );
    out[1] = -cLSF[0];
    for( k = 1; k < dd; k++ ) {
        ftmp = cLSF[2*k];            // Q20
        out[k+1] = SKP_LSHIFT( out[k-1], 1 ) - (SKP_int32)SKP_RSHIFT_ROUND64( SKP
_SMULL( ftmp, out[k] ), 20 );
        for( n = k; n > 1; n-- ) {
            out[n] += out[n-2] - (SKP_int32)SKP_RSHIFT_ROUND64( SKP_SMULL( ftmp,
out[n-1] ), 20 );
        }
        out[1] -= ftmp;
    }
}

/* compute whitening filter coefficients from normalized line spectral frequencie
s */
void SKP_Silk_NLSF2A(
    SKP_int16       *a,              /* o    monic whitening filter coefficients
 in Q12,  [d]    */
    const SKP_int    *NLSF,          /* i    normalized line spectral frequencie
s in Q15, [d]    */
    const SKP_int    d               /* i    filter order (should be even)
                */
)
{
    SKP_int k, i, dd;
    SKP_int32 cos_LSF_Q20[SigProc_MAX_ORDER_LPC];
    SKP_int32 P[SigProc_MAX_ORDER_LPC/2+1], Q[SigProc_MAX_ORDER_LPC/2+1];
    SKP_int32 Ptmp, Qtmp;
    SKP_int32 f_int;
    SKP_int32 f_frac;
    SKP_int32 cos_val, delta;
    SKP_int32 a_int32[SigProc_MAX_ORDER_LPC];
    SKP_int32 maxabs, absval, idx=0, sc_Q16;

    SKP_assert(LSF_COS_TAB_SZ_FIX == 128);

    /* convert LSFs to 2*cos(LSF(i)), using piecewise linear curve from table */
    for( k = 0; k < d; k++ ) {
        SKP_assert(NLSF[k] >= 0 );
        SKP_assert(NLSF[k] <= 32767 );

        /* f_int on a scale 0-127 (rounded down) */
        f_int = SKP_RSHIFT( NLSF[k], 15 - 7 );

        /* f_frac, range: 0..255 */
        f_frac = NLSF[k] - SKP_LSHIFT( f_int, 15 - 7 );
```

```
        SKP_assert(f_int >= 0);
        SKP_assert(f_int < LSF_COS_TAB_SZ_FIX );

        /* Read start and end value from table */
        cos_val = SKP_Silk_LSFCosTab_FIX_Q12[ f_int ];                /* Q12 */
        delta   = SKP_Silk_LSFCosTab_FIX_Q12[ f_int + 1 ] - cos_val;  /* Q12, wit
h a range of 0..200 */

        /* Linear interpolation */
        cos_LSF_Q20[k] = SKP_LSHIFT( cos_val, 8 ) + SKP_MUL( delta, f_frac ); /*
Q20 */
    }

    dd = SKP_RSHIFT( d, 1 );

    /* generate even and odd polynomials using convolution */
    SKP_Silk_NLSF2A_find_poly( P, &cos_LSF_Q20[0], dd );
    SKP_Silk_NLSF2A_find_poly( Q, &cos_LSF_Q20[1], dd );

    /* convert even and odd polynomials to SKP_int32 Q12 filter coefs */
    for( k = 0; k < dd; k++ ) {
        Ptmp = P[k+1] + P[k];
        Qtmp = Q[k+1] - Q[k];

        /* the Ptmp and Qtmp values at this stage need to fit in int32 */

        a_int32[k]     = -SKP_RSHIFT_ROUND( Ptmp + Qtmp, 9 ); /* Q20 -> Q12 */
        a_int32[d-k-1] =  SKP_RSHIFT_ROUND( Qtmp - Ptmp, 9 ); /* Q20 -> Q12 */
    }

    /* Limit the maximum absolute value of the prediction coefficients */
    for( i = 0; i < 10; i++ ) {
        /* Find maximum absolute value and its index */
        maxabs = 0;
        for( k = 0; k < d; k++ ) {
            absval = SKP_abs( a_int32[k] );
            if( absval > maxabs ) {
                maxabs = absval;
                idx       = k;
            }
        }

        if( maxabs > SKP_int16_MAX ) {
            /* Reduce magnitude of prediction coefficients */
            sc_Q16 = 65470 - SKP_DIV32( SKP_MUL( 65470 >> 2, maxabs - SKP_int16_M
AX ),
                                        SKP_RSHIFT32( SKP_MUL( maxabs, idx + 1),
2 ) );
            SKP_Silk_bwexpander_32( a_int32, d, sc_Q16 );
        } else {
            break;
        }
```

```
    }

    /* Reached the last iteration */
    if( i == 10 ) {
        SKP_assert(0);
        for( k = 0; k < d; k++ ) {
            a_int32[k] = SKP_SAT16( a_int32[k] );
        }
    }

    /* Return as SKP_int16 Q12 coefficients */
    for( k = 0; k < d; k++ ) {
        a[k] = (SKP_int16)a_int32[k];
    }
}
```

A.69. src/SKP_Silk_NLSF2A_stable.c


```
#include "SKP_Silk_main.h"
```

```
/* Convert NLSF parameters to stable AR prediction filter coefficients */
void SKP_Silk_NLSF2A_stable(
    SKP_int16                       pAR_Q12[ MAX_LPC_ORDER ],   /* O    Stabilize
d AR coefs [LPC_order]     */
    const SKP_int                   pNLSF[ MAX_LPC_ORDER ],     /* I    NLSF vect
or         [LPC_order]     */
    const SKP_int                   LPC_order                   /* I    LPC/LSF o
rder                       */
)
{
    SKP_int   i;
    SKP_int32 invGain_Q30;

    SKP_Silk_NLSF2A( pAR_Q12, pNLSF, LPC_order );

    /* Ensure stable LPCs */
    for( i = 0; i < MAX_LPC_STABILIZE_ITERATIONS; i++ ) {
        if( SKP_Silk_LPC_inverse_pred_gain( &invGain_Q30, pAR_Q12, LPC_order ) ==
 1 ) {
            SKP_Silk_bwexpander( pAR_Q12, LPC_order, 65536 - SKP_SMULBB( 66, i )
); /* 66_Q16 = 0.001 */
        } else {
            break;
        }
    }

    /* Reached the last iteration */
    if( i == MAX_LPC_STABILIZE_ITERATIONS ) {
        for( i = 0; i < LPC_order; i++ ) {
            pAR_Q12[ i ] = 0;
        }
    }
}
```

A.70. src/SKP_Silk_NLSF_MSVQ_decode.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
```

```
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

/* NLSF vector decoder */
void SKP_Silk_NLSF_MSVQ_decode(
    SKP_int                         *pNLSF_Q15,     /* O    Pointer to decoded ou
tput vector [LPC_ORDER x 1]    */
    const SKP_Silk_NLSF_CB_struct   *psNLSF_CB,     /* I    Pointer to NLSF codeb
ook struct                     */
    const SKP_int                   *NLSFIndices,   /* I    Pointer to NLSF indic
es          [nStages x 1]      */
    const SKP_int                   LPC_order       /* I    LPC order used
                                */
)
{
    const SKP_int16 *pCB_element;
          SKP_int    s;
          SKP_int    i;

    /* Check that each index is within valid range */
    SKP_assert( 0 <= NLSFIndices[ 0 ] && NLSFIndices[ 0 ] < psNLSF_CB->CBStages[
0 ].nVectors );

    /* Point to the first vector element */
    pCB_element = &psNLSF_CB->CBStages[ 0 ].CB_NLSF_Q15[ SKP_MUL( NLSFIndices[ 0
], LPC_order ) ];

    /* Initialize with the codebook vector from stage 0 */
    for( i = 0; i < LPC_order; i++ ) {
        pNLSF_Q15[ i ] = ( SKP_int )pCB_element[ i ];
    }

    for( s = 1; s < psNLSF_CB->nStages; s++ ) {
        /* Check that each index is within valid range */
        SKP_assert( 0 <= NLSFIndices[ s ] && NLSFIndices[ s ] < psNLSF_CB->CBStag
es[ s ].nVectors );

        if( LPC_order == 16 ) {
            /* Point to the first vector element */
            pCB_element = &psNLSF_CB->CBStages[ s ].CB_NLSF_Q15[ SKP_LSHIFT( NLSF
Indices[ s ], 4 ) ];

            /* Add the codebook vector from the current stage */
            pNLSF_Q15[  0 ] += pCB_element[  0 ];
            pNLSF_Q15[  1 ] += pCB_element[  1 ];
```

```
            pNLSF_Q15[  2 ] += pCB_element[  2 ];
            pNLSF_Q15[  3 ] += pCB_element[  3 ];
            pNLSF_Q15[  4 ] += pCB_element[  4 ];
            pNLSF_Q15[  5 ] += pCB_element[  5 ];
            pNLSF_Q15[  6 ] += pCB_element[  6 ];
            pNLSF_Q15[  7 ] += pCB_element[  7 ];
            pNLSF_Q15[  8 ] += pCB_element[  8 ];
            pNLSF_Q15[  9 ] += pCB_element[  9 ];
            pNLSF_Q15[ 10 ] += pCB_element[ 10 ];
            pNLSF_Q15[ 11 ] += pCB_element[ 11 ];
            pNLSF_Q15[ 12 ] += pCB_element[ 12 ];
            pNLSF_Q15[ 13 ] += pCB_element[ 13 ];
            pNLSF_Q15[ 14 ] += pCB_element[ 14 ];
            pNLSF_Q15[ 15 ] += pCB_element[ 15 ];
        } else {
            /* Point to the first vector element */
            pCB_element = &psNLSF_CB->CBStages[ s ].CB_NLSF_Q15[ SKP_SMULBB( NLSF
Indices[ s ], LPC_order ) ];

            /* Add the codebook vector from the current stage */
            for( i = 0; i < LPC_order; i++ ) {
                pNLSF_Q15[ i ] += pCB_element[ i ];
            }
        }
    }

    /* NLSF stabilization */
    SKP_Silk_NLSF_stabilize( pNLSF_Q15, psNLSF_CB->NDeltaMin_Q15, LPC_order );
}
```

A.71. src/SKP_Silk_NLSF_MSVQ_encode_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
```

```
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/***********************/
/* NLSF vector encoder */
/***********************/
void SKP_Silk_NLSF_MSVQ_encode_FIX(
          SKP_int                     *NLSFIndices,           /* O    Codebook path
 vector [ CB_STAGES ]      */
          SKP_int                     *pNLSF_Q15,             /* I/O  Quantized NLS
F vector [ LPC_ORDER ]     */
    const SKP_Silk_NLSF_CB_struct     *psNLSF_CB,             /* I    Codebook obje
ct                         */
    const SKP_int                     *pNLSF_q_Q15_prev,      /* I    Prev. quantiz
ed NLSF vector [LPC_ORDER] */
    const SKP_int                     *pW_Q6,                 /* I    NLSF weight v
ector [ LPC_ORDER ]        */
    const SKP_int                     NLSF_mu_Q15,            /* I    Rate weight f
or the RD optimization     */
    const SKP_int                     NLSF_mu_fluc_red_Q16,   /* I    Fluctuation r
eduction error weight      */
    const SKP_int                     NLSF_MSVQ_Survivors,    /* I    Max survivors
 from each stage           */
    const SKP_int                     LPC_order,              /* I    LPC order
                           */
    const SKP_int                     deactivate_fluc_red     /* I    Deactivate fl
uctuation reduction        */
)
{
    SKP_int     i, s, k, cur_survivors = 0, prev_survivors, input_index, cb_index
, bestIndex;
    SKP_int32   rateDistThreshold_Q18;
    SKP_int     pNLSF_in_Q15[ MAX_LPC_ORDER ];
#if( NLSF_MSVQ_FLUCTUATION_REDUCTION == 1 )
    SKP_int32   se_Q15, wsse_Q20, bestRateDist_Q20;
#endif

#if( LOW_COMPLEXITY_ONLY == 1 )
    SKP_int32   pRateDist_Q18[  NLSF_MSVQ_TREE_SEARCH_MAX_VECTORS_EVALUATED_LC_MO
DE ];
    SKP_int32   pRate_Q5[       MAX_NLSF_MSVQ_SURVIVORS_LC_MODE ];
    SKP_int32   pRate_new_Q5[   MAX_NLSF_MSVQ_SURVIVORS_LC_MODE ];
    SKP_int     pTempIndices[   MAX_NLSF_MSVQ_SURVIVORS_LC_MODE ];
    SKP_int     pPath[          MAX_NLSF_MSVQ_SURVIVORS_LC_MODE * NLSF_MSVQ_MAX_C
B_STAGES ];
    SKP_int     pPath_new[      MAX_NLSF_MSVQ_SURVIVORS_LC_MODE * NLSF_MSVQ_MAX_C
B_STAGES ];
    SKP_int     pRes_Q15[       MAX_NLSF_MSVQ_SURVIVORS_LC_MODE * MAX_LPC_ORDER ]
;
    SKP_int     pRes_new_Q15[   MAX_NLSF_MSVQ_SURVIVORS_LC_MODE * MAX_LPC_ORDER ]
;
```

```
#else
    SKP_int32   pRateDist_Q18[  NLSF_MSVQ_TREE_SEARCH_MAX_VECTORS_EVALUATED ];
```

```
    SKP_int32   pRate_Q5[       MAX_NLSF_MSVQ_SURVIVORS ];
    SKP_int32   pRate_new_Q5[   MAX_NLSF_MSVQ_SURVIVORS ];
    SKP_int     pTempIndices[   MAX_NLSF_MSVQ_SURVIVORS ];
    SKP_int     pPath[          MAX_NLSF_MSVQ_SURVIVORS * NLSF_MSVQ_MAX_CB_STAGES
 ];
    SKP_int     pPath_new[      MAX_NLSF_MSVQ_SURVIVORS * NLSF_MSVQ_MAX_CB_STAGES
 ];
    SKP_int     pRes_Q15[       MAX_NLSF_MSVQ_SURVIVORS * MAX_LPC_ORDER ];
    SKP_int     pRes_new_Q15[   MAX_NLSF_MSVQ_SURVIVORS * MAX_LPC_ORDER ];
#endif

    const SKP_int   *pConstInt;
          SKP_int   *pInt;
    const SKP_int16 *pCB_element;
    const SKP_Silk_NLSF_CBS *pCurrentCBStage;

    SKP_assert( NLSF_MSVQ_Survivors <= MAX_NLSF_MSVQ_SURVIVORS );
    SKP_assert( ( LOW_COMPLEXITY_ONLY == 0 ) || ( NLSF_MSVQ_Survivors <= MAX_NLSF
_MSVQ_SURVIVORS_LC_MODE ) );



    /* Copy the input vector */
    SKP_memcpy( pNLSF_in_Q15, pNLSF_Q15, LPC_order * sizeof( SKP_int ) );

    /****************************************************/
    /* Tree search for the multi-stage vector quantizer */
    /****************************************************/

    /* Clear accumulated rates */
    SKP_memset( pRate_Q5, 0, NLSF_MSVQ_Survivors * sizeof( SKP_int32 ) );

    /* Copy NLSFs into residual signal vector */
    for( i = 0; i < LPC_order; i++ ) {
        pRes_Q15[ i ] = pNLSF_Q15[ i ];
    }

    /* Set first stage values */
    prev_survivors = 1;

    /* Loop over all stages */
    for( s = 0; s < psNLSF_CB->nStages; s++ ) {

        /* Set a pointer to the current stage codebook */
        pCurrentCBStage = &psNLSF_CB->CBStages[ s ];

        /* Calculate the number of survivors in the current stage */
        cur_survivors = SKP_min_32( NLSF_MSVQ_Survivors, SKP_SMULBB( prev_survivo
rs, pCurrentCBStage->nVectors ) );

#if( NLSF_MSVQ_FLUCTUATION_REDUCTION == 0 )
        /* Find a single best survivor in the last stage, if we */
```

```
        /* do not need candidates for fluctuation reduction     */
        if( s == psNLSF_CB->nStages - 1 ) {
            cur_survivors = 1;
        }
#endif

        /* Nearest neighbor clustering for multiple input data vectors */
        SKP_Silk_NLSF_VQ_rate_distortion_FIX( pRateDist_Q18, pCurrentCBStage, pRe
s_Q15, pW_Q6,
            pRate_Q5, NLSF_mu_Q15, prev_survivors, LPC_order );

        /* Sort the rate-distortion errors */
        SKP_Silk_insertion_sort_increasing( pRateDist_Q18, pTempIndices,
            prev_survivors * pCurrentCBStage->nVectors, cur_survivors );

        /* Discard survivors with rate-distortion values too far above the best o
ne */
        if( pRateDist_Q18[ 0 ] < SKP_int32_MAX / NLSF_MSVQ_SURV_MAX_REL_RD ) {
            rateDistThreshold_Q18 = SKP_MUL( NLSF_MSVQ_SURV_MAX_REL_RD, pRateDist
_Q18[ 0 ] );
            while( pRateDist_Q18[ cur_survivors - 1 ] > rateDistThreshold_Q18 &&
cur_survivors > 1 ) {
                cur_survivors--;
            }
        }
        /* Update accumulated codebook contributions for the 'cur_survivors' best
 codebook indices */
        for( k = 0; k < cur_survivors; k++ ) {
            if( s > 0 ) {
                /* Find the indices of the input and the codebook vector */
                if( pCurrentCBStage->nVectors == 8 ) {
                    input_index = SKP_RSHIFT( pTempIndices[ k ], 3 );
                    cb_index    = pTempIndices[ k ] & 7;
                } else {
                    input_index = SKP_DIV32_16( pTempIndices[ k ], pCurrentCBStag
e->nVectors );
                    cb_index    = pTempIndices[ k ] - SKP_SMULBB( input_index, pC
urrentCBStage->nVectors );
                }
            } else {
                /* Find the indices of the input and the codebook vector */
                input_index = 0;
                cb_index    = pTempIndices[ k ];
            }

            /* Subtract new contribution from the previous residual vector for ea
ch of 'cur_survivors' */
            pConstInt   = &pRes_Q15[ SKP_SMULBB( input_index, LPC_order ) ];
            pCB_element = &pCurrentCBStage->CB_NLSF_Q15[ SKP_SMULBB( cb_index, LP
C_order ) ];
            pInt        = &pRes_new_Q15[ SKP_SMULBB( k, LPC_order ) ];
            for( i = 0; i < LPC_order; i++ ) {
                pInt[ i ] = pConstInt[ i ] - ( SKP_int )pCB_element[ i ];
            }

            /* Update accumulated rate for stage 1 to the current */
            pRate_new_Q5[ k ] = pRate_Q5[ input_index ] + pCurrentCBStage->Rates_
Q5[ cb_index ];
```

```
            /* Copy paths from previous matrix, starting with the best path */
            pConstInt = &pPath[ SKP_SMULBB( input_index, psNLSF_CB->nStages ) ];
            pInt      = &pPath_new[ SKP_SMULBB( k, psNLSF_CB->nStages ) ];
            for( i = 0; i < s; i++ ) {
                pInt[ i ] = pConstInt[ i ];
            }
            /* Write the current stage indices for the 'cur_survivors' to the bes
t path matrix */
            pInt[ s ] = cb_index;
        }

        if( s < psNLSF_CB->nStages - 1 ) {
            /* Copy NLSF residual matrix for next stage */
            SKP_memcpy( pRes_Q15, pRes_new_Q15, SKP_SMULBB( cur_survivors, LPC_or
der ) * sizeof( SKP_int ) );

            /* Copy rate vector for next stage */
            SKP_memcpy( pRate_Q5, pRate_new_Q5, cur_survivors * sizeof( SKP_int32
 ) );

            /* Copy best path matrix for next stage */
            SKP_memcpy( pPath, pPath_new, SKP_SMULBB( cur_survivors, psNLSF_CB->n
Stages ) * sizeof( SKP_int ) );
        }

        prev_survivors = cur_survivors;
    }

    /* (Preliminary) index of the best survivor, later to be decoded */
    bestIndex = 0;

#if( NLSF_MSVQ_FLUCTUATION_REDUCTION == 1 )
    /*****************************/
    /* NLSF fluctuation reduction */
    /*****************************/
    if( deactivate_fluc_red != 1 ) {

        /* Search among all survivors, now taking also weighted fluctuation error
s into account */
        bestRateDist_Q20 = SKP_int32_MAX;
        for( s = 0; s < cur_survivors; s++ ) {
            /* Decode survivor to compare with previous quantized NLSF vector */
            SKP_Silk_NLSF_MSVQ_decode( pNLSF_Q15, psNLSF_CB, &pPath_new[ SKP_SMUL
BB( s, psNLSF_CB->nStages ) ], LPC_order );

            /* Compare decoded NLSF vector with the previously quantized vector *
/
            wsse_Q20 = 0;
            for( i = 0; i < LPC_order; i += 2 ) {
                /* Compute weighted squared quantization error for index i */
                se_Q15 = pNLSF_Q15[ i ] - pNLSF_q_Q15_prev[ i ]; // range: [ -327
67 : 32767 ]
                wsse_Q20 = SKP_SMLAWB( wsse_Q20, SKP_SMULBB( se_Q15, se_Q15 ), pW
_Q6[ i ] );

                /* Compute weighted squared quantization error for index i + 1 */
                se_Q15 = pNLSF_Q15[ i + 1 ] - pNLSF_q_Q15_prev[ i + 1 ]; // range
: [ -32767 : 32767 ]
```

```
                wsse_Q20 = SKP_SMLAWB( wsse_Q20, SKP_SMULBB( se_Q15, se_Q15 ), pW
_Q6[ i + 1 ] );
            }
            SKP_assert( wsse_Q20 >= 0 );

            /* Add the fluctuation reduction penalty to the rate distortion error
 */
            wsse_Q20 = SKP_ADD_POS_SAT32( pRateDist_Q18[ s ], SKP_SMULWB( wsse_Q2
0, NLSF_mu_fluc_red_Q16 ) );

            /* Keep index of best survivor */
            if( wsse_Q20 < bestRateDist_Q20 ) {
                bestRateDist_Q20 = wsse_Q20;
                bestIndex = s;
            }
        }
    }
#endif

    /* Copy best path to output argument */
    SKP_memcpy( NLSFIndices, &pPath_new[ SKP_SMULBB( bestIndex, psNLSF_CB->nStage
s ) ], psNLSF_CB->nStages * sizeof( SKP_int ) );

    /* Decode and stabilize the best survivor */
    SKP_Silk_NLSF_MSVQ_decode( pNLSF_Q15, psNLSF_CB, NLSFIndices, LPC_order );

}
```

A.72. src/SKP_Silk_NLSF_stabilize.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
```

```
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/* NLSF stabilizer:                                                    */
/*                                                                     */
/* - Moves NLSFs futher apart if they are too close                    */
/* - Moves NLSFs away from borders if they are too close               */
/* - High effort to achieve a modification with minimum                */
/*     Euclidean distance to input vector                              */
/* - Output are sorted NLSF coefficients                               */
/*                                                                     */
#include "SKP_Silk_SigProc_FIX.h"

/* Constant Definitions */
#define MAX_LOOPS        20

/* NLSF stabilizer, for a single input data vector */
void SKP_Silk_NLSF_stabilize(
          SKP_int    *NLSF_Q15,            /* I/O:  Unstable/stabilized normalize
d LSF vector in Q15 [L]                    */
    const SKP_int    *NDeltaMin_Q15,       /* I:    Normalized delta min vector i
n Q15, NDeltaMin_Q15[L] must be >= 1 [L+1] */
    const SKP_int     L                    /* I:    Number of NLSF parameters in
the input vector                           */
)
{
    SKP_int        center_freq_Q15, diff_Q15, min_center_Q15, max_center_Q15;
    SKP_int32    min_diff_Q15;
    SKP_int        loops;
    SKP_int        i, I=0, k;

    /* This is necessary to ensure an output within range of a SKP_int16 */
    SKP_assert( NDeltaMin_Q15[L] >= 1 );

    for( loops = 0; loops < MAX_LOOPS; loops++ ) {
        /***************************/
        /* Find smallest distance */
        /***************************/
        /* First element */
        min_diff_Q15 = NLSF_Q15[0] - NDeltaMin_Q15[0];
        I = 0;
        /* Middle elements */
        for( i = 1; i <= L-1; i++ ) {
            diff_Q15 = NLSF_Q15[i] - ( NLSF_Q15[i-1] + NDeltaMin_Q15[i] );
            if( diff_Q15 < min_diff_Q15 ) {
                min_diff_Q15 = diff_Q15;
                I = i;
```

```
            }
        }
        /* Last element */
        diff_Q15 = (1<<15) - ( NLSF_Q15[L-1] + NDeltaMin_Q15[L] );
        if( diff_Q15 < min_diff_Q15 ) {
            min_diff_Q15 = diff_Q15;
            I = L;
        }

        /***************************************************/
        /* Now check if the smallest distance non-negative */
        /***************************************************/
        if (min_diff_Q15 >= 0) {
            return;
        }

        if( I == 0 ) {
            /* Move away from lower limit */
            NLSF_Q15[0] = NDeltaMin_Q15[0];

        } else if( I == L) {
            /* Move away from higher limit */
            NLSF_Q15[L-1] = (1<<15) - NDeltaMin_Q15[L];

        } else {
            /* Find the lower extreme for the location of the current center freq
uency */
            min_center_Q15 = 0;
            for( k = 0; k < I; k++ ) {
                min_center_Q15 += NDeltaMin_Q15[k];
            }
            min_center_Q15 += SKP_RSHIFT( NDeltaMin_Q15[I], 1 );

            /* Find the upper extreme for the location of the current center freq
uency */
            max_center_Q15 = (1<<15);
            for( k = L; k > I; k-- ) {
                max_center_Q15 -= NDeltaMin_Q15[k];
            }
            max_center_Q15 -= ( NDeltaMin_Q15[I] - SKP_RSHIFT( NDeltaMin_Q15[I],
1 ) );

            /* Move apart, sorted by value, keeping the same center frequency */
            center_freq_Q15 = SKP_LIMIT( SKP_RSHIFT_ROUND( (SKP_int32)NLSF_Q15[I-
1] + (SKP_int32)NLSF_Q15[I], 1 ),
                min_center_Q15, max_center_Q15 );
            NLSF_Q15[I-1] = center_freq_Q15 - SKP_RSHIFT( NDeltaMin_Q15[I], 1 );
            NLSF_Q15[I] = NLSF_Q15[I-1] + NDeltaMin_Q15[I];
        }
    }

    /* Safe and simple fall back method, which is less ideal than the above */
```

```
    if( loops == MAX_LOOPS )
    {
        /* Insertion sort (fast for already almost sorted arrays):   */
        /* Best case:  O(n)   for an already sorted array            */
        /* Worst case: O(n^2) for an inversely sorted array          */
        SKP_Silk_insertion_sort_increasing_all_values(&NLSF_Q15[0], L);

        /* First NLSF should be no less than NDeltaMin[0] */
        NLSF_Q15[0] = SKP_max_int( NLSF_Q15[0], NDeltaMin_Q15[0] );

        /* Keep delta_min distance between the NLSFs */
        for( i = 1; i < L; i++ )
            NLSF_Q15[i] = SKP_max_int( NLSF_Q15[i], NLSF_Q15[i-1] + NDeltaMin_Q15
[i] );

        /* Last NLSF should be no higher than 1 - NDeltaMin[L] */
        NLSF_Q15[L-1] = SKP_min_int( NLSF_Q15[L-1], (1<<15) - NDeltaMin_Q15[L] );

        /* Keep NDeltaMin distance between the NLSFs */
        for( i = L-2; i >= 0; i-- )
            NLSF_Q15[i] = SKP_min_int( NLSF_Q15[i], NLSF_Q15[i+1] - NDeltaMin_Q15
[i+1] );
    }
}

/* NLSF stabilizer, over multiple input column data vectors */
void SKP_Silk_NLSF_stabilize_multi(
          SKP_int        *NLSF_Q15,        /* I/O:  Unstable/stabilized normalize
d LSF vectors in Q15 [LxN]                 */
    const SKP_int        *NDeltaMin_Q15,   /* I:    Normalized delta min vector i
n Q15, NDeltaMin_Q15[L] must be >= 1 [L+1] */
    const SKP_int         N,               /* I:    Number of input vectors to be
 stabilized                                */
    const SKP_int         L                /* I:    NLSF vector dimension
                                           */
)
{
    SKP_int n;

    /* loop over input data */
    for( n = 0; n < N; n++ ) {
        SKP_Silk_NLSF_stabilize( &NLSF_Q15[n * L], NDeltaMin_Q15, L );
    }
}
```

A.73. src/SKP_Silk_NLSF_VQ_rate_distortion_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
```

```
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/* Rate-Distortion calculations for multiple input data vectors */
void SKP_Silk_NLSF_VQ_rate_distortion_FIX(
    SKP_int32                       *pRD_Q20,           /* O    Rate-distortion v
alues [psNLSF_CBS->nVectors*N] */
    const SKP_Silk_NLSF_CBS         *psNLSF_CBS,        /* I    NLSF codebook sta
ge struct                       */
    const SKP_int                   *in_Q15,            /* I    Input vectors to
be quantized                    */
    const SKP_int                   *w_Q6,              /* I    Weight vector
                                */
    const SKP_int32                 *rate_acc_Q5,       /* I    Accumulated rates
 from previous stage            */
    const SKP_int                   mu_Q15,             /* I    Weight between we
ighted error and rate           */
    const SKP_int                   N,                  /* I    Number of input v
ectors to be quantized          */
    const SKP_int                   LPC_order           /* I    LPC order
                                */
)
{
    SKP_int   i, n;
    SKP_int32 *pRD_vec_Q20;

    /* Compute weighted quantization errors for all input vectors over one codebo
ok stage */
    SKP_Silk_NLSF_VQ_sum_error_FIX( pRD_Q20, in_Q15, w_Q6, psNLSF_CBS->CB_NLSF_Q1
5,
        N, psNLSF_CBS->nVectors, LPC_order );

    /* Loop over input vectors */
    pRD_vec_Q20 = pRD_Q20;
    for( n = 0; n < N; n++ ) {
        /* Add rate cost to error for each codebook vector */
        for( i = 0; i < psNLSF_CBS->nVectors; i++ ) {
```

```
            SKP_assert( rate_acc_Q5[ n ] + psNLSF_CBS->Rates_Q5[ i ] >= 0 );
            SKP_assert( rate_acc_Q5[ n ] + psNLSF_CBS->Rates_Q5[ i ] <= SKP_int16
_MAX );
            pRD_vec_Q20[ i ] = SKP_SMLABB( pRD_vec_Q20[ i ], rate_acc_Q5[ n ] + p
sNLSF_CBS->Rates_Q5[ i ], mu_Q15 );
            SKP_assert( pRD_vec_Q20[ i ] >= 0 );
        }
        pRD_vec_Q20 += psNLSF_CBS->nVectors;
    }
}
```

A.74. src/SKP_Silk_NLSF_VQ_sum_error_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/* Compute weighted quantization errors for an LPC_order element input vector, ov
er one codebook stage */
void SKP_Silk_NLSF_VQ_sum_error_FIX(
    SKP_int32                       *err_Q20,           /* O    Weighted quantiza
tion errors  [N*K]         */
    const SKP_int                   *in_Q15,            /* I    Input vectors to
be quantized [N*LPC_order] */
    const SKP_int                   *w_Q6,              /* I    Weighting vectors
             [N*LPC_order] */
    const SKP_int16                 *pCB_Q15,           /* I    Codebook vectors
             [K*LPC_order] */
```

```
    const SKP_int                   N,                  /* I    Number of input v
ectors                      */
    const SKP_int                   K,                  /* I    Number of codeboo
k vectors                   */
    const SKP_int                   LPC_order           /* I    Number of LPCs
                            */
)
{
    SKP_int         i, n, m;
    SKP_int32       diff_Q15, sum_error, Wtmp_Q6;
    SKP_int32       Wcpy_Q6[ MAX_LPC_ORDER / 2 ];
    const SKP_int16 *cb_vec_Q15;

    SKP_assert( LPC_order <= 16 );
    SKP_assert( ( LPC_order & 1 ) == 0 );

    /* Copy to local stack and pack two weights per int32 */
    for( m = 0; m < SKP_RSHIFT( LPC_order, 1 ); m++ ) {
        Wcpy_Q6[ m ] = w_Q6[ 2 * m ] | SKP_LSHIFT( ( SKP_int32 )w_Q6[ 2 * m + 1 ]
, 16 );
    }

    /* Loop over input vectors */
    for( n = 0; n < N; n++ ) {
        /* Loop over codebook */
        cb_vec_Q15 = pCB_Q15;
        for( i = 0; i < K; i++ ) {
            sum_error = 0;
            for( m = 0; m < LPC_order; m += 2 ) {
                /* Get two weights packed in an int32 */
                Wtmp_Q6 = Wcpy_Q6[ SKP_RSHIFT( m, 1 ) ];

                /* Compute weighted squared quantization error for index m */
                diff_Q15 = in_Q15[ m ] - *cb_vec_Q15++; // range: [ -32767 : 3276
7 ]
                sum_error = SKP_SMLAWB( sum_error, SKP_SMULBB( diff_Q15, diff_Q15
 ), Wtmp_Q6 );

                /* Compute weighted squared quantization error for index m + 1 */
                diff_Q15 = in_Q15[m + 1] - *cb_vec_Q15++; // range: [ -32767 : 32
767 ]
                sum_error = SKP_SMLAWT( sum_error, SKP_SMULBB( diff_Q15, diff_Q15
 ), Wtmp_Q6 );
            }
            SKP_assert( sum_error >= 0 );
            err_Q20[ i ] = sum_error;
        }
        err_Q20 += K;
        in_Q15 += LPC_order;
    }
}
```

A.75. src/SKP_Silk_NLSF_VQ_weights_laroia.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_SigProc_FIX.h"

/*
R. Laroia, N. Phamdo and N. Farvardin, "Robust and Efficient Quantization of Spee
ch LSP
Parameters Using Structured Vector Quantization", Proc. IEEE Int. Conf. Acoust.,
Speech,
Signal Processing, pp. 641-644, 1991.
*/

#define Q_OUT                       6

/* Laroia low complexity NLSF weights */
void SKP_Silk_NLSF_VQ_weights_laroia(
    SKP_int             *pNLSFW_Q6,         /* O: Pointer to input vector weights
           [D x 1]     */
    const SKP_int       *pNLSF_Q15,         /* I: Pointer to input vector
           [D x 1]     */
    const SKP_int       D                   /* I: Input vector dimension (even)
                        */
)
{
    SKP_int   k;
    SKP_int32 tmp1_int, tmp2_int;

    /* Check that we are guaranteed to end up within the required range */
```

```
    SKP_assert( D > 0 );
    SKP_assert( ( D & 1 ) == 0 );

    /* First value */
    tmp1_int = SKP_max_int( pNLSF_Q15[ 0 ], 1 );
    tmp1_int = SKP_DIV32_16( 1 << ( 15 + Q_OUT ), tmp1_int );
    tmp2_int = SKP_max_int( pNLSF_Q15[ 1 ] - pNLSF_Q15[ 0 ], 1 );
    tmp2_int = SKP_DIV32_16( 1 << ( 15 + Q_OUT ), tmp2_int );
    pNLSFW_Q6[ 0 ] = (SKP_int)SKP_min_int( tmp1_int + tmp2_int, SKP_int16_MAX );
    SKP_assert( pNLSFW_Q6[ 0 ] > 0 );

    /* Main loop */
    for( k = 1; k < D - 1; k += 2 ) {
        tmp1_int = SKP_max_int( pNLSF_Q15[ k + 1 ] - pNLSF_Q15[ k ], 1 );
        tmp1_int = SKP_DIV32_16( 1 << ( 15 + Q_OUT ), tmp1_int );
        pNLSFW_Q6[ k ] = (SKP_int)SKP_min_int( tmp1_int + tmp2_int, SKP_int16_MAX
 );
        SKP_assert( pNLSFW_Q6[ k ] > 0 );

        tmp2_int = SKP_max_int( pNLSF_Q15[ k + 2 ] - pNLSF_Q15[ k + 1 ], 1 );
        tmp2_int = SKP_DIV32_16( 1 << ( 15 + Q_OUT ), tmp2_int );
        pNLSFW_Q6[ k + 1 ] = (SKP_int)SKP_min_int( tmp1_int + tmp2_int, SKP_int16
_MAX );
        SKP_assert( pNLSFW_Q6[ k + 1 ] > 0 );
    }

    /* Last value */
    tmp1_int = SKP_max_int( ( 1 << 15 ) - pNLSF_Q15[ D - 1 ], 1 );
    tmp1_int = SKP_DIV32_16( 1 << ( 15 + Q_OUT ), tmp1_int );
    pNLSFW_Q6[ D - 1 ] = (SKP_int)SKP_min_int( tmp1_int + tmp2_int, SKP_int16_MAX
 );
    SKP_assert( pNLSFW_Q6[ D - 1 ] > 0 );
}
```

A.76. src/SKP_Silk_noise_shape_analysis_FIX.c

```
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"
#include "SKP_Silk_perceptual_parameters_FIX.h"


/*************************************************************/
/* Compute noise shaping coefficients and initial gain values */
/*************************************************************/
void SKP_Silk_noise_shape_analysis_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  Encoder state FIX
                    */
    SKP_Silk_encoder_control_FIX    *psEncCtrl,     /* I/O  Encoder control FIX
                    */
    const SKP_int16                 *pitch_res,     /* I    LPC residual from pit
ch analysis            */
    const SKP_int16                 *x              /* I    Input signal [ 2 * fr
ame_length + la_shape ]*/
)
{
    SKP_Silk_shape_state_FIX *psShapeSt = &psEnc->sShape;
    SKP_int     k, nSamples, lz, Qnrg, b_Q14, scale = 0, sz;
    SKP_int32   SNR_adj_dB_Q7, HarmBoost_Q16, HarmShapeGain_Q16, Tilt_Q16, tmp32;
    SKP_int32   nrg, pre_nrg_Q30, log_energy_Q7, log_energy_prev_Q7, energy_varia
tion_Q7;
    SKP_int32   delta_Q16, BWExp1_Q16, BWExp2_Q16, gain_mult_Q16, gain_add_Q16, s
trength_Q16, b_Q8;
    SKP_int32   auto_corr[     SHAPE_LPC_ORDER_MAX + 1 ];
    SKP_int32   refl_coef_Q16[ SHAPE_LPC_ORDER_MAX ];
    SKP_int32   AR_Q24[        SHAPE_LPC_ORDER_MAX ];
    SKP_int16   x_windowed[    SHAPE_LPC_WIN_MAX ];
    const SKP_int16 *x_ptr, *pitch_res_ptr;

    SKP_int32   sqrt_nrg[ NB_SUBFR ], Qnrg_vec[ NB_SUBFR ];

    /* Point to start of first LPC analysis block */
    x_ptr = x + psEnc->sCmn.la_shape - SKP_SMULBB( SHAPE_LPC_WIN_MS, psEnc->sCmn.
fs_kHz ) + psEnc->sCmn.frame_length / NB_SUBFR;

    /****************/
    /* CONTROL SNR  */
    /****************/
    /* Reduce SNR_dB values if recent bitstream has exceeded TargetRate */
```

```
    psEncCtrl->current_SNR_dB_Q7 = psEnc->SNR_dB_Q7 - SKP_SMULWB( SKP_LSHIFT( ( S
KP_int32 )psEnc->BufferedInChannel_ms, 7 ), 3277 );

    /* Reduce SNR_dB if inband FEC used */
    if( psEnc->speech_activity_Q8 > LBRR_SPEECH_ACTIVITY_THRES_Q8 ) {
        psEncCtrl->current_SNR_dB_Q7 -= SKP_RSHIFT( psEnc->inBandFEC_SNR_comp_Q8,
 1 );
    }

    /****************/
    /* GAIN CONTROL */
    /****************/
    /* Input quality is the average of the quality in the lowest two VAD bands */
    psEncCtrl->input_quality_Q14 = ( SKP_int )SKP_RSHIFT( ( SKP_int32 )psEncCtrl-
>input_quality_bands_Q15[ 0 ]
        + psEncCtrl->input_quality_bands_Q15[ 1 ], 2 );
    /* Coding quality level, between 0.0_Q0 and 1.0_Q0, but in Q14 */
    psEncCtrl->coding_quality_Q14 = SKP_RSHIFT( SKP_Silk_sigm_Q15( SKP_RSHIFT_ROU
ND( psEncCtrl->current_SNR_dB_Q7 - ( 18 << 7 ), 4 ) ), 1 );

    /* Reduce coding SNR during low speech activity */
    b_Q8 = ( 1 << 8 ) - psEnc->speech_activity_Q8;
    b_Q8 = SKP_SMULWB( SKP_LSHIFT( b_Q8, 8 ), b_Q8 );
    SNR_adj_dB_Q7 = SKP_SMLAWB( psEncCtrl->current_SNR_dB_Q7,
        SKP_SMULBB( -BG_SNR_DECR_dB_Q7 >> ( 4 + 1 ), b_Q8 ),
                        // Q11
        SKP_SMULWB( ( 1 << 14 ) + psEncCtrl->input_quality_Q14, psEncCtrl->coding
_quality_Q14 ) );      // Q12

    if( psEncCtrl->sCmn.sigtype == SIG_TYPE_VOICED ) {
        /* Reduce gains for periodic signals */
        SNR_adj_dB_Q7 = SKP_SMLAWB( SNR_adj_dB_Q7, HARM_SNR_INCR_dB_Q7 << 1, psEn
c->LTPCorr_Q15 );
    } else {
        /* For unvoiced signals and low-quality input, adjust the quality slower
than SNR_dB setting */
        SNR_adj_dB_Q7 = SKP_SMLAWB( SNR_adj_dB_Q7,
            SKP_SMLAWB( 6 << ( 7 + 2 ), -104856, psEncCtrl->current_SNR_dB_Q7 ),
   //-104856_Q18 = -0.4_Q0, Q9
            ( 1 << 14 ) - psEncCtrl->input_quality_Q14 );
   // Q14
    }

    /************************/
    /* SPARSENESS PROCESSING */
    /************************/
    /* Set quantizer offset */
    if( psEncCtrl->sCmn.sigtype == SIG_TYPE_VOICED ) {
        /* Initally set to 0; may be overruled in process_gains(..) */
        psEncCtrl->sCmn.QuantOffsetType = 0;
        psEncCtrl->sparseness_Q8 = 0;
    } else {
        /* Sparseness measure, based on relative fluctuations of energy per 2 mil
liseconds */
        nSamples = SKP_LSHIFT( psEnc->sCmn.fs_kHz, 1 );
        energy_variation_Q7 = 0;
        log_energy_prev_Q7  = 0;
        pitch_res_ptr = pitch_res;
        for( k = 0; k < FRAME_LENGTH_MS / 2; k++ ) {
```

```
            SKP_Silk_sum_sqr_shift( &nrg, &scale, pitch_res_ptr, nSamples );
            nrg += SKP_RSHIFT( nSamples, scale );           // Q(-scale)

            log_energy_Q7 = SKP_Silk_lin2log( nrg );
            if( k > 0 ) {
                energy_variation_Q7 += SKP_abs( log_energy_Q7 - log_energy_prev_Q
7 );
            }
            log_energy_prev_Q7 = log_energy_Q7;
            pitch_res_ptr += nSamples;
        }

        psEncCtrl->sparseness_Q8 = SKP_RSHIFT( SKP_Silk_sigm_Q15( SKP_SMULWB( ene
rgy_variation_Q7 - ( 5 << 7 ), 6554 ) ), 7 );    // 6554_Q16 = 0.1_Q0

        /* Set quantization offset depending on sparseness measure */
        if( psEncCtrl->sparseness_Q8 > SPARSENESS_THRESHOLD_QNT_OFFSET_Q8 ) {
            psEncCtrl->sCmn.QuantOffsetType = 0;
        } else {
            psEncCtrl->sCmn.QuantOffsetType = 1;
        }

        /* Increase coding SNR for sparse signals */
        SNR_adj_dB_Q7 = SKP_SMLAWB( SNR_adj_dB_Q7, SPARSE_SNR_INCR_dB_Q7 << 8, ps
EncCtrl->sparseness_Q8 - ( 1 << 7 ) );
    }

    /*******************************/
    /* Control bandwidth expansion */
    /*******************************/
    delta_Q16  = SKP_SMULWB( ( 1 << 16 ) - SKP_SMULBB( 3, psEncCtrl->coding_quali
ty_Q14 ), LOW_RATE_BANDWIDTH_EXPANSION_DELTA_Q16 );
    BWExp1_Q16 = BANDWIDTH_EXPANSION_Q16 - delta_Q16;
    BWExp2_Q16 = BANDWIDTH_EXPANSION_Q16 + delta_Q16;
    if( psEnc->sCmn.fs_kHz == 24 ) {
        /* Less bandwidth expansion for super wideband */
        BWExp1_Q16 = ( 1 << 16 ) - SKP_SMULWB( SWB_BANDWIDTH_EXPANSION_REDUCTION_
Q16, ( 1 << 16 ) - BWExp1_Q16 );
        BWExp2_Q16 = ( 1 << 16 ) - SKP_SMULWB( SWB_BANDWIDTH_EXPANSION_REDUCTION_
Q16, ( 1 << 16 ) - BWExp2_Q16 );
    }
    /* BWExp1 will be applied after BWExp2, so make it relative */
    BWExp1_Q16 = SKP_DIV32_16( SKP_LSHIFT( BWExp1_Q16, 14 ), SKP_RSHIFT( BWExp2_Q
16, 2 ) );

    /*****************************************/
    /* Compute noise shaping AR coefs and gains */
    /*****************************************/
    sz = ( SKP_int )SKP_SMULBB( SHAPE_LPC_WIN_MS, psEnc->sCmn.fs_kHz );
    for( k = 0; k < NB_SUBFR; k++ ) {
        /* Apply window */
        SKP_Silk_apply_sine_window( x_windowed, x_ptr, 0, SHAPE_LPC_WIN_MS * psEn
c->sCmn.fs_kHz );

        /* Update pointer: next LPC analysis block */
        x_ptr += psEnc->sCmn.frame_length / NB_SUBFR;
```

```
        /* Calculate auto correlation */
        SKP_Silk_autocorr( auto_corr, &scale, x_windowed, sz, psEnc->sCmn.shaping
LPCOrder + 1 );

        /* Add white noise, as a fraction of energy */
        auto_corr[0] = SKP_ADD32( auto_corr[0], SKP_max_32( SKP_SMULWB( SKP_RSHIF
T( auto_corr[ 0 ], 4 ), SHAPE_WHITE_NOISE_FRACTION_Q20 ), 1 ) );

        /* Calculate the reflection coefficients using schur */
        nrg = SKP_Silk_schur64( refl_coef_Q16, auto_corr, psEnc->sCmn.shapingLPCO
rder );

        /* Convert reflection coefficients to prediction coefficients */
        SKP_Silk_k2a_Q16( AR_Q24, refl_coef_Q16, psEnc->sCmn.shapingLPCOrder );

        /* Bandwidth expansion for synthesis filter shaping */
        SKP_Silk_bwexpander_32( AR_Q24, psEnc->sCmn.shapingLPCOrder, BWExp2_Q16 )
;

        /* Make sure to fit in Q13 SKP_int16 */
        SKP_Silk_LPC_fit( &psEncCtrl->AR2_Q13[ k * SHAPE_LPC_ORDER_MAX ], AR_Q24,
 13, psEnc->sCmn.shapingLPCOrder );

        /* Compute noise shaping filter coefficients */
        SKP_memcpy(
            &psEncCtrl->AR1_Q13[ k * SHAPE_LPC_ORDER_MAX ],
            &psEncCtrl->AR2_Q13[ k * SHAPE_LPC_ORDER_MAX ],
            psEnc->sCmn.shapingLPCOrder * sizeof( SKP_int16 ) );

        /* Bandwidth expansion for analysis filter shaping */
        SKP_assert( BWExp1_Q16 <= ( 1 << 16 ) ); // If ever breaking, use LPC_sta
bilize() in these cases to stay within range
        SKP_Silk_bwexpander( &psEncCtrl->AR1_Q13[ k * SHAPE_LPC_ORDER_MAX ], psEn
c->sCmn.shapingLPCOrder, BWExp1_Q16 );

        /* Increase residual energy */
        nrg = SKP_SMLAWB( nrg, SKP_RSHIFT( auto_corr[ 0 ], 8 ), SHAPE_MIN_ENERGY_
RATIO_Q24 );

        Qnrg = -scale;          // range: -12...30
        SKP_assert( Qnrg >= -12 );
        SKP_assert( Qnrg <=  30 );

        /* Make sure that Qnrg is an even number */
        if( Qnrg & 1 ) {
            Qnrg -= 1;
            nrg >>= 1;
        }

        tmp32 = SKP_Silk_SQRT_APPROX( nrg );
        Qnrg >>= 1;             // range: -6...15

        sqrt_nrg[ k ] = tmp32;
        Qnrg_vec[ k ] = Qnrg;

        psEncCtrl->Gains_Q16[ k ] = SKP_LSHIFT_SAT32( tmp32, 16 - Qnrg );
```

```
        /* Ratio of prediction gains, in energy domain */
        SKP_Silk_LPC_inverse_pred_gain_Q13( &pre_nrg_Q30, &psEncCtrl->AR2_Q13[ k
* SHAPE_LPC_ORDER_MAX ], psEnc->sCmn.shapingLPCOrder );
        SKP_Silk_LPC_inverse_pred_gain_Q13( &nrg,         &psEncCtrl->AR1_Q13[ k
* SHAPE_LPC_ORDER_MAX ], psEnc->sCmn.shapingLPCOrder );

        lz = SKP_min_32( SKP_Silk_CLZ32( pre_nrg_Q30 ) - 1, 19 );
        pre_nrg_Q30 = SKP_DIV32( SKP_LSHIFT( pre_nrg_Q30, lz ), SKP_RSHIFT( nrg,
20 - lz ) + 1 ); // Q20
        pre_nrg_Q30 = SKP_RSHIFT( SKP_LSHIFT_SAT32( pre_nrg_Q30, 9 ), 1 );  /* Q2
8 */
        psEncCtrl->GainsPre_Q14[ k ] = ( SKP_int )SKP_Silk_SQRT_APPROX( pre_nrg_Q
30 );
    }

    /*****************/
    /* Gain tweaking */
    /*****************/
    /* Increase gains during low speech activity and put lower limit on gains */
    gain_mult_Q16 = SKP_Silk_log2lin( -SKP_SMLAWB( -16 << 7, SNR_adj_dB_Q7,
     10486 ) ); // 10486_Q16 = 0.16_Q0
    gain_add_Q16  = SKP_Silk_log2lin(  SKP_SMLAWB(  16 << 7, NOISE_FLOOR_dB_Q7,
     10486 ) ); // 10486_Q16 = 0.16_Q0
    tmp32         = SKP_Silk_log2lin(  SKP_SMLAWB(  16 << 7, RELATIVE_MIN_GAIN_dB
_Q7, 10486 ) ); // 10486_Q16 = 0.16_Q0
    tmp32 = SKP_SMULWW( psEnc->avgGain_Q16, tmp32 );
    gain_add_Q16 = SKP_ADD_SAT32( gain_add_Q16, tmp32 );
    SKP_assert( gain_mult_Q16 >= 0 );

    for( k = 0; k < NB_SUBFR; k++ ) {
        psEncCtrl->Gains_Q16[ k ] = SKP_SMULWW( psEncCtrl->Gains_Q16[ k ], gain_m
ult_Q16 );
        SKP_assert( psEncCtrl->Gains_Q16[ k ] >= 0 );
    }

    for( k = 0; k < NB_SUBFR; k++ ) {
        psEncCtrl->Gains_Q16[ k ] = SKP_ADD_POS_SAT32( psEncCtrl->Gains_Q16[ k ],
 gain_add_Q16 );
        psEnc->avgGain_Q16 = SKP_ADD_SAT32(
            psEnc->avgGain_Q16,
            SKP_SMULWB(
                psEncCtrl->Gains_Q16[ k ] - psEnc->avgGain_Q16,
                SKP_RSHIFT_ROUND( SKP_SMULBB( psEnc->speech_activity_Q8, GAIN_SMO
OTHING_COEF_Q10 ), 2 )
            ) );
    }

    /************************************************/
    /* Decrease level during fricatives (de-essing) */
    /************************************************/
    gain_mult_Q16 = ( 1 << 16 ) + SKP_RSHIFT_ROUND( SKP_MLA( INPUT_TILT_Q26, psEn
cCtrl->coding_quality_Q14, HIGH_RATE_INPUT_TILT_Q12 ), 10 );

    if( psEncCtrl->input_tilt_Q15 <= 0 && psEncCtrl->sCmn.sigtype == SIG_TYPE_UNV
OICED ) {
        if( psEnc->sCmn.fs_kHz == 24 ) {
            SKP_int32 essStrength_Q15 = SKP_SMULWW( -psEncCtrl->input_tilt_Q15,
                SKP_SMULBB( psEnc->speech_activity_Q8, ( 1 << 8 ) - psEncCtrl->sp
arseness_Q8 ) );
            tmp32 = SKP_Silk_log2lin( ( 16 << 7 ) - SKP_SMULWB( essStrength_Q15,
                SKP_SMULWB( DE_ESSER_COEF_SWB_dB_Q7, 20972 ) ) ); // 20972_Q17 =
0.16_Q0
```

```
            gain_mult_Q16 = SKP_SMULWW( gain_mult_Q16, tmp32 );
```

```
        } else if( psEnc->sCmn.fs_kHz == 16 ) {
            SKP_int32 essStrength_Q15 = SKP_SMULWW(-psEncCtrl->input_tilt_Q15,
                SKP_SMULBB( psEnc->speech_activity_Q8, ( 1 << 8 ) - psEncCtrl->sp
arseness_Q8 ));
            tmp32 = SKP_Silk_log2lin( ( 16 << 7 ) - SKP_SMULWB( essStrength_Q15,
                SKP_SMULWB( DE_ESSER_COEF_WB_dB_Q7, 20972 ) ) ); // 20972_Q17 = 0
.16_Q0
            gain_mult_Q16 = SKP_SMULWW( gain_mult_Q16, tmp32 );
        } else {
            SKP_assert( psEnc->sCmn.fs_kHz == 12 || psEnc->sCmn.fs_kHz == 8 );
        }
    }

    for( k = 0; k < NB_SUBFR; k++ ) {
        psEncCtrl->GainsPre_Q14[ k ] = SKP_SMULWB( gain_mult_Q16, psEncCtrl->Gain
sPre_Q14[ k ] );
    }

    /*************************************************/
    /* Control low-frequency shaping and noise tilt */
    /*************************************************/
    /* Less low frequency shaping for noisy inputs */
    strength_Q16 = SKP_MUL( LOW_FREQ_SHAPING_Q0, ( 1 << 16 ) + SKP_SMULBB( LOW_QU
ALITY_LOW_FREQ_SHAPING_DECR_Q1, psEncCtrl->input_quality_bands_Q15[ 0 ] - ( 1 <<
15 ) ) );
    if( psEncCtrl->sCmn.sigtype == SIG_TYPE_VOICED ) {
        /* Reduce low frequencies quantization noise for periodic signals, depend
ing on pitch lag */
        /*f = 400; freqz([1, -0.98 + 2e-4 * f], [1, -0.97 + 7e-4 * f], 2^12, Fs);
 axis([0, 1000, -10, 1])*/
        SKP_int fs_kHz_inv = SKP_DIV32_16( 3277, psEnc->sCmn.fs_kHz );      // 0.
2_Q0 = 3277_Q14
        for( k = 0; k < NB_SUBFR; k++ ) {
            b_Q14 = fs_kHz_inv + SKP_DIV32_16( ( 3 << 14 ), psEncCtrl->sCmn.pitch
L[ k ] );
            /* Pack two coefficients in one int32 */
            psEncCtrl->LF_shp_Q14[ k ]  = SKP_LSHIFT( ( 1 << 14 ) - b_Q14 - SKP_S
MULWB( strength_Q16, b_Q14 ), 16 );
            psEncCtrl->LF_shp_Q14[ k ] |= (SKP_uint16)( b_Q14 - ( 1 << 14 ) );
        }
        SKP_assert( HARM_HP_NOISE_COEF_Q24 < ( 1 << 23 ) ); // Guarantees that se
cond argument to SMULWB() is within range of an SKP_int16
        Tilt_Q16 = - HP_NOISE_COEF_Q16 -
            SKP_SMULWB( ( 1 << 16 ) - HP_NOISE_COEF_Q16, SKP_SMULWB( HARM_HP_NOIS
E_COEF_Q24, psEnc->speech_activity_Q8 ) );
    } else {
        b_Q14 = SKP_DIV32_16( 21299, psEnc->sCmn.fs_kHz ); // 1.3_Q0 = 21299_Q14
        /* Pack two coefficients in one int32 */
        psEncCtrl->LF_shp_Q14[ 0 ]  = SKP_LSHIFT( ( 1 << 14 ) - b_Q14 - SKP_SMULW
B( strength_Q16, SKP_SMULWB( 39322, b_Q14 ) ), 16 ); // 0.6_Q0 = 39322_Q16
        psEncCtrl->LF_shp_Q14[ 0 ] |= (SKP_uint16)( b_Q14 - ( 1 << 14 ) );
        for( k = 1; k < NB_SUBFR; k++ ) {
            psEncCtrl->LF_shp_Q14[ k ] = psEncCtrl->LF_shp_Q14[ k - 1 ];
        }
        Tilt_Q16 = -HP_NOISE_COEF_Q16;
    }

    /****************************/
    /* HARMONIC SHAPING CONTROL */
    /****************************/
    /* Control boosting of harmonic frequencies */
```

```
    HarmBoost_Q16 = SKP_SMULWB( SKP_SMULWB( ( 1 << 17 ) - SKP_LSHIFT( psEncCtrl->
coding_quality_Q14, 3 ),
        psEnc->LTPCorr_Q15 ), LOW_RATE_HARMONIC_BOOST_Q16 );

    /* More harmonic boost for noisy input signals */
    HarmBoost_Q16 = SKP_SMLAWB( HarmBoost_Q16,
        ( 1 << 16 ) - SKP_LSHIFT( psEncCtrl->input_quality_Q14, 2 ), LOW_INPUT_QU
ALITY_HARMONIC_BOOST_Q16 );

    if( USE_HARM_SHAPING && psEncCtrl->sCmn.sigtype == SIG_TYPE_VOICED ) {
        /* More harmonic noise shaping for high bitrates or noisy input */
        HarmShapeGain_Q16 = SKP_SMLAWB( HARMONIC_SHAPING_Q16,
                ( 1 << 16 ) - SKP_SMULWB( ( 1 << 18 ) - SKP_LSHIFT( psEncCtrl->co
ding_quality_Q14, 4 ),
                psEncCtrl->input_quality_Q14 ), HIGH_RATE_OR_LOW_QUALITY_HARMONIC
_SHAPING_Q16 );

        /* Less harmonic noise shaping for less periodic signals */
        HarmShapeGain_Q16 = SKP_SMULWB( SKP_LSHIFT( HarmShapeGain_Q16, 1 ),
            SKP_Silk_SQRT_APPROX( SKP_LSHIFT( psEnc->LTPCorr_Q15, 15 ) ) );
    } else {
        HarmShapeGain_Q16 = 0;
    }

    /*************************/
    /* Smooth over subframes */
    /*************************/
    for( k = 0; k < NB_SUBFR; k++ ) {
        psShapeSt->HarmBoost_smth_Q16 =
            SKP_SMLAWB( psShapeSt->HarmBoost_smth_Q16,     HarmBoost_Q16     - ps
ShapeSt->HarmBoost_smth_Q16,     SUBFR_SMTH_COEF_Q16 );
        psShapeSt->HarmShapeGain_smth_Q16 =
            SKP_SMLAWB( psShapeSt->HarmShapeGain_smth_Q16, HarmShapeGain_Q16 - ps
ShapeSt->HarmShapeGain_smth_Q16, SUBFR_SMTH_COEF_Q16 );
        psShapeSt->Tilt_smth_Q16 =
            SKP_SMLAWB( psShapeSt->Tilt_smth_Q16,          Tilt_Q16          - ps
ShapeSt->Tilt_smth_Q16,          SUBFR_SMTH_COEF_Q16 );

        psEncCtrl->HarmBoost_Q14[ k ]     = ( SKP_int )SKP_RSHIFT_ROUND( psShapeS
t->HarmBoost_smth_Q16,     2 );
        psEncCtrl->HarmShapeGain_Q14[ k ] = ( SKP_int )SKP_RSHIFT_ROUND( psShapeS
t->HarmShapeGain_smth_Q16, 2 );
        psEncCtrl->Tilt_Q14[ k ]          = ( SKP_int )SKP_RSHIFT_ROUND( psShapeS
t->Tilt_smth_Q16,          2 );
    }
}
```

A.77. src/SKP_Silk_NSQ.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
```

```
#include "SKP_Silk_main.h"

SKP_INLINE void SKP_Silk_nsq_scale_states(
    SKP_Silk_nsq_state  *NSQ,               /* I/O NSQ state
   */
    const SKP_int16     x[],                /* I input in Q0
   */
    SKP_int32           x_sc_Q10[],         /* O input scaled with 1/Gain
   */
    SKP_int             length,             /* I length of input
   */
    SKP_int16           sLTP[],             /* I re-whitened LTP state in Q0
   */
    SKP_int32           sLTP_Q16[],         /* O LTP state matching scaled input
   */
    SKP_int             subfr,              /* I subframe number
   */
    const SKP_int       LTP_scale_Q14,      /* I
   */
    const SKP_int32     Gains_Q16[ NB_SUBFR ], /* I
   */
    const SKP_int       pitchL[ NB_SUBFR ]  /* I
   */
);

SKP_INLINE void SKP_Silk_noise_shape_quantizer(
    SKP_Silk_nsq_state  *NSQ,               /* I/O  NSQ state
   */
    SKP_int             sigtype,            /* I    Signal type
   */
    const SKP_int32     x_sc_Q10[],         /* I
   */
    SKP_int             q[],                /* O
   */
    SKP_int16           xq[],               /* O
   */
    SKP_int32           sLTP_Q16[],         /* I/O  LTP state
   */
    const SKP_int16     a_Q12[],            /* I    Short term prediction coefs
   */
    const SKP_int16     b_Q14[],            /* I    Long term prediction coefs
   */
```

```
    const SKP_int16     AR_shp_Q13[],       /* I    Noise shaping AR coefs
    */
    SKP_int             lag,                /* I    Pitch lag
    */
    SKP_int32           HarmShapeFIRPacked_Q14, /* I
    */
    SKP_int             Tilt_Q14,           /* I    Spectral tilt
    */
```

```
    SKP_int32           LF_shp_Q14,         /* I
   */
    SKP_int32           Gain_Q16,           /* I
   */
    SKP_int             Lambda_Q10,         /* I
   */
    SKP_int             offset_Q10,         /* I
   */
    SKP_int             length,             /* I    Input length
   */
    SKP_int             shapingLPCOrder,    /* I    Noise shaping AR filter order
   */
    SKP_int             predictLPCOrder     /* I    Prediction filter order
   */
);

void SKP_Silk_NSQ(
    SKP_Silk_encoder_state          *psEncC,                                    /
* I/O  Encoder State                     */
    SKP_Silk_encoder_control        *psEncCtrlC,                                /
* I    Encoder Control                   */
    SKP_Silk_nsq_state              *NSQ,                                       /
* I/O  NSQ state                         */
    const SKP_int16                 x[],                                        /
* I    prefiltered input signal          */
    SKP_int                         q[],                                        /
* O    quantized qulse signal            */
    const SKP_int                   LSFInterpFactor_Q2,                         /
* I    LSF interpolation factor in Q2    */
    const SKP_int16                 PredCoef_Q12[ 2 * MAX_LPC_ORDER ],          /
* I    Short term prediction coefficients  */
    const SKP_int16                 LTPCoef_Q14[ LTP_ORDER * NB_SUBFR ],        /
* I    Long term prediction coefficients   */
    const SKP_int16                 AR2_Q13[ NB_SUBFR * SHAPE_LPC_ORDER_MAX ],  /
* I                                      */
    const SKP_int                   HarmShapeGain_Q14[ NB_SUBFR ],              /
* I                                      */
    const SKP_int                   Tilt_Q14[ NB_SUBFR ],                       /
* I    Spectral tilt                     */
    const SKP_int32                 LF_shp_Q14[ NB_SUBFR ],                     /
* I                                      */
    const SKP_int32                 Gains_Q16[ NB_SUBFR ],                      /
* I                                      */
    const SKP_int                   Lambda_Q10,                                 /
* I                                      */
    const SKP_int                   LTP_scale_Q14                               /
* I    LTP state scaling                 */
)
{
    SKP_int     k, lag, start_idx, subfr_length, LSF_interpolation_flag;
    const SKP_int16 *A_Q12, *B_Q14, *AR_shp_Q13;
    SKP_int16   *pxq;
    SKP_int32   sLTP_Q16[ 2 * MAX_FRAME_LENGTH ];
    SKP_int16   sLTP[     2 * MAX_FRAME_LENGTH ];
    SKP_int32   HarmShapeFIRPacked_Q14;
    SKP_int     offset_Q10;
    SKP_int32   FiltState[ MAX_LPC_ORDER ];
    SKP_int32   x_sc_Q10[ MAX_FRAME_LENGTH / NB_SUBFR ];

    subfr_length = psEncC->frame_length / NB_SUBFR;

    NSQ->rand_seed  =  psEncCtrlC->Seed;
```

```
    /* Set unvoiced lag to the previous one, overwrite later for voiced */
    lag             = NSQ->lagPrev;

    SKP_assert( NSQ->prev_inv_gain_Q16 != 0 );

    offset_Q10 = SKP_Silk_Quantization_Offsets_Q10[ psEncCtrlC->sigtype ][ psEncC
trlC->QuantOffsetType ];

    if( LSFInterpFactor_Q2 == ( 1 << 2 ) ) {
```

```
        LSF_interpolation_flag = 0;
    } else {
        LSF_interpolation_flag = 1;
    }

    /* Setup pointers to start of sub frame */
    NSQ->sLTP_shp_buf_idx = psEncC->frame_length;
    NSQ->sLTP_buf_idx     = psEncC->frame_length;
    pxq                   = &NSQ->xq[ psEncC->frame_length ];
    for( k = 0; k < NB_SUBFR; k++ ) {
        A_Q12      = &PredCoef_Q12[ (( k >> 1 ) | ( 1 - LSF_interpolation_flag ))
 * MAX_LPC_ORDER ];
        B_Q14      = &LTPCoef_Q14[ k * LTP_ORDER ];
        AR_shp_Q13 = &AR2_Q13[     k * SHAPE_LPC_ORDER_MAX ];

        /* Noise shape parameters */
        SKP_assert( HarmShapeGain_Q14[ k ] >= 0 );
        HarmShapeFIRPacked_Q14  =                       SKP_RSHIFT( HarmShapeGai
n_Q14[ k ], 2 );
        HarmShapeFIRPacked_Q14 |= SKP_LSHIFT( ( SKP_int32 )SKP_RSHIFT( HarmShapeG
ain_Q14[ k ], 1 ), 16 );

        if( psEncCtrlC->sigtype == SIG_TYPE_VOICED ) {
            /* Voiced */
            lag = psEncCtrlC->pitchL[ k ];

            NSQ->rewhite_flag = 0;
            /* Re-whitening */
            if( ( k & ( 3 - SKP_LSHIFT( LSF_interpolation_flag, 1 ) ) ) == 0 ) {
                /* Rewhiten with new A coefs */

                start_idx = psEncC->frame_length - lag - psEncC->predictLPCOrder
- LTP_ORDER / 2;
                start_idx = SKP_LIMIT( start_idx, 0, psEncC->frame_length - psEnc
C->predictLPCOrder ); /* Limit */

                SKP_memset( FiltState, 0, psEncC->predictLPCOrder * sizeof( SKP_i
nt32 ) );
                SKP_Silk_MA_Prediction( &NSQ->xq[ start_idx + k * ( psEncC->frame
_length >> 2 ) ],
                    A_Q12, FiltState, sLTP + start_idx, psEncC->frame_length - st
art_idx, psEncC->predictLPCOrder );

                NSQ->rewhite_flag = 1;
                NSQ->sLTP_buf_idx = psEncC->frame_length;
            }
        }

        SKP_Silk_nsq_scale_states( NSQ, x, x_sc_Q10, psEncC->subfr_length, sLTP,
            sLTP_Q16, k, LTP_scale_Q14, Gains_Q16, psEncCtrlC->pitchL );

        SKP_Silk_noise_shape_quantizer( NSQ, psEncCtrlC->sigtype, x_sc_Q10, q, px
q, sLTP_Q16, A_Q12, B_Q14,
            AR_shp_Q13, lag, HarmShapeFIRPacked_Q14, Tilt_Q14[ k ], LF_shp_Q14[ k
 ], Gains_Q16[ k ], Lambda_Q10,
            offset_Q10, psEncC->subfr_length, psEncC->shapingLPCOrder, psEncC->pr
edictLPCOrder
        );
```

```
        x          += psEncC->subfr_length;
        q          += psEncC->subfr_length;
        pxq        += psEncC->subfr_length;
    }

    /* Save scalars for this layer */
    NSQ->sLF_AR_shp_Q12             = NSQ->sLF_AR_shp_Q12;
    NSQ->prev_inv_gain_Q16          = NSQ->prev_inv_gain_Q16;
    NSQ->lagPrev                        = psEncCtrlC->pitchL[ NB_SUBFR - 1 ];
    /* Save quantized speech and noise shaping signals */
    SKP_memcpy( NSQ->xq,           &NSQ->xq[           psEncC->frame_length ], ps
EncC->frame_length * sizeof( SKP_int16 ) );
    SKP_memcpy( NSQ->sLTP_shp_Q10, &NSQ->sLTP_shp_Q10[ psEncC->frame_length ], ps
EncC->frame_length * sizeof( SKP_int32 ) );

}

/***********************************/
/* SKP_Silk_noise_shape_quantizer  */
/***********************************/
SKP_INLINE void SKP_Silk_noise_shape_quantizer(
    SKP_Silk_nsq_state  *NSQ,               /* I/O  NSQ state
   */
    SKP_int             sigtype,            /* I    Signal type
   */
    const SKP_int32     x_sc_Q10[],         /* I
   */
    SKP_int             q[],                /* O
   */
    SKP_int16           xq[],               /* O
   */
    SKP_int32           sLTP_Q16[],         /* I/O  LTP state
   */
    const SKP_int16     a_Q12[],            /* I    Short term prediction coefs
   */
    const SKP_int16     b_Q14[],            /* I    Long term prediction coefs
   */
    const SKP_int16     AR_shp_Q13[],       /* I    Noise shaping AR coefs
   */
    SKP_int             lag,                /* I    Pitch lag
   */
    SKP_int32           HarmShapeFIRPacked_Q14, /* I
   */
    SKP_int             Tilt_Q14,           /* I    Spectral tilt
   */
    SKP_int32           LF_shp_Q14,         /* I
   */
    SKP_int32           Gain_Q16,           /* I
   */
    SKP_int             Lambda_Q10,         /* I
   */
    SKP_int             offset_Q10,         /* I
   */
    SKP_int             length,             /* I    Input length
   */
    SKP_int             shapingLPCOrder,    /* I    Noise shaping AR filter order
   */
    SKP_int             predictLPCOrder     /* I    Prediction filter order
   */
)
{
    SKP_int     i, j;
```

```
    SKP_int32   LTP_pred_Q14, LPC_pred_Q10, n_AR_Q10, n_LTP_Q14;
    SKP_int32   n_LF_Q10, r_Q10, q_Q0, q_Q10;
    SKP_int32   thr1_Q10, thr2_Q10, thr3_Q10;
    SKP_int32   Atmp, dither;
    SKP_int32   exc_Q10, LPC_exc_Q10, xq_Q10;
    SKP_int32   tmp, sLF_AR_shp_Q10;
    SKP_int32   *psLPC_Q14;
```

```
    SKP_int32   *shp_lag_ptr, *pred_lag_ptr;
    SKP_int32   a_Q12_tmp[ MAX_LPC_ORDER / 2 ], AR_shp_Q13_tmp[ MAX_LPC_ORDER / 2
 ];

    shp_lag_ptr  = &NSQ->sLTP_shp_Q10[ NSQ->sLTP_shp_buf_idx - lag + HARM_SHAPE_F
IR_TAPS / 2 ];
    pred_lag_ptr = &sLTP_Q16[ NSQ->sLTP_buf_idx - lag + LTP_ORDER / 2 ];

    /* Setup short term AR state */
    psLPC_Q14     = &NSQ->sLPC_Q14[ MAX_LPC_ORDER - 1 ];

    /* Quantization thresholds */
    thr1_Q10 = SKP_SUB_RSHIFT32( -1536, Lambda_Q10, 1);
    thr2_Q10 = SKP_SUB_RSHIFT32( -512,  Lambda_Q10, 1);
    thr2_Q10 = SKP_ADD_RSHIFT32( thr2_Q10, SKP_SMULBB( offset_Q10, Lambda_Q10 ),
10 );
    thr3_Q10 = SKP_ADD_RSHIFT32(  512,  Lambda_Q10, 1);

    /* Preload LPC coeficients to array on stack. Gives small performance gain */
    SKP_memcpy( a_Q12_tmp, a_Q12, predictLPCOrder * sizeof( SKP_int16 ) );
    SKP_memcpy( AR_shp_Q13_tmp, AR_shp_Q13, shapingLPCOrder * sizeof( SKP_int16 )
 );

    for( i = 0; i < length; i++ ) {
        /* Generate dither */
        NSQ->rand_seed = SKP_RAND( NSQ->rand_seed );

        /* dither = rand_seed < 0 ? 0xFFFFFFFF : 0; */
        dither = SKP_RSHIFT( NSQ->rand_seed, 31 );

        /* Short-term prediction */
        SKP_assert( ( predictLPCOrder  & 1 ) == 0 );    /* check that order is ev
en */
        SKP_assert( ( (SKP_int64)a_Q12 & 3 ) == 0 );    /* check that array start
s at 4-byte aligned address */
        SKP_assert( predictLPCOrder >= 10 );            /* check that unrolling w
orks */

        /* NOTE: the code below loads two int16 values in an int32, and multiplie
s each using the   */
        /* SMLAWB and SMLAWT instructions. On a big-endian CPU the two int16 vari
ables would be     */
        /* loaded in reverse order and the code will give the wrong result. In th
at case swapping   */
        /* the SMLAWB and SMLAWT instructions should solve the problem.
                   */
        /* Partially unrolled */
        Atmp = a_Q12_tmp[ 0 ];      /* read two coefficients at once */
        LPC_pred_Q10 = SKP_SMULWB(               psLPC_Q14[ 0  ], Atmp );
        LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -1 ], Atmp );
        Atmp = a_Q12_tmp[ 1 ];
        LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psLPC_Q14[ -2 ], Atmp );
        LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -3 ], Atmp );
        Atmp = a_Q12_tmp[ 2 ];
        LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psLPC_Q14[ -4 ], Atmp );
        LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -5 ], Atmp );
        Atmp = a_Q12_tmp[ 3 ];
        LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psLPC_Q14[ -6 ], Atmp );
        LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -7 ], Atmp );
```

```
        Atmp = a_Q12_tmp[ 4 ];
        LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psLPC_Q14[ -8 ], Atmp );
        LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -9 ], Atmp );
        for( j = 10; j < predictLPCOrder; j += 2 ) {
            Atmp = a_Q12_tmp[ j >> 1 ];     /* read two coefficients at once */
            LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psLPC_Q14[ -j     ], Atmp );
            LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -j - 1 ], Atmp );
        }

        /* Long-term prediction */
        if( sigtype == SIG_TYPE_VOICED ) {
            /* Unrolled loop */
            LTP_pred_Q14 = SKP_SMULWB(               pred_lag_ptr[  0 ], b_Q14[ 0
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -1 ], b_Q14[ 1
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -2 ], b_Q14[ 2
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -3 ], b_Q14[ 3
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -4 ], b_Q14[ 4
 ] );
            pred_lag_ptr++;
        } else {
            LTP_pred_Q14 = 0;
        }

        /* Noise shape feedback */
        SKP_assert( ( shapingLPCOrder       & 1 ) == 0 );   /* check that order i
s even */
        SKP_assert( ( (SKP_int64)AR_shp_Q13 & 3 ) == 0 );   /* check that array s
tarts at 4-byte aligned address */
        SKP_assert( shapingLPCOrder >= 12 );                /* check that unrolli
ng works */

        /* Partially unrolled */
        Atmp = AR_shp_Q13_tmp[ 0 ];     /* read two coefficients at once */
        n_AR_Q10 = SKP_SMULWB(           psLPC_Q14[ 0  ], Atmp );
        n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -1 ], Atmp );
        Atmp = AR_shp_Q13_tmp[ 1 ];
        n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -2 ], Atmp );
        n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -3 ], Atmp );
        Atmp = AR_shp_Q13_tmp[ 2 ];
        n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -4 ], Atmp );
        n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -5 ], Atmp );
        Atmp = AR_shp_Q13_tmp[ 3 ];
        n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -6 ], Atmp );
        n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -7 ], Atmp );
        Atmp = AR_shp_Q13_tmp[ 4 ];
        n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -8 ], Atmp );
        n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -9 ], Atmp );
        Atmp = AR_shp_Q13_tmp[ 5 ];
        n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -10 ], Atmp );
        n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -11 ], Atmp );
        for( j = 12; j < shapingLPCOrder; j += 2 ) {
            Atmp = AR_shp_Q13_tmp[ j >> 1 ];        /* read two coefficients at o
nce */
```

```
            n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -j     ], Atmp );
            n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -j - 1 ], Atmp );
        }
        n_AR_Q10 = SKP_RSHIFT( n_AR_Q10, 1 );   /* Q11 -> Q10 */
        n_AR_Q10  = SKP_SMLAWB( n_AR_Q10, NSQ->sLF_AR_shp_Q12, Tilt_Q14 );

        n_LF_Q10   = SKP_LSHIFT( SKP_SMULWB( NSQ->sLTP_shp_Q10[ NSQ->sLTP_shp_buf
_idx - 1 ], LF_shp_Q14 ), 2 );
        n_LF_Q10   = SKP_SMLAWT( n_LF_Q10, NSQ->sLF_AR_shp_Q12, LF_shp_Q14 );

        SKP_assert( lag > 0 || sigtype == SIG_TYPE_UNVOICED);

        /* Long-term shaping */
        if( lag > 0 ) {
            /* Symmetric, packed FIR coefficients */
            n_LTP_Q14 = SKP_SMULWB( SKP_ADD32( shp_lag_ptr[ 0 ], shp_lag_ptr[ -2
] ), HarmShapeFIRPacked_Q14 );
            n_LTP_Q14 = SKP_SMLAWT( n_LTP_Q14, shp_lag_ptr[ -1 ],
     HarmShapeFIRPacked_Q14 );
            shp_lag_ptr++;
            n_LTP_Q14 = SKP_LSHIFT( n_LTP_Q14, 6 );
        } else {
            n_LTP_Q14 = 0;
        }

        /* Input minus prediction plus noise feedback  */
        //r = x[ i ] - LTP_pred - LPC_pred + n_AR + n_Tilt + n_LF + n_LTP;
        tmp   = SKP_SUB32( LTP_pred_Q14, n_LTP_Q14 );                       /* Ad
d Q14 stuff */
        tmp   = SKP_RSHIFT_ROUND( tmp, 4 );                                 /* ro
und to Q10  */
        tmp   = SKP_ADD32( tmp, LPC_pred_Q10 );                             /* ad
d Q10 stuff */
        tmp   = SKP_SUB32( tmp, n_AR_Q10 );                                 /* su
btract Q10 stuff */
        tmp   = SKP_SUB32( tmp, n_LF_Q10 );                                 /* su
btract Q10 stuff */
        r_Q10 = SKP_SUB32( x_sc_Q10[ i ], tmp );


        /* Flip sign depending on dither */
        r_Q10 = ( r_Q10 ^ dither ) - dither;
        r_Q10 = SKP_SUB32( r_Q10, offset_Q10 );
        r_Q10 = SKP_LIMIT( r_Q10, -64 << 10, 64 << 10 );

        /* Quantize */
        if( r_Q10 < thr1_Q10 ) {
            q_Q0 = SKP_RSHIFT_ROUND( SKP_ADD_RSHIFT32( r_Q10, Lambda_Q10, 1 ), 10
 );
            q_Q10 = SKP_LSHIFT( q_Q0, 10 );
        } else if( r_Q10 < thr2_Q10 ) {
            q_Q0 = -1;
            q_Q10 = -1024;
        } else if( r_Q10 > thr3_Q10 ) {
            q_Q0 = SKP_RSHIFT_ROUND( SKP_SUB_RSHIFT32( r_Q10, Lambda_Q10, 1 ), 10
 );
            q_Q10 = SKP_LSHIFT( q_Q0, 10 );
        } else {
```

```
            q_Q0 = 0;
            q_Q10 = 0;
        }
        q[ i ] = q_Q0;

        /* Excitation */
        exc_Q10 = SKP_ADD32( q_Q10, offset_Q10 );
        exc_Q10 = ( exc_Q10 ^ dither ) - dither;

        /* Add predictions */
        LPC_exc_Q10 = SKP_ADD32( exc_Q10, SKP_RSHIFT_ROUND( LTP_pred_Q14, 4 ) );
        xq_Q10      = SKP_ADD32( LPC_exc_Q10, LPC_pred_Q10 );

        /* Scale XQ back to normal level before saving */
        xq[ i ] = ( SKP_int16 )SKP_SAT16( SKP_RSHIFT_ROUND( SKP_SMULWW( xq_Q10, G
ain_Q16 ), 10 ) );


        /* Update states */
        psLPC_Q14++;
        *psLPC_Q14 = SKP_LSHIFT( xq_Q10, 4 );
        sLF_AR_shp_Q10 = SKP_SUB32( xq_Q10, n_AR_Q10 );
        NSQ->sLF_AR_shp_Q12 = SKP_LSHIFT( sLF_AR_shp_Q10, 2 );

        NSQ->sLTP_shp_Q10[ NSQ->sLTP_shp_buf_idx ] = SKP_SUB32( sLF_AR_shp_Q10, n
_LF_Q10 );
        sLTP_Q16[NSQ->sLTP_buf_idx] = SKP_LSHIFT( LPC_exc_Q10, 6 );
        NSQ->sLTP_shp_buf_idx++;
        NSQ->sLTP_buf_idx++;

        /* Make dither dependent on quantized signal */
        NSQ->rand_seed += q[ i ];
    }
    /* Update LPC synth buffer */
    SKP_memcpy( NSQ->sLPC_Q14, &NSQ->sLPC_Q14[ length ], MAX_LPC_ORDER * sizeof(
SKP_int32 ) );
}

SKP_INLINE void SKP_Silk_nsq_scale_states(
    SKP_Silk_nsq_state  *NSQ,               /* I/O NSQ state
   */
    const SKP_int16     x[],                /* I input in Q0
   */
    SKP_int32           x_sc_Q10[],         /* O input scaled with 1/Gain
   */
    SKP_int             length,             /* I length of input
   */
    SKP_int16           sLTP[],             /* I re-whitened LTP state in Q0
   */
    SKP_int32           sLTP_Q16[],         /* O LTP state matching scaled input
   */
    SKP_int             subfr,              /* I subframe number
   */
    const SKP_int       LTP_scale_Q14,      /* I
   */
    const SKP_int32     Gains_Q16[ NB_SUBFR ], /* I
   */
    const SKP_int       pitchL[ NB_SUBFR ]  /* I
   */
)
{
```

```
    SKP_int   i, scale_length, lag;
    SKP_int32 inv_gain_Q16, gain_adj_Q16, inv_gain_Q32;

    inv_gain_Q16 = SKP_DIV32( SKP_int32_MAX, SKP_RSHIFT( Gains_Q16[ subfr ], 1) )
;
    inv_gain_Q16 = SKP_min( inv_gain_Q16, SKP_int16_MAX );
    lag          = pitchL[ subfr ];

    /* After rewhitening the LTP state is un-scaled */
    if( NSQ->rewhite_flag ) {
        inv_gain_Q32 = SKP_LSHIFT( inv_gain_Q16, 16 );
        if( subfr == 0 ) {
            /* Do LTP downscaling */
            inv_gain_Q32 = SKP_LSHIFT( SKP_SMULWB( inv_gain_Q32, LTP_scale_Q14 ),
 2 );
        }
        for( i = NSQ->sLTP_buf_idx - lag - LTP_ORDER / 2; i < NSQ->sLTP_buf_idx;
i++ ) {
            sLTP_Q16[ i ] = SKP_SMULWB( inv_gain_Q32, sLTP[ i ] );
        }
    }

    /* Prepare for Worst case. Next frame starts with max lag voiced */
    scale_length = length * NB_SUBFR;                                          /
* approx max lag */
    scale_length = scale_length - SKP_SMULBB( NB_SUBFR - (subfr + 1), length ); /
* subtract samples that will be too old in next frame */
    scale_length = SKP_max_int( scale_length, lag + LTP_ORDER );               /
* make sure to scale whole pitch period if voiced */

    /* Adjust for changing gain */
    if( inv_gain_Q16 != NSQ->prev_inv_gain_Q16 ) {
        gain_adj_Q16 =  SKP_DIV32_varQ( inv_gain_Q16, NSQ->prev_inv_gain_Q16, 16
);

        for( i = NSQ->sLTP_shp_buf_idx - scale_length; i < NSQ->sLTP_shp_buf_idx;
 i++ ) {
            NSQ->sLTP_shp_Q10[ i ] = SKP_SMULWW( gain_adj_Q16, NSQ->sLTP_shp_Q10[
 i ] );
        }

        /* Scale LTP predict state */
        if( NSQ->rewhite_flag == 0 ) {
            for( i = NSQ->sLTP_buf_idx - lag - LTP_ORDER / 2; i < NSQ->sLTP_buf_i
dx; i++ ) {
                sLTP_Q16[ i ] = SKP_SMULWW( gain_adj_Q16, sLTP_Q16[ i ] );
            }
        }
        NSQ->sLF_AR_shp_Q12 = SKP_SMULWW( gain_adj_Q16, NSQ->sLF_AR_shp_Q12 );

        /* scale short term state */
        for( i = 0; i < MAX_LPC_ORDER; i++ ) {
            NSQ->sLPC_Q14[ i ] = SKP_SMULWW( gain_adj_Q16, NSQ->sLPC_Q14[ i ] );
        }
    }

    /* Scale input */
    for( i = 0; i < length; i++ ) {
```

```
        x_sc_Q10[ i ] = SKP_RSHIFT( SKP_SMULBB( x[ i ], ( SKP_int16 )inv_gain_Q16
 ), 6 );
    }

    /* save inv_gain */
    SKP_assert( inv_gain_Q16 != 0 );
    NSQ->prev_inv_gain_Q16 = inv_gain_Q16;
}
```

A.78. src/SKP_Silk_NSQ_del_dec.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

typedef struct {
    SKP_int   RandState[ DECISION_DELAY ];
    SKP_int32 Q_Q10[     DECISION_DELAY ];
    SKP_int32 Xq_Q10[    DECISION_DELAY ];
    SKP_int32 Pred_Q16[  DECISION_DELAY ];
    SKP_int32 Shape_Q10[ DECISION_DELAY ];
    SKP_int32 Gain_Q16[  DECISION_DELAY ];
```

```
    SKP_int32 sLPC_Q14[ MAX_FRAME_LENGTH / NB_SUBFR + NSQ_LPC_BUF_LENGTH ];
    SKP_int32 LF_AR_Q12;
    SKP_int32 Seed;
    SKP_int32 SeedInit;
    SKP_int32 RD_Q10;
} NSQ_del_dec_struct;

typedef struct {
    SKP_int32 Q_Q10;
    SKP_int32 RD_Q10;
    SKP_int32 xq_Q14;
    SKP_int32 LF_AR_Q12;
    SKP_int32 sLTP_shp_Q10;
    SKP_int32 LPC_exc_Q16;
} NSQ_sample_struct;

SKP_INLINE void SKP_Silk_copy_del_dec_state(
    NSQ_del_dec_struct  *DD_dst,                /* I    Dst del dec state
           */
    NSQ_del_dec_struct  *DD_src,                /* I    Src del dec state
           */
    SKP_int             LPC_state_idx           /* I    Index to LPC buffer
           */
);

SKP_INLINE void SKP_Silk_nsq_del_dec_scale_states(
    SKP_Silk_nsq_state  *NSQ,                   /* I/O  NSQ state
           */
    NSQ_del_dec_struct  psDelDec[],             /* I/O  Delayed decision states
           */
    const SKP_int16     x[],                    /* I    Input in Q0
           */
    SKP_int32           x_sc_Q10[],             /* O    Input scaled with 1/Gain
in Q10     */
    SKP_int             length,                 /* I    Length of input
           */
    SKP_int16           sLTP[],                 /* I    Re-whitened LTP state in
Q0         */
    SKP_int32           sLTP_Q16[],             /* O    LTP state matching scaled
 input     */
    SKP_int             subfr,                  /* I    Subframe number
           */
    SKP_int             nStatesDelayedDecision, /* I    Number of del dec states
           */
    SKP_int             smpl_buf_idx,           /* I    Index to newest samples i
n buffers  */
    const SKP_int       LTP_scale_Q14,          /* I    LTP state scaling
           */
    const SKP_int32     Gains_Q16[ NB_SUBFR ],  /* I
           */
    const SKP_int       pitchL[ NB_SUBFR ]      /* I    Pitch lag
           */
);

/***************************************/
/* Noise shape quantizer for one subframe */
/***************************************/
SKP_INLINE void SKP_Silk_noise_shape_quantizer_del_dec(
    SKP_Silk_nsq_state  *NSQ,                   /* I/O  NSQ state
           */
    NSQ_del_dec_struct  psDelDec[],             /* I/O  Delayed decision states
           */
```

```
    SKP_int             sigtype,                /* I    Signal type
           */
    const SKP_int32     x_Q10[],                /* I
           */
    SKP_int             q[],                    /* O
           */
    SKP_int16           xq[],                   /* O
           */
```

```
    SKP_int32           sLTP_Q16[],             /* I/O  LTP filter state
            */
    const SKP_int16     a_Q12[],                /* I    Short term prediction coe
fs          */
    const SKP_int16     b_Q14[],                /* I    Long term prediction coef
s           */
    const SKP_int16     AR_shp_Q13[],           /* I    Noise shaping coefs
            */
    SKP_int             lag,                    /* I    Pitch lag
            */
    SKP_int32           HarmShapeFIRPacked_Q14, /* I
            */
    SKP_int             Tilt_Q14,               /* I    Spectral tilt
            */
    SKP_int32           LF_shp_Q14,             /* I
            */
    SKP_int32           Gain_Q16,               /* I
            */
    SKP_int             Lambda_Q10,             /* I
            */
    SKP_int             offset_Q10,             /* I
            */
    SKP_int             length,                 /* I    Input length
            */
    SKP_int             subfr,                  /* I    Subframe number
            */
    SKP_int             shapingLPCOrder,        /* I    Shaping LPC filter order
            */
    SKP_int             predictLPCOrder,        /* I    Prediction LPC filter ord
er          */
    SKP_int             nStatesDelayedDecision, /* I    Number of states in decis
ion tree    */
    SKP_int             *smpl_buf_idx,          /* I    Index to newest samples i
n buffers   */
    SKP_int             decisionDelay           /* I
            */
);

void SKP_Silk_NSQ_del_dec(
    SKP_Silk_encoder_state          *psEncC,                                    /
* I/O  Encoder State                     */
    SKP_Silk_encoder_control        *psEncCtrlC,                                /
* I    Encoder Control                   */
    SKP_Silk_nsq_state              *NSQ,                                       /
* I/O  NSQ state                         */
    const SKP_int16                 x[],                                        /
* I    Prefiltered input signal          */
    SKP_int                         q[],                                        /
* O    Quantized pulse signal            */
    const SKP_int                   LSFInterpFactor_Q2,                         /
* I    LSF interpolation factor in Q2    */
    const SKP_int16                 PredCoef_Q12[ 2 * MAX_LPC_ORDER ],          /
* I    Prediction coefs                  */
    const SKP_int16                 LTPCoef_Q14[ LTP_ORDER * NB_SUBFR ],        /
* I    LT prediction coefs               */
    const SKP_int16                 AR2_Q13[ NB_SUBFR * SHAPE_LPC_ORDER_MAX ],  /
* I                                      */
    const SKP_int                   HarmShapeGain_Q14[ NB_SUBFR ],              /
* I                                      */
    const SKP_int                   Tilt_Q14[ NB_SUBFR ],                       /
* I    Spectral tilt                     */
    const SKP_int32                 LF_shp_Q14[ NB_SUBFR ],                     /
```

```
* I                                          */
    const SKP_int32                 Gains_Q16[ NB_SUBFR ],                      /
* I                                          */
    const SKP_int                   Lambda_Q10,                                 /
* I                                          */
    const SKP_int                   LTP_scale_Q14                               /
* I    LTP state scaling                     */
)
{
    SKP_int     i, k, lag, start_idx, LSF_interpolation_flag, Winner_ind, subfr;
    SKP_int     last_smple_idx, smpl_buf_idx, decisionDelay, subfr_length;
    const SKP_int16 *A_Q12, *B_Q14, *AR_shp_Q13;
    SKP_int16   *pxq;
    SKP_int32   sLTP_Q16[ 2 * MAX_FRAME_LENGTH ];
    SKP_int16   sLTP[     2 * MAX_FRAME_LENGTH ];
    SKP_int32   HarmShapeFIRPacked_Q14;
    SKP_int     offset_Q10;
    SKP_int32   FiltState[ MAX_LPC_ORDER ], RDmin_Q10;
    SKP_int32   x_sc_Q10[ MAX_FRAME_LENGTH / NB_SUBFR ];
```

```
    NSQ_del_dec_struct psDelDec[ DEL_DEC_STATES_MAX ];
    NSQ_del_dec_struct *psDD;

    subfr_length = psEncC->frame_length / NB_SUBFR;

    /* Set unvoiced lag to the previous one, overwrite later for voiced */
    lag = NSQ->lagPrev;

    SKP_assert( NSQ->prev_inv_gain_Q16 != 0 );

    /* Initialize delayed decision states */
    SKP_memset( psDelDec, 0, psEncC->nStatesDelayedDecision * sizeof( NSQ_del_dec
_struct ) );
    for( k = 0; k < psEncC->nStatesDelayedDecision; k++ ) {
        psDD                = &psDelDec[ k ];
        psDD->Seed          = ( k + psEncCtrlC->Seed ) & 3;
        psDD->SeedInit      = psDD->Seed;
        psDD->RD_Q10        = 0;
        psDD->LF_AR_Q12     = NSQ->sLF_AR_shp_Q12;
        psDD->Shape_Q10[ 0 ] = NSQ->sLTP_shp_Q10[ psEncC->frame_length - 1 ];
        SKP_memcpy( psDD->sLPC_Q14, NSQ->sLPC_Q14, NSQ_LPC_BUF_LENGTH * sizeof( S
KP_int32 ) );
    }

    offset_Q10   = SKP_Silk_Quantization_Offsets_Q10[ psEncCtrlC->sigtype ][ psEn
cCtrlC->QuantOffsetType ];
    smpl_buf_idx = 0; /* index of oldest samples */

    decisionDelay = SKP_min_int( DECISION_DELAY, subfr_length );
    /* For voiced frames limit the decision delay to lower than the pitch lag */
    if( psEncCtrlC->sigtype == SIG_TYPE_VOICED ) {
        for( k = 0; k < NB_SUBFR; k++ ) {
            decisionDelay = SKP_min_int( decisionDelay, psEncCtrlC->pitchL[ k ] -
 LTP_ORDER / 2 - 1 );
        }
    }

    if( LSFInterpFactor_Q2 == ( 1 << 2 ) ) {
        LSF_interpolation_flag = 0;
    } else {
        LSF_interpolation_flag = 1;
    }

    /* Setup pointers to start of sub frame */
    pxq                   = &NSQ->xq[ psEncC->frame_length ];
    NSQ->sLTP_shp_buf_idx = psEncC->frame_length;
    NSQ->sLTP_buf_idx     = psEncC->frame_length;
    subfr = 0;
    for( k = 0; k < NB_SUBFR; k++ ) {
        A_Q12      = &PredCoef_Q12[ ( ( k >> 1 ) | ( 1 - LSF_interpolation_flag )
 ) * MAX_LPC_ORDER ];
        B_Q14      = &LTPCoef_Q14[ k * LTP_ORDER           ];
        AR_shp_Q13 = &AR2_Q13[     k * SHAPE_LPC_ORDER_MAX ];
```

```
        NSQ->rewhite_flag = 0;
        if( psEncCtrlC->sigtype == SIG_TYPE_VOICED ) {
            /* Voiced */
            lag = psEncCtrlC->pitchL[ k ];

            /* Re-whitening */
            if( ( k & ( 3 - SKP_LSHIFT( LSF_interpolation_flag, 1 ) ) ) == 0 ) {
                if( k == 2 ) {
                    /* RESET DELAYED DECISIONS */
                    /* Find winner */
                    RDmin_Q10 = psDelDec[ 0 ].RD_Q10;
                    Winner_ind = 0;
                    for( i = 1; i < psEncC->nStatesDelayedDecision; i++ ) {
                        if( psDelDec[ i ].RD_Q10 < RDmin_Q10 ) {
                            RDmin_Q10 = psDelDec[ i ].RD_Q10;
                            Winner_ind = i;
                        }
                    }
                    for( i = 0; i < psEncC->nStatesDelayedDecision; i++ ) {
                        if( i != Winner_ind ) {
                            psDelDec[ i ].RD_Q10 += ( SKP_int32_MAX >> 4 );
                            SKP_assert( psDelDec[ i ].RD_Q10 >= 0 );
                        }
                    }

                    /* Copy final part of signals from winner state to output and
 long-term filter states */
                    psDD = &psDelDec[ Winner_ind ];
                    last_smple_idx = smpl_buf_idx + decisionDelay;
                    for( i = 0; i < decisionDelay; i++ ) {
                        last_smple_idx = ( last_smple_idx - 1 ) & DECISION_DELAY_
MASK;
                        q[   i - decisionDelay ] = ( SKP_int )SKP_RSHIFT( psDD->Q
_Q10[ last_smple_idx ], 10 );
                        pxq[ i - decisionDelay ] = ( SKP_int16 )SKP_SAT16( SKP_RS
HIFT_ROUND(
                            SKP_SMULWW( psDD->Xq_Q10[ last_smple_idx ],
                            psDD->Gain_Q16[ last_smple_idx ] ), 10 ) );
                        NSQ->sLTP_shp_Q10[ NSQ->sLTP_shp_buf_idx - decisionDelay
+ i ] = psDD->Shape_Q10[ last_smple_idx ];
                    }

                    subfr = 0;
                }

                /* Rewhiten with new A coefs */
                start_idx = psEncC->frame_length - lag - psEncC->predictLPCOrder
- LTP_ORDER / 2;
                start_idx = SKP_LIMIT( start_idx, 0, psEncC->frame_length - psEnc
C->predictLPCOrder );

                SKP_memset( FiltState, 0, psEncC->predictLPCOrder * sizeof( SKP_i
nt32 ) );
                SKP_Silk_MA_Prediction( &NSQ->xq[ start_idx + k * psEncC->subfr_l
ength ],
                    A_Q12, FiltState, sLTP + start_idx, psEncC->frame_length - st
art_idx, psEncC->predictLPCOrder );
```

```
                NSQ->sLTP_buf_idx = psEncC->frame_length;
                NSQ->rewhite_flag = 1;
            }
        }

        /* Noise shape parameters */
        SKP_assert( HarmShapeGain_Q14[ k ] >= 0 );
        HarmShapeFIRPacked_Q14  =                       SKP_RSHIFT( HarmShapeGai
n_Q14[ k ], 2 );
        HarmShapeFIRPacked_Q14 |= SKP_LSHIFT( ( SKP_int32 )SKP_RSHIFT( HarmShapeG
ain_Q14[ k ], 1 ), 16 );

        SKP_Silk_nsq_del_dec_scale_states( NSQ, psDelDec, x, x_sc_Q10,
            subfr_length, sLTP, sLTP_Q16, k, psEncC->nStatesDelayedDecision, smpl
_buf_idx,
            LTP_scale_Q14, Gains_Q16, psEncCtrlC->pitchL );

        SKP_Silk_noise_shape_quantizer_del_dec( NSQ, psDelDec, psEncCtrlC->sigtyp
e, x_sc_Q10, q, pxq, sLTP_Q16,
            A_Q12, B_Q14, AR_shp_Q13, lag, HarmShapeFIRPacked_Q14, Tilt_Q14[ k ],
 LF_shp_Q14[ k ], Gains_Q16[ k ],
            Lambda_Q10, offset_Q10, psEncC->subfr_length, subfr++, psEncC->shapin
gLPCOrder, psEncC->predictLPCOrder,
            psEncC->nStatesDelayedDecision, &smpl_buf_idx, decisionDelay
        );

        x   += psEncC->subfr_length;
        q   += psEncC->subfr_length;
        pxq += psEncC->subfr_length;
    }

    /* Find winner */
    RDmin_Q10 = psDelDec[ 0 ].RD_Q10;
    Winner_ind = 0;
    for( k = 1; k < psEncC->nStatesDelayedDecision; k++ ) {
        if( psDelDec[ k ].RD_Q10 < RDmin_Q10 ) {
            RDmin_Q10 = psDelDec[ k ].RD_Q10;
            Winner_ind = k;
        }
    }

    /* Copy final part of signals from winner state to output and long-term filte
r states */
    psDD = &psDelDec[ Winner_ind ];
    psEncCtrlC->Seed = psDD->SeedInit;
    last_smple_idx = smpl_buf_idx + decisionDelay;
    for( i = 0; i < decisionDelay; i++ ) {
        last_smple_idx = ( last_smple_idx - 1 ) & DECISION_DELAY_MASK;
        q[   i - decisionDelay ] = ( SKP_int )SKP_RSHIFT( psDD->Q_Q10[ last_smple
_idx ], 10 );
        pxq[ i - decisionDelay ] = ( SKP_int16 )SKP_SAT16( SKP_RSHIFT_ROUND(
            SKP_SMULWW( psDD->Xq_Q10[ last_smple_idx ], psDD->Gain_Q16[ last_smpl
e_idx ] ), 10 ) );
        NSQ->sLTP_shp_Q10[ NSQ->sLTP_shp_buf_idx - decisionDelay + i ] = psDD->Sh
ape_Q10[ last_smple_idx ];
        sLTP_Q16[          NSQ->sLTP_buf_idx     - decisionDelay + i ] = psDD->Pr
ed_Q16[  last_smple_idx ];

    }
```

```
    SKP_memcpy( NSQ->sLPC_Q14, &psDD->sLPC_Q14[ psEncC->subfr_length ], NSQ_LPC_B
UF_LENGTH * sizeof( SKP_int32 ) );

    /* Update states */
    NSQ->sLF_AR_shp_Q12    = psDD->LF_AR_Q12;
    NSQ->prev_inv_gain_Q16 = NSQ->prev_inv_gain_Q16;
    NSQ->lagPrev           = psEncCtrlC->pitchL[ NB_SUBFR - 1 ];

    /* Save quantized speech and noise shaping signals */
    SKP_memcpy( NSQ->xq,           &NSQ->xq[           psEncC->frame_length ], ps
EncC->frame_length * sizeof( SKP_int16 ) );
    SKP_memcpy( NSQ->sLTP_shp_Q10, &NSQ->sLTP_shp_Q10[ psEncC->frame_length ], ps
EncC->frame_length * sizeof( SKP_int32 ) );

}

/***************************************/
/* Noise shape quantizer for one subframe */
/***************************************/
SKP_INLINE void SKP_Silk_noise_shape_quantizer_del_dec(
    SKP_Silk_nsq_state  *NSQ,                   /* I/O  NSQ state
          */
    NSQ_del_dec_struct  psDelDec[],             /* I/O  Delayed decision states
          */
    SKP_int             sigtype,                /* I    Signal type
          */
    const SKP_int32     x_Q10[],                /* I
          */
    SKP_int             q[],                    /* O
          */
    SKP_int16           xq[],                   /* O
          */
    SKP_int32           sLTP_Q16[],             /* I/O  LTP filter state
          */
    const SKP_int16     a_Q12[],                /* I    Short term prediction coe
fs        */
    const SKP_int16     b_Q14[],                /* I    Long term prediction coef
s         */
    const SKP_int16     AR_shp_Q13[],           /* I    Noise shaping coefs
          */
    SKP_int             lag,                    /* I    Pitch lag
          */
    SKP_int32           HarmShapeFIRPacked_Q14, /* I
          */
    SKP_int             Tilt_Q14,               /* I    Spectral tilt
          */
    SKP_int32           LF_shp_Q14,             /* I
          */
    SKP_int32           Gain_Q16,               /* I
          */
    SKP_int             Lambda_Q10,             /* I
          */
    SKP_int             offset_Q10,             /* I
          */
    SKP_int             length,                 /* I    Input length
          */
    SKP_int             subfr,                  /* I    Subframe number
          */
    SKP_int             shapingLPCOrder,        /* I    Shaping LPC filter order
          */
    SKP_int             predictLPCOrder,        /* I    Prediction LPC filter ord
er        */
```

```
    SKP_int             nStatesDelayedDecision, /* I    Number of states in decis
ion tree   */
    SKP_int             *smpl_buf_idx,          /* I    Index to newest samples i
n buffers  */
    SKP_int             decisionDelay           /* I
           */
)
{
    SKP_int     i, j, k, Winner_ind, RDmin_ind, RDmax_ind, last_smple_idx;
    SKP_int32   Winner_rand_state;
    SKP_int32   LTP_pred_Q14, LPC_pred_Q10, n_AR_Q10, n_LTP_Q14;
    SKP_int32   n_LF_Q10;
    SKP_int32   r_Q10, rr_Q20, rd1_Q10, rd2_Q10, RDmin_Q10, RDmax_Q10;
```

```
    SKP_int32   q1_Q10, q2_Q10;
    SKP_int32   Atmp, dither;
    SKP_int32   exc_Q10, LPC_exc_Q10, xq_Q10;
    SKP_int32   tmp, sLF_AR_shp_Q10;
    SKP_int32   *pred_lag_ptr, *shp_lag_ptr;
    SKP_int32   *psLPC_Q14;
    SKP_int32   a_Q12_tmp[ MAX_LPC_ORDER / 2 ], AR_shp_Q13_tmp[ MAX_LPC_ORDER / 2
 ];
    NSQ_sample_struct  psSampleState[ DEL_DEC_STATES_MAX ][ 2 ];
    NSQ_del_dec_struct *psDD;
    NSQ_sample_struct  *psSS;

    shp_lag_ptr  = &NSQ->sLTP_shp_Q10[ NSQ->sLTP_shp_buf_idx - lag + HARM_SHAPE_F
IR_TAPS / 2 ];
    pred_lag_ptr = &sLTP_Q16[ NSQ->sLTP_buf_idx - lag + LTP_ORDER / 2 ];

    /* Preload LPC coeficients to array on stack. Gives small performance gain */
    SKP_memcpy( a_Q12_tmp, a_Q12, predictLPCOrder * sizeof( SKP_int16 ) );
    SKP_memcpy( AR_shp_Q13_tmp, AR_shp_Q13, shapingLPCOrder * sizeof( SKP_int16 )
 );

    for( i = 0; i < length; i++ ) {
        /* Perform common calculations used in all states */

        /* Long-term prediction */
        if( sigtype == SIG_TYPE_VOICED ) {
            /* Unrolled loop */
            LTP_pred_Q14 = SKP_SMULWB(               pred_lag_ptr[  0 ], b_Q14[ 0
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -1 ], b_Q14[ 1
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -2 ], b_Q14[ 2
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -3 ], b_Q14[ 3
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -4 ], b_Q14[ 4
 ] );
            pred_lag_ptr++;
        } else {
            LTP_pred_Q14 = 0;
        }

        /* Long-term shaping */
        if( lag > 0 ) {
            /* Symmetric, packed FIR coefficients */
            n_LTP_Q14 = SKP_SMULWB( SKP_ADD32( shp_lag_ptr[ 0 ], shp_lag_ptr[ -2
] ), HarmShapeFIRPacked_Q14 );
            n_LTP_Q14 = SKP_SMLAWT( n_LTP_Q14, shp_lag_ptr[ -1 ],
     HarmShapeFIRPacked_Q14 );
            n_LTP_Q14 = SKP_LSHIFT( n_LTP_Q14, 6 );
            shp_lag_ptr++;
        } else {
            n_LTP_Q14 = 0;
        }

        for( k = 0; k < nStatesDelayedDecision; k++ ) {
            /* Delayed decision state */
            psDD = &psDelDec[ k ];
```

```
            /* Sample state */
            psSS = psSampleState[ k ];

            /* Generate dither */
            psDD->Seed = SKP_RAND( psDD->Seed );

            /* dither = rand_seed < 0 ? 0xFFFFFFFF : 0; */
            dither = SKP_RSHIFT( psDD->Seed, 31 );

            /* Pointer used in short term prediction and shaping */
            psLPC_Q14 = &psDD->sLPC_Q14[ NSQ_LPC_BUF_LENGTH - 1 + i ];
            /* Short-term prediction */
            SKP_assert( ( predictLPCOrder  & 1 ) == 0 );    /* check that order i
s even */
            SKP_assert( ( (SKP_int64)a_Q12 & 3 ) == 0 );    /* check that array s
tarts at 4-byte aligned address */
            SKP_assert( predictLPCOrder >= 10 );            /* check that unrolli
ng works */

            /* Partially unrolled */
            Atmp = a_Q12_tmp[ 0 ];          /* read two coefficients at once */
            LPC_pred_Q10 = SKP_SMULWB(               psLPC_Q14[ 0  ], Atmp );
            LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -1 ], Atmp );
            Atmp = a_Q12_tmp[ 1 ];
            LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psLPC_Q14[ -2 ], Atmp );
            LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -3 ], Atmp );
            Atmp = a_Q12_tmp[ 2 ];
            LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psLPC_Q14[ -4 ], Atmp );
            LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -5 ], Atmp );
            Atmp = a_Q12_tmp[ 3 ];
            LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psLPC_Q14[ -6 ], Atmp );
            LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -7 ], Atmp );
            Atmp = a_Q12_tmp[ 4 ];
            LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psLPC_Q14[ -8 ], Atmp );
            LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -9 ], Atmp );
            for( j = 10; j < predictLPCOrder; j += 2 ) {
                Atmp = a_Q12_tmp[ j >> 1 ]; /* read two coefficients at once */
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psLPC_Q14[ -j     ], Atm
p );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psLPC_Q14[ -j - 1 ], Atm
p );
            }

            /* Noise shape feedback */
            SKP_assert( ( shapingLPCOrder       & 1 ) == 0 );   /* check that ord
er is even */
            SKP_assert( ( (SKP_int64)AR_shp_Q13 & 3 ) == 0 );   /* check that arr
ay starts at 4-byte aligned address */
            SKP_assert( shapingLPCOrder >= 12 );                /* check that unr
olling works */
            /* NOTE: the code below loads two int16 values in an int32, and multi
plies each using the   */
            /* SMLAWB and SMLAWT instructions. On a big-endian CPU the two int16
variables would be     */
            /* loaded in reverse order and the code will give the wrong result. I
n that case swapping   */
            /* the SMLAWB and SMLAWT instructions should solve the problem.
                        */

            /* Partially unrolled */
```

```
            Atmp = AR_shp_Q13_tmp[ 0 ];         /* read two coefficients at once
*/
            n_AR_Q10 = SKP_SMULWB(           psLPC_Q14[ 0  ], Atmp );
            n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -1 ], Atmp );
            Atmp = AR_shp_Q13_tmp[ 1 ];
            n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -2 ], Atmp );
            n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -3 ], Atmp );
            Atmp = AR_shp_Q13_tmp[ 2 ];
            n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -4 ], Atmp );
            n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -5 ], Atmp );
            Atmp = AR_shp_Q13_tmp[ 3 ];
            n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -6 ], Atmp );
            n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -7 ], Atmp );
            Atmp = AR_shp_Q13_tmp[ 4 ];
            n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -8 ], Atmp );
            n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -9 ], Atmp );
            Atmp = AR_shp_Q13_tmp[ 5 ];
            n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -10 ], Atmp );
            n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -11 ], Atmp );
            for( j = 12; j < shapingLPCOrder; j += 2 ) {
                Atmp = AR_shp_Q13_tmp[ j >> 1 ];        /* read two coefficients
at once */
                n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psLPC_Q14[ -j     ], Atmp );
                n_AR_Q10 = SKP_SMLAWT( n_AR_Q10, psLPC_Q14[ -j - 1 ], Atmp );
            }
            n_AR_Q10 = SKP_RSHIFT( n_AR_Q10, 1 );           /* Q11 -> Q10 */
            n_AR_Q10 = SKP_SMLAWB( n_AR_Q10, psDD->LF_AR_Q12, Tilt_Q14 );

            n_LF_Q10   = SKP_LSHIFT( SKP_SMULWB( psDD->Shape_Q10[ *smpl_buf_idx ]
, LF_shp_Q14 ), 2 );
            n_LF_Q10   = SKP_SMLAWT( n_LF_Q10, psDD->LF_AR_Q12, LF_shp_Q14 );

            /* Input minus prediction plus noise feedback                        *
/
            /* r = x[ i ] - LTP_pred - LPC_pred + n_AR + n_Tilt + n_LF + n_LTP  *
/
            tmp   = SKP_SUB32( LTP_pred_Q14, n_LTP_Q14 );                      /
* Add Q14 stuff */
            tmp   = SKP_RSHIFT_ROUND( tmp, 4 );                                /
* round to Q10  */
            tmp   = SKP_ADD32( tmp, LPC_pred_Q10 );                            /
* add Q10 stuff */
            tmp   = SKP_SUB32( tmp, n_AR_Q10 );                                /
* subtract Q10 stuff */
            tmp   = SKP_SUB32( tmp, n_LF_Q10 );                                /
* subtract Q10 stuff */
            r_Q10 = SKP_SUB32( x_Q10[ i ], tmp );                              /
* residual error Q10 */


            /* Flip sign depending on dither */
            r_Q10 = ( r_Q10 ^ dither ) - dither;
            r_Q10 = SKP_SUB32( r_Q10, offset_Q10 );
            r_Q10 = SKP_LIMIT( r_Q10, -64 << 10, 64 << 10 );

            /* Find two quantization level candidates and measure their rate-dist
ortion */
            if( r_Q10 < -1536 ) {
                q1_Q10  = SKP_LSHIFT( SKP_RSHIFT_ROUND( r_Q10, 10 ), 10 );
                r_Q10   = SKP_SUB32( r_Q10, q1_Q10 );
```

```
                rd1_Q10 = SKP_RSHIFT( SKP_SMLABB( SKP_MUL( -SKP_ADD32( q1_Q10, of
fset_Q10 ), Lambda_Q10 ), r_Q10, r_Q10 ), 10 );
                rd2_Q10 = SKP_ADD32( rd1_Q10, 1024 );
                rd2_Q10 = SKP_SUB32( rd2_Q10, SKP_ADD_LSHIFT32( Lambda_Q10, r_Q10
, 1 ) );
                q2_Q10  = SKP_ADD32( q1_Q10, 1024 );
            } else if( r_Q10 > 512 ) {
                q1_Q10  = SKP_LSHIFT( SKP_RSHIFT_ROUND( r_Q10, 10 ), 10 );
                r_Q10   = SKP_SUB32( r_Q10, q1_Q10 );
                rd1_Q10 = SKP_RSHIFT( SKP_SMLABB( SKP_MUL( SKP_ADD32( q1_Q10, off
set_Q10 ), Lambda_Q10 ), r_Q10, r_Q10 ), 10 );
                rd2_Q10 = SKP_ADD32( rd1_Q10, 1024 );
                rd2_Q10 = SKP_SUB32( rd2_Q10, SKP_SUB_LSHIFT32( Lambda_Q10, r_Q10
, 1 ) );
                q2_Q10  = SKP_SUB32( q1_Q10, 1024 );
            } else {            /* r_Q10 >= -1536 && q1_Q10 <= 512 */
                rr_Q20  = SKP_SMULBB( offset_Q10, Lambda_Q10 );
                rd2_Q10 = SKP_RSHIFT( SKP_SMLABB( rr_Q20, r_Q10, r_Q10 ), 10 );
                rd1_Q10 = SKP_ADD32( rd2_Q10, 1024 );
                rd1_Q10 = SKP_ADD32( rd1_Q10, SKP_SUB_RSHIFT32( SKP_ADD_LSHIFT32(
 Lambda_Q10, r_Q10, 1 ), rr_Q20, 9 ) );
                q1_Q10  = -1024;
                q2_Q10  = 0;
            }

            if( rd1_Q10 < rd2_Q10 ) {
                psSS[ 0 ].RD_Q10 = SKP_ADD32( psDD->RD_Q10, rd1_Q10 );
                psSS[ 1 ].RD_Q10 = SKP_ADD32( psDD->RD_Q10, rd2_Q10 );
                psSS[ 0 ].Q_Q10 = q1_Q10;
                psSS[ 1 ].Q_Q10 = q2_Q10;
            } else {
                psSS[ 0 ].RD_Q10 = SKP_ADD32( psDD->RD_Q10, rd2_Q10 );
                psSS[ 1 ].RD_Q10 = SKP_ADD32( psDD->RD_Q10, rd1_Q10 );
                psSS[ 0 ].Q_Q10 = q2_Q10;
                psSS[ 1 ].Q_Q10 = q1_Q10;
            }

            /* Update states for best quantization */

            /* Quantized excitation */
            exc_Q10 = SKP_ADD32( offset_Q10, psSS[ 0 ].Q_Q10 );
            exc_Q10 = ( exc_Q10 ^ dither ) - dither;

            /* Add predictions */
            LPC_exc_Q10 = exc_Q10 + SKP_RSHIFT_ROUND( LTP_pred_Q14, 4 );
            xq_Q10      = SKP_ADD32( LPC_exc_Q10, LPC_pred_Q10 );

            /* Update states */
            sLF_AR_shp_Q10         = SKP_SUB32(  xq_Q10, n_AR_Q10 );
            psSS[ 0 ].sLTP_shp_Q10 = SKP_SUB32(  sLF_AR_shp_Q10, n_LF_Q10 );
            psSS[ 0 ].LF_AR_Q12    = SKP_LSHIFT( sLF_AR_shp_Q10, 2 );
            psSS[ 0 ].xq_Q14       = SKP_LSHIFT( xq_Q10,         4 );
            psSS[ 0 ].LPC_exc_Q16  = SKP_LSHIFT( LPC_exc_Q10,    6 );
```

```
            /* Update states for second best quantization */

            /* Quantized excitation */
            exc_Q10 = SKP_ADD32( offset_Q10, psSS[ 1 ].Q_Q10 );
            exc_Q10 = ( exc_Q10 ^ dither ) - dither;

            /* Add predictions */
            LPC_exc_Q10 = exc_Q10 + SKP_RSHIFT_ROUND( LTP_pred_Q14, 4 );
            xq_Q10      = SKP_ADD32( LPC_exc_Q10, LPC_pred_Q10 );

            /* Update states */
            sLF_AR_shp_Q10         = SKP_SUB32(  xq_Q10, n_AR_Q10 );
            psSS[ 1 ].sLTP_shp_Q10 = SKP_SUB32(  sLF_AR_shp_Q10, n_LF_Q10 );
            psSS[ 1 ].LF_AR_Q12    = SKP_LSHIFT( sLF_AR_shp_Q10, 2 );
            psSS[ 1 ].xq_Q14       = SKP_LSHIFT( xq_Q10,         4 );
            psSS[ 1 ].LPC_exc_Q16  = SKP_LSHIFT( LPC_exc_Q10,    6 );
        }

        *smpl_buf_idx  = ( *smpl_buf_idx - 1 ) & DECISION_DELAY_MASK;
       /* Index to newest samples                */
        last_smple_idx = ( *smpl_buf_idx + decisionDelay ) & DECISION_DELAY_MASK;
       /* Index to decisionDelay old samples   */

        /* Find winner */
        RDmin_Q10 = psSampleState[ 0 ][ 0 ].RD_Q10;
        Winner_ind = 0;
        for( k = 1; k < nStatesDelayedDecision; k++ ) {
            if( psSampleState[ k ][ 0 ].RD_Q10 < RDmin_Q10 ) {
                RDmin_Q10   = psSampleState[ k ][ 0 ].RD_Q10;
                Winner_ind = k;
            }
        }

        /* Increase RD values of expired states */
        Winner_rand_state = psDelDec[ Winner_ind ].RandState[ last_smple_idx ];
        for( k = 0; k < nStatesDelayedDecision; k++ ) {
            if( psDelDec[ k ].RandState[ last_smple_idx ] != Winner_rand_state )
{
                psSampleState[ k ][ 0 ].RD_Q10 = SKP_ADD32( psSampleState[ k ][ 0
 ].RD_Q10, ( SKP_int32_MAX >> 4 ) );
                psSampleState[ k ][ 1 ].RD_Q10 = SKP_ADD32( psSampleState[ k ][ 1
 ].RD_Q10, ( SKP_int32_MAX >> 4 ) );
                SKP_assert( psSampleState[ k ][ 0 ].RD_Q10 >= 0 );
            }
        }

        /* Find worst in first set and best in second set */
        RDmax_Q10  = psSampleState[ 0 ][ 0 ].RD_Q10;
        RDmin_Q10  = psSampleState[ 0 ][ 1 ].RD_Q10;
        RDmax_ind = 0;
        RDmin_ind = 0;
        for( k = 1; k < nStatesDelayedDecision; k++ ) {
            /* find worst in first set */
```

```
            if( psSampleState[ k ][ 0 ].RD_Q10 > RDmax_Q10 ) {
                RDmax_Q10  = psSampleState[ k ][ 0 ].RD_Q10;
                RDmax_ind = k;
            }
            /* find best in second set */
            if( psSampleState[ k ][ 1 ].RD_Q10 < RDmin_Q10 ) {
                RDmin_Q10  = psSampleState[ k ][ 1 ].RD_Q10;
                RDmin_ind = k;
            }
        }

        /* Replace a state if best from second set outperforms worst in first set
 */
        if( RDmin_Q10 < RDmax_Q10 ) {
            SKP_Silk_copy_del_dec_state( &psDelDec[ RDmax_ind ], &psDelDec[ RDmin
_ind ], i );
            SKP_memcpy( &psSampleState[ RDmax_ind ][ 0 ], &psSampleState[ RDmin_i
nd ][ 1 ], sizeof( NSQ_sample_struct ) );
        }

        /* Write samples from winner to output and long-term filter states */
        psDD = &psDelDec[ Winner_ind ];
        if( subfr > 0 || i >= decisionDelay ) {
            q[  i - decisionDelay ] = ( SKP_int )SKP_RSHIFT( psDD->Q_Q10[ last_sm
ple_idx ], 10 );
            xq[ i - decisionDelay ] = ( SKP_int16 )SKP_SAT16( SKP_RSHIFT_ROUND(
                SKP_SMULWW( psDD->Xq_Q10[ last_smple_idx ], psDD->Gain_Q16[ last_
smple_idx ] ), 10 ) );
            NSQ->sLTP_shp_Q10[ NSQ->sLTP_shp_buf_idx - decisionDelay ] = psDD->Sh
ape_Q10[ last_smple_idx ];
            sLTP_Q16[          NSQ->sLTP_buf_idx     - decisionDelay ] = psDD->Pr
ed_Q16[  last_smple_idx ];
        }
        NSQ->sLTP_shp_buf_idx++;
        NSQ->sLTP_buf_idx++;

        /* Update states */
        for( k = 0; k < nStatesDelayedDecision; k++ ) {
            psDD                                    = &psDelDec[ k ];
            psSS                                    = &psSampleState[ k ][ 0 ];
            psDD->LF_AR_Q12                         = psSS->LF_AR_Q12;
            psDD->sLPC_Q14[ NSQ_LPC_BUF_LENGTH + i ] = psSS->xq_Q14;
            psDD->Xq_Q10[    *smpl_buf_idx ]        = SKP_RSHIFT( psSS->xq_Q14,
4 );
            psDD->Q_Q10[     *smpl_buf_idx ]        = psSS->Q_Q10;
            psDD->Pred_Q16[  *smpl_buf_idx ]        = psSS->LPC_exc_Q16;
            psDD->Shape_Q10[ *smpl_buf_idx ]        = psSS->sLTP_shp_Q10;
            psDD->Seed                              = SKP_ADD_RSHIFT32( psDD->Se
ed, psSS->Q_Q10, 10 );
            psDD->RandState[ *smpl_buf_idx ]        = psDD->Seed;
            psDD->RD_Q10                            = psSS->RD_Q10;
            psDD->Gain_Q16[  *smpl_buf_idx ]        = Gain_Q16;
        }
    }
    /* Update LPC states */
    for( k = 0; k < nStatesDelayedDecision; k++ ) {
        psDD = &psDelDec[ k ];
```

```
        SKP_memcpy( psDD->sLPC_Q14, &psDD->sLPC_Q14[ length ], NSQ_LPC_BUF_LENGTH
 * sizeof( SKP_int32 ) );
    }
}

SKP_INLINE void SKP_Silk_nsq_del_dec_scale_states(
    SKP_Silk_nsq_state  *NSQ,                   /* I/O  NSQ state
           */
    NSQ_del_dec_struct  psDelDec[],             /* I/O  Delayed decision states
           */
    const SKP_int16     x[],                    /* I    Input in Q0
           */
    SKP_int32           x_sc_Q10[],             /* O    Input scaled with 1/Gain
in Q10     */
    SKP_int             length,                 /* I    Length of input
           */
    SKP_int16           sLTP[],                 /* I    Re-whitened LTP state in
Q0         */
    SKP_int32           sLTP_Q16[],             /* O    LTP state matching scaled
 input     */
    SKP_int             subfr,                  /* I    Subframe number
           */
    SKP_int             nStatesDelayedDecision, /* I    Number of del dec states
           */
    SKP_int             smpl_buf_idx,           /* I    Index to newest samples i
n buffers  */
    const SKP_int       LTP_scale_Q14,          /* I    LTP state scaling
           */
    const SKP_int32     Gains_Q16[ NB_SUBFR ],  /* I
           */
    const SKP_int       pitchL[ NB_SUBFR ]      /* I    Pitch lag
           */
)
{
    SKP_int             i, k, scale_length, lag;
    SKP_int32           inv_gain_Q16, gain_adj_Q16, inv_gain_Q32;
    NSQ_del_dec_struct *psDD;

    inv_gain_Q16 = SKP_DIV32( SKP_int32_MAX, SKP_RSHIFT( Gains_Q16[ subfr ], 1 )
);
    inv_gain_Q16 = SKP_min( inv_gain_Q16, SKP_int16_MAX );
    lag          = pitchL[ subfr ];
    /* After rewhitening the LTP state is un-scaled. So scale with inv_gain_Q16 *
/
    if( NSQ->rewhite_flag ) {
        inv_gain_Q32 = SKP_LSHIFT( inv_gain_Q16, 16 );
        if( subfr == 0 ) {
            /* Do LTP downscaling */
            inv_gain_Q32 = SKP_LSHIFT( SKP_SMULWB( inv_gain_Q32, LTP_scale_Q14 ),
 2 );
        }
        for( i = NSQ->sLTP_buf_idx - lag - LTP_ORDER / 2; i < NSQ->sLTP_buf_idx;
i++ ) {
            SKP_assert( i < MAX_FRAME_LENGTH );
            sLTP_Q16[ i ] = SKP_SMULWB( inv_gain_Q32, sLTP[ i ] );
        }
    }

    /* Adjust for changing gain */
    if( inv_gain_Q16 != NSQ->prev_inv_gain_Q16 ) {
        gain_adj_Q16 = SKP_DIV32_varQ( inv_gain_Q16, NSQ->prev_inv_gain_Q16, 16 )
;
```

```
        for( k = 0; k < nStatesDelayedDecision; k++ ) {
            psDD = &psDelDec[ k ];

            /* Scale scalar states */
```

```
            psDD->LF_AR_Q12 = SKP_SMULWW( gain_adj_Q16, psDD->LF_AR_Q12 );

            /* scale short term state */
            for( i = 0; i < NSQ_LPC_BUF_LENGTH; i++ ) {
                psDD->sLPC_Q14[ NSQ_LPC_BUF_LENGTH - i - 1 ] = SKP_SMULWW( gain_a
dj_Q16, psDD->sLPC_Q14[ NSQ_LPC_BUF_LENGTH - i - 1 ] );
            }
            for( i = 0; i < DECISION_DELAY; i++ ) {
                psDD->Pred_Q16[  i ] = SKP_SMULWW( gain_adj_Q16, psDD->Pred_Q16[
 i ] );
                psDD->Shape_Q10[ i ] = SKP_SMULWW( gain_adj_Q16, psDD->Shape_Q10[
 i ] );
            }
        }

        /* Scale long term shaping state */

        /* Calculate length to be scaled, Worst case: Next frame is voiced with m
ax lag */
        scale_length = length * NB_SUBFR;
       /* aprox max lag */
        scale_length = scale_length - SKP_SMULBB( NB_SUBFR - ( subfr + 1 ), lengt
h );   /* subtract samples that will be too old in next frame */
        scale_length = SKP_max_int( scale_length, lag + LTP_ORDER );
       /* make sure to scale whole pitch period if voiced */

        for( i = NSQ->sLTP_shp_buf_idx - scale_length; i < NSQ->sLTP_shp_buf_idx;
 i++ ) {
            NSQ->sLTP_shp_Q10[ i ] = SKP_SMULWW( gain_adj_Q16, NSQ->sLTP_shp_Q10[
 i ] );
        }

        /* Scale LTP predict state */
        if( NSQ->rewhite_flag == 0 ) {
            for( i = NSQ->sLTP_buf_idx - lag - LTP_ORDER / 2; i < NSQ->sLTP_buf_i
dx; i++ ) {
                sLTP_Q16[ i ] = SKP_SMULWW( gain_adj_Q16, sLTP_Q16[ i ] );
            }
        }
    }

    /* Scale input */
    for( i = 0; i < length; i++ ) {
        x_sc_Q10[ i ] = SKP_RSHIFT( SKP_SMULBB( x[ i ], ( SKP_int16 )inv_gain_Q16
 ), 6 );
    }

    /* save inv_gain */
    SKP_assert( inv_gain_Q16 != 0 );
    NSQ->prev_inv_gain_Q16 = inv_gain_Q16;
}

SKP_INLINE void SKP_Silk_copy_del_dec_state(
    NSQ_del_dec_struct  *DD_dst,                /* I    Dst del dec state
          */
    NSQ_del_dec_struct  *DD_src,                /* I    Src del dec state
          */
    SKP_int             LPC_state_idx           /* I    Index to LPC buffer
          */
)
{
    SKP_memcpy( DD_dst->RandState, DD_src->RandState,   DECISION_DELAY * sizeof(
```

```
SKP_int   ) );
```

```
    SKP_memcpy( DD_dst->Q_Q10,     DD_src->Q_Q10,       DECISION_DELAY * sizeof(
SKP_int32 ) );
    SKP_memcpy( DD_dst->Pred_Q16,  DD_src->Pred_Q16,    DECISION_DELAY * sizeof(
SKP_int32 ) );
    SKP_memcpy( DD_dst->Shape_Q10, DD_src->Shape_Q10,   DECISION_DELAY * sizeof(
SKP_int32 ) );
    SKP_memcpy( DD_dst->Xq_Q10,    DD_src->Xq_Q10,      DECISION_DELAY * sizeof(
SKP_int32 ) );

    SKP_memcpy( &DD_dst->sLPC_Q14[ LPC_state_idx ], &DD_src->sLPC_Q14[ LPC_state_
idx ], NSQ_LPC_BUF_LENGTH * sizeof( SKP_int32 ) );
    DD_dst->LF_AR_Q12 = DD_src->LF_AR_Q12;
    DD_dst->Seed      = DD_src->Seed;
    DD_dst->SeedInit  = DD_src->SeedInit;
    DD_dst->RD_Q10    = DD_src->RD_Q10;
}
```

A.79. src/SKP_Silk_perceptual_parameters_FIX.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_PERCEPTUAL_PARAMETERS_FIX_H
#define SKP_SILK_PERCEPTUAL_PARAMETERS_FIX_H

#ifdef __cplusplus
extern "C"
```

```
{
#endif

/* reduction in coding SNR during low speech activity */
#define BG_SNR_DECR_dB_Q7                               (3<<7)

/* factor for reducing quantization noise during voiced speech */
#define HARM_SNR_INCR_dB_Q7                             (2<<7)

/* factor for reducing quantization noise for unvoiced sparse signals */
#define SPARSE_SNR_INCR_dB_Q7                           (2<<7)

/* threshold for sparseness measure above which to use lower quantization offset
during unvoiced */
#define SPARSENESS_THRESHOLD_QNT_OFFSET_Q8              (3<<6) // 0.75


/* noise shaping filter chirp factor */
#define BANDWIDTH_EXPANSION_Q16                         61604 // 0.94

/* difference between chirp factors for analysis and synthesis noise shaping filt
ers at low bitrates */
#define LOW_RATE_BANDWIDTH_EXPANSION_DELTA_Q16          655 //0.01f

/* factor to reduce all bandwidth expansion coefficients for super wideband, rela
tive to wideband */
#define SWB_BANDWIDTH_EXPANSION_REDUCTION_Q16           (1<<16) // 1.0f;

/* gain reduction for fricatives */
#define DE_ESSER_COEF_SWB_dB_Q7                         (2 << 7)
#define DE_ESSER_COEF_WB_dB_Q7                          (1 << 7)


/* extra harmonic boosting (signal shaping) at low bitrates */
#define LOW_RATE_HARMONIC_BOOST_Q16                     6554 // 0.1

/* extra harmonic boosting (signal shaping) for noisy input signals */
#define LOW_INPUT_QUALITY_HARMONIC_BOOST_Q16            6554 // 0.1

/* harmonic noise shaping */
#define HARMONIC_SHAPING_Q16                            19661 // 0.3

/* extra harmonic noise shaping for high bitrates or noisy input */
#define HIGH_RATE_OR_LOW_QUALITY_HARMONIC_SHAPING_Q16   13107 // 0.2


/* parameter for shaping noise towards higher frequencies */
#define HP_NOISE_COEF_Q16                               19661 // 0.3

/* parameter for shaping noise extra towards higher frequencies during voiced spe
ech */
#define HARM_HP_NOISE_COEF_Q24                          7549747 // 0.45
```

```
/* parameter for applying a high-pass tilt to the input signal */
#define INPUT_TILT_Q26                                  2684355 // 0.04

/* parameter for extra high-pass tilt to the input signal at high rates */
#define HIGH_RATE_INPUT_TILT_Q12                        246 // 0.06

/* parameter for reducing noise at the very low frequencies */
#define LOW_FREQ_SHAPING_Q0                             3

/* less reduction of noise at the very low frequencies for signals with low SNR a
t low frequencies */
#define LOW_QUALITY_LOW_FREQ_SHAPING_DECR_Q1            1 // 0.5_Q0

/* fraction added to first autocorrelation value */
#define SHAPE_WHITE_NOISE_FRACTION_Q20                  50 // 50_Q20 = 4.7684e-5

/* fraction of first autocorrelation value added to residual energy value; limits
 prediction gain */
#define SHAPE_MIN_ENERGY_RATIO_Q24                      256

/* noise floor to put a low limit on the quantization step size */
#define NOISE_FLOOR_dB_Q7                               (4 << 7)

/* noise floor relative to active speech gain level */
#define RELATIVE_MIN_GAIN_dB_Q7                         -6400 // -50_Q0 = -6400_Q
7

/* subframe smoothing coefficient for determining active speech gain level (lower
 -> more smoothing) */
#define GAIN_SMOOTHING_COEF_Q10                         1 // 1e-3_Q0 = 1.024_Q10

/* subframe smoothing coefficient for HarmBoost, HarmShapeGain, Tilt (lower -> mo
re smoothing) */
#define SUBFR_SMTH_COEF_Q16                             26214 // 0.4

#define NOISE_GAIN_VL_Q16                               7864
#define NOISE_GAIN_VH_Q16                               7864
#define NOISE_GAIN_UVL_Q16                              6554
#define NOISE_GAIN_UVH_Q16                              9830

#ifdef __cplusplus
}
#endif

#endif //SKP_SILK_PERCEPTUAL_PARAMETERS_FIX_H
```

A.80. src/SKP_Silk_pitch_analysis_core.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
```

```
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*****************************************************
* Pitch analyser function
******************************************************* */
#include "SKP_Silk_SigProc_FIX.h"
#include "SKP_Silk_pitch_est_defines.h"
#include "SKP_Silk_resample_rom.h"

#define SCRATCH_SIZE    22

/*********************************************************/
/* Internally used functions                             */
/*********************************************************/
void SKP_FIX_P_Ana_calc_corr_st3(
    SKP_int32        cross_corr_st3[PITCH_EST_NB_SUBFR][PITCH_EST_NB_CBKS_STAGE3_
MAX][PITCH_EST_NB_STAGE3_LAGS],/* (O) 3 DIM correlation array */
    const SKP_int16  signal[],                      /* I vector to correlate
      */
    SKP_int          start_lag,                     /* I lag offset to search a
round */
    SKP_int          sf_length,                     /* I length of a 5 ms subfr
ame   */
    SKP_int          complexity                     /* I Complexity setting
      */
);

void SKP_FIX_P_Ana_calc_energy_st3(
    SKP_int32        energies_st3[PITCH_EST_NB_SUBFR][PITCH_EST_NB_CBKS_STAGE3_MA
X][PITCH_EST_NB_STAGE3_LAGS],/* (O) 3 DIM energy array */
    const SKP_int16  signal[],                      /* I vector to calc energy
in    */
    SKP_int          start_lag,                     /* I lag offset to search a
round */
```

```
    SKP_int          sf_length,                      /* I length of one 5 ms sub
frame */
    SKP_int          complexity                      /* I Complexity setting
      */
);

SKP_int32 SKP_FIX_P_Ana_find_scaling(
    const SKP_int16  *signal,
    const SKP_int    signal_length,
    const SKP_int    sum_sqr_len
);

void SKP_Silk_decode_pitch(
    SKP_int          lagIndex,                       /* I
      */
    SKP_int          contourIndex,                   /* O
      */
    SKP_int          pitch_lags[],                   /* O 4 pitch values
      */
    SKP_int          Fs_kHz                          /* I sampling frequency (kH
z)    */
)
{
    SKP_int lag, i, min_lag;

    min_lag = SKP_SMULBB( PITCH_EST_MIN_LAG_MS, Fs_kHz );

    /* Only for 24 / 16 kHz version for now */
    lag = min_lag + lagIndex;
    if( Fs_kHz == 8 ) {
        /* Only a small codebook for 8 khz */
        for( i = 0; i < PITCH_EST_NB_SUBFR; i++ ) {
            pitch_lags[ i ] = lag + SKP_Silk_CB_lags_stage2[ i ][ contourIndex ];
        }
    } else {
        for( i = 0; i < PITCH_EST_NB_SUBFR; i++ ) {
            pitch_lags[ i ] = lag + SKP_Silk_CB_lags_stage3[ i ][ contourIndex ];
        }
    }
}

/***********************************************************/
/*      FIXED POINT CORE PITCH ANALYSIS FUNCTION           */
/***********************************************************/
SKP_int SKP_Silk_pitch_analysis_core(  /* O    Voicing estimate: 0 voiced, 1 unvo
iced                        */
    const SKP_int16  *signal,           /* I    Signal of length PITCH_EST_FRAME
_LENGTH_MS*Fs_kHz           */
    SKP_int          *pitch_out,        /* O    4 pitch lag values
                                */
    SKP_int          *lagIndex,         /* O    Lag Index
                                */
    SKP_int          *contourIndex,     /* O    Pitch contour Index
                                */
    SKP_int          *LTPCorr_Q15,      /* I/O  Normalized correlation; input: v
alue from previous frame    */
    SKP_int          prevLag,           /* I    Last lag of previous frame; set
to zero is unvoiced         */
    const SKP_int32  search_thres1_Q16, /* I    First stage threshold for lag ca
ndidates 0 - 1              */
    const SKP_int    search_thres2_Q15, /* I    Final threshold for lag candidat
es 0 - 1                    */
```

```
    const SKP_int    Fs_kHz,             /* I    Sample frequency (kHz)
                                                                    */
```

```
    const SKP_int    complexity          /* I    Complexity setting, 0-2, where 2
 is highest                 */
)
{
    SKP_int16 signal_8kHz[ PITCH_EST_MAX_FRAME_LENGTH_ST_2 ];
    SKP_int16 signal_4kHz[ PITCH_EST_MAX_FRAME_LENGTH_ST_1 ];
    SKP_int32 scratch_mem[ 3 * PITCH_EST_MAX_FRAME_LENGTH ];
    SKP_int16 *input_signal_ptr;
    SKP_int32 filt_state[ PITCH_EST_MAX_DECIMATE_STATE_LENGTH ];
    SKP_int   i, k, d, j;
    SKP_int16 C[ PITCH_EST_NB_SUBFR ][ ( PITCH_EST_MAX_LAG >> 1 ) + 5 ];
    const SKP_int16 *target_ptr, *basis_ptr;
    SKP_int32 cross_corr, normalizer, energy, shift, energy_basis, energy_target;
    SKP_int   d_srch[ PITCH_EST_D_SRCH_LENGTH ];
    SKP_int16 d_comp[ ( PITCH_EST_MAX_LAG >> 1 ) + 5 ];
    SKP_int   Cmax, length_d_srch, length_d_comp;
    SKP_int32 sum, threshold, temp32;
    SKP_int   CBimax, CBimax_new, CBimax_old, lag, start_lag, end_lag, lag_new;
    SKP_int32 CC[ PITCH_EST_NB_CBKS_STAGE2_EXT ], CCmax, CCmax_b, CCmax_new_b, CC
max_new;
    SKP_int32 energies_st3[  PITCH_EST_NB_SUBFR ][ PITCH_EST_NB_CBKS_STAGE3_MAX ]
[ PITCH_EST_NB_STAGE3_LAGS ];
    SKP_int32 crosscorr_st3[ PITCH_EST_NB_SUBFR ][ PITCH_EST_NB_CBKS_STAGE3_MAX ]
[ PITCH_EST_NB_STAGE3_LAGS ];
    SKP_int32 lag_counter;
    SKP_int   frame_length, frame_length_8kHz, frame_length_4kHz, max_sum_sq_leng
th;
    SKP_int   sf_length, sf_length_8kHz, sf_length_4kHz;
    SKP_int   min_lag, min_lag_8kHz, min_lag_4kHz;
    SKP_int   max_lag, max_lag_8kHz, max_lag_4kHz;
    SKP_int32 contour_bias, diff;
    SKP_int32 lz, lshift;
    SKP_int   cbk_offset, cbk_size, nb_cbks_stage2;
    SKP_int32 delta_lag_log2_sqr_Q7, lag_log2_Q7, prevLag_log2_Q7, prev_lag_bias_
Q15, corr_thres_Q15;

    /* Check for valid sampling frequency */
    SKP_assert( Fs_kHz == 8 || Fs_kHz == 12 || Fs_kHz == 16 || Fs_kHz == 24 );

    /* Check for valid complexity setting */
    SKP_assert( complexity >= SigProc_PITCH_EST_MIN_COMPLEX );
    SKP_assert( complexity <= SigProc_PITCH_EST_MAX_COMPLEX );

    SKP_assert( search_thres1_Q16 >= 0 && search_thres1_Q16 <= (1<<16) );
    SKP_assert( search_thres2_Q15 >= 0 && search_thres2_Q15 <= (1<<15) );

    /* Setup frame lengths max / min lag for the sampling frequency */
    frame_length      = PITCH_EST_FRAME_LENGTH_MS * Fs_kHz;
    frame_length_4kHz = PITCH_EST_FRAME_LENGTH_MS * 4;
    frame_length_8kHz = PITCH_EST_FRAME_LENGTH_MS * 8;
    sf_length         = SKP_RSHIFT( frame_length,      3 );
    sf_length_4kHz    = SKP_RSHIFT( frame_length_4kHz, 3 );
    sf_length_8kHz    = SKP_RSHIFT( frame_length_8kHz, 3 );
    min_lag           = PITCH_EST_MIN_LAG_MS * Fs_kHz;
```

```
    min_lag_4kHz      = PITCH_EST_MIN_LAG_MS * 4;
    min_lag_8kHz      = PITCH_EST_MIN_LAG_MS * 8;
    max_lag           = PITCH_EST_MAX_LAG_MS * Fs_kHz;
    max_lag_4kHz      = PITCH_EST_MAX_LAG_MS * 4;
    max_lag_8kHz      = PITCH_EST_MAX_LAG_MS * 8;

    SKP_memset( C, 0, sizeof( SKP_int16 ) * PITCH_EST_NB_SUBFR * ( ( PITCH_EST_MA
X_LAG >> 1 ) + 5) );

    /* Resample from input sampled at Fs_kHz to 8 kHz */
    if( Fs_kHz == 12 ) {
        SKP_int16 R23[ SigProc_Resample_2_3_coarsest_NUM_FIR_COEFS - 1 ];
        SKP_memset( R23, 0, ( SigProc_Resample_2_3_coarsest_NUM_FIR_COEFS - 1 ) *
 sizeof( SKP_int16 ) );

        SKP_Silk_resample_2_3_coarsest( signal_8kHz, R23, signal,
            PITCH_EST_FRAME_LENGTH_MS * 12, (SKP_int16*)scratch_mem );
    } else if( Fs_kHz == 16 ) {
        if( complexity == SigProc_PITCH_EST_MAX_COMPLEX ) {
            SKP_assert( 4 <= PITCH_EST_MAX_DECIMATE_STATE_LENGTH );
            SKP_memset( filt_state, 0, 4 * sizeof( SKP_int32 ) );

            SKP_Silk_resample_1_2_coarse( signal, filt_state, signal_8kHz,
                scratch_mem, frame_length_8kHz );
        } else {
            SKP_assert( 2 <= PITCH_EST_MAX_DECIMATE_STATE_LENGTH );
            SKP_memset( filt_state, 0, 2 * sizeof( SKP_int32 ) );

            SKP_Silk_resample_1_2_coarsest( signal, filt_state, signal_8kHz,
                scratch_mem, frame_length_8kHz );
        }
    } else if( Fs_kHz == 24 ) {
        /* Resample to 24 -> 8 khz */
        SKP_assert( 7 <= PITCH_EST_MAX_DECIMATE_STATE_LENGTH );
        SKP_memset( filt_state, 0, 7 * sizeof( SKP_int32 ) );

        SKP_Silk_resample_1_3( signal_8kHz, filt_state, signal, 24 * PITCH_EST_FR
AME_LENGTH_MS );

    } else {
        SKP_assert( Fs_kHz == 8 );
        SKP_memcpy( signal_8kHz, signal, frame_length_8kHz * sizeof( SKP_int16 )
);
    }

    /* Decimate again to 4 kHz. Set mem to zero */
    if( complexity == SigProc_PITCH_EST_MAX_COMPLEX ) {
        SKP_assert( 4 <= PITCH_EST_MAX_DECIMATE_STATE_LENGTH );
        SKP_memset( filt_state, 0, 4 * sizeof( SKP_int32 ) );
        SKP_Silk_resample_1_2_coarse( signal_8kHz, filt_state,
            signal_4kHz, scratch_mem, frame_length_4kHz );
    } else {
```

```
        SKP_assert( 2 <= PITCH_EST_MAX_DECIMATE_STATE_LENGTH );
        SKP_memset( filt_state, 0, 2 * sizeof( SKP_int32 ) );
        SKP_Silk_resample_1_2_coarsest( signal_8kHz, filt_state,
            signal_4kHz, scratch_mem, frame_length_4kHz );
    }

    /* Low-pass filter */
    for( i = frame_length_4kHz - 1; i > 0; i-- ) {
        signal_4kHz[ i ] = SKP_ADD_SAT16( signal_4kHz[ i ], signal_4kHz[ i - 1 ]
);
    }

    /**********************************************************************
***
    ** Scale 4 kHz signal down to prevent correlations measures from overflowing
    ** find scaling as max scaling for each 8kHz(?) subframe
    **********************************************************************
**/

    /* Inner product is calculated with different lengths, so scale for the worst
 case */
    max_sum_sq_length = SKP_max_32( sf_length_8kHz, SKP_RSHIFT( frame_length_4kHz
, 1 ) );
    shift = SKP_FIX_P_Ana_find_scaling( signal_4kHz, frame_length_4kHz, max_sum_s
q_length );
    if( shift > 0 ) {
        for( i = 0; i < frame_length_4kHz; i++ ) {
            signal_4kHz[ i ] = SKP_RSHIFT( signal_4kHz[ i ], shift );
        }
    }

    /**********************************************************************
**
    * FIRST STAGE, operating in 4 khz
    **********************************************************************
*/
    target_ptr = &signal_4kHz[ SKP_RSHIFT( frame_length_4kHz, 1 ) ];
    for( k = 0; k < 2; k++ ) {
        /* Check that we are within range of the array */
        SKP_assert( target_ptr >= signal_4kHz );
        SKP_assert( target_ptr + sf_length_8kHz <= signal_4kHz + frame_length_4kH
z );

        basis_ptr = target_ptr - min_lag_4kHz;

        /* Check that we are within range of the array */
        SKP_assert( basis_ptr >= signal_4kHz );
        SKP_assert( basis_ptr + sf_length_8kHz <= signal_4kHz + frame_length_4kHz
 );

        normalizer = 0;
        cross_corr = 0;
        /* Calculate first vector products before loop */
        cross_corr = SKP_Silk_inner_prod_aligned( target_ptr, basis_ptr, sf_lengt
h_8kHz );
        normalizer = SKP_Silk_inner_prod_aligned( basis_ptr,  basis_ptr, sf_lengt
h_8kHz );
        normalizer = SKP_ADD_SAT32( normalizer, 1000 );

        temp32 = SKP_DIV32( cross_corr, SKP_Silk_SQRT_APPROX( normalizer ) + 1 );
```

```
        C[ k ][ min_lag_4kHz ] = (SKP_int16)SKP_SAT16( temp32 );        /* Q0 */

        /* From now on normalizer is computed recursively */
        for( d = min_lag_4kHz + 1; d <= max_lag_4kHz; d++ ) {
            basis_ptr--;

            /* Check that we are within range of the array */
            SKP_assert( basis_ptr >= signal_4kHz );
            SKP_assert( basis_ptr + sf_length_8kHz <= signal_4kHz + frame_length_
4kHz );

            cross_corr = SKP_Silk_inner_prod_aligned( target_ptr, basis_ptr, sf_l
ength_8kHz );

            /* Add contribution of new sample and remove contribution from oldest
 sample */
            normalizer +=
                SKP_SMULBB( basis_ptr[ 0 ], basis_ptr[ 0 ] ) -
                SKP_SMULBB( basis_ptr[ sf_length_8kHz ], basis_ptr[ sf_length_8kH
z ] );

            temp32 = SKP_DIV32( cross_corr, SKP_Silk_SQRT_APPROX( normalizer ) +
1 );
            C[ k ][ d ] = (SKP_int16)SKP_SAT16( temp32 );
/* Q0 */
        }
        /* Update target pointer */
        target_ptr += sf_length_8kHz;
    }

    /* Combine two subframes into single correlation measure and apply short-lag
bias */
    for( i = max_lag_4kHz; i >= min_lag_4kHz; i-- ) {
        sum = (SKP_int32)C[ 0 ][ i ] + (SKP_int32)C[ 1 ][ i ];                /*
Q0 */
        SKP_assert( SKP_RSHIFT( sum, 1 ) == SKP_SAT16( SKP_RSHIFT( sum, 1 ) ) );
        sum = SKP_RSHIFT( sum, 1 );                                           /*
Q-1 */
        SKP_assert( SKP_LSHIFT( (SKP_int32)-i, 4 ) == SKP_SAT16( SKP_LSHIFT( (SKP
_int32)-i, 4 ) ) );
        sum = SKP_SMLAWB( sum, sum, SKP_LSHIFT( -i, 4 ) );                    /*
Q-1 */
        SKP_assert( sum == SKP_SAT16( sum ) );
        C[ 0 ][ i ] = (SKP_int16)sum;                                         /*
Q-1 */
    }

    /* Sort */
    length_d_srch = 5 + complexity;
    SKP_assert( length_d_srch <= PITCH_EST_D_SRCH_LENGTH );
    SKP_Silk_insertion_sort_decreasing_int16( &C[ 0 ][ min_lag_4kHz ], d_srch, ma
x_lag_4kHz - min_lag_4kHz + 1, length_d_srch );

    /* Escape if correlation is very low already here */
    target_ptr = &signal_4kHz[ SKP_RSHIFT( frame_length_4kHz, 1 ) ];
    energy = SKP_Silk_inner_prod_aligned( target_ptr, target_ptr, SKP_RSHIFT( fra
me_length_4kHz, 1 ) );
    energy = SKP_ADD_SAT32( energy, 1000 );                                   /* Q
0 */
    Cmax = (SKP_int)C[ 0 ][ min_lag_4kHz ];                                   /* Q
-1 */
    threshold = SKP_SMULBB( Cmax, Cmax );                                     /* Q
```

```
-2 */
    /* Compare in Q-2 domain */
    if( SKP_RSHIFT( energy, 4 + 2 ) > threshold ) {
```

```
        SKP_memset( pitch_out, 0, PITCH_EST_NB_SUBFR * sizeof( SKP_int ) );
        *LTPCorr_Q15  = 0;
        *lagIndex     = 0;
        *contourIndex = 0;
        return 1;
    }

    threshold = SKP_SMULWB( search_thres1_Q16, Cmax );
    for( i = 0; i < length_d_srch; i++ ) {
        /* Convert to 8 kHz indices for the sorted correlation that exceeds the t
hreshold */
        if( C[ 0 ][ min_lag_4kHz + i ] > threshold ) {
            d_srch[ i ] = SKP_LSHIFT( d_srch[ i ] + min_lag_4kHz, 1 );
        } else {
            length_d_srch = i;
            break;
        }
    }
    SKP_assert( length_d_srch > 0 );

    for( i = min_lag_8kHz - 5; i < max_lag_8kHz + 5; i++ ) {
        d_comp[ i ] = 0;
    }
    for( i = 0; i < length_d_srch; i++ ) {
        d_comp[ d_srch[ i ] ] = 1;
    }

    /* Convolution */
    for( i = max_lag_8kHz + 3; i >= min_lag_8kHz; i-- ) {
        d_comp[ i ] += d_comp[ i - 1 ] + d_comp[ i - 2 ];
    }

    length_d_srch = 0;
    for( i = min_lag_8kHz; i < max_lag_8kHz + 1; i++ ) {
        if( d_comp[ i + 1 ] > 0 ) {
            d_srch[ length_d_srch ] = i;
            length_d_srch++;
        }
    }

    /* Convolution */
    for( i = max_lag_8kHz + 3; i >= min_lag_8kHz; i-- ) {
        d_comp[ i ] += d_comp[ i - 1 ] + d_comp[ i - 2 ] + d_comp[ i - 3 ];
    }

    length_d_comp = 0;
    for( i = min_lag_8kHz; i < max_lag_8kHz + 4; i++ ) {
        if( d_comp[ i ] > 0 ) {
            d_comp[ length_d_comp ] = i - 2;
```

```
            length_d_comp++;
        }
    }

    /*********************************************************************
******
    ** SECOND STAGE, operating at 8 kHz, on lag sections with high correlation
    *********************************************************************
********/

    /*********************************************************************
**
    ** Scale signal down to avoid correlations measures from overflowing
    *********************************************************************
**/
    /* find scaling as max scaling for each subframe */
    shift = SKP_FIX_P_Ana_find_scaling( signal_8kHz, frame_length_8kHz, sf_length
_8kHz );
    if( shift > 0 ) {
        for( i = 0; i < frame_length_8kHz; i++ ) {
            signal_8kHz[ i ] = SKP_RSHIFT( signal_8kHz[ i ], shift );
        }
    }

    /*********************************************************************
*****
    * Find energy of each subframe projected onto its history, for a range of del
ays
    *********************************************************************
****/
    SKP_memset( C, 0, PITCH_EST_NB_SUBFR * ( ( PITCH_EST_MAX_LAG >> 1 ) + 5 ) * s
izeof( SKP_int16 ) );

    target_ptr = &signal_8kHz[ frame_length_4kHz ]; /* point to middle of frame *
/
    for( k = 0; k < PITCH_EST_NB_SUBFR; k++ ) {

        /* Check that we are within range of the array */
        SKP_assert( target_ptr >= signal_8kHz );
        SKP_assert( target_ptr + sf_length_8kHz <= signal_8kHz + frame_length_8kH
z );

        energy_target = SKP_Silk_inner_prod_aligned( target_ptr, target_ptr, sf_l
ength_8kHz );
        // ToDo: Calculate 1 / energy_target here and save one division inside ne
xt for loop
        for( j = 0; j < length_d_comp; j++ ) {
            d = d_comp[ j ];
            basis_ptr = target_ptr - d;

            /* Check that we are within range of the array */
            SKP_assert( basis_ptr >= signal_8kHz );
            SKP_assert( basis_ptr + sf_length_8kHz <= signal_8kHz + frame_length_
8kHz );

            cross_corr   = SKP_Silk_inner_prod_aligned( target_ptr, basis_ptr, sf
_length_8kHz );
            energy_basis = SKP_Silk_inner_prod_aligned( basis_ptr,  basis_ptr, sf
_length_8kHz );
            if( cross_corr > 0 ) {
                energy = SKP_max( energy_target, energy_basis ); /* Find max to m
ake sure first division < 1.0 */
```

```
                lz = SKP_Silk_CLZ32( cross_corr );
                lshift = SKP_LIMIT( lz - 1, 0, 15 );
                temp32 = SKP_DIV32( SKP_LSHIFT( cross_corr, lshift ), SKP_RSHIFT(
 energy, 15 - lshift ) + 1 ); /* Q15 */
```

```
                SKP_assert( temp32 == SKP_SAT16( temp32 ) );
                temp32 = SKP_SMULWB( cross_corr, temp32 ); /* Q(-1), cc * ( cc /
max(b, t) ) */
                temp32 = SKP_ADD_SAT32( temp32, temp32 );  /* Q(0) */
                lz = SKP_Silk_CLZ32( temp32 );
                lshift = SKP_LIMIT( lz - 1, 0, 15 );
                energy = SKP_min( energy_target, energy_basis );
                C[ k ][ d ] = SKP_DIV32( SKP_LSHIFT( temp32, lshift ), SKP_RSHIFT
( energy, 15 - lshift ) + 1 ); // Q15
            } else {
                C[ k ][ d ] = 0;
            }
        }
        target_ptr += sf_length_8kHz;
    }

    /* search over lag range and lags codebook */
    /* scale factor for lag codebook, as a function of center lag */

    CCmax   = SKP_int32_MIN;
    CCmax_b = SKP_int32_MIN;

    CBimax = 0; /* To avoid returning undefined lag values */
    lag = -1;   /* To check if lag with strong enough correlation has been found
*/

    if( prevLag > 0 ) {
        if( Fs_kHz == 12 ) {
            prevLag = SKP_DIV32_16( SKP_LSHIFT( prevLag, 1 ), 3 );
        } else if( Fs_kHz == 16 ) {
            prevLag = SKP_RSHIFT( prevLag, 1 );
        } else if( Fs_kHz == 24 ) {
            prevLag = SKP_DIV32_16( prevLag, 3 );
        }
        prevLag_log2_Q7 = SKP_Silk_lin2log( (SKP_int32)prevLag );
    } else {
        prevLag_log2_Q7 = 0;
    }
    SKP_assert( search_thres2_Q15 == SKP_SAT16( search_thres2_Q15 ) );
    corr_thres_Q15 = SKP_RSHIFT( SKP_SMULBB( search_thres2_Q15, search_thres2_Q15
 ), 13 );

    /* If input is 8 khz use a larger codebook here because it is last stage */
    if( Fs_kHz == 8 && complexity > SigProc_PITCH_EST_MIN_COMPLEX ) {
        nb_cbks_stage2 = PITCH_EST_NB_CBKS_STAGE2_EXT;
    } else {
        nb_cbks_stage2 = PITCH_EST_NB_CBKS_STAGE2;
    }

    for( k = 0; k < length_d_srch; k++ ) {
        d = d_srch[ k ];
        for( j = 0; j < nb_cbks_stage2; j++ ) {
```

```
            CC[ j ] = 0;
            for( i = 0; i < PITCH_EST_NB_SUBFR; i++ ) {
                /* Try all codebooks */
                CC[ j ] = CC[ j ] + (SKP_int32)C[ i ][ d + SKP_Silk_CB_lags_stage
2[ i ][ j ] ];
            }
        }
        /* Find best codebook */
        CCmax_new = SKP_int32_MIN;
        CBimax_new = 0;
        for( i = 0; i < nb_cbks_stage2; i++ ) {
            if( CC[ i ] > CCmax_new ) {
                CCmax_new = CC[ i ];
                CBimax_new = i;
            }
        }

        /* Bias towards shorter lags */
        lag_log2_Q7 = SKP_Silk_lin2log( (SKP_int32)d ); /* Q7 */
        SKP_assert( lag_log2_Q7 == SKP_SAT16( lag_log2_Q7 ) );
        SKP_assert( PITCH_EST_NB_SUBFR * PITCH_EST_SHORTLAG_BIAS_Q15 == SKP_SAT16
( PITCH_EST_NB_SUBFR * PITCH_EST_SHORTLAG_BIAS_Q15 ) );
        CCmax_new_b = CCmax_new - SKP_RSHIFT( SKP_SMULBB( PITCH_EST_NB_SUBFR * PI
TCH_EST_SHORTLAG_BIAS_Q15, lag_log2_Q7 ), 7 ); /* Q15 */

        /* Bias towards previous lag */
        SKP_assert( PITCH_EST_NB_SUBFR * PITCH_EST_PREVLAG_BIAS_Q15 == SKP_SAT16(
 PITCH_EST_NB_SUBFR * PITCH_EST_PREVLAG_BIAS_Q15 ) );
        if( prevLag > 0 ) {
            delta_lag_log2_sqr_Q7 = lag_log2_Q7 - prevLag_log2_Q7;
            SKP_assert( delta_lag_log2_sqr_Q7 == SKP_SAT16( delta_lag_log2_sqr_Q7
 ) );
            delta_lag_log2_sqr_Q7 = SKP_RSHIFT( SKP_SMULBB( delta_lag_log2_sqr_Q7
, delta_lag_log2_sqr_Q7 ), 7 );
            prev_lag_bias_Q15 = SKP_RSHIFT( SKP_SMULBB( PITCH_EST_NB_SUBFR * PITC
H_EST_PREVLAG_BIAS_Q15, ( *LTPCorr_Q15 ) ), 15 ); /* Q15 */
            prev_lag_bias_Q15 = SKP_DIV32( SKP_MUL( prev_lag_bias_Q15, delta_lag_
log2_sqr_Q7 ), delta_lag_log2_sqr_Q7 + ( 1 << 6 ) );
            CCmax_new_b -= prev_lag_bias_Q15; /* Q15 */
        }

        if( CCmax_new_b > CCmax_b && CCmax_new > corr_thres_Q15 ) {
            CCmax_b = CCmax_new_b;
            CCmax   = CCmax_new;
            lag     = d;
            CBimax  = CBimax_new;
        }
    }

    if( lag == -1 ) {
        /* No suitable candidate found */
        SKP_memset( pitch_out, 0, PITCH_EST_NB_SUBFR * sizeof( SKP_int ) );
        *LTPCorr_Q15  = 0;
        *lagIndex     = 0;
        *contourIndex = 0;
        return 1;
```

```
    }

    if( Fs_kHz > 8 ) {

        /*******************************************************************
******
        ** Scale input signal down to avoid correlations measures from overflowin
g
        *******************************************************************
******/
        /* find scaling as max scaling for each subframe */
        shift = SKP_FIX_P_Ana_find_scaling( signal, frame_length, sf_length );
        if( shift > 0 ) {
            /* Move signal to scratch mem because the input signal should be unch
anged */
            /* Reuse the 32 bit scratch mem vector, use a 16 bit pointer from now
 */
            input_signal_ptr = (SKP_int16*)scratch_mem;
            for( i = 0; i < frame_length; i++ ) {
                input_signal_ptr[ i ] = SKP_RSHIFT( signal[ i ], shift );
            }
        } else {
            input_signal_ptr = (SKP_int16*)signal;
        }
        /*******************************************************************
*********/

        /* Search in original signal */

        CBimax_old = CBimax;
        /* Compensate for decimation */
        SKP_assert( lag == SKP_SAT16( lag ) );
        if( Fs_kHz == 12 ) {
            lag = SKP_RSHIFT( SKP_SMULBB( lag, 3 ), 1 );
        } else if( Fs_kHz == 16 ) {
            lag = SKP_LSHIFT( lag, 1 );
        } else {
            lag = SKP_SMULBB( lag, 3 );
        }

        lag = SKP_LIMIT( lag, min_lag, max_lag );
        start_lag = SKP_max_int( lag - 2, min_lag );
        end_lag   = SKP_min_int( lag + 2, max_lag );
        lag_new   = lag;                                    /* to avoid undefined
 lag */
        CBimax    = 0;                                      /* to avoid undefin
ed lag */
        SKP_assert( SKP_LSHIFT( CCmax, 13 ) >= 0 );
        *LTPCorr_Q15 = (SKP_int)SKP_Silk_SQRT_APPROX( SKP_LSHIFT( CCmax, 13 ) );
/* Output normalized correlation */

        CCmax = SKP_int32_MIN;
        /* pitch lags according to second stage */
        for( k = 0; k < PITCH_EST_NB_SUBFR; k++ ) {
            pitch_out[ k ] = lag + 2 * SKP_Silk_CB_lags_stage2[ k ][ CBimax_old ]
;
        }
        /* Calculate the correlations and energies needed in stage 3 */
```

```
        SKP_FIX_P_Ana_calc_corr_st3(  crosscorr_st3, input_signal_ptr, start_lag,
 sf_length, complexity );
        SKP_FIX_P_Ana_calc_energy_st3( energies_st3, input_signal_ptr, start_lag,
 sf_length, complexity );

        lag_counter = 0;
        SKP_assert( lag == SKP_SAT16( lag ) );
        contour_bias = SKP_DIV32_16( PITCH_EST_FLATCONTOUR_BIAS_Q20, lag );

        /* Setup cbk parameters acording to complexity setting */
        cbk_size   = (SKP_int)SKP_Silk_cbk_sizes_stage3[   complexity ];
        cbk_offset = (SKP_int)SKP_Silk_cbk_offsets_stage3[ complexity ];

        for( d = start_lag; d <= end_lag; d++ ) {
            for( j = cbk_offset; j < ( cbk_offset + cbk_size ); j++ ) {
                cross_corr = 0;
                energy     = 0;
                for( k = 0; k < PITCH_EST_NB_SUBFR; k++ ) {
                    SKP_assert( PITCH_EST_NB_SUBFR == 4 );
                    energy     += SKP_RSHIFT( energies_st3[  k ][ j ][ lag_counte
r ], 2 ); /* use mean, to avoid overflow */
                    SKP_assert( energy >= 0 );
                    cross_corr += SKP_RSHIFT( crosscorr_st3[ k ][ j ][ lag_counte
r ], 2 ); /* use mean, to avoid overflow */
                }
                if( cross_corr > 0 ) {
                    /* Divide cross_corr / energy and get result in Q15 */
                    lz = SKP_Silk_CLZ32( cross_corr );
                    /* Divide with result in Q13, cross_corr could be larger than
 energy */
                    lshift = SKP_LIMIT( lz - 1, 0, 13 );
                    CCmax_new = SKP_DIV32( SKP_LSHIFT( cross_corr, lshift ), SKP_
RSHIFT( energy, 13 - lshift ) + 1 );
                    CCmax_new = SKP_SAT16( CCmax_new );
                    CCmax_new = SKP_SMULWB( cross_corr, CCmax_new );
                    /* Saturate */
                    if( CCmax_new > SKP_RSHIFT( SKP_int32_MAX, 3 ) ) {
                        CCmax_new = SKP_int32_MAX;
                    } else {
                        CCmax_new = SKP_LSHIFT( CCmax_new, 3 );
                    }
                    /* Reduce depending on flatness of contour */
                    diff = j - SKP_RSHIFT( PITCH_EST_NB_CBKS_STAGE3_MAX, 1 );
                    diff = SKP_MUL( diff, diff );
                    diff = SKP_int16_MAX - SKP_RSHIFT( SKP_MUL( contour_bias, dif
f ), 5 ); /* Q20 -> Q15 */
                    SKP_assert( diff == SKP_SAT16( diff ) );
                    CCmax_new = SKP_LSHIFT( SKP_SMULWB( CCmax_new, diff ), 1 );
                } else {
                    CCmax_new = 0;
                }

                if( CCmax_new > CCmax ) {
                    CCmax   = CCmax_new;
                    lag_new = d;
```

```
                    CBimax  = j;
                }
            }
            lag_counter++;
        }

        for( k = 0; k < PITCH_EST_NB_SUBFR; k++ ) {
            pitch_out[ k ] = lag_new + SKP_Silk_CB_lags_stage3[ k ][ CBimax ];
        }
        *lagIndex = lag_new - min_lag;
        *contourIndex = CBimax;
    } else {
        /* Save Lags and correlation */
        CCmax = SKP_max( CCmax, 0 );
        *LTPCorr_Q15 = (SKP_int)SKP_Silk_SQRT_APPROX( SKP_LSHIFT( CCmax, 13 ) );
/* Output normalized correlation */
        for( k = 0; k < PITCH_EST_NB_SUBFR; k++ ) {
            pitch_out[ k ] = lag + SKP_Silk_CB_lags_stage2[ k ][ CBimax ];
        }
        *lagIndex = lag - min_lag_8kHz;
        *contourIndex = CBimax;
    }
    SKP_assert( *lagIndex >= 0 );
    /* return as voiced */
    return 0;
}

/*****************************************************************************/
/* Calculates the correlations used in stage 3 search. In order to cover */
/* the whole lag codebook for all the searched offset lags (lag +- 2),   */
/*****************************************************************************/
void SKP_FIX_P_Ana_calc_corr_st3(
    SKP_int32        cross_corr_st3[ PITCH_EST_NB_SUBFR ][ PITCH_EST_NB_CBKS_STAG
E3_MAX ][ PITCH_EST_NB_STAGE3_LAGS ],/* (O) 3 DIM correlation array */
    const SKP_int16  signal[],                          /* I vector to correlate
      */
    SKP_int          start_lag,                         /* I lag offset to search a
round */
    SKP_int          sf_length,                         /* I length of a 5 ms subfr
ame   */
    SKP_int          complexity                         /* I Complexity setting
      */
)
{
    const SKP_int16 *target_ptr, *basis_ptr;
    SKP_int32    cross_corr;
    SKP_int        i, j, k, lag_counter;
    SKP_int        cbk_offset, cbk_size, delta, idx;
    SKP_int32    scratch_mem[ SCRATCH_SIZE ];

    SKP_assert( complexity >= SigProc_PITCH_EST_MIN_COMPLEX );
    SKP_assert( complexity <= SigProc_PITCH_EST_MAX_COMPLEX );

    cbk_offset = SKP_Silk_cbk_offsets_stage3[ complexity ];
```

```
    cbk_size   = SKP_Silk_cbk_sizes_stage3[   complexity ];

    target_ptr = &signal[ SKP_LSHIFT( sf_length, 2 ) ]; /* Pointer to middle of f
rame */
    for( k = 0; k < PITCH_EST_NB_SUBFR; k++ ) {
        lag_counter = 0;

        /* Calculate the correlations for each subframe */
        for( j = SKP_Silk_Lag_range_stage3[ complexity ][ k ][ 0 ]; j <= SKP_Silk
_Lag_range_stage3[ complexity ][ k ][ 1 ]; j++ ) {
            basis_ptr = target_ptr - ( start_lag + j );
            cross_corr = SKP_Silk_inner_prod_aligned( (SKP_int16*)target_ptr, (SK
P_int16*)basis_ptr, sf_length );
            SKP_assert( lag_counter < SCRATCH_SIZE );
            scratch_mem[ lag_counter ] = cross_corr;
            lag_counter++;
        }

        delta = SKP_Silk_Lag_range_stage3[ complexity ][ k ][ 0 ];
        for( i = cbk_offset; i < ( cbk_offset + cbk_size ); i++ ) {
            /* Fill out the 3 dim array that stores the correlations for */
            /* each code_book vector for each start lag */
            idx = SKP_Silk_CB_lags_stage3[ k ][ i ] - delta;
            for( j = 0; j < PITCH_EST_NB_STAGE3_LAGS; j++ ) {
                SKP_assert( idx + j < SCRATCH_SIZE );
                SKP_assert( idx + j < lag_counter );
                cross_corr_st3[ k ][ i ][ j ] = scratch_mem[ idx + j ];
            }
        }
        target_ptr += sf_length;
    }
}

/***********************************************************************/
/* Calculate the energies for first two subframes. The energies are */
/* calculated recursively.                                             */
/***********************************************************************/
void SKP_FIX_P_Ana_calc_energy_st3(
    SKP_int32        energies_st3[ PITCH_EST_NB_SUBFR ][ PITCH_EST_NB_CBKS_STAGE3
_MAX ][ PITCH_EST_NB_STAGE3_LAGS ],/* (O) 3 DIM energy array */
    const SKP_int16  signal[],                          /* I vector to calc energy
in    */
    SKP_int          start_lag,                         /* I lag offset to search a
round */
    SKP_int          sf_length,                         /* I length of one 5 ms sub
frame */
    SKP_int          complexity                         /* I Complexity setting
      */
)
{
    const SKP_int16 *target_ptr, *basis_ptr;
    SKP_int32    energy;
    SKP_int        k, i, j, lag_counter;
    SKP_int        cbk_offset, cbk_size, delta, idx;
    SKP_int32    scratch_mem[ SCRATCH_SIZE ];
```

```
    SKP_assert( complexity >= SigProc_PITCH_EST_MIN_COMPLEX );
    SKP_assert( complexity <= SigProc_PITCH_EST_MAX_COMPLEX );

    cbk_offset = SKP_Silk_cbk_offsets_stage3[ complexity ];
    cbk_size   = SKP_Silk_cbk_sizes_stage3[   complexity ];

    target_ptr = &signal[ SKP_LSHIFT( sf_length, 2 ) ];
    for( k = 0; k < PITCH_EST_NB_SUBFR; k++ ) {
        lag_counter = 0;

        /* Calculate the energy for first lag */
        basis_ptr = target_ptr - ( start_lag + SKP_Silk_Lag_range_stage3[ complex
ity ][ k ][ 0 ] );
        energy = SKP_Silk_inner_prod_aligned( basis_ptr, basis_ptr, sf_length );
        SKP_assert( energy >= 0 );
        scratch_mem[ lag_counter ] = energy;
        lag_counter++;

        for( i = 1; i < ( SKP_Silk_Lag_range_stage3[ complexity ][ k ][ 1 ] - SKP
_Silk_Lag_range_stage3[ complexity ][ k ][ 0 ] + 1 ); i++ ) {
            /* remove part outside new window */
            energy -= SKP_SMULBB( basis_ptr[ sf_length - i ], basis_ptr[ sf_lengt
h - i ] );
            SKP_assert( energy >= 0 );

            /* add part that comes into window */
            energy = SKP_ADD_SAT32( energy, SKP_SMULBB( basis_ptr[ -i ], basis_pt
r[ -i ] ) );
            SKP_assert( energy >= 0 );
            SKP_assert( lag_counter < SCRATCH_SIZE );
            scratch_mem[ lag_counter ] = energy;
            lag_counter++;
        }

        delta = SKP_Silk_Lag_range_stage3[ complexity ][ k ][ 0 ];
        for( i = cbk_offset; i < ( cbk_offset + cbk_size ); i++ ) {
            /* Fill out the 3 dim array that stores the correlations for    */
            /* each code_book vector for each start lag                     */
            idx = SKP_Silk_CB_lags_stage3[ k ][ i ] - delta;
            for( j = 0; j < PITCH_EST_NB_STAGE3_LAGS; j++ ) {
                SKP_assert( idx + j < SCRATCH_SIZE );
                SKP_assert( idx + j < lag_counter );
                energies_st3[ k ][ i ][ j ] = scratch_mem[ idx + j ];
                SKP_assert( energies_st3[ k ][ i ][ j ] >= 0.0f );
            }
        }
        target_ptr += sf_length;
    }
}

SKP_int32 SKP_FIX_P_Ana_find_scaling(
    const SKP_int16  *signal,
```

```
    const SKP_int    signal_length,
    const SKP_int    sum_sqr_len
)
{
    SKP_int32 nbits, x_max;

    x_max = SKP_Silk_int16_array_maxabs( signal, signal_length );

    if( x_max < SKP_int16_MAX ) {
        /* Number of bits needed for the sum of the squares */
        nbits = 32 - SKP_Silk_CLZ32( SKP_SMULBB( x_max, x_max ) );
    } else {
        /* Here we don't know if x_max should have been SKP_int16_MAX + 1, so we
expect the worst case */
        nbits = 30;
    }
    nbits += 17 - SKP_Silk_CLZ16( sum_sqr_len );

    /* Without a guarantee of saturation, we need to keep the 31st bit free */
    if( nbits < 31 ) {
        return 0;
    } else {
        return( nbits - 30 );
    }
}
```

A.81. src/SKP_Silk_pitch_est_defines.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SIGPROCFIX_PITCH_EST_DEFINES_H
#define SIGPROCFIX_PITCH_EST_DEFINES_H

#include "SKP_Silk_SigProc_FIX.h"
#include "SKP_Silk_common_pitch_est_defines.h"

/*********************************************************/
/* Definitions For Fix pitch estimator                   */
/*********************************************************/

#define PITCH_EST_SHORTLAG_BIAS_Q15         6554    /* 0.2f. for logarithmic weig
hting    */
#define PITCH_EST_PREVLAG_BIAS_Q15          6554    /* Prev lag bias    */
#define PITCH_EST_FLATCONTOUR_BIAS_Q20      52429   /* 0.05f */

#endif
```

A.82. src/SKP_Silk_pitch_est_tables.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_typedef.h"
#include "SKP_Silk_pitch_est_defines.h"

/*********************************************************/
/* Auto Generated File from generate_pitch_est_tables.m */
/*********************************************************/

const SKP_int16 SKP_Silk_CB_lags_stage2[PITCH_EST_NB_SUBFR][PITCH_EST_NB_CBKS_STA
GE2_EXT] =
{
    {0, 2,-1,-1,-1, 0, 0, 1, 1, 0, 1},
    {0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0},
    {0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0},
    {0,-1, 2, 1, 0, 1, 1, 0, 0,-1,-1}
};

const SKP_int16 SKP_Silk_CB_lags_stage3[PITCH_EST_NB_SUBFR][PITCH_EST_NB_CBKS_STA
GE3_MAX] =
{
    {-9,-7,-6,-5,-5,-4,-4,-3,-3,-2,-2,-2,-1,-1,-1, 0, 0, 0, 1, 1, 0, 1, 2, 2, 2,
3, 3, 4, 4, 5, 6, 5, 6, 8},
```

```
    {-3,-2,-2,-2,-1,-1,-1,-1,-1, 0, 0,-1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0,
1, 1, 2, 1, 2, 2, 2, 2, 3},
    { 3, 3, 2, 2, 2, 2, 1, 2, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,-1, 0,
0,-1,-1,-1,-1,-1,-2,-2,-2},
    { 9, 8, 6, 5, 6, 5, 4, 4, 3, 3, 2, 2, 2, 1, 0, 1, 1, 0, 0, 0,-1,-1,-1,-2,-2,-
2,-3,-3,-4,-4,-5,-5,-6,-7}
 };

const SKP_int16 SKP_Silk_Lag_range_stage3[ SigProc_PITCH_EST_MAX_COMPLEX + 1 ] [
PITCH_EST_NB_SUBFR ][ 2 ] =
{
    /* Lags to search for low number of stage3 cbks */
    {
        {-2,6},
        {-1,5},
        {-1,5},
        {-2,7}
    },
    /* Lags to search for middle number of stage3 cbks */
    {
        {-4,8},
        {-1,6},
        {-1,6},
        {-4,9}
    },
    /* Lags to search for max number of stage3 cbks */
    {
        {-9,12},
        {-3,7},
        {-2,7},
        {-7,13}
    }
};

const SKP_int16 SKP_Silk_cbk_sizes_stage3[SigProc_PITCH_EST_MAX_COMPLEX + 1] =
{
    PITCH_EST_NB_CBKS_STAGE3_MIN,
    PITCH_EST_NB_CBKS_STAGE3_MID,
    PITCH_EST_NB_CBKS_STAGE3_MAX
};

const SKP_int16 SKP_Silk_cbk_offsets_stage3[SigProc_PITCH_EST_MAX_COMPLEX + 1] =
{
    ((PITCH_EST_NB_CBKS_STAGE3_MAX - PITCH_EST_NB_CBKS_STAGE3_MIN) >> 1),
    ((PITCH_EST_NB_CBKS_STAGE3_MAX - PITCH_EST_NB_CBKS_STAGE3_MID) >> 1),
    0
};
```

A.83. src/SKP_Silk_PLC.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"
#include "SKP_Silk_PLC.h"

#define NB_ATT 2
static const SKP_int16 HARM_ATT_Q15[NB_ATT]              = { 32440, 31130 }; /* 0
.99, 0.95 */
static const SKP_int16 PLC_RAND_ATTENUATE_V_Q15[NB_ATT]  = { 31130, 26214 }; /* 0
.95, 0.8 */
static const SKP_int16 PLC_RAND_ATTENUATE_UV_Q15[NB_ATT] = { 32440, 29491 }; /* 0
.99, 0.9 */

void SKP_Silk_PLC_Reset(
    SKP_Silk_decoder_state      *psDec              /* I/O Decoder state        *
/
)
{
    psDec->sPLC.pitchL_Q8 = SKP_RSHIFT( psDec->frame_length, 1 );
}

void SKP_Silk_PLC(
    SKP_Silk_decoder_state      *psDec,             /* I Decoder state          *
/
    SKP_Silk_decoder_control    *psDecCtrl,         /* I Decoder control        *
/
    SKP_int16                   signal[],           /* O Concealed signal       *
/
    SKP_int                     length,             /* I length of residual     *
/
    SKP_int                     lost                /* I Loss flag              *
/
```

```
)
{
    /* PLC control function */
    if( psDec->fs_kHz != psDec->sPLC.fs_kHz ) {
        SKP_Silk_PLC_Reset( psDec );
        psDec->sPLC.fs_kHz = psDec->fs_kHz;
    }

    if( lost ) {
        /****************************/
        /* Generate Signal          */
        /****************************/
        SKP_Silk_PLC_conceal( psDec, psDecCtrl, signal, length );
    } else {
        /****************************/
        /* Update state             */
        /****************************/
        SKP_Silk_PLC_update( psDec, psDecCtrl, signal, length );
    }
}

/*************************************************/
/* Update state of PLC                           */
/*************************************************/
void SKP_Silk_PLC_update(
    SKP_Silk_decoder_state      *psDec,             /* (I/O) Decoder state
   */
    SKP_Silk_decoder_control    *psDecCtrl,         /* (I/O) Decoder control
   */
    SKP_int16                   signal[],
    SKP_int                     length
)
{
    SKP_int32 LTP_Gain_Q14, temp_LTP_Gain_Q14;
    SKP_int   i, j;
    SKP_Silk_PLC_struct *psPLC;

    psPLC = &psDec->sPLC;

    /* Update parameters used in case of packet loss */
    psDec->prev_sigtype = psDecCtrl->sigtype;
    LTP_Gain_Q14 = 0;
    if( psDecCtrl->sigtype == SIG_TYPE_VOICED ) {
        /* Find the parameters for the last subframe which contains a pitch pulse
 */
        for( j = 0; j * psDec->subfr_length  < psDecCtrl->pitchL[ NB_SUBFR - 1 ];
 j++ ) {
            temp_LTP_Gain_Q14 = 0;
            for( i = 0; i < LTP_ORDER; i++ ) {
                temp_LTP_Gain_Q14 += psDecCtrl->LTPCoef_Q14[ ( NB_SUBFR - 1 - j )
 * LTP_ORDER  + i ];
            }
            if( temp_LTP_Gain_Q14 > LTP_Gain_Q14 ) {
```

```
                LTP_Gain_Q14 = temp_LTP_Gain_Q14;
                SKP_memcpy( psPLC->LTPCoef_Q14,
                    &psDecCtrl->LTPCoef_Q14[ SKP_SMULBB( NB_SUBFR - 1 - j, LTP_OR
DER ) ],
                    LTP_ORDER * sizeof( SKP_int16 ) );

                psPLC->pitchL_Q8 = SKP_LSHIFT( psDecCtrl->pitchL[ NB_SUBFR - 1 -
j ], 8 );
            }
        }

#if USE_SINGLE_TAP
        SKP_memset( psPLC->LTPCoef_Q14, 0, LTP_ORDER * sizeof( SKP_int16 ) );
        psPLC->LTPCoef_Q14[ LTP_ORDER / 2 ] = LTP_Gain_Q14;
#endif

        /* Limit LT coefs */
        if( LTP_Gain_Q14 < V_PITCH_GAIN_START_MIN_Q14 ) {
            SKP_int   scale_Q10;
            SKP_int32 tmp;

            tmp = SKP_LSHIFT( V_PITCH_GAIN_START_MIN_Q14, 10 );
            scale_Q10 = SKP_DIV32( tmp, SKP_max( LTP_Gain_Q14, 1 ) );
            for( i = 0; i < LTP_ORDER; i++ ) {
                psPLC->LTPCoef_Q14[ i ] = SKP_RSHIFT( SKP_SMULBB( psPLC->LTPCoef_
Q14[ i ], scale_Q10 ), 10 );
            }
        } else if( LTP_Gain_Q14 > V_PITCH_GAIN_START_MAX_Q14 ) {
            SKP_int   scale_Q14;
            SKP_int32 tmp;

            tmp = SKP_LSHIFT( V_PITCH_GAIN_START_MAX_Q14, 14 );
            scale_Q14 = SKP_DIV32( tmp, SKP_max( LTP_Gain_Q14, 1 ) );
            for( i = 0; i < LTP_ORDER; i++ ) {
                psPLC->LTPCoef_Q14[ i ] = SKP_RSHIFT( SKP_SMULBB( psPLC->LTPCoef_
Q14[ i ], scale_Q14 ), 14 );
            }
        }
    } else {
        psPLC->pitchL_Q8 = SKP_LSHIFT( SKP_SMULBB( psDec->fs_kHz, 18 ), 8 );
        SKP_memset( psPLC->LTPCoef_Q14, 0, LTP_ORDER * sizeof( SKP_int16 ));
    }

    /* Save LPC coeficients */
    SKP_memcpy( psPLC->prevLPC_Q12, psDecCtrl->PredCoef_Q12[ 1 ], psDec->LPC_orde
r * sizeof( SKP_int16 ) );
    psPLC->prevLTP_scale_Q14 = psDecCtrl->LTP_scale_Q14;

    /* Save Gains */
    SKP_memcpy( psPLC->prevGain_Q16, psDecCtrl->Gains_Q16, NB_SUBFR * sizeof( SKP
_int32 ) );
}

void SKP_Silk_PLC_conceal(
```

```
    SKP_Silk_decoder_state      *psDec,             /* I/O Decoder state */
    SKP_Silk_decoder_control    *psDecCtrl,         /* I/O Decoder control */
    SKP_int16                   signal[],           /* O concealed signal */
    SKP_int                     length              /* I length of residual */
)
{
    SKP_int   i, j, k;
    SKP_int16 *B_Q14, exc_buf[ MAX_FRAME_LENGTH ], *exc_buf_ptr;
    SKP_int16 rand_scale_Q14, A_Q12_tmp[ MAX_LPC_ORDER ];
    SKP_int32 rand_seed, harm_Gain_Q15, rand_Gain_Q15;
    SKP_int   lag, idx, shift1, shift2;
    SKP_int32 energy1, energy2, *rand_ptr, *pred_lag_ptr, Atmp;
    SKP_int32 sig_Q10[ MAX_FRAME_LENGTH ], *sig_Q10_ptr, LPC_exc_Q10, LPC_pred_Q1
0,  LTP_pred_Q14;
    SKP_Silk_PLC_struct *psPLC;

    psPLC = &psDec->sPLC;

    /* Update LTP buffer */
    SKP_memcpy( psDec->sLTP_Q16, &psDec->sLTP_Q16[ psDec->frame_length ], psDec->
frame_length * sizeof( SKP_int32 ) );

    /* LPC concealment. Apply BWE to previous LPC */
    SKP_Silk_bwexpander( psPLC->prevLPC_Q12, psDec->LPC_order, BWE_COEF_Q16 );

    /* Find random noise component */
    /* Scale previous excitation signal */
    exc_buf_ptr = exc_buf;
    for( k = ( NB_SUBFR >> 1 ); k < NB_SUBFR; k++ ) {
        for( i = 0; i < psDec->subfr_length; i++ ) {
            exc_buf_ptr[ i ] = ( SKP_int16 )SKP_RSHIFT(
                SKP_SMULWW( psDec->exc_Q10[ i + k * psDec->subfr_length ], psPLC-
>prevGain_Q16[ k ] ), 10 );
        }
        exc_buf_ptr += psDec->subfr_length;
    }
    /* Find the subframe with lowest energy of the last two and use that as rando
m noise generator */
    SKP_Silk_sum_sqr_shift( &energy1, &shift1, exc_buf,                         p
sDec->subfr_length );
    SKP_Silk_sum_sqr_shift( &energy2, &shift2, &exc_buf[ psDec->subfr_length ], p
sDec->subfr_length );

    if( SKP_RSHIFT( energy1, shift2 ) < SKP_RSHIFT( energy1, shift2 ) ) {
        /* First sub-frame has lowest energy */
        rand_ptr = &psDec->exc_Q10[ SKP_max_int( 0, 3 * psDec->subfr_length - RAN
D_BUF_SIZE ) ];
    } else {
        /* Second sub-frame has lowest energy */
        rand_ptr = &psDec->exc_Q10[ SKP_max_int( 0, psDec->frame_length - RAND_BU
F_SIZE ) ];
    }

    /* Setup Gain to random noise component */
    B_Q14          = psPLC->LTPCoef_Q14;
    rand_scale_Q14 = psPLC->randScale_Q14;
```

```
    /* Setup attenuation gains */
    harm_Gain_Q15 = HARM_ATT_Q15[ SKP_min_int( NB_ATT - 1, psDec->lossCnt ) ];
    if( psDec->prev_sigtype == SIG_TYPE_VOICED ) {
        rand_Gain_Q15 = PLC_RAND_ATTENUATE_V_Q15[  SKP_min_int( NB_ATT - 1, psDec
->lossCnt ) ];
    } else {
        rand_Gain_Q15 = PLC_RAND_ATTENUATE_UV_Q15[ SKP_min_int( NB_ATT - 1, psDec
->lossCnt ) ];
    }

    /* First Lost frame */
    if( psDec->lossCnt == 0 ) {
        rand_scale_Q14 = (1 << 14 );

        /* Reduce random noise Gain for voiced frames */
        if( psDec->prev_sigtype == SIG_TYPE_VOICED ) {
            for( i = 0; i < LTP_ORDER; i++ ) {
                rand_scale_Q14 -= B_Q14[ i ];
            }
            rand_scale_Q14 = SKP_max_16( 3277, rand_scale_Q14 ); /* 0.2 */
            rand_scale_Q14 = ( SKP_int16 )SKP_RSHIFT( SKP_SMULBB( rand_scale_Q14,
 psPLC->prevLTP_scale_Q14 ), 14 );
        }

        /* Reduce random noise for unvoiced frames with high LPC gain */
        if( psDec->prev_sigtype == SIG_TYPE_UNVOICED ) {
            SKP_int32 invGain_Q30, down_scale_Q30;

            SKP_Silk_LPC_inverse_pred_gain( &invGain_Q30, psPLC->prevLPC_Q12, psD
ec->LPC_order );

            down_scale_Q30 = SKP_min_32( SKP_RSHIFT( ( 1 << 30 ), LOG2_INV_LPC_GA
IN_HIGH_THRES ), invGain_Q30 );
            down_scale_Q30 = SKP_max_32( SKP_RSHIFT( ( 1 << 30 ), LOG2_INV_LPC_GA
IN_LOW_THRES ), down_scale_Q30 );
            down_scale_Q30 = SKP_LSHIFT( down_scale_Q30, LOG2_INV_LPC_GAIN_HIGH_T
HRES );

            rand_Gain_Q15 = SKP_RSHIFT( SKP_SMULWB( down_scale_Q30, rand_Gain_Q15
 ), 14 );
        }
    }

    rand_seed           = psPLC->rand_seed;
    lag                 = SKP_RSHIFT_ROUND( psPLC->pitchL_Q8, 8 );
    psDec->sLTP_buf_idx = psDec->frame_length;

    /***************************/
    /* LTP synthesis filtering */
    /***************************/
    sig_Q10_ptr = sig_Q10;
    for( k = 0; k < NB_SUBFR; k++ ) {
        /* Setup pointer */
        pred_lag_ptr = &psDec->sLTP_Q16[ psDec->sLTP_buf_idx - lag + LTP_ORDER /
2 ];
        for( i = 0; i < psDec->subfr_length; i++ ) {
            rand_seed = SKP_RAND( rand_seed );
```

```
            idx = SKP_RSHIFT( rand_seed, 25 ) & RAND_BUF_MASK;

            /* Unrolled loop */
            LTP_pred_Q14 = SKP_SMULWB(               pred_lag_ptr[  0 ], B_Q14[ 0
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -1 ], B_Q14[ 1
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -2 ], B_Q14[ 2
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -3 ], B_Q14[ 3
 ] );
            LTP_pred_Q14 = SKP_SMLAWB( LTP_pred_Q14, pred_lag_ptr[ -4 ], B_Q14[ 4
 ] );
            pred_lag_ptr++;

            /* Generate LPC residual */
            LPC_exc_Q10 = SKP_LSHIFT( SKP_SMULWB( rand_ptr[ idx ], rand_scale_Q14
 ), 2 ); /* Random noise part */
            LPC_exc_Q10 = SKP_ADD32( LPC_exc_Q10, SKP_RSHIFT_ROUND( LTP_pred_Q14,
 4 ) );  /* Harmonic part */

            /* Update states */
            psDec->sLTP_Q16[ psDec->sLTP_buf_idx ] = SKP_LSHIFT( LPC_exc_Q10, 6 )
;
            psDec->sLTP_buf_idx++;

            /* Save LPC residual */
            sig_Q10_ptr[ i ] = LPC_exc_Q10;
        }
        sig_Q10_ptr += psDec->subfr_length;
        /* Gradually reduce LTP gain */
        for( j = 0; j < LTP_ORDER; j++ ) {
            B_Q14[ j ] = SKP_RSHIFT( SKP_SMULBB( harm_Gain_Q15, B_Q14[ j ] ), 15
);
        }
        /* Gradually reduce excitation gain */
        rand_scale_Q14 = SKP_RSHIFT( SKP_SMULBB( rand_scale_Q14, rand_Gain_Q15 ),
 15 );

        /* Slowly increase pitch lag */
        psPLC->pitchL_Q8 += SKP_SMULWB( psPLC->pitchL_Q8, PITCH_DRIFT_FAC_Q16 );
        psPLC->pitchL_Q8 = SKP_min_32( psPLC->pitchL_Q8, SKP_LSHIFT( SKP_SMULBB(
MAX_PITCH_LAG_MS, psDec->fs_kHz ), 8 ) );
        lag = SKP_RSHIFT_ROUND( psPLC->pitchL_Q8, 8 );
    }

    /***************************/
    /* LPC synthesis filtering */
    /***************************/
    sig_Q10_ptr = sig_Q10;
    /* Preload LPC coeficients to array on stack. Gives small performance gain */
    SKP_memcpy( A_Q12_tmp, psPLC->prevLPC_Q12, psDec->LPC_order * sizeof( SKP_int
16 ) );
    SKP_assert( psDec->LPC_order >= 10 ); /* check that unrolling works */
    for( k = 0; k < NB_SUBFR; k++ ) {
        for( i = 0; i < psDec->subfr_length; i++ ){
            /* unrolled */
            Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 0 ] );    /* read two coefficient
s at once */
            LPC_pred_Q10 = SKP_SMULWB(               psDec->sLPC_Q14[ MAX_LPC_ORD
ER + i -  1 ], Atmp );
            LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC_ORD
```

```
ER + i -  2 ], Atmp );
```

```
            Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 2 ] );
            LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC_ORD
ER + i -  3 ], Atmp );
            LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC_ORD
ER + i -  4 ], Atmp );
            Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 4 ] );
            LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC_ORD
ER + i -  5 ], Atmp );
            LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC_ORD
ER + i -  6 ], Atmp );
            Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 6 ] );
            LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC_ORD
ER + i -  7 ], Atmp );
            LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC_ORD
ER + i -  8 ], Atmp );
            Atmp = *( ( SKP_int32* )&A_Q12_tmp[ 8 ] );
            LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC_ORD
ER + i -  9 ], Atmp );
            LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC_ORD
ER + i - 10 ], Atmp );
            for( j = 10 ; j < psDec->LPC_order ; j+=2 ) {
                Atmp = *( ( SKP_int32* )&A_Q12_tmp[ j ] );
                LPC_pred_Q10 = SKP_SMLAWB( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  1 - j ], Atmp );
                LPC_pred_Q10 = SKP_SMLAWT( LPC_pred_Q10, psDec->sLPC_Q14[ MAX_LPC
_ORDER + i -  2 - j ], Atmp );
            }

            /* Add prediction to LPC residual */
            sig_Q10_ptr[ i ] = SKP_ADD32( sig_Q10_ptr[ i ], LPC_pred_Q10 );

            /* Update states */
            psDec->sLPC_Q14[ MAX_LPC_ORDER + i ] = SKP_LSHIFT( sig_Q10_ptr[ i ],
4 );
        }
        sig_Q10_ptr += psDec->subfr_length;
        /* Update LPC filter state */
        SKP_memcpy( psDec->sLPC_Q14, &psDec->sLPC_Q14[ psDec->subfr_length ], MAX
_LPC_ORDER * sizeof( SKP_int32 ) );
    }

    /* Scale with Gain */
    for( i = 0; i < psDec->frame_length; i++ ) {
        signal[ i ] = ( SKP_int16 )SKP_SAT16( SKP_RSHIFT_ROUND( SKP_SMULWW( sig_Q
10[ i ], psPLC->prevGain_Q16[ NB_SUBFR - 1 ] ), 10 ) );
    }

    /***************************************/
    /* Update states                       */
    /***************************************/
    psPLC->rand_seed     = rand_seed;
    psPLC->randScale_Q14 = rand_scale_Q14;
    for( i = 0; i < NB_SUBFR; i++ ) {
        psDecCtrl->pitchL[ i ] = lag;
    }
}

/* Glues concealed frames with new good recieved frames             */
void SKP_Silk_PLC_glue_frames(
    SKP_Silk_decoder_state      *psDec,             /* I/O decoder state    */
    SKP_Silk_decoder_control    *psDecCtrl,         /* I/O Decoder control  */
```

```
    SKP_int16                   signal[],           /* I/O signal           */
    SKP_int                     length              /* I length of residual */
)
{
    SKP_int   i, energy_shift;
    SKP_int32 energy;
    SKP_Silk_PLC_struct *psPLC;
    psPLC = &psDec->sPLC;

    if( psDec->lossCnt ) {
        /* Calculate energy in concealed residual */
        SKP_Silk_sum_sqr_shift( &psPLC->conc_energy, &psPLC->conc_energy_shift, s
ignal, length );

        psPLC->last_frame_lost = 1;
    } else {
        if( psDec->sPLC.last_frame_lost ) {
            /* Calculate residual in decoded signal if last frame was lost */
            SKP_Silk_sum_sqr_shift( &energy, &energy_shift, signal, length );

            /* Normalize energies */
            if( energy_shift > psPLC->conc_energy_shift ) {
                psPLC->conc_energy = SKP_RSHIFT( psPLC->conc_energy, energy_shift
 - psPLC->conc_energy_shift );
            } else if( energy_shift < psPLC->conc_energy_shift ) {
                energy = SKP_RSHIFT( energy, psPLC->conc_energy_shift - energy_sh
ift );
            }

            /* Fade in the energy difference */
            if( energy > psPLC->conc_energy ) {
                SKP_int32 frac_Q24, LZ;
                SKP_int32 gain_Q12, slope_Q12;

                LZ = SKP_Silk_CLZ32( psPLC->conc_energy );
                LZ = LZ - 1;
                psPLC->conc_energy = SKP_LSHIFT( psPLC->conc_energy, LZ );
                energy = SKP_RSHIFT( energy, SKP_max_32( 24 - LZ, 0 ) );

                frac_Q24 = SKP_DIV32( psPLC->conc_energy, SKP_max( energy, 1 ) );

                gain_Q12 = SKP_Silk_SQRT_APPROX( frac_Q24 );
                slope_Q12 = SKP_DIV32_16( ( 1 << 12 ) - gain_Q12, length );

                for( i = 0; i < length; i++ ) {
                    signal[ i ] = SKP_RSHIFT( SKP_MUL( gain_Q12, signal[ i ] ), 1
2 );
                    gain_Q12 += slope_Q12;
                    gain_Q12 = SKP_min( gain_Q12, ( 1 << 12 ) );
                }
            }
        }
```

```
        psPLC->last_frame_lost = 0;

    }
}
```

A.84. src/SKP_Silk_PLC.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_PLC_FIX_H
#define SKP_SILK_PLC_FIX_H

#include "SKP_Silk_main_FIX.h"

#define BWE_COEF_Q16                    64880           /* 0.99 in Q16
          */
#define V_PITCH_GAIN_START_MIN_Q14      11469           /* 0.7 in Q14
          */
#define V_PITCH_GAIN_START_MAX_Q14      15565           /* 0.95 in Q14
          */
#define MAX_PITCH_LAG_MS                18
#define SA_THRES_Q8                     50
#define USE_SINGLE_TAP                  1
#define RAND_BUF_SIZE                   128
```

```
#define RAND_BUF_MASK                   (RAND_BUF_SIZE - 1)
#define LOG2_INV_LPC_GAIN_HIGH_THRES    4               /* 2^4 = 12 dB LPC gain
        */
#define LOG2_INV_LPC_GAIN_LOW_THRES     8               /* 2^8 = 24 dB LPC gain
        */
#define PITCH_DRIFT_FAC_Q16             655             /* 0.01 in Q16
        */

void SKP_Silk_PLC_Reset(
    SKP_Silk_decoder_state      *psDec              /* I/O Decoder state        *
/
);

void SKP_Silk_PLC(
    SKP_Silk_decoder_state      *psDec,             /* I/O Decoder state        *
/
    SKP_Silk_decoder_control    *psDecCtrl,         /* I/O Decoder control      *
/
    SKP_int16                   signal[],           /* I/O  signal              *
/
    SKP_int                     length,             /* I length of residual     *
/
    SKP_int                     lost                /* I Loss flag              *
/
);

void SKP_Silk_PLC_update(
    SKP_Silk_decoder_state      *psDec,             /* I/O Decoder state        *
/
    SKP_Silk_decoder_control    *psDecCtrl,         /* I/O Decoder control      *
/
    SKP_int16                   signal[],
    SKP_int                     length
);

void SKP_Silk_PLC_conceal(
    SKP_Silk_decoder_state      *psDec,             /* I/O Decoder state        *
/
    SKP_Silk_decoder_control    *psDecCtrl,         /* I/O Decoder control      *
/
    SKP_int16                   signal[],           /* O LPC residual signal    *
/
    SKP_int                     length              /* I length of signal       *
/
);

void SKP_Silk_PLC_glue_frames(
    SKP_Silk_decoder_state      *psDec,             /* I/O decoder state        *
/
    SKP_Silk_decoder_control    *psDecCtrl,         /* I/O Decoder control      *
/
    SKP_int16                   signal[],           /* I/O signal               *
/
    SKP_int                     length              /* I length of signal       *
/
);

#endif
```

A.85. src/SKP_Silk_prefilter_FIX.c

```
/**********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
```

```
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"
#include "SKP_Silk_perceptual_parameters_FIX.h"

/* SKP_Silk_prefilter. Prefilter for finding Quantizer input signal */
SKP_INLINE void SKP_Silk_prefilt_FIX(
    SKP_Silk_prefilter_state_FIX *P,                    /* I/O state
            */
    SKP_int32   st_res_Q12[],                           /* I short term residual
signal       */
    SKP_int16   xw[],                                   /* O prefiltered signal
            */
    SKP_int32   HarmShapeFIRPacked_Q12,                 /* I Harmonic shaping coe
ficients     */
    SKP_int     Tilt_Q14,                               /* I Tilt shaping coefici
ent          */
    SKP_int32   LF_shp_Q14,                             /* I Low-frequancy shapin
g coeficients*/
    SKP_int     lag,                                    /* I Lag for harmonic sha
ping         */
    SKP_int     length                                  /* I Length of signals
            */
);

void SKP_Silk_prefilter_FIX(
    SKP_Silk_encoder_state_FIX          *psEnc,         /* I/O  Encoder state FIX
                        */
    const SKP_Silk_encoder_control_FIX  *psEncCtrl,     /* I    Encoder control F
IX                      */
    SKP_int16                           xw[],           /* O    Weighted signal
                        */
    const SKP_int16                     x[]             /* I    Speech signal
                        */
)
{
    SKP_Silk_prefilter_state_FIX *P = &psEnc->sPrefilt;
    SKP_int   j, k, lag;
```

```
    SKP_int32 tmp_32, B_Q12;
    const SKP_int16 *AR1_shp_Q13;
    const SKP_int16 *px;
    SKP_int16 *pxw, *pst_res;
    SKP_int   HarmShapeGain_Q12, Tilt_Q14, LF_shp_Q14;
    SKP_int32 HarmShapeFIRPacked_Q12;
    SKP_int32 x_filt_Q12[ MAX_FRAME_LENGTH / NB_SUBFR ], filterState[ MAX_LPC_ORD
ER ];
    SKP_int16 st_res[ ( MAX_FRAME_LENGTH / NB_SUBFR ) + MAX_LPC_ORDER ];

    /* Setup pointers */
    px  = x;
    pxw = xw;
    lag = P->lagPrev;
    for( k = 0; k < NB_SUBFR; k++ ) {
        /* Update Variables that change per sub frame */
        if( psEncCtrl->sCmn.sigtype == SIG_TYPE_VOICED ) {
            lag = psEncCtrl->sCmn.pitchL[ k ];
        }

        /* Noise shape parameters */
        HarmShapeGain_Q12 = SKP_SMULWB( psEncCtrl->HarmShapeGain_Q14[ k ], 16384
- psEncCtrl->HarmBoost_Q14[ k ] );
        SKP_assert( HarmShapeGain_Q12 >= 0 );
        HarmShapeFIRPacked_Q12  =                       SKP_RSHIFT( HarmShapeGai
n_Q12, 2 );
        HarmShapeFIRPacked_Q12 |= SKP_LSHIFT( ( SKP_int32 )SKP_RSHIFT( HarmShapeG
ain_Q12, 1 ), 16 );
        Tilt_Q14    = psEncCtrl->Tilt_Q14[   k ];
        LF_shp_Q14  = psEncCtrl->LF_shp_Q14[ k ];
        AR1_shp_Q13 = &psEncCtrl->AR1_Q13[   k * SHAPE_LPC_ORDER_MAX ];

        /* Short term FIR filtering*/
        SKP_memset( filterState, 0, psEnc->sCmn.shapingLPCOrder * sizeof( SKP_int
32 ) );
        SKP_Silk_MA_Prediction_Q13( px - psEnc->sCmn.shapingLPCOrder, AR1_shp_Q13
, filterState,
            st_res, psEnc->sCmn.subfr_length + psEnc->sCmn.shapingLPCOrder, psEnc
->sCmn.shapingLPCOrder );

        pst_res = st_res + psEnc->sCmn.shapingLPCOrder; /* Point to first sample
*/

        /* reduce (mainly) low frequencies during harmonic emphasis */
        B_Q12 = SKP_RSHIFT_ROUND( psEncCtrl->GainsPre_Q14[ k ], 2 );
        tmp_32 = SKP_SMLABB( INPUT_TILT_Q26, psEncCtrl->HarmBoost_Q14[ k ], HarmS
hapeGain_Q12 ); /* Q26 */
        tmp_32 = SKP_SMLABB( tmp_32, psEncCtrl->coding_quality_Q14, HIGH_RATE_INP
UT_TILT_Q12 );  /* Q26 */
        tmp_32 = SKP_SMULWB( tmp_32, -psEncCtrl->GainsPre_Q14[ k ] );
                /* Q24 */
        tmp_32 = SKP_RSHIFT_ROUND( tmp_32, 12 );
                /* Q12 */
        B_Q12 |= SKP_LSHIFT( SKP_SAT16( tmp_32 ), 16 );

        /* NOTE: the code below loads two int16 values in an int32, and multiplie
s each using the   */
        /* SMLABB and SMLABT instructions. On a big-endian CPU the two int16 vari
ables would be     */
        /* loaded in reverse order and the code will give the wrong result. In th
at case swapping   */
        /* the SMLABB and SMLABT instructions should solve the problem.
```

```
                    */
        x_filt_Q12[ 0 ] = SKP_SMLABT( SKP_SMULBB( pst_res[ 0 ], B_Q12 ), P->sHarm
HP, B_Q12 );
```

```
        for( j = 1; j < psEnc->sCmn.subfr_length; j++ ) {
            x_filt_Q12[ j ] = SKP_SMLABT( SKP_SMULBB( pst_res[ j ], B_Q12 ), pst_
res[ j - 1 ], B_Q12 );
        }
        P->sHarmHP = pst_res[ psEnc->sCmn.subfr_length - 1 ];

        SKP_Silk_prefilt_FIX( P, x_filt_Q12, pxw, HarmShapeFIRPacked_Q12, Tilt_Q1
4,
            LF_shp_Q14, lag, psEnc->sCmn.subfr_length );

        px  += psEnc->sCmn.subfr_length;
        pxw += psEnc->sCmn.subfr_length;
    }

    P->lagPrev = psEncCtrl->sCmn.pitchL[ NB_SUBFR - 1 ];
}

/* SKP_Silk_prefilter. Prefilter for finding Quantizer input signal
            */
SKP_INLINE void SKP_Silk_prefilt_FIX(
    SKP_Silk_prefilter_state_FIX *P,                    /* I/O state
            */
    SKP_int32   st_res_Q12[],                           /* I short term residual
signal       */
    SKP_int16   xw[],                                   /* O prefiltered signal
            */
    SKP_int32   HarmShapeFIRPacked_Q12,                 /* I Harmonic shaping coe
ficients     */
    SKP_int     Tilt_Q14,                               /* I Tilt shaping coefici
ent          */
    SKP_int32   LF_shp_Q14,                             /* I Low-frequancy shapin
g coeficients*/
    SKP_int     lag,                                    /* I Lag for harmonic sha
ping         */
    SKP_int     length                                  /* I Length of signals
            */
)
{
    SKP_int   i, idx, LTP_shp_buf_idx;
    SKP_int32 n_LTP_Q12, n_Tilt_Q10, n_LF_Q10;
    SKP_int32 sLF_MA_shp_Q12, sLF_AR_shp_Q12;
    SKP_int16 *LTP_shp_buf;

    /* To speed up use temp variables instead of using the struct */
    LTP_shp_buf     = P->sLTP_shp1;
    LTP_shp_buf_idx = P->sLTP_shp_buf_idx1;
    sLF_AR_shp_Q12  = P->sLF_AR_shp1_Q12;
    sLF_MA_shp_Q12  = P->sLF_MA_shp1_Q12;

    for( i = 0; i < length; i++ ) {
        if( lag > 0 ) {
            /* unrolled loop */
            SKP_assert( HARM_SHAPE_FIR_TAPS == 3 );
            idx = lag + LTP_shp_buf_idx;
            n_LTP_Q12 = SKP_SMULBB(            LTP_shp_buf[ ( idx - HARM_SHAPE_FI
R_TAPS / 2 - 1) & LTP_MASK ], HarmShapeFIRPacked_Q12 );
            n_LTP_Q12 = SKP_SMLABT( n_LTP_Q12, LTP_shp_buf[ ( idx - HARM_SHAPE_FI
R_TAPS / 2    ) & LTP_MASK ], HarmShapeFIRPacked_Q12 );
            n_LTP_Q12 = SKP_SMLABB( n_LTP_Q12, LTP_shp_buf[ ( idx - HARM_SHAPE_FI
R_TAPS / 2 + 1) & LTP_MASK ], HarmShapeFIRPacked_Q12 );
        } else {
            n_LTP_Q12 = 0;
```

```
        }

        n_LF_Q10   = SKP_SMLAWB( SKP_SMULWT( sLF_AR_shp_Q12, LF_shp_Q14 ), sLF_MA
_shp_Q12, LF_shp_Q14 );
        n_Tilt_Q10 = SKP_SMULWB( sLF_AR_shp_Q12, Tilt_Q14 );

        sLF_AR_shp_Q12 = SKP_SUB32( st_res_Q12[ i ], SKP_LSHIFT( n_Tilt_Q10, 2 )
);
        sLF_MA_shp_Q12 = SKP_SUB32( sLF_AR_shp_Q12,  SKP_LSHIFT( n_LF_Q10,   2 )
);

        LTP_shp_buf_idx                 = ( LTP_shp_buf_idx - 1 ) & LTP_MASK;
        LTP_shp_buf[ LTP_shp_buf_idx ] = ( SKP_int16 )SKP_SAT16( SKP_RSHIFT_ROUND
( sLF_MA_shp_Q12, 12 ) );

        xw[i] = ( SKP_int16 )SKP_SAT16( SKP_RSHIFT_ROUND( SKP_SUB32( sLF_MA_shp_Q
12, n_LTP_Q12 ), 12 ) );
    }

    /* Copy temp variable back to state */
    P->sLF_AR_shp1_Q12   = sLF_AR_shp_Q12;
    P->sLF_MA_shp1_Q12   = sLF_MA_shp_Q12;
    P->sLTP_shp_buf_idx1 = LTP_shp_buf_idx;
}
```

A.86. src/SKP_Silk_process_gains_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
```

```
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/* Processing of gains */
void SKP_Silk_process_gains_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,         /* I/O  Encoder state_FIX
                        */
    SKP_Silk_encoder_control_FIX    *psEncCtrl      /* I/O  Encoder control_FIX
                        */
)
{
    SKP_Silk_shape_state_FIX    *psShapeSt = &psEnc->sShape;
    SKP_int     k;
    SKP_int32   s_Q16, InvMaxSqrVal_Q16, gain, gain_squared, ResNrg, ResNrgPart;

    /* Gain reduction when LTP coding gain is high */
    if( psEncCtrl->sCmn.sigtype == SIG_TYPE_VOICED ) {
        /*s = -0.5f * SKP_sigmoid( 0.25f * ( psEncCtrl->LTPredCodGain - 12.0f ) )
; */
        s_Q16 = -SKP_Silk_sigm_Q15( SKP_RSHIFT_ROUND( psEncCtrl->LTPredCodGain_Q7
 - (12 << 7), 4 ) );
        for( k = 0; k < NB_SUBFR; k++ ) {
            psEncCtrl->Gains_Q16[ k ] = SKP_SMLAWB( psEncCtrl->Gains_Q16[ k ], ps
EncCtrl->Gains_Q16[ k ], s_Q16 );
        }
    }

    /* Limit the quantized signal */
    /*  69 = 21.0f + 16/0.33    */
    InvMaxSqrVal_Q16 = SKP_DIV32_16( SKP_Silk_log2lin(
        SKP_SMULWB( (69 << 7) - psEncCtrl->current_SNR_dB_Q7, SKP_FIX_CONST( 0.33
, 16 )) ), psEnc->sCmn.subfr_length );

    for( k = 0; k < NB_SUBFR; k++ ) {
        /* Soft limit on ratio residual energy and squared gains */
        ResNrg     = psEncCtrl->ResNrg[ k ];
        ResNrgPart = SKP_SMULWW( ResNrg, InvMaxSqrVal_Q16 );
        if( psEncCtrl->ResNrgQ[ k ] > 0 ) {
            if( psEncCtrl->ResNrgQ[ k ] < 32 ) {
                ResNrgPart = SKP_RSHIFT_ROUND( ResNrgPart, psEncCtrl->ResNrgQ[ k
] );
            } else {
                ResNrgPart = 0;
            }
        } else if( psEncCtrl->ResNrgQ[k] != 0 ) {
            if( ResNrgPart > SKP_RSHIFT( SKP_int32_MAX, -psEncCtrl->ResNrgQ[ k ]
) ) {
                ResNrgPart = SKP_int32_MAX;
            } else {
                ResNrgPart = SKP_LSHIFT( ResNrgPart, -psEncCtrl->ResNrgQ[ k ] );
            }
        }
        gain = psEncCtrl->Gains_Q16[ k ];
        gain_squared = SKP_ADD_SAT32( ResNrgPart, SKP_SMMUL( gain, gain ) );
```

```
        if( gain_squared < SKP_int16_MAX ) {
            /* recalculate with higher precision */
            gain_squared = SKP_SMLAWW( SKP_LSHIFT( ResNrgPart, 16 ), gain, gain )
;
            SKP_assert( gain_squared > 0 );
            gain = SKP_Silk_SQRT_APPROX( gain_squared );                  /* Q8
 */
            psEncCtrl->Gains_Q16[ k ] = SKP_LSHIFT_SAT32( gain, 8 );        /* Q1
6  */
        } else {
            gain = SKP_Silk_SQRT_APPROX( gain_squared );                  /* Q0
 */
            psEncCtrl->Gains_Q16[ k ] = SKP_LSHIFT_SAT32( gain, 16 );       /* Q1
6  */
        }
    }

    /* Noise shaping quantization */
    SKP_Silk_gains_quant( psEncCtrl->sCmn.GainsIndices, psEncCtrl->Gains_Q16,
        &psShapeSt->LastGainIndex, psEnc->sCmn.nFramesInPayloadBuf );
    /* Set quantizer offset for voiced signals. Larger offset when LTP coding gai
n is low or tilt is high (ie low-pass) */
    if( psEncCtrl->sCmn.sigtype == SIG_TYPE_VOICED ) {
        if( psEncCtrl->LTPredCodGain_Q7 + SKP_RSHIFT( psEncCtrl->input_tilt_Q15,
8 ) > ( 1 << 7 ) ) {
            psEncCtrl->sCmn.QuantOffsetType = 0;
        } else {
            psEncCtrl->sCmn.QuantOffsetType = 1;
        }
    }

    /* Quantizer boundary adjustment */
    if( psEncCtrl->sCmn.sigtype == SIG_TYPE_VOICED ) {
        psEncCtrl->Lambda_Q10 = SKP_FIX_CONST( 1.3, 10 )
                    - SKP_SMULWB( SKP_FIX_CONST( 0.5, 18 ), psEnc->speech_activity_
Q8       )
                    - SKP_SMULWB( SKP_FIX_CONST( 0.3, 12 ), psEncCtrl->input_qualit
y_Q14    )
                    + SKP_SMULBB( SKP_FIX_CONST( 0.2, 10 ), psEncCtrl->sCmn.QuantOf
fsetType )
                    - SKP_SMULWB( SKP_FIX_CONST( 0.1, 12 ), psEncCtrl->coding_quali
ty_Q14   );
    } else {
        psEncCtrl->Lambda_Q10 = SKP_FIX_CONST( 1.3, 10 )
                    - SKP_SMULWB( SKP_FIX_CONST( 0.5, 18 ), psEnc->speech_activity_
Q8       )
                    - SKP_SMULWB( SKP_FIX_CONST( 0.4, 12 ), psEncCtrl->input_qualit
y_Q14    )
                    + SKP_SMULBB( SKP_FIX_CONST( 0.4, 10 ), psEncCtrl->sCmn.QuantOf
fsetType )
                    - SKP_SMULWB( SKP_FIX_CONST( 0.1, 12 ), psEncCtrl->coding_quali
ty_Q14   );
    }
    SKP_assert( psEncCtrl->Lambda_Q10 >= 0 );
    SKP_assert( psEncCtrl->Lambda_Q10 < SKP_FIX_CONST( 2, 10 ) );

}
```

A.87. src/SKP_Silk_process_NLSFs_FIX.c

/***

```
#include "SKP_Silk_main_FIX.h"

/* Limit, stabilize, convert and quantize NLSFs.    */
void SKP_Silk_process_NLSFs_FIX(
    SKP_Silk_encoder_state_FIX      *psEnc,             /* I/O  Encoder state FIX
                            */
    SKP_Silk_encoder_control_FIX    *psEncCtrl,         /* I/O  Encoder control F
IX                          */
    SKP_int                         *pNLSF_Q15          /* I/O  Normalized LSFs (
quant out) (0 - (2^15-1))  */
)
{
    SKP_int     doInterpolate;
    SKP_int     pNLSFW_Q6[ MAX_LPC_ORDER ];
    SKP_int     NLSF_mu_Q15, NLSF_mu_fluc_red_Q16;
    SKP_int32   i_sqr_Q15;
    const SKP_Silk_NLSF_CB_struct *psNLSF_CB;

    /* Used only for NLSF interpolation */
    SKP_int     pNLSF0_temp_Q15[ MAX_LPC_ORDER ];
    SKP_int     pNLSFW0_temp_Q6[ MAX_LPC_ORDER ];
    SKP_int     i;

    SKP_assert( psEnc->speech_activity_Q8 >=   0 );
    SKP_assert( psEnc->speech_activity_Q8 <= 256 );
```

```
    SKP_assert( psEncCtrl->sparseness_Q8  >=   0 );
    SKP_assert( psEncCtrl->sparseness_Q8  <= 256 );
    SKP_assert( psEncCtrl->sCmn.sigtype == SIG_TYPE_VOICED || psEncCtrl->sCmn.sig
type == SIG_TYPE_UNVOICED );

    /***********************/
    /* Calculate mu values */
    /***********************/
    if( psEncCtrl->sCmn.sigtype == SIG_TYPE_VOICED ) {
        /* NLSF_mu           = 0.002f - 0.001f * psEnc->speech_activity; */
        /* NLSF_mu_fluc_red  = 0.1f   - 0.05f  * psEnc->speech_activity; */
        NLSF_mu_Q15          = SKP_SMLAWB(   66,   -8388, psEnc->speech_activity_
Q8 );
        NLSF_mu_fluc_red_Q16 = SKP_SMLAWB( 6554, -838848, psEnc->speech_activity_
Q8 );
    } else {
        /* NLSF_mu           = 0.005f - 0.004f * psEnc->speech_activity; */
        /* NLSF_mu_fluc_red  = 0.2f   - 0.1f   * psEnc->speech_activity - 0.1f *
psEncCtrl->sparseness; */
        NLSF_mu_Q15          = SKP_SMLAWB(   164,   -33554, psEnc->speech_activit
y_Q8 );
        NLSF_mu_fluc_red_Q16 = SKP_SMLAWB( 13107, -1677696, psEnc->speech_activit
y_Q8 + psEncCtrl->sparseness_Q8 );
    }
    SKP_assert( NLSF_mu_Q15          >= 0     );
    SKP_assert( NLSF_mu_Q15          <= 164   );
    SKP_assert( NLSF_mu_fluc_red_Q16 >= 0     );
    SKP_assert( NLSF_mu_fluc_red_Q16 <= 13107 );

    NLSF_mu_Q15 = SKP_max( NLSF_mu_Q15, 1 );

    /* Calculate NLSF weights */
    TIC(NLSF_weights_FIX)
    SKP_Silk_NLSF_VQ_weights_laroia( pNLSFW_Q6, pNLSF_Q15, psEnc->sCmn.predictLPC
Order );
    TOC(NLSF_weights_FIX)

    /* Update NLSF weights for interpolated NLSFs */
    doInterpolate = ( psEnc->sCmn.useInterpolatedNLSFs == 1 ) && ( psEncCtrl->sCm
n.NLSFInterpCoef_Q2 < ( 1 << 2 ) );
    if( doInterpolate ) {

        /* Calculate the interpolated NLSF vector for the first half */
        SKP_Silk_interpolate( pNLSF0_temp_Q15, psEnc->sPred.prev_NLSFq_Q15, pNLSF
_Q15,
            psEncCtrl->sCmn.NLSFInterpCoef_Q2, psEnc->sCmn.predictLPCOrder );

        /* Calculate first half NLSF weights for the interpolated NLSFs */
        TIC(NLSF_weights_FIX)
        SKP_Silk_NLSF_VQ_weights_laroia( pNLSFW0_temp_Q6, pNLSF0_temp_Q15, psEnc-
>sCmn.predictLPCOrder );
        TOC(NLSF_weights_FIX)

        /* Update NLSF weights with contribution from first half */
        i_sqr_Q15 = SKP_LSHIFT( SKP_SMULBB( psEncCtrl->sCmn.NLSFInterpCoef_Q2, ps
EncCtrl->sCmn.NLSFInterpCoef_Q2 ), 11 );
        for( i = 0; i < psEnc->sCmn.predictLPCOrder; i++ ) {
            pNLSFW_Q6[ i ] = SKP_SMLAWB( SKP_RSHIFT( pNLSFW_Q6[ i ], 1 ), pNLSFW0
_temp_Q6[ i ], i_sqr_Q15 );
            SKP_assert( pNLSFW_Q6[ i ] <= SKP_int16_MAX );
```

```
            SKP_assert( pNLSFW_Q6[ i ] >= 1 );
        }
    }

    /* Set pointer to the NLSF codebook for the current signal type and LPC order
 */
    psNLSF_CB = psEnc->sCmn.psNLSF_CB[ psEncCtrl->sCmn.sigtype ];

    /* Quantize NLSF parameters given the trained NLSF codebooks */
    TIC(MSVQ_encode_FIX)
    SKP_Silk_NLSF_MSVQ_encode_FIX( psEncCtrl->sCmn.NLSFIndices, pNLSF_Q15, psNLSF
_CB,
        psEnc->sPred.prev_NLSFq_Q15, pNLSFW_Q6, NLSF_mu_Q15, NLSF_mu_fluc_red_Q16
,
        psEnc->sCmn.NLSF_MSVQ_Survivors, psEnc->sCmn.predictLPCOrder, psEnc->sCmn
.first_frame_after_reset );
    TOC(MSVQ_encode_FIX)

    /* Convert quantized NLSFs back to LPC coefficients */
    SKP_Silk_NLSF2A_stable( psEncCtrl->PredCoef_Q12[ 1 ], pNLSF_Q15, psEnc->sCmn.
predictLPCOrder );

    if( doInterpolate ) {
        /* Calculate the interpolated, quantized LSF vector for the first half */
        SKP_Silk_interpolate( pNLSF0_temp_Q15, psEnc->sPred.prev_NLSFq_Q15, pNLSF
_Q15,
            psEncCtrl->sCmn.NLSFInterpCoef_Q2, psEnc->sCmn.predictLPCOrder );

        /* Convert back to LPC coefficients */
        SKP_Silk_NLSF2A_stable( psEncCtrl->PredCoef_Q12[ 0 ], pNLSF0_temp_Q15, ps
Enc->sCmn.predictLPCOrder );

    } else {
        /* Copy LPC coefficients for first half from second half */
        SKP_memcpy( psEncCtrl->PredCoef_Q12[ 0 ], psEncCtrl->PredCoef_Q12[ 1 ], p
sEnc->sCmn.predictLPCOrder * sizeof( SKP_int16 ) );
    }
}
```

A.88. src/SKP_Silk_pulses_to_bytes.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
```

```
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*
 * File Name:   SKP_Silk_pulses_to_bytes.c
 */

#include <stdlib.h>
#include "SKP_Silk_main.h"

/* nBytes = sum_over_shell_blocks( POLY_FIT_0 + POLY_FIT_1 * sum_abs_val + POLY_F
IT_2 * sum_abs_val^2 ) */
#define POLY_FIT_0_Q15     12520
#define POLY_FIT_1_Q15     15862
#define POLY_FIT_2_Q20     -9222 // ToDo better training with

/* Predict number of bytes used to encode q */
SKP_int SKP_Silk_pulses_to_bytes( /* O  Return value, predicted number of bytes u
sed to encode q */
    SKP_Silk_encoder_state          *psEncC,        /* I/O  Encoder State */
    SKP_int                         q[]             /* I    Pulse signal  */
)
{
    SKP_int i, j, iter, *q_ptr;
    SKP_int32 sum_abs_val, nBytes, acc_nBytes;
    /* Take the absolute value of the pulses */
    iter = psEncC->frame_length / SHELL_CODEC_FRAME_LENGTH;

    /* Calculate rate as a nonlinaer mapping of sum abs value of each Shell block
 */
    q_ptr      = q;
    acc_nBytes = 0;
    for( j = 0; j < iter; j++ ) {
        sum_abs_val = 0;
        for(i = 0; i < SHELL_CODEC_FRAME_LENGTH; i+=4){
            sum_abs_val += SKP_abs( q_ptr[ i + 0 ] );
            sum_abs_val += SKP_abs( q_ptr[ i + 1 ] );
            sum_abs_val += SKP_abs( q_ptr[ i + 2 ] );
            sum_abs_val += SKP_abs( q_ptr[ i + 3 ] );
        }
```

```
        /* Calculate nBytes used for thi sshell frame */
        nBytes = SKP_SMULWB( SKP_SMULBB( sum_abs_val, sum_abs_val ), POLY_FIT_2_Q
20 );  // Q4
        nBytes = SKP_LSHIFT_SAT32( nBytes, 11 );
       // Q15
        nBytes += SKP_SMULBB( sum_abs_val, POLY_FIT_1_Q15 );
       // Q15
        nBytes += POLY_FIT_0_Q15;
       // Q15

        acc_nBytes += nBytes;

        q_ptr += SHELL_CODEC_FRAME_LENGTH; /* update pointer */
    }

    acc_nBytes = SKP_RSHIFT_ROUND( acc_nBytes, 15 );
       // Q0
    acc_nBytes = SKP_SAT16( acc_nBytes ); // just to be sure
       // Q0

    return(( SKP_int )acc_nBytes);
}
```

A.89. src/SKP_Silk_quant_LTP_gains_FIX.c

```
/**********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
**********************************************************************/
```

```
#include "SKP_Silk_main_FIX.h"

void SKP_Silk_quant_LTP_gains_FIX(
    SKP_int16               B_Q14[],                /* I/O  (un)quantized LTP gai
ns     */
    SKP_int                 cbk_index[],            /* O    Codebook Index
       */
    SKP_int                 *periodicity_index,     /* O    Periodicity Index
       */
    const SKP_int32         W_Q18[],                /* I    Error Weights in Q18
       */
    SKP_int                 mu_Q8,                  /* I    Mu value (R/D tradeof
f)     */
    SKP_int                 lowComplexity           /* I    Flag for low complexi
ty     */
)
{
    SKP_int             j, k, temp_idx[ NB_SUBFR ], cbk_size;
    const SKP_uint16    *cdf_ptr;
    const SKP_int16     *cl_ptr;
    const SKP_int16     *cbk_ptr_Q14;
    const SKP_int16     *b_Q14_ptr;
    const SKP_int32     *W_Q18_ptr;
    SKP_int32           rate_dist_subfr, rate_dist, min_rate_dist;



    /***************************************************/
    /* iterate over different codebooks with different */
    /* rates/distortions, and choose best */
    /***************************************************/
    min_rate_dist = SKP_int32_MAX;
    for( k = 0; k < 3; k++ ) {
        cdf_ptr     = SKP_Silk_LTP_gain_CDF_ptrs[     k ];
        cl_ptr      = SKP_Silk_LTP_gain_BITS_Q6_ptrs[ k ];
        cbk_ptr_Q14 = SKP_Silk_LTP_vq_ptrs_Q14[       k ];
        cbk_size    = SKP_Silk_LTP_vq_sizes[          k ];

        /* Setup pointer to first subframe */
        W_Q18_ptr = W_Q18;
        b_Q14_ptr = B_Q14;

        rate_dist = 0;
        for( j = 0; j < NB_SUBFR; j++ ) {

            SKP_Silk_VQ_WMat_EC_FIX(
                &temp_idx[ j ],         /* O    index of best codebook vector
                   */
                &rate_dist_subfr,       /* O    best weighted quantization error
+ mu * rate            */
                b_Q14_ptr,              /* I    input vector to be quantized
                   */
                W_Q18_ptr,              /* I    weighting matrix
                   */
                cbk_ptr_Q14,            /* I    codebook
                   */
                cl_ptr,                 /* I    code length for each codebook vec
tor                */
                mu_Q8,                  /* I    tradeoff between weighted error a
nd rate            */
                cbk_size                /* I    number of vectors in codebook
                   */
```

```
            );

            rate_dist = SKP_ADD_POS_SAT32( rate_dist, rate_dist_subfr );

            b_Q14_ptr += LTP_ORDER;
            W_Q18_ptr += LTP_ORDER * LTP_ORDER;
        }

        /* Avoid never finding a codebook */
        rate_dist = SKP_min( SKP_int32_MAX - 1, rate_dist );

        if( rate_dist < min_rate_dist ) {
            min_rate_dist = rate_dist;
            SKP_memcpy( cbk_index, temp_idx, NB_SUBFR * sizeof( SKP_int ) );
            *periodicity_index = k;
        }

        /* Break early in low-complexity mode if rate distortion is below thresho
ld */
        if( lowComplexity && ( rate_dist < SKP_Silk_LTP_gain_middle_avg_RD_Q14 )
) {
            break;
        }
    }

    cbk_ptr_Q14 = SKP_Silk_LTP_vq_ptrs_Q14[ *periodicity_index ];
    for( j = 0; j < NB_SUBFR; j++ ) {
        for( k = 0; k < LTP_ORDER; k++ ) {
            B_Q14[ j * LTP_ORDER + k ] = cbk_ptr_Q14[ SKP_MLA( k, cbk_index[ j ],
 LTP_ORDER ) ];
        }
    }
}
```

A.90. src/SKP_Silk_range_coder.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
```

```
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main.h"

/* Range encoder for one symbol */
void SKP_Silk_range_encoder(
    SKP_Silk_range_coder_state      *psRC,              /* I/O  compressor data s
tructure                    */
    const SKP_int                   data,               /* I    uncompressed data
                            */
    const SKP_uint16                prob[]              /* I    cumulative densit
y functions                 */
)
{
    SKP_uint32 low_Q16, high_Q16;
    SKP_uint32 base_tmp, range_Q32;

    /* Copy structure data */
    SKP_uint32 base_Q32  = psRC->base_Q32;
    SKP_uint32 range_Q16 = psRC->range_Q16;
    SKP_int32  bufferIx  = psRC->bufferIx;
    SKP_uint8  *buffer   = psRC->buffer;

    if( psRC->error ) {
        return;
    }

    /* Update interval */
    low_Q16  = prob[ data ];
    high_Q16 = prob[ data + 1 ];
    base_tmp = base_Q32; /* save current base, to test for carry */
    base_Q32 += SKP_MUL_uint( range_Q16, low_Q16 );
    range_Q32 = SKP_MUL_uint( range_Q16, high_Q16 - low_Q16 );

    /* Check for carry */
    if( base_Q32 < base_tmp ) {
        /* Propagate carry in buffer */
        SKP_int bufferIx_tmp = bufferIx;
```

```
        while( ( ++buffer[ --bufferIx_tmp ] ) == 0 );
    }

    /* Check normalization */
    if( range_Q32 & 0xFF000000 ) {
        /* No normalization */
        range_Q16 = SKP_RSHIFT_uint( range_Q32, 16 );
    } else {
        if( range_Q32 & 0xFFFF0000 ) {
            /* Normalization of 8 bits shift */
            range_Q16 = SKP_RSHIFT_uint( range_Q32, 8 );
        } else {
            /* Normalization of 16 bits shift */
            range_Q16 = range_Q32;
            /* Make sure not to write beyond buffer */
            if( bufferIx >= psRC->bufferLength ) {
                psRC->error = RANGE_CODER_WRITE_BEYOND_BUFFER;
                return;
            }
            /* Write one byte to buffer */
            buffer[ bufferIx++ ] = (SKP_uint8)( SKP_RSHIFT_uint( base_Q32, 24 ) )
;
            base_Q32 = SKP_LSHIFT_ovflw( base_Q32, 8 );
        }
        /* Make sure not to write beyond buffer */
        if( bufferIx >= psRC->bufferLength ) {
            psRC->error = RANGE_CODER_WRITE_BEYOND_BUFFER;
            return;
        }
        /* Write one byte to buffer */
        buffer[ bufferIx++ ] = (SKP_uint8)( SKP_RSHIFT_uint( base_Q32, 24 ) );
        base_Q32 = SKP_LSHIFT_ovflw( base_Q32, 8 );
    }

    /* Copy structure data back */
    psRC->base_Q32  = base_Q32;
    psRC->range_Q16 = range_Q16;
    psRC->bufferIx  = bufferIx;
}

/* Range encoder for multiple symbols */
void SKP_Silk_range_encoder_multi(
    SKP_Silk_range_coder_state      *psRC,              /* I/O  compressor data s
tructure                    */
    const SKP_int                   data[],             /* I    uncompressed data
    [nSymbols]              */
    const SKP_uint16 * const        prob[],             /* I    cumulative densit
y functions                 */
    const SKP_int                   nSymbols            /* I    number of data sy
mbols                       */
)
{
    SKP_int k;
```

```
    for( k = 0; k < nSymbols; k++ ) {
        SKP_Silk_range_encoder( psRC, data[ k ], prob[ k ] );
    }
}

/* Range decoder for one symbol */
void SKP_Silk_range_decoder(
    SKP_int                         data[],             /* O    uncompressed data
                        */
    SKP_Silk_range_coder_state      *psRC,              /* I/O  compressor data s
tructure                */
    const SKP_uint16                prob[],             /* I    cumulative densit
y function              */
    SKP_int                         probIx              /* I    initial (middle)
entry of cdf            */
)
{
    SKP_uint32 low_Q16, high_Q16;
    SKP_uint32 base_tmp, range_Q32;

    /* Copy structure data */
    SKP_uint32 base_Q32  = psRC->base_Q32;
    SKP_uint32 range_Q16 = psRC->range_Q16;
    SKP_int32  bufferIx  = psRC->bufferIx;
    SKP_uint8  *buffer   = &psRC->buffer[ 4 ];

    if( psRC->error ) {
        /* Set output to zero */
        *data = 0;
        return;
    }

    high_Q16 = prob[ probIx ];
    base_tmp = SKP_MUL_uint( range_Q16, high_Q16 );
    if( base_tmp > base_Q32 ) {
        while( 1 ) {
            low_Q16 = prob[ --probIx ];
            base_tmp = SKP_MUL_uint( range_Q16, low_Q16 );
            if( base_tmp <= base_Q32 ) {
                break;
            }
            high_Q16 = low_Q16;
            /* Test for out of range */
            if( high_Q16 == 0 ) {
                psRC->error = RANGE_CODER_CDF_OUT_OF_RANGE;
                /* Set output to zero */
                *data = 0;
                return;
            }
        }
    } else {
        while( 1 ) {
```

```
            low_Q16  = high_Q16;
            high_Q16 = prob[ ++probIx ];
            base_tmp = SKP_MUL_uint( range_Q16, high_Q16 );
            if( base_tmp > base_Q32 ) {
                probIx--;
                break;
            }
            /* Test for out of range */
            if( high_Q16 == 0xFFFF ) {
                psRC->error = RANGE_CODER_CDF_OUT_OF_RANGE;
                /* Set output to zero */
                *data = 0;
                return;
            }
        }
    }
    *data = probIx;
    base_Q32 -= SKP_MUL_uint( range_Q16, low_Q16 );
    range_Q32 = SKP_MUL_uint( range_Q16, high_Q16 - low_Q16 );

    /* Check normalization */
    if( range_Q32 & 0xFF000000 ) {
        /* No normalization */
        range_Q16 = SKP_RSHIFT_uint( range_Q32, 16 );
    } else {
        if( range_Q32 & 0xFFFF0000 ) {
            /* Normalization of 8 bits shift */
            range_Q16 = SKP_RSHIFT_uint( range_Q32, 8 );
            /* Check for errors */
            if( SKP_RSHIFT_uint( base_Q32, 24 ) ) {
                psRC->error = RANGE_CODER_NORMALIZATION_FAILED;
                /* Set output to zero */
                *data = 0;
                return;
            }
        } else {
            /* Normalization of 16 bits shift */
            range_Q16 = range_Q32;
            /* Check for errors */
            if( SKP_RSHIFT( base_Q32, 16 ) ) {
                psRC->error = RANGE_CODER_NORMALIZATION_FAILED;
                /* Set output to zero */
                *data = 0;
                return;
            }
            /* Update base */
            base_Q32 = SKP_LSHIFT_uint( base_Q32, 8 );
            /* Make sure not to read beyond buffer */
```

```
            if( bufferIx < psRC->bufferLength ) {
                /* Read one byte from buffer */
                base_Q32 |= (SKP_uint32)buffer[ bufferIx++ ];
            }
        }
        /* Update base */
        base_Q32 = SKP_LSHIFT_uint( base_Q32, 8 );
        /* Make sure not to read beyond buffer */
        if( bufferIx < psRC->bufferLength ) {
            /* Read one byte from buffer */
            base_Q32 |= (SKP_uint32)buffer[ bufferIx++ ];
        }
    }

    /* Check for zero interval length */
    if( range_Q16 == 0 ) {
        psRC->error = RANGE_CODER_ZERO_INTERVAL_WIDTH;
        /* Set output to zero */
        *data = 0;
        return;
    }

    /* Copy structure data back */
    psRC->base_Q32  = base_Q32;
    psRC->range_Q16 = range_Q16;
    psRC->bufferIx  = bufferIx;
}

/* Range decoder for multiple symbols */
void SKP_Silk_range_decoder_multi(
    SKP_int                         data[],             /* O    uncompressed data
                [nSymbols] */
    SKP_Silk_range_coder_state      *psRC,              /* I/O  compressor data s
tructure                    */
    const SKP_uint16 * const        prob[],             /* I    cumulative densit
y functions                 */
    const SKP_int                   probStartIx[],      /* I    initial (middle)
entries of cdfs [nSymbols] */
    const SKP_int                   nSymbols            /* I    number of data sy
mbols                       */
)
{
    SKP_int k;
    for( k = 0; k < nSymbols; k++ ) {
        SKP_Silk_range_decoder( &data[ k ], psRC, prob[ k ], probStartIx[ k ] );
    }
}

/* Initialize range encoder */
void SKP_Silk_range_enc_init(
    SKP_Silk_range_coder_state      *psRC               /* O    compressor data s
tructure                    */
)
{
```

```
    /* Initialize structure */
    psRC->bufferLength = MAX_ARITHM_BYTES;
    psRC->range_Q16    = 0x0000FFFF;
    psRC->bufferIx     = 0;
    psRC->base_Q32     = 0;
    psRC->error        = 0;
}

/* Initialize range decoder */
void SKP_Silk_range_dec_init(
    SKP_Silk_range_coder_state      *psRC,              /* O    compressor data s
tructure                    */
    const SKP_uint8                 buffer[],           /* I    buffer for compre
ssed data [bufferLength]    */
    const SKP_int32                 bufferLength        /* I    buffer length (in
 bytes)                     */
)
{
    /* check input */
    if( bufferLength > MAX_ARITHM_BYTES ) {
        psRC->error = RANGE_CODER_DEC_PAYLOAD_TOO_LONG;
        return;
    }
    /* Initialize structure */
    /* Copy to internal buffer */
    SKP_memcpy( psRC->buffer, buffer, bufferLength * sizeof( SKP_uint8 ) );
    psRC->bufferLength = bufferLength;
    psRC->bufferIx = 0;
    psRC->base_Q32 =
        SKP_LSHIFT_uint( (SKP_uint32)buffer[ 0 ], 24 ) |
        SKP_LSHIFT_uint( (SKP_uint32)buffer[ 1 ], 16 ) |
        SKP_LSHIFT_uint( (SKP_uint32)buffer[ 2 ],  8 ) |
                         (SKP_uint32)buffer[ 3 ];
    psRC->range_Q16 = 0x0000FFFF;
    psRC->error     = 0;
}

/* Determine length of bitstream */
SKP_int SKP_Silk_range_coder_get_length(                /* O    returns number of
 BITS in stream             */
    const SKP_Silk_range_coder_state    *psRC,          /* I    compressed data s
tructure                    */
    SKP_int                             *nBytes         /* O    number of BYTES i
n stream                    */
)
{
    SKP_int nBits;

    /* Number of additional bits (1..9) required to be stored to stream */
    nBits = SKP_LSHIFT( psRC->bufferIx, 3 ) + SKP_Silk_CLZ32( psRC->range_Q16 - 1
 ) - 14;

    *nBytes = SKP_RSHIFT( nBits + 7, 3 );

    /* Return number of bits in bitstream */
```

```
    return nBits;
}

/* Write shortest uniquely decodable stream to buffer, and determine its length *
/
void SKP_Silk_range_enc_wrap_up(
    SKP_Silk_range_coder_state      *psRC               /* I/O  compressed data s
tructure                    */
)
{
    SKP_int bufferIx_tmp, bits_to_store, bits_in_stream, nBytes, mask;
    SKP_uint32 base_Q24;

    /* Lower limit of interval, shifted 8 bits to the right */
    base_Q24 = SKP_RSHIFT_uint( psRC->base_Q32, 8 );

    bits_in_stream = SKP_Silk_range_coder_get_length( psRC, &nBytes );

    /* Number of additional bits (1..9) required to be stored to stream */
    bits_to_store = bits_in_stream - SKP_LSHIFT( psRC->bufferIx, 3 );
    /* Round up to required resolution */
    base_Q24 += SKP_RSHIFT_uint(  0x00800000, bits_to_store - 1 );
    base_Q24 &= SKP_LSHIFT_ovflw( 0xFFFFFFFF, 24 - bits_to_store );

    /* Check for carry */
    if( base_Q24 & 0x01000000 ) {
        /* Propagate carry in buffer */
        bufferIx_tmp = psRC->bufferIx;
        while( ( ++( psRC->buffer[ --bufferIx_tmp ] ) ) == 0 );
    }

    /* Store to stream, making sure not to write beyond buffer */
    if( psRC->bufferIx < psRC->bufferLength ) {
        psRC->buffer[ psRC->bufferIx++ ] = (SKP_uint8)SKP_RSHIFT_uint( base_Q24,
16 );
        if( bits_to_store > 8 ) {
            if( psRC->bufferIx < psRC->bufferLength ) {
                psRC->buffer[ psRC->bufferIx++ ] = (SKP_uint8)SKP_RSHIFT_uint( ba
se_Q24, 8 );
            }
        }
    }

    /* Fill up any remaining bits in the last byte with 1s */
    if( bits_in_stream & 7 ) {
        mask = SKP_RSHIFT( 0xFF, bits_in_stream & 7 );
        if( nBytes - 1 < psRC->bufferLength ) {
            psRC->buffer[ nBytes - 1 ] |= mask;
        }
    }
}
```

```
/* Check that any remaining bits in the last byte are set to 1 */
void SKP_Silk_range_coder_check_after_decoding(
    SKP_Silk_range_coder_state      *psRC               /* I/O  compressed data s
tructure                    */
)
{
    SKP_int bits_in_stream, nBytes, mask;

    bits_in_stream = SKP_Silk_range_coder_get_length( psRC, &nBytes );

    /* Make sure not to read beyond buffer */
    if( nBytes - 1 >= psRC->bufferLength ) {
        psRC->error = RANGE_CODER_DECODER_CHECK_FAILED;
        return;
    }

    /* Test any remaining bits in last byte */
    if( bits_in_stream & 7 ) {
        mask = SKP_RSHIFT( 0xFF, bits_in_stream & 7 );
        if( ( psRC->buffer[ nBytes - 1 ] & mask ) != mask ) {
            psRC->error = RANGE_CODER_DECODER_CHECK_FAILED;
            return;
        }
    }
}
```

A.91. src/SKP_Silk_regularize_correlations_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/* Add noise to matrix diagonal */
void SKP_Silk_regularize_correlations_FIX(
    SKP_int32                       *XX,            /* I/O  Correlation matri
ces                     */
    SKP_int32                       *xx,            /* I/O  Correlation value
s                       */
    SKP_int32                       noise,          /* I    Noise to add
                        */
    SKP_int                         D               /* I    Dimension of XX
                        */
)
{
    SKP_int i;
    for( i = 0; i < D; i++ ) {
        matrix_ptr( &XX[ 0 ], i, i, D ) = SKP_ADD32( matrix_ptr( &XX[ 0 ], i, i,
D ), noise );
    }
    xx[ 0 ] += noise;
}
```

A.92. src/SKP_Silk_resample_1_2.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_Silk_resample_1_2                                              *
 *                                                                      *
 * Downsample by a factor 2                                             *
 *                                                                      *
 * Copyright 2006 (c), Skype Limited                                    *
 * Date: 060221                                                         *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"

/* Coefficients for 2-fold resampling */
static SKP_int16 A20_Resample_1_2[ 3 ] = { 1254, 10102, 22898 };
static SKP_int16 A21_Resample_1_2[ 3 ] = { 4810, 16371, 29374 };


/* Downsample by a factor 2 */
void SKP_Silk_resample_1_2(
    const SKP_int16      *in,            /* I:   16 kHz signal [2*len]   */
```

```
    SKP_int32           *S,             /* I/O: State vector [6]        */
    SKP_int16           *out,           /* O:   8 kHz signal [len]      */
    SKP_int32           *scratch,       /* I:   Scratch memory [4*len]  */
    const SKP_int32     len             /* I:   Number of OUTPUT samples*/
)
{
    SKP_int32    k, idx;

    /* De-interleave allpass inputs, and convert Q15 -> Q25 */
    for( k = 0; k < len; k++ ) {
        idx = SKP_LSHIFT( k, 1 );
        scratch[ k ]       = SKP_LSHIFT( (SKP_int32)in[ idx     ], 10 );
        scratch[ k + len ] = SKP_LSHIFT( (SKP_int32)in[ idx + 1 ], 10 );
    }

    idx = SKP_LSHIFT( len, 1 );

    /* Allpass filters */
    SKP_Silk_allpass_int( scratch,             S,     A21_Resample_1_2[ 0 ], scra
tch + idx,       len );
    SKP_Silk_allpass_int( scratch + idx,       S + 1, A21_Resample_1_2[ 1 ], scra
tch + idx + len, len );
    SKP_Silk_allpass_int( scratch + idx + len, S + 2, A21_Resample_1_2[ 2 ], scra
tch,             len );

    SKP_Silk_allpass_int( scratch + len,       S + 3, A20_Resample_1_2[ 0 ], scra
tch + idx,       len );
    SKP_Silk_allpass_int( scratch + idx,       S + 4, A20_Resample_1_2[ 1 ], scra
tch + idx + len, len );
    SKP_Silk_allpass_int( scratch + idx + len, S + 5, A20_Resample_1_2[ 2 ], scra
tch + len,       len );

    /* Add two allpass outputs */
    for( k = 0; k < len; k++ ) {
        out[ k ] = (SKP_int16)SKP_SAT16( SKP_RSHIFT_ROUND( scratch[ k ] + scratch
[ k + len ], 11 ) );
    }
}
```

A.93. src/SKP_Silk_resample_1_2_coarse.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
```

```
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_resample_1_2_coarse.c                                    *
 *                                                                     *
 * Downsample by a factor 2, coarser                                   *
 *                                                                     *
 * Copyright 2006 (c), Skype Limited                                   *
 * Date: 060221                                                        *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"

/* downsample by a factor 2, coarser */
void SKP_Silk_resample_1_2_coarse(
    const SKP_int16      *in,            /* I:   16 kHz signal [2*len]   */
    SKP_int32            *S,             /* I/O: State vector [4]        */
    SKP_int16            *out,           /* O:   8 kHz signal [len]      */
    SKP_int32            *scratch,       /* I:   Scratch memory [3*len]  */
    const SKP_int32      len             /* I:   Number of OUTPUT samples*/
)
{
    SKP_int32 k, idx;

    /* Coefficients for coarser 2-fold resampling */
    const SKP_int16 A20c[ 2 ] = { 2119, 16663 };
    const SKP_int16 A21c[ 2 ] = { 8050, 26861 };

    /* De-interleave allpass inputs, and convert Q15 -> Q25 */
    for( k = 0; k < len; k++ ) {
        idx = SKP_LSHIFT( k, 1 );
        scratch[ k ]       = SKP_LSHIFT( (SKP_int32)in[ idx     ], 10 );
        scratch[ k + len ] = SKP_LSHIFT( (SKP_int32)in[ idx + 1 ], 10 );
    }
```

```
    idx = SKP_LSHIFT( len, 1 );
    /* Allpass filters */
    SKP_Silk_allpass_int( scratch,       S,     A21c[ 0 ], scratch + idx, len );
    SKP_Silk_allpass_int( scratch + idx, S + 1, A21c[ 1 ], scratch,       len );

    SKP_Silk_allpass_int( scratch + len, S + 2, A20c[ 0 ], scratch + idx, len );
    SKP_Silk_allpass_int( scratch + idx, S + 3, A20c[ 1 ], scratch + len, len );

    /* Add two allpass outputs */
    for( k = 0; k < len; k++ ) {
        out[ k ] = (SKP_int16)SKP_SAT16( SKP_RSHIFT_ROUND( scratch[ k ] + scratch
[ k + len ], 11 ) );
    }
}
```

A.94. src/SKP_Silk_resample_1_2_coarsest.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
```

```
 * SKP_Silk_resample_1_2_coarsest.c                                    *
 *                                                                      *
 * Downsample by a factor 2, coarsest                                   *
 *                                                                      *
 * Copyright 2006 (c), Skype Limited                                    *
 * Date: 060221                                                         *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"


/* Coefficients for coarsest 2-fold resampling */
static SKP_int16 A20cst[ 1 ] = {  3786 };
static SKP_int16 A21cst[ 1 ] = { 17908 };

/* Downsample by a factor 2, coarsest */
void SKP_Silk_resample_1_2_coarsest(
    const SKP_int16     *in,                /* I:   16 kHz signal [2*len]   */
    SKP_int32           *S,                 /* I/O: State vector [2]        */
    SKP_int16           *out,               /* O:   8 kHz signal [len]      */
    SKP_int32           *scratch,           /* I:   Scratch memory [3*len]  */
    const SKP_int32     len                 /* I:   Number of OUTPUT samples*/
)
{
    SKP_int32 k, idx;

    /* De-interleave allpass inputs, and convert Q15 -> Q25 */
    for( k = 0; k < len; k++ ) {
        idx = SKP_LSHIFT( k, 1 );
        scratch[ k ]       = SKP_LSHIFT( (SKP_int32)in[ idx     ], 10 );
        scratch[ k + len ] = SKP_LSHIFT( (SKP_int32)in[ idx + 1 ], 10 );
    }

    idx = SKP_LSHIFT( len, 1 );
    /* Allpass filters */
    SKP_Silk_allpass_int( scratch,       S,     A21cst[ 0 ], scratch + idx, len )
;
    SKP_Silk_allpass_int( scratch + len, S + 1, A20cst[ 0 ], scratch,       len )
;

    /* Add two allpass outputs */
    for( k = 0; k < len; k++ ) {
        out[ k ] = (SKP_int16)SKP_SAT16( SKP_RSHIFT_ROUND( scratch[ k ] + scratch
[ k + idx ], 11 ) );
    }
}
```

A.95. src/SKP_Silk_resample_1_3.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_Silk_resample_1_3.c                                            *
 *                                                                      *
 * Downsamples by a factor 3                                            *
 *                                                                      *
 * Copyright 2008 (c), Skype Limited                                    *
 * Date: 081113                                                         *
 *                                                                      */
#include "SKP_Silk_SigProc_FIX.h"

#define OUT_SUBFR_LEN        80

/* Downsamples by a factor 3 */
void SKP_Silk_resample_1_3(
    SKP_int16           *out,       /* O:   Fs_low signal  [inLen/3]
    */
    SKP_int32           *S,         /* I/O: State vector   [7]
    */
    const SKP_int16     *in,        /* I:   Fs_high signal [inLen]
    */
    const SKP_int32     inLen       /* I:   Input length, must be a multiple of
3   */
)
{
    SKP_int      k, outLen, LSubFrameIn, LSubFrameOut;
```

```
    SKP_int32    out_tmp, limit = 102258000; // (102258000 + 1560) * 21 * 2^(-16)
 = 32767.5
    SKP_int32    scratch0[ 3 * OUT_SUBFR_LEN ];
    SKP_int32    scratch10[ OUT_SUBFR_LEN ], scratch11[ OUT_SUBFR_LEN ], scratch1
2[ OUT_SUBFR_LEN ];
    /* coefficients for 3-fold resampling */
    const SKP_int16 A30[ 2 ] = {  1773, 17818 };
    const SKP_int16 A31[ 2 ] = {  4942, 25677 };
    const SKP_int16 A32[ 2 ] = { 11786, 29304 };

    /* Check that input is multiple of 3 */
    SKP_assert( inLen % 3 == 0 );

    outLen = SKP_DIV32_16( inLen, 3 );
    while( outLen > 0 ) {
        LSubFrameOut = SKP_min_int( OUT_SUBFR_LEN, outLen );
        LSubFrameIn  = SKP_SMULBB( 3, LSubFrameOut );

        /* Low-pass filter, Q15 -> Q25 */
        SKP_Silk_lowpass_short( in, S, scratch0, LSubFrameIn );

        /* De-interleave three allpass inputs */
        for( k = 0; k < LSubFrameOut; k++ ) {
            scratch10[ k ] = scratch0[ 3 * k     ];
            scratch11[ k ] = scratch0[ 3 * k + 1 ];
            scratch12[ k ] = scratch0[ 3 * k + 2 ];
        }

        /* Allpass filters */
        SKP_Silk_allpass_int( scratch10, S + 1, A32[ 0 ], scratch0,  LSubFrameOut
 );
        SKP_Silk_allpass_int( scratch0,  S + 2, A32[ 1 ], scratch10, LSubFrameOut
 );

        SKP_Silk_allpass_int( scratch11, S + 3, A31[ 0 ], scratch0,  LSubFrameOut
 );
        SKP_Silk_allpass_int( scratch0,  S + 4, A31[ 1 ], scratch11, LSubFrameOut
 );

        SKP_Silk_allpass_int( scratch12, S + 5, A30[ 0 ], scratch0,  LSubFrameOut
 );
        SKP_Silk_allpass_int( scratch0,  S + 6, A30[ 1 ], scratch12, LSubFrameOut
 );

        /* Add three allpass outputs */
        for( k = 0; k < LSubFrameOut; k++ ) {
            out_tmp = scratch10[ k ] + scratch11[ k ] + scratch12[ k ];
            if( out_tmp - limit > 0 ) {
                out[ k ] = SKP_int16_MAX;
            } else if( out_tmp + limit < 0 ) {
                out[ k ] = SKP_int16_MIN;
            } else {
                out[ k ] = (SKP_int16) SKP_SMULWB( out_tmp + 1560, 21 );
            }
        }
```

```
        in     += LSubFrameIn;
        out    += LSubFrameOut;
        outLen -= LSubFrameOut;
    }
}
```

A.96. src/SKP_Silk_resample_2_1_coarse.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_resample_2_1_coarse.c                                    *
 *                                                                     *
 * Upsample by a factor 2, coarser                                     *
 *                                                                     *
 * Copyright 2006 (c), Skype Limited                                   *
 * Date: 060221                                                        *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"

/* Upsample by a factor 2, coarser */
```

```
void SKP_Silk_resample_2_1_coarse(
    const SKP_int16      *in,            /* I:   8 kHz signal [len]      */
    SKP_int32            *S,             /* I/O: State vector [4]        */
    SKP_int16            *out,           /* O:   16 kHz signal [2*len]   */
    SKP_int32            *scratch,       /* I:   Scratch memory [3*len]  */
    const SKP_int32      len             /* I:   Number of INPUT samples */
)
{
    SKP_int32 k, idx;

    /* Coefficients for coarser 2-fold resampling */
    const SKP_int16 A20c[ 2 ] = { 2119, 16663 };
    const SKP_int16 A21c[ 2 ] = { 8050, 26861 };

    /* Convert Q15 -> Q25 */
    for( k = 0; k < len; k++ ) {
        scratch[ k ] = SKP_LSHIFT( (SKP_int32)in[ k ], 10 );
    }

    idx = SKP_LSHIFT( len, 1 );

    /* Allpass filters */
    SKP_Silk_allpass_int( scratch,       S,     A20c[ 0 ], scratch + idx, len );
    SKP_Silk_allpass_int( scratch + idx, S + 1, A20c[ 1 ], scratch + len, len );

    SKP_Silk_allpass_int( scratch,       S + 2, A21c[ 0 ], scratch + idx, len );
    SKP_Silk_allpass_int( scratch + idx, S + 3, A21c[ 1 ], scratch,       len );

    /* Interleave two allpass outputs */
    for( k = 0; k < len; k++ ) {
        idx = SKP_LSHIFT( k, 1 );
        out[ idx     ] = (SKP_int16)SKP_SAT16( SKP_RSHIFT_ROUND( scratch[ k + len
 ], 10 ) );
        out[ idx + 1 ] = (SKP_int16)SKP_SAT16( SKP_RSHIFT_ROUND( scratch[ k ],
    10 ) );
    }
}
```

A.97. src/SKP_Silk_resample_2_3.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
```

```
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * File Name:    SKP_Silk_resample_2_3.c                             *
 *                                                                     *
 * Resamples by a factor 2/3                                           *
 *                                                                     *
 * Copyright 2008 (c), Skype Limited                                   *
 * All rights reserved.                                                *
 *                                                                     *
 * Date: 081113                                                        *
 *                                                                     */

#include "SKP_Silk_SigProc_FIX.h"

#define OUT_SUBFR_LEN        80

/* Resamples by a factor 2/3 */
void SKP_Silk_resample_2_3(
    SKP_int16            *out,       /* O:   Fs_low signal    [inLen * 2/3]
     */
    SKP_int32            *S,         /* I/O: State vector     [7+4]
     */
    const SKP_int16      *in,        /* I:   Fs_high signal   [inLen]
     */
    const SKP_int        inLen       /* I:   Input length, must be a multiple of
3    */
)
{
    SKP_int      outLen, LSubFrameIn, LSubFrameOut;
    SKP_int16    outH[      3 * OUT_SUBFR_LEN ];
    SKP_int32    scratch[ ( 9 * OUT_SUBFR_LEN ) / 2 ];

    /* Check that input length is multiple of 3 */
    SKP_assert( inLen % 3 == 0 );
```

```
    outLen = SKP_DIV32_16( SKP_LSHIFT( inLen, 1 ), 3 );
    while( outLen > 0 ) {
        LSubFrameOut = SKP_min_int( OUT_SUBFR_LEN, outLen );
        LSubFrameIn  = SKP_SMULWB( 98304, LSubFrameOut ); /* 98304_Q16 = 3/2_Q0 *
/

        /* Upsample by a factor 2 */
        /* Scratch size needs to be: 3 * LSubFrameIn * sizeof( SKP_int32 ) */
        SKP_Silk_resample_2_1_coarse( in, &S[ 0 ], outH, scratch, LSubFrameIn );

        /* Downsample by a factor 3 */
        SKP_Silk_resample_1_3( out, &S[ 4 ], outH, SKP_LSHIFT( LSubFrameIn, 1 ) )
;

        in     += LSubFrameIn;
        out    += LSubFrameOut;
        outLen -= LSubFrameOut;
    }
}
```

A.98. src/SKP_Silk_resample_2_3_coarse.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/
```

```
/*                                                                      *
 * File Name:   SKP_Silk_resample_2_3_coarse.c                          *
 *                                                                      *
 * Description: Linear phase FIR polyphase implementation of resampling *
 *                                                                      *
 * Copyright 2009 (c), Skype Limited                                    *
 * All rights reserved.                                                 *
 *                                                                      *
 * Date: 090423                                                         *
 *                                                                      */

#include "SKP_Silk_SigProc_FIX.h"
#include "SKP_Silk_resample_rom.h"

/* Resamples input data with a factor 2/3 */
void SKP_Silk_resample_2_3_coarse(
    SKP_int16           *out,           /* O:   Output signal
                                           */
    SKP_int16           *S,             /* I/O: Resampler state [ SigProc_Resampl
e_2_3_coarse_NUM_FIR_COEFS - 1 ]           */
    const SKP_int16     *in,            /* I:   Input signal
                                           */
    const SKP_int       frameLenIn,     /* I:   Number of input samples
                                           */
    SKP_int16           *scratch        /* I:   Scratch memory [ frameLenIn + Sig
Proc_Resample_2_3_coarse_NUM_FIR_COEFS - 1 ]   */
)
{
    SKP_int32 n, ind, interpol_ind, tmp, index_Q16;
    SKP_int16 *in_ptr;
    SKP_int   frameLenOut;
    const SKP_int16 *interpol_ptr;

    /* Copy buffered samples to start of scratch */
    SKP_memcpy( scratch, S, ( SigProc_Resample_2_3_coarse_NUM_FIR_COEFS - 1 ) * s
izeof( SKP_int16 ) );

    /* Then append by the input signal */
    SKP_memcpy( &scratch[ SigProc_Resample_2_3_coarse_NUM_FIR_COEFS - 1 ], in, fr
ameLenIn * sizeof( SKP_int16 ) );

    frameLenOut = SKP_DIV32_16( SKP_MUL( 2, frameLenIn ), 3 );
    index_Q16 = 0;

    SKP_assert( frameLenIn == ( ( frameLenOut * 3 ) / 2 ) );

    /* Interpolate */
    for( n = frameLenOut; n > 0; n-- ) {

        /* Integer part */
        ind = SKP_RSHIFT( index_Q16, 16 );

        /* Pointer to buffered input */
        in_ptr = scratch + ind;
```

```
        /* Fractional part */
        interpol_ind = ( SKP_SMULWB( index_Q16, SigProc_Resample_2_3_coarse_NUM_I
NTERPOLATORS ) &
                         ( SigProc_Resample_2_3_coarse_NUM_INTERPOLATORS - 1 ) );

        /* Pointer to FIR taps */
        interpol_ptr = SigProc_Resample_2_3_coarse_INTERPOL[ interpol_ind ];

        /* Interpolate */
        /* Hardcoded for 32 FIR taps */
        SKP_assert( SigProc_Resample_2_3_coarse_NUM_FIR_COEFS == 32 );
        tmp = (SKP_int32)interpol_ptr[  0 ] * in_ptr[  0 ] + (SKP_int32)interpol_
ptr[  1 ] * in_ptr[  1 ] +
              (SKP_int32)interpol_ptr[  2 ] * in_ptr[  2 ] + (SKP_int32)interpol_
ptr[  3 ] * in_ptr[  3 ] +
              (SKP_int32)interpol_ptr[  4 ] * in_ptr[  4 ] + (SKP_int32)interpol_
ptr[  5 ] * in_ptr[  5 ] +
              (SKP_int32)interpol_ptr[  6 ] * in_ptr[  6 ] + (SKP_int32)interpol_
ptr[  7 ] * in_ptr[  7 ] +
              (SKP_int32)interpol_ptr[  8 ] * in_ptr[  8 ] + (SKP_int32)interpol_
ptr[  9 ] * in_ptr[  9 ] +
              (SKP_int32)interpol_ptr[ 10 ] * in_ptr[ 10 ] + (SKP_int32)interpol_
ptr[ 11 ] * in_ptr[ 11 ] +
              (SKP_int32)interpol_ptr[ 12 ] * in_ptr[ 12 ] + (SKP_int32)interpol_
ptr[ 13 ] * in_ptr[ 13 ] +
              (SKP_int32)interpol_ptr[ 14 ] * in_ptr[ 14 ] + (SKP_int32)interpol_
ptr[ 15 ] * in_ptr[ 15 ] +
              (SKP_int32)interpol_ptr[ 16 ] * in_ptr[ 16 ] + (SKP_int32)interpol_
ptr[ 17 ] * in_ptr[ 17 ] +
              (SKP_int32)interpol_ptr[ 18 ] * in_ptr[ 18 ] + (SKP_int32)interpol_
ptr[ 19 ] * in_ptr[ 19 ] +
              (SKP_int32)interpol_ptr[ 20 ] * in_ptr[ 20 ] + (SKP_int32)interpol_
ptr[ 21 ] * in_ptr[ 21 ] +
              (SKP_int32)interpol_ptr[ 22 ] * in_ptr[ 22 ] + (SKP_int32)interpol_
ptr[ 23 ] * in_ptr[ 23 ] +
              (SKP_int32)interpol_ptr[ 24 ] * in_ptr[ 24 ] + (SKP_int32)interpol_
ptr[ 25 ] * in_ptr[ 25 ] +
              (SKP_int32)interpol_ptr[ 26 ] * in_ptr[ 26 ] + (SKP_int32)interpol_
ptr[ 27 ] * in_ptr[ 27 ] +
              (SKP_int32)interpol_ptr[ 28 ] * in_ptr[ 28 ] + (SKP_int32)interpol_
ptr[ 29 ] * in_ptr[ 29 ];

        /* Round, saturate and store to output array */
        *out++ = (SKP_int16)SKP_SAT16( SKP_RSHIFT_ROUND( tmp, 15 ) );

        /* Update index */
        index_Q16 += ( ( 1 << 16 ) + ( 1 << 15 ) ); // (3/2)_Q0;
    }

    /* Move last part of input signal to the sample buffer to prepare for the nex
t call */
    SKP_memcpy( S, &in[ frameLenIn - ( SigProc_Resample_2_3_coarse_NUM_FIR_COEFS
- 1 ) ],
                ( SigProc_Resample_2_3_coarse_NUM_FIR_COEFS - 1 ) * sizeof( SKP_i
nt16 ) );
}
```

A.99. src/SKP_Silk_resample_2_3_coarsest.c

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

```
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * File Name:   SKP_Silk_resample_2_3_coarsest.c                       *
 *                                                                     *
 * Description: Linear phase FIR polyphase implementation of resampling *
 *                                                                     *
 * Copyright 2009 (c), Skype Limited                                   *
 * All rights reserved.                                                *
 *                                                                     *
 * Date: 090423                                                        *
 *                                                                     */

#include "SKP_Silk_SigProc_FIX.h"
#include "SKP_Silk_resample_rom.h"

/* Resamples input data with a factor 2/3 */
void SKP_Silk_resample_2_3_coarsest(
    SKP_int16           *out,           /* O:   Output signal
                                           */
    SKP_int16           *S,             /* I/O: Resampler state [ SigProc_Resampl
e_2_3_coarsest_NUM_FIR_COEFS - 1 ]          */
    const SKP_int16     *in,            /* I:   Input signal
                                           */
    const SKP_int       frameLenIn,     /* I:   Number of input samples
                                           */
    SKP_int16           *scratch        /* I:   Scratch memory [ frameLenIn + Sig
Proc_Resample_2_3_coarsest_NUM_FIR_COEFS - 1 ] */
)
{
    SKP_int32 n, ind, interpol_ind, tmp, index_Q16;
    SKP_int16 *in_ptr;
    SKP_int   frameLenOut;
    const SKP_int16 *interpol_ptr;
```

```
    /* Copy buffered samples to start of scratch */
    SKP_memcpy( scratch, S, ( SigProc_Resample_2_3_coarsest_NUM_FIR_COEFS - 1 ) *
 sizeof( SKP_int16 ) );

    /* Then append by the input signal */
    SKP_memcpy( &scratch[ SigProc_Resample_2_3_coarsest_NUM_FIR_COEFS - 1 ], in,
frameLenIn * sizeof( SKP_int16 ) );

    frameLenOut = SKP_SMULWB( SKP_LSHIFT( (SKP_int32)frameLenIn, 1 ), 21846 ); //
 21846_Q15 = (2/3)_Q0 rounded _up_
    index_Q16 = 0;

    SKP_assert( frameLenIn == ( ( frameLenOut * 3 ) / 2 ) );

    /* Interpolate */
    for( n = frameLenOut; n > 0; n-- ) {

        /* Integer part */
        ind = SKP_RSHIFT( index_Q16, 16 );

        /* Pointer to buffered input */
        in_ptr = scratch + ind;

        /* Fractional part */
        interpol_ind = ( SKP_SMULWB( index_Q16, SigProc_Resample_2_3_coarsest_NUM
_INTERPOLATORS ) &
                         ( SigProc_Resample_2_3_coarsest_NUM_INTERPOLATORS - 1 ) );

        /* Pointer to FIR taps */
        interpol_ptr = SigProc_Resample_2_3_coarsest_INTERPOL[ interpol_ind ];

        /* Interpolate: Hardcoded for 10 FIR taps */
        SKP_assert( SigProc_Resample_2_3_coarsest_NUM_FIR_COEFS == 10 );
        tmp = SKP_SMULBB(      interpol_ptr[ 0 ], in_ptr[ 0 ] );
        tmp = SKP_SMLABB( tmp, interpol_ptr[ 1 ], in_ptr[ 1 ] );
        tmp = SKP_SMLABB( tmp, interpol_ptr[ 2 ], in_ptr[ 2 ] );
        tmp = SKP_SMLABB( tmp, interpol_ptr[ 3 ], in_ptr[ 3 ] );
        tmp = SKP_SMLABB( tmp, interpol_ptr[ 4 ], in_ptr[ 4 ] );
        tmp = SKP_SMLABB( tmp, interpol_ptr[ 5 ], in_ptr[ 5 ] );
        tmp = SKP_SMLABB( tmp, interpol_ptr[ 6 ], in_ptr[ 6 ] );
        tmp = SKP_SMLABB( tmp, interpol_ptr[ 7 ], in_ptr[ 7 ] );
        tmp = SKP_SMLABB( tmp, interpol_ptr[ 8 ], in_ptr[ 8 ] );
        tmp = SKP_SMLABB( tmp, interpol_ptr[ 9 ], in_ptr[ 9 ] );
        /* Round, saturate and store to output array */
        *out++ = (SKP_int16)SKP_SAT16( SKP_RSHIFT_ROUND( tmp, 15 ) );

        /* Update index */
        index_Q16 += ( ( 1 << 16 ) + ( 1 << 15 ) ); // (3/2)_Q0;
    }

    /* Move last part of input signal to the sample buffer to prepare for the nex
t call */
    SKP_memcpy( S, &in[ frameLenIn - ( SigProc_Resample_2_3_coarsest_NUM_FIR_COEF
S - 1 ) ],
```

```
                ( SigProc_Resample_2_3_coarsest_NUM_FIR_COEFS - 1 ) * sizeof( SKP
_int16 ) );
}
```

A.100. src/SKP_Silk_resample_2_3_rom.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * File Name:    SKP_Silk_resample_2_3_rom.c                          *
 *                                                                      *
 * Description: Filter coefficients for FIR polyphase resampling       *
 *                                                                      *
 * Copyright 2009 (c), Skype Limited                                    *
 * All rights reserved.                                                 *
 *                                                                      *
 * Date: 090424                                                         *
 *                                                                      */

#include "SKP_Silk_resample_rom.h"

const SKP_int16 SigProc_Resample_2_3_coarse_INTERPOL[ SigProc_Resample_2_3_coarse
_NUM_INTERPOLATORS ][ SigProc_Resample_2_3_coarse_NUM_FIR_COEFS ] = {
```

```
    {    0,   -74,   109,     0,  -234,   329,     0,  -610,   813,     0, -1437,
  1954,     0, -4358,  8953, 21845,  8953, -4358,     0,  1954, -1437,     0,   8
13,  -610,     0,   329,  -234,     0,   109,   -74,     0,    45 },
    {   48,   -62,     0,   133,  -195,     0,   386,  -526,     0,   936, -1243,
     0,  2311, -3417,     0, 18026, 18026,     0, -3417,  2311,     0, -1243,   9
36,     0,  -526,   386,     0,  -195,   133,     0,   -62,    48 },
};

const SKP_int16 SigProc_Resample_2_3_coarsest_INTERPOL[ SigProc_Resample_2_3_coar
sest_NUM_INTERPOLATORS ][ SigProc_Resample_2_3_coarsest_NUM_FIR_COEFS ] = {
    {  379,     0, -3081,  8239, 21845,  8239, -3081,     0,   379,  -145 },
    {    0,   696, -1951,     0, 17659, 17659,     0, -1951,   696,     0 },
};
```

A.101. src/SKP_Silk_resample_3_1.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * SKP_Silk_resample_3_1.c                                            *
 *                                                                      *
 * Upsamples by a factor 3                                              *
 *                                                                      *
 * Copyright 2008 (c), Skype Limited                                    *
 * Date: 081113                                                         *
```

```
 *                                                                   */
#include "SKP_Silk_SigProc_FIX.h"

#define IN_SUBFR_LEN_RESAMPLE_3_1       40

/* Resamples by a factor 3/1 */
void SKP_Silk_resample_3_1(
    SKP_int16           *out,       /* O:   Fs_high signal [inLen*3]        */
    SKP_int32           *S,         /* I/O: State vector   [7]              */
    const SKP_int16     *in,        /* I:   Fs_low signal  [inLen]          */
    const SKP_int32     inLen       /* I:   Input length                    */
)
{
    SKP_int     k, LSubFrameIn, LSubFrameOut;
    SKP_int32   out_tmp, idx, inLenTmp = inLen;
    SKP_int32   scratch00[    IN_SUBFR_LEN_RESAMPLE_3_1 ];
    SKP_int32   scratch0[ 3 * IN_SUBFR_LEN_RESAMPLE_3_1 ];
    SKP_int32   scratch1[ 3 * IN_SUBFR_LEN_RESAMPLE_3_1 ];

    /* Coefficients for 3-fold resampling */
    const SKP_int16 A30[ 2 ] = {  1773, 17818 };
    const SKP_int16 A31[ 2 ] = {  4942, 25677 };
    const SKP_int16 A32[ 2 ] = { 11786, 29304 };

    while( inLenTmp > 0 ) {
        LSubFrameIn  = SKP_min_int( IN_SUBFR_LEN_RESAMPLE_3_1, inLenTmp );
        LSubFrameOut = SKP_SMULBB( 3, LSubFrameIn );

        /* Convert Q15 -> Q25 */
        for( k = 0; k < LSubFrameIn; k++ ) {
            scratch00[k] = SKP_LSHIFT( (SKP_int32)in[ k ], 10 );
        }

        /* Allpass filtering */
        /* Scratch size: 2 * 3* LSubFrame * sizeof(SKP_int32) */
        SKP_Silk_allpass_int( scratch00, S + 1, A30[ 0 ], scratch1, LSubFrameIn )
;
        SKP_Silk_allpass_int( scratch1,  S + 2, A30[ 1 ], scratch0, LSubFrameIn )
;

        SKP_Silk_allpass_int( scratch00, S + 3, A31[ 0 ], scratch1, LSubFrameIn )
;
        SKP_Silk_allpass_int( scratch1,  S + 4, A31[ 1 ], scratch0 +     IN_SUBFR
_LEN_RESAMPLE_3_1, LSubFrameIn );

        SKP_Silk_allpass_int( scratch00, S + 5, A32[ 0 ], scratch1, LSubFrameIn )
;
        SKP_Silk_allpass_int( scratch1,  S + 6, A32[ 1 ], scratch0 + 2 * IN_SUBFR
_LEN_RESAMPLE_3_1, LSubFrameIn );

        /* Interleave three allpass outputs */
        for( k = 0; k < LSubFrameIn; k++ ) {
            idx = SKP_SMULBB( 3, k );
            scratch1[ idx     ] = scratch0[ k ];
```

```
            scratch1[ idx + 1 ] = scratch0[ k +     IN_SUBFR_LEN_RESAMPLE_3_1 ];
            scratch1[ idx + 2 ] = scratch0[ k + 2 * IN_SUBFR_LEN_RESAMPLE_3_1 ];
        }

        /* Low-pass filtering */
        SKP_Silk_lowpass_int( scratch1, S, scratch0, LSubFrameOut );

        /* Saturate and convert to SKP_int16 */
        for( k = 0; k < LSubFrameOut; k++ ) {
            out_tmp  = scratch0[ k ];
            out[ k ] = (SKP_int16) SKP_SAT16( SKP_RSHIFT_ROUND( out_tmp, 10 ) );
        }

        in       += LSubFrameIn;
        inLenTmp -= LSubFrameIn;
        out      += LSubFrameOut;
    }
}
```

A.102. src/SKP_Silk_resample_3_2.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

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

/*                                                                     *
 * File Name:   SKP_Silk_resample_3_2.c                               *
 *                                                                     *
 * Resamples by a factor 3/2                                           *
 *                                                                     *
 * Copyright 2008 (c), Skype Limited                                   *
 * All rights reserved.                                                *
 *                                                                     *
 * Date: 081113                                                        *
 *                                                                     */

#include "SKP_Silk_SigProc_FIX.h"

#define IN_SUBFR_LEN_RESAMPLE_3_2       80

/* Resamples by a factor 3/2 */
void SKP_Silk_resample_3_2(
    SKP_int16           *out,       /*   O: Fs_high signal  [inLen*3/2]
   */
    SKP_int32           *S,         /* I/O: State vector    [7+4]
   */
    const SKP_int16     *in,        /* I:   Fs_low signal   [inLen]
   */
    SKP_int             inLen       /* I:   Input length, must be a multiple of 2
   */
)
{
    SKP_int     LSubFrameIn, LSubFrameOut;
    SKP_int16   outH[      3 * IN_SUBFR_LEN_RESAMPLE_3_2 ];
    SKP_int32   scratch[ ( 9 * IN_SUBFR_LEN_RESAMPLE_3_2 ) / 2 ];

    /* Check that input is multiple of 2 */
    SKP_assert( inLen % 2 == 0 );

    while( inLen > 0 ) {
        LSubFrameIn  = SKP_min_int( IN_SUBFR_LEN_RESAMPLE_3_2, inLen );
        LSubFrameOut = SKP_SMULWB( 98304, LSubFrameIn );

        /* Upsample by a factor 3 */
        SKP_Silk_resample_3_1( outH, &S[ 0 ], in, LSubFrameIn );

        /* Downsample by a factor 2 */
        /* Scratch size needs to be: 3 * LSubFrameOut * sizeof( SKP_int32 ) */
        SKP_Silk_resample_1_2_coarse( outH, &S[ 7 ], out, scratch, LSubFrameOut )
;

        in    += LSubFrameIn;
        out   += LSubFrameOut;
        inLen -= LSubFrameIn;
    }
}
```

A.103. src/SKP_Silk_resample_3_2_rom.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * File Name:   SKP_Silk_resample_3_2_rom.c                           *
 *                                                                      *
 * Description: Filter coefficients for FIR polyphase resampling        *
 *                                                                      *
 * Copyright 2009 (c), Skype Limited                                    *
 * All rights reserved.                                                 *
 *                                                                      *
 * Date: 090424                                                         *
 *                                                                      */

#include "SKP_Silk_resample_rom.h"

const SKP_int16 SigProc_Resample_3_2_coarse_INTERPOL[ SigProc_Resample_3_2_coarse
_NUM_INTERPOLATORS ][ SigProc_Resample_3_2_coarse_NUM_FIR_COEFS ] = {
    {    0,     0,     0, 32768,     0,     0,     0,     0 },
    { -384,  1630, -5217, 26674, 12714, -3572,  1050,  -236 },
    { -236,  1050, -3572, 12714, 26674, -5217,  1630,  -384 },
};
```

A.104. src/SKP_Silk_resample_3_4.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                      *
 * File Name:    SKP_Silk_resample_3_4.c                                *
 *                                                                      *
 * Resamples by a factor 3/4                                            *
 *                                                                      *
 * Copyright 2009 (c), Skype Limited                                    *
 * All rights reserved.                                                 *
 *                                                                      *
 * Date: 090408                                                         *
 *                                                                      */

#include "SKP_Silk_SigProc_FIX.h"

#define IN_SUBFR_LEN_RESAMPLE_3_4        80

/* Resamples by a factor 3/4 */
void SKP_Silk_resample_3_4(
    SKP_int16           *out,       /* O:   Fs_high signal  [inLen*3/4]
     */
```

```
    SKP_int32           *S,         /* I/O: State vector    [7+2+6]
     */
    const SKP_int16     *in,        /* I:   Fs_low signal   [inLen]
     */
    SKP_int             inLen       /* I:   Input length, must be a multiple of
4    */
)
{
    SKP_int      LSubFrameIn, LSubFrameOut;
    SKP_int16    outH[      3 * IN_SUBFR_LEN_RESAMPLE_3_4 ];
    SKP_int16    outL[    ( 3 * IN_SUBFR_LEN_RESAMPLE_3_4 ) / 2 ];
    SKP_int32    scratch[ ( 9 * IN_SUBFR_LEN_RESAMPLE_3_4 ) / 2 ];

    /* Check that input is multiple of 4 */
    SKP_assert( inLen % 4 == 0 );

    while( inLen > 0 ) {
        LSubFrameIn  = SKP_min_int( IN_SUBFR_LEN_RESAMPLE_3_4, inLen );
        LSubFrameOut = SKP_SMULWB( 49152, LSubFrameIn );

        /* Upsample by a factor 3 */
        SKP_Silk_resample_3_1( outH, &S[ 0 ], in, LSubFrameIn );

        /* Downsample by a factor 2 twice */
        /* Scratch size needs to be: 3 * 2 * LSubFrameOut * sizeof( SKP_int32 ) *
/
        /* I: state vector [2], scratch memory [3*len] */
        SKP_Silk_resample_1_2_coarsest( outH, &S[ 7 ], outL, scratch, SKP_LSHIFT(
 LSubFrameOut, 1 ) );

        /* Scratch size needs to be: 3 * LSubFrameOut * sizeof( SKP_int32 ) */
        /* I: state vector [6], scratch memory [3*len]    */
        SKP_Silk_resample_1_2_coarse( outL, &S[ 9 ], out, scratch, LSubFrameOut )
;

        in    += LSubFrameIn;
        out   += LSubFrameOut;
        inLen -= LSubFrameIn;
    }
}
```

A.105. src/SKP_Silk_resample_4_3.c

```
/**********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
```

```
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * File Name:    SKP_Silk_resample_4_3.c                               *
 *                                                                     *
 * Resamples by a factor 4/3                                           *
 *                                                                     *
 * Copyright 2009 (c), Skype Limited                                   *
 * All rights reserved.                                                *
 *                                                                     *
 * Date: 090407                                                        *
 *                                                                     */

#include "SKP_Silk_SigProc_FIX.h"

#define OUT_SUBFR_LEN        80

/* Resamples by a factor 4/3 */
void SKP_Silk_resample_4_3(
    SKP_int16            *out,       /* O:   Fs_low signal    [inLen * 4/3]
     */
    SKP_int32            *S,         /* I/O: State vector    [7+4+4]
     */
    const SKP_int16      *in,        /* I:   Fs_high signal    [inLen]
     */
    const SKP_int        inLen       /* I:   input length, must be a multiple of
3    */
)
{
    SKP_int      outLen, LSubFrameIn, LSubFrameOut;
    SKP_int16    outH[    3 * OUT_SUBFR_LEN ];
    SKP_int16    outHH[   6 * OUT_SUBFR_LEN ];
    SKP_int32    scratch[ 9 * OUT_SUBFR_LEN / 2 ];

    /* Check that input is multiple of 3 */
    SKP_assert( inLen % 3 == 0 );
```

```
    outLen = SKP_DIV32_16( SKP_LSHIFT( inLen, 2 ), 3 );
    while( outLen > 0 ) {
        LSubFrameOut = SKP_min_int( OUT_SUBFR_LEN, outLen );
        LSubFrameIn  = SKP_SMULWB( 49152, LSubFrameOut );

        /* Upsample two times by a factor 2 */
        /* Scratch size needs to be: 3 * LSubFrameIn * sizeof( SKP_int32 ) */
        SKP_Silk_resample_2_1_coarse( in,   &S[ 0 ], outH,  scratch,
LSubFrameIn      );
        /* Scratch size needs to be: 6 * LSubFrameIn * sizeof( SKP_int32 ) */
        SKP_Silk_resample_2_1_coarse( outH, &S[ 4 ], outHH, scratch, SKP_LSHIFT(
LSubFrameIn, 1 ) );

        /* Downsample by a factor 3 */
        SKP_Silk_resample_1_3( out, &S[ 8 ], outHH, SKP_LSHIFT( LSubFrameIn, 2 )
);

        in     += LSubFrameIn;
        out    += LSubFrameOut;
        outLen -= LSubFrameOut;
    }
}
```

A.106. src/SKP_Silk_resample_rom.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
```

```
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * File Name:   SKP_Silk_resample_rom.h                               *
 *                                                                     *
 * Description: Header file for FIR resampling of                      *
 *              32 and 44 kHz input                                    *
 *                                                                     *
 * Copyright 2007 (c), Skype Limited                                   *
 * All rights reserved.                                                *
 *                                                                     *
 * Date: 070807                                                        *
 *                                                                     */

#ifndef _SKP_SILK_FIX_RESAMPLE_ROM_H_
#define _SKP_SILK_FIX_RESAMPLE_ROM_H_

#include "SKP_Silk_typedef.h"

#ifdef  __cplusplus
extern "C"
{
#endif

#define SigProc_Resample_bw_1_4_NUM_INTERPOLATORS_LOG2              7
#define SigProc_Resample_bw_1_4_NUM_INTERPOLATORS                   (1 << SigProc
_Resample_bw_1_4_NUM_INTERPOLATORS_LOG2)
#define SigProc_Resample_bw_1_4_NUM_FIR_COEFS                       6

extern const SKP_int16 SigProc_Resample_bw_1_4_INTERPOL[SigProc_Resample_bw_1_4_N
UM_INTERPOLATORS][SigProc_Resample_bw_1_4_NUM_FIR_COEFS];


#define SigProc_Resample_bw_80_441_NUM_INTERPOLATORS_LOG2           6
#define SigProc_Resample_bw_80_441_NUM_INTERPOLATORS                (1 << SigProc
_Resample_bw_80_441_NUM_INTERPOLATORS_LOG2)
#define SigProc_Resample_bw_80_441_NUM_FIR_COEFS                    4

extern const SKP_int16 SigProc_Resample_bw_80_441_INTERPOL[SigProc_Resample_bw_80
_441_NUM_INTERPOLATORS][SigProc_Resample_bw_80_441_NUM_FIR_COEFS];

#define SigProc_Resample_2_3_coarse_NUM_INTERPOLATORS               2
#define SigProc_Resample_2_3_coarse_NUM_FIR_COEFS                  32

extern const SKP_int16 SigProc_Resample_2_3_coarse_INTERPOL[SigProc_Resample_2_3_
coarse_NUM_INTERPOLATORS][SigProc_Resample_2_3_coarse_NUM_FIR_COEFS];

#define SigProc_Resample_2_3_coarsest_NUM_INTERPOLATORS             2
#define SigProc_Resample_2_3_coarsest_NUM_FIR_COEFS                10

extern const SKP_int16 SigProc_Resample_2_3_coarsest_INTERPOL[SigProc_Resample_2_
3_coarsest_NUM_INTERPOLATORS][SigProc_Resample_2_3_coarsest_NUM_FIR_COEFS];
```

```
#define SigProc_Resample_3_2_coarse_NUM_INTERPOLATORS              3
#define SigProc_Resample_3_2_coarse_NUM_FIR_COEFS                  8

extern const SKP_int16 SigProc_Resample_3_2_coarse_INTERPOL[SigProc_Resample_3_2_
coarse_NUM_INTERPOLATORS][SigProc_Resample_3_2_coarse_NUM_FIR_COEFS];

#define SigProc_Resample_147_40_NUM_INTERPOLATORS                147
#define SigProc_Resample_147_40_NUM_FIR_COEFS                     20

extern const SKP_int16 SigProc_Resample_147_40_INTERPOL[SigProc_Resample_147_40_N
UM_INTERPOLATORS][SigProc_Resample_147_40_NUM_FIR_COEFS];

#define SigProc_Resample_147_40_alt_NUM_INTERPOLATORS            147
#define SigProc_Resample_147_40_alt_NUM_FIR_COEFS                 10

extern const SKP_int16 SigProc_Resample_147_40_alt_INTERPOL[SigProc_Resample_147_
40_alt_NUM_INTERPOLATORS][SigProc_Resample_147_40_alt_NUM_FIR_COEFS];

#define SigProc_Resample_147_40_coarse_NUM_INTERPOLATORS         147
#define SigProc_Resample_147_40_coarse_NUM_FIR_COEFS              16

extern const SKP_int16 SigProc_Resample_147_40_coarse_INTERPOL[SigProc_Resample_1
47_40_coarse_NUM_INTERPOLATORS][SigProc_Resample_147_40_coarse_NUM_FIR_COEFS];

#define SigProc_Resample_40_147_NUM_INTERPOLATORS                 40
#define SigProc_Resample_40_147_NUM_FIR_COEFS                     60

extern const SKP_int16 SigProc_Resample_40_147_INTERPOL[SigProc_Resample_40_147_N
UM_INTERPOLATORS][SigProc_Resample_40_147_NUM_FIR_COEFS];

#define SigProc_Resample_40_147_coarse_NUM_INTERPOLATORS          40
#define SigProc_Resample_40_147_coarse_NUM_FIR_COEFS              30

extern const SKP_int16 SigProc_Resample_40_147_coarse_INTERPOL[SigProc_Resample_4
0_147_coarse_NUM_INTERPOLATORS][SigProc_Resample_40_147_coarse_NUM_FIR_COEFS];

#ifdef  __cplusplus
}
#endif

#endif // _SKP_SILK_FIX_RESAMPLE_ROM_H_
```

A.107. src/SKP_Silk_residual_energy16_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
```

```
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/* Residual energy: nrg = wxx - 2 * wXx * c + c' * wXX * c */
SKP_int32 SKP_Silk_residual_energy16_covar_FIX(
    const SKP_int16                 *c,                 /* I    Prediction vector
                        */
    const SKP_int32                 *wXX,               /* I    Correlation matri
x                       */
    const SKP_int32                 *wXx,               /* I    Correlation vecto
r                       */
    SKP_int32                       wxx,                /* I    Signal energy
                        */
    SKP_int                         D,                  /* I    Dimension
                        */
    SKP_int                         cQ                  /* I    Q value for c vec
tor 0 - 15              */
)
{
    SKP_int   i, j, lshifts, Qxtra;
    SKP_int32 c_max, w_max, tmp, tmp2, nrg;
    SKP_int   cn[ MAX_MATRIX_SIZE ];
    const SKP_int32 *pRow;

    /* Safety checks */
    SKP_assert( D >=  0 );
    SKP_assert( D <= 16 );
    SKP_assert( cQ >  0 );
    SKP_assert( cQ < 16 );

    lshifts = 16 - cQ;
    Qxtra = lshifts;

    c_max = 0;
    for( i = 0; i < D; i++ ) {
        c_max = SKP_max_32( c_max, SKP_abs( ( SKP_int32 )c[ i ] ) );
```

```
    }
    Qxtra = SKP_min_int( Qxtra, SKP_Silk_CLZ32( c_max ) - 17 );

    w_max = SKP_max_32( wXX[ 0 ], wXX[ D * D - 1 ] );
    Qxtra = SKP_min_int( Qxtra, SKP_Silk_CLZ32( SKP_MUL( D, SKP_RSHIFT( SKP_SMULW
B( w_max, c_max ), 4 ) ) ) - 5 );
    Qxtra = SKP_max_int( Qxtra, 0 );
    for( i = 0; i < D; i++ ) {
        cn[ i ] = SKP_LSHIFT( ( SKP_int )c[ i ], Qxtra );
        SKP_assert( SKP_abs(cn[i]) <= ( SKP_int16_MAX + 1 ) ); /* Check that SKP_
SMLAWB can be used */
    }
    lshifts -= Qxtra;

    /* Compute wxx - 2 * wXx * c */
    tmp = 0;
    for( i = 0; i < D; i++ ) {
        tmp = SKP_SMLAWB( tmp, wXx[ i ], cn[ i ] );
    }
    nrg = SKP_RSHIFT( wxx, 1 + lshifts ) - tmp;                         /* Q: -ls
hifts - 1 */

    /* Add c' * wXX * c, assuming wXX is symmetric */
    tmp2 = 0;
    for( i = 0; i < D; i++ ) {
        tmp = 0;
        pRow = &wXX[ i * D ];
        for( j = i + 1; j < D; j++ ) {
            tmp = SKP_SMLAWB( tmp, pRow[ j ], cn[ j ] );
        }
        tmp  = SKP_SMLAWB( tmp,  SKP_RSHIFT( pRow[ i ], 1 ), cn[ i ] );
        tmp2 = SKP_SMLAWB( tmp2, tmp,                        cn[ i ] );
    }
    nrg = SKP_ADD_LSHIFT32( nrg, tmp2, lshifts );                       /* Q: -ls
hifts - 1 */

    /* Keep one bit free always, because we add them for LSF interpolation */
    if( nrg < 1 ) {
        nrg = 1;
    } else if( nrg > SKP_RSHIFT( SKP_int32_MAX, lshifts + 2 ) ) {
        nrg = SKP_int32_MAX >> 1;
    } else {
        nrg = SKP_LSHIFT( nrg, lshifts + 1 );                           /* Q0 */
    }
    return nrg;

}
```

A.108. src/SKP_Silk_residual_energy_FIX.c

```
#include "SKP_Silk_main_FIX.h"

/* Calculates residual energies of input subframes where all subframes have LPC_o
rder   */
/* of preceeding samples
       */
void SKP_Silk_residual_energy_FIX(
          SKP_int32 nrgs[ NB_SUBFR ],          /* O    Residual energy per subfr
ame    */
          SKP_int   nrgsQ[ NB_SUBFR ],         /* O    Q value per subframe
       */
    const SKP_int16 x[],                       /* I    Input signal
       */
    const SKP_int16 a_Q12[ 2 ][ MAX_LPC_ORDER ],/* I    AR coefs for each frame h
alf    */
    const SKP_int32 gains[ NB_SUBFR ],         /* I    Quantization gains
       */
    const SKP_int   subfr_length,              /* I    Subframe length
       */
    const SKP_int   LPC_order                  /* I    LPC order
       */
)
{
    SKP_int         offset, i, j, rshift, lz1, lz2;
    SKP_int16       *LPC_res_ptr, LPC_res[ ( MAX_FRAME_LENGTH + NB_SUBFR * MAX_LP
C_ORDER ) / 2 ];
    const SKP_int16 *x_ptr;
    SKP_int16       S[ MAX_LPC_ORDER ];
    SKP_int32       tmp32;

    x_ptr  = x;
```

```
    offset = LPC_order + subfr_length;

    /* Filter input to create the LPC residual for each frame half, and measure s
ubframe energies */
    for( i = 0; i < 2; i++ ) {
        /* Calculate half frame LPC residual signal including preceeding samples
*/
        SKP_memset( S, 0, LPC_order * sizeof( SKP_int16 ) );
        SKP_Silk_LPC_analysis_filter( x_ptr, a_Q12[ i ], S, LPC_res, ( NB_SUBFR >
> 1 ) * offset, LPC_order );

        /* Point to first subframe of the just calculated LPC residual signal */
        LPC_res_ptr = LPC_res + LPC_order;
        for( j = 0; j < ( NB_SUBFR >> 1 ); j++ ) {
            /* Measure subframe energy */
            SKP_Silk_sum_sqr_shift( &nrgs[ i * ( NB_SUBFR >> 1 ) + j ], &rshift,
LPC_res_ptr, subfr_length );

            /* Set Q values for the measured energy */
            nrgsQ[ i * ( NB_SUBFR >> 1 ) + j ] = -rshift;

            /* Move to next subframe */
            LPC_res_ptr += offset;
        }
        /* Move to next frame half */
        x_ptr += ( NB_SUBFR >> 1 ) * offset;
    }

    /* Apply the squared subframe gains */
    for( i = 0; i < NB_SUBFR; i++ ) {
        /* Fully upscale gains and energies */
        lz1 = SKP_Silk_CLZ32( nrgs[  i ] ) - 1;
        lz2 = SKP_Silk_CLZ32( gains[ i ] ) - 1;

        tmp32 = SKP_LSHIFT32( gains[ i ], lz2 );

        /* Find squared gains */
        tmp32 = SKP_SMMUL( tmp32, tmp32 ); // Q( 2 * lz2 - 32 )

        /* Scale energies */
        nrgs[ i ] = SKP_SMMUL( tmp32, SKP_LSHIFT32( nrgs[ i ], lz1 ) ); // Q( nrg
sQ[ i ] + lz1 + 2 * lz2 - 32 - 32 )
        nrgsQ[ i ] += lz1 + 2 * lz2 - 32 - 32;
    }
}
```

A.109. src/SKP_Silk_scale_copy_vector16.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_SigProc_FIX.h"

/* Copy and multiply a vector by a constant */
void SKP_Silk_scale_copy_vector16(
    SKP_int16           *data_out,
    const SKP_int16     *data_in,
    SKP_int32           gain_Q16,                   /* (I):   gain in Q16   */
    const SKP_int       dataSize                    /* (I):   length        */
)
{
    SKP_int  i;
    SKP_int32 tmp32;

    for( i = 0; i < dataSize; i++ ) {
        tmp32 = SKP_SMULWB( gain_Q16, data_in[ i ] );
        data_out[ i ] = (SKP_int16)SKP_CHECK_FIT16( tmp32 );
    }
}
```

A.110. src/SKP_Silk_scale_vector.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_SigProc_FIX.h"

/* Multiply a vector by a constant */
void SKP_Silk_scale_vector16_Q14(
    SKP_int16           *data1,
    SKP_int             gain_Q14,       /* Gain in Q14 */
    SKP_int             dataSize
)
{
    SKP_int   i;
    SKP_int32 data32, gain_Q16;

    SKP_assert( gain_Q14 <   32768 );
    SKP_assert( gain_Q14 >= -32768 );

    gain_Q16 = SKP_LSHIFT( gain_Q14, 2 );

    if( (SKP_int32)( (SKP_int_ptr_size)data1 & 3 ) != 0 ) {
```

```
        /* Input is not 4-byte aligned */
        data1[ 0 ] = SKP_SMULWB( gain_Q16, data1[ 0 ] );
        i = 1;
    } else {
        i = 0;
    }
    dataSize--;
    for( ; i < dataSize; i += 2 ) {
        data32 = *( (SKP_int32 *)&data1[ i ] );                    /* load two v
alues at once */
        data1[ i     ] = SKP_SMULWB( gain_Q16, data32 );
        data1[ i + 1 ] = SKP_SMULWT( gain_Q16, data32 );
    }
    if( i == dataSize ) {
        /* One sample left to process */
        data1[ i ] = SKP_SMULWB( gain_Q16, data1[ i ] );
    }
}

/* Multiply a vector by a constant */
void SKP_Silk_scale_vector32_Q26_lshift_18(
    SKP_int32           *data1,                    /* (I/O): Q0/Q18        */
    SKP_int32           gain_Q26,                  /* (I):   Q26           */
    SKP_int             dataSize                   /* (I):   length        */
)
{
    SKP_int  i;

    for( i = 0; i < dataSize; i++ ) {
        data1[ i ] = (SKP_int32)SKP_CHECK_FIT32( SKP_RSHIFT64( SKP_SMULL( data1[
i ], gain_Q26 ), 8 ) );// OUTPUT: Q18
    }
}

/* Multiply a vector by a constant */
void SKP_Silk_scale_vector32_16_Q14(
    SKP_int32           *data1,                    /* (I/O): Q0/Q0         */
    SKP_int             gain_Q14,                  /* (I):   Q14           */
    SKP_int             dataSize                   /* (I):   length        */
)
{
    SKP_int  i, gain_Q16;

    if( gain_Q14 < ( SKP_int16_MAX >> 2 ) ) {
        gain_Q16 = SKP_LSHIFT( gain_Q14, 2 );
        for( i = 0; i < dataSize; i++ ) {
            data1[ i ] = SKP_SMULWB( data1[ i ], gain_Q16 );
        }
    } else {
        SKP_assert( gain_Q14 >= SKP_int16_MIN );
```

```
        for( i = 0; i < dataSize; i++ ) {
            data1[ i ] = SKP_LSHIFT( SKP_SMULWB( data1[ i ], gain_Q14 ), 2 );
        }
    }
}

/* Multiply a vector by a constant, does not saturate output data */
void SKP_Silk_scale_vector32_Q16(
    SKP_int32           *data1,                     /* (I/O): Q0/Q0         */
    SKP_int32           gain_Q16,                   /* (I):   gain in Q16 ( SKP_i
nt16_MIN <= gain_Q16 <= SKP_int16_MAX + 65536 ) */
    const SKP_int       dataSize                    /* (I):   length        */
)
{
    SKP_int     i;

    SKP_assert( gain_Q16 <= SKP_int16_MAX + 65536 );
    SKP_assert( gain_Q16 >= SKP_int16_MIN );

    if( gain_Q16 > SKP_int16_MAX ) {
        gain_Q16 -= 65536;
        for( i = 0; i < dataSize; i++ ) {
            data1[ i ] = SKP_SMLAWB( data1[ i ], data1[ i ], gain_Q16 );
        }
    } else {
        for( i = 0; i < dataSize; i++ ) {
            data1[ i ] = SKP_SMULWB( data1[ i ], gain_Q16 );
        }
    }
}
```

A.111. src/SKP_Silk_schur.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
```

```
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_schur.c                                                   *
 *                                                                     *
 * Calculates the reflection coefficients from the correlation sequence *
 *                                                                     *
 * Copyright 2008 (c), Skype Limited                                   *
 * Date: 080103                                                        *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"

/* Faster than schur64(), but much less accurate.                     */
/* uses SMLAWB(), requiring armv5E and higher.                        */
void SKP_Silk_schur(
    SKP_int16           *rc_Q15,                /* O:    reflection coefficients
 [order] Q15         */
    const SKP_int32     *c,                     /* I:    correlations [order+1]
                    */
    const SKP_int32     order                   /* I:    prediction order
                    */
)
{
    SKP_int        k, n, lz;
    SKP_int32    C[ SigProc_MAX_ORDER_LPC + 1 ][ 2 ];
    SKP_int32    Ctmp1, Ctmp2, rc_tmp_Q15;

    /* Get number of leading zeros */
    lz = SKP_Silk_CLZ32( c[ 0 ] );

    /* Copy correlations and adjust level to Q30 */
    if( lz < 2 ) {
        /* lz must be 1, so shift one to the right */
        for( k = 0; k < order + 1; k++ ) {
            C[ k ][ 0 ] = C[ k ][ 1 ] = SKP_RSHIFT( c[ k ], 1 );
        }
    } else if( lz > 2 ) {
        /* Shift to the left */
        lz -= 2;
        for( k = 0; k < order + 1; k++ ) {
```

```
            C[ k ][ 0 ] = C[ k ][ 1 ] = SKP_LSHIFT( c[k], lz );
        }
    } else {
        /* No need to shift */
        for( k = 0; k < order + 1; k++ ) {
            C[ k ][ 0 ] = C[ k ][ 1 ] = c[ k ];
        }
    }

    for( k = 0; k < order; k++ ) {

        /* Get reflection coefficient */
        rc_tmp_Q15 = -SKP_DIV32_16( C[ k + 1 ][ 0 ], SKP_max_32( SKP_RSHIFT( C[ 0
 ][ 1 ], 15 ), 1 ) );

        /* Clip (shouldn't happen for properly conditioned inputs) */
        rc_tmp_Q15 = SKP_SAT16( rc_tmp_Q15 );

        /* Store */
        rc_Q15[ k ] = (SKP_int16)rc_tmp_Q15;

        /* Update correlations */
        for( n = 0; n < order - k; n++ ) {
            Ctmp1 = C[ n + k + 1 ][ 0 ];
            Ctmp2 = C[ n ][ 1 ];
            C[ n + k + 1 ][ 0 ] = SKP_SMLAWB( Ctmp1, SKP_LSHIFT( Ctmp2, 1 ), rc_t
mp_Q15 );
            C[ n ][ 1 ]         = SKP_SMLAWB( Ctmp2, SKP_LSHIFT( Ctmp1, 1 ), rc_t
mp_Q15 );
        }
    }
}
```

A.112. src/SKP_Silk_schur64.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
```

```
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_schur64.c                                                  *
 *                                                                     *
 * Calculates the reflection coefficients from the correlation sequence *
 * using extra precision                                               *
 *                                                                     *
 * Copyright 2008 (c), Skype Limited                                   *
 * Date: 080103                                                        *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"

/* Slower than schur(), but more accurate.                             */
/* Uses SMULL(), available on armv4                                    */
SKP_int32 SKP_Silk_schur64(                     /* O:    Returns residual energy
                */
    SKP_int32           rc_Q16[],               /* O:    Reflection coefficients
 [order] Q16         */
    const SKP_int32     c[],                    /* I:    Correlations [order+1]
                    */
    SKP_int32           order                   /* I:    Prediction order
                    */
)
{
    SKP_int   k, n;
    SKP_int32 C[ SigProc_MAX_ORDER_LPC + 1 ][ 2 ];
    SKP_int32 Ctmp1_Q30, Ctmp2_Q30, rc_tmp_Q31;

    /* Check for invalid input */
    if( c[ 0 ] <= 0 ) {
        SKP_memset( rc_Q16, 0, order * sizeof( SKP_int32 ) );
        return 0;
    }

    for( k = 0; k < order + 1; k++ ) {
        C[ k ][ 0 ] = C[ k ][ 1 ] = c[ k ];
    }

    for( k = 0; k < order; k++ ) {
        /* Get reflection coefficient: divide two Q30 values and get result in Q3
1 */
```

```
        rc_tmp_Q31 = SKP_DIV32_varQ( -C[ k + 1 ][ 0 ], C[ 0 ][ 1 ], 31 );

        /* Save the output */
        rc_Q16[ k ] = SKP_RSHIFT_ROUND( rc_tmp_Q31, 15 );

        /* Update correlations */
        for( n = 0; n < order - k; n++ ) {
            Ctmp1_Q30 = C[ n + k + 1 ][ 0 ];
            Ctmp2_Q30 = C[ n ][ 1 ];

            /* Multiply and add the highest int32 */
            C[ n + k + 1 ][ 0 ] = Ctmp1_Q30 + SKP_SMMUL( SKP_LSHIFT( Ctmp2_Q30, 1
 ), rc_tmp_Q31 );
            C[ n ][ 1 ]         = Ctmp2_Q30 + SKP_SMMUL( SKP_LSHIFT( Ctmp1_Q30, 1
 ), rc_tmp_Q31 );
        }
    }

    return( C[ 0 ][ 1 ] );
}
```

A.113. src/SKP_Silk_shell_coder.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
```

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

#include "SKP_Silk_main.h"

/* shell coder; pulse-subframe length is hardcoded */

SKP_INLINE void combine_pulses(
    SKP_int         *out,   /* O:   combined pulses vector [len] */
    const SKP_int   *in,    /* I:   input vector       [2 * len] */
    const SKP_int   len     /* I:   number of OUTPUT samples     */
)
{
    SKP_int k;
    for( k = 0; k < len; k++ ) {
        out[ k ] = in[ 2 * k ] + in[ 2 * k + 1 ];
    }
}

SKP_INLINE void encode_split(
    SKP_Silk_range_coder_state  *sRC,           /* I/O: compressor data structure
                    */
    const SKP_int               p_child1,       /* I:   pulse amplitude of first
child subframe     */
    const SKP_int               p,              /* I:   pulse amplitude of curren
t subframe         */
    const SKP_uint16            *shell_table    /* I:   table of shell cdfs
                    */
)
{
    const SKP_uint16 *cdf;

    if( p > 0 ) {
        cdf = &shell_table[ SKP_Silk_shell_code_table_offsets[ p ] ];
        SKP_Silk_range_encoder( sRC, p_child1, cdf );
    }
}

SKP_INLINE void decode_split(
    SKP_int                     *p_child1,      /* O:   pulse amplitude of first
child subframe     */
    SKP_int                     *p_child2,      /* O:   pulse amplitude of second
 child subframe    */
    SKP_Silk_range_coder_state  *sRC,           /* I/O: compressor data structure
                    */
    const SKP_int               p,              /* I:   pulse amplitude of curren
t subframe         */
    const SKP_uint16            *shell_table    /* I:   table of shell cdfs
                    */
)
{
    SKP_int cdf_middle;
    const SKP_uint16 *cdf;

    if( p > 0 ) {
        cdf_middle = SKP_RSHIFT( p, 1 );
        cdf = &shell_table[ SKP_Silk_shell_code_table_offsets[ p ] ];
        SKP_Silk_range_decoder( p_child1, sRC, cdf, cdf_middle );
```

```
        p_child2[ 0 ] = p - p_child1[ 0 ];
    } else {
        p_child1[ 0 ] = 0;
        p_child2[ 0 ] = 0;
    }
}

/* Shell encoder, operates on one shell code frame of 16 pulses */
void SKP_Silk_shell_encoder(
    SKP_Silk_range_coder_state      *sRC,               /* I/O  compressor data s
tructure                    */
    const SKP_int                   *pulses0            /* I    data: nonnegative
 pulse amplitudes          */
)
{
    SKP_int pulses1[ 8 ], pulses2[ 4 ], pulses3[ 2 ], pulses4[ 1 ];

    /* this function operates on one shell code frame of 16 pulses */
    SKP_assert( SHELL_CODEC_FRAME_LENGTH == 16 );

    /* tree representation per pulse-subframe */
    combine_pulses( pulses1, pulses0, 8 );
    combine_pulses( pulses2, pulses1, 4 );
    combine_pulses( pulses3, pulses2, 2 );
    combine_pulses( pulses4, pulses3, 1 );

    encode_split( sRC, pulses3[  0 ], pulses4[ 0 ], SKP_Silk_shell_code_table3 );

    encode_split( sRC, pulses2[  0 ], pulses3[ 0 ], SKP_Silk_shell_code_table2 );

    encode_split( sRC, pulses1[  0 ], pulses2[ 0 ], SKP_Silk_shell_code_table1 );
    encode_split( sRC, pulses0[  0 ], pulses1[ 0 ], SKP_Silk_shell_code_table0 );
    encode_split( sRC, pulses0[  2 ], pulses1[ 1 ], SKP_Silk_shell_code_table0 );

    encode_split( sRC, pulses1[  2 ], pulses2[ 1 ], SKP_Silk_shell_code_table1 );
    encode_split( sRC, pulses0[  4 ], pulses1[ 2 ], SKP_Silk_shell_code_table0 );
    encode_split( sRC, pulses0[  6 ], pulses1[ 3 ], SKP_Silk_shell_code_table0 );

    encode_split( sRC, pulses2[  2 ], pulses3[ 1 ], SKP_Silk_shell_code_table2 );

    encode_split( sRC, pulses1[  4 ], pulses2[ 2 ], SKP_Silk_shell_code_table1 );
    encode_split( sRC, pulses0[  8 ], pulses1[ 4 ], SKP_Silk_shell_code_table0 );
    encode_split( sRC, pulses0[ 10 ], pulses1[ 5 ], SKP_Silk_shell_code_table0 );

    encode_split( sRC, pulses1[  6 ], pulses2[ 3 ], SKP_Silk_shell_code_table1 );
    encode_split( sRC, pulses0[ 12 ], pulses1[ 6 ], SKP_Silk_shell_code_table0 );
    encode_split( sRC, pulses0[ 14 ], pulses1[ 7 ], SKP_Silk_shell_code_table0 );
}
```

```
/* Shell decoder, operates on one shell code frame of 16 pulses */
void SKP_Silk_shell_decoder(
    SKP_int                         *pulses0,           /* O    data: nonnegative
 pulse amplitudes          */
    SKP_Silk_range_coder_state      *sRC,               /* I/O  compressor data s
tructure                  */
    const SKP_int                   pulses4             /* I    number of pulses
per pulse-subframe        */
)
{
    SKP_int pulses3[ 2 ], pulses2[ 4 ], pulses1[ 8 ];

    /* this function operates on one shell code frame of 16 pulses */
    SKP_assert( SHELL_CODEC_FRAME_LENGTH == 16 );

    decode_split( &pulses3[  0 ], &pulses3[  1 ], sRC, pulses4,      SKP_Silk_she
ll_code_table3 );

    decode_split( &pulses2[  0 ], &pulses2[  1 ], sRC, pulses3[ 0 ], SKP_Silk_she
ll_code_table2 );

    decode_split( &pulses1[  0 ], &pulses1[  1 ], sRC, pulses2[ 0 ], SKP_Silk_she
ll_code_table1 );
    decode_split( &pulses0[  0 ], &pulses0[  1 ], sRC, pulses1[ 0 ], SKP_Silk_she
ll_code_table0 );
    decode_split( &pulses0[  2 ], &pulses0[  3 ], sRC, pulses1[ 1 ], SKP_Silk_she
ll_code_table0 );

    decode_split( &pulses1[  2 ], &pulses1[  3 ], sRC, pulses2[ 1 ], SKP_Silk_she
ll_code_table1 );
    decode_split( &pulses0[  4 ], &pulses0[  5 ], sRC, pulses1[ 2 ], SKP_Silk_she
ll_code_table0 );
    decode_split( &pulses0[  6 ], &pulses0[  7 ], sRC, pulses1[ 3 ], SKP_Silk_she
ll_code_table0 );

    decode_split( &pulses2[  2 ], &pulses2[  3 ], sRC, pulses3[ 1 ], SKP_Silk_she
ll_code_table2 );

    decode_split( &pulses1[  4 ], &pulses1[  5 ], sRC, pulses2[ 2 ], SKP_Silk_she
ll_code_table1 );
    decode_split( &pulses0[  8 ], &pulses0[  9 ], sRC, pulses1[ 4 ], SKP_Silk_she
ll_code_table0 );
    decode_split( &pulses0[ 10 ], &pulses0[ 11 ], sRC, pulses1[ 5 ], SKP_Silk_she
ll_code_table0 );

    decode_split( &pulses1[  6 ], &pulses1[  7 ], sRC, pulses2[ 3 ], SKP_Silk_she
ll_code_table1 );
    decode_split( &pulses0[ 12 ], &pulses0[ 13 ], sRC, pulses1[ 6 ], SKP_Silk_she
ll_code_table0 );
    decode_split( &pulses0[ 14 ], &pulses0[ 15 ], sRC, pulses1[ 7 ], SKP_Silk_she
ll_code_table0 );
}
```

A.114. src/SKP_Silk_sigm_Q15.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
```

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the

```
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_sigm_Q15.c                                                      *
 *                                                                     *
 * Approximate sigmoid function                                        *
 *                                                                     *
 * Copyright 2006 (c), Skype Limited                                   *
 * Date: 060221                                                        *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"
/********************************/
/* approximate sigmoid function */
/********************************/
/* fprintf(1, '%d, ', round(1024 * ([1 ./ (1 + exp(-(1:5))), 1] - 1 ./ (1 + exp(-
(0:5))))); */
static const SKP_int32 sigm_LUT_slope_Q10[ 6 ] = {
    237, 153, 73, 30, 12, 7
};
/* fprintf(1, '%d, ', round(32767 * 1 ./ (1 + exp(-(0:5))))); */
static const SKP_int32 sigm_LUT_pos_Q15[ 6 ] = {
    16384, 23955, 28861, 31213, 32178, 32548
};
/* fprintf(1, '%d, ', round(32767 * 1 ./ (1 + exp((0:5))))); */
static const SKP_int32 sigm_LUT_neg_Q15[ 6 ] = {
    16384, 8812, 3906, 1554, 589, 219
};

SKP_int SKP_Silk_sigm_Q15( SKP_int in_Q5 )
{
    SKP_int ind;

    if( in_Q5 < 0 ) {
```

```
        /* Negative input */
        in_Q5 = -in_Q5;
        if( in_Q5 >= 6 * 32 ) {
            return 0;        /* Clip */
        } else {
            /* Linear interpolation of look up table */
            ind = SKP_RSHIFT( in_Q5, 5 );
            return( sigm_LUT_neg_Q15[ ind ] - SKP_SMULBB( sigm_LUT_slope_Q10[ ind
 ], in_Q5 & 0x1F ) );
        }
    } else {
        /* Positive input */
        if( in_Q5 >= 6 * 32 ) {
            return 32767;        /* clip */
        } else {
            /* Linear interpolation of look up table */
            ind = SKP_RSHIFT( in_Q5, 5 );
            return( sigm_LUT_pos_Q15[ ind ] + SKP_SMULBB( sigm_LUT_slope_Q10[ ind
 ], in_Q5 & 0x1F ) );
        }
    }
}
```

A.115. src/SKP_Silk_SigProc_FIX.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
```

```
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef _SKP_SILK_SIGPROC_FIX_H_
#define _SKP_SILK_SIGPROC_FIX_H_

#ifdef  __cplusplus
extern "C"
{
#endif

#define SigProc_MAX_ORDER_LPC           16                  /* max order of th
e LPC analysis in schur() and k2a()    */
#define SigProc_MAX_CORRELATION_LENGTH   640                /* max input lengt
h to the correlation                    */
#include "SKP_Silk_typedef.h"
#include <string.h>
#include <stdlib.h>                                         /* for abs() */
#include "SKP_Silk_resample_rom.h"

#include "SKP_Silk_macros.h"



/************************************************************************/
/*                    SIGNAL PROCESSING FUNCTIONS                       */
/************************************************************************/

/* downsample by a factor 2 */
void SKP_Silk_resample_1_2(
    const SKP_int16      *in,            /* I:   16 kHz signal [2*len]    */
    SKP_int32            *S,             /* I/O: State vector [6]         */
    SKP_int16            *out,           /* O:   8 kHz signal [len]       */
    SKP_int32            *scratch,       /* I:   Scratch memory [4*len]   */
    const SKP_int32      len             /* I:   Number of OUTPUT samples */
);

/*!
 * downsample by a factor 2, coarser (good for resampling audio)
 */
void SKP_Silk_resample_1_2_coarse(
    const SKP_int16    *in,              /* I:   16 kHz signal [2*len]    */
          SKP_int32    *S,               /* I/O: state vector [4]         */
          SKP_int16    *out,             /* O:   8 kHz signal [len]       */
          SKP_int32    *scratch,         /* I:   scratch memory [3*len]   */
    const SKP_int32    len               /* I:   number of OUTPUT samples */
);
```

```
/*!
 * downsample by a factor 2, coarsest (good for signals that are already oversamp
led, or for analysis purposes)
 */
void SKP_Silk_resample_1_2_coarsest(
    const SKP_int16     *in,             /* I:   16 kHz signal [2*len]    */
    SKP_int32           *S,              /* I/O: State vector [2]         */
    SKP_int16           *out,            /* O:   8 kHz signal [len]       */
    SKP_int32           *scratch,        /* I:   Scratch memory [3*len]   */
    const SKP_int32     len              /* I:   Number of OUTPUT samples */
);

/*!
 * upsample by a factor 2, coarser (good for resampling audio)
 */
void SKP_Silk_resample_2_1_coarse(
    const SKP_int16      *in,            /* I:   8 kHz signal [len]      */
    SKP_int32            *S,             /* I/O: State vector [4]        */
    SKP_int16            *out,           /* O:   16 kHz signal [2*len]   */
    SKP_int32            *scratch,       /* I:   Scratch memory [3*len]  */
    const SKP_int32      len             /* I:   Number of INPUT samples */
);

/*!
 * Resamples by a factor 1/3
 */
void SKP_Silk_resample_1_3(
    SKP_int16            *out,          /* O:   Fs_low signal  [inLen/3]
     */
    SKP_int32            *S,            /* I/O: State vector   [7]
     */
    const SKP_int16      *in,           /* I:   Fs_high signal [inLen]
     */
    const SKP_int32      inLen          /* I:   Input length, must be a multiple
of 3 */
);

/*!
 * Resamples by a factor 3/1
 */
void SKP_Silk_resample_3_1(
    SKP_int16            *out,          /* O:   Fs_high signal [inLen*3]
 */
    SKP_int32            *S,            /* I/O: State vector   [7]
 */
    const SKP_int16      *in,           /* I:   Fs_low signal  [inLen]
 */
    const SKP_int32      inLen          /* I:   Input length
 */
);

/*!
 * Resamples by a factor 2/3
 */
void SKP_Silk_resample_2_3(
    SKP_int16            *out,          /* O:   Fs_low signal    [inLen * 2/3]
     */
    SKP_int32            *S,            /* I/O: State vector    [7+4]
     */
```

```
    const SKP_int16      *in,           /* I:   Fs_high signal    [inLen]
     */
    const SKP_int        inLen          /* I:   Input length, must be a multiple
of 3 */
);

/*!
 * Resamples by a factor 3/2
 */
void SKP_Silk_resample_3_2(
    SKP_int16            *out,          /*   O: Fs_high signal  [inLen*3/2]
         */
    SKP_int32            *S,            /* I/O: State vector    [7+4]
         */
    const SKP_int16      *in,           /* I:   Fs_low signal   [inLen]
         */
    SKP_int              inLen          /* I:   Input length, must be a multiple
of 2    */
);

/*!
 * Resamples by a factor 4/3
 */
void SKP_Silk_resample_4_3(
    SKP_int16            *out,          /* O:   Fs_low signal    [inLen * 4/3]
         */
    SKP_int32            *S,            /* I/O: State vector    [7+4+4]
         */
    const SKP_int16      *in,           /* I:   Fs_high signal    [inLen]
         */
    const SKP_int        inLen          /* I:   input length, must be a multiple
of 3    */
);

/*!
 * Resamples by a factor 3/4
 */
 void SKP_Silk_resample_3_4(
    SKP_int16            *out,          /* O:   Fs_high signal  [inLen*3/4]
         */
    SKP_int32            *S,            /* I/O: State vector    [7+2+6]
         */
    const SKP_int16      *in,           /* I:   Fs_low signal   [inLen]
         */
    SKP_int              inLen          /* I:   Input length, must be a multiple
of 4    */
);

/*!
 * resample with a factor 2/3 coarse
 */
void SKP_Silk_resample_2_3_coarse(
    SKP_int16            *out,          /* O:   Output signal
                                           */
    SKP_int16            *S,            /* I/O: Resampler state [ SigProc_Resampl
e_2_3_coarse_NUM_FIR_COEFS - 1 ]           */
    const SKP_int16      *in,           /* I:   Input signal
                                           */
    const SKP_int        frameLenIn,    /* I:   Number of input samples
                                           */
    SKP_int16            *scratch       /* I:   Scratch memory [ frameLenIn + Sig
Proc_Resample_2_3_coarse_NUM_FIR_COEFS - 1 ] */
```

```
);

/*!
 * resample with a factor 2/3 coarsest
 */
```

```
void SKP_Silk_resample_2_3_coarsest(
    SKP_int16           *out,           /* O:   Output signal
                                           */
    SKP_int16           *S,             /* I/O: Resampler state [ SigProc_Resampl
e_2_3_coarsest_NUM_FIR_COEFS - 1 ]         */
    const SKP_int16     *in,            /* I:   Input signal
                                           */
    const SKP_int       frameLenIn,     /* I:   Number of input samples
                                           */
    SKP_int16           *scratch        /* I:   Scratch memory [ frameLenIn + Sig
Proc_Resample_2_3_coarsest_NUM_FIR_COEFS - 1 ] */
);

/*!
 * First order low-pass filter, with input as SKP_int16, running at 48 kHz
 */
void SKP_Silk_lowpass_short(
    const SKP_int16     *in,            /* I:   Q15 48 kHz signal; [len]    */
    SKP_int32           *S,             /* I/O: Q25 state; length = 1       */
    SKP_int32           *out,           /* O:   Q25 48 kHz signal; [len]    */
    const SKP_int32     len             /* O:   Signal length               */
);

/*!
 * First order low-pass filter, with input as SKP_int32, running at 48 kHz
 */
void SKP_Silk_lowpass_int(
    const SKP_int32     *in,            /* I:   Q25 48 kHz signal; length = len
*/
    SKP_int32           *S,             /* I/O: Q25 state; length = 1
*/
    SKP_int32           *out,           /* O:   Q25 48 kHz signal; length = len
*/
    const SKP_int32     len             /* I:   Number of samples
*/
);

/*!
 * First-order allpass filter
 */
void SKP_Silk_allpass_int(
    const SKP_int32     *in,          /* I:   Q25 input signal [len]
   */
    SKP_int32           *S,           /* I/O: Q25 state [1]
   */
    SKP_int             A,            /* I:   Q15 coefficient    (0 <= A < 32768
)  */
    SKP_int32           *out,         /* O:   Q25 output signal [len]
   */
    const SKP_int32     len           /* I:   Number of samples
   */
);

/*!
 * second order ARMA filter
 * can handle (slowly) varying coefficients
 */
void SKP_Silk_biquad(
    const SKP_int16     *in,          /* I:   input signal                */
    const SKP_int16     *B,           /* I:   MA coefficients, Q13 [3]    */
    const SKP_int16     *A,           /* I:   AR coefficients, Q13 [2]    */
          SKP_int32     *S,           /* I/O: state vector [2]            */
```

```
          SKP_int16      *out,         /* O:   output signal               */
    const SKP_int32      len           /* I:   signal length               */
);
/*!
 * second order ARMA filter;
 * slower than biquad() but uses more precise coefficients
 * can handle (slowly) varying coefficients
 */
void SKP_Silk_biquad_alt(
    const SKP_int16     *in,           /* I:    input signal                */
    const SKP_int32     *B_Q28,        /* I:    MA coefficients [3]         */
    const SKP_int32     *A_Q28,        /* I:    AR coefficients [2]         */
    SKP_int32           *S,            /* I/O: state vector [2]             */
    SKP_int16           *out,          /* O:    output signal               */
    const SKP_int32     len            /* I:    signal length (must be even) */
);

/*!
 * variable order MA filter. Prediction error filter implementation. Coeficients
negated and starting with coef to x[n - 1]
 */
void SKP_Silk_MA_Prediction(
    const SKP_int16      *in,          /* I:   Input signal
          */
    const SKP_int16      *B,           /* I:   MA prediction coefficients, Q12 [o
rder]     */
    SKP_int32            *S,           /* I/O: State vector [order]
          */
    SKP_int16            *out,         /* O:   Output signal
          */
    const SKP_int32      len,          /* I:   Signal length
          */
    const SKP_int32      order         /* I:  Filter order
          */
);

void SKP_Silk_MA_Prediction_Q13(
    const SKP_int16      *in,          /* I:   input signal
          */
    const SKP_int16      *B,           /* I:   MA prediction coefficients, Q13 [o
rder]     */
    SKP_int32            *S,           /* I/O: state vector [order]
          */
    SKP_int16            *out,         /* O:   output signal
          */
    SKP_int32            len,          /* I:   signal length
          */
    SKP_int32            order         /* I:   filter order
          */
);

/*!
 * 16th order AR filter for LPC synthesis, coefficients are in Q12
 */
void SKP_Silk_LPC_synthesis_order16(
    const SKP_int16      *in,          /* I:   excitation signal
           */
    const SKP_int16      *A_Q12,       /* I:   AR coefficients [16], between -8_Q
0 and 8_Q0 */
    const SKP_int32      Gain_Q26,     /* I:   gain
           */
          SKP_int32      *S,           /* I/O: state vector [16]
```

```
          */
         SKP_int16      *out,          /* O:   output signal
          */
   const SKP_int32      len            /* I:   signal length, must be multiple of
16        */
```

```
);

/* variable order MA prediction error filter. */
/* Inverse filter of SKP_Silk_LPC_synthesis_filter */
void SKP_Silk_LPC_analysis_filter(
    const SKP_int16      *in,          /* I:   Input signal
          */
    const SKP_int16      *B,           /* I:   MA prediction coefficients, Q12 [o
rder]     */
    SKP_int16            *S,           /* I/O: State vector [order]
          */
    SKP_int16            *out,         /* O:   Output signal
          */
    const SKP_int32      len,          /* I:   Signal length
          */
    const SKP_int32      Order         /* I:   Filter order
          */
);

/* even order AR filter */
void SKP_Silk_LPC_synthesis_filter(
    const SKP_int16      *in,          /* I:   excitation signal
             */
    const SKP_int16      *A_Q12,       /* I:   AR coefficients [Order], between -
8_Q0 and 8_Q0 */
    const SKP_int32      Gain_Q26,     /* I:   gain
             */
    SKP_int32            *S,           /* I/O: state vector [Order]
             */
    SKP_int16            *out,         /* O:   output signal
             */
    const SKP_int32      len,          /* I:   signal length
             */
    const SKP_int        Order         /* I:   filter order, must be even
             */
);

/* Chirp (bandwidth expand) LP AR filter */
void SKP_Silk_bwexpander(
    SKP_int16            *ar,          /* I/O  AR filter to be expanded (without
leading 1)    */
    const SKP_int        d,            /* I    Length of ar
             */
    SKP_int32            chirp_Q16     /* I    Chirp factor (typically in the ran
ge 0 to 1)    */
);

/* Chirp (bandwidth expand) LP AR filter */
void SKP_Silk_bwexpander_32(
    SKP_int32            *ar,          /* I/O  AR filter to be expanded (without
leading 1)    */
    const SKP_int        d,            /* I    Length of ar
             */
    SKP_int32            chirp_Q16     /* I    Chirp factor in Q16
             */
);

/* Compute inverse of LPC prediction gain, and                          */
/* test if LPC coefficients are stable (all poles within unit circle)    */
SKP_int SKP_Silk_LPC_inverse_pred_gain( /* O:  Returns 1 if unstable, otherwise 0
          */
    SKP_int32            *invGain_Q30,  /* O:  Inverse prediction gain, Q30 energ
```

```
y domain  */
    const SKP_int16      *A_Q12,        /* I:  Prediction coefficients, Q12 [orde
r]        */
    const SKP_int        order          /* I:  Prediction order
          */
);

SKP_int SKP_Silk_LPC_inverse_pred_gain_Q13( /* O:  returns 1 if unstable, otherwi
se 0      */
    SKP_int32            *invGain_Q30,  /* O:  Inverse prediction gain, Q30 energ
y domain  */
```

```
    const SKP_int16      *A_Q13,        /* I:  Prediction coefficients, Q13 [orde
r]        */
    const SKP_int        order          /* I:  Prediction order
          */
);

/* split signal in two decimated bands using first-order allpass filters */
void SKP_Silk_ana_filt_bank_1(
    const SKP_int16      *in,           /* I:   Input signal [N]        */
    SKP_int32            *S,            /* I/O: State vector [2]        */
    SKP_int16            *outL,         /* O:   Low band [N/2]          */
    SKP_int16            *outH,         /* O:   High band [N/2]         */
    SKP_int32            *scratch,      /* I:   Scratch memory [3*N/2]  */
    const SKP_int32      N              /* I:   Number of input samples */
);

/*****************************************************************/
/*                         SCALAR FUNCTIONS                      */
/*****************************************************************/

/* approximation of 128 * log2() (exact inverse of approx 2^() below) */
/* convert input to a log scale    */
SKP_int32 SKP_Silk_lin2log(const SKP_int32 inLin);        /* I: input in linear s
cale        */

/* Approximation of a sigmoid function */
SKP_int SKP_Silk_sigm_Q15(SKP_int in_Q5);

/* approximation of 2^() (exact inverse of approx log2() above) */
/* convert input to a linear scale    */
SKP_int32 SKP_Silk_log2lin(const SKP_int32 inLog_Q7);    /* I: input on log scale
 */

/* Function that returns the maximum absolut value of the input vector */
SKP_int16 SKP_Silk_int16_array_maxabs(  /* O   Maximum absolute value, max: 2^15-
1   */
    const SKP_int16     *vec,           /* I   Input vector  [len]
    */
    const SKP_int32     len             /* I   Length of input vector
    */
);

/* Compute number of bits to right shift the sum of squares of a vector    */
/* of int16s to make it fit in an int32                                    */
void SKP_Silk_sum_sqr_shift(
    SKP_int32           *energy,        /* O   Energy of x, after shifting to the
 right            */
    SKP_int             *shift,         /* O   Number of bits right shift applied
 to energy        */
    const SKP_int16     *x,             /* I   Input vector
                */
    SKP_int             len             /* I   Length of input vector
                */
);

/* Calculates the reflection coefficients from the correlation sequence    */
/* Faster than schur64(), but much less accurate.                          */
/* Uses SMLAWB(), requiring armv5E and higher.                             */
void SKP_Silk_schur(
```

```
    SKP_int16           *rc_Q15,        /* O:  reflection coefficients [order] Q1
5         */
    const SKP_int32     *c,             /* I:  correlations [order+1]
          */
    const SKP_int32     order           /* I:  prediction order
          */
);

/* Calculates the reflection coefficients from the correlation sequence    */
/* Slower than schur(), but more accurate.                                 */
/* Uses SMULL(), available on armv4                                        */
SKP_int32 SKP_Silk_schur64(             /* O:  returns residual energy
          */
    SKP_int32           rc_Q16[],       /* O:  Reflection coefficients [order] Q1
6         */
    const SKP_int32     c[],            /* I:  Correlations [order+1]
          */
    SKP_int32           order           /* I:  Prediction order
          */
);

/* Step up function, converts reflection coefficients to prediction coefficients
*/
void SKP_Silk_k2a(
    SKP_int32           *A_Q24,         /* O:  Prediction coefficients [order] Q2
4         */
    const SKP_int16     *rc_Q15,        /* I:  Reflection coefficients [order] Q1
5         */
    const SKP_int32     order           /* I:  Prediction order
          */
);

/* Step up function, converts reflection coefficients to prediction coefficients
*/
void SKP_Silk_k2a_Q16(
    SKP_int32           *A_Q24,         /* O:  Prediction coefficients [order] Q2
4         */
    const SKP_int32     *rc_Q16,        /* I:  Reflection coefficients [order] Q1
6         */
    const SKP_int32     order           /* I:  Prediction order
          */
);

/* Apply sine window to signal vector.                                      */
/* Window types:                                                            */
/*    0 -> sine window from 0 to pi                                         */
/*    1 -> sine window from 0 to pi/2                                       */
/*    2 -> sine window from pi/2 to pi                                      */
/* every other sample of window is linearly interpolated, for speed        */
void SKP_Silk_apply_sine_window(
    SKP_int16           px_win[],       /* O  Pointer to windowed signal
          */
    const SKP_int16     px[],           /* I  Pointer to input signal
          */
    const SKP_int       win_type,       /* I  Selects a window type
          */
    const SKP_int       length          /* I  Window length, multiple of 4
          */
);

/* Compute autocorrelation */
void SKP_Silk_autocorr(
```

```
    SKP_int32           *results,       /* O  Result (length correlationCount)
         */
    SKP_int32           *scale,         /* O  Scaling of the correlation vector
         */
    const SKP_int16     *inputData,     /* I  Input data to correlate
         */
    const SKP_int       inputDataSize,  /* I  Length of input
         */
    const SKP_int       correlationCount /* I  Number of correlation taps to comp
ute      */
```

```
);

/* Pitch estimator */
#define SigProc_PITCH_EST_MIN_COMPLEX        0
#define SigProc_PITCH_EST_MID_COMPLEX        1
#define SigProc_PITCH_EST_MAX_COMPLEX        2

void SKP_Silk_decode_pitch(
    SKP_int             lagIndex,        /* I
         */
    SKP_int             contourIndex,    /* O
         */
    SKP_int             pitch_lags[],    /* O 4 pitch values
         */
    SKP_int             Fs_kHz           /* I sampling frequency (kHz)
         */
);

SKP_int SKP_Silk_pitch_analysis_core(     /* O    Voicing estimate: 0 voiced, 1 u
nvoiced                      */
    const SKP_int16     *signal,          /* I    Signal of length PITCH_EST_FRAM
E_LENGTH_MS*Fs_kHz           */
    SKP_int             *pitch_out,       /* O    4 pitch lag values
                             */
    SKP_int             *lagIndex,        /* O    Lag Index
                             */
    SKP_int             *contourIndex,    /* O    Pitch contour Index
                             */
    SKP_int             *LTPCorr_Q15,     /* I/O  Normalized correlation; input:
value from previous frame    */
    SKP_int             prevLag,          /* I    Last lag of previous frame; set
 to zero is unvoiced         */
    const SKP_int32     search_thres1_Q16, /* I    First stage threshold for lag c
andidates 0 - 1              */
    const SKP_int       search_thres2_Q15, /* I    Final threshold for lag candida
tes 0 - 1                    */
    const SKP_int       Fs_kHz,           /* I    Sample frequency (kHz)
                             */
    const SKP_int       complexity        /* I    Complexity setting, 0-2, where
2 is highest                 */
);

/* parameter defining the size and accuracy of the piecewise linear    */
/* cosine approximatin table.                                         */

#define LSF_COS_TAB_SZ_FIX      128
/* rom table with cosine values */
extern const SKP_int SKP_Silk_LSFCosTab_FIX_Q12[ LSF_COS_TAB_SZ_FIX + 1 ];

void SKP_Silk_LPC_fit(
          SKP_int16     *a_QQ,            /* O    stabilized LPC vector, Q(24-rshi
ft) [L]          */
          SKP_int32     *a_Q24,           /* I    LPC vector [L]
                 */
    const SKP_int       QQ,               /* I    Q domain of output LPC vector
                 */
    const SKP_int       L                 /* I    Number of LPC parameters in the
input vector    */
);

/* Compute Normalized Line Spectral Frequencies (NLSFs) from whitening filter coe
fficients        */
```

```
/* If not all roots are found, the a_Q16 coefficients are bandwidth expanded unti
l convergence.    */
void SKP_Silk_A2NLSF(
    SKP_int             *NLSF,            /* O    Normalized Line Spectral Frequen
cies, Q15 (0 - (2^15-1)), [d] */
    SKP_int32           *a_Q16,           /* I/O  Monic whitening filter coefficie
nts in Q16 [d]                */
    const SKP_int       d                 /* I    Filter order (must be even)
                              */
);
```

```
/* compute whitening filter coefficients from normalized line spectral frequencie
s */
void SKP_Silk_NLSF2A(
    SKP_int16          *a,              /* o    monic whitening filter coefficie
nts in Q12,  [d]    */
    const SKP_int      *NLSF,           /* i    normalized line spectral frequen
cies in Q15, [d]    */
    const SKP_int      d                /* i    filter order (should be even)
                    */
);

void SKP_Silk_insertion_sort_increasing(
    SKP_int32          *a,            /* I/O   Unsorted / Sorted vector
        */
    SKP_int            *index,        /* O:    Index vector for the sorted elem
ents    */
    const SKP_int      L,             /* I:    Vector length
        */
    const SKP_int      K              /* I:    Number of correctly sorted posit
ions    */
);

void SKP_Silk_insertion_sort_decreasing(
    SKP_int            *a,            /* I/O:  Unsorted / Sorted vector
        */
    SKP_int            *index,        /* O:    Index vector for the sorted elem
ents    */
    const SKP_int      L,             /* I:    Vector length
        */
    const SKP_int      K              /* I:    Number of correctly sorted posit
ions    */
);

void SKP_Silk_insertion_sort_decreasing_int16(
    SKP_int16          *a,            /* I/O:  Unsorted / Sorted vector
        */
    SKP_int            *index,        /* O:    Index vector for the sorted elem
ents    */
    const SKP_int      L,             /* I:    Vector length
        */
    const SKP_int      K              /* I:    Number of correctly sorted posit
ions    */
);

void SKP_Silk_insertion_sort_increasing_all_values(
     SKP_int           *a,            /* I/O:  Unsorted / Sorted vector
        */
     const SKP_int     L              /* I:    Vector length
        */
);

/* NLSF stabilizer, for a single input data vector */
void SKP_Silk_NLSF_stabilize(
          SKP_int      *NLSF_Q15,     /* I/O:  Unstable/stabilized normalized
LSF vector in Q15 [L]                */
    const SKP_int      *NDeltaMin_Q15, /* I:    Normalized delta min vector in
Q15, NDeltaMin_Q15[L] must be >= 1 [L+1] */
    const SKP_int      L              /* I:    Number of NLSF parameters in th
e input vector                       */
);

/* NLSF stabilizer, over multiple input column data vectors */
```

```
void SKP_Silk_NLSF_stabilize_multi(
          SKP_int        *NLSF_Q15,      /* I/O:  Unstable/stabilized normalized
LSF vectors in Q15 [LxN]                 */
    const SKP_int        *NDeltaMin_Q15, /* I:    Normalized delta min vector in
Q15, NDeltaMin_Q15[L] must be >= 1 [L+1] */
    const SKP_int        N,              /* I:    Number of input vectors to be s
tabilized                                */
    const SKP_int        L               /* I:    NLSF vector dimension
                                         */
);
```

```
/* Laroia low complexity NLSF weights */
void SKP_Silk_NLSF_VQ_weights_laroia(
    SKP_int             *pNLSFW_Q6,     /* O:    Pointer to input vector weights
            [D x 1]      */
    const SKP_int       *pNLSF_Q15,     /* I:    Pointer to input vector
            [D x 1]      */
    const SKP_int       D               /* I:    Input vector dimension (even)
                          */
);

/* Compute reflection coefficients from input signal */
void SKP_Silk_burg_modified(
    SKP_int32           *res_nrg,           /* O   residual energy
                                            */
    SKP_int             *res_nrgQ,          /* O   residual energy Q value
                                            */
    SKP_int32           A_Q16[],            /* O   prediction coefficients (leng
th order)                                   */
    const SKP_int16     x[],                /* I   input signal, length: nb_subf
r * ( D + subfr_length )                    */
    const SKP_int       subfr_length,       /* I   input signal subframe length
(including D preceeding samples)    */
    const SKP_int       nb_subfr,           /* I   number of subframes stacked i
n x                                         */
    const SKP_int32     WhiteNoiseFrac_Q32, /* I   fraction added to zero-lag au
tocorrelation                               */
    const SKP_int       D                   /* I   order
                                            */
);

/* Multiply a vector by a constant */
void SKP_Silk_scale_vector16_Q14(
    SKP_int16           *data1,
    SKP_int             gain_Q14,           /* Gain in Q14 */
    SKP_int             dataSize
);

/* Copy and multiply a vector by a constant */
void SKP_Silk_scale_copy_vector16(
    SKP_int16           *data_out,
    const SKP_int16     *data_in,
    SKP_int32           gain_Q16,           /* I:   gain in Q16   */
    const SKP_int       dataSize            /* I:   length        */
);

void SKP_Silk_scale_vector32_16_Q14(
    SKP_int32           *data1,             /* I/O: Q0/Q0         */
    SKP_int             gain_Q14,           /* I:   Q14           */
    SKP_int             dataSize            /* I:   length        */
);

/* Multiply a vector by a constant, does not saturate output data */
void SKP_Silk_scale_vector32_Q16(
    SKP_int32           *data1,             /* I/O: Q0/Q0         */
    SKP_int32           gain_Q16,           /* I:   gain in Q16 ( SKP_int16_MIN
<= gain_Q16 <= SKP_int16_MAX + 65536 ) */
    const SKP_int       dataSize            /* I:   length        */
);

/* Some for the LTP related function requires Q26 to work.*/
```

```
void SKP_Silk_scale_vector32_Q26_lshift_18(
    SKP_int32           *data1,             /* I/O: Q0/Q18        */
    SKP_int32           gain_Q26,           /* I:   Q26           */
    SKP_int             dataSize            /* I:   length        */
);

/*****************************************************************/
/*                         INLINE ARM MATH                       */
/*****************************************************************/

/*    return sum(inVec1[i]*inVec2[i])    */
/*    inVec1 and inVec2 should be increasing ordered, and starting address should
 be 4 byte aligned. (a factor of 4)*/
SKP_int32 SKP_Silk_inner_prod_aligned(
    const SKP_int16* const inVec1,          /* I   input vector 1    */
    const SKP_int16* const inVec2,          /* I   input vector 2    */
    const SKP_int          len              /* I   vector lengths    */
);

SKP_int32 SKP_Silk_inner_prod16_aligned_sat(
    const SKP_int16* const inVec1,          /* I   input vector 1  */
    const SKP_int16* const inVec2,          /* I   input vector 2  */
    const SKP_int          len              /* I   vector lengths  */
);

SKP_int64 SKP_Silk_inner_prod_aligned_64(
    const SKP_int32        *inVec1,         /* I   input vector 1    */
    const SKP_int32        *inVec2,         /* I   input vector 2    */
    const SKP_int          len              /* I   vector lengths    */
);

SKP_int64 SKP_Silk_inner_prod16_aligned_64(
    const SKP_int16        *inVec1,         /* I   input vector 1    */
    const SKP_int16        *inVec2,         /* I   input vector 2    */
    const SKP_int          len              /* I   vector lengths    */
);
/*****************************************************************/
/*                                 MACROS                        */
/*****************************************************************/

/* Define 4-byte aligned array of SKP_int16 */
#define SKP_array_of_int16_4_byte_aligned( arrayName, nElements )    \
    SKP_int32 dummy_int32 ## arrayName;                               \
    SKP_int16 arrayName[ (nElements) ]

/* Useful Macros that can be adjusted to other platforms */
#define SKP_memcpy(a, b, c)               memcpy((a), (b), (c))    /* Dest, Src,
 ByteCount */
#define SKP_memset(a, b, c)               memset((a), (b), (c))    /* Dest, valu
e, ByteCount */
#define SKP_memmove(a, b, c)              memmove((a), (b), (c))    /* Dest, Src
, ByteCount */
```

```
/* fixed point macros */

// (a32 * b32) output have to be 32bit int
#define SKP_MUL(a32, b32)                   ((a32) * (b32))

// (a32 * b32) output have to be 32bit uint
#define SKP_MUL_uint(a32, b32)              SKP_MUL(a32, b32)

// a32 + (b32 * c32) output have to be 32bit int
#define SKP_MLA(a32, b32, c32)              SKP_ADD32((a32),((b32) * (c32)))

// a32 + (b32 * c32) output have to be 32bit uint
#define SKP_MLA_uint(a32, b32, c32)         SKP_MLA(a32, b32, c32)

// ((a32 >> 16)  * (b32 >> 16)) output have to be 32bit int
#define SKP_SMULTT(a32, b32)                (((a32) >> 16) * ((b32) >> 16))

// a32 + ((a32 >> 16)  * (b32 >> 16)) output have to be 32bit int
#define SKP_SMLATT(a32, b32, c32)           SKP_ADD32((a32),((b32) >> 16) * ((c32)
 >> 16))

#define SKP_SMLALBB(a64, b16, c16)          SKP_ADD64((a64),(SKP_int64)((SKP_int32
)(b16) * (SKP_int32)(c16)))

// (a32 * b32)
#define SKP_SMULL(a32, b32)                 ((SKP_int64)(a32) * /*(SKP_int64)*/(b3
2))

// multiply-accumulate macros that allow overflow in the addition (ie, no asserts
 in debug mode)
#define SKP_MLA_ovflw(a32, b32, c32)        SKP_MLA(a32, b32, c32)
#ifndef SKP_SMLABB_ovflw
#    define SKP_SMLABB_ovflw(a32, b32, c32)    SKP_SMLABB(a32, b32, c32)
#endif
#define SKP_SMLABT_ovflw(a32, b32, c32)     SKP_SMLABT(a32, b32, c32)
#define SKP_SMLATT_ovflw(a32, b32, c32)     SKP_SMLATT(a32, b32, c32)
#define SKP_SMLAWB_ovflw(a32, b32, c32)     SKP_SMLAWB(a32, b32, c32)
#define SKP_SMLAWT_ovflw(a32, b32, c32)     SKP_SMLAWT(a32, b32, c32)

#define SKP_DIV64_32(a64, b32)              ((a64)/(b32))        /* TODO: rewrite
it as a set of SKP_DIV32.*/

#define SKP_DIV32_16(a32, b16)              ((SKP_int32)((a32) / (b16)))
#define SKP_DIV32(a32, b32)                 ((SKP_int32)((a32) / (b32)))

// These macros enables checking for overflow in SKP_Silk_API_Debug.h
#define SKP_ADD16(a, b)                     ((a) + (b))
#define SKP_ADD32(a, b)                     ((a) + (b))
#define SKP_ADD64(a, b)                     ((a) + (b))

#define SKP_SUB16(a, b)                     ((a) - (b))
#define SKP_SUB32(a, b)                     ((a) - (b))
#define SKP_SUB64(a, b)                     ((a) - (b))
```

```
#define SKP_SAT8(a)                         ((a) > SKP_int8_MAX ? SKP_int8_MAX  :
\
                                            ((a) < SKP_int8_MIN ? SKP_int8_MIN  :
(a)))
#define SKP_SAT16(a)                        ((a) > SKP_int16_MAX ? SKP_int16_MAX :
 \
                                            ((a) < SKP_int16_MIN ? SKP_int16_MIN :
 (a)))
#define SKP_SAT32(a)                        ((a) > SKP_int32_MAX ? SKP_int32_MAX :
 \
                                            ((a) < SKP_int32_MIN ? SKP_int32_MIN :
 (a)))

#define SKP_CHECK_FIT8(a)                   (a)
#define SKP_CHECK_FIT16(a)                  (a)
#define SKP_CHECK_FIT32(a)                  (a)

#define SKP_ADD_SAT16(a, b)                 (SKP_int16)SKP_SAT16( SKP_ADD32( (SKP_
int32)(a), (b) ) )
#define SKP_ADD_SAT64(a, b)                 ((((a) + (b)) & 0x8000000000000000LL)
== 0 ?                              \
                                            ((((a) & (b)) & 0x8000000000000000LL)
!= 0 ? SKP_int64_MIN : (a)+(b)) :    \
                                            ((((a) | (b)) & 0x8000000000000000LL)
== 0 ? SKP_int64_MAX : (a)+(b)) )

#define SKP_SUB_SAT16(a, b)                 (SKP_int16)SKP_SAT16( SKP_SUB32( (SKP_
int32)(a), (b) ) )
#define SKP_SUB_SAT64(a, b)                 ((((a)-(b)) & 0x8000000000000000LL) ==
 0 ?                                                 \
                                            (( (a) & ((b)^0x8000000000000000LL) &
0x8000000000000000LL) ? SKP_int64_MIN : (a)-(b)) :    \
                                            ((((a)^0x8000000000000000LL) & (b)  &
0x8000000000000000LL) ? SKP_int64_MAX : (a)-(b)) )

/* Saturation for positive input values */
#define SKP_POS_SAT32(a)                    ((a) > SKP_int32_MAX ? SKP_int32_MAX :
 (a))

/* Add with saturation for positive input values */
#define SKP_ADD_POS_SAT8(a, b)              ((((a)+(b)) & 0x80)                 ?
SKP_int8_MAX  : ((a)+(b)))
#define SKP_ADD_POS_SAT16(a, b)             ((((a)+(b)) & 0x8000)               ?
SKP_int16_MAX : ((a)+(b)))
#define SKP_ADD_POS_SAT32(a, b)             ((((a)+(b)) & 0x80000000)           ?
SKP_int32_MAX : ((a)+(b)))
#define SKP_ADD_POS_SAT64(a, b)             ((((a)+(b)) & 0x8000000000000000LL) ?
SKP_int64_MAX : ((a)+(b)))

#define SKP_LSHIFT8(a, shift)               ((a)<<(shift))              // shift
 >= 0, shift < 8
#define SKP_LSHIFT16(a, shift)              ((a)<<(shift))              // shift
 >= 0, shift < 16
#define SKP_LSHIFT32(a, shift)              ((a)<<(shift))              // shift
 >= 0, shift < 32
#define SKP_LSHIFT64(a, shift)              ((a)<<(shift))              // shift
 >= 0, shift < 64
#define SKP_LSHIFT(a, shift)                SKP_LSHIFT32(a, shift)      // shift
 >= 0, shift < 32

#define SKP_RSHIFT8(a, shift)               ((a)>>(shift))              // shift
 >= 0, shift < 8
```

```
#define SKP_RSHIFT16(a, shift)             ((a)>>(shift))                // shift
 >= 0, shift < 16
#define SKP_RSHIFT32(a, shift)             ((a)>>(shift))                // shift
 >= 0, shift < 32
#define SKP_RSHIFT64(a, shift)             ((a)>>(shift))                // shift
 >= 0, shift < 64
#define SKP_RSHIFT(a, shift)               SKP_RSHIFT32(a, shift)        // shift
 >= 0, shift < 32

/* saturates before shifting */
#define SKP_LSHIFT_SAT16(a, shift)         (SKP_LSHIFT16( SKP_LIMIT( (a), SKP_RSH
IFT16( SKP_int16_MIN, (shift) ),    \
                                                                          SKP_RSH
IFT16( SKP_int16_MAX, (shift) ) ), (shift) ))
#define SKP_LSHIFT_SAT32(a, shift)         (SKP_LSHIFT32( SKP_LIMIT( (a), SKP_RSH
IFT32( SKP_int32_MIN, (shift) ),    \
                                                                          SKP_RSH
IFT32( SKP_int32_MAX, (shift) ) ), (shift) ))
```

```
#define SKP_LSHIFT_ovflw(a, shift)        ((a)<<(shift))        // shift >= 0, al
lowed to overflow
#define SKP_LSHIFT_uint(a, shift)         ((a)<<(shift))        // shift >= 0
#define SKP_RSHIFT_uint(a, shift)         ((a)>>(shift))        // shift >= 0

#define SKP_ADD_LSHIFT(a, b, shift)       ((a) + SKP_LSHIFT((b), (shift)))
     // shift >= 0
#define SKP_ADD_LSHIFT32(a, b, shift)     SKP_ADD32((a), SKP_LSHIFT32((b), (shift
)))    // shift >= 0
#define SKP_ADD_LSHIFT_uint(a, b, shift)  ((a) + SKP_LSHIFT_uint((b), (shift)))
      // shift >= 0
#define SKP_ADD_RSHIFT(a, b, shift)       ((a) + SKP_RSHIFT((b), (shift)))
     // shift >= 0
#define SKP_ADD_RSHIFT32(a, b, shift)     SKP_ADD32((a), SKP_RSHIFT32((b), (shift
)))    // shift >= 0
#define SKP_ADD_RSHIFT_uint(a, b, shift)  ((a) + SKP_RSHIFT_uint((b), (shift)))
      // shift >= 0
#define SKP_SUB_LSHIFT32(a, b, shift)     SKP_SUB32((a), SKP_LSHIFT32((b), (shift
)))    // shift >= 0
#define SKP_SUB_RSHIFT32(a, b, shift)     SKP_SUB32((a), SKP_RSHIFT32((b), (shift
)))    // shift >= 0

/* Requires that shift > 0 */
#define SKP_RSHIFT_ROUND(a, shift)        ((shift) == 1 ? ((a) >> 1) + ((a) & 1)
: (((a) >> ((shift) - 1)) + 1) >> 1)
#define SKP_RSHIFT_ROUND64(a, shift)      ((shift) == 1 ? ((a) >> 1) + ((a) & 1)
: (((a) >> ((shift) - 1)) + 1) >> 1)

/* Number of rightshift required to fit the multiplication */
#define SKP_NSHIFT_MUL_32_32(a, b)        ( -(31- (32-SKP_Silk_CLZ32(SKP_abs(a))
+ (32-SKP_Silk_CLZ32(SKP_abs(b))))) )
#define SKP_NSHIFT_MUL_16_16(a, b)        ( -(15- (16-SKP_Silk_CLZ16(SKP_abs(a))
+ (16-SKP_Silk_CLZ16(SKP_abs(b))))) )


#define SKP_min(a, b)                     (((a) < (b)) ? (a) : (b))
#define SKP_max(a, b)                     (((a) > (b)) ? (a) : (b))

/* Macro to convert floating-point constants to fixed-point */
#define SKP_FIX_CONST( C, Q )             ((SKP_int32)((C) * (1 << (Q)) + 0.5))

/* SKP_min() versions with typecast in the function call */
SKP_INLINE SKP_int SKP_min_int(SKP_int a, SKP_int b)
{
    return (((a) < (b)) ? (a) : (b));
}
SKP_INLINE SKP_int16 SKP_min_16(SKP_int16 a, SKP_int16 b)
{
    return (((a) < (b)) ? (a) : (b));
}
SKP_INLINE SKP_int32 SKP_min_32(SKP_int32 a, SKP_int32 b)
{
    return (((a) < (b)) ? (a) : (b));
}
SKP_INLINE SKP_int64 SKP_min_64(SKP_int64 a, SKP_int64 b)
{
    return (((a) < (b)) ? (a) : (b));
}

/* SKP_min() versions with typecast in the function call */
SKP_INLINE SKP_int SKP_max_int(SKP_int a, SKP_int b)
```

```
{
    return (((a) > (b)) ? (a) : (b));
}
SKP_INLINE SKP_int16 SKP_max_16(SKP_int16 a, SKP_int16 b)
{
    return (((a) > (b)) ? (a) : (b));
}
SKP_INLINE SKP_int32 SKP_max_32(SKP_int32 a, SKP_int32 b)
{
    return (((a) > (b)) ? (a) : (b));
}
SKP_INLINE SKP_int64 SKP_max_64(SKP_int64 a, SKP_int64 b)
{
    return (((a) > (b)) ? (a) : (b));
}

#define SKP_LIMIT( a, limit1, limit2)    ((limit1) > (limit2) ? ((a) > (limit1) ?
 (limit1) : ((a) < (limit2) ? (limit2) : (a))) \
                                                    : ((a) > (limit2) ?
(limit2) : ((a) < (limit1) ? (limit1) : (a))))

//#define SKP_non_neg(a)                 ((a) & ((-(a)) >> (8 * sizeof(a) - 1)))
  /* doesn't seem faster than SKP_max(0, a);

#define SKP_abs(a)                       (((a) >  0)  ? (a) : -(a))           //
 Be careful, SKP_abs returns wrong when input equals to SKP_intXX_MIN
#define SKP_abs_int(a)                   (((a) ^ ((a) >> (8 * sizeof(a) - 1))) -
((a) >> (8 * sizeof(a) - 1)))
#define SKP_abs_int32(a)                 (((a) ^ ((a) >> 31)) - ((a) >> 31))
#define SKP_abs_int64(a)                 (((a) >  0)  ? (a) : -(a))

#define SKP_sign(a)                      ((a) > 0 ? 1 : ( (a) < 0 ? -1 : 0 ))

#define SKP_sqrt(a)                      (sqrt(a))

/* PSEUDO-RANDOM GENERATOR
   */
/* Make sure to store the result as the seed for the next call (also in between
   */
/* frames), otherwise result won't be random at all. When only using some of the
   */
/* bits, take the most significant bits by right-shifting. Do not just mask off
   */
/* the lowest bits.
   */
#define SKP_RAND(seed)                   (SKP_MLA_ovflw(907633515, (seed), 196314
165))

// Add some multiplication functions that can be easily mapped to ARM.

//    SKP_SMMUL: Signed top word multiply.
//        ARMv6        2 instruction cycles.
//        ARMv3M+        3 instruction cycles. use SMULL and ignore LSB registers
.(except xM)
//#define SKP_SMMUL(a32, b32)            (SKP_int32)SKP_RSHIFT(SKP_SMLAL(SKP_SMUL
WB((a32), (b32)), (a32), SKP_RSHIFT_ROUND((b32), 16)), 16)
// the following seems faster on x86
#define SKP_SMMUL(a32, b32)              (SKP_int32)SKP_RSHIFT64(SKP_SMULL((a32),
 (b32)), 32)

#include "SKP_Silk_Inlines.h"
```

```
#ifdef  __cplusplus
}
#endif

#endif
```

A.116. src/SKP_Silk_solve_LS_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/*****************************/
/* Internal function headers */
/*****************************/

typedef struct {
    SKP_int32 Q36_part;
    SKP_int32 Q48_part;
} inv_D_t;
```

```
/* Factorize square matrix A into LDL form */
SKP_INLINE void SKP_Silk_LDL_factorize_FIX(
    SKP_int32           *A,         /* I/O Pointer to Symetric Square Matrix */
    SKP_int             M,          /* I   Size of Matrix */
    SKP_int32           *L_Q16,     /* I/O Pointer to Square Upper triangular Mat
rix */
    inv_D_t             *inv_D      /* I/O Pointer to vector holding inverted dia
gonal elements of D */
);

/* Solve Lx = b, when L is lower triangular and has ones on the diagonal */
SKP_INLINE void SKP_Silk_LS_SolveFirst_FIX(
    const SKP_int32     *L_Q16,     /* I Pointer to Lower Triangular Matrix */
    SKP_int             M,          /* I Dim of Matrix equation */
    const SKP_int32     *b,         /* I b Vector */
    SKP_int32           *x_Q16      /* O x Vector */
);

/* Solve L^t*x = b, where L is lower triangular with ones on the diagonal */
SKP_INLINE void SKP_Silk_LS_SolveLast_FIX(
    const SKP_int32     *L_Q16,     /* I Pointer to Lower Triangular Matrix */
    const SKP_int       M,          /* I Dim of Matrix equation */
    const SKP_int32     *b,         /* I b Vector */
    SKP_int32           *x_Q16      /* O x Vector */
);

SKP_INLINE void SKP_Silk_LS_divide_Q16_FIX(
    SKP_int32           T[],    /* I/O Numenator vector */
    inv_D_t             *inv_D, /* I   1 / D vector     */
    SKP_int             M       /* I   dimension        */
);

/* Solves Ax = b, assuming A is symmetric */
void SKP_Silk_solve_LDL_FIX(
    SKP_int32                   *A,             /* I    Pointer to symetr
ic square matrix A         */
    SKP_int                     M,              /* I    Size of matrix
                           */
    const SKP_int32             *b,             /* I    Pointer to b vect
or                         */
    SKP_int32                   *x_Q16          /* O    Pointer to x solu
tion vector                */
)
{
    SKP_int32 L_Q16[  MAX_MATRIX_SIZE * MAX_MATRIX_SIZE ];
    SKP_int32 Y[      MAX_MATRIX_SIZE ];
    inv_D_t   inv_D[  MAX_MATRIX_SIZE ];

    SKP_assert( M <= MAX_MATRIX_SIZE );

    /***************************************************
    Factorize A by LDL such that A = L*D*L',
    where L is lower triangular with ones on diagonal
    ****************************************************/
```

```
    SKP_Silk_LDL_factorize_FIX( A, M, L_Q16, inv_D );

    /***************************************************
    * substitute D*L'*x = Y. ie:
    L*D*L'*x = b => L*Y = b <=> Y = inv(L)*b
    ***************************************************/
    SKP_Silk_LS_SolveFirst_FIX( L_Q16, M, b, Y );

    /***************************************************
    D*L'*x = Y <=> L'*x = inv(D)*Y, because D is
    diagonal just multiply with 1/d_i
    ***************************************************/
    SKP_Silk_LS_divide_Q16_FIX( Y, inv_D, M );

    /***************************************************
    x = inv(L') * inv(D) * Y
    ***************************************************/
    SKP_Silk_LS_SolveLast_FIX( L_Q16, M, Y, x_Q16 );
}

SKP_INLINE void SKP_Silk_LDL_factorize_FIX(
    SKP_int32           *A,         /* I   Pointer to Symetric Square Matrix */
    SKP_int             M,          /* I   Size of Matrix */
    SKP_int32           *L_Q16,     /* I/O Pointer to Square Upper triangular Mat
rix */
    inv_D_t             *inv_D      /* I/O Pointer to vector holding inverted dia
gonal elements of D */
)
{
    SKP_int   i, j, k, status, loop_count;
    const SKP_int32 *ptr1, *ptr2;
    SKP_int32 diag_min_value, tmp_32, err;
    SKP_int32 v_Q0[ MAX_MATRIX_SIZE ], D_Q0[ MAX_MATRIX_SIZE ];
    SKP_int32 one_div_diag_Q36, one_div_diag_Q40, one_div_diag_Q48;

    SKP_assert( M <= MAX_MATRIX_SIZE );

    status = 1;
    diag_min_value = SKP_max_32( SKP_SMMUL( SKP_ADD_SAT32( A[ 0 ], A[ SKP_SMULBB(
 M, M ) - 1 ] ), FIND_LTP_COND_FAC_Q31 ), 1 << 9 );
    for( loop_count = 0; loop_count < M && status == 1; loop_count++ ) {
        status = 0;
        for( j = 0; j < M; j++ ) {
            ptr1 = matrix_adr( L_Q16, j, 0, M );
            tmp_32 = 0;
            for( i = 0; i < j; i++ ) {
                v_Q0[ i ] = SKP_SMULWW(          D_Q0[ i ], ptr1[ i ] ); /* Q0 */
                tmp_32    = SKP_SMLAWW( tmp_32, v_Q0[ i ], ptr1[ i ] ); /* Q0 */
            }
            tmp_32 = SKP_SUB32( matrix_ptr( A, j, j, M ), tmp_32 );
```

```
            if( tmp_32 < diag_min_value ) {
                tmp_32 = SKP_SUB32( SKP_SMULBB( loop_count + 1, diag_min_value ),
 tmp_32 );
                /* Matrix not positive semi-definite, or ill conditioned */
                for( i = 0; i < M; i++ ) {
                    matrix_ptr( A, i, i, M ) = SKP_ADD32( matrix_ptr( A, i, i, M
), tmp_32 );
                }
                status = 1;
                break;
            }
            D_Q0[ j ] = tmp_32;                         /* always < max(Correlati
on) */

            /* two-step division */
            one_div_diag_Q36 = SKP_INVERSE32_varQ( tmp_32, 36 );
   /* Q36 */
            one_div_diag_Q40 = SKP_LSHIFT( one_div_diag_Q36, 4 );
   /* Q40 */
            err = SKP_SUB32( 1 << 24, SKP_SMULWW( tmp_32, one_div_diag_Q40 ) );
   /* Q24 */
            one_div_diag_Q48 = SKP_SMULWW( err, one_div_diag_Q40 );
   /* Q48 */

            /* Save 1/Ds */
            inv_D[ j ].Q36_part = one_div_diag_Q36;
            inv_D[ j ].Q48_part = one_div_diag_Q48;

            matrix_ptr( L_Q16, j, j, M ) = 65536; /* 1.0 in Q16 */
            ptr1 = matrix_adr( A, j, 0, M );
            ptr2 = matrix_adr( L_Q16, j + 1, 0, M );
            for( i = j + 1; i < M; i++ ) {
                tmp_32 = 0;
                for( k = 0; k < j; k++ ) {
                    tmp_32 = SKP_SMLAWW( tmp_32, v_Q0[ k ], ptr2[ k ] ); /* Q0 */
                }
                tmp_32 = SKP_SUB32( ptr1[ i ], tmp_32 ); /* always < max(Correlat
ion) */

                /* tmp_32 / D_Q0[j] : Divide to Q16 */
                matrix_ptr( L_Q16, i, j, M ) = SKP_ADD32( SKP_SMMUL( tmp_32, one_
div_diag_Q48 ),
                    SKP_RSHIFT( SKP_SMULWW( tmp_32, one_div_diag_Q36 ), 4 ) );

                /* go to next column */
                ptr2 += M;
            }
        }
    }

    SKP_assert( status == 0 );
}

SKP_INLINE void SKP_Silk_LS_divide_Q16_FIX(
    SKP_int32 T[],      /* I/O Numenator vector */
    inv_D_t *inv_D,     /* I   1 / D vector     */
    SKP_int M           /* I   Order */
```

```
)
{
    SKP_int   i;
    SKP_int32 tmp_32;
    SKP_int32 one_div_diag_Q36, one_div_diag_Q48;

    for( i = 0; i < M; i++ ) {
        one_div_diag_Q36 = inv_D[ i ].Q36_part;
        one_div_diag_Q48 = inv_D[ i ].Q48_part;

        tmp_32 = T[ i ];
        T[ i ] = SKP_ADD32( SKP_SMMUL( tmp_32, one_div_diag_Q48 ), SKP_RSHIFT( SK
P_SMULWW( tmp_32, one_div_diag_Q36 ), 4 ) );
    }
}

/* Solve Lx = b, when L is lower triangular and has ones on the diagonal */
SKP_INLINE void SKP_Silk_LS_SolveFirst_FIX(
    const SKP_int32     *L_Q16, /* I Pointer to Lower Triangular Matrix */
    SKP_int             M,      /* I Dim of Matrix equation */
    const SKP_int32     *b,     /* I b Vector */
    SKP_int32           *x_Q16  /* O x Vector */
)
{
    SKP_int i, j;
    const SKP_int32 *ptr32;
    SKP_int32 tmp_32;

    for( i = 0; i < M; i++ ) {
        ptr32 = matrix_adr( L_Q16, i, 0, M );
        tmp_32 = 0;
        for( j = 0; j < i; j++ ) {
            tmp_32 = SKP_SMLAWW( tmp_32, ptr32[ j ], x_Q16[ j ] );
        }
        x_Q16[ i ] = SKP_SUB32( b[ i ], tmp_32 );
    }
}

/* Solve L^t*x = b, where L is lower triangular with ones on the diagonal */
SKP_INLINE void SKP_Silk_LS_SolveLast_FIX(
    const SKP_int32     *L_Q16,     /* I Pointer to Lower Triangular Matrix */
    const SKP_int       M,          /* I Dim of Matrix equation */
    const SKP_int32     *b,         /* I b Vector */
    SKP_int32           *x_Q16      /* O x Vector */
)
{
    SKP_int i, j;
    const SKP_int32 *ptr32;
    SKP_int32 tmp_32;
```

```
    for( i = M - 1; i >= 0; i-- ) {
        ptr32 = matrix_adr( L_Q16, 0, i, M );
        tmp_32 = 0;
        for( j = M - 1; j > i; j-- ) {
            tmp_32 = SKP_SMLAWW( tmp_32, ptr32[ SKP_SMULBB( j, M ) ], x_Q16[ j ]
);
        }
        x_Q16[ i ] = SKP_SUB32( b[ i ], tmp_32 );
    }
}
```

A.117. src/SKP_Silk_sort.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/* Insertion sort (fast for already almost sorted arrays):   */
/* Best case:  O(n)   for an already sorted array            */
/* Worst case: O(n^2) for an inversely sorted array          */
/*                                                           */
/* To be implemented:                                        */
/* Shell short:    http://en.wikipedia.org/wiki/Shell_sort   */
```

```
#include "SKP_Silk_SigProc_FIX.h"

void SKP_Silk_insertion_sort_increasing(
    SKP_int32           *a,             /* I/O:  Unsorted / Sorted vector
       */
    SKP_int             *index,         /* O:    Index vector for the sorted elem
ents   */
    const SKP_int       L,              /* I:    Vector length
       */
    const SKP_int       K               /* I:    Number of correctly sorted posit
ions   */
)
{
    SKP_int32    value;
    SKP_int        i, j;

    /* Safety checks */
    SKP_assert( K >  0 );
    SKP_assert( L >  0 );
    SKP_assert( L >= K );

    /* Write start indices in index vector */
    for( i = 0; i < K; i++ ) {
        index[ i ] = i;
    }

    /* Sort vector elements by value, increasing order */
    for( i = 1; i < K; i++ ) {
        value = a[ i ];
        for( j = i - 1; ( j >= 0 ) && ( value < a[ j ] ); j-- ) {
            a[ j + 1 ]     = a[ j ];     /* Shift value */
            index[ j + 1 ] = index[ j ]; /* Shift index */
        }
        a[ j + 1 ]     = value; /* Write value */
        index[ j + 1 ] = i;     /* Write index */
    }

    /* If less than L values are asked check the remaining values,      */
    /* but only spend CPU to ensure that the K first values are correct */
    for( i = K; i < L; i++ ) {
        value = a[ i ];
        if( value < a[ K - 1 ] ) {
            for( j = K - 2; ( j >= 0 ) && ( value < a[ j ] ); j-- ) {
                a[ j + 1 ]     = a[ j ];     /* Shift value */
                index[ j + 1 ] = index[ j ]; /* Shift index */
            }
            a[ j + 1 ]     = value; /* Write value */
            index[ j + 1 ] = i;        /* Write index */
        }
    }
}
```

```
void SKP_Silk_insertion_sort_decreasing(
    SKP_int             *a,             /* I/O: Unsorted / Sorted vector
       */
    SKP_int             *index,         /* O:   Index vector for the sorted eleme
nts    */
    const SKP_int       L,              /* I:   Vector length
       */
    const SKP_int       K               /* I:   Number of correctly sorted positi
ons    */
)
{
    SKP_int    value;
    SKP_int    i, j;

    /* Safety checks */
    SKP_assert( K >  0 );
    SKP_assert( L >  0 );
    SKP_assert( L >= K );

    /* Write start indices in index vector */
    for( i = 0; i < K; i++ ) {
        index[ i ] = i;
    }

    /* Sort vector elements by value, decreasing order */
    for( i = 1; i < K; i++ ) {
        value = a[ i ];
        for( j = i - 1; ( j >= 0 ) && ( value > a[ j ] ); j-- ) {
            a[ j + 1 ]     = a[ j ];     /* Shift value */
            index[ j + 1 ] = index[ j ]; /* Shift index */
        }
        a[ j + 1 ]     = value; /* Write value */
        index[ j + 1 ] = i;     /* Write index */
    }

    /* If less than L values are asked check the remaining values,      */
    /* but only spend CPU to ensure that the K first values are correct */
    for( i = K; i < L; i++ ) {
        value = a[ i ];
        if( value > a[ K - 1 ] ) {
            for( j = K - 2; ( j >= 0 ) && ( value > a[ j ] ); j-- ) {
                a[ j + 1 ]     = a[ j ];     /* Shift value */
                index[ j + 1 ] = index[ j ]; /* Shift index */
            }
            a[ j + 1 ]     = value; /* Write value */
            index[ j + 1 ] = i;     /* Write index */
        }
    }
}

void SKP_Silk_insertion_sort_decreasing_int16(
    SKP_int16           *a,             /* I/O: Unsorted / Sorted vector
       */
```

```
    SKP_int             *index,         /* O:   Index vector for the sorted eleme
nts    */
    const SKP_int       L,              /* I:   Vector length
    */
    const SKP_int       K               /* I:   Number of correctly sorted positi
ons    */
)
{
    SKP_int i, j;
    SKP_int value;

    /* Safety checks */
    SKP_assert( K >  0 );
    SKP_assert( L >  0 );
    SKP_assert( L >= K );

    /* Write start indices in index vector */
    for( i = 0; i < K; i++ ) {
        index[ i ] = i;
    }

    /* Sort vector elements by value, decreasing order */
    for( i = 1; i < K; i++ ) {
        value = a[ i ];
        for( j = i - 1; ( j >= 0 ) && ( value > a[ j ] ); j-- ) {
            a[ j + 1 ]     = a[ j ];     /* Shift value */
            index[ j + 1 ] = index[ j ]; /* Shift index */
        }
        a[ j + 1 ]     = value; /* Write value */
        index[ j + 1 ] = i;     /* Write index */
    }

    /* If less than L values are asked check the remaining values,        */
    /* but only spend CPU to ensure that the K first values are correct */
    for( i = K; i < L; i++ ) {
        value = a[ i ];
        if( value > a[ K - 1 ] ) {
            for( j = K - 2; ( j >= 0 ) && ( value > a[ j ] ); j-- ) {
                a[ j + 1 ]     = a[ j ];     /* Shift value */
                index[ j + 1 ] = index[ j ]; /* Shift index */
            }
            a[ j + 1 ]     = value; /* Write value */
            index[ j + 1 ] = i;     /* Write index */
        }
    }
}

void SKP_Silk_insertion_sort_increasing_all_values(
    SKP_int             *a,             /* I/O: Unsorted / Sorted vector
    */
    const SKP_int       L               /* I:   Vector length
    */
)
```

```
{
    SKP_int    value;
    SKP_int    i, j;

    /* Safety checks */
    SKP_assert( L >  0 );

    /* Sort vector elements by value, increasing order */
    for( i = 1; i < L; i++ ) {
        value = a[ i ];
        for( j = i - 1; ( j >= 0 ) && ( value < a[ j ] ); j-- ) {
            a[ j + 1 ] = a[ j ]; /* Shift value */
        }
        a[ j + 1 ] = value; /* Write value */
    }
}
```

A.118. src/SKP_Silk_structs.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/
```

```
#ifndef SKP_SILK_STRUCTS_H
#define SKP_SILK_STRUCTS_H

#include "SKP_Silk_typedef.h"
#include "SKP_Silk_SigProc_FIX.h"
#include "SKP_Silk_define.h"

#ifdef __cplusplus
extern "C"
{
#endif


/*********************************/
/* Noise shaping quantization state */
/*********************************/
typedef struct {
    SKP_int16   xq[           2 * MAX_FRAME_LENGTH ]; /* Buffer for quantized out
put signal */
    SKP_int32   sLTP_shp_Q10[ 2 * MAX_FRAME_LENGTH ];
    SKP_int32   sLPC_Q14[ MAX_FRAME_LENGTH / NB_SUBFR + MAX_LPC_ORDER ];
    SKP_int32   sLF_AR_shp_Q12;
    SKP_int     lagPrev;
    SKP_int     sLTP_buf_idx;
    SKP_int     sLTP_shp_buf_idx;
    SKP_int32   rand_seed;
    SKP_int32   prev_inv_gain_Q16;
    SKP_int     rewhite_flag;
} SKP_Silk_nsq_state; /* FIX*/

/* Struct for Low BitRate Redundant (LBRR) information */
typedef struct {
    SKP_uint8   payload[ MAX_ARITHM_BYTES ];
    SKP_int     nBytes;                         /* Number of bytes in payload
                        */
    SKP_int     usage;                          /* Tells how the payload should b
e used as FEC               */
} SKP_SILK_LBRR_struct;

/********************************/
/* VAD state                    */
/********************************/
typedef struct {
    SKP_int32   AnaState[ 2 ];                  /* Analysis filterbank state: 0-8
 kHz                    */
    SKP_int32   AnaState1[ 2 ];                 /* Analysis filterbank state: 0-4
 kHz                    */
    SKP_int32   AnaState2[ 2 ];                 /* Analysis filterbank state: 0-2
 kHz                    */
    SKP_int32   XnrgSubfr[ VAD_N_BANDS ];       /* Subframe energies
                        */
    SKP_int32   NrgRatioSmth_Q8[ VAD_N_BANDS ]; /* Smoothed energy level in each
band                    */
    SKP_int16   HPstate;                        /* State of differentiator in the
 lowest band            */
    SKP_int32   NL[ VAD_N_BANDS ];              /* Noise energy level in each ban
d                       */
    SKP_int32   inv_NL[ VAD_N_BANDS ];          /* Inverse noise energy level in
each band               */
```

```
    SKP_int32   NoiseLevelBias[ VAD_N_BANDS ];  /* Noise level estimator bias/off
set                          */
    SKP_int32   counter;                        /* Frame counter used in the init
ial phase                    */
} SKP_Silk_VAD_state;

/*******************************/
/* Range encoder/decoder state */
/*******************************/
typedef struct {
    SKP_int32   bufferLength;
    SKP_int32   bufferIx;
    SKP_uint32  base_Q32;
    SKP_uint32  range_Q16;
    SKP_int32   error;
    SKP_uint8   buffer[ MAX_ARITHM_BYTES ];     /* Buffer containing payload
                         */
} SKP_Silk_range_coder_state;

/* Input frequency range detection struct */
typedef struct {
    SKP_int32                   S_HP_8_kHz[ NB_SOS ][ 2 ];  /* HP filter State */
    SKP_int32                   ConsecSmplsAboveThres;
    SKP_int32                   ActiveSpeech_ms;            /* Accumulated time w
ith active speech */
    SKP_int                     SWB_detected;               /* Flag to indicate S
WB input */
    SKP_int                     WB_detected;                /* Flag to indicate W
B input */
} SKP_Silk_detect_SWB_state;

#if SWITCH_TRANSITION_FILTERING
/* Variable cut-off low-pass filter state */
typedef struct {
    SKP_int32                   In_LP_State[ 2 ];           /* Low pass filter st
ate */
    SKP_int32                   transition_frame_no;        /* Counter which is m
apped to a cut-off frequency */
    SKP_int                     mode;                       /* Operating mode, 0:
 switch down, 1: switch up */
} SKP_Silk_LP_state;
#endif

/* Structure for one stage of MSVQ */
typedef struct {
    const SKP_int32             nVectors;
    const SKP_int16             *CB_NLSF_Q15;
    const SKP_int16             *Rates_Q5;
} SKP_Silk_NLSF_CBS;

/* Structure containing NLSF MSVQ codebook */
typedef struct {
    const SKP_int32             nStages;

    /* Fields for (de)quantizing */
    const SKP_Silk_NLSF_CBS     *CBStages;
    const SKP_int               *NDeltaMin_Q15;
```

```
    /* Fields for arithmetic (de)coding */
    const SKP_uint16            *CDF;
    const SKP_uint16 * const    *StartPtr;
    const SKP_int               *MiddleIx;
} SKP_Silk_NLSF_CB_struct;

/*******************************/
/* Encoder state               */
/*******************************/
typedef struct {
    SKP_Silk_range_coder_state      sRC;                        /* Range code
r state                                                         */
    SKP_Silk_range_coder_state      sRC_LBRR;                   /* Range code
r state (for low bitrate redundancy)                            */
#if HIGH_PASS_INPUT
    SKP_int32                       In_HP_State[ 2 ];           /* High pass
filter state                                                    */
#endif
#if SWITCH_TRANSITION_FILTERING
    SKP_Silk_LP_state               sLP;                        /* Low pass f
ilter state */
#endif
    SKP_Silk_VAD_state              sVAD;                       /* Voice acti
vity detector state                                             */

    SKP_int                         LBRRprevLastGainIndex;
    SKP_int                         prev_sigtype;
    SKP_int                         typeOffsetPrev;             /* Previous s
ignal type and quantization offset                              */
    SKP_int                         prevLag;
    SKP_int                         prev_lagIndex;
    SKP_int                         fs_kHz;                     /* Sampling f
requency (kHz)                                                  */
    SKP_int                         fs_kHz_changed;             /* Did we swi
tch yet?                                                        */
    SKP_int                         frame_length;               /* Frame leng
th (samples)                                                    */
    SKP_int                         subfr_length;               /* Subframe l
ength (samples)                                                 */
    SKP_int                         la_pitch;                   /* Look-ahead
 for pitch analysis (samples)                                   */
    SKP_int                         la_shape;                   /* Look-ahead
 for noise shape analysis (samples)                             */
    SKP_int32                       TargetRate_bps;             /* Target bit
rate (bps)                                                      */
    SKP_int                         PacketSize_ms;              /* Number of
milliseconds to put in each packet                              */
    SKP_int                         PacketLoss_perc;            /* Packet los
s rate measured by farend                                       */
    SKP_int32                       frameCounter;
    SKP_int                         Complexity;                 /* Complexity
 setting: 0-> low; 1-> medium; 2->high                          */
    SKP_int                         nStatesDelayedDecision;     /* Number of
states in delayed decision quantization                         */
    SKP_int                         useInterpolatedNLSFs;       /* Flag for u
sing NLSF interpolation                                         */
    SKP_int                         shapingLPCOrder;            /* Filter ord
er for noise shaping filters                                    */
    SKP_int                         predictLPCOrder;            /* Filter ord
er for prediction filters                                       */
    SKP_int                         pitchEstimationComplexity;  /* Complexity
 level for pitch estimator                                      */
```

```
    SKP_int                         pitchEstimationLPCOrder;        /* Whitening
filter order for pitch estimator                                    */
    SKP_int                         LTPQuantLowComplexity;          /* Flag for l
ow complexity LTP quantization                                      */
    SKP_int                         NLSF_MSVQ_Survivors;            /* Number of
survivors in NLSF MSVQ                                              */
    SKP_int                         first_frame_after_reset;        /* Flag for d
eactivating NLSF interp. and fluc. reduction after resets  */

    /* Input/output buffering */
    SKP_int16                       inputBuf[ MAX_FRAME_LENGTH ];   /* buffer con
tainin input signal                                                 */
```

```
    SKP_int                         inputBufIx;
    SKP_int                         nFramesInPayloadBuf;            /* number of
frames sitting in outputBuf                                         */
    SKP_int                         nBytesInPayloadBuf;             /* number of
bytes sitting in outputBuf                                          */

    /* Parameters For LTP scaling Control */
    SKP_int                         frames_since_onset;

    const SKP_Silk_NLSF_CB_struct   *psNLSF_CB[ 2 ];                /* Pointers t
o voiced/unvoiced NLSF codebooks */

    /* Struct for Inband LBRR */
    SKP_SILK_LBRR_struct            LBRR_buffer[ MAX_LBRR_DELAY ];
    SKP_int                         oldest_LBRR_idx;
    SKP_int                         LBRR_enabled;
    SKP_int                         LBRR_GainIncreases;             /* Number of
shifts to Gains to get LBRR rate Voiced frames              */

    /* Bitrate control */
    SKP_int32                       bitrateDiff;                    /* accumulate
d diff. between the target bitrate and the SWB/WB limits    */

#if LOW_COMPLEXITY_ONLY
    /* state for downsampling from 24 to 16 kHz in low complexity mode */
    SKP_int16                       resample24To16state[ SigProc_Resample_2_3_coa
rse_NUM_FIR_COEFS - 1 ];
#else
    SKP_int32                       resample24To16state[ 11 ];      /* state for
downsampling from 24 to 16 kHz                              */
#endif
    SKP_int32                       resample24To12state[ 6 ];       /* state for
downsampling from 24 to 12 kHz                              */
    SKP_int32                       resample24To8state[ 7 ];        /* state for
downsampling from 24 to  8 kHz                              */
    SKP_int32                       resample16To12state[ 15 ];      /* state for
downsampling from 16 to 12 kHz                              */
    SKP_int32                       resample16To8state[ 6 ];        /* state for
downsampling from 16 to  8 kHz                              */
#if LOW_COMPLEXITY_ONLY
    /* state for downsampling from 12 to 8 kHz in low complexity mode */
    SKP_int16                       resample12To8state[ SigProc_Resample_2_3_coar
se_NUM_FIR_COEFS - 1 ];
#else
    SKP_int32                       resample12To8state[ 11 ];       /* state for
downsampling from 12 to  8 kHz                              */
#endif

    /* DTX */
    SKP_int                         noSpeechCounter;                /* Counts con
cecutive nonactive frames, used by DTX                      */
    SKP_int                         useDTX;                         /* Flag to en
able DTX                                                    */
    SKP_int                         inDTX;                          /* Flag to si
gnal DTX period                                             */
    SKP_int                         vadFlag;                        /* Flag to in
dicate Voice Activity                                       */

    /* Struct for detecting SWB input */
    SKP_Silk_detect_SWB_state       sSWBdetect;
```

```
    /*****************************************/
    /* Buffers etc used by the new bitstream V4 */
    /*****************************************/
```

```
    SKP_int                         q[ MAX_FRAME_LENGTH * MAX_FRAMES_PER_PACKET ]
;       /* pulse signal buffer */
    SKP_int                         q_LBRR[ MAX_FRAME_LENGTH * MAX_FRAMES_PER_PAC
KET ]; /* pulse signal buffer */
    SKP_int                         sigtype[ MAX_FRAMES_PER_PACKET ];
    SKP_int                         QuantOffsetType[ MAX_FRAMES_PER_PACKET ];
    SKP_int                         extension_layer;
       /* Add extension layer      */
    SKP_int                         bitstream_v;
       /* Holds bitstream version  */
} SKP_Silk_encoder_state;


/************************/
/* Encoder control      */
/************************/
typedef struct {
    /* Quantization indices */
    SKP_int     lagIndex;
    SKP_int     contourIndex;
    SKP_int     PERIndex;
    SKP_int     LTPIndex[ NB_SUBFR ];
    SKP_int     NLSFIndices[ NLSF_MSVQ_MAX_CB_STAGES ];  /* NLSF path of quantize
d LSF vector   */
    SKP_int     NLSFInterpCoef_Q2;
    SKP_int     GainsIndices[ NB_SUBFR ];
    SKP_int32   Seed;
    SKP_int     LTP_scaleIndex;
    SKP_int     RateLevelIndex;
    SKP_int     QuantOffsetType;
    SKP_int     sigtype;

    /* Prediction and coding parameters */
    SKP_int     pitchL[ NB_SUBFR ];

    SKP_int     LBRR_usage;                     /* Low bitrate redundancy usage
                        */
} SKP_Silk_encoder_control;

/* Struct for Packet Loss Concealment */
typedef struct {
    SKP_int32   pitchL_Q8;                      /* Pitch lag to use for voiced co
ncealment                   */
    SKP_int16   LTPCoef_Q14[ LTP_ORDER ];       /* LTP coeficients to use for voi
ced concealment             */
    SKP_int16   prevLPC_Q12[ MAX_LPC_ORDER ];
    SKP_int     last_frame_lost;                /* Was previous frame lost
                        */
    SKP_int32   rand_seed;                      /* Seed for unvoiced signal gener
ation                       */
    SKP_int16   randScale_Q14;                  /* Scaling of unvoiced random sig
nal                         */
    SKP_int32   conc_energy;
    SKP_int     conc_energy_shift;
    SKP_int16   prevLTP_scale_Q14;
    SKP_int32   prevGain_Q16[ NB_SUBFR ];
    SKP_int     fs_kHz;
} SKP_Silk_PLC_struct;
```

```
/* Struct for CNG */
typedef struct {
    SKP_int32   CNG_exc_buf_Q10[ MAX_FRAME_LENGTH ];
    SKP_int     CNG_smth_NLSF_Q15[ MAX_LPC_ORDER ];
    SKP_int32   CNG_synth_state[ MAX_LPC_ORDER ];
    SKP_int32   CNG_smth_Gain_Q16;
    SKP_int32   rand_seed;
    SKP_int     fs_kHz;
} SKP_Silk_CNG_struct;

/*******************************/
/* Decoder state               */
/*******************************/
typedef struct {
    SKP_Silk_range_coder_state  sRC;                            /* Range coder st
ate                                                             */
    SKP_int32       prev_inv_gain_Q16;
    SKP_int32       sLTP_Q16[ 2 * MAX_FRAME_LENGTH ];
    SKP_int32       sLPC_Q14[ MAX_FRAME_LENGTH / NB_SUBFR + MAX_LPC_ORDER ];
    SKP_int32       exc_Q10[ MAX_FRAME_LENGTH ];
    SKP_int32       res_Q10[ MAX_FRAME_LENGTH ];
    SKP_int16       outBuf[ 2 * MAX_FRAME_LENGTH ];             /* Buffer for out
put signal                                                      */
    SKP_int         sLTP_buf_idx;                               /* LTP_buf_index
                                                                */
    SKP_int         lagPrev;                                    /* Previous Lag
                                                                */
    SKP_int         LastGainIndex;                              /* Previous gain
index                                                           */
    SKP_int         LastGainIndex_EnhLayer;                     /* Previous gain
index                                                           */
    SKP_int         typeOffsetPrev;                             /* Previous signa
l type and quantization offset                                  */
    SKP_int32       HPState[ DEC_HP_ORDER ];                    /* HP filter stat
e                                                               */
    const SKP_int16 *HP_A;                                      /* HP filter AR c
oefficients                                                     */
    const SKP_int16 *HP_B;                                      /* HP filter MA c
oefficients                                                     */
    SKP_int         fs_kHz;                                     /* Sampling frequ
ency in kHz                                                     */
    SKP_int         frame_length;                               /* Frame length (
samples)                                                        */
    SKP_int         subfr_length;                               /* Subframe lengt
h (samples)                                                     */
    SKP_int         LPC_order;                                  /* LPC order
                                                                */
    SKP_int         prevNLSF_Q15[ MAX_LPC_ORDER ];              /* Used to interp
olate LSFs                                                      */
    SKP_int         first_frame_after_reset;                    /* Flag for deact
ivating NLSF interp. and fluc. reduction after resets  */

    /* For buffering payload in case of more frames per packet */
    SKP_int         nBytesLeft;
    SKP_int         nFramesDecoded;
    SKP_int         nFramesInPacket;
    SKP_int         moreInternalDecoderFrames;
    SKP_int         FrameTermination;

    SKP_int32       resampleState[ 15 ];

    const SKP_Silk_NLSF_CB_struct *psNLSF_CB[ 2 ];      /* Pointers to voiced/unv
```

```
oiced NLSF codebooks */
    SKP_int         sigtype[                MAX_FRAMES_PER_PACKET ];
```

```
    SKP_int         QuantOffsetType[       MAX_FRAMES_PER_PACKET ];
    SKP_int         GainsIndices[          MAX_FRAMES_PER_PACKET ][ NB_SUBFR ];
    SKP_int         GainsIndices_EnhLayer[ MAX_FRAMES_PER_PACKET ][ NB_SUBFR ];
    SKP_int         NLSFIndices[           MAX_FRAMES_PER_PACKET ][ NLSF_MSVQ_MAX
_CB_STAGES ];
    SKP_int         NLSFInterpCoef_Q2[     MAX_FRAMES_PER_PACKET ];
    SKP_int         lagIndex[              MAX_FRAMES_PER_PACKET ];
    SKP_int         contourIndex[          MAX_FRAMES_PER_PACKET ];
    SKP_int         PERIndex[              MAX_FRAMES_PER_PACKET ];
    SKP_int         LTPIndex[              MAX_FRAMES_PER_PACKET ][ NB_SUBFR ];
    SKP_int         LTP_scaleIndex[        MAX_FRAMES_PER_PACKET ];
    SKP_int         Seed[                  MAX_FRAMES_PER_PACKET ];
    SKP_int         vadFlagBuf[            MAX_FRAMES_PER_PACKET ];

    /* Parameters used to investigate if inband FEC is used */
    SKP_int         vadFlag;
    SKP_int         no_FEC_counter;                             /* Counts number
of frames wo inband FEC                                 */
    SKP_int         inband_FEC_offset;                          /* 0: no FEC, 1:
FEC with 1 packet offset, 2: FEC w 2 packets offset     */

    /* CNG state */
    SKP_Silk_CNG_struct sCNG;

    /* Stuff used for PLC */
    SKP_Silk_PLC_struct sPLC;
    SKP_int         lossCnt;
    SKP_int         prev_sigtype;                               /* Previous sigty
pe                                                      */



    SKP_int                 bitstream_v;                        /* Holds bitstrea
m version                                               */
} SKP_Silk_decoder_state;

/************************/
/* Decoder control      */
/************************/
typedef struct {
    /* prediction and coding parameters */
    SKP_int             pitchL[ NB_SUBFR ];
    SKP_int32           Gains_Q16[ NB_SUBFR ];
    SKP_int32           Seed;
    /* holds interpolated and final coefficients, 4-byte aligned */
    SKP_array_of_int16_4_byte_aligned( PredCoef_Q12[ 2 ], MAX_LPC_ORDER );
    SKP_int16           LTPCoef_Q14[ LTP_ORDER * NB_SUBFR ];
    SKP_int             LTP_scale_Q14;

    /* quantization indices */
    SKP_int             PERIndex;
    SKP_int             RateLevelIndex;
```

```
    SKP_int             QuantOffsetType;
    SKP_int             sigtype;
    SKP_int             NLSFInterpCoef_Q2;
} SKP_Silk_decoder_control;

#ifdef __cplusplus
}
#endif

#endif
```

A.119. src/SKP_Silk_structs_FIX.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_STRUCTS_FIX_H
#define SKP_SILK_STRUCTS_FIX_H

#include "SKP_Silk_typedef.h"
#include "SKP_Silk_define_FIX.h"
#include "SKP_Silk_main.h"
```

```
#include "SKP_Silk_structs.h"

#if LOW_COMPLEXITY_ONLY
#include "SKP_Silk_resample_rom.h"
#endif

#ifdef __cplusplus
extern "C"
{
#endif

/*********************************/
/* Noise shaping analysis state */
/*********************************/
typedef struct {
    SKP_int     LastGainIndex;
    SKP_int32   HarmBoost_smth_Q16;
    SKP_int32   HarmShapeGain_smth_Q16;
    SKP_int32   Tilt_smth_Q16;
} SKP_Silk_shape_state_FIX;

/*********************************/
/* Prefilter state               */
/*********************************/
typedef struct {
    SKP_int16   sLTP_shp1[ LTP_BUF_LENGTH ];
    SKP_int32   sAR_shp2_Q14[ SHAPE_LPC_ORDER_MAX ];
    SKP_int16   sLTP_shp2_FIX[ LTP_BUF_LENGTH ];
    SKP_int     sLTP_shp_buf_idx1;
    SKP_int     sAR_shp_buf_idx2;
    SKP_int     sLTP_shp_buf_idx2;
    SKP_int32   sLF_AR_shp1_Q12;
    SKP_int32   sLF_MA_shp1_Q12;
    SKP_int32   sLF_AR_shp2_Q12;
    SKP_int32   sLF_MA_shp2_Q12;
    SKP_int     sHarmHP;
    SKP_int32   rand_seed;
    SKP_int     lagPrev;
} SKP_Silk_prefilter_state_FIX;

/*****************************/
/* Prediction analysis state */
/*****************************/
typedef struct {
    SKP_int   pitch_LPC_win_length;
    SKP_int   min_pitch_lag;                                    /* Lowest pos
sible pitch lag (samples)  */
    SKP_int   max_pitch_lag;                                    /* Highest po
ssible pitch lag (samples) */
    SKP_int   prev_NLSFq_Q15[ MAX_LPC_ORDER ];                  /* Prev. quan
t. normalized LSF vector   */
```

```
} SKP_Silk_predict_state_FIX;


/*********************************/
/* Encoder state FIX             */
/*********************************/
typedef struct {
    SKP_Silk_encoder_state          sCmn;                           /* Common str
uct, shared with floating-point code */

#if HIGH_PASS_INPUT
    SKP_int32                       variable_HP_smth1_Q15;          /* State of f
irst smoother                                                       */
    SKP_int32                       variable_HP_smth2_Q15;          /* State of s
econd smoother                                                      */
#endif
    SKP_Silk_shape_state_FIX        sShape;                         /* Shape stat
e                                                                   */
    SKP_Silk_prefilter_state_FIX    sPrefilt;                       /* Prefilter
State                                                               */
    SKP_Silk_predict_state_FIX      sPred;                          /* Prediction
 state                                                              */
    SKP_Silk_nsq_state              sNSQ;                           /* Noise Shap
e Quantizer State                                                   */
    SKP_Silk_nsq_state              sNSQ_LBRR;                      /* Noise Shap
e Quantizer State ( for low bitrate redundancy )                    */

    /* Function pointer to noise shaping quantizer (will be set to SKP_Silk_NSQ o
r SKP_Silk_NSQ_del_dec) */
    void    (* NoiseShapingQuantizer)( SKP_Silk_encoder_state *, SKP_Silk_encoder
_control *, SKP_Silk_nsq_state *, const SKP_int16 *,
                                       SKP_int *, const SKP_int, const SKP_int16
 *, const SKP_int16 *, const SKP_int16 *, const SKP_int *,
                                       const SKP_int *, const SKP_int32 *, const
 SKP_int32 *, SKP_int, const SKP_int
    );

    /* Buffer for find pitch and noise shape analysis */
    SKP_array_of_int16_4_byte_aligned( x_buf, 2 * MAX_FRAME_LENGTH + LA_SHAPE_MAX
 );
    SKP_int                         LTPCorr_Q15;                    /* Normalized
 correlation from pitch lag estimator                               */
    SKP_int                         mu_LTP_Q8;                      /* Rate-disto
rtion tradeoff in LTP quantization                                  */
    SKP_int32                       SNR_dB_Q7;                      /* Quality se
tting                                                               */
    SKP_int32                       avgGain_Q16;                    /* average ga
in during active speech                                             */
    SKP_int32                       avgGain_Q16_one_bit_per_sample; /* average ga
in during active speech                                             */
    SKP_int                         BufferedInChannel_ms;           /* Simulated
number of ms buffer because of exceeded TargetRate_bps              */
    SKP_int                         speech_activity_Q8;             /* Speech act
ivity in Q8                                                         */
    SKP_int32                       pitchEstimationThreshold_Q16;   /* Threshold
for pitch estimator                                                 */

    /* Parameters For LTP scaling Control */
    SKP_int                         prevLTPredCodGain_Q7;
    SKP_int                         HPLTPredCodGain_Q7;

    SKP_int32                       inBandFEC_SNR_comp_Q8;          /* Compensati
```

```
on to SNR_dB when using inband FEC Voiced      */

} SKP_Silk_encoder_state_FIX;

/************************/
/* Encoder control FIX  */
/************************/
typedef struct {
```

```
    SKP_Silk_encoder_control        sCmn;                           /* Common str
uct, shared with floating-point code */

    /* Prediction and coding parameters */
    SKP_int32                   Gains_Q16[ NB_SUBFR ];
    SKP_array_of_int16_4_byte_aligned( PredCoef_Q12[ 2 ], MAX_LPC_ORDER );
    SKP_int16                   LTPCoef_Q14[ LTP_ORDER * NB_SUBFR ];
    SKP_int                     LTP_scale_Q14;

    /* Noise shaping parameters */
    /* Testing */
    SKP_array_of_int16_4_byte_aligned( AR1_Q13, NB_SUBFR * SHAPE_LPC_ORDER_MAX );
    SKP_array_of_int16_4_byte_aligned( AR2_Q13, NB_SUBFR * SHAPE_LPC_ORDER_MAX );
    SKP_int32   LF_shp_Q14[        NB_SUBFR ];          /* Packs two int16 coeffi
cients per int32 value             */
    SKP_int     GainsPre_Q14[      NB_SUBFR ];
    SKP_int     HarmBoost_Q14[     NB_SUBFR ];
    SKP_int     Tilt_Q14[          NB_SUBFR ];
    SKP_int     HarmShapeGain_Q14[ NB_SUBFR ];
    SKP_int     Lambda_Q10;
    SKP_int     input_quality_Q14;
    SKP_int     coding_quality_Q14;
    SKP_int32   pitch_freq_low_Hz;
    SKP_int     current_SNR_dB_Q7;

    /* measures */
    SKP_int     sparseness_Q8;
    SKP_int     LTPredCodGain_Q7;
    SKP_int     input_quality_bands_Q15[ VAD_N_BANDS ];
    SKP_int     input_tilt_Q15;
    SKP_int32   ResNrg[ NB_SUBFR ];             /* Residual energy per subframe
                    */
    SKP_int     ResNrgQ[ NB_SUBFR ];            /* Q domain for the residual ener
gy > 0                      */

} SKP_Silk_encoder_control_FIX;


#ifdef __cplusplus
}
#endif

#endif
```

A.120. src/SKP_Silk_sum_sqr_shift.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
```

```
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*                                                                     *
 * SKP_Silk_sum_sqr_shift.c                                          *
 *                                                                     *
 * compute number of bits to right shift the sum of squares of a vector *
 * of int16s to make it fit in an int32                                *
 *                                                                     *
 * Copyright 2006-2008 (c), Skype Limited                              *
 *                                                                     */
#include "SKP_Silk_SigProc_FIX.h"
/* Compute number of bits to right shift the sum of squares of a vector */
/* of int16s to make it fit in an int32                                */
void SKP_Silk_sum_sqr_shift(
    SKP_int32           *energy,           /* O    Energy of x, after shifting
to the right           */
    SKP_int             *shift,            /* O    Number of bits right shift a
pplied to energy       */
    const SKP_int16     *x,                /* I    Input vector
                       */
    SKP_int             len                /* I    Length of input vector
                       */
)
{
    SKP_int   i, shft;
    SKP_int32 in32, nrg_tmp, nrg;

    if( (SKP_int32)( (SKP_int_ptr_size)x & 2 ) != 0 ) {
        /* Input is not 4-byte aligned */
        nrg = SKP_SMULBB( x[ 0 ], x[ 0 ] );
        i = 1;
```

```
    } else {
        nrg = 0;
        i   = 0;
    }
    shft = 0;
    len--;
    for( ; i < len; i += 2 ) {
        /* Load two values at once */
        in32 = *( (SKP_int32 *)&x[ i ] );
        nrg = SKP_SMLABB_ovflw( nrg, in32, in32 );
        nrg = SKP_SMLATT_ovflw( nrg, in32, in32 );
        if( nrg < 0 ) {
            /* Scale down */
            nrg = (SKP_int32)SKP_RSHIFT_uint( (SKP_uint32)nrg, 2 );
            shft = 2;
            break;
        }
    }
    for( ; i < len; i += 2 ) {
        /* Load two values at once */
        in32 = *( (SKP_int32 *)&x[ i ] );
        nrg_tmp = SKP_SMULBB( in32, in32 );
        nrg_tmp = SKP_SMLATT_ovflw( nrg_tmp, in32, in32 );
        nrg = (SKP_int32)SKP_ADD_RSHIFT_uint( nrg, (SKP_uint32)nrg_tmp, shft );
        if( nrg < 0 ) {
            /* Scale down */
            nrg = (SKP_int32)SKP_RSHIFT_uint( (SKP_uint32)nrg, 2 );
            shft += 2;
        }
    }
    if( i == len ) {
        /* One sample left to process */
        nrg_tmp = SKP_SMULBB( x[ i ], x[ i ] );
        nrg = (SKP_int32)SKP_ADD_RSHIFT_uint( nrg, nrg_tmp, shft );
    }

    /* Make sure to have at least one extra leading zero (two leading zeros in to
tal) */
    if( nrg & 0xC0000000 ) {
        nrg = SKP_RSHIFT_uint( (SKP_uint32)nrg, 2 );
        shft += 2;
    }

    /* Output arguments */
    *shift  = shft;
    *energy = nrg;
}
```

A.121. src/SKP_Silk_tables.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_TABLES_H
#define SKP_SILK_TABLES_H

#include "SKP_Silk_define.h"
#include "SKP_Silk_structs.h"

#define PITCH_EST_MAX_LAG_MS                18          /* 18 ms -> 56 Hz */
#define PITCH_EST_MIN_LAG_MS                2           /* 2 ms -> 500 Hz */

#ifdef __cplusplus
extern "C"
{
#endif

/* entropy coding tables */
extern const SKP_uint16 SKP_Silk_type_offset_CDF[ 5 ];
                    /*   5 */
extern const SKP_uint16 SKP_Silk_type_offset_joint_CDF[ 4 ][ 5 ];
                    /*  20 */
extern const SKP_int    SKP_Silk_type_offset_CDF_offset;
```

```
extern const SKP_uint16 SKP_Silk_gain_CDF[ 2 ][ N_LEVELS_QGAIN + 1 ];
                    /* 130 */
extern const SKP_int    SKP_Silk_gain_CDF_offset;
extern const SKP_uint16 SKP_Silk_delta_gain_CDF[ MAX_DELTA_GAIN_QUANT - MIN_DELTA
_GAIN_QUANT + 2 ]; /*  46 */
extern const SKP_int    SKP_Silk_delta_gain_CDF_offset;

extern const SKP_uint16 SKP_Silk_pitch_lag_NB_CDF[ 8 * ( PITCH_EST_MAX_LAG_MS - P
ITCH_EST_MIN_LAG_MS ) + 2 ];   /* 130 */
extern const SKP_int    SKP_Silk_pitch_lag_NB_CDF_offset;
extern const SKP_uint16 SKP_Silk_pitch_lag_MB_CDF[ 12 * ( PITCH_EST_MAX_LAG_MS -
PITCH_EST_MIN_LAG_MS ) + 2 ];  /* 194 */
extern const SKP_int    SKP_Silk_pitch_lag_MB_CDF_offset;
extern const SKP_uint16 SKP_Silk_pitch_lag_WB_CDF[ 16 * ( PITCH_EST_MAX_LAG_MS -
PITCH_EST_MIN_LAG_MS ) + 2 ];  /* 258 */
extern const SKP_int    SKP_Silk_pitch_lag_WB_CDF_offset;
extern const SKP_uint16 SKP_Silk_pitch_lag_SWB_CDF[ 24 * ( PITCH_EST_MAX_LAG_MS -
 PITCH_EST_MIN_LAG_MS ) + 2 ]; /* 386 */
extern const SKP_int    SKP_Silk_pitch_lag_SWB_CDF_offset;

extern const SKP_uint16 SKP_Silk_pitch_contour_CDF[ 35 ];
                    /*  35 */
extern const SKP_int    SKP_Silk_pitch_contour_CDF_offset;
extern const SKP_uint16 SKP_Silk_pitch_contour_NB_CDF[ 12 ];
                    /*  12 */
extern const SKP_int    SKP_Silk_pitch_contour_NB_CDF_offset;
extern const SKP_uint16 SKP_Silk_pitch_delta_CDF[23];
                    /* 23 */
extern const SKP_int    SKP_Silk_pitch_delta_CDF_offset;

extern const SKP_uint16 SKP_Silk_pulses_per_block_CDF[ N_RATE_LEVELS ][ MAX_PULSE
S + 3 ];            /* 210 */
extern const SKP_int    SKP_Silk_pulses_per_block_CDF_offset;
extern const SKP_int16  SKP_Silk_pulses_per_block_BITS_Q6[ N_RATE_LEVELS - 1 ][ M
AX_PULSES + 2 ];   /* 180 */

extern const SKP_uint16 SKP_Silk_rate_levels_CDF[ 2 ][ N_RATE_LEVELS ];
                    /*  20 */
extern const SKP_int    SKP_Silk_rate_levels_CDF_offset;
extern const SKP_int16  SKP_Silk_rate_levels_BITS_Q6[ 2 ][ N_RATE_LEVELS - 1 ];
                    /*  18 */

extern const SKP_int    SKP_Silk_max_pulses_table[ 4 ];
                    /*   4 */

extern const SKP_uint16 SKP_Silk_shell_code_table0[  33 ];
                    /*  33 */
extern const SKP_uint16 SKP_Silk_shell_code_table1[  52 ];
                    /*  52 */
extern const SKP_uint16 SKP_Silk_shell_code_table2[ 102 ];
                    /* 102 */
extern const SKP_uint16 SKP_Silk_shell_code_table3[ 207 ];
                    /* 207 */
extern const SKP_uint16 SKP_Silk_shell_code_table_offsets[ 19 ];
                    /*  19 */

extern const SKP_uint16 SKP_Silk_lsb_CDF[ 3 ];
                    /*   3 */

extern const SKP_uint16 SKP_Silk_sign_CDF[ 36 ][ 3 ];
                    /* 108 */
```

```
extern const SKP_uint16 SKP_Silk_LTP_per_index_CDF[ 4 ];
                    /*   4 */
extern const SKP_int    SKP_Silk_LTP_per_index_CDF_offset;
extern const SKP_int16  * const SKP_Silk_LTP_gain_BITS_Q6_ptrs[ NB_LTP_CBKS ];
                    /*   3 */
extern const SKP_uint16 * const SKP_Silk_LTP_gain_CDF_ptrs[ NB_LTP_CBKS ];
                    /*   3 */
extern const SKP_int    SKP_Silk_LTP_gain_CDF_offsets[ NB_LTP_CBKS ];
                    /*   3 */
extern const SKP_int32  SKP_Silk_LTP_gain_middle_avg_RD_Q14;
extern const SKP_uint16 SKP_Silk_LTPscale_CDF[ 4 ];
                    /*   4 */
```

```
extern const SKP_int    SKP_Silk_LTPscale_offset;

/* Tables for LTPScale */
extern const SKP_int16  SKP_Silk_LTPScales_table_Q14[ 3 ];

extern const SKP_uint16 SKP_Silk_vadflag_CDF[ 3 ];
                    /*   3 */
extern const SKP_int    SKP_Silk_vadflag_offset;

extern const SKP_int    SKP_Silk_SamplingRates_table[ 4 ];
                    /*   4 */
extern const SKP_uint16 SKP_Silk_SamplingRates_CDF[ 5 ];
                    /*   5 */
extern const SKP_int    SKP_Silk_SamplingRates_offset;

extern const SKP_uint16 SKP_Silk_NLSF_interpolation_factor_CDF[ 6 ];
extern const SKP_int    SKP_Silk_NLSF_interpolation_factor_offset;

/* NLSF codebooks */
extern const SKP_Silk_NLSF_CB_struct SKP_Silk_NLSF_CB0_16, SKP_Silk_NLSF_CB1_16;
extern const SKP_Silk_NLSF_CB_struct SKP_Silk_NLSF_CB0_10, SKP_Silk_NLSF_CB1_10;

/* quantization tables */
extern const SKP_int16 * const SKP_Silk_LTP_vq_ptrs_Q14[ NB_LTP_CBKS ];
                    /* 168 */
extern const SKP_int    SKP_Silk_LTP_vq_sizes[ NB_LTP_CBKS ];
                    /*   3 */

/* Piece-wise linear mapping from bitrate in kbps to coding quality in dB SNR */
extern const SKP_int32  TargetRate_table_NB[  TARGET_RATE_TAB_SZ ];
extern const SKP_int32  TargetRate_table_MB[  TARGET_RATE_TAB_SZ ];
extern const SKP_int32  TargetRate_table_WB[  TARGET_RATE_TAB_SZ ];
extern const SKP_int32  TargetRate_table_SWB[ TARGET_RATE_TAB_SZ ];
extern const SKP_int32  SNR_table_Q1[         TARGET_RATE_TAB_SZ ];

extern const SKP_int32  SNR_table_one_bit_per_sample_Q7[ 4 ];

/* Filter coeficicnts for HP filter: 4. Order filter implementad as two biquad fi
lters  */
extern const SKP_int16  SKP_Silk_SWB_detect_B_HP_Q13[ NB_SOS ][ 3 ];
extern const SKP_int16  SKP_Silk_SWB_detect_A_HP_Q13[ NB_SOS ][ 2 ];

/* Decoder high-pass filter coefficients for 24 kHz sampling */
extern const SKP_int16  SKP_Silk_Dec_A_HP_24[ DEC_HP_ORDER ];
                    /*   2 */
extern const SKP_int16  SKP_Silk_Dec_B_HP_24[ DEC_HP_ORDER + 1 ];
                    /*   3 */

/* Decoder high-pass filter coefficients for 16 kHz sampling */
extern const SKP_int16  SKP_Silk_Dec_A_HP_16[ DEC_HP_ORDER ];
                    /*   2 */
extern const SKP_int16  SKP_Silk_Dec_B_HP_16[ DEC_HP_ORDER + 1 ];
                    /*   3 */

/* Decoder high-pass filter coefficients for 12 kHz sampling */
extern const SKP_int16  SKP_Silk_Dec_A_HP_12[ DEC_HP_ORDER ];
                    /*   2 */
extern const SKP_int16  SKP_Silk_Dec_B_HP_12[ DEC_HP_ORDER + 1 ];
                    /*   3 */
```

```
/* Decoder high-pass filter coefficients for 8 kHz sampling */
extern const SKP_int16  SKP_Silk_Dec_A_HP_8[ DEC_HP_ORDER ];
                    /*   2 */
extern const SKP_int16  SKP_Silk_Dec_B_HP_8[ DEC_HP_ORDER + 1 ];
                    /*   3 */

/* Table for frame termination indication */
extern const SKP_uint16 SKP_Silk_FrameTermination_CDF[ 5 ];
extern const SKP_int    SKP_Silk_FrameTermination_offset;

extern const SKP_uint16 SKP_Silk_FrameTermination_v4_CDF[ 6 ];
extern const SKP_int    SKP_Silk_FrameTermination_v4_offset;

/* Table for random seed */
extern const SKP_uint16 SKP_Silk_Seed_CDF[ 5 ];
extern const SKP_int    SKP_Silk_Seed_offset;

/* Quantization offsets */
extern const SKP_int16  SKP_Silk_Quantization_Offsets_Q10[ 2 ][ 2 ];

#if SWITCH_TRANSITION_FILTERING
/* Interpolation points for filter coefficients used in the bandwidth transition
smoother */
extern const SKP_int32 SKP_Silk_Transition_LP_B_Q28[ TRANSITION_INT_NUM ][ TRANSI
TION_NB ];
extern const SKP_int32 SKP_Silk_Transition_LP_A_Q28[ TRANSITION_INT_NUM ][ TRANSI
TION_NA ];
#endif

#ifdef __cplusplus
}
#endif

#endif
```

A.122. src/SKP_Silk_tables_gain.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
```

```
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_tables.h"

#ifdef __cplusplus
extern "C"
{
#endif

const SKP_uint16 SKP_Silk_gain_CDF[ 2 ][ 65 ] =
{
{
        0,     18,     45,     94,    181,    320,    519,    777,
     1093,   1468,   1909,   2417,   2997,   3657,   4404,   5245,
     6185,   7228,   8384,   9664,  11069,  12596,  14244,  16022,
    17937,  19979,  22121,  24345,  26646,  29021,  31454,  33927,
    36438,  38982,  41538,  44068,  46532,  48904,  51160,  53265,
    55184,  56904,  58422,  59739,  60858,  61793,  62568,  63210,
    63738,  64165,  64504,  64769,  64976,  65133,  65249,  65330,
    65386,  65424,  65451,  65471,  65487,  65501,  65513,  65524,
    65535
},
{
        0,    214,    581,   1261,   2376,   3920,   5742,   7632,
     9449,  11157,  12780,  14352,  15897,  17427,  18949,  20462,
    21957,  23430,  24889,  26342,  27780,  29191,  30575,  31952,
    33345,  34763,  36200,  37642,  39083,  40519,  41930,  43291,
    44602,  45885,  47154,  48402,  49619,  50805,  51959,  53069,
    54127,  55140,  56128,  57101,  58056,  58979,  59859,  60692,
    61468,  62177,  62812,  63368,  63845,  64242,  64563,  64818,
    65023,  65184,  65306,  65391,  65447,  65482,  65505,  65521,
    65535
}
};

const SKP_int SKP_Silk_gain_CDF_offset = 32;
```

```
const SKP_uint16 SKP_Silk_delta_gain_CDF[ 46 ] = {
        0,   2358,   3856,   7023,  15376,  53058,  59135,  61555,
    62784,  63498,  63949,  64265,  64478,  64647,  64783,  64894,
    64986,  65052,  65113,  65169,  65213,  65252,  65284,  65314,
    65338,  65359,  65377,  65392,  65403,  65415,  65424,  65432,
    65440,  65448,  65455,  65462,  65470,  65477,  65484,  65491,
    65499,  65506,  65513,  65521,  65528,  65535
};

const SKP_int SKP_Silk_delta_gain_CDF_offset = 5;

#ifdef __cplusplus
}
#endif
```

A.123. src/SKP_Silk_tables_LTP.c


```
#include "SKP_Silk_tables.h"
```

```
const SKP_uint16 SKP_Silk_LTP_per_index_CDF[ 4 ] = {
        0,  20992,  40788,  65535
};

const SKP_int SKP_Silk_LTP_per_index_CDF_offset = 1;


const SKP_uint16 SKP_Silk_LTP_gain_CDF_0[ 11 ] = {
        0,  49380,  54463,  56494,  58437,  60101,  61683,  62985,
    64066,  64823,  65535
};

const SKP_uint16 SKP_Silk_LTP_gain_CDF_1[ 21 ] = {
        0,  25290,  30654,  35710,  40386,  42937,  45250,  47459,
    49411,  51348,  52974,  54517,  55976,  57423,  58865,  60285,
    61667,  62895,  63827,  64724,  65535
};

const SKP_uint16 SKP_Silk_LTP_gain_CDF_2[ 41 ] = {
        0,   4958,   9439,  13581,  17638,  21651,  25015,  28025,
    30287,  32406,  34330,  36240,  38130,  39790,  41281,  42764,
    44229,  45676,  47081,  48431,  49675,  50849,  51932,  52966,
    53957,  54936,  55869,  56789,  57708,  58504,  59285,  60043,
    60796,  61542,  62218,  62871,  63483,  64076,  64583,  65062,
    65535
};

const SKP_int SKP_Silk_LTP_gain_CDF_offsets[ 3 ] = {
        1,     3,     10
};

const SKP_int32 SKP_Silk_LTP_gain_middle_avg_RD_Q14 = 11010;

const SKP_int16 SKP_Silk_LTP_gain_BITS_Q6_0[ 10 ] = {
       26,    236,    321,    325,    339,    344,    362,    379,
      412,    418
};

const SKP_int16 SKP_Silk_LTP_gain_BITS_Q6_1[ 20 ] = {
       88,    231,    237,    244,    300,    309,    313,    324,
      325,    341,    346,    351,    352,    352,    354,    356,
      367,    393,    396,    406
};

const SKP_int16 SKP_Silk_LTP_gain_BITS_Q6_2[ 40 ] = {
      238,    248,    255,    257,    258,    274,    284,    311,
      317,    326,    326,    327,    339,    349,    350,    351,
      352,    355,    358,    366,    371,    379,    383,    387,
```

```
       388,    393,    394,    394,    407,    409,    412,    412,
       413,    422,    426,    432,    434,    449,    454,    455
};

const SKP_uint16 * const SKP_Silk_LTP_gain_CDF_ptrs[ NB_LTP_CBKS ] = {
    SKP_Silk_LTP_gain_CDF_0,
    SKP_Silk_LTP_gain_CDF_1,
    SKP_Silk_LTP_gain_CDF_2
};

const SKP_int16 * const SKP_Silk_LTP_gain_BITS_Q6_ptrs[ NB_LTP_CBKS ] = {
    SKP_Silk_LTP_gain_BITS_Q6_0,
    SKP_Silk_LTP_gain_BITS_Q6_1,
    SKP_Silk_LTP_gain_BITS_Q6_2
};

const SKP_int16 SKP_Silk_LTP_gain_vq_0_Q14[ 10 ][ 5 ] =
{
{
       594,    984,   2840,   1021,    669
},
{
        10,     35,    304,     -1,     23
},
{
      -694,   1923,   4603,   2975,   2335
},
{
      2437,   3176,   3778,   1940,    481
},
{
       214,    -46,   7870,   4406,   -521
},
{
      -896,   4818,   8501,   1623,   -887
},
{
      -696,   3178,   6480,   -302,   1081
},
{
       517,    599,   1002,    567,    560
},
{
     -2075,   -834,   4712,   -340,    896
},
{
      1435,   -644,   3993,   -612,  -2063
}
```

```
};

const SKP_int16 SKP_Silk_LTP_gain_vq_1_Q14[ 20 ][ 5 ] =
{
{
      1655,   2918,   5001,   3010,   1775
},
{
       113,    198,    856,    176,    178
},
{
      -843,   2479,   7858,   5371,    574
},
{
        59,   5356,   7648,   2850,   -315
},
{
      3840,   4851,   6527,   1583,  -1233
},
{
      1620,   1760,   2330,   1876,   2045
},
{
      -545,   1854,  11792,   1547,   -307
},
{
      -604,    689,   5369,   5074,   4265
},
{
       521,  -1331,   9829,   6209,  -1211
},
{
     -1315,   6747,   9929,  -1410,    546
},
{
       117,   -144,   2810,   1649,   5240
},
{
      5392,   3476,   2425,    -38,    633
},
{
        14,   -449,   5274,   3547,   -171
},
{
       -98,    395,   9114,   1676,    844
},
{
      -908,   3843,   8861,   -957,   1474
```

```
},
{
       396,   6747,   5379,   -329,   1269
},
{
      -335,   2830,   4281,    270,    -54
},
{
      1502,   5609,   8958,   6045,   2059
},
{
      -370,    479,   5267,   5726,   1174
},
{
      5237,  -1144,   6510,    455,    512
}
};

const SKP_int16 SKP_Silk_LTP_gain_vq_2_Q14[ 40 ][ 5 ] =
{
{
      -278,    415,   9345,   7106,   -431
},
{
     -1006,   3863,   9524,   4724,   -871
},
{
      -954,   4624,  11722,    973,   -300
},
{
      -117,   7066,   8331,   1959,   -901
},
{
       593,   3412,   6070,   4914,   1567
},
{
        54,    -51,  12618,   4228,   -844
},
{
      3157,   4822,   5229,   2313,    717
},
{
      -244,   1161,  14198,    779,     69
},
{
     -1218,   5603,  12894,  -2301,   1001
},
{
```

```
       -132,   3960,   9526,    577,   1806
},
{
      -1633,   8815,  10484,  -2452,    895
},
{
        235,    450,   1243,    667,    437
},
{
        959,  -2630,  10897,   8772,  -1852
},
{
       2420,   2046,   8893,   4427,  -1569
},
{
         23,   7091,   8356,  -1285,   1508
},
{
      -1133,    835,   7662,   6043,   2800
},
{
        439,    391,  11016,   2253,   1362
},
{
      -1020,   2876,  13436,   4015,  -3020
},
{
       1060,  -2690,  13512,   5565,  -1394
},
{
      -1420,   8007,  11421,   -152,  -1672
},
{
       -893,   2895,  15434,  -1490,    159
},
{
      -1054,    428,  12208,   8538,  -3344
},
{
       1772,  -1304,   7593,   6185,    561
},
{
        525,  -1207,   6659,  11151,  -1170
},
{
        439,   2667,   4743,   2359,   5515
},
{
```

```
      2951,   7432,   7909,   -230,  -1564
},
{
       -72,   2140,   5477,   1391,   1580
},
{
       476,  -1312,  15912,   2174,  -1027
},
{
      5737,    441,   2493,   2043,   2757
},
{
       228,    -43,   1803,   6663,   7064
},
{
      4596,   9182,   1917,   -200,    203
},
{
      -704,  12039,   5451,  -1188,    542
},
{
      1782,  -1040,  10078,   7513,  -2767
},
{
     -2626,   7747,   9019,     62,   1710
},
{
       235,   -233,   2954,  10921,   1947
},
{
     10854,   2814,   1232,   -111,    222
},
{
      2267,   2778,  12325,    156,  -1658
},
{
     -2950,   8095,  16330,    268,  -3626
},
{
        67,   2083,   7950,    -80,  -2432
},
{
       518,    -66,   1718,    415,  11435
}
};

const SKP_int16 * const SKP_Silk_LTP_vq_ptrs_Q14[ NB_LTP_CBKS ] = {
    &SKP_Silk_LTP_gain_vq_0_Q14[ 0 ][ 0 ],
```

```
    &SKP_Silk_LTP_gain_vq_1_Q14[ 0 ][ 0 ],
    &SKP_Silk_LTP_gain_vq_2_Q14[ 0 ][ 0 ]
};

const SKP_int SKP_Silk_LTP_vq_sizes[ NB_LTP_CBKS ] = {
    10, 20, 40
};
```

A.124. src/SKP_Silk_tables_NLSF_CB0_10.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*********************************************/
/* This file has been automatically generated */
/*                                           */
/* ROM usage:   138+1331 Words               */
/*********************************************/

#include "SKP_Silk_structs.h"
#include "SKP_Silk_tables_NLSF_CB0_10.h"
```

```
const SKP_uint16 SKP_Silk_NLSF_MSVQ_CB0_10_CDF[ NLSF_MSVQ_CB0_10_VECTORS + NLSF_M
SVQ_CB0_10_STAGES ] =
{
            0,
         2658,
         4420,
         6107,
         7757,
         9408,
        10955,
        12502,
        13983,
        15432,
        16882,
        18331,
        19750,
        21108,
        22409,
        23709,
        25010,
        26256,
        27501,
        28747,
        29965,
        31158,
        32351,
        33544,
        34736,
        35904,
        36997,
        38091,
        39185,
        40232,
        41280,
        42327,
        43308,
        44290,
        45271,
        46232,
        47192,
        48132,
        49032,
        49913,
        50775,
        51618,
        52462,
        53287,
        54095,
        54885,
```

```
        55675,
        56449,
        57222,
        57979,
        58688,
        59382,
        60076,
        60726,
        61363,
        61946,
        62505,
        63052,
        63543,
        63983,
        64396,
        64766,
        65023,
        65279,
        65535,
            0,
         4977,
         9542,
        14106,
        18671,
        23041,
        27319,
        31596,
        35873,
        39969,
        43891,
        47813,
        51652,
        55490,
        59009,
        62307,
        65535,
            0,
         8571,
        17142,
        25529,
        33917,
        42124,
        49984,
        57844,
        65535,
            0,
         8732,
        17463,
```

```
        25825,
        34007,
        42189,
        50196,
        58032,
        65535,
            0,
         8948,
        17704,
        25733,
        33762,
        41791,
        49821,
        57678,
        65535,
            0,
         4374,
         8655,
        12936,
        17125,
        21313,
        25413,
        29512,
        33611,
        37710,
        41809,
        45820,
        49832,
        53843,
        57768,
        61694,
        65535
};

const SKP_uint16 * const SKP_Silk_NLSF_MSVQ_CB0_10_CDF_start_ptr[ NLSF_MSVQ_CB0_1
0_STAGES ] =
{
     &SKP_Silk_NLSF_MSVQ_CB0_10_CDF[   0 ],
     &SKP_Silk_NLSF_MSVQ_CB0_10_CDF[  65 ],
     &SKP_Silk_NLSF_MSVQ_CB0_10_CDF[  82 ],
     &SKP_Silk_NLSF_MSVQ_CB0_10_CDF[  91 ],
     &SKP_Silk_NLSF_MSVQ_CB0_10_CDF[ 100 ],
     &SKP_Silk_NLSF_MSVQ_CB0_10_CDF[ 109 ]
};

const SKP_int SKP_Silk_NLSF_MSVQ_CB0_10_CDF_middle_idx[ NLSF_MSVQ_CB0_10_STAGES ]
 =
{
      23,
       8,
```

```
        5,
        5,
        5,
        9
};

const SKP_int16 SKP_Silk_NLSF_MSVQ_CB0_10_rates_Q5[ NLSF_MSVQ_CB0_10_VECTORS ] =
{
                148,            167,
                169,            170,
                170,            173,
                173,            175,
                176,            176,
                176,            177,
                179,            181,
                181,            181,
                183,            183,
                183,            184,
                185,            185,
                185,            185,
                186,            189,
                189,            189,
                191,            191,
                191,            194,
                194,            194,
                195,            195,
                196,            198,
                199,            200,
                201,            201,
                202,            203,
                204,            204,
                205,            205,
                206,            209,
                210,            210,
                213,            214,
                218,            220,
                221,            226,
                231,            234,
                239,            256,
                256,            256,
                119,            123,
                123,            123,
                125,            126,
                126,            126,
                128,            130,
                130,            131,
                131,            135,
                138,            139,
```

```
                94,              94,
                95,              95,
                96,              98,
                98,              99,
                93,              93,
                95,              96,
                96,              97,
                98,             100,
                92,              93,
                97,              97,
                97,              97,
                98,              98,
               125,             126,
               126,             127,
               127,             128,
               128,             128,
               128,             128,
               129,             129,
               129,             130,
               130,             131
};

const SKP_int SKP_Silk_NLSF_MSVQ_CB0_10_ndelta_min_Q15[ 10 + 1 ] =
{
               563,
                 3,
                22,
                20,
                 3,
                 3,
               132,
               119,
               358,
                86,
               964
};

const SKP_int16 SKP_Silk_NLSF_MSVQ_CB0_10_Q15[ 10 * NLSF_MSVQ_CB0_10_VECTORS ] =
{
             2210,            4023,
             6981,            9260,
            12573,           15687,
            19207,           22383,
            25981,           29142,
             3285,            4172,
             6116,           10856,
            15289,           16826,
            19701,           22010,
```

```
            24721,          29313,
             1554,           2511,
             6577,          10337,
            13837,          16511,
            20086,          23214,
            26480,          29464,
             3062,           4017,
             5771,          10037,
            13365,          14952,
            20140,          22891,
            25229,          29603,
             2085,           3457,
             5934,           8718,
            11501,          13670,
            17997,          21817,
            24935,          28745,
             2776,           4093,
             6421,          10413,
            15111,          16806,
            20825,          23826,
            26308,          29411,
             2717,           4034,
             5697,           8463,
            14301,          16354,
            19007,          23413,
            25812,          28506,
             2872,           3702,
             5881,          11034,
            17141,          18879,
            21146,          23451,
            25817,          29600,
             2999,           4015,
             7357,          11219,
            12866,          17307,
            20081,          22644,
            26774,          29107,
             2942,           3866,
             5918,          11915,
            13909,          16072,
            20453,          22279,
            27310,          29826,
             2271,           3527,
             6606,           9729,
            12943,          17382,
            20224,          22345,
            24602,          28290,
             2207,           3310,
             5844,           9339,
```

```
            11141,          15651,
            18576,          21177,
            25551,          28228,
             3963,           4975,
             6901,          11588,
            13466,          15577,
            19231,          21368,
            25510,          27759,
             2749,           3549,
             6966,          13808,
            15653,          17645,
            20090,          22599,
            26467,          28537,
             2126,           3504,
             5109,           9954,
            12550,          14620,
            19703,          21687,
            26457,          29106,
             3966,           5745,
             7442,           9757,
            14468,          16404,
            19135,          23048,
            25375,          28391,
             3197,           4751,
             6451,           9298,
            13038,          14874,
            17962,          20627,
            23835,          28464,
             3195,           4081,
             6499,          12252,
            14289,          16040,
            18357,          20730,
            26980,          29309,
             1533,           2471,
             4486,           7796,
            12332,          15758,
            19567,          22298,
            25673,          29051,
             2002,           2971,
             4985,           8083,
            13181,          15435,
            18237,          21517,
            24595,          28351,
             3808,           4925,
             6710,          10201,
            12011,          14300,
            18457,          20391,
            26525,          28956,
```

```
 2281,  3418,
 4979,  8726,
15964, 18104,
20250, 22771,
25286, 28954,
 3051,  5479,
 7290,  9848,
12744, 14503,
18665, 23684,
26065, 28947,
 2364,  3565,
 5502,  9621,
14922, 16621,
19005, 20996,
26310, 29302,
 4093,  5212,
 6833,  9880,
16303, 18286,
20571, 23614,
26067, 29128,
 2941,  3996,
 6038, 10638,
12668, 14451,
16798, 19392,
26051, 28517,
 3863,  5212,
 7019,  9468,
11039, 13214,
19942, 22344,
25126, 29539,
 4615,  6172,
 7853, 10252,
12611, 14445,
19719, 22441,
24922, 29341,
 3566,  4512,
 6985,  8684,
10544, 16097,
18058, 22475,
26066, 28167,
 4481,  5489,
 7432, 11414,
13191, 15225,
20161, 22258,
26484, 29716,
 3320,  4320,
 6621,  9867,
11581, 14034,
```

```
            21168,          23210,
            26588,          29903,
             3794,           4689,
             6916,           8655,
            10143,          16144,
            19568,          21588,
            27557,          29593,
             2446,           3276,
             5918,          12643,
            16601,          18013,
            21126,          23175,
            27300,          29634,
             2450,           3522,
             5437,           8560,
            15285,          19911,
            21826,          24097,
            26567,          29078,
             2580,           3796,
             5580,           8338,
             9969,          12675,
            18907,          22753,
            25450,          29292,
             3325,           4312,
             6241,           7709,
             9164,          14452,
            21665,          23797,
            27096,          29857,
             3338,           4163,
             7738,          11114,
            12668,          14753,
            16931,          22736,
            25671,          28093,
             3840,           4755,
             7755,          13471,
            15338,          17180,
            20077,          22353,
            27181,          29743,
             2504,           4079,
             8351,          12118,
            15046,          18595,
            21684,          24704,
            27519,          29937,
             5234,           6342,
             8267,          11821,
            15155,          16760,
            20667,          23488,
            25949,          29307,
             2681,           3562,
```

```
 6028,  10827,
18458,  20458,
22303,  24701,
26912,  29956,
 3374,   4528,
 6230,   8256,
 9513,  12730,
18666,  20720,
26007,  28425,
 2731,   3629,
 8320,  12450,
14112,  16431,
18548,  22098,
25329,  27718,
 3481,   4401,
 7321,   9319,
11062,  13093,
15121,  22315,
26331,  28740,
 3577,   4945,
 6669,   8792,
10299,  12645,
19505,  24766,
26996,  29634,
 4058,   5060,
 7288,  10190,
11724,  13936,
15849,  18539,
26701,  29845,
 4262,   5390,
 7057,   8982,
10187,  15264,
20480,  22340,
25958,  28072,
 3404,   4329,
 6629,   7946,
10121,  17165,
19640,  22244,
25062,  27472,
 3157,   4168,
 6195,   9319,
10771,  13325,
15416,  19816,
24672,  27634,
 2503,   3473,
 5130,   6767,
 8571,  14902,
19033,  21926,
```

```
            26065,          28728,
             4133,           5102,
             7553,          10054,
            11757,          14924,
            17435,          20186,
            23987,          26272,
             4972,           6139,
             7894,           9633,
            11320,          14295,
            21737,          24306,
            26919,          29907,
             2958,           3816,
             6851,           9204,
            10895,          18052,
            20791,          23338,
            27556,          29609,
             5234,           6028,
             8034,          10154,
            11242,          14789,
            18948,          20966,
            26585,          29127,
             5241,           6838,
            10526,          12819,
            14681,          17328,
            19928,          22336,
            26193,          28697,
             3412,           4251,
             5988,           7094,
             9907,          18243,
            21669,          23777,
            26969,          29087,
             2470,           3217,
             7797,          15296,
            17365,          19135,
            21979,          24256,
            27322,          29442,
             4939,           5804,
             8145,          11809,
            13873,          15598,
            17234,          19423,
            26476,          29645,
             5051,           6167,
             8223,           9655,
            12159,          17995,
            20464,          22832,
            26616,          28462,
             4987,           5907,
             9319,          11245,
```

```
            13132,          15024,
            17485,          22687,
            26011,          28273,
             5137,           6884,
            11025,          14950,
            17191,          19425,
            21807,          24393,
            26938,          29288,
             7057,           7884,
             9528,          10483,
            10960,          14811,
            19070,          21675,
            25645,          28019,
             6759,           7160,
             8546,          11779,
            12295,          13023,
            16627,          21099,
            24697,          28287,
             3863,           9762,
            11068,          11445,
            12049,          13960,
            18085,          21507,
            25224,          28997,
              397,            335,
              651,           1168,
              640,            765,
              465,            331,
              214,           -194,
             -578,           -647,
             -657,            750,
              564,            613,
              549,            630,
              304,            -52,
              828,            922,
              443,            111,
              138,            124,
              169,             14,
              144,             83,
              132,             58,
             -413,           -752,
              869,            336,
              385,             69,
               56,            830,
             -227,           -266,
             -368,           -440,
            -1195,            163,
              126,           -228,
              802,            156,
```

```
   188,    120,
   376,     59,
  -358,   -558,
 -1326,   -254,
  -202,   -789,
   296,     92,
   -70,   -129,
  -718,  -1135,
   292,    -29,
  -631,    487,
  -157,   -153,
  -279,      2,
  -419,   -342,
   -34,   -514,
  -799,  -1571,
  -687,   -609,
  -546,   -130,
  -215,   -252,
  -446,   -574,
 -1337,    207,
   -72,     32,
   103,   -642,
   942,    733,
   187,     29,
  -211,   -814,
   143,    225,
    20,     24,
  -268,   -377,
  1623,   1133,
   667,    164,
   307,    366,
   187,     34,
    62,   -313,
  -832,  -1482,
 -1181,    483,
   -42,    -39,
  -450,  -1406,
  -587,    -52,
  -760,    334,
    98,    -60,
  -500,   -488,
 -1058,    299,
   131,   -250,
  -251,   -703,
  1037,    568,
  -413,   -265,
  1687,    573,
   345,    323,
```

```
   98,    61,
 -102,    31,
  135,   149,
  617,   365,
  -39,    34,
 -611,  1201,
 1421,   736,
 -414,  -393,
 -492,  -343,
 -316,  -532,
  528,   172,
   90,   322,
 -294,  -319,
 -541,   503,
  639,   401,
    1,  -149,
  -73,  -167,
  150,   118,
  308,   218,
  121,   195,
 -143,  -261,
-1013,  -802,
  387,   436,
  130,  -427,
 -448,  -681,
  123,   -87,
 -251,  -113,
  274,   310,
  445,   501,
  354,   272,
  141,  -285,
  569,   656,
   37,   -49,
  251,  -386,
 -263,  1122,
  604,   606,
  336,    95,
   34,     0,
   85,   180,
  207,  -367,
 -622,  1070,
   -6,   -79,
 -160,   -92,
 -137,  -276,
 -323,  -371,
 -696, -1036,
  407,   102,
  -86,  -214,
```

```
  -482,   -647,
   -28,   -291,
   -97,   -180,
  -250,   -435,
   -18,    -76,
  -332,    410,
   407,    168,
   539,    411,
   254,    111,
    58,   -145,
   200,     30,
   187,    116,
   131,   -367,
  -475,    781,
  -559,    561,
   195,   -115,
     8,   -168,
    30,     55,
  -122,    131,
    82,     -5,
  -273,    -50,
  -632,    668,
     4,     32,
   -26,   -279,
   315,    165,
   197,    377,
   155,    -41,
  -138,   -324,
  -109,   -617,
   360,     98,
   -53,   -319,
  -114,   -245,
   -82,    507,
   468,    263,
  -137,   -389,
   652,    354,
   -18,   -227,
  -462,   -135,
   317,     53,
   -16,     66,
   -72,   -126,
  -356,   -347,
  -328,    -72,
  -337,    324,
   152,    349,
   169,   -196,
   179,    254,
   260,    325,
```

```
   -74,    -80,
    75,    -31,
   270,    275,
    87,    278,
  -446,   -301,
   309,     71,
   -25,   -242,
   516,    161,
  -162,    -83,
   329,    230,
  -311,   -259,
   177,    -26,
  -462,     89,
   257,      6,
  -130,    -93,
  -456,   -317,
  -221,   -206,
  -417,   -182,
   -74,    234,
    48,    261,
   359,    231,
   258,     85,
  -282,    252,
  -147,   -222,
   251,   -207,
   443,    123,
  -417,    -36,
   273,   -241,
   240,   -112,
    44,   -167,
   126,   -124,
   -77,     58,
  -401,    333,
  -118,     82,
   126,    151,
  -433,    359,
  -130,   -102,
   131,   -244,
    86,     85,
  -462,    414,
  -240,     16,
   145,     28,
  -205,   -481,
   373,    293,
   -72,   -174,
    62,    259,
    -8,    -18,
   362,    233,
```

```
 185,  43,
 278,  27,
 193, 570,
-248, 189,
  92,  31,
-275,  -3,
 243, 176,
 438, 209,
 206, -51,
  79, 109,
 168,-185,
-308, -68,
-618, 385,
-310,-108,
-164, 165,
  61,-152,
-101,-412,
-268,-257,
 -40, -20,
 -28,-158,
-301, 271,
 380,-338,
-367,-132,
  64, 114,
-131,-225,
-156,-260,
 -63,-116,
 155,-586,
-202, 254,
-287, 178,
 227,-106,
-294, 164,
 298,-100,
 185, 317,
 193, -45,
  28,  80,
 -87,-433,
  22, -48,
  48,-237,
-229,-139,
 120,-364,
 268,-136,
 396, 125,
 130, -89,
-272, 118,
-256, -68,
-451, 488,
 143,-165,
```

```
                -48,             -190,
                106,              219,
                 47,              435,
                245,               97,
                 75,             -418,
                121,             -187,
                570,             -200,
               -351,              225,
                -21,             -217,
                234,             -111,
                194,               14,
                242,              118,
                140,             -397,
                355,              361,
                -45,             -195
};

const SKP_Silk_NLSF_CBS SKP_Silk_NLSF_CB0_10_Stage_info[ NLSF_MSVQ_CB0_10_STAGES
] =
{
        {  64, &SKP_Silk_NLSF_MSVQ_CB0_10_Q15[ 10 *   0 ], &SKP_Silk_NLSF_MSVQ_CB
0_10_rates_Q5[   0 ] },
        {  16, &SKP_Silk_NLSF_MSVQ_CB0_10_Q15[ 10 *  64 ], &SKP_Silk_NLSF_MSVQ_CB
0_10_rates_Q5[  64 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB0_10_Q15[ 10 *  80 ], &SKP_Silk_NLSF_MSVQ_CB
0_10_rates_Q5[  80 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB0_10_Q15[ 10 *  88 ], &SKP_Silk_NLSF_MSVQ_CB
0_10_rates_Q5[  88 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB0_10_Q15[ 10 *  96 ], &SKP_Silk_NLSF_MSVQ_CB
0_10_rates_Q5[  96 ] },
        {  16, &SKP_Silk_NLSF_MSVQ_CB0_10_Q15[ 10 * 104 ], &SKP_Silk_NLSF_MSVQ_CB
0_10_rates_Q5[ 104 ] }
};

const SKP_Silk_NLSF_CB_struct SKP_Silk_NLSF_CB0_10 =
{
        NLSF_MSVQ_CB0_10_STAGES,
        SKP_Silk_NLSF_CB0_10_Stage_info,
        SKP_Silk_NLSF_MSVQ_CB0_10_ndelta_min_Q15,
        SKP_Silk_NLSF_MSVQ_CB0_10_CDF,
        SKP_Silk_NLSF_MSVQ_CB0_10_CDF_start_ptr,
        SKP_Silk_NLSF_MSVQ_CB0_10_CDF_middle_idx
};
```

A.125. src/SKP_Silk_tables_NLSF_CB0_10.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
```

```
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_TABLES_NLSF_CB0_10_H
#define SKP_SILK_TABLES_NLSF_CB0_10_H

#include "SKP_Silk_define.h"

#ifdef __cplusplus
extern "C"
{
#endif

#define NLSF_MSVQ_CB0_10_STAGES       6
#define NLSF_MSVQ_CB0_10_VECTORS      120

/* NLSF codebook entropy coding tables */
extern const SKP_uint16         SKP_Silk_NLSF_MSVQ_CB0_10_CDF[ NLSF_MSVQ_CB0_10_V
ECTORS + NLSF_MSVQ_CB0_10_STAGES ];
extern const SKP_uint16 * const SKP_Silk_NLSF_MSVQ_CB0_10_CDF_start_ptr[
         NLSF_MSVQ_CB0_10_STAGES ];
extern const SKP_int            SKP_Silk_NLSF_MSVQ_CB0_10_CDF_middle_idx[
         NLSF_MSVQ_CB0_10_STAGES ];

#ifdef __cplusplus
}
#endif

#endif
```

A.126. src/SKP_Silk_tables_NLSF_CB0_16.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*******************************************/
/* This file has been automatically generated */
/*                                         */
/* ROM usage:   246+3689 Words             */
/*******************************************/

#include "SKP_Silk_structs.h"
#include "SKP_Silk_tables_NLSF_CB0_16.h"

const SKP_uint16 SKP_Silk_NLSF_MSVQ_CB0_16_CDF[ NLSF_MSVQ_CB0_16_VECTORS + NLSF_M
SVQ_CB0_16_STAGES ] =
{
           0,
        1449,
        2749,
        4022,
        5267,
        6434,
        7600,
        8647,
        9695,
       10742,
```

```
        11681,
        12601,
        13444,
        14251,
        15008,
        15764,
        16521,
        17261,
        18002,
        18710,
        19419,
        20128,
        20837,
        21531,
        22225,
        22919,
        23598,
        24277,
        24956,
        25620,
        26256,
        26865,
        27475,
        28071,
        28667,
        29263,
        29859,
        30443,
        31026,
        31597,
        32168,
        32727,
        33273,
        33808,
        34332,
        34855,
        35379,
        35902,
        36415,
        36927,
        37439,
        37941,
        38442,
        38932,
        39423,
        39914,
        40404,
        40884,
```

```
        41364,
        41844,
        42324,
        42805,
        43285,
        43754,
        44224,
        44694,
        45164,
        45623,
        46083,
        46543,
        46993,
        47443,
        47892,
        48333,
        48773,
        49213,
        49653,
        50084,
        50515,
        50946,
        51377,
        51798,
        52211,
        52614,
        53018,
        53422,
        53817,
        54212,
        54607,
        55002,
        55388,
        55775,
        56162,
        56548,
        56910,
        57273,
        57635,
        57997,
        58352,
        58698,
        59038,
        59370,
        59702,
        60014,
        60325,
        60630,
```

```
        60934,
        61239,
        61537,
        61822,
        62084,
        62346,
        62602,
        62837,
        63072,
        63302,
        63517,
        63732,
        63939,
        64145,
        64342,
        64528,
        64701,
        64867,
        65023,
        65151,
        65279,
        65407,
        65535,
            0,
         5099,
         9982,
        14760,
        19538,
        24213,
        28595,
        32976,
        36994,
        41012,
        44944,
        48791,
        52557,
        56009,
        59388,
        62694,
        65535,
            0,
         9955,
        19697,
        28825,
        36842,
        44686,
        52198,
        58939,
```

```
        65535,
            0,
         8949,
        17335,
        25720,
        33926,
        41957,
        49987,
        57845,
        65535,
            0,
         9724,
        18642,
        26998,
        35355,
        43532,
        51534,
        59365,
        65535,
            0,
         8750,
        17499,
        26249,
        34448,
        42471,
        50494,
        58178,
        65535,
            0,
         8730,
        17273,
        25816,
        34176,
        42536,
        50203,
        57869,
        65535,
            0,
         8769,
        17538,
        26307,
        34525,
        42742,
        50784,
        58319,
        65535,
            0,
         8736,
```

```
        17101,
        25466,
        33653,
        41839,
        50025,
        57864,
        65535,
            0,
         4368,
         8735,
        12918,
        17100,
        21283,
        25465,
        29558,
        33651,
        37744,
        41836,
        45929,
        50022,
        54027,
        57947,
        61782,
        65535
};

const SKP_uint16 * const SKP_Silk_NLSF_MSVQ_CB0_16_CDF_start_ptr[ NLSF_MSVQ_CB0_1
6_STAGES ] =
{
     &SKP_Silk_NLSF_MSVQ_CB0_16_CDF[   0 ],
     &SKP_Silk_NLSF_MSVQ_CB0_16_CDF[ 129 ],
     &SKP_Silk_NLSF_MSVQ_CB0_16_CDF[ 146 ],
     &SKP_Silk_NLSF_MSVQ_CB0_16_CDF[ 155 ],
     &SKP_Silk_NLSF_MSVQ_CB0_16_CDF[ 164 ],
     &SKP_Silk_NLSF_MSVQ_CB0_16_CDF[ 173 ],
     &SKP_Silk_NLSF_MSVQ_CB0_16_CDF[ 182 ],
     &SKP_Silk_NLSF_MSVQ_CB0_16_CDF[ 191 ],
     &SKP_Silk_NLSF_MSVQ_CB0_16_CDF[ 200 ],
     &SKP_Silk_NLSF_MSVQ_CB0_16_CDF[ 209 ]
};

const SKP_int SKP_Silk_NLSF_MSVQ_CB0_16_CDF_middle_idx[ NLSF_MSVQ_CB0_16_STAGES ]
 =
{
      42,
       8,
       4,
       5,
       5,
       5,
```

```
        5,
        5,
        5,
        9
};

const SKP_int16 SKP_Silk_NLSF_MSVQ_CB0_16_rates_Q5[ NLSF_MSVQ_CB0_16_VECTORS ] =
{
                176,            181,
                182,            183,
                186,            186,
                191,            191,
                191,            196,
                197,            201,
                203,            206,
                206,            206,
                207,            207,
                209,            209,
                209,            209,
                210,            210,
                210,            211,
                211,            211,
                212,            214,
                216,            216,
                217,            217,
                217,            217,
                218,            218,
                219,            219,
                220,            221,
                222,            223,
                223,            223,
                223,            224,
                224,            224,
                225,            225,
                226,            226,
                226,            226,
                227,            227,
                227,            227,
                227,            227,
                228,            228,
                228,            228,
                229,            229,
                229,            230,
                230,            230,
                231,            231,
                231,            231,
                232,            232,
                232,            232,
```

```
            233,            234,
            235,            235,
            235,            236,
            236,            236,
            236,            237,
            237,            237,
            237,            240,
            240,            240,
            240,            241,
            242,            243,
            244,            244,
            247,            247,
            248,            248,
            248,            249,
            251,            255,
            255,            256,
            260,            260,
            261,            264,
            264,            266,
            266,            268,
            271,            274,
            276,            279,
            288,            288,
            288,            288,
            118,            120,
            121,            121,
            122,            125,
            125,            129,
            129,            130,
            131,            132,
            136,            137,
            138,            145,
             87,             88,
             91,             97,
             98,            100,
            105,            106,
             92,             95,
             95,             96,
             97,             97,
             98,             99,
             88,             92,
             95,             95,
             96,             97,
             98,            109,
             93,             93,
             93,             96,
             97,             97,
             99,            101,
```

```
                93,              94,
                94,              95,
                95,              99,
                99,              99,
                93,              93,
                93,              96,
                96,              97,
               100,             102,
                93,              95,
                95,              96,
                96,              96,
                98,              99,
               125,             125,
               127,             127,
               127,             127,
               128,             128,
               128,             128,
               128,             128,
               129,             130,
               131,             132
};

const SKP_int SKP_Silk_NLSF_MSVQ_CB0_16_ndelta_min_Q15[ 16 + 1 ] =
{
               266,
                 3,
                40,
                 3,
                 3,
                16,
                78,
                89,
               107,
               141,
               188,
               146,
               272,
               240,
               235,
               215,
               632
};

const SKP_int16 SKP_Silk_NLSF_MSVQ_CB0_16_Q15[ 16 * NLSF_MSVQ_CB0_16_VECTORS ] =
{
             1170,            2278,            3658,            5374,
             7666,            9113,           11298,           13304,
            15371,           17549,           19587,           21487,
```

```
 23798,  26038,  28318,  30201,
  1628,   2334,   4115,   6036,
  7818,   9544,  11777,  14021,
 15787,  17408,  19466,  21261,
 22886,  24565,  26714,  28059,
  1724,   2670,   4056,   6532,
  8357,  10119,  12093,  14061,
 16491,  18795,  20417,  22402,
 24251,  26224,  28410,  29956,
  1493,   3427,   4789,   6399,
  8435,  10168,  12000,  14066,
 16229,  18210,  20040,  22098,
 24153,  26095,  28183,  30121,
  1119,   2089,   4295,   6245,
  8691,  10741,  12688,  15057,
 17028,  18792,  20717,  22514,
 24497,  26548,  28619,  30630,
  1363,   2417,   3927,   5556,
  7422,   9315,  11879,  13767,
 16143,  18520,  20458,  22578,
 24539,  26436,  28318,  30318,
  1122,   2503,   5216,   7148,
  9310,  11078,  13175,  14800,
 16864,  18700,  20436,  22488,
 24572,  26602,  28555,  30426,
   600,   1317,   2970,   5609,
  7694,   9784,  12169,  14087,
 16379,  18378,  20551,  22686,
 24739,  26697,  28646,  30355,
   941,   1882,   4274,   5540,
  8482,   9858,  11940,  14287,
 16091,  18501,  20326,  22612,
 24711,  26638,  28814,  30430,
   635,   1699,   4376,   5948,
  8097,  10115,  12274,  14178,
 16111,  17813,  19695,  21773,
 23927,  25866,  28022,  30134,
  1408,   2222,   3524,   5615,
  7345,   8849,  10989,  12772,
 15352,  17026,  18919,  21062,
 23329,  25215,  27209,  29023,
   701,   1307,   3548,   6301,
  7744,   9574,  11227,  12978,
 15170,  17565,  19775,  22097,
 24230,  26335,  28377,  30231,
  1752,   2364,   4879,   6569,
  7813,   9796,  11199,  14290,
 15795,  18000,  20396,  22417,
```

```
  24308,  26124,  28360,  30633,
    901,   1629,   3356,   4635,
   7256,   8767,   9971,  11558,
  15215,  17544,  19523,  21852,
  23900,  25978,  28133,  30184,
    981,   1669,   3323,   4693,
   6213,   8692,  10614,  12956,
  15211,  17711,  19856,  22122,
  24344,  26592,  28723,  30481,
   1607,   2577,   4220,   5512,
   8532,  10388,  11627,  13671,
  15752,  17199,  19840,  21859,
  23494,  25786,  28091,  30131,
    811,   1471,   3144,   5041,
   7430,   9389,  11174,  13255,
  15157,  16741,  19583,  22167,
  24115,  26142,  28383,  30395,
   1543,   2144,   3629,   6347,
   7333,   9339,  10710,  13596,
  15099,  17340,  20102,  21886,
  23732,  25637,  27818,  29917,
    492,   1185,   2940,   5488,
   7095,   8751,  11596,  13579,
  16045,  18015,  20178,  22127,
  24265,  26406,  28484,  30357,
   1547,   2282,   3693,   6341,
   7758,   9607,  11848,  13236,
  16564,  18069,  19759,  21404,
  24110,  26606,  28786,  30655,
    685,   1338,   3409,   5262,
   6950,   9222,  11414,  14523,
  16337,  17893,  19436,  21298,
  23293,  25181,  27973,  30520,
    887,   1581,   3057,   4318,
   7192,   8617,  10047,  13106,
  16265,  17893,  20233,  22350,
  24379,  26384,  28314,  30189,
   2285,   3745,   5662,   7576,
   9323,  11320,  13239,  15191,
  17175,  19225,  21108,  22972,
  24821,  26655,  28561,  30460,
   1496,   2108,   3448,   6898,
   8328,   9656,  11252,  12823,
  14979,  16482,  18180,  20085,
  22962,  25160,  27705,  29629,
    575,   1261,   3861,   6627,
   8294,  10809,  12705,  14768,
  17076,  19047,  20978,  23055,
```

```
        24972,      26703,      28720,      30345,
         1682,       2213,       3882,       6238,
         7208,       9646,      10877,      13431,
        14805,      16213,      17941,      20873,
        23550,      25765,      27756,      29461,
          888,       1616,       3924,       5195,
         7206,       8647,       9842,      11473,
        16067,      18221,      20343,      22774,
        24503,      26412,      28054,      29731,
          805,       1454,       2683,       4472,
         7936,       9360,      11398,      14345,
        16205,      17832,      19453,      21646,
        23899,      25928,      28387,      30463,
         1640,       2383,       3484,       5082,
         6032,       8606,      11640,      12966,
        15842,      17368,      19346,      21182,
        23638,      25889,      28368,      30299,
         1632,       2204,       4510,       7580,
         8718,      10512,      11962,      14096,
        15640,      17194,      19143,      22247,
        24563,      26561,      28604,      30509,
         2043,       2612,       3985,       6851,
         8038,       9514,      10979,      12789,
        15426,      16728,      18899,      20277,
        22902,      26209,      28711,      30618,
         2224,       2798,       4465,       5320,
         7108,       9436,      10986,      13222,
        14599,      18317,      20141,      21843,
        23601,      25700,      28184,      30582,
          835,       1541,       4083,       5769,
         7386,       9399,      10971,      12456,
        15021,      18642,      20843,      23100,
        25292,      26966,      28952,      30422,
         1795,       2343,       4809,       5896,
         7178,       8545,      10223,      13370,
        14606,      16469,      18273,      20736,
        23645,      26257,      28224,      30390,
         1734,       2254,       4031,       5188,
         6506,       7872,       9651,      13025,
        14419,      17305,      19495,      22190,
        24403,      26302,      28195,      30177,
         1841,       2349,       3968,       4764,
         6376,       9825,      11048,      13345,
        14682,      16252,      18183,      21363,
        23918,      26156,      28031,      29935,
         1432,       2047,       5631,       6927,
         8198,       9675,      11358,      13506,
        14802,      16419,      18339,      22019,
```

```
 24124,  26177,  28130,  30586,
  1730,   2320,   3744,   4808,
  6007,   9666,  10997,  13622,
 15234,  17495,  20088,  22002,
 23603,  25400,  27379,  29254,
  1267,   1915,   5483,   6812,
  8229,   9919,  11589,  13337,
 14747,  17965,  20552,  22167,
 24519,  26819,  28883,  30642,
  1526,   2229,   4240,   7388,
  8953,  10450,  11899,  13718,
 16861,  18323,  20379,  22672,
 24797,  26906,  28906,  30622,
  2175,   2791,   4104,   6875,
  8612,   9798,  12152,  13536,
 15623,  17682,  19213,  21060,
 24382,  26760,  28633,  30248,
   454,   1231,   4339,   5738,
  7550,   9006,  10320,  13525,
 16005,  17849,  20071,  21992,
 23949,  26043,  28245,  30175,
  2250,   2791,   4230,   5283,
  6762,  10607,  11879,  13821,
 15797,  17264,  20029,  22266,
 24588,  26437,  28244,  30419,
  1696,   2216,   4308,   8385,
  9766,  11030,  12556,  14099,
 16322,  17640,  19166,  20590,
 23967,  26858,  28798,  30562,
  2452,   3236,   4369,   6118,
  7156,   9003,  11509,  12796,
 15749,  17291,  19491,  22241,
 24530,  26474,  28273,  30073,
  1811,   2541,   3555,   5480,
  9123,  10527,  11894,  13659,
 15262,  16899,  19366,  21069,
 22694,  24314,  27256,  29983,
  1553,   2246,   4559,   5500,
  6754,   7874,  11739,  13571,
 15188,  17879,  20281,  22510,
 24614,  26649,  28786,  30755,
  1982,   2768,   3834,   5964,
  8732,   9908,  11797,  14813,
 16311,  17946,  21097,  22851,
 24456,  26304,  28166,  29755,
  1824,   2529,   3817,   5449,
  6854,   8714,  10381,  12286,
 14194,  15774,  19524,  21374,
```

```
 23695,  26069,  28096,  30212,
  2212,   2854,   3947,   5898,
  9930,  11556,  12854,  14788,
 16328,  17700,  20321,  22098,
 23672,  25291,  26976,  28586,
  2023,   2599,   4024,   4916,
  6613,  11149,  12457,  14626,
 16320,  17822,  19673,  21172,
 23115,  26051,  28825,  30758,
  1628,   2206,   3467,   4364,
  8679,  10173,  11864,  13679,
 14998,  16938,  19207,  21364,
 23850,  26115,  28124,  30273,
  2014,   2603,   4114,   7254,
  8516,  10043,  11822,  13503,
 16329,  17826,  19697,  21280,
 23151,  24661,  26807,  30161,
  2376,   2980,   4422,   5770,
  7016,   9723,  11125,  13516,
 15485,  16985,  19160,  20587,
 24401,  27180,  29046,  30647,
  2454,   3502,   4624,   6019,
  7632,   8849,  10792,  13964,
 15523,  17085,  19611,  21238,
 22856,  25108,  28106,  29890,
  1573,   2274,   3308,   5999,
  8977,  10104,  12457,  14258,
 15749,  18180,  19974,  21253,
 23045,  25058,  27741,  30315,
  1943,   2730,   4140,   6160,
  7491,   8986,  11309,  12775,
 14820,  16558,  17909,  19757,
 21512,  23605,  27274,  29527,
  2021,   2582,   4494,   5835,
  6993,   8245,   9827,  14733,
 16462,  17894,  19647,  21083,
 23764,  26667,  29072,  30990,
  1052,   1775,   3218,   4378,
  7666,   9403,  11248,  13327,
 14972,  17962,  20758,  22354,
 25071,  27209,  29001,  30609,
  2218,   2866,   4223,   5352,
  6581,   9980,  11587,  13121,
 15193,  16583,  18386,  20080,
 22013,  25317,  28127,  29880,
  2146,   2840,   4397,   5840,
  7449,   8721,  10512,  11936,
 13595,  17253,  19310,  20891,
```

```
 23417,  25627,  27749,  30231,
  1972,   2619,   3756,   6367,
  7641,   8814,  12286,  13768,
 15309,  18036,  19557,  20904,
 22582,  24876,  27800,  30440,
  2005,   2577,   4272,   7373,
  8558,  10223,  11770,  13402,
 16502,  18000,  19645,  21104,
 22990,  26806,  29505,  30942,
  1153,   1822,   3724,   5443,
  6990,   8702,  10289,  11899,
 13856,  15315,  17601,  21064,
 23692,  26083,  28586,  30639,
  1304,   1869,   3318,   7195,
  9613,  10733,  12393,  13728,
 15822,  17474,  18882,  20692,
 23114,  25540,  27684,  29244,
  2093,   2691,   4018,   6658,
  7947,   9147,  10497,  11881,
 15888,  17821,  19333,  21233,
 23371,  25234,  27553,  29998,
   575,   1331,   5304,   6910,
  8425,  10086,  11577,  13498,
 16444,  18527,  20565,  22847,
 24914,  26692,  28759,  30157,
  1435,   2024,   3283,   4156,
  7611,  10592,  12049,  13927,
 15459,  18413,  20495,  22270,
 24222,  26093,  28065,  30099,
  1632,   2168,   5540,   7478,
  8630,  10391,  11644,  14321,
 15741,  17357,  18756,  20434,
 22799,  26060,  28542,  30696,
  1407,   2245,   3405,   5639,
  9419,  10685,  12104,  13495,
 15535,  18357,  19996,  21689,
 24351,  26550,  28853,  30564,
  1675,   2226,   4005,   8223,
  9975,  11155,  12822,  14316,
 16504,  18137,  19574,  21050,
 22759,  24912,  28296,  30634,
  1080,   1614,   3622,   7565,
  8748,  10303,  11713,  13848,
 15633,  17434,  19761,  21825,
 23571,  25393,  27406,  29063,
  1693,   2229,   3456,   4354,
  5670,  10890,  12563,  14167,
 15879,  17377,  19817,  21971,
```

```
 24094,  26131,  28298,  30099,
  2042,   2959,   4195,   5740,
  7106,   8267,  11126,  14973,
 16914,  18295,  20532,  21982,
 23711,  25769,  27609,  29351,
   984,   1612,   3808,   5265,
  6885,   8411,   9547,  10889,
 12522,  16520,  19549,  21639,
 23746,  26058,  28310,  30374,
  2036,   2538,   4166,   7761,
  9146,  10412,  12144,  13609,
 15588,  17169,  18559,  20113,
 21820,  24313,  28029,  30612,
  1871,   2355,   4061,   5143,
  7464,  10129,  11941,  15001,
 16680,  18354,  19957,  22279,
 24861,  26872,  28988,  30615,
  2566,   3161,   4643,   6227,
  7406,   9970,  11618,  13416,
 15889,  17364,  19121,  20817,
 22592,  24720,  28733,  31082,
  1700,   2327,   4828,   5939,
  7567,   9154,  11087,  12771,
 14209,  16121,  20222,  22671,
 24648,  26656,  28696,  30745,
  3169,   3873,   5046,   6868,
  8184,   9480,  12335,  14068,
 15774,  17971,  20231,  21711,
 23520,  25245,  27026,  28730,
  1564,   2391,   4229,   6730,
  8905,  10459,  13026,  15033,
 17265,  19809,  21849,  23741,
 25490,  27312,  29061,  30527,
  2864,   3559,   4719,   6441,
  9592,  11055,  12763,  14784,
 16428,  18164,  20486,  22262,
 24183,  26263,  28383,  30224,
  2673,   3449,   4581,   5983,
  6863,   8311,  12464,  13911,
 15738,  17791,  19416,  21182,
 24025,  26561,  28723,  30440,
  2419,   3049,   4274,   6384,
  8564,   9661,  11288,  12676,
 14447,  17578,  19816,  21231,
 23099,  25270,  26899,  28926,
  1278,   2001,   3000,   5353,
  9995,  11777,  13018,  14570,
 16050,  17762,  19982,  21617,
```

```
 23371,  25083,  27656,  30172,
   932,   1624,   2798,   4570,
  8592,   9988,  11552,  13050,
 16921,  18677,  20415,  22810,
 24817,  26819,  28804,  30385,
  2324,   2973,   4156,   5702,
  6919,   8806,  10259,  12503,
 15015,  16567,  19418,  21375,
 22943,  24550,  27024,  29849,
  1564,   2373,   3455,   4907,
  5975,   7436,  11786,  14505,
 16107,  18148,  20019,  21653,
 23740,  25814,  28578,  30372,
  3025,   3729,   4866,   6520,
  9487,  10943,  12358,  14258,
 16174,  17501,  19476,  21408,
 23227,  24906,  27347,  29407,
  1270,   1965,   6802,   7995,
  9204,  10828,  12507,  14230,
 15759,  17860,  20369,  22502,
 24633,  26514,  28535,  30525,
  2210,   2749,   4266,   7487,
  9878,  11018,  12823,  14431,
 16247,  18626,  20450,  22054,
 23739,  25291,  27074,  29169,
  1275,   1926,   4330,   6573,
  8441,  10920,  13260,  15008,
 16927,  18573,  20644,  22217,
 23983,  25474,  27372,  28645,
  3015,   3670,   5086,   6372,
  7888,   9309,  10966,  12642,
 14495,  16172,  18080,  19972,
 22454,  24899,  27362,  29975,
  2882,   3733,   5113,   6482,
  8125,   9685,  11598,  13288,
 15405,  17192,  20178,  22426,
 24801,  27014,  29212,  30811,
  2300,   2968,   4101,   5442,
  6327,   7910,  12455,  13862,
 15747,  17505,  19053,  20679,
 22615,  24658,  27499,  30065,
  2257,   2940,   4430,   5991,
  7042,   8364,   9414,  11224,
 15723,  17420,  19253,  21469,
 23915,  26053,  28430,  30384,
  1227,   2045,   3818,   5011,
  6990,   9231,  11024,  13011,
 17341,  19017,  20583,  22799,
```

```
    25195,  26876,  29351,  30805,
     1354,   1924,   3789,   8077,
    10453,  11639,  13352,  14817,
    16743,  18189,  20095,  22014,
    24593,  26677,  28647,  30256,
     3142,   4049,   6197,   7417,
     8753,  10156,  11533,  13181,
    15947,  17655,  19606,  21402,
    23487,  25659,  28123,  30304,
     1317,   2263,   4725,   7611,
     9667,  11634,  14143,  16258,
    18724,  20698,  22379,  24007,
    25775,  27251,  28930,  30593,
     1570,   2323,   3818,   6215,
     9893,  11556,  13070,  14631,
    16152,  18290,  21386,  23346,
    25114,  26923,  28712,  30168,
     2297,   3905,   6287,   8558,
    10668,  12766,  15019,  17102,
    19036,  20677,  22341,  23871,
    25478,  27085,  28851,  30520,
     1915,   2507,   4033,   5749,
     7059,   8871,  10659,  12198,
    13937,  15383,  16869,  18707,
    23175,  25818,  28514,  30501,
     2404,   2918,   5190,   6252,
     7426,   9887,  12387,  14795,
    16754,  18368,  20338,  22003,
    24236,  26456,  28490,  30397,
     1621,   2227,   3479,   5085,
     9425,  12892,  14246,  15652,
    17205,  18674,  20446,  22209,
    23778,  25867,  27931,  30093,
     1869,   2390,   4105,   7021,
    11221,  12775,  14059,  15590,
    17024,  18608,  20595,  22075,
    23649,  25154,  26914,  28671,
     2551,   3252,   4688,   6562,
     7869,   9125,  10475,  11800,
    15402,  18780,  20992,  22555,
    24289,  25968,  27465,  29232,
     2705,   3493,   4735,   6360,
     7905,   9352,  11538,  13430,
    15239,  16919,  18619,  20094,
    21800,  23342,  25200,  29257,
     2166,   2791,   4011,   5081,
     5896,   9038,  13407,  14703,
    16543,  18189,  19896,  21857,
```

```
 24872,  26971,  28955,  30514,
  1865,   3021,   4696,   6534,
  8343,   9914,  12789,  14103,
 16533,  17729,  21340,  22439,
 24873,  26330,  28428,  30154,
  3369,   4345,   6573,   8763,
 10309,  11713,  13367,  14784,
 16483,  18145,  19839,  21247,
 23292,  25477,  27555,  29447,
  1265,   2184,   5443,   7893,
 10591,  13139,  15105,  16639,
 18402,  19826,  21419,  22995,
 24719,  26437,  28363,  30125,
  1584,   2004,   3535,   4450,
  8662,  10764,  12832,  14978,
 16972,  18794,  20932,  22547,
 24636,  26521,  28701,  30567,
  3419,   4528,   6602,   7890,
  9508,  10875,  12771,  14357,
 16051,  18330,  20630,  22490,
 25070,  26936,  28946,  30542,
  1726,   2252,   4597,   6950,
  8379,   9823,  11363,  12794,
 14306,  15476,  16798,  18018,
 21671,  25550,  28148,  30367,
  3385,   3870,   5307,   6388,
  7141,   8684,  12695,  14939,
 16480,  18277,  20537,  22048,
 23947,  25965,  28214,  29956,
  2771,   3306,   4450,   5560,
  6453,   9493,  13548,  14754,
 16743,  18447,  20028,  21736,
 23746,  25353,  27141,  29066,
  3028,   3900,   6617,   7893,
  9211,  10480,  12047,  13583,
 15182,  16662,  18502,  20092,
 22190,  24358,  26302,  28957,
  2000,   2550,   4067,   6837,
  9628,  11002,  12594,  14098,
 15589,  17195,  18679,  20099,
 21530,  23085,  24641,  29022,
  2844,   3302,   5103,   6107,
  6911,   8598,  12416,  14054,
 16026,  18567,  20672,  22270,
 23952,  25771,  27658,  30026,
  4043,   5150,   7268,   9056,
 10916,  12638,  14543,  16184,
 17948,  19691,  21357,  22981,
```

```
 24825,  26591,  28479,  30233,
  2109,   2625,   4320,   5525,
  7454,  10220,  12980,  14698,
 17627,  19263,  20485,  22381,
 24279,  25777,  27847,  30458,
  1550,   2667,   6473,   9496,
 10985,  12352,  13795,  15233,
 17099,  18642,  20461,  22116,
 24197,  26291,  28403,  30132,
  2411,   3084,   4145,   5394,
  6367,   8154,  13125,  16049,
 17561,  19125,  21258,  22762,
 24459,  26317,  28255,  29702,
  4159,   4516,   5956,   7635,
  8254,   8980,  11208,  14133,
 16210,  17875,  20196,  21864,
 23840,  25747,  28058,  30012,
  2026,   2431,   2845,   3618,
  7950,   9802,  12721,  14460,
 16576,  18984,  21376,  23319,
 24961,  26718,  28971,  30640,
  3429,   3833,   4472,   4912,
  7723,  10386,  12981,  15322,
 16699,  18807,  20778,  22551,
 24627,  26494,  28334,  30482,
  4740,   5169,   5796,   6485,
  6998,   8830,  11777,  14414,
 16831,  18413,  20789,  22369,
 24236,  25835,  27807,  30021,
   150,    168,    -17,   -107,
  -142,   -229,   -320,   -406,
  -503,   -620,   -867,   -935,
  -902,   -680,   -398,   -114,
  -398,   -355,     49,    255,
   114,    260,    399,    264,
   317,    431,    514,    531,
   435,    356,    238,    106,
   -43,    -36,   -169,   -224,
  -391,   -633,   -776,   -970,
  -844,   -455,   -181,    -12,
    85,     85,    164,    195,
   122,     85,   -158,   -640,
  -903,      9,      7,   -124,
   149,     32,    220,    369,
   242,    115,     79,     84,
  -146,   -216,    -70,   1024,
   751,    574,    440,    377,
   352,    203,     30,     16,
```

```
    -3,    81,   161,   100,
  -148,  -176,   933,   750,
   404,   171,    -2,  -146,
  -411,  -442,  -541,  -552,
  -442,  -269,  -240,   -52,
   603,   635,   405,   178,
   215,    19,  -153,  -167,
  -290,  -219,   151,   271,
   151,   119,   303,   266,
   100,    69,  -293,  -657,
   939,   659,   442,   351,
   132,    98,   -16,    -1,
  -135,  -200,  -223,   -89,
   167,   154,   172,   237,
   -45,  -183,  -228,  -486,
   263,   608,   158,  -125,
  -390,  -227,  -118,    43,
  -457,  -392,  -769,  -840,
    20,  -117,  -194,  -189,
  -173,  -173,   -33,    32,
   174,   144,   115,   167,
    57,    44,    14,   147,
    96,   -54,  -142,  -129,
  -254,  -331,   304,   310,
   -52,  -419,  -846, -1060,
   -88,  -123,  -202,  -343,
  -554,  -961,  -951,   327,
   159,    81,   255,   227,
   120,   203,   256,   192,
   164,   224,   290,   195,
   216,   209,   128,   832,
  1028,   889,   698,   504,
   408,   355,   218,    32,
  -115,   -84,  -276,  -100,
  -312,  -484,   899,   682,
   465,   456,   241,   -12,
  -275,  -425,  -461,  -367,
   -33,   -28,  -102,  -194,
  -527,   863,   906,   463,
   245,    13,  -212,  -305,
  -105,   163,   279,   176,
    93,    67,   115,   192,
    61,   -50,  -132,  -175,
  -224,  -271,  -629,  -252,
  1158,   972,   638,   280,
   300,   326,   143,  -152,
  -214,  -287,    53,   -42,
  -236,  -352,  -423,  -248,
```

```
  -129,  -163,  -178,  -119,
    85,    57,   514,   382,
   374,   402,   424,   423,
   271,   197,    97,    40,
    39,   -97,  -191,  -164,
  -230,  -256,  -410,   396,
   327,   127,    10,  -119,
  -167,  -291,  -274,  -141,
   -99,  -226,  -218,  -139,
  -224,  -209,  -268,  -442,
  -413,   222,    58,   521,
   344,   258,    76,   -42,
  -142,  -165,  -123,   -92,
    47,     8,    -3,  -191,
   -11,  -164,  -167,  -351,
  -740,   311,   538,   291,
   184,    29,  -105,     9,
   -30,   -54,   -17,   -77,
  -271,  -412,  -622,  -648,
   476,   186,   -66,  -197,
   -73,   -94,   -15,    47,
    28,   112,   -58,   -33,
    65,    19,    84,    86,
   276,   114,   472,   786,
   799,   625,   415,   178,
   -35,   -26,     5,     9,
    83,    39,    37,    39,
  -184,  -374,  -265,  -362,
  -501,   337,   716,   478,
   -60,  -125,  -163,   362,
    17,  -122,  -233,   279,
   138,   157,   318,   193,
   189,   209,   266,   252,
   -46,   -56,  -277,  -429,
   464,   386,   142,    44,
   -43,    66,   264,   182,
    47,    14,   -26,   -79,
    49,    15,  -128,  -203,
  -400,  -478,   325,    27,
   234,   411,   205,   129,
    12,    58,   123,    57,
   171,   137,    96,   128,
   -32,   134,   -12,    57,
   119,    26,   -22,  -165,
  -500,  -701,  -528,  -116,
    64,    -8,    97,    -9,
  -162,   -66,  -156,  -194,
  -303,  -546,  -341,   546,
```

```
  358,   95,   45,   76,
  270,  403,  205,  100,
  123,   50,  -53, -144,
 -110,  -13,   32, -228,
 -130,  353,  296,   56,
 -372, -253,  365,   73,
   10,  -34, -139, -191,
  -96,    5,   44,  -85,
 -179, -129, -192, -246,
  -85, -110, -155,  -44,
  -27,  145,  138,   79,
   32, -148, -577, -634,
  191,   94,   -9,  -35,
  -77,  -84,  -56, -171,
 -298, -271, -243, -156,
 -328, -235,  -76, -128,
 -121,  129,   13,  -22,
   32,   45, -248,  -65,
  193,  -81,  299,   57,
 -147,  192, -165, -354,
 -334, -106, -156,  -40,
   -3,  -68,  124, -257,
   78,  124,  170,  412,
  227,  105, -104,   12,
  154,  250,  274,  258,
    4,  -27,  235,  152,
   51,  338,  300,    7,
 -314, -411,  215,  170,
   -9,  -93,  -77,   76,
   67,   54,  200,  315,
  163,   72,  -91, -402,
  158,  187, -156,  -91,
  290,  267,  167,   91,
  140,  171,  112,    9,
  -42, -177, -440,  385,
   80,   15,  172,  129,
   41, -129, -372,  -24,
  -75,  -30, -170,   10,
 -118,   57,   78, -101,
  232,  161,  123,  256,
  277,  101, -192, -629,
 -100,  -60, -232,   66,
   13,  -13,  -80, -239,
  239,   37,   32,   89,
 -319, -579,  450,  360,
    3,  -29, -299,  -89,
  -54, -110, -246, -164,
    6, -188,  338,  176,
```

```
   -92,   197,   137,   134,
    12,    -2,    56,  -183,
   114,   -36,  -131,  -204,
    75,   -25,  -174,   191,
   -15,  -290,  -429,  -267,
    79,    37,   106,    23,
  -384,   425,    70,   -14,
   212,   105,    15,    -2,
   -42,   -37,  -123,   108,
    28,   -48,   193,   197,
   173,   -33,    37,    73,
   -57,   256,   137,   -58,
  -430,  -228,   217,   -51,
   -10,   -58,    -6,    22,
   104,    61,  -119,   169,
   144,    16,   -46,  -394,
    60,   454,   -80,  -298,
   -65,    25,     0,   -24,
   -65,  -417,   465,   276,
    -3,  -194,   -13,   130,
    19,    -6,   -21,   -24,
  -180,   -53,   -85,    20,
   118,   147,   113,   -75,
  -289,   226,  -122,   227,
   270,   125,   109,   197,
   125,   138,    44,    60,
    25,   -55,  -167,   -32,
  -139,  -193,  -173,  -316,
   287,  -208,   253,   239,
    27,   -80,  -188,   -28,
  -182,  -235,   156,  -117,
   128,   -48,   -58,  -226,
   172,   181,   167,    19,
    62,    10,     2,   181,
   151,   108,   -16,   -11,
   -78,  -331,   411,   133,
    17,   104,    64,  -184,
    24,   -30,    -3,  -283,
   121,   204,    -8,  -199,
   -21,   -80,  -169,  -157,
  -191,  -136,    81,   155,
    14,  -131,   244,    74,
   -57,   -47,  -280,   347,
   111,   -77,  -128,  -142,
  -194,  -125,    -6,   -68,
    91,     1,    23,    14,
  -154,   -34,    23,   -38,
  -343,   503,   146,   -38,
```

```
  -46,   -41,    58,    31,
   63,   -48,  -117,    45,
   28,     1,   -89,    -5,
  -44,   -29,  -448,   487,
  204,    81,    46,  -106,
 -302,   380,   120,   -38,
  -12,   -39,    70,    -3,
   25,   -65,    30,   -11,
   34,   -15,    22,  -115,
    0,   -79,   -83,    45,
  114,    43,   150,    36,
  233,   149,   195,     5,
   25,   -52,  -475,   274,
   28,   -39,    -8,   -66,
 -255,   258,    56,   143,
  -45,  -190,   165,   -60,
   20,     2,   125,  -129,
   51,    -8,  -335,   288,
   38,    59,    25,   -42,
   23,  -118,  -112,    11,
  -55,  -133,  -109,    24,
 -105,    78,   -64,  -245,
  202,   -65,  -127,   162,
   40,   -94,    89,   -85,
 -119,  -103,    97,     9,
  -70,   -28,   194,    86,
 -112,   -92,  -114,    74,
  -49,    46,   -84,  -178,
  113,    52,  -205,   333,
   88,   222,    56,   -55,
   13,    86,     4,   -77,
  224,   114,  -105,   112,
  125,   -29,   -18,  -144,
   22,   -58,   -99,    28,
  114,   -66,   -32,  -169,
 -314,   285,    72,   -74,
  179,    28,   -79,  -182,
   13,   -55,   147,    13,
   12,   -54,    31,   -84,
  -17,   -75,  -228,    83,
 -375,   436,   110,   -63,
  -27,  -136,   169,   -56,
   -8,  -171,   184,   -42,
  148,    68,   204,   235,
  110,  -229,    91,   171,
  -43,    -3,   -26,   -99,
 -111,    71,  -170,   202,
  -67,   181,   -37,   109,
```

```
  -120,    3,  -55, -260,
   -16,  152,   91,  142,
    42,   44,  134,   47,
    17,  -35,   22,   79,
  -169,   41,   46,  277,
   -93,  -49, -126,   37,
  -103,  -34,  -22,  -90,
  -134, -205,   92,   -9,
     1, -195, -239,   45,
    54,   18,  -23,   -1,
   -80,  -98,  -20, -261,
   306,   72,   20,  -89,
  -217,   11,    6,  -82,
    89,   13, -129,  -89,
    83,  -71,  -55,  130,
   -98, -146,  -27,  -57,
    53,  275,   17,  170,
    -5,  -54,  132,  -64,
    72,  160, -125, -168,
    72,   40,  170,   78,
   248,  116,   20,   84,
    31,  -34,  190,   38,
    13, -106,  225,   27,
  -168,   24, -157, -122,
   165,   11, -161, -213,
   -12,  -51, -101,   42,
   101,   27,   55,  111,
    75,   71,  -96,   -1,
    65, -277,  393,  -26,
   -44,  -68,  -84,  -66,
   -95,  235,  179,  -25,
   -41,   27,  -91, -128,
  -222,  146,  -72,  -30,
   -24,   55, -126,  -68,
   -58, -127,   13,  -97,
  -106,  174, -100,  155,
   101, -146,  -21,  261,
    22,   38,  -66,   65,
     4,   70,   64,  144,
    59,  213,   71, -337,
   303,  -52,   51,  -56,
     1,   10,  -15,   -5,
    34,   52,  228,  131,
   161, -127, -214,  238,
   123,   64, -147,  -50,
   -34, -127,  204,  162,
    85,   41,    5, -140,
    73, -150,   56,  -96,
```

```
                -66,            -20,              2,            -235,
                 59,            -22,           -107,             150,
                -16,            -47,             -4,              81,
                -67,            167,            149,             149,
               -157,            288,           -156,             -27,
                 -8,             18,             83,             -24,
                -41,           -167,            158,            -100,
                 93,             53,            201,              15,
                 42,            266,            278,             -12,
                 -6,            -37,             85,               6,
                 20,           -188,           -271,             107,
                -13,            -80,             51,             202,
                173,            -69,             78,            -188,
                 46,              4,            153,              12,
               -138,            169,              5,             -58,
               -123,           -108,           -243,             150,
                 10,           -191,            246,             -15,
                 38,             25,            -10,              14,
                 61,             50,           -206,            -215,
               -220,             90,              5,            -149,
               -219,             56,            142,              24,
               -376,             77,            -80,              75,
                  6,             42,           -101,              16,
                 56,             14,            -57,               3,
                -17,             80,             57,             -36,
                 88,            -59,            -97,             -19,
               -148,             46,           -219,             226,
                114,             -4,            -72,             -15,
                 37,            -49,            -28,             247,
                 44,            123,             47,            -122,
                -38,             17,              4,            -113,
                -32,           -224,            154,            -134,
                196,             71,           -267,             -85,
                 28,            -70,             89,            -120,
                 99,             -2,             64,              76,
               -166,            -48,            189,             -35,
                -92,           -169,           -123,             339,
                 38,            -25,             38,             -35,
                225,           -139,            -50,             -63,
                246,             60,           -185,            -109,
                -49,            -53,           -167,              51,
                149,             60,           -101,             -33,
                 25,            -76,            120,              32,
                -30,            -83,            102,              91,
               -186,           -261,            131,            -197
};

const SKP_Silk_NLSF_CBS SKP_Silk_NLSF_CB0_16_Stage_info[ NLSF_MSVQ_CB0_16_STAGES
] =
```

```
{
        { 128, &SKP_Silk_NLSF_MSVQ_CB0_16_Q15[ 16 *   0 ], &SKP_Silk_NLSF_MSVQ_CB
0_16_rates_Q5[   0 ] },
        {  16, &SKP_Silk_NLSF_MSVQ_CB0_16_Q15[ 16 * 128 ], &SKP_Silk_NLSF_MSVQ_CB
0_16_rates_Q5[ 128 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB0_16_Q15[ 16 * 144 ], &SKP_Silk_NLSF_MSVQ_CB
0_16_rates_Q5[ 144 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB0_16_Q15[ 16 * 152 ], &SKP_Silk_NLSF_MSVQ_CB
0_16_rates_Q5[ 152 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB0_16_Q15[ 16 * 160 ], &SKP_Silk_NLSF_MSVQ_CB
0_16_rates_Q5[ 160 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB0_16_Q15[ 16 * 168 ], &SKP_Silk_NLSF_MSVQ_CB
0_16_rates_Q5[ 168 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB0_16_Q15[ 16 * 176 ], &SKP_Silk_NLSF_MSVQ_CB
0_16_rates_Q5[ 176 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB0_16_Q15[ 16 * 184 ], &SKP_Silk_NLSF_MSVQ_CB
0_16_rates_Q5[ 184 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB0_16_Q15[ 16 * 192 ], &SKP_Silk_NLSF_MSVQ_CB
0_16_rates_Q5[ 192 ] },
        {  16, &SKP_Silk_NLSF_MSVQ_CB0_16_Q15[ 16 * 200 ], &SKP_Silk_NLSF_MSVQ_CB
0_16_rates_Q5[ 200 ] }
};

const SKP_Silk_NLSF_CB_struct SKP_Silk_NLSF_CB0_16 =
{
        NLSF_MSVQ_CB0_16_STAGES,
        SKP_Silk_NLSF_CB0_16_Stage_info,
        SKP_Silk_NLSF_MSVQ_CB0_16_ndelta_min_Q15,
        SKP_Silk_NLSF_MSVQ_CB0_16_CDF,
        SKP_Silk_NLSF_MSVQ_CB0_16_CDF_start_ptr,
        SKP_Silk_NLSF_MSVQ_CB0_16_CDF_middle_idx
};
```

A.127. src/SKP_Silk_tables_NLSF_CB0_16.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
```

```
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_TABLES_NLSF_CB0_16_H
#define SKP_SILK_TABLES_NLSF_CB0_16_H

#include "SKP_Silk_define.h"

#ifdef __cplusplus
extern "C"
{
#endif

#define NLSF_MSVQ_CB0_16_STAGES       10
#define NLSF_MSVQ_CB0_16_VECTORS      216

/* NLSF codebook entropy coding tables */
extern const SKP_uint16         SKP_Silk_NLSF_MSVQ_CB0_16_CDF[ NLSF_MSVQ_CB0_16_V
ECTORS + NLSF_MSVQ_CB0_16_STAGES ];
extern const SKP_uint16 * const SKP_Silk_NLSF_MSVQ_CB0_16_CDF_start_ptr[
         NLSF_MSVQ_CB0_16_STAGES ];
extern const SKP_int            SKP_Silk_NLSF_MSVQ_CB0_16_CDF_middle_idx[
         NLSF_MSVQ_CB0_16_STAGES ];

#ifdef __cplusplus
}
#endif

#endif
```

A.128. src/SKP_Silk_tables_NLSF_CB1_10.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
```

```
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*********************************************/
/* This file has been automatically generated */
/*                                           */
/* ROM usage:    90+803 Words                */
/*********************************************/

#include "SKP_Silk_structs.h"
#include "SKP_Silk_tables_NLSF_CB1_10.h"

const SKP_uint16 SKP_Silk_NLSF_MSVQ_CB1_10_CDF[ NLSF_MSVQ_CB1_10_VECTORS + NLSF_M
SVQ_CB1_10_STAGES ] =
{
            0,
        17096,
        24130,
        28997,
        33179,
        36696,
        40213,
        42493,
        44252,
        45973,
        47551,
        49095,
        50542,
        51898,
        53196,
        54495,
        55685,
        56851,
        57749,
        58628,
        59435,
        60207,
```

```
        60741,
        61220,
        61700,
        62179,
        62659,
        63138,
        63617,
        64097,
        64576,
        65056,
        65535,
            0,
        20378,
        33032,
        40395,
        46721,
        51707,
        56585,
        61157,
        65535,
            0,
        15055,
        25472,
        35447,
        42501,
        48969,
        54773,
        60212,
        65535,
            0,
        12069,
        22440,
        32812,
        40145,
        46870,
        53595,
        59630,
        65535,
            0,
        10839,
        19954,
        27957,
        35961,
        43965,
        51465,
        58805,
        65535,
            0,
```

```
         8933,
        17674,
        26415,
        34785,
        42977,
        50820,
        58496,
        65535
};

const SKP_uint16 * const SKP_Silk_NLSF_MSVQ_CB1_10_CDF_start_ptr[ NLSF_MSVQ_CB1_1
0_STAGES ] =
{
     &SKP_Silk_NLSF_MSVQ_CB1_10_CDF[   0 ],
     &SKP_Silk_NLSF_MSVQ_CB1_10_CDF[  33 ],
     &SKP_Silk_NLSF_MSVQ_CB1_10_CDF[  42 ],
     &SKP_Silk_NLSF_MSVQ_CB1_10_CDF[  51 ],
     &SKP_Silk_NLSF_MSVQ_CB1_10_CDF[  60 ],
     &SKP_Silk_NLSF_MSVQ_CB1_10_CDF[  69 ]
};

const SKP_int SKP_Silk_NLSF_MSVQ_CB1_10_CDF_middle_idx[ NLSF_MSVQ_CB1_10_STAGES ]
 =
{
       5,
       3,
       4,
       4,
       5,
       5
};

const SKP_int16 SKP_Silk_NLSF_MSVQ_CB1_10_rates_Q5[ NLSF_MSVQ_CB1_10_VECTORS ] =
{
               62,             103,
              120,             127,
              135,             135,
              155,             167,
              168,             172,
              173,             176,
              179,             181,
              181,             185,
              186,             198,
              199,             203,
              205,             222,
              227,             227,
              227,             227,
              227,             227,
              227,             227,
              227,             227,
```

```
                54,              76,
               101,             108,
               119,             120,
               123,             125,
                68,              85,
                87,             103,
               107,             112,
               115,             116,
                78,              85,
                85,             101,
               105,             105,
               110,             111,
                83,              91,
                97,              97,
                97,             100,
               101,             105,
                92,              93,
                93,              95,
                96,              98,
                99,             103
};

const SKP_int SKP_Silk_NLSF_MSVQ_CB1_10_ndelta_min_Q15[ 10 + 1 ] =
{
               462,
                 3,
                64,
                74,
                98,
                50,
                97,
                68,
               120,
                53,
               639
};

const SKP_int16 SKP_Silk_NLSF_MSVQ_CB1_10_Q15[ 10 * NLSF_MSVQ_CB1_10_VECTORS ] =
{
              1877,             4646,
              7712,            10745,
             13964,            17028,
             20239,            23182,
             26471,            29287,
              1612,             3278,
              7086,             9975,
             13228,            16264,
             19596,            22690,
```

```
            26037,          28965,
             2169,           3830,
             6460,           8958,
            11960,          14750,
            18408,          21659,
            25018,          28043,
             3680,           6024,
             8986,          12256,
            15201,          18188,
            21741,          24460,
            27484,          30059,
             2584,           5187,
             7799,          10902,
            13179,          15765,
            19017,          22431,
            25891,          28698,
             3731,           5751,
             8650,          11742,
            15090,          17407,
            20391,          23421,
            26228,          29247,
             2107,           6323,
             8915,          12226,
            14775,          17791,
            20664,          23679,
            26829,          29353,
             1677,           2870,
             5386,           8077,
            11817,          15176,
            18657,          22006,
            25513,          28689,
             2111,           3625,
             7027,          10588,
            14059,          17193,
            21137,          24260,
            27577,          30036,
             2428,           4010,
             5765,           9376,
            13805,          15821,
            19444,          22389,
            25295,          29310,
             2256,           4628,
             8377,          12441,
            15283,          19462,
            22257,          25551,
            28432,          30304,
             2352,           3675,
             6129,          11868,
```

```
            14551,          16655,
            19624,          21883,
            26526,          28849,
             5243,           7248,
            10558,          13269,
            15651,          17919,
            21141,          23827,
            27102,          29519,
             4422,           6725,
            10449,          13273,
            16124,          19921,
            22826,          26061,
            28763,          30583,
             4508,           6291,
             9504,          11809,
            13827,          15950,
            19077,          22084,
            25740,          28658,
             2540,           4297,
             8579,          13578,
            16634,          19101,
            21547,          23887,
            26777,          29146,
             3377,           6358,
            10224,          14518,
            17905,          21056,
            23637,          25784,
            28161,          30109,
             4177,           5942,
             8159,          10108,
            12130,          15470,
            20191,          23326,
            26782,          29359,
             2492,           3801,
             6144,           9825,
            16000,          18671,
            20893,          23663,
            25899,          28974,
             3011,           4727,
             6834,          10505,
            12465,          14496,
            17065,          20052,
            25265,          28057,
             4149,           7197,
            12338,          15076,
            18002,          20190,
            22187,          24723,
            27083,          29125,
```

```
 2975,  4578,
 6448,  8378,
 9671, 13225,
19502, 22277,
26058, 28850,
 4102,  5760,
 7744,  9484,
10744, 12308,
14677, 19607,
24841, 28381,
 4931,  9287,
12477, 13395,
13712, 14351,
16048, 19867,
24188, 28994,
 4141,  7867,
13140, 17720,
20064, 21108,
21692, 22722,
23736, 27449,
 4011,  8720,
13234, 16206,
17601, 18289,
18524, 19689,
23234, 27882,
 3420,  5995,
11230, 15117,
15907, 16783,
17762, 23347,
26898, 29946,
 3080,  6786,
10465, 13676,
18059, 23615,
27058, 29082,
29563, 29905,
 3038,  5620,
 9266, 12870,
18803, 19610,
20010, 20802,
23882, 29306,
 3314,  6420,
 9046, 13262,
15869, 23117,
23667, 24215,
24487, 25915,
 3469,  6963,
10103, 15282,
20531, 23240,
```

```
            25024,          26021,
            26736,          27255,
             3041,           6459,
             9777,          12896,
            16315,          19410,
            24070,          29353,
            31795,          32075,
             -200,           -134,
             -113,           -204,
             -347,           -440,
             -352,           -211,
             -418,           -172,
             -313,             59,
              495,            772,
              721,            614,
              334,            444,
              225,            242,
              161,             16,
              274,            564,
              -73,           -188,
             -395,           -171,
              777,            508,
             1340,           1145,
              699,            196,
              223,            173,
               90,             25,
              -26,             18,
              133,           -105,
             -360,           -277,
              859,            634,
               41,           -557,
             -768,           -926,
             -601,          -1021,
            -1189,           -365,
              225,            107,
              374,            -50,
              433,            417,
              156,             39,
             -597,          -1397,
            -1594,           -592,
             -485,           -292,
              253,             87,
               -0,             -6,
              -25,           -345,
             -240,            120,
             1261,            946,
              166,           -277,
              241,            167,
```

```
 170,  429,
 518,  714,
 602,  254,
 134,   92,
-152, -324,
-394,   49,
-151, -304,
-724, -657,
-162, -369,
 -35,    3,
  -2, -312,
-200,  -92,
-227,  242,
 628,  565,
-124, 1056,
 770,  101,
 -84,  -33,
   4, -192,
-272,    5,
-627, -977,
 419,  472,
  53, -103,
 145,  322,
 -95,  -31,
-100, -303,
-560, -1067,
-413,  714,
 283,    2,
-223, -367,
 523,  360,
 -38, -115,
 378, -591,
-718,  448,
-481, -274,
 180,  -88,
-581, -157,
-696, -1265,
 394, -479,
 -23,  124,
 -43,   19,
-113, -236,
-412, -659,
-200,    2,
 -69, -342,
 199,   55,
  58,  -36,
 -51,  -62,
 507,  507,
```

```
 427,  442,
  36,  601,
-141,   68,
 274,  274,
  68,  -12,
  -4,   71,
-193, -464,
-425, -383,
 408,  203,
-337,  236,
 410,  -59,
 -25, -341,
-449,   28,
  -9,   90,
 332,  -14,
-905,   96,
-540, -242,
 679,  -59,
 192,  -24,
  60, -217,
   5,  -37,
 179,  -20,
 311,  519,
 274,   72,
-326, -1030,
-262,  213,
 380,   82,
 328,  411,
-540,  574,
-283,  151,
 181, -402,
-278, -240,
-110, -227,
-264,  -89,
-250, -259,
 -27,  106,
-239,  -98,
-390,  118,
  61,  104,
 294,  532,
  92,  -13,
  60, -233,
 335,  541,
 307,  -26,
-110,  -91,
-231, -460,
 170,  201,
  96, -372,
```

```
 132,  435,
-302,  216,
-279,  -41,
  74,  190,
 368,  273,
-186, -608,
-157,  159,
  12,  278,
 245,  307,
  25, -187,
 -16,   55,
  30, -163,
 548, -307,
 106,   -5,
  27,  330,
-416,  475,
 438, -235,
 104,  137,
  21,   -5,
-300, -468,
 521, -347,
 170, -200,
-219,  308,
-122, -133,
 219,  -16,
 359,  412,
 -89, -111,
  48,  322,
 142,  177,
-286, -127,
 -39,  -63,
 -42, -451,
 160,  308,
 -57,  193,
 -48,   74,
-346,   59,
 -27,   27,
-469, -277,
-344,  282,
 262,  122,
 171, -249,
  27,  258,
 188,   -3,
  67, -206,
-284,  291,
-117,  -88,
-477,  375,
  50,  106,
```

```
                99,            -182,
               438,            -376,
              -401,             -49,
               119,             -23,
               -10,             -48,
              -116,            -200,
              -310,             121,
                73,               7,
               237,            -226,
               139,            -456,
               397,              35,
                 3,            -108,
               323,             -75,
               332,             198,
               -99,             -21
};

const SKP_Silk_NLSF_CBS SKP_Silk_NLSF_CB1_10_Stage_info[ NLSF_MSVQ_CB1_10_STAGES
] =
{
        {  32, &SKP_Silk_NLSF_MSVQ_CB1_10_Q15[ 10 *   0 ], &SKP_Silk_NLSF_MSVQ_CB
1_10_rates_Q5[   0 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_10_Q15[ 10 *  32 ], &SKP_Silk_NLSF_MSVQ_CB
1_10_rates_Q5[  32 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_10_Q15[ 10 *  40 ], &SKP_Silk_NLSF_MSVQ_CB
1_10_rates_Q5[  40 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_10_Q15[ 10 *  48 ], &SKP_Silk_NLSF_MSVQ_CB
1_10_rates_Q5[  48 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_10_Q15[ 10 *  56 ], &SKP_Silk_NLSF_MSVQ_CB
1_10_rates_Q5[  56 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_10_Q15[ 10 *  64 ], &SKP_Silk_NLSF_MSVQ_CB
1_10_rates_Q5[  64 ] }
};

const SKP_Silk_NLSF_CB_struct SKP_Silk_NLSF_CB1_10 =
{
        NLSF_MSVQ_CB1_10_STAGES,
        SKP_Silk_NLSF_CB1_10_Stage_info,
        SKP_Silk_NLSF_MSVQ_CB1_10_ndelta_min_Q15,
        SKP_Silk_NLSF_MSVQ_CB1_10_CDF,
        SKP_Silk_NLSF_MSVQ_CB1_10_CDF_start_ptr,
        SKP_Silk_NLSF_MSVQ_CB1_10_CDF_middle_idx
};
```

A.129. src/SKP_Silk_tables_NLSF_CB1_10.h

```
/**********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
```

```
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_TABLES_NLSF_CB1_10_H
#define SKP_SILK_TABLES_NLSF_CB1_10_H

#include "SKP_Silk_define.h"

#ifdef __cplusplus
extern "C"
{
#endif

#define NLSF_MSVQ_CB1_10_STAGES       6
#define NLSF_MSVQ_CB1_10_VECTORS      72

/* NLSF codebook entropy coding tables */
extern const SKP_uint16         SKP_Silk_NLSF_MSVQ_CB1_10_CDF[ NLSF_MSVQ_CB1_10_V
ECTORS + NLSF_MSVQ_CB1_10_STAGES ];
extern const SKP_uint16 * const SKP_Silk_NLSF_MSVQ_CB1_10_CDF_start_ptr[
         NLSF_MSVQ_CB1_10_STAGES ];
extern const SKP_int            SKP_Silk_NLSF_MSVQ_CB1_10_CDF_middle_idx[
         NLSF_MSVQ_CB1_10_STAGES ];

#ifdef __cplusplus
}
#endif

#endif
```

A.130. src/SKP_Silk_tables_NLSF_CB1_16.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*******************************************/
/* This file has been automatically generated */
/*                                         */
/* ROM usage:   134+1785 Words             */
/*******************************************/

#include "SKP_Silk_structs.h"
#include "SKP_Silk_tables_NLSF_CB1_16.h"

const SKP_uint16 SKP_Silk_NLSF_MSVQ_CB1_16_CDF[ NLSF_MSVQ_CB1_16_VECTORS + NLSF_M
SVQ_CB1_16_STAGES ] =
{
           0,
       19099,
       26957,
       30639,
       34242,
       37546,
       40447,
       43287,
       46005,
       48445,
```

```
        49865,
        51284,
        52673,
        53975,
        55221,
        56441,
        57267,
        58025,
        58648,
        59232,
        59768,
        60248,
        60729,
        61210,
        61690,
        62171,
        62651,
        63132,
        63613,
        64093,
        64574,
        65054,
        65535,
            0,
        28808,
        38775,
        46801,
        51785,
        55886,
        59410,
        62572,
        65535,
            0,
        27376,
        38639,
        45052,
        51465,
        55448,
        59021,
        62594,
        65535,
            0,
        33403,
        39569,
        45102,
        49961,
        54047,
        57959,
```

```
        61788,
        65535,
            0,
        25851,
        43356,
        47828,
        52204,
        55964,
        59413,
        62507,
        65535,
            0,
        34277,
        40337,
        45432,
        50311,
        54326,
        58171,
        61853,
        65535,
            0,
        33538,
        39865,
        45302,
        50076,
        54549,
        58478,
        62159,
        65535,
            0,
        27445,
        35258,
        40665,
        46072,
        51362,
        56540,
        61086,
        65535,
            0,
        22080,
        30779,
        37065,
        43085,
        48849,
        54613,
        60133,
        65535,
            0,
```

```
        13417,
        21748,
        30078,
        38231,
        46383,
        53091,
        59515,
        65535
};

const SKP_uint16 * const SKP_Silk_NLSF_MSVQ_CB1_16_CDF_start_ptr[ NLSF_MSVQ_CB1_1
6_STAGES ] =
{
     &SKP_Silk_NLSF_MSVQ_CB1_16_CDF[   0 ],
     &SKP_Silk_NLSF_MSVQ_CB1_16_CDF[  33 ],
     &SKP_Silk_NLSF_MSVQ_CB1_16_CDF[  42 ],
     &SKP_Silk_NLSF_MSVQ_CB1_16_CDF[  51 ],
     &SKP_Silk_NLSF_MSVQ_CB1_16_CDF[  60 ],
     &SKP_Silk_NLSF_MSVQ_CB1_16_CDF[  69 ],
     &SKP_Silk_NLSF_MSVQ_CB1_16_CDF[  78 ],
     &SKP_Silk_NLSF_MSVQ_CB1_16_CDF[  87 ],
     &SKP_Silk_NLSF_MSVQ_CB1_16_CDF[  96 ],
     &SKP_Silk_NLSF_MSVQ_CB1_16_CDF[ 105 ]
};

const SKP_int SKP_Silk_NLSF_MSVQ_CB1_16_CDF_middle_idx[ NLSF_MSVQ_CB1_16_STAGES ]
 =
{
       5,
       2,
       2,
       2,
       2,
       2,
       2,
       3,
       3,
       4
};

const SKP_int16 SKP_Silk_NLSF_MSVQ_CB1_16_rates_Q5[ NLSF_MSVQ_CB1_16_VECTORS ] =
{
               57,              98,
              133,             134,
              138,             144,
              145,             147,
              152,             177,
              177,             178,
              181,             183,
              184,             202,
```

```
                206,             215,
                218,             222,
                227,             227,
                227,             227,
                227,             227,
                227,             227,
                227,             227,
                227,             227,
                 38,              87,
                 97,             119,
                128,             135,
                140,             143,
                 40,              81,
                107,             107,
                129,             134,
                134,             143,
                 31,             109,
                114,             120,
                128,             130,
                131,             132,
                 43,              61,
                124,             125,
                132,             136,
                141,             142,
                 30,             110,
                118,             120,
                129,             131,
                133,             133,
                 31,             108,
                115,             121,
                124,             130,
                133,             137,
                 40,              98,
                115,             115,
                116,             117,
                123,             124,
                 50,              93,
                108,             110,
                112,             112,
                114,             115,
                 73,              95,
                 95,              96,
                 96,             105,
                107,             110
};

const SKP_int SKP_Silk_NLSF_MSVQ_CB1_16_ndelta_min_Q15[ 16 + 1 ] =
{
```

```
                148,
                  3,
                 60,
                 68,
                117,
                 86,
                121,
                124,
                152,
                153,
                207,
                151,
                225,
                239,
                126,
                183,
                792
};

const SKP_int16 SKP_Silk_NLSF_MSVQ_CB1_16_Q15[ 16 * NLSF_MSVQ_CB1_16_VECTORS ] =
{
                1309,            3060,            5071,            6996,
                9028,           10938,           12934,           14891,
               16933,           18854,           20792,           22764,
               24753,           26659,           28626,           30501,
                1264,            2745,            4610,            6408,
                8286,           10043,           12084,           14108,
               16118,           18163,           20095,           22164,
               24264,           26316,           28329,           30251,
                1044,            2080,            3672,            5179,
                7140,            9100,           11070,           13065,
               15423,           17790,           19931,           22101,
               24290,           26361,           28499,           30418,
                1131,            2476,            4478,            6149,
                7902,            9875,           11938,           13809,
               15869,           17730,           19948,           21707,
               23761,           25535,           27426,           28917,
                1040,            2004,            4026,            6100,
                8432,           10494,           12610,           14694,
               16797,           18775,           20799,           22782,
               24772,           26682,           28631,           30516,
                2310,            3812,            5913,            7933,
               10033,           11881,           13885,           15798,
               17751,           19576,           21482,           23276,
               25157,           27010,           28833,           30623,
                1254,            2847,            5013,            6781,
                8626,           10370,           12726,           14633,
               16281,           17852,           19870,           21472,
```

```
    23002,  24629,  26710,  27960,
     1468,   3059,   4987,   7026,
     8741,  10412,  12281,  14020,
    15970,  17723,  19640,  21522,
    23472,  25661,  27986,  30225,
     2171,   3566,   5605,   7384,
     9404,  11220,  13030,  14758,
    16687,  18417,  20346,  22091,
    24055,  26212,  28356,  30397,
     2409,   4676,   7543,   9786,
    11419,  12935,  14368,  15653,
    17366,  18943,  20762,  22477,
    24440,  26327,  28284,  30242,
     2354,   4222,   6820,   9107,
    11596,  13934,  15973,  17682,
    19158,  20517,  21991,  23420,
    25178,  26936,  28794,  30527,
     1323,   2414,   4184,   6039,
     7534,   9398,  11099,  13097,
    14799,  16451,  18434,  20887,
    23490,  25838,  28046,  30225,
     1361,   3243,   6048,   8511,
    11001,  13145,  15073,  16608,
    18126,  19381,  20912,  22607,
    24660,  26668,  28663,  30566,
     1216,   2648,   5901,   8422,
    10037,  11425,  12973,  14603,
    16686,  18600,  20555,  22415,
    24450,  26280,  28206,  30077,
     2417,   4048,   6316,   8433,
    10510,  12757,  15072,  17295,
    19573,  21503,  23329,  24782,
    26235,  27689,  29214,  30819,
     1012,   2345,   4991,   7377,
     9465,  11916,  14296,  16566,
    18672,  20544,  22292,  23838,
    25415,  27050,  28848,  30551,
     1937,   3693,   6267,   8019,
    10372,  12194,  14287,  15657,
    17431,  18864,  20769,  22206,
    24037,  25463,  27383,  28602,
     1969,   3305,   5017,   6726,
     8375,   9993,  11634,  13280,
    15078,  16751,  18464,  20119,
    21959,  23858,  26224,  29298,
     1198,   2647,   5428,   7423,
     9775,  12155,  14665,  16344,
    18121,  19790,  21557,  22847,
```

```
 24484,  25742,  27639,  28711,
  1636,   3353,   5447,   7597,
  9837,  11647,  13964,  16019,
 17862,  20116,  22319,  24037,
 25966,  28086,  29914,  31294,
  2676,   4105,   6378,   8223,
 10058,  11549,  13072,  14453,
 15956,  17355,  18931,  20402,
 22183,  23884,  25717,  27723,
  1373,   2593,   4449,   5633,
  7300,   8425,   9474,  10818,
 12769,  15722,  19002,  21429,
 23682,  25924,  28135,  30333,
  1596,   3183,   5378,   7164,
  8670,  10105,  11470,  12834,
 13991,  15042,  16642,  17903,
 20759,  25283,  27770,  30240,
  2037,   3987,   6237,   8117,
  9954,  12245,  14217,  15892,
 17775,  20114,  22314,  25942,
 26305,  26483,  26796,  28561,
  2181,   3858,   5760,   7924,
 10041,  11577,  13769,  15700,
 17429,  19879,  23583,  24538,
 25212,  25693,  28688,  30507,
  1992,   3882,   6474,   7883,
  9381,  12672,  14340,  15701,
 16658,  17832,  20850,  22885,
 24677,  26457,  28491,  30460,
  2391,   3988,   5448,   7432,
 11014,  12579,  13140,  14146,
 15898,  18592,  21104,  22993,
 24673,  27186,  28142,  29612,
  1713,   5102,   6989,   7798,
  8670,  10110,  12746,  14881,
 16709,  18407,  20126,  22107,
 24181,  26198,  28237,  30137,
  1612,   3617,   6148,   8359,
  9576,  11528,  14936,  17809,
 18287,  18729,  19001,  21111,
 24631,  26596,  28740,  30643,
  2266,   4168,   7862,   9546,
  9618,   9703,  10134,  13897,
 16265,  18432,  20587,  22605,
 24754,  26994,  29125,  30840,
  1840,   3917,   6272,   7809,
  9714,  11438,  13767,  15799,
 19244,  21972,  22980,  23180,
```

```
    23723,      25650,      29117,      31085,
     1458,       3612,       6008,       7488,
     9827,      11893,      14086,      15734,
    17440,      19535,      22424,      24767,
    29246,      29928,      30516,      30947,
     -102,       -121,        -31,         -6,
        5,         -2,          8,        -18,
       -4,          6,         14,         -2,
      -12,        -16,        -12,        -60,
     -126,       -353,       -574,       -677,
     -657,       -617,       -498,       -393,
     -348,       -277,       -225,       -164,
     -102,        -70,        -31,         33,
        4,        379,        387,        551,
      605,        620,        532,        482,
      442,        454,        385,        347,
      322,        299,        266,        200,
     1168,        951,        672,        246,
       60,       -161,       -259,       -234,
     -253,       -282,       -203,       -187,
     -155,       -176,       -198,       -178,
       10,        170,        393,        609,
      555,        208,       -330,       -571,
     -769,       -633,       -319,        -43,
       95,        105,        106,        116,
     -152,       -140,       -125,          5,
      173,        274,        264,        331,
      -37,       -293,       -609,       -786,
     -959,       -814,       -645,       -238,
      -91,         36,        -11,       -101,
     -279,       -227,        -40,         90,
      530,        677,        890,       1104,
      999,        835,        564,        295,
     -280,       -364,       -340,       -331,
     -284,        288,        761,        880,
      988,        627,        146,       -226,
     -203,       -181,       -142,         39,
       24,        -26,       -107,        -92,
     -161,       -135,       -131,        -88,
     -160,       -156,        -75,        -43,
      -36,         -6,        -33,         33,
     -324,       -415,       -108,        124,
      157,        191,        203,        197,
      144,        109,        152,        176,
      190,        122,        101,        159,
      663,        668,        480,        400,
      379,        444,        446,        458,
      343,        351,        310,        228,
```

```
   133,    44,    75,    63,
   -84,    39,   -29,    35,
   -94,  -233,  -261,  -354,
    77,   262,   -24,  -145,
  -333,  -409,  -404,  -597,
  -488,  -300,   910,   592,
   412,   120,   130,   -51,
   -37,   -77,  -172,  -181,
  -159,  -148,   -72,   -62,
   510,   516,   113,  -585,
 -1075,  -957,  -417,  -195,
     9,     7,   -88,  -173,
   -91,    54,    98,    95,
   -28,   197,  -527,  -621,
   157,   122,  -168,   147,
   309,   300,   336,   315,
   396,   408,   376,   106,
  -162,  -170,  -315,    98,
   821,   908,   570,   -33,
  -312,  -568,  -572,  -378,
  -107,    23,   156,    93,
  -129,   -87,    20,   -72,
   -37,    40,    21,    27,
    48,    75,    77,    65,
    46,    71,    66,    47,
   136,   344,   236,   322,
   170,   283,   269,   291,
   162,   -43,  -204,  -259,
  -240,  -305,  -350,  -312,
   447,   348,   345,   257,
    71,  -131,   -77,  -190,
  -202,   -40,    35,   133,
   261,   365,   438,   303,
    -8,    22,   140,   137,
  -300,  -641,  -764,  -268,
   -23,   -25,    73,  -162,
  -150,  -212,   -72,     6,
    39,    78,   104,   -93,
  -308,  -136,   117,   -71,
  -513,  -820,  -700,  -450,
  -161,   -23,    29,    78,
   337,   106,  -406,  -782,
  -112,   233,   383,    62,
  -126,     6,   -77,   -29,
  -146,  -123,   -51,   -27,
   -27,  -381,  -641,   402,
   539,     8,  -207,  -366,
   -36,   -27,  -204,  -227,
```

```
  -237,  -189,   -64,    51,
   -92,  -137,  -281,    62,
   233,    92,   148,   294,
   363,   416,   564,   625,
   370,   -36,  -469,  -462,
   102,   168,    32,   117,
   -21,    97,   139,    89,
   104,    35,     4,    82,
    66,    58,    73,    93,
   -76,  -320,  -236,  -189,
  -203,  -142,   -27,   -73,
     9,    -9,   -25,    12,
   -15,     4,     4,   -50,
   314,   180,   162,   -49,
   199,  -108,  -227,   -66,
  -447,   -67,  -264,  -394,
     5,    55,  -133,  -176,
  -116,  -241,   272,   109,
   282,   262,   192,   -64,
  -392,  -514,   156,   203,
   154,    72,   -34,  -160,
   -73,     3,   -33,  -431,
   321,    18,  -567,  -590,
  -108,    88,    66,    51,
   -31,  -193,   -46,    65,
   -29,   -23,   215,   -31,
   101,  -113,    32,   304,
    88,   320,   448,     5,
  -439,  -562,  -508,  -135,
   -13,  -171,    -8,   182,
   -99,  -181,  -149,   376,
   476,    64,  -396,  -652,
  -150,   176,   222,    65,
  -590,   719,   271,   399,
   245,    72,  -156,  -152,
  -176,    59,    94,   125,
    -9,    -7,     9,     1,
   -61,  -116,   -82,     1,
    79,    22,   -44,   -15,
   -48,   -65,   -62,  -101,
  -102,   -54,   -70,   -78,
   -80,   -25,   398,    71,
   139,    38,    90,   194,
   222,   249,   165,    94,
   221,   262,   163,    91,
  -206,   573,   200,  -287,
  -147,     5,   -18,   -85,
   -74,  -125,   -87,    85,
```

```
  141,    4,   -4,   28,
  234,   48,  -150,  -111,
 -506,  237,  -209,   345,
   94,  -124,   77,   121,
  143,   12,   -80,   -48,
  191,  144,   -93,   -65,
 -151,  -643,  435,   106,
   87,    7,   65,   102,
   94,   68,    5,    99,
  222,   93,   94,   355,
  -13,  -89,  -228,  -503,
  287,  109,   108,   449,
  253,  -29,  -109,  -116,
   15,  -73,   -20,   131,
 -147,   72,   59,  -150,
 -594,  273,  316,   132,
  199,  106,   198,   212,
  220,   82,   45,   -13,
  223,  137,   270,   38,
  252,  135,  -177,  -207,
 -360,  -102,  403,   406,
  -14,   83,   64,    51,
   -7,  -99,   -97,   -88,
 -124,  -65,   42,    32,
   28,   29,   12,    20,
  119,  -26,  -212,  -201,
  373,  251,   141,   103,
   36,  -52,   66,    18,
   -6,  -95,  -196,    5,
   98,  -85,  -108,   218,
 -164,   20,   356,   172,
   37,  266,   23,    112,
  -24,  -99,  -92,   -178,
   29,  -278,  388,   -60,
 -220,  300,  -13,   154,
  191,   15,   -37,  -110,
 -153,  -150,  -114,   -7,
  -94,  -31,   -62,  -177,
    4,   -70,   35,   453,
  147,  -247,  -328,  101,
   20,  -114,  147,   108,
 -119,  -109,  -102,  -238,
   55,  -102,  173,   -89,
  129,   138,  -330,  -160,
  485,   154,  -59,  -170,
  -20,  -34,  -261,   -40,
 -129,   77,   -84,   69,
   83,   160,   169,   63,
```

```
  -516,   30,  336,   52,
    -0,  -52, -124,  158,
    19,  197,  -10, -375,
   405,  285,  114, -395,
   -47,  196,   62,   87,
  -106,  -65,  -75,  -69,
   -13,   34,   99,   59,
    83,   98,   44,    0,
    24,   18,   17,   70,
   -22,  194,  208,  144,
   -79,  -15,   32, -104,
   -28, -105, -186, -212,
  -228,  -79,  -76,   51,
   -71,   72,  118,  -34,
    -3, -171,    5,    2,
  -108, -125,   62,  -58,
    58, -121,   73, -466,
    92,   63,  -94,  -78,
   -76,  212,   36, -225,
   -71, -354,  152,  143,
   -79, -246,  -51,  -31,
    -6, -270,  240,  210,
    30, -157, -231,   74,
  -146,   88, -273,  156,
    92,   56,   71,    2,
   318,  164,   32, -110,
   -35,  -41,  -95, -106,
    11,  132,  -68,   55,
   123,  -83, -149,  212,
   132,    0, -194,   55,
   206, -108, -353,  289,
  -195,    1,  233,  -22,
   -60,   20,   26,   68,
   166,   27,  -58,  130,
   112,  107,   27, -165,
   115,  -93,  -37,   38,
    83,  483,   65, -229,
   -13,  157,   85,   50,
   136,   10,   32,   83,
    82,   55,    5,   -9,
   -52,  -78,  -81,  -51,
    40,   18, -127, -224,
   -41,   53, -210, -113,
    24,  -17, -187,  -89,
     8,  121,   83,   77,
    91,  -74,  -35, -112,
  -161, -173,  102,  132,
  -125,  -61,  103, -260,
```

```
   52,   166,   -32,  -156,
  -87,   -56,    60,   -70,
 -124,   242,   114,  -251,
 -166,   201,   127,    28,
  -11,    23,   -80,  -115,
  -20,   -51,  -348,   340,
  -34,   133,    13,    92,
 -124,  -136,  -120,   -26,
   -6,    17,    28,    21,
  120,  -168,   160,   -35,
  115,    28,     9,     7,
  -56,    39,   156,   256,
  -18,     1,   277,    82,
  -70,  -144,   -88,   -13,
  -59,  -157,     8,  -134,
   21,   -40,    58,   -21,
  194,  -276,    97,   279,
  -56,  -140,   125,    57,
 -184,  -204,   -70,    -2,
  128,  -202,   -78,   230,
  -23,   161,  -102,     1,
    1,   180,   -31,   -86,
 -167,   -57,   -60,    27,
  -13,    99,   108,   111,
   76,    69,    34,   -21,
   53,    38,    34,    78,
   73,   219,    51,    15,
  -72,  -103,  -207,    30,
  213,   -14,    31,   -94,
  -40,  -144,    67,     4,
  105,    59,  -240,    25,
  244,    69,    58,    23,
  -24,    -5,   -15,  -133,
  -71,   -67,   181,    29,
  -45,   121,    96,    51,
  -72,   -53,    56,  -153,
  -27,    85,   183,   211,
  105,   -34,   -46,    43,
  -72,   -93,    36,  -128,
   29,   111,   -95,  -156,
 -179,  -235,    21,   -39,
  -71,   -33,   -61,  -252,
  230,  -131,   157,   -21,
  -85,   -28,  -123,    80,
 -160,    63,    47,    -6,
  -49,   -96,   -19,    17,
  -58,    17,    -0,   -13,
 -170,    25,   -35,    59,
```

```
                10,             -31,            -413,              81,
                62,              18,            -164,             245,
                92,            -165,              42,              26,
               126,            -248,             193,             -55,
                16,              39,              14,              50
};

const SKP_Silk_NLSF_CBS SKP_Silk_NLSF_CB1_16_Stage_info[ NLSF_MSVQ_CB1_16_STAGES
] =
{
        {  32, &SKP_Silk_NLSF_MSVQ_CB1_16_Q15[ 16 *   0 ], &SKP_Silk_NLSF_MSVQ_CB
1_16_rates_Q5[   0 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_16_Q15[ 16 *  32 ], &SKP_Silk_NLSF_MSVQ_CB
1_16_rates_Q5[  32 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_16_Q15[ 16 *  40 ], &SKP_Silk_NLSF_MSVQ_CB
1_16_rates_Q5[  40 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_16_Q15[ 16 *  48 ], &SKP_Silk_NLSF_MSVQ_CB
1_16_rates_Q5[  48 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_16_Q15[ 16 *  56 ], &SKP_Silk_NLSF_MSVQ_CB
1_16_rates_Q5[  56 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_16_Q15[ 16 *  64 ], &SKP_Silk_NLSF_MSVQ_CB
1_16_rates_Q5[  64 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_16_Q15[ 16 *  72 ], &SKP_Silk_NLSF_MSVQ_CB
1_16_rates_Q5[  72 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_16_Q15[ 16 *  80 ], &SKP_Silk_NLSF_MSVQ_CB
1_16_rates_Q5[  80 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_16_Q15[ 16 *  88 ], &SKP_Silk_NLSF_MSVQ_CB
1_16_rates_Q5[  88 ] },
        {   8, &SKP_Silk_NLSF_MSVQ_CB1_16_Q15[ 16 *  96 ], &SKP_Silk_NLSF_MSVQ_CB
1_16_rates_Q5[  96 ] }
};

const SKP_Silk_NLSF_CB_struct SKP_Silk_NLSF_CB1_16 =
{
        NLSF_MSVQ_CB1_16_STAGES,
        SKP_Silk_NLSF_CB1_16_Stage_info,
        SKP_Silk_NLSF_MSVQ_CB1_16_ndelta_min_Q15,
        SKP_Silk_NLSF_MSVQ_CB1_16_CDF,
        SKP_Silk_NLSF_MSVQ_CB1_16_CDF_start_ptr,
        SKP_Silk_NLSF_MSVQ_CB1_16_CDF_middle_idx
};
```

A.131. src/SKP_Silk_tables_NLSF_CB1_16.h

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
```

```
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#ifndef SKP_SILK_TABLES_NLSF_CB1_16_H
#define SKP_SILK_TABLES_NLSF_CB1_16_H

#include "SKP_Silk_define.h"

#ifdef __cplusplus
extern "C"
{
#endif

#define NLSF_MSVQ_CB1_16_STAGES       10
#define NLSF_MSVQ_CB1_16_VECTORS      104

/* NLSF codebook entropy coding tables */
extern const SKP_uint16         SKP_Silk_NLSF_MSVQ_CB1_16_CDF[ NLSF_MSVQ_CB1_16_V
ECTORS + NLSF_MSVQ_CB1_16_STAGES ];
extern const SKP_uint16 * const SKP_Silk_NLSF_MSVQ_CB1_16_CDF_start_ptr[
         NLSF_MSVQ_CB1_16_STAGES ];
extern const SKP_int            SKP_Silk_NLSF_MSVQ_CB1_16_CDF_middle_idx[
         NLSF_MSVQ_CB1_16_STAGES ];

#ifdef __cplusplus
}
#endif

#endif
```

A.132. src/SKP_Silk_tables_other.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
```

```
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_structs.h"
#include "SKP_Silk_define.h"
#include "SKP_Silk_tables.h"
#ifdef __cplusplus
extern "C"
{
#endif

/* Piece-wise linear mapping from bitrate in kbps to coding quality in dB SNR */
const SKP_int32 TargetRate_table_NB[ TARGET_RATE_TAB_SZ ] = {
    0,      8000,   9000,   11000,  13000,  16000,  22000,  100000
};
const SKP_int32 TargetRate_table_MB[ TARGET_RATE_TAB_SZ ] = {
    0,      10000,  12000,  14000,  17000,  21000,  28000,  100000
};
const SKP_int32 TargetRate_table_WB[ TARGET_RATE_TAB_SZ ] = {
    0,      11000,  14000,  17000,  21000,  26000,  36000,  100000
};
const SKP_int32 TargetRate_table_SWB[ TARGET_RATE_TAB_SZ ] = {
    0,      13000,  16000,  19000,  25000,  32000,  46000,  100000
};
const SKP_int32 SNR_table_Q1[ TARGET_RATE_TAB_SZ ] = {
    19,     31,     35,     39,     43,     47,     54,     59
};
```

```
const SKP_int32 SNR_table_one_bit_per_sample_Q7[ 4 ] = {
    1984,   2240,   2408,   2708
};

/* Filter coeficicnts for HP filter: 4. Order filter implementad as two biquad fi
lters  */
const SKP_int16 SKP_Silk_SWB_detect_B_HP_Q13[ NB_SOS ][ 3 ] = {
    //{400, -550, 400}, {400, 130, 400}, {400, 390, 400}
    {575, -948, 575}, {575, -221, 575}, {575, 104, 575}
};
const SKP_int16 SKP_Silk_SWB_detect_A_HP_Q13[ NB_SOS ][ 2 ] = {
    {14613, 6868}, {12883, 7337}, {11586, 7911}
    //{14880, 6900}, {14400, 7300}, {13700, 7800}
};

/* Decoder high-pass filter coefficients for 24 kHz sampling, -6 dB @ 44 Hz */
const SKP_int16 SKP_Silk_Dec_A_HP_24[ DEC_HP_ORDER     ] = {-16220, 8030};
       // second order AR coefs, Q13
const SKP_int16 SKP_Silk_Dec_B_HP_24[ DEC_HP_ORDER + 1 ] = {8000, -16000, 8000};
       // second order MA coefs, Q13

/* Decoder high-pass filter coefficients for 16 kHz sampling, - 6 dB @ 46 Hz */
const SKP_int16 SKP_Silk_Dec_A_HP_16[ DEC_HP_ORDER     ] = {-16127, 7940};
       // second order AR coefs, Q13
const SKP_int16 SKP_Silk_Dec_B_HP_16[ DEC_HP_ORDER + 1 ] = {8000, -16000, 8000};
       // second order MA coefs, Q13

/* Decoder high-pass filter coefficients for 12 kHz sampling, -6 dB @ 44 Hz */
const SKP_int16 SKP_Silk_Dec_A_HP_12[ DEC_HP_ORDER     ] = {-16043, 7859};
       // second order AR coefs, Q13
const SKP_int16 SKP_Silk_Dec_B_HP_12[ DEC_HP_ORDER + 1 ] = {8000, -16000, 8000};
       // second order MA coefs, Q13

/* Decoder high-pass filter coefficients for 8 kHz sampling, -6 dB @ 43 Hz */
const SKP_int16 SKP_Silk_Dec_A_HP_8[ DEC_HP_ORDER     ] = {-15885, 7710};
       // second order AR coefs, Q13
const SKP_int16 SKP_Silk_Dec_B_HP_8[ DEC_HP_ORDER + 1 ] = {8000, -16000, 8000};
       // second order MA coefs, Q13

/* table for LSB coding */
const SKP_uint16 SKP_Silk_lsb_CDF[ 3 ] = {0,  40000,  65535};

/* tables for LTPScale */
const SKP_uint16 SKP_Silk_LTPscale_CDF[ 4 ] = {0,  32000,  48000,  65535};
const SKP_int    SKP_Silk_LTPscale_offset   = 2;

/* tables for VAD flag */
const SKP_uint16 SKP_Silk_vadflag_CDF[ 3 ] = {0,  22000,  65535}; // 66% for spee
ch, 33% for no speech
const SKP_int    SKP_Silk_vadflag_offset   = 1;

/* tables for sampling rate */
const SKP_int    SKP_Silk_SamplingRates_table[ 4 ] = {8, 12, 16, 24};
const SKP_uint16 SKP_Silk_SamplingRates_CDF[ 5 ]   = {0,  16000,  32000,  48000,
 65535};
const SKP_int    SKP_Silk_SamplingRates_offset     = 2;

/* tables for NLSF interpolation factor */
const SKP_uint16 SKP_Silk_NLSF_interpolation_factor_CDF[ 6 ] = {0,   3706,   8703
,  19226,  30926,  65535};
```

```
const SKP_int    SKP_Silk_NLSF_interpolation_factor_offset   = 4;

/* Table for frame termination indication */
const SKP_uint16 SKP_Silk_FrameTermination_CDF[ 5 ] = {0, 20000, 45000, 56000, 65
535};
const SKP_int    SKP_Silk_FrameTermination_offset   = 2;

const SKP_uint16 SKP_Silk_FrameTermination_v4_CDF[ 6 ] = {0,   13107,   26214,
39321,   52428,   65535};
const SKP_int    SKP_Silk_FrameTermination_v4_offset   = 4;

/* Table for random seed */
const SKP_uint16 SKP_Silk_Seed_CDF[ 5 ] = {0, 16384, 32768, 49152, 65535};
const SKP_int    SKP_Silk_Seed_offset   = 2;

/* Quantization offsets */
const SKP_int16  SKP_Silk_Quantization_Offsets_Q10[ 2 ][ 2 ] = {
    { OFFSET_VL_Q10, OFFSET_VH_Q10 }, { OFFSET_UVL_Q10, OFFSET_UVH_Q10 }
};

/* Table for LTPScale */
const SKP_int16 SKP_Silk_LTPScales_table_Q14[ 3 ] = { 15565, 11469, 8192 };

#if SWITCH_TRANSITION_FILTERING
/*  Elliptic/Cauer filters designed with 0.1 dB passband ripple,
        80 dB minimum stopband attenuation, and
        [0.95 : 0.15 : 0.35] normalized cut off frequencies. */

/* Interpolation points for filter coefficients used in the bandwidth transition
smoother */
const SKP_int32 SKP_Silk_Transition_LP_B_Q28[ TRANSITION_INT_NUM ][ TRANSITION_NB
 ] =
{
{    250767114,  501534038,  250767114  },
{    209867381,  419732057,  209867381  },
{    170987846,  341967853,  170987846  },
{    131531482,  263046905,  131531482  },
{     89306658,  178584282,   89306658  }
};

/* Interpolation points for filter coefficients used in the bandwidth transition
smoother */
const SKP_int32 SKP_Silk_Transition_LP_A_Q28[ TRANSITION_INT_NUM ][ TRANSITION_NA
 ] =
{
{    506393414,  239854379  },
{    411067935,  169683996  },
{    306733530,  116694253  },
{    185807084,   77959395  },
{     35497197,   57401098  }
};
#endif

#ifdef __cplusplus
```

```
}
#endif
```

A.133. src/SKP_Silk_tables_pitch_lag.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_tables.h"

const SKP_uint16 SKP_Silk_pitch_lag_NB_CDF[ 8 * ( PITCH_EST_MAX_LAG_MS - PITCH_ES
T_MIN_LAG_MS ) + 2 ] = {
         0,    194,    395,    608,    841,   1099,   1391,   1724,
      2105,   2544,   3047,   3624,   4282,   5027,   5865,   6799,
      7833,   8965,  10193,  11510,  12910,  14379,  15905,  17473,
     19065,  20664,  22252,  23814,  25335,  26802,  28206,  29541,
     30803,  31992,  33110,  34163,  35156,  36098,  36997,  37861,
     38698,  39515,  40319,  41115,  41906,  42696,  43485,  44273,
     45061,  45847,  46630,  47406,  48175,  48933,  49679,  50411,
     51126,  51824,  52502,  53161,  53799,  54416,  55011,  55584,
     56136,  56666,  57174,  57661,  58126,  58570,  58993,  59394,
     59775,  60134,  60472,  60790,  61087,  61363,  61620,  61856,
```

```
     62075,  62275,  62458,  62625,  62778,  62918,  63045,  63162,
     63269,  63368,  63459,  63544,  63623,  63698,  63769,  63836,
     63901,  63963,  64023,  64081,  64138,  64194,  64248,  64301,
     64354,  64406,  64457,  64508,  64558,  64608,  64657,  64706,
     64754,  64803,  64851,  64899,  64946,  64994,  65041,  65088,
     65135,  65181,  65227,  65272,  65317,  65361,  65405,  65449,
     65492,  65535
};

const SKP_int SKP_Silk_pitch_lag_NB_CDF_offset = 43;

const SKP_uint16 SKP_Silk_pitch_contour_NB_CDF[ 12 ] = {
         0,  14445,  18587,  25628,  30013,  34859,  40597,  48426,
     54460,  59033,  62990,  65535
};

const SKP_int SKP_Silk_pitch_contour_NB_CDF_offset = 5;

const SKP_uint16 SKP_Silk_pitch_lag_MB_CDF[ 12 * ( PITCH_EST_MAX_LAG_MS - PITCH_E
ST_MIN_LAG_MS ) + 2 ] = {
         0,    132,    266,    402,    542,    686,    838,    997,
      1167,   1349,   1546,   1760,   1993,   2248,   2528,   2835,
      3173,   3544,   3951,   4397,   4882,   5411,   5984,   6604,
      7270,   7984,   8745,   9552,  10405,  11300,  12235,  13206,
     14209,  15239,  16289,  17355,  18430,  19507,  20579,  21642,
     22688,  23712,  24710,  25677,  26610,  27507,  28366,  29188,
     29971,  30717,  31427,  32104,  32751,  33370,  33964,  34537,
     35091,  35630,  36157,  36675,  37186,  37692,  38195,  38697,
     39199,  39701,  40206,  40713,  41222,  41733,  42247,  42761,
     43277,  43793,  44309,  44824,  45336,  45845,  46351,  46851,
     47347,  47836,  48319,  48795,  49264,  49724,  50177,  50621,
     51057,  51484,  51902,  52312,  52714,  53106,  53490,  53866,
     54233,  54592,  54942,  55284,  55618,  55944,  56261,  56571,
     56873,  57167,  57453,  57731,  58001,  58263,  58516,  58762,
     58998,  59226,  59446,  59656,  59857,  60050,  60233,  60408,
     60574,  60732,  60882,  61024,  61159,  61288,  61410,  61526,
     61636,  61742,  61843,  61940,  62033,  62123,  62210,  62293,
     62374,  62452,  62528,  62602,  62674,  62744,  62812,  62879,
     62945,  63009,  63072,  63135,  63196,  63256,  63316,  63375,
     63434,  63491,  63549,  63605,  63661,  63717,  63772,  63827,
     63881,  63935,  63988,  64041,  64094,  64147,  64199,  64252,
     64304,  64356,  64409,  64461,  64513,  64565,  64617,  64669,
     64721,  64773,  64824,  64875,  64925,  64975,  65024,  65072,
     65121,  65168,  65215,  65262,  65308,  65354,  65399,  65445,
     65490,  65535
};

const SKP_int SKP_Silk_pitch_lag_MB_CDF_offset = 64;
```

```
const SKP_uint16 SKP_Silk_pitch_lag_WB_CDF[ 16 * ( PITCH_EST_MAX_LAG_MS - PITCH_E
ST_MIN_LAG_MS ) + 2 ] = {
         0,    106,    213,    321,    429,    539,    651,    766,
       884,   1005,   1132,   1264,   1403,   1549,   1705,   1870,
      2047,   2236,   2439,   2658,   2893,   3147,   3420,   3714,
      4030,   4370,   4736,   5127,   5546,   5993,   6470,   6978,
      7516,   8086,   8687,   9320,   9985,  10680,  11405,  12158,
     12938,  13744,  14572,  15420,  16286,  17166,  18057,  18955,
     19857,  20759,  21657,  22547,  23427,  24293,  25141,  25969,
     26774,  27555,  28310,  29037,  29736,  30406,  31048,  31662,
     32248,  32808,  33343,  33855,  34345,  34815,  35268,  35704,
     36127,  36537,  36938,  37330,  37715,  38095,  38471,  38844,
     39216,  39588,  39959,  40332,  40707,  41084,  41463,  41844,
     42229,  42615,  43005,  43397,  43791,  44186,  44583,  44982,
     45381,  45780,  46179,  46578,  46975,  47371,  47765,  48156,
     48545,  48930,  49312,  49690,  50064,  50433,  50798,  51158,
     51513,  51862,  52206,  52544,  52877,  53204,  53526,  53842,
     54152,  54457,  54756,  55050,  55338,  55621,  55898,  56170,
     56436,  56697,  56953,  57204,  57449,  57689,  57924,  58154,
     58378,  58598,  58812,  59022,  59226,  59426,  59620,  59810,
     59994,  60173,  60348,  60517,  60681,  60840,  60993,  61141,
     61284,  61421,  61553,  61679,  61800,  61916,  62026,  62131,
     62231,  62326,  62417,  62503,  62585,  62663,  62737,  62807,
     62874,  62938,  62999,  63057,  63113,  63166,  63217,  63266,
     63314,  63359,  63404,  63446,  63488,  63528,  63567,  63605,
     63642,  63678,  63713,  63748,  63781,  63815,  63847,  63879,
     63911,  63942,  63973,  64003,  64033,  64063,  64092,  64121,
     64150,  64179,  64207,  64235,  64263,  64291,  64319,  64347,
     64374,  64401,  64428,  64455,  64481,  64508,  64534,  64560,
     64585,  64610,  64635,  64660,  64685,  64710,  64734,  64758,
     64782,  64807,  64831,  64855,  64878,  64902,  64926,  64950,
     64974,  64998,  65022,  65045,  65069,  65093,  65116,  65139,
     65163,  65186,  65209,  65231,  65254,  65276,  65299,  65321,
     65343,  65364,  65386,  65408,  65429,  65450,  65471,  65493,
     65514,  65535
};

const SKP_int SKP_Silk_pitch_lag_WB_CDF_offset = 86;


const SKP_uint16 SKP_Silk_pitch_lag_SWB_CDF[ 24 * ( PITCH_EST_MAX_LAG_MS - PITCH_
EST_MIN_LAG_MS ) + 2 ] = {
         0,    253,    505,    757,   1008,   1258,   1507,   1755,
      2003,   2249,   2494,   2738,   2982,   3225,   3469,   3713,
      3957,   4202,   4449,   4698,   4949,   5203,   5460,   5720,
      5983,   6251,   6522,   6798,   7077,   7361,   7650,   7942,
      8238,   8539,   8843,   9150,   9461,   9775,  10092,  10411,
     10733,  11057,  11383,  11710,  12039,  12370,  12701,  13034,
     13368,  13703,  14040,  14377,  14716,  15056,  15398,  15742,
     16087,  16435,  16785,  17137,  17492,  17850,  18212,  18577,
```

```
    18946,  19318,  19695,  20075,  20460,  20849,  21243,  21640,
    22041,  22447,  22856,  23269,  23684,  24103,  24524,  24947,
    25372,  25798,  26225,  26652,  27079,  27504,  27929,  28352,
    28773,  29191,  29606,  30018,  30427,  30831,  31231,  31627,
    32018,  32404,  32786,  33163,  33535,  33902,  34264,  34621,
    34973,  35320,  35663,  36000,  36333,  36662,  36985,  37304,
    37619,  37929,  38234,  38535,  38831,  39122,  39409,  39692,
    39970,  40244,  40513,  40778,  41039,  41295,  41548,  41796,
    42041,  42282,  42520,  42754,  42985,  43213,  43438,  43660,
    43880,  44097,  44312,  44525,  44736,  44945,  45153,  45359,
    45565,  45769,  45972,  46175,  46377,  46578,  46780,  46981,
    47182,  47383,  47585,  47787,  47989,  48192,  48395,  48599,
    48804,  49009,  49215,  49422,  49630,  49839,  50049,  50259,
    50470,  50682,  50894,  51107,  51320,  51533,  51747,  51961,
    52175,  52388,  52601,  52813,  53025,  53236,  53446,  53655,
    53863,  54069,  54274,  54477,  54679,  54879,  55078,  55274,
    55469,  55662,  55853,  56042,  56230,  56415,  56598,  56779,
    56959,  57136,  57311,  57484,  57654,  57823,  57989,  58152,
    58314,  58473,  58629,  58783,  58935,  59084,  59230,  59373,
    59514,  59652,  59787,  59919,  60048,  60174,  60297,  60417,
    60533,  60647,  60757,  60865,  60969,  61070,  61167,  61262,
    61353,  61442,  61527,  61609,  61689,  61765,  61839,  61910,
    61979,  62045,  62109,  62170,  62230,  62287,  62343,  62396,
    62448,  62498,  62547,  62594,  62640,  62685,  62728,  62770,
    62811,  62852,  62891,  62929,  62967,  63004,  63040,  63075,
    63110,  63145,  63178,  63212,  63244,  63277,  63308,  63340,
    63371,  63402,  63432,  63462,  63491,  63521,  63550,  63578,
    63607,  63635,  63663,  63690,  63718,  63744,  63771,  63798,
    63824,  63850,  63875,  63900,  63925,  63950,  63975,  63999,
    64023,  64046,  64069,  64092,  64115,  64138,  64160,  64182,
    64204,  64225,  64247,  64268,  64289,  64310,  64330,  64351,
    64371,  64391,  64411,  64431,  64450,  64470,  64489,  64508,
    64527,  64545,  64564,  64582,  64600,  64617,  64635,  64652,
    64669,  64686,  64702,  64719,  64735,  64750,  64766,  64782,
    64797,  64812,  64827,  64842,  64857,  64872,  64886,  64901,
    64915,  64930,  64944,  64959,  64974,  64988,  65003,  65018,
    65033,  65048,  65063,  65078,  65094,  65109,  65125,  65141,
    65157,  65172,  65188,  65204,  65220,  65236,  65252,  65268,
    65283,  65299,  65314,  65330,  65345,  65360,  65375,  65390,
    65405,  65419,  65434,  65449,  65463,  65477,  65492,  65506,
    65521,  65535
};

const SKP_int SKP_Silk_pitch_lag_SWB_CDF_offset = 128;


const SKP_uint16 SKP_Silk_pitch_contour_CDF[ 35 ] = {
         0,    372,    843,   1315,   1836,   2644,   3576,   4719,
```

```
     6088,   7621,   9396,  11509,  14245,  17618,  20777,  24294,
    27992,  33116,  40100,  44329,  47558,  50679,  53130,  55557,
    57510,  59022,  60285,  61345,  62316,  63140,  63762,  64321,
    64729,  65099,  65535
};

const SKP_int SKP_Silk_pitch_contour_CDF_offset = 17;

const SKP_uint16 SKP_Silk_pitch_delta_CDF[23] = {
        0,    343,    740,   1249,   1889,   2733,   3861,   5396,
     7552,  10890,  16053,  24152,  30220,  34680,  37973,  40405,
    42243,  43708,  44823,  45773,  46462,  47055,  65535
};

const SKP_int SKP_Silk_pitch_delta_CDF_offset = 11;
```

A.134. src/SKP_Silk_tables_pulses_per_block.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_tables.h"
```

```
const SKP_int SKP_Silk_max_pulses_table[ 4 ] = {
        6,      8,     12,     18
};

const SKP_uint16 SKP_Silk_pulses_per_block_CDF[ 10 ][ 21 ] =
{
{
        0,  47113,  61501,  64590,  65125,  65277,  65352,  65407,
    65450,  65474,  65488,  65501,  65508,  65514,  65516,  65520,
    65521,  65523,  65524,  65526,  65535
},
{
        0,  26368,  47760,  58803,  63085,  64567,  65113,  65333,
    65424,  65474,  65498,  65511,  65517,  65520,  65523,  65525,
    65526,  65528,  65529,  65530,  65535
},
{
        0,   9601,  28014,  45877,  57210,  62560,  64611,  65260,
    65447,  65500,  65511,  65519,  65521,  65525,  65526,  65529,
    65530,  65531,  65532,  65534,  65535
},
{
        0,   3351,  12462,  25972,  39782,  50686,  57644,  61525,
    63521,  64506,  65009,  65255,  65375,  65441,  65471,  65488,
    65497,  65505,  65509,  65512,  65535
},
{
        0,    488,   2944,   9295,  19712,  32160,  43976,  53121,
    59144,  62518,  64213,  65016,  65346,  65470,  65511,  65515,
    65525,  65529,  65531,  65534,  65535
},
{
        0,  17013,  30405,  40812,  48142,  53466,  57166,  59845,
    61650,  62873,  63684,  64223,  64575,  64811,  64959,  65051,
    65111,  65143,  65165,  65183,  65535
},
{
        0,   2994,   8323,  15845,  24196,  32300,  39340,  45140,
    49813,  53474,  56349,  58518,  60167,  61397,  62313,  62969,
    63410,  63715,  63906,  64056,  65535
},
{
        0,     88,    721,   2795,   7542,  14888,  24420,  34593,
    43912,  51484,  56962,  60558,  62760,  64037,  64716,  65069,
    65262,  65358,  65398,  65420,  65535
},
{
        0,    287,    789,   2064,   4398,   8174,  13534,  20151,
```

```
    27347,  34533,  41295,  47242,  52070,  55772,  58458,  60381,
    61679,  62533,  63109,  63519,  65535
},
{
        0,      1,      3,     91,   4521,  14708,  28329,  41955,
    52116,  58375,  61729,  63534,  64459,  64924,  65092,  65164,
    65182,  65198,  65203,  65211,  65535
}
};

const SKP_int SKP_Silk_pulses_per_block_CDF_offset = 6;


const SKP_int16 SKP_Silk_pulses_per_block_BITS_Q6[ 9 ][ 20 ] =
{
{
       30,    140,    282,    444,    560,    625,    654,    677,
      731,    780,    787,    844,    859,    960,    896,   1024,
      960,   1024,    960,    821
},
{
       84,    103,    164,    252,    350,    442,    526,    607,
      663,    731,    787,    859,    923,    923,    960,   1024,
      960,   1024,   1024,    875
},
{
      177,    117,    120,    162,    231,    320,    426,    541,
      657,    803,    832,    960,    896,   1024,    923,   1024,
     1024,   1024,    960,   1024
},
{
      275,    182,    146,    144,    166,    207,    261,    322,
      388,    450,    516,    582,    637,    710,    762,    821,
      832,    896,    923,    734
},
{
      452,    303,    216,    170,    153,    158,    182,    220,
      274,    337,    406,    489,    579,    681,    896,    811,
      896,    960,    923,   1024
},
{
      125,    147,    170,    202,    232,    265,    295,    332,
      368,    406,    443,    483,    520,    563,    606,    646,
      704,    739,    757,    483
},
{
      285,    232,    200,    190,    193,    206,    224,    244,
      266,    289,    315,    340,    367,    394,    425,    462,
```

```
       496,    539,    561,    350
},
{
       611,    428,    319,    242,    202,    178,    172,    180,
       199,    229,    268,    313,    364,    422,    482,    538,
       603,    683,    739,    586
},
{
       501,    450,    364,    308,    264,    231,    212,    204,
       204,    210,    222,    241,    265,    295,    326,    362,
       401,    437,    469,    321
}
};

const SKP_uint16 SKP_Silk_rate_levels_CDF[ 2 ][ 10 ] =
{
{
         0,   2005,  12717,  20281,  31328,  36234,  45816,  57753,
     63104,  65535
},
{
         0,   8553,  23489,  36031,  46295,  53519,  56519,  59151,
     64185,  65535
}
};

const SKP_int SKP_Silk_rate_levels_CDF_offset = 4;


const SKP_int16 SKP_Silk_rate_levels_BITS_Q6[ 2 ][ 9 ] =
{
{
       322,    167,    199,    164,    239,    178,    157,    231,
       304
},
{
       188,    137,    153,    171,    204,    285,    297,    237,
       358
}
};

const SKP_uint16 SKP_Silk_shell_code_table0[ 33 ] = {
         0,  32748,  65535,      0,   9505,  56230,  65535,      0,
      4093,  32204,  61720,  65535,      0,   2285,  16207,  48750,
     63424,  65535,      0,   1709,   9446,  32026,  55752,  63876,
     65535,      0,   1623,   6986,  21845,  45381,  59147,  64186,
     65535
};
```

```
const SKP_uint16 SKP_Silk_shell_code_table1[ 52 ] = {
         0,  32691,  65535,      0,  12782,  52752,  65535,      0,
      4847,  32665,  60899,  65535,      0,   2500,  17305,  47989,
     63369,  65535,      0,   1843,  10329,  32419,  55433,  64277,
     65535,      0,   1485,   7062,  21465,  43414,  59079,  64623,
     65535,      0,      0,   4841,  14797,  31799,  49667,  61309,
     65535,  65535,      0,      0,      0,   8032,  21695,  41078,
     56317,  65535,  65535,  65535
};

const SKP_uint16 SKP_Silk_shell_code_table2[ 102 ] = {
         0,  32615,  65535,      0,  14447,  50912,  65535,      0,
      6301,  32587,  59361,  65535,      0,   3038,  18640,  46809,
     62852,  65535,      0,   1746,  10524,  32509,  55273,  64278,
     65535,      0,   1234,   6360,  21259,  43712,  59651,  64805,
     65535,      0,   1020,   4461,  14030,  32286,  51249,  61904,
     65100,  65535,      0,    851,   3435,  10006,  23241,  40797,
     55444,  63009,  65252,  65535,      0,      0,   2075,   7137,
     17119,  31499,  46982,  58723,  63976,  65535,  65535,      0,
         0,      0,   3820,  11572,  23038,  37789,  51969,  61243,
     65535,  65535,  65535,      0,      0,      0,      0,   6882,
     16828,  30444,  44844,  57365,  65535,  65535,  65535,  65535,
         0,      0,      0,      0,      0,  10093,  22963,  38779,
     54426,  65535,  65535,  65535,  65535,  65535
};

const SKP_uint16 SKP_Silk_shell_code_table3[ 207 ] = {
         0,  32324,  65535,      0,  15328,  49505,  65535,      0,
      7474,  32344,  57955,  65535,      0,   3944,  19450,  45364,
     61873,  65535,      0,   2338,  11698,  32435,  53915,  63734,
     65535,      0,   1506,   7074,  21778,  42972,  58861,  64590,
     65535,      0,   1027,   4490,  14383,  32264,  50980,  61712,
     65043,  65535,      0,    760,   3022,   9696,  23264,  41465,
     56181,  63253,  65251,  65535,      0,    579,   2256,   6873,
     16661,  31951,  48250,  59403,  64198,  65360,  65535,      0,
       464,   1783,   5181,  12269,  24247,  39877,  53490,  61502,
     64591,  65410,  65535,      0,    366,   1332,   3880,   9273,
     18585,  32014,  45928,  56659,  62616,  64899,  65483,  65535,
         0,    286,   1065,   3089,   6969,  14148,  24859,  38274,
     50715,  59078,  63448,  65091,  65481,  65535,      0,      0,
       482,   2010,   5302,  10408,  18988,  30698,  43634,  54233,
     60828,  64119,  65288,  65535,  65535,      0,      0,      0,
      1006,   3531,   7857,  14832,  24543,  36272,  47547,  56883,
     62327,  64746,  65535,  65535,  65535,      0,      0,      0,
         0,   1863,   4950,  10730,  19284,  29397,  41382,  52335,
     59755,  63834,  65535,  65535,  65535,  65535,      0,      0,
         0,      0,      0,   2513,   7290,  14487,  24275,  35312,
     46240,  55841,  62007,  65535,  65535,  65535,  65535,  65535,
```

```
        0,      0,      0,      0,      0,      0,   3606,   9573,
    18764,  28667,  40220,  51290,  59924,  65535,  65535,  65535,
    65535,  65535,  65535,      0,      0,      0,      0,      0,
        0,      0,   4879,  13091,  23376,  36061,  49395,  59315,
    65535,  65535,  65535,  65535,  65535,  65535,  65535
};

const SKP_uint16 SKP_Silk_shell_code_table_offsets[ 19 ] = {
        0,      0,      3,      7,     12,     18,     25,     33,
       42,     52,     63,     75,     88,    102,    117,    133,
      150,    168,    187
};
```

A.135. src/SKP_Silk_tables_sign.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_tables.h"

const SKP_uint16 SKP_Silk_sign_CDF[ 36 ][ 3 ] =
```

```
{
{
        0,  37840,  65535
},
{
        0,  36944,  65535
},
{
        0,  36251,  65535
},
{
        0,  35304,  65535
},
{
        0,  34715,  65535
},
{
        0,  35503,  65535
},
{
        0,  34529,  65535
},
{
        0,  34296,  65535
},
{
        0,  34016,  65535
},
{
        0,  47659,  65535
},
{
        0,  44945,  65535
},
{
        0,  42503,  65535
},
{
        0,  40235,  65535
},
{
        0,  38569,  65535
},
{
        0,  40254,  65535
},
{
        0,  37851,  65535
```

```
},
{
        0,  37243,  65535
},
{
        0,  36595,  65535
},
{
        0,  43410,  65535
},
{
        0,  44121,  65535
},
{
        0,  43127,  65535
},
{
        0,  40978,  65535
},
{
        0,  38845,  65535
},
{
        0,  40433,  65535
},
{
        0,  38252,  65535
},
{
        0,  37795,  65535
},
{
        0,  36637,  65535
},
{
        0,  59159,  65535
},
{
        0,  55630,  65535
},
{
        0,  51806,  65535
},
{
        0,  48073,  65535
},
{
        0,  45036,  65535
```

```
},
{
        0,  48416,  65535
},
{
        0,  43857,  65535
},
{
        0,  42678,  65535
},
{
        0,  41146,  65535
}
};
```

A.136. src/SKP_Silk_tables_type_offset.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_tables.h"
```

```
const SKP_uint16 SKP_Silk_type_offset_CDF[ 5 ] = {
        0,  37522,  41030,  44212,  65535
};

const SKP_int SKP_Silk_type_offset_CDF_offset = 2;


const SKP_uint16 SKP_Silk_type_offset_joint_CDF[ 4 ][ 5 ] =
{
{
        0,  57686,  61230,  62358,  65535
},
{
        0,  18346,  40067,  43659,  65535
},
{
        0,  22694,  24279,  35507,  65535
},
{
        0,   6067,   7215,  13010,  65535
}
};
```

A.137. src/SKP_Silk_VAD.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
```

```
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*
 * File Name:   SKP_Silk_VAD.c
 * Description: Silk VAD.
 */

#include <stdlib.h>
#include "SKP_Silk_main.h"

/**********************************/
/* Initialization of the Silk VAD */
/**********************************/
SKP_int SKP_Silk_VAD_Init(                              /* O    Return value, 0 i
f success                   */
    SKP_Silk_VAD_state              *psSilk_VAD         /* I/O  Pointer to Silk V
AD state                    */
)
{
    SKP_int b, ret = 0;

    /* reset state memory */
    SKP_memset( psSilk_VAD, 0, sizeof( SKP_Silk_VAD_state ) );

    /* init noise levels */
    /* Initialize array with approx pink noise levels (psd proportional to invers
e of frequency) */
    for( b = 0; b < VAD_N_BANDS; b++ ) {
        psSilk_VAD->NoiseLevelBias[ b ] = SKP_max_32( SKP_DIV32_16( VAD_NOISE_LEV
ELS_BIAS, b + 1 ), 1 );
    }

    /* Initialize state */
    for( b = 0; b < VAD_N_BANDS; b++ ) {
        psSilk_VAD->NL[ b ]     = SKP_MUL( 100, psSilk_VAD->NoiseLevelBias[ b ] )
;
        psSilk_VAD->inv_NL[ b ] = SKP_DIV32( SKP_int32_MAX, psSilk_VAD->NL[ b ] )
;
    }
    psSilk_VAD->counter = 15;

    /* init smoothed energy-to-noise ratio*/
    for( b = 0; b < VAD_N_BANDS; b++ ) {
        psSilk_VAD->NrgRatioSmth_Q8[ b ] = 100 * 256;       /* 100 * 256 --> 20 d
B SNR */
    }

    return( ret );
}
```

```
/* Weighting factors for tilt measure */
const static SKP_int32 tiltWeights[ VAD_N_BANDS ] = { 30000, 6000, -12000, -12000
 };

/***************************************/
/* Get the speech activity level in Q8 */
/***************************************/
SKP_int SKP_Silk_VAD_GetSA_Q8(                                  /* O    Retur
n value, 0 if success      */
    SKP_Silk_VAD_state              *psSilk_VAD,                /* I/O  Silk
VAD state                  */
    SKP_int                         *pSA_Q8,                    /* O    Speec
h activity level in Q8     */
    SKP_int                         *pSNR_dB_Q7,                /* O    SNR f
or current frame in Q7     */
    SKP_int                         pQuality_Q15[ VAD_N_BANDS ],/* O    Smoot
hed SNR for each band      */
    SKP_int                         *pTilt_Q15,                 /* O    curre
nt frame's frequency tilt  */
    const SKP_int16                 pIn[],                      /* I    PCM i
nput       [framelength]   */
    const SKP_int                   framelength                 /* I    Input
 frame length              */
)
{
    SKP_int   SA_Q15, input_tilt;
    SKP_int32 scratch[ 3 * MAX_FRAME_LENGTH / 2 ];
    SKP_int   decimated_framelength, dec_subframe_length, dec_subframe_offset, SN
R_Q7, i, b, s;
    SKP_int32 sumSquared, smooth_coef_Q16;
    SKP_int16 HPstateTmp;

    SKP_int16 X[ VAD_N_BANDS ][ MAX_FRAME_LENGTH / 2 ];
    SKP_int32 Xnrg[ VAD_N_BANDS ];
    SKP_int32 NrgToNoiseRatio_Q8[ VAD_N_BANDS ];
    SKP_int32 speech_nrg, x_tmp;
    SKP_int   ret = 0;

    /* Safety checks */
    SKP_assert( VAD_N_BANDS == 4 );
    SKP_assert( MAX_FRAME_LENGTH >= framelength );
    SKP_assert( framelength <= 512 );

    /***********************/
    /* Filter and Decimate */
    /***********************/
    /* 0-8 kHz to 0-4 kHz and 4-8 kHz */
    SKP_Silk_ana_filt_bank_1( pIn,          &psSilk_VAD->AnaState[  0 ], &X[ 0 ][
 0 ], &X[ 3 ][ 0 ], &scratch[ 0 ], framelength );

    /* 0-4 kHz to 0-2 kHz and 2-4 kHz */
    SKP_Silk_ana_filt_bank_1( &X[ 0 ][ 0 ], &psSilk_VAD->AnaState1[ 0 ], &X[ 0 ][
 0 ], &X[ 2 ][ 0 ], &scratch[ 0 ], SKP_RSHIFT( framelength, 1 ) );

    /* 0-2 kHz to 0-1 kHz and 1-2 kHz */
    SKP_Silk_ana_filt_bank_1( &X[ 0 ][ 0 ], &psSilk_VAD->AnaState2[ 0 ], &X[ 0 ][
 0 ], &X[ 1 ][ 0 ], &scratch[ 0 ], SKP_RSHIFT( framelength, 2 ) );

    /*********************************************/
    /* HP filter on lowest band (differentiator) */
    /*********************************************/
```

```
    decimated_framelength = SKP_RSHIFT( framelength, 3 );
    X[ 0 ][ decimated_framelength - 1 ] = SKP_RSHIFT( X[ 0 ][ decimated_frameleng
th - 1 ], 1 );
    HPstateTmp = X[ 0 ][ decimated_framelength - 1 ];
    for( i = decimated_framelength - 1; i > 0; i-- ) {
        X[ 0 ][ i - 1 ]  = SKP_RSHIFT( X[ 0 ][ i - 1 ], 1 );
        X[ 0 ][ i ]     -= X[ 0 ][ i - 1 ];
    }
    X[ 0 ][ 0 ] -= psSilk_VAD->HPstate;
    psSilk_VAD->HPstate = HPstateTmp;

    /************************************/
    /* Calculate the energy in each band */
    /************************************/
    for( b = 0; b < VAD_N_BANDS; b++ ) {
        /* Find the decimated framelength in the non-uniformly divided bands */
        decimated_framelength = SKP_RSHIFT( framelength, SKP_min_int( VAD_N_BANDS
 - b, VAD_N_BANDS - 1 ) );

        /* Split length into subframe lengths */
        dec_subframe_length = SKP_RSHIFT( decimated_framelength, VAD_INTERNAL_SUB
FRAMES_LOG2 );
        dec_subframe_offset = 0;

        /* Compute energy per sub-frame */
        /* initialize with summed energy of last subframe */
        Xnrg[ b ] = psSilk_VAD->XnrgSubfr[ b ];
        for( s = 0; s < VAD_INTERNAL_SUBFRAMES; s++ ) {
            sumSquared = 0;
            for( i = 0; i < dec_subframe_length; i++ ) {
                /* The energy will be less than dec_subframe_length * ( SKP_int16
_MIN / 8 )^2.               */
                /* Therefore we can accumulate with no risk of overflow (unless d
ec_subframe_length > 128)  */
                x_tmp = SKP_RSHIFT( X[ b ][ i + dec_subframe_offset ], 3 );
                sumSquared = SKP_SMLABB( sumSquared, x_tmp, x_tmp );

                /* Safety check */
                SKP_assert( sumSquared >= 0 );
            }

            /* add/saturate summed energy of current subframe */
            if( s < VAD_INTERNAL_SUBFRAMES - 1 ) {
                Xnrg[ b ] = SKP_ADD_POS_SAT32( Xnrg[ b ], sumSquared );
            } else {
                /* look-ahead subframe */
                Xnrg[ b ] = SKP_ADD_POS_SAT32( Xnrg[ b ], SKP_RSHIFT( sumSquared,
 1 ) );
            }

            dec_subframe_offset += dec_subframe_length;
        }
        psSilk_VAD->XnrgSubfr[ b ] = sumSquared;
    }
```

```
    /*********************/
    /* Noise estimation */
    /*********************/
    SKP_Silk_VAD_GetNoiseLevels( &Xnrg[ 0 ], psSilk_VAD );

    /***********************************************/
    /* Signal-plus-noise to noise ratio estimation */
    /***********************************************/
    sumSquared = 0;
    input_tilt = 0;
    for( b = 0; b < VAD_N_BANDS; b++ ) {
        speech_nrg = Xnrg[ b ] - psSilk_VAD->NL[ b ];
        if( speech_nrg > 0 ) {
            /* Divide, with sufficient resolution */
            if( ( Xnrg[ b ] & 0xFF800000 ) == 0 ) {
                NrgToNoiseRatio_Q8[ b ] = SKP_DIV32( SKP_LSHIFT( Xnrg[ b ], 8 ),
psSilk_VAD->NL[ b ] + 1 );
            } else {
                NrgToNoiseRatio_Q8[ b ] = SKP_DIV32( Xnrg[ b ], SKP_RSHIFT( psSil
k_VAD->NL[ b ], 8 ) + 1 );
            }

            /* Convert to log domain */
            SNR_Q7 = SKP_Silk_lin2log( NrgToNoiseRatio_Q8[ b ] ) - 8 * 128;

            /* Sum-of-squares */
            sumSquared = SKP_SMLABB( sumSquared, SNR_Q7, SNR_Q7 );          /* Q1
4 */

            /* Tilt measure */
            if( speech_nrg < ( 1 << 20 ) ) {
                /* Scale down SNR value for small subband speech energies */
                SNR_Q7 = SKP_SMULWB( SKP_LSHIFT( SKP_Silk_SQRT_APPROX( speech_nrg
 ), 6 ), SNR_Q7 );
            }
            input_tilt = SKP_SMLAWB( input_tilt, tiltWeights[ b ], SNR_Q7 );
        } else {
            NrgToNoiseRatio_Q8[ b ] = 256;
        }
    }

    /* Mean-of-squares */
    sumSquared = SKP_DIV32_16( sumSquared, VAD_N_BANDS );           /* Q14 */

    /* Root-mean-square approximation, scale to dBs, and write to output pointer
*/
    *pSNR_dB_Q7 = ( SKP_int16 )( 3 * SKP_Silk_SQRT_APPROX( sumSquared ) );  /* Q7
 */

    /*********************************/
    /* Speech Probability Estimation */
    /*********************************/
    SA_Q15 = SKP_Silk_sigm_Q15( SKP_SMULWB( VAD_SNR_FACTOR_Q16, *pSNR_dB_Q7 ) - V
AD_NEGATIVE_OFFSET_Q5 );
```

```
    /*************************/
    /* Frequency Tilt Measure */
    /*************************/
    *pTilt_Q15 = SKP_LSHIFT( SKP_Silk_sigm_Q15( input_tilt ) - 16384, 1 );

    /**********************************************/
    /* Scale the sigmoid output based on power levels */
    /**********************************************/
    speech_nrg = 0;
    for( b = 0; b < VAD_N_BANDS; b++ ) {
        /* Accumulate signal-without-noise energies, higher frequency bands have
more weight */
        speech_nrg += ( b + 1 ) * SKP_RSHIFT( Xnrg[ b ] - psSilk_VAD->NL[ b ], 4
);
    }

    /* Power scaling */
    if( speech_nrg <= 0 ) {
        SA_Q15 = SKP_RSHIFT( SA_Q15, 1 );
    } else if( speech_nrg < 32768 ) {
        /* square-root */
        speech_nrg = SKP_Silk_SQRT_APPROX( SKP_LSHIFT( speech_nrg, 15 ) );
        SA_Q15 = SKP_SMULWB( 32768 + speech_nrg, SA_Q15 );
    }

    /* Copy the resulting speech activity in Q8 to *pSA_Q8 */
    *pSA_Q8 = SKP_min_int( SKP_RSHIFT( SA_Q15, 7 ), SKP_uint8_MAX );

    /***********************************/
    /* Energy Level and SNR estimation */
    /***********************************/
    /* smoothing coefficient */
    smooth_coef_Q16 = SKP_SMULWB( VAD_SNR_SMOOTH_COEF_Q18, SKP_SMULWB( SA_Q15, SA
_Q15 ) );
    for( b = 0; b < VAD_N_BANDS; b++ ) {
        /* compute smoothed energy-to-noise ratio per band */
        psSilk_VAD->NrgRatioSmth_Q8[ b ] = SKP_SMLAWB( psSilk_VAD->NrgRatioSmth_Q
8[ b ],
            NrgToNoiseRatio_Q8[ b ] - psSilk_VAD->NrgRatioSmth_Q8[ b ], smooth_co
ef_Q16 );

        /* signal to noise ratio in dB per band */
        SNR_Q7 = 3 * ( SKP_Silk_lin2log( psSilk_VAD->NrgRatioSmth_Q8[b] ) - 8 * 1
28 );
        /* quality = sigmoid( 0.25 * ( SNR_dB - 16 ) ); */
        pQuality_Q15[ b ] = SKP_Silk_sigm_Q15( SKP_RSHIFT( SNR_Q7 - 16 * 128, 4 )
 );
    }

    return( ret );
}

/*************************/
/* Noise level estimation */
/*************************/
```

```
void SKP_Silk_VAD_GetNoiseLevels(
    const SKP_int32                 pX[ VAD_N_BANDS ],  /* I    subband energies
                        */
    SKP_Silk_VAD_state              *psSilk_VAD         /* I/O  Pointer to Silk V
AD state                    */
)
{
    SKP_int   k;
    SKP_int32 nl, nrg, inv_nrg;
    SKP_int   coef, min_coef;

    /* Initially faster smoothing */
    if( psSilk_VAD->counter < 1000 ) { /* 1000 = 20 sec */
        min_coef = SKP_DIV32_16( SKP_int16_MAX, SKP_RSHIFT( psSilk_VAD->counter,
4 ) + 1 );
    } else {
        min_coef = 0;
    }

    for( k = 0; k < VAD_N_BANDS; k++ ) {
        /* Get old noise level estimate for current band */
        nl = psSilk_VAD->NL[ k ];
        SKP_assert( nl >= 0 );

        /* Add bias */
        nrg = SKP_ADD_POS_SAT32( pX[ k ], psSilk_VAD->NoiseLevelBias[ k ] );
        SKP_assert( nrg > 0 );

        /* Invert energies */
        inv_nrg = SKP_DIV32( SKP_int32_MAX, nrg );
        SKP_assert( inv_nrg >= 0 );

        /* Less update when subband energy is high */
        if( nrg > SKP_LSHIFT( nl, 3 ) ) {
            coef = VAD_NOISE_LEVEL_SMOOTH_COEF_Q16 >> 3;
        } else if( nrg < nl ) {
            coef = VAD_NOISE_LEVEL_SMOOTH_COEF_Q16;
        } else {
            coef = SKP_SMULWB( SKP_SMULWW( inv_nrg, nl ), VAD_NOISE_LEVEL_SMOOTH_
COEF_Q16 << 1 );
        }

        /* Initially faster smoothing */
        coef = SKP_max_int( coef, min_coef );

        /* Smooth inverse energies */
        psSilk_VAD->inv_NL[ k ] = SKP_SMLAWB( psSilk_VAD->inv_NL[ k ], inv_nrg -
psSilk_VAD->inv_NL[ k ], coef );
        SKP_assert( psSilk_VAD->inv_NL[ k ] >= 0 );

        /* Compute noise level by inverting again */
        nl = SKP_DIV32( SKP_int32_MAX, psSilk_VAD->inv_NL[ k ] );
        SKP_assert( nl >= 0 );
```

```
        /* Limit noise levels (guarantee 7 bits of head room) */
        nl = SKP_min( nl, 0x00FFFFFF );

        /* Store as part of state */
        psSilk_VAD->NL[ k ] = nl;
    }

    /* Increment frame counter */
    psSilk_VAD->counter++;
}
```

A.138. src/SKP_Silk_VQ_nearest_neighbor_FIX.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

#include "SKP_Silk_main_FIX.h"

/* Entropy constrained MATRIX-weighted VQ, hard-coded to 5-element vectors, for a
 single input data vector */
void SKP_Silk_VQ_WMat_EC_FIX(
    SKP_int                         *ind,               /* O    index of best cod
ebook vector               */
    SKP_int32                       *rate_dist_Q14,     /* O    best weighted qua
ntization error + mu * rate*/
```

```
    const SKP_int16                 *in_Q14,            /* I    input vector to b
e quantized                 */
    const SKP_int32                 *W_Q18,             /* I    weighting matrix
                            */
    const SKP_int16                 *cb_Q14,            /* I    codebook
                            */
    const SKP_int16                 *cl_Q6,             /* I    code length for e
ach codebook vector         */
    const SKP_int                   mu_Q8,              /* I    tradeoff between
weighted error and rate     */
    SKP_int                         L                   /* I    number of vectors
 in codebook                */
)
{
    SKP_int   k;
    const SKP_int16 *cb_row_Q14;
    SKP_int32 sum1_Q14, sum2_Q16, diff_Q14_01, diff_Q14_23, diff_Q14_4;

    /* Loop over codebook */
    *rate_dist_Q14 = SKP_int32_MAX;
    cb_row_Q14 = cb_Q14;
    for( k = 0; k < L; k++ ) {
        /* Pack pairs of int16 values per int32 */
        diff_Q14_01 = (SKP_uint16)( in_Q14[ 0 ] - cb_row_Q14[ 0 ] ) | SKP_LSHIFT(
 ( SKP_int32 )in_Q14[ 1 ] - cb_row_Q14[ 1 ], 16 );
        diff_Q14_23 = (SKP_uint16)( in_Q14[ 2 ] - cb_row_Q14[ 2 ] ) | SKP_LSHIFT(
 ( SKP_int32 )in_Q14[ 3 ] - cb_row_Q14[ 3 ], 16 );
        diff_Q14_4  = in_Q14[ 4 ] - cb_row_Q14[ 4 ];

        /* Weighted rate */
        sum1_Q14 = SKP_SMULBB( mu_Q8, cl_Q6[ k ] );

        SKP_assert( sum1_Q14 >= 0 );

        /* Add weighted quantization error, assuming W_Q18 is symmetric */
        /* NOTE: the code below loads two int16 values as one int32, and multipli
es each using the  */
        /* SMLAWB and SMLAWT instructions. On a big-endian CPU the two int16 vari
ables would be     */
        /* loaded in reverse order and the code will give the wrong result. In th
at case swapping   */
        /* the SMLAWB and SMLAWT instructions should solve the problem.
                   */
        /* first row of W_Q18 */
        sum2_Q16 = SKP_SMULWT(           W_Q18[ 1 ], diff_Q14_01 );
        sum2_Q16 = SKP_SMLAWB( sum2_Q16, W_Q18[ 2 ], diff_Q14_23 );
        sum2_Q16 = SKP_SMLAWT( sum2_Q16, W_Q18[ 3 ], diff_Q14_23 );
        sum2_Q16 = SKP_SMLAWB( sum2_Q16, W_Q18[ 4 ], diff_Q14_4  );
        sum2_Q16 = SKP_LSHIFT( sum2_Q16, 1 );
        sum2_Q16 = SKP_SMLAWB( sum2_Q16, W_Q18[ 0 ], diff_Q14_01 );
        sum1_Q14 = SKP_SMLAWB( sum1_Q14, sum2_Q16,   diff_Q14_01 );

        /* second row of W_Q18 */
        sum2_Q16 = SKP_SMULWB(           W_Q18[ 7 ], diff_Q14_23 );
        sum2_Q16 = SKP_SMLAWT( sum2_Q16, W_Q18[ 8 ], diff_Q14_23 );
        sum2_Q16 = SKP_SMLAWB( sum2_Q16, W_Q18[ 9 ], diff_Q14_4  );
        sum2_Q16 = SKP_LSHIFT( sum2_Q16, 1 );
        sum2_Q16 = SKP_SMLAWT( sum2_Q16, W_Q18[ 6 ], diff_Q14_01 );
        sum1_Q14 = SKP_SMLAWT( sum1_Q14, sum2_Q16,   diff_Q14_01 );
```

```
        /* third row of W_Q18 */
        sum2_Q16 = SKP_SMULWT(           W_Q18[ 13 ], diff_Q14_23 );
        sum2_Q16 = SKP_SMLAWB( sum2_Q16, W_Q18[ 14 ], diff_Q14_4  );
        sum2_Q16 = SKP_LSHIFT( sum2_Q16, 1 );
        sum2_Q16 = SKP_SMLAWB( sum2_Q16, W_Q18[ 12 ], diff_Q14_23 );
        sum1_Q14 = SKP_SMLAWB( sum1_Q14, sum2_Q16,    diff_Q14_23 );

        /* fourth row of W_Q18 */
        sum2_Q16 = SKP_SMULWB(           W_Q18[ 19 ], diff_Q14_4  );
        sum2_Q16 = SKP_LSHIFT( sum2_Q16, 1 );
        sum2_Q16 = SKP_SMLAWT( sum2_Q16, W_Q18[ 18 ], diff_Q14_23 );
        sum1_Q14 = SKP_SMLAWT( sum1_Q14, sum2_Q16,    diff_Q14_23 );

        /* last row of W_Q18 */
        sum2_Q16 = SKP_SMULWB(           W_Q18[ 24 ], diff_Q14_4  );
        sum1_Q14 = SKP_SMLAWB( sum1_Q14, sum2_Q16,    diff_Q14_4  );

        SKP_assert( sum1_Q14 >= 0 );

        /* find best */
        if( sum1_Q14 < *rate_dist_Q14 ) {
            *rate_dist_Q14 = sum1_Q14;
            *ind = k;
        }

        /* Go to next cbk vector */
        cb_row_Q14 += LTP_ORDER;
    }
}
```

A.139. test/Decoder.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
```

```
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*****************************/
/* Silk decoder test program */
/*****************************/

#ifdef _WIN32
#define _CRT_SECURE_NO_DEPRECATE    1
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "SKP_Silk_SDK_API.h"
#include "SKP_Silk_SigProc_FIX.h"

/* Define codec specific settings should be moved to h file */
#define MAX_BYTES_PER_FRAME     1024
#define MAX_INPUT_FRAMES        5
#define MAX_FRAME_LENGTH        480
#define MAX_LBRR_DELAY          2

static void print_usage(char* argv[]) {
    printf( "\nusage: %s in.bit out.pcm [settings]\n", argv[ 0 ] );
    printf( "\nin.bit        : Bitstream input to decoder" );
    printf( "\nout.pcm       : Speech output from decoder" );
    printf( "\n   settings:" );
    printf( "\n-fs <kHz>     : Sampling rate of output signal in kHz; default: 24
" );
    printf( "\n-loss <perc>  : Simulated packet loss percentage (0-100); default:
 0" );
    printf( "\n" );
}

int main( int argc, char* argv[] )
{
    size_t    counter;
    SKP_int   args, totPackets, i, k;
    SKP_int16 ret, len, tot_len;
```

```
    SKP_int16 nBytes;
    SKP_uint8 payload[    MAX_BYTES_PER_FRAME * MAX_INPUT_FRAMES * ( MAX_LBRR_DEL
AY + 1 ) ];
    SKP_uint8 FECpayload[ MAX_BYTES_PER_FRAME * MAX_INPUT_FRAMES ], *payloadPtr;
    SKP_uint8 *payloadEnd = NULL, *payloadToDec = NULL;
    SKP_int16 nBytesFEC;
    SKP_int16 nBytesPerPacket[ MAX_LBRR_DELAY + 1 ], totBytes;
    SKP_int16 out[ ( MAX_FRAME_LENGTH << 1 ) * MAX_INPUT_FRAMES ], *outPtr;
    char      speechOutFileName[ 150 ], bitInFileName[ 150 ];
    FILE      *bitInFile, *speechOutFile;
    SKP_int   Fs_kHz = 0;
    SKP_int32 decSizeBytes;
    void      *psDec;
    float     loss_prob;
    SKP_int   frames, lost, quiet;
    SKP_SILK_SDK_DecControlStruct DecControl;

    if( argc < 3 ) {
        print_usage( argv );
        exit( 0 );
    }

    /* default settings */
    quiet     = 0;
    loss_prob = 0.0f;

    /* get arguments */
    args = 1;
    strcpy( bitInFileName, argv[ args ] );
    args++;
    strcpy( speechOutFileName, argv[ args ] );
    args++;
    while( args < argc ) {
        if( SKP_STR_CASEINSENSITIVE_COMPARE( argv[ args ], "-loss" ) == 0 ) {
            sscanf( argv[ args + 1 ], "%f", &loss_prob );
            args += 2;
        } else if( SKP_STR_CASEINSENSITIVE_COMPARE( argv[ args ], "-fs" ) == 0 )
{
            sscanf( argv[ args + 1 ], "%d", &Fs_kHz );
            args += 2;
        } else if( SKP_STR_CASEINSENSITIVE_COMPARE( argv[ args ], "-quiet" ) == 0
 ) {
            quiet = 1;
            args++;
        } else {
            printf( "Error: unrecognized setting: %s\n\n", argv[ args ] );
            print_usage( argv );
            exit( 0 );
        }
    }
```

```
    if( !quiet ) {
        printf("****************** Silk Decoder v %s ***************\n", SKP_Si
lk_SDK_get_version());
        printf("****************** Compiled for %d bit cpu ******** \n", (int)s
izeof(void*) * 8 );
        printf( "Input:                         %s\n", bitInFileName );
        printf( "Output:                        %s\n", speechOutFileName );
    }

    /* Open files */
    bitInFile = fopen( bitInFileName, "rb" );
    if( bitInFile == NULL ) {
        printf( "Error: could not open input file %s\n", bitInFileName );
        exit( 0 );
    }
    speechOutFile = fopen( speechOutFileName, "wb" );
    if( speechOutFile == NULL ) {
        printf( "Error: could not open output file %s\n", speechOutFileName );
        exit( 0 );
    }

    /* Set the samplingrate that is requested for the output */
    if( Fs_kHz == 0 ) {
        DecControl.sampleRate = 24000;
    } else {
        DecControl.sampleRate = Fs_kHz * 1000;
    }

    /* Create decoder */
    ret = SKP_Silk_SDK_Get_Decoder_Size( &decSizeBytes );
    if( ret ) {
        printf( "\nSKP_Silk_SDK_Get_Decoder_Size returned %d", ret );
    }
    psDec = malloc( decSizeBytes );

    /* Reset decoder */
    ret = SKP_Silk_SDK_InitDecoder( psDec );
    if( ret ) {
        printf( "\nSKP_Silk_InitDecoder returned %d", ret );
    }

    totPackets = 0;
    payloadEnd = payload;

    /* Simulate the jitter buffer holding MAX_FEC_DELAY packets */
    for( i = 0; i < MAX_LBRR_DELAY; i++ ) {
        /* Read payload size */
        counter = fread( &nBytes, sizeof( SKP_int16 ), 1, bitInFile );
        /* Read payload */
        counter = fread( payloadEnd, sizeof( SKP_uint8 ), nBytes, bitInFile );
```

```
        if( (SKP_int16)counter < nBytes ) {
            break;
        }
        nBytesPerPacket[ i ] = nBytes;
        payloadEnd          += nBytes;
    }

    while( 1 ) {
        /* Read payload size */
        counter = fread( &nBytes, sizeof( SKP_int16 ), 1, bitInFile );
        if( nBytes < 0 || counter < 1 ) {
            break;
        }

        /* Read payload */
        counter = fread( payloadEnd, sizeof( SKP_uint8 ), nBytes, bitInFile );
        if( (SKP_int16)counter < nBytes ) {
            break;
        }

        /* Simulate losses */
        if( ( (float)rand() / (float)RAND_MAX >= loss_prob / 100 ) && counter > 0
 ) {
            nBytesPerPacket[ MAX_LBRR_DELAY ] = nBytes;
            payloadEnd                        += nBytes;
        } else {
            nBytesPerPacket[ MAX_LBRR_DELAY ] = 0;
        }

        if( nBytesPerPacket[ 0 ] == 0 ) {
            /* Indicate lost packet */
            lost = 1;

            /* Packet loss. Search after FEC in next packets. Should be done in t
he jitter buffer */
            payloadPtr = payload;
            for( i = 0; i < MAX_LBRR_DELAY; i++ ) {
                if( nBytesPerPacket[ i + 1 ] > 0 ) {
                    SKP_Silk_SDK_search_for_LBRR( psDec, payloadPtr, nBytesPerPac
ket[ i + 1 ], i + 1, FECpayload, &nBytesFEC );
                    if( nBytesFEC > 0 ) {
                        payloadToDec = FECpayload;
                        nBytes = nBytesFEC;
                        lost = 0;
                        break;
                    }
                }
                payloadPtr += nBytesPerPacket[ i + 1 ];
            }
        } else {
            lost = 0;
```

```
            nBytes = nBytesPerPacket[ 0 ];
            payloadToDec = payload;
        }

        /* Silk decoder */
        outPtr = out;
        tot_len = 0;

        if( lost == 0 ) {
            /* No Loss: Decode all frames in the packet */
            frames = 0;
            do {
                /* Decode 20 ms */
                ret = SKP_Silk_SDK_Decode( psDec, &DecControl, 0, payloadToDec, n
Bytes, outPtr, &len );
                if( ret ) {
                    printf( "\nSKP_Silk_SDK_Decode returned %d", ret );
                }

                frames++;
                outPtr  += len;
                tot_len += len;
                if( frames > MAX_INPUT_FRAMES ) {
                    /* Hack for corrupt stream that could generate too many frame
s */
                    outPtr  = out;
                    tot_len = 0;
                    frames  = 0;
                }
                /* Until last 20 ms frame of packet has been decoded */
            } while( DecControl.moreInternalDecoderFrames );
        } else {
            /* Loss: Decode enough frames to cover one packet duration */
            for( i = 0; i < DecControl.framesPerPacket; i++ ) {
                /* Generate 20 ms */
                ret = SKP_Silk_SDK_Decode( psDec, &DecControl, 1, payloadToDec, n
Bytes, outPtr, &len );
                if( ret ) {
                    printf( "\nSKP_Silk_Decode returned %d", ret );
                }
                outPtr  += len;
                tot_len += len;
            }
        }
        totPackets++;

        /* Write output to file */
        fwrite( out, sizeof( SKP_int16 ), tot_len, speechOutFile );

        /* Update buffer */
        totBytes = 0;
```

```
        for( i = 0; i < MAX_LBRR_DELAY; i++ ) {
            totBytes += nBytesPerPacket[ i + 1 ];
        }
        SKP_memmove( payload, &payload[ nBytesPerPacket[ 0 ] ], totBytes * sizeof
( SKP_uint8 ) );
        payloadEnd -= nBytesPerPacket[ 0 ];
        SKP_memmove( nBytesPerPacket, &nBytesPerPacket[ 1 ], MAX_LBRR_DELAY * siz
eof( SKP_int16 ) );

        if( !quiet ) {
            fprintf( stderr, "\rFrames decoded:             %d", totPackets );
        }
    }

    /* Empty the recieve buffer */
    for( k = 0; k < MAX_LBRR_DELAY; k++ ) {
        if( nBytesPerPacket[ 0 ] == 0 ) {
            /* Indicate lost packet */
            lost = 1;

            /* Packet loss. Search after FEC in next packets. Should be done in t
he jitter buffer */
            payloadPtr = payload;
            for( i = 0; i < MAX_LBRR_DELAY; i++ ) {
                if( nBytesPerPacket[ i + 1 ] > 0 ) {
                    SKP_Silk_SDK_search_for_LBRR( psDec, payloadPtr, nBytesPerPac
ket[ i + 1 ], i + 1, FECpayload, &nBytesFEC );
                    if( nBytesFEC > 0 ) {
                        payloadToDec = FECpayload;
                        nBytes = nBytesFEC;
                        lost = 0;
                        break;
                    }
                }
                payloadPtr += nBytesPerPacket[ i + 1 ];
            }
        } else {
            lost = 0;
            nBytes = nBytesPerPacket[ 0 ];
            payloadToDec = payload;
        }

        /* Silk decoder */
        outPtr  = out;
        tot_len = 0;

        if( lost == 0 ) {
            /* No loss: Decode all frames in the packet */
            frames = 0;
            do {
                /* Decode 20 ms */
                ret = SKP_Silk_SDK_Decode( psDec, &DecControl, 0, payloadToDec, n
Bytes, outPtr, &len );
```

```
                if( ret ) {
                    printf( "\nSKP_Silk_SDK_Decode returned %d", ret );
                }

                frames++;
                outPtr  += len;
                tot_len += len;
                if( frames > MAX_INPUT_FRAMES ) {
                    /* Hack for corrupt stream that could generate too many frame
s */
                    outPtr  = out;
                    tot_len = 0;
                    frames  = 0;
                }
            /* Until last 20 ms frame of packet has been decoded */
            } while( DecControl.moreInternalDecoderFrames );
        } else {
            /* Loss: Decode enough frames to cover one packet duration */

            /* Generate 20 ms */
            for( i = 0; i < DecControl.framesPerPacket; i++ ) {
                ret = SKP_Silk_SDK_Decode( psDec, &DecControl, 1, payloadToDec, n
Bytes, outPtr, &len );
                if( ret ) {
                    printf( "\nSKP_Silk_Decode returned %d", ret );
                }
                outPtr  += len;
                tot_len += len;
            }
        }
        totPackets++;

        /* Write output to file */
        fwrite( out, sizeof( SKP_int16 ), tot_len, speechOutFile );

        /* Update Buffer */
        totBytes = 0;
        for( i = 0; i < MAX_LBRR_DELAY; i++ ) {
            totBytes += nBytesPerPacket[ i + 1 ];
        }
        SKP_memmove( payload, &payload[ nBytesPerPacket[ 0 ] ], totBytes * sizeof
( SKP_uint8 ) );
        payloadEnd -= nBytesPerPacket[ 0 ];
        SKP_memmove( nBytesPerPacket, &nBytesPerPacket[ 1 ], MAX_LBRR_DELAY * siz
eof( SKP_int16 ) );

        if( !quiet ) {
            fprintf( stderr, "\rPackets decoded:              %d", totPackets );
        }
    }

    if( !quiet ) {
```

```
        printf( "\nDecoding Finished \n" );
    }

    /* Free decoder */
    free( psDec );

    /* Close files */
    fclose( speechOutFile );
    fclose( bitInFile );

    return 0;
}
```

A.140. test/Encoder.c

```
/***********************************************************************
Copyright (c) 2006-2010, Skype Limited. All rights reserved.
Redistribution and use in source and binary forms, with or without
modification, (subject to the limitations in the disclaimer below)
are permitted provided that the following conditions are met:
- Redistributions of source code must retain the above copyright notice,
this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in the
documentation and/or other materials provided with the distribution.
- Neither the name of Skype Limited, nor the names of specific
contributors, may be used to endorse or promote products derived from
this software without specific prior written permission.
NO EXPRESS OR IMPLIED LICENSES TO ANY PARTY'S PATENT RIGHTS ARE GRANTED
BY THIS LICENSE. THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
CONTRIBUTORS ''AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING,
BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND
FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE
COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON
ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
***********************************************************************/

/*****************************/
/* Silk encoder test program */
/*****************************/
```

```
#ifdef _WIN32
#define _CRT_SECURE_NO_DEPRECATE    1
#endif

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "SKP_Silk_SDK_API.h"

/* Define codec specific settings */
#define MAX_BYTES_PER_FRAME     250 // Equals peak bitrate of 100 kbps
#define MAX_INPUT_FRAMES        5
#define MAX_LBRR_DELAY          2
#define MAX_FRAME_LENGTH        480

static void print_usage( char* argv[] ) {
    printf( "\nusage: %s in.pcm out.bit [settings]\n", argv[ 0 ] );
    printf( "\nin.pcm              : Speech input to encoder" );
    printf( "\nout.bit             : Bitstream output from encoder" );
    printf( "\n   settings:" );
    printf( "\n-fs <kHz>           : Sampling rate in kHz, default: 24" );
    printf( "\n-packetlength <ms>  : Packet interval in ms, default: 20" );
    printf( "\n-rate <bps>         : Target bitrate; default: 25000" );
    printf( "\n-loss <perc>        : Uplink loss estimate, in percent (0-100); de
fault: 0" );
    printf( "\n-inbandFEC <flag>   : Enable inband FEC usage (0/1); default: 0" )
;
    printf( "\n-complexity <comp>  : Set complexity, 0: low, 1: medium, 2: high;
default: 2" );
    printf( "\n-DTX <flag>         : Enable DTX (0/1); default: 0" );
    printf( "\n-quiet              : Print only some basic values" );
    printf( "\n");
}

int main( int argc, char* argv[] )
{
    size_t    counter;
    SKP_int   k, args, totPackets, totActPackets, ret;
    SKP_int16 nBytes;
    double    sumBytes, sumWBytes, sumActBytes, avg_rate, act_rate, wght_rate, nr
g;
    SKP_uint8 payload[ MAX_BYTES_PER_FRAME * MAX_INPUT_FRAMES ];
    SKP_int16 in[ MAX_FRAME_LENGTH * MAX_INPUT_FRAMES ];
    char      speechInFileName[ 150 ], bitOutFileName[ 150 ];
    FILE      *bitOutFile, *speechInFile;
    SKP_int32 encSizeBytes;
    void      *psEnc;

    /* default settings */
    SKP_int   fs_kHz = 24;
    SKP_int   targetRate_bps = 25000;
```

```
    SKP_int   packetSize_ms = 20;
    SKP_int   frameSizeReadFromFile_ms = 20;
    SKP_int   packetLoss_perc = 0, complexity_mode = 2, smplsSinceLastPacket;
    SKP_int   INBandFec_enabled = 0, DTX_enabled = 0, quiet = 0;
    SKP_SILK_SDK_EncControlStruct encControl; // Struct for input to encoder

    if( argc < 3 ) {
        print_usage( argv );
        exit( 0 );
    }

    /* get arguments */
    args = 1;
    strcpy( speechInFileName, argv[ args ] );
    args++;
    strcpy( bitOutFileName,   argv[ args ] );
    args++;
    while( args < argc ) {
        if( SKP_STR_CASEINSENSITIVE_COMPARE( argv[ args ], "-fs" ) == 0 ) {
            sscanf( argv[ args + 1 ], "%d", &fs_kHz );
            args += 2;
        } else if( SKP_STR_CASEINSENSITIVE_COMPARE( argv[ args ], "-packetlength"
 ) == 0 ) {
            sscanf( argv[ args + 1 ], "%d", &packetSize_ms );
            args += 2;
        } else if( SKP_STR_CASEINSENSITIVE_COMPARE( argv[ args ], "-rate" ) == 0
) {
            sscanf( argv[ args + 1 ], "%d", &targetRate_bps );
            args += 2;
        } else if( SKP_STR_CASEINSENSITIVE_COMPARE( argv[ args ], "-loss" ) == 0
) {
            sscanf( argv[ args + 1 ], "%d", &packetLoss_perc );
            args += 2;
        } else if( SKP_STR_CASEINSENSITIVE_COMPARE( argv[ args ], "-complexity" )
 == 0 ) {
            sscanf( argv[ args + 1 ], "%d", &complexity_mode );
            args += 2;
        } else if( SKP_STR_CASEINSENSITIVE_COMPARE( argv[ args ], "-inbandFEC" )
== 0 ) {
            sscanf( argv[ args + 1 ], "%d", &INBandFec_enabled );
            args += 2;
        } else if( SKP_STR_CASEINSENSITIVE_COMPARE( argv[ args ], "-DTX") == 0 )
{
            sscanf( argv[ args + 1 ], "%d", &DTX_enabled );
            args += 2;
        } else if( SKP_STR_CASEINSENSITIVE_COMPARE( argv[ args ], "-quiet" ) == 0
 ) {
            quiet = 1;
            args++;
        } else {
            printf( "Error: unrecognized setting: %s\n\n", argv[ args ] );
            print_usage( argv );
            exit( 0 );
        }
    }
```

```
    /* Print options */
    if( !quiet ) {
        printf("****************** Silk Encoder v %s ***************\n", SKP_Si
lk_SDK_get_version());
        printf("****************** Compiled for %d bit cpu ******** \n", (int)s
izeof(void*) * 8 );
        printf( "Input:                          %s\n",        speechInFileName );
        printf( "Output:                         %s\n",        bitOutFileName );
        printf( "Sampling rate:                  %d kHz\n",    fs_kHz);
        printf( "Packet interval:                %d ms\n",     packetSize_ms);
        printf( "Inband FEC used:                %d\n",        INBandFec_enabled);
        printf( "DTX used:                       %d\n",        DTX_enabled);
        printf( "Complexity:                     %d\n",        complexity_mode);
        printf( "Target bitrate:                 %d bps\n",    targetRate_bps);
    }

    /* Open files */
    speechInFile = fopen( speechInFileName, "rb" );
    if( speechInFile == NULL ) {
        printf( "Error: could not open input file %s\n", speechInFileName );
        exit( 0 );
    }
    bitOutFile = fopen( bitOutFileName, "wb" );
    if( bitOutFile == NULL ) {
        printf( "Error: could not open output file %s\n", bitOutFileName );
        exit( 0 );
    }

    /* Create Encoder */
    ret = SKP_Silk_SDK_Get_Encoder_Size( &encSizeBytes );
    if( ret ) {
        printf( "\nSKP_Silk_create_encoder returned %d", ret );
    }

    psEnc = malloc( encSizeBytes );

    /* Reset Encoder */
    ret = SKP_Silk_SDK_InitEncoder( psEnc, &encControl );
    if( ret ) {
        printf( "\nSKP_Silk_reset_encoder returned %d", ret );
    }

    /* Set Encoder parameters */
    encControl.sampleRate          = fs_kHz * 1000;
    encControl.packetSize          = packetSize_ms * fs_kHz;
    encControl.packetLossPercentage = packetLoss_perc;
    encControl.useInBandFEC        = INBandFec_enabled;
    encControl.useDTX              = DTX_enabled;
    encControl.complexity          = complexity_mode;
    encControl.bitRate             = targetRate_bps;
```

```
    if( fs_kHz > 24 || fs_kHz < 0 ) {
        printf( "\nError: Sampling rate = %d out of range, valid range 8 - 24 \n
\n", fs_kHz );
        exit( 0 );
    }

    totPackets           = 0;
    totActPackets        = 0;
    smplsSinceLastPacket = 0;
    sumBytes             = 0.0;
    sumActBytes          = 0.0;
    sumWBytes            = 0.0;

    while( 1 ) {
        /* Read input to file */
        counter = fread( in, sizeof( SKP_int16 ), frameSizeReadFromFile_ms * fs_k
Hz, speechInFile );
        if( (SKP_int)counter < frameSizeReadFromFile_ms * fs_kHz ) {
            break;
        }

        /* max payload size */
        nBytes = MAX_BYTES_PER_FRAME * MAX_INPUT_FRAMES;

        /* Silk Encoder */
        ret = SKP_Silk_SDK_Encode( psEnc, &encControl, in, (SKP_int16)counter, pa
yload, &nBytes );
        if( ret ) {
            printf( "\nSKP_Silk_Encode returned %d", ret );
            break;
        }

        /* Get packet size */
        packetSize_ms = (SKP_int)( 1000 * encControl.packetSize ) / encControl.sa
mpleRate;

        smplsSinceLastPacket += (SKP_int)counter;

        if( ( smplsSinceLastPacket / fs_kHz ) == packetSize_ms ) {
            /* Sends a dummy zero size packet in case of DTX period  */
            /* to make it work with the decoder test program.        */
            /* In practice should be handled by RTP sequence numbers */
            totPackets++;
            sumBytes  += nBytes;
            sumWBytes += pow( (double)nBytes, 10.0 );
            nrg = 0.0;
            for( k = 0; k < (SKP_int)counter; k++ ) {
                nrg += in[ k ] * (double)in[ k ];
            }
            if( ( nrg / (SKP_int)counter ) > 1e3 ) {
                sumActBytes += nBytes;
                totActPackets++;
```

```
            }

            /* Write payload size */
            fwrite( &nBytes, sizeof( SKP_int16 ), 1, bitOutFile );

            /* Write payload */
            fwrite( payload, sizeof( SKP_uint8 ), nBytes, bitOutFile );

            if( !quiet ) {
                fprintf( stderr, "\rPackets encoded:              %d", totPackets
 );
            }
            smplsSinceLastPacket = 0;
        }
    }

    /* Write dummy because it can not end with 0 bytes */
    nBytes = -1;

    /* Write payload size */
    fwrite( &nBytes, sizeof( SKP_int16 ), 1, bitOutFile );

    /* Query encoder */
    packetSize_ms = (SKP_int)( 1000 * encControl.packetSize ) / encControl.sample
Rate;

    /* Free Encoder */
    free( psEnc );

    fclose( speechInFile );
    fclose( bitOutFile   );

    avg_rate  = 8.0 / packetSize_ms * sumBytes       / totPackets;
    act_rate  = 8.0 / packetSize_ms * sumActBytes    / totActPackets;
    wght_rate = 8.0 / packetSize_ms * pow( sumWBytes / totPackets, 0.1 );
    if( !quiet ) {
        printf( "\nAverage bitrate:             %.3f kbps", avg_rate  );
        printf( "\nActive bitrate:              %.3f kbps", act_rate  );
        printf( "\nWeighted bitrate:            %.3f kbps", wght_rate );
        printf( "\n\n" );
    } else {
        /* print average and weighted bitrates */
        printf( "%.3f %.3f %.3f \n", avg_rate, act_rate, wght_rate );
    }
    return 0;
}
```

Authors' Addresses

Koen Vos
Skype Technologies S.A.
Stadsgaarden 6
Stockholm 11645
SE

Phone: +46 855 921 989
Email: koen.vos@skype.net

Soeren Skak Jensen
Skype Technologies S.A.
Stadsgaarden 6
Stockholm 11645
SE

Phone: +46 855 921 989
Email: soren.skak.jensen@skype.net

Karsten Vandborg Soerensen
Skype Technologies S.A.
Stadsgaarden 6
Stockholm 11645
SE

Phone: +46 855 921 989
Email: karsten.vandborg.sorensen@skype.net