ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 53556.7— 2013

Звуковое вещание цифровое

КОДИРОВАНИЕ СИГНАЛОВ ЗВУКОВОГО ВЕЩАНИЯ С СОКРАЩЕНИЕМ ИЗБЫТОЧНОСТИ ДЛЯ ПЕРЕДАЧИ ПО ЦИФРОВЫМ КАНАЛАМ СВЯЗИ. ЧАСТЬ III

(MPEG-4 AUDIO)

Параметрическое кодирование звуковых сигналов (HILN)

ISO/IEC 14496-3:2009
(NEQ)

Издание официальное

Москва
Стандартинформ
2014

gost.e-ecolog.ru

Предисловие

1 РАЗРАБОТАН Санкт-Петербургским филиалом Центрального научно-исследовательского института связи «Ленинградское отделение» (ФГУП ЛО ЦНИИС)

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 480 «Связь»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 22 ноября 2013 г. ТК 1704-ст

4 Настоящий стандарт разработан с учетом основных нормативных положений международного стандарта ИСО/МЭК 14496-3:2009 «Информационные технологии. Кодирование аудиовизуальных объектов. Часть 3. Аудио» (ISO/IEC 14496-3:2009 «Information technology — Coding of audio-visual objects — Part 3: Audio», NEQ)

5 ВВЕДЕН ВПЕРВЫЕ

Правила применения настоящего стандарта установлены в ГОСТ Р 1.0—2012 (раздел 8). Информация об изменениях к настоящему стандарту публикуется в годовом (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячно издаваемом информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ближайшем выпуске ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (gost.ru)

© Стандартинформ, 2014

Настоящий стандарт не может быть полностью или частично воспроизведен, тиражирован и распространен в качестве официального издания без разрешения Федерального агентства по техническому регулированию и метрологии

Содержание

1 Область действия 1
1.1 Технический обзор 1
2 Термины и определения 2
3 Синтаксис потока битов 2
3.1 Конфигурация декодера *(ParametricSpecificConfig)* 2
3.2 Фрейм потока битов *(slPacketPayload)* 5
4 Семантика потока битов 26
4.1 Конфигурация декодера *(ParametricSpecificConfig)* 26
4.2 Фрейм потока битов *(slPacketPayload)* 27
5 Инструменты параметрического декодера 29
5.1 Инструменты декодера *HILN* 29
5.2 Интегрированный параметрический кодер 47
6 Устойчивые к ошибкам полезные нагрузки потока битов 47
6.1 Обзор инструментов 47
6.2 *ER HILN* 48
Приложение А (справочное) Параметрический аудиокодер 49
Библиография 54

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Звуковое вещание цифровое

КОДИРОВАНИЕ СИГНАЛОВ ЗВУКОВОГО ВЕЩАНИЯ С СОКРАЩЕНИЕМ ИЗБЫТОЧНОСТИ ДЛЯ ПЕРЕДАЧИ ПО ЦИФРОВЫМ КАНАЛАМ СВЯЗИ. ЧАСТЬ III (MPEG-4 AUDIO)

Параметрическое кодирование звуковых сигналов (HILN)

Sound broadcasting digital.
Coding of signals of sound broadcasting with reduction of redundancy for transfer on digital communication channels.
A part III (MPEG-4 audio). Parametric audio coding (HILN)

Дата введения — 2014—09—01

1 Область действия

Параметрическое аудиокодирование обеспечивает инструменты *HILN*, которые дополняют другие инструменты кодирования естественного аудио в области уровней очень низких битовых скоростей. Их внимание сосредоточено на представлении монофонических музыкальных сигналов с низкой и промежуточной сложностью контента в диапазоне 4 — 16 Кбит/с. *HILN* задействует высокую степень интерактивности путем неявной поддержки изменения скорости и шага во время воспроизведения с возможностью масштабируемости скорости передачи. Кроме того возможная комбинация с инструментами параметрического кодирования речи *HVXC* допускает очень эффективные схемы кодирования речи и музыкальных сигналов.

1.1 Технический обзор

Параметрическое аудиокодирование *MPEG*-4 использует метод *HILN* (Гармонические и отдельные линии плюс шум), чтобы кодировать такие аудиосигналы, как музыка на скоростях передачи 4 Кбит/с и выше, используя масштабируемое параметрическое представление аудиосигнала. *HILN* позволяет независимо изменять скорости и шаг во время декодирования. Кроме того *HILN* может быть объединено с параметрическим кодированием речи *MPEG*-4 (*HVXC*), чтобы сформировать интегрированный параметрический кодер, охватывающий более широкий диапазон сигналов и скоростей передачи.

Интегрированный параметрический кодер может работать в следующих режимах:

Таблица 1 — Режимы работы параметрического кодера

PARAmode	Описание
0	только *HVXC*
1	только *HILN*
2	переключение *HVXC/HILN*
3	смешанный *HVXC/HILN*

PARAmodes 0 и 1 представляют режимы фиксированных *HVXC* и *HILN*. *PARAmode* 2 разрешает автоматическое переключение между *HVXC* и *HILN* в зависимости от типа текущего входного сигнала. В *PARAmode* 3 декодеры *HVXC* и *HILN* могут использоваться одновременно, и их выходные сигналы добавляются (смешиваются) в параметрическом декодере.

В режимах "коммутируемые *HVXC/HILN*" и "смешанные *HVXC/HILN*" инструменты декодера *HVXC* и *HILN* работают альтернативно или одновременно согласно *PARAswitchMode* или *PARAmixMode* те-

Издание официальное

кущего фрейма. Чтобы получить надлежащее временное выравнивание выходных сигналов декодера *HVXC* и *HILN* прежде, чем они будут добавлены, буфер *FIFO* компенсирует разницу во времени между задержкой декодера *HVXC* и *HILN*.

Чтобы избежать трудных переходов на границах фрейма, когда декодеры *HVXC* или *HILN* включаются или выключаются, соответствующие выходные сигналы декодера появляются и спадают постепенно. Для декодера *HVXC* применяется линейное изменение 20 мс, когда он включается или выключается. Декодер *HILN* не требует дополнительного замирания из-за гладких окон синтеза, используемых в синтезаторе *HILN.* Необходимо только сбросить декодер *HILN* (*numLine* = 0), если текущий фрейм потока битов не содержит *HILNframe ()*.

2 Термины и определения

В настоящем стандарте применены термины и сокращения с соответствующими определениями, используемые в ГОСТ Р 53556.0—2009.

3 Синтаксис потока битов

Естественный объект аудио *MPEG*-4, использующий параметрическое кодирование, передается в одном или нескольких элементарных потоках: поток базового уровня, поток дополнительного уровня улучшения и один или более дополнительных потоков уровня расширения.

Синтаксис потока битов описывается в коде *pseudo-C*.

Мнемоники *LARH*1*, LARH*2*, LARH*3*, LARN*1*, LARN*2*, DIA, DIF, DHF, DFS* указывают, что используется кодовая комбинация *"vlclbf".*

Мнемонический *SDC* указывает, что используется кодовая комбинация *"vlclbf"*, которая декодируется *HILN SubDivisionCode*, используя параметры для *SDCdecode (),* как дано в описании синтаксиса потока битов.

3.1 Конфигурация декодера *(ParametricSpecificConfig)*

Информация о конфигурации декодера для параметрического кодирования передается в *ParametricSpecificConfig* () базового уровня и элементарном потоке уровня улучшения или расширения.

Параметрический базовый уровень — конфигурация

Параметрический кодер в немасштабируемом режиме или базовом уровне в масштабируемом режиме *HILN* используют *ParametricSpecificConfig* () с *isBaseLayer* == 1.

Параметрический уровень улучшения/расширения *HILN* — конфигурация

Чтобы использовать *HILN* в качестве ядра в режиме «масштабируемый *T/F* с ядром», в дополнение к базовому уровню *HILN* требуется уровень улучшения *HILN*. При работе с масштабируемой скоростью передачи *HILN* в дополнение к базовому уровню *HILN* разрешаются один или более уровней расширения *HILN.* Уровень улучшения и уровень расширения используют *ParametricSpecificConfig* () с *isBaseLayer* == 0.

Таблица 2 — Синтаксис *ParametricSpecificConfig ()*

Синтаксис	Количество битов	Мнемоника
ParametricSpecificConfig () *{* *isBaseLayer;* *if (isBaseLayer) {* *PARAconfig ();* *}* *else {* *HILNenexConfig ();* *}* *}*	1	*uimsbf*

3.1.1 Конфигурация параметрического декодера аудио

Таблица 3 — Синтаксис *PARAconfig* ()

Синтаксис	Количество битов	Мнемоника
PARAconfig () *{* *PARAmode;* *if (PARAmode! = 1) {* *ErHVXCconfig ();* *}* *if (PARAmode! = 0) {* *HILNconfig ();* *}* *PARAextensionFlag;* *if (PARAextensionFlag) {* */* to be defined in MPEG 4 Phase 3 */* *}* *}*	 2 1 	 *uimsbf* *uimsbf*

Таблица 4 — *PARAmode*

PARAmode	Длина фрейма	Описание
0	20 мс (*N* = 160 выборок)	только *HVXC*
1	см. 3.1.2 и 5.1.4.3.3	только *HILN*
2	40 мс (*N* = 320 выборок)	переключение *HVXC/HILN*
3	40 мс (*N* = 320 выборок)	смешивание *HVXC/HILN*

3.1.2 Конфигурация декодера *HILN*

Таблица 5 — Синтаксис *HILNconfig*

Синтаксис	Количество битов	Мнемоника
HILNconfig () *{* *HILNquantMode;* *HILNmaxNumLine;* *HILNsampleRateCode;* *HILNframeLength;* *HILNcontMode;* *}*	 1 8 4 12 2 	 *uimsbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf*

Таблица 6 — Синтаксис *HILNenexConfig ()*

Синтаксис	Количество битов	Мнемоника
HILNconfig () *{* *HILNenhaLayer*	 1	 *uimsbf*

Окончание таблицы 6

Синтаксис	Количество битов	Мнемоника
if (HILNenhaLayer) { *HILNenhaQuantMode* *}* *}*	2	*uimsbf*

Таблица 7 — *HILNsampleRateCode*

HILNsampleRateCode	*sampleRate*	*maxFIndex*
0	96000	890
1	88200	876
2	64000	825
3	48000	779
4	44100	765
5	32000	714
6	24000	668
7	22050	654
8	16000	603
9	12000	557
10	11025	544
11	8000	492
12	7350	479
13	зарезервировано	зарезервировано
14	зарезервировано	зарезервировано
15	зарезервировано	зарезервировано

Таблица 8 — *linebits*

HILNmaxNumLine	0	1	2.. 3	4.. 7	8.. 15	16.. 31	32.. 63	64.. 127	128.. 255
linebits	0	1	2	3	4	5	6	7	8

Таблица 9 — *HILNcontMode*

HILNcontMode	Дополнительное продолжение линии декодером (см. подпункт 5.1.4.3.1)
0	гармонические линии <-> отдельные линии и линии гармоник <-> линии гармоник
1	режим 0 плюс отдельные линии <-> отдельные линии
2	дополнительное продолжение линий декодером отсутствует
3	(зарезервировано)

Число битов улучшения частоты *(fEnhbits [i]) в HILNenhaFrame ()* вычисляется следующим образом:

• отдельная линия:

fEnhbits [i] = max (0, fEnhbitsBase [ILFreqIndex [i]] + fEnhbitsMode [HILNenhaQuantMode])

• линия гармоники:

fEnhbits [i] = max (0, fEnhbitsBase [harmFreqIndex] + fEnhbitsMode [HILNenhaQuantMode] + fEnhbitsHarm [i])

Таблица 10 — *fEnhbitsBase*

ILFreqIndex	*harmFreqIndex*	*fEnhbitsBase*
0.. 159	0.. 1243	0
160.. 269	1244.. 1511	1
270.. 380	1512.. 1779	2
381.. 491	1780.. 2047	3
492.. 602		4
603.. 713		5
714.. 890		6

Таблица 11 — *fEnhbitsMode*

HILNenhaQuantMode	0	1	2	3
fEnhbitsMode	–3	–2	–1	0

Таблица 12 — *fEnhbitsHarm*

i	0	1	2.. 3	4.. 7	8.. 9
fEnhbitsHarm [i]	0	1	2	3	4

Таблица 13 — Константы *HILN*

tmbits	4
atkbits	4
decbits	4
tmEnhbits	3
atkEnhbits	2
decEnhbits	2
phasebits	5

3.2 Фрейм потока битов *(slPacketPayload)*

Динамические данные для параметрического кодирования передаются как пакетная полезная нагрузка *SL* в элементарном потоке базового уровня и дополнительного уровня улучшения или расширения.

Параметрический базовый уровень — полезная нагрузка устройства доступа.

Для параметрического кодера в немасштабируемом режиме или для базового уровня в масштабируемом режиме *HILN* определяется следующая полезная нагрузка фрейма потока битов:

```
slPacketPayload {
    PARAframe ();
}
```

Параметрический уровень улучшения/расширения *HILN* — полезная нагрузка устройства доступа.

Чтобы проанализировать и декодировать уровень улучшения *HILN*, запрашивается декодируемая информация из базового уровня *HILN*.

Чтобы проанализировать и декодировать уровень расширения *HILN*, запрашивается декодируемая информация из базового уровня *HILN* и возможного нижнего уровня расширения *HILN*. Синтаксис потока битов уровней расширения *HILN* описывается способом, который требует, чтобы фреймы базового потока битов *HILN* и расширения анализировались в надлежащем порядке:

1	*HILNbasicFrame ()*	фрейм базового потока битов
2	*HILNextFrame (1)*	фрейм 1-го потока битов расширения (если доступен фрейм базового потока битов)
3	*HILNextFrame (2)*	фрейм 2-го потока битов расширения (если доступны фреймы базового потока и 1-го потока битов расширения)
4	и т. д.	

Для уровня улучшения и уровня расширения в масштабируемом режиме HILN определяется следующая полезная нагрузка фрейма потока битов:

```
slPacketPayload {
    HILNenexFrame ();
}
```

3.2.1 Фрейм параметрического потока битов аудио

Т а б л и ц а 14 — Синтаксис *PARAframe ()*

Синтаксис	Количество битов	Мнемоника
PARAframe () *{* *if (PARAmode == 0) {* *ErHVXCframe (HVXCrate);* *}* *else if (PARAmode == 1) {* *HILNframe ();* *}* *else if (PARAmode == 2) {* *switchFrame ();* *}* *else if (PARAmode == 3) {* *mixFrame ();* *}* *}*		

Т а б л и ц а 15 — Синтаксис *switchFrame ()*

Синтаксис	Количество битов	Мнемоника
switchFrame () *{* *PARAswitchMode;* *if (PARAswitchMode == 0) {* *ErHVXCdoubleframe (HVXCrate);* *}* *else {* *HILNframe ();* *}* *}*	1	*uimsbf*

В каждом фрейме выбирается один из следующих *PARAswitchModes*:

Таблица 16 — *PARAswitchMode*

PARAswitchMode	Описание
0	только *HVXC*
1	только *HILN*

Таблица 17 — Синтаксис *mixFrame* ()

Синтаксис	Количество битов	Мнемоника
mixFrame () *{* *PARAmixMode;* *if (PARAmixMode == 0) {* *ErHVXCdoubleframe (HVXCrate);* *}* *else if (PARAmixMode == 1) {* *ILNframe ();* *ErHVXCdoubleframe (2000);* *}* *else if (PARAmixMode == 2) {* *HILNframe ();* *ErHVXCdoubleframe (4000);* *}* *else if (PARAmixMode == 3) {* *HILNframe ();* } }	2	*uimsbf*

В каждом фрейме выбирается один из следующих *PARAmixModes:*

Таблица 18 — *PARAmixMode*

PARAmixMode	Описание
0	только *HVXC*
1	*HVXC* 2 Кбит/с и *HILN*
2	*HVXC* 4 Кбит/с и *HILN*
3	только *HILN*

Таблица 19 — Синтаксис *HVXCdoubleframe ()*

Синтаксис	Количество битов	Мнемоника
ErHVXCdoubleframe (rate) *{* *if (rate*> = 3000*) {* *ErHVXCfixframe (*4000*);*		

Окончание таблицы 19

Синтаксис	Количество битов	Мнемоника
ErHVXCfixframe (rate); *}* *else {* *ErHVXCfixframe (2000);* *ErHVXCfixframe (rate);* *}* *}*		

3.2.2 Фрейм потока битов *HILN*

Таблица 20 — Синтаксис *HILNframe ()*

Синтаксис	Количество битов	Мнемоника
HILNframe () *{* *numLayer* = 0; *HILNbasicFrameESC0 ();* *HILNbasicFrameESC1 ();* *HILNbasicFrameESC2 ();* *HILNbasicFrameESC3 ();* *HILNbasicFrameESC4 ();* *layNumLine [0] = numLine;* *layPrevNumLine [0] = prevNumLine;* *for (k = 0; k <prevNumLine; k ++) {* *layPrevLineContFlag [0] [k] = prevLineContFlag [k];* *}* *}*		

Таблица 21 — Синтаксис *HILNbasicFrameESC0 ()*

Синтаксис	Количество битов	Мнемоника
HILNbasicFrameESC0 () *{* *prevNumLine = numLine;* */* prevNumLine is set to the number of lines */* */* in the previous frame */* */* prevNumLine = 0 for the first bitstream frame */* *numLine;* *harmFlag;* *noiseFlag;* *envFlag;* *phaseFlag;*	 *linebits* 1 1 1 1	 *uimsbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf*

Окончание таблицы 21

Синтаксис	Количество битов	Мнемоника
maxAmplIndexCoded; *maxAmplIndex = 4*maxAmplIndexCoded;* *if (harmFlag) {* *HARMbasicParaESC0 ();* *}* *if (noiseFlag) {* *NOISEbasicParaESC0 ();* *}* *}*	4	*uimsbf*

Т а б л и ц а 22 — Синтаксис *HILNbasicFrameESC1 ()*

Синтаксис	Количество битов	Мнемоника
HILNbasicFrameESC1 () *{* *if (harmFlag) {* *HARMbasicParaESC1 ();* *}* *if (noiseFlag) {* *NOISEbasicParaESC1 ();* *}* *INDIbasicParaESC1 ();* *}*		

Т а б л и ц а 23 — Синтаксис *HILNbasicFrameESC2 ()*

Синтаксис	Количество битов	Мнемоника
HILNbasicFrameESC2 () *{* *INDIbasicParaESC2 ();* *}*		

Т а б л и ц а 24 — Синтаксис *HILNbasicFrameESC3 ()*

Синтаксис	Количество битов	Мнемоника
HILNbasicFrameESC3 () *{* *if (envFlag) {* *envTmax;* *envRatk;* *envRdec* *}* *if (harmFlag) {* *HARMbasicParaESC3 ();*	 *tmbits* *atkbits* *decbits*	 *uimsbf* *uimsbf* *uimsbf*

Окончание таблицы 24

Синтаксис	Количество битов	Мнемоника
} *if (noiseFlag) {* *NOISEbasicParaESC3 ();* *}* *INDIbasicParaESC3 ();* *if (harmFlag) {* *harmFreqStretch;* *}* *}*	1..7	*HFS*

Т а б л и ц а 25 — Синтаксис *HILNbasicFrameESC4 ()*

Синтаксис	Количество битов	Мнемоника
HILNbasicFrameESC4 () *{* *if (harmFlag) {* *HARMbasicParaESC4 ();* *}* *if (noiseFlag) {* *NOISEbasicParaESC4 ();* *}* *INDIbasicParaESC4 ();* *}*		

Т а б л и ц а 26 — Синтаксис *HARMbasicParaESC0()*

Синтаксис	Количество битов	Мнемоника
HARMbasicParaESC0 () *{* *prevHarmAmplIndex = harmAmplIndex;* *prevHarmFreqIndex = harmFreqIndex;* *harmContFlag;* *harmEnvFlag;* *if (! harmContFlag) {* *harmAmplRel;* *harmAmplIndex = maxAmplIndex + harmAmplRel;* *harmFreqIndex;* *}* *}*	 1 1 6 11	 *uimsbf* *uimsbf* *uimsbf* *uimsbf*

Т а б л и ц а 27 — Синтаксис *HARMbasicParaESC1 ()*

Синтаксис	Количество битов	Мнемоника
HARMbasicParaESC1 () *{*		

Окончание таблицы 27

Синтаксис	Количество битов	Мнемоника
numHarmParaIndex;	4	*uimsbf*
numHarmPara = numHarmParaTable [numHarmParaIndex];		
numHarmLineIndex;	5	*uimsbf*
numHarmLine = numHarmLineTable [numHarmLineIndex];		
if (harmContFlag) {		
contHarmAmpl	3..8	*DIA*
harmAmplIndex = prevHarmAmplIndex + contHarmAmpl;		
contHarmFreq	2..9	*DHF*
harmFreqIndex = prevHarmFreqIndex + contHarmFreq;		
}		
for (i = 0; i <2; i ++) {		
harmLAR [i];	4..19	*LARH1*
}		
}		

Т а б л и ц а 28 — Синтаксис *HARMbasicParaESC3 ()*

Синтаксис	Количество битов	Мнемоника
HARMbasicParaESC3 ()		
{		
for (i = 2; i <min (7, numHarmPara); i ++) {		
harmLAR [i];	3..18	*LARH2*
}		
for (i = 7; i <numHarmPara; i ++) {		
harmLAR [i];	2..17	*LARH3*
}		
}		

Т а б л и ц а 29 — Синтаксис *HARMbasicParaESC4 ()*

Синтаксис	Количество битов	Мнемоника
HARMbasicParaESC4 ()		
{		
if (phaseFlag &&! harmContFlag) {		
numHarmPhase;	6	*uimsbf*
}		
else {		
numHarmPhase = 0;		
}		
for (i = 0; i <numHarmPhase; i ++) {		
harmPhase [i];	*phasebits*	*uimsbf*
harmPhaseAvail [i] = 1;		
}		

Окончание таблицы 29

Синтаксис	Количество битов	Мнемоника
for (i = numHarmPhase; i <numHarmLine; i ++) { *harmPhaseAvail [i] = 0;* *}* *}*		

Т а б л и ц а 30 — *numHarmParaTable*

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
numHarmParaTable [i]	2	3	4	5	6	7	8	9	11	13	15	17	19	21	23	25

Т а б л и ц а 31 — *numHarmLineTable*

i	0	1	2	3	4	5	6	7
numHarmLineTable [i]	3	4	5	6	7	8	9	10
i	8	9	10	11	12	13	14	15
numHarmLineTable [i]	12	14	16	19	22	25	29	33
i	16	17	18	19	20	21	22	23
numHarmLineTable [i]	38	43	49	56	64	73	83	94
i	24	25	26	27	28	29	30	31
numHarmLineTable [i]	107	121	137	155	175	197	222	250

Т а б л и ц а 32 — Синтаксис *NOISEbasicParaESC*0 *()*

Синтаксис	Количество битов	Мнемоника
NOISEbasicParaESC0 () *{* *prevNoiseAmplIndex = noiseAmplIndex;* *noiseContFlag;* *noiseEnvFlag;* *if (! noiseContFlag) {* *noiseAmplRel;* *noiseAmplIndex = maxAmplIndex + noiseAmplRel;* *}* *}*	 1 1 6 	 *uimsbf* *uimsbf* *uimsbf*

Т а б л и ц а 33 — Синтаксис *NOISEbasicParaESC*1 *()*

Синтаксис	Количество битов	Мнемоника
NOISEbasicParaESC1 () *{* *if (noiseContFlag) {* *contNoiseAmpl;* *noiseAmplIndex = prevNoiseAmplIndex +* *contNoiseAmpl;* *}*	 3.. 8 	 *DIA*

Окончание таблицы 33

Синтаксис	Количество битов	Мнемоника
numNoiseParaIndex;	4	*uimsbf*
numNoisePara = numNoiseParaTable [numNoiseParaIndex];		
for (i = 0; i <min (2, numNoisePara); i ++) {		
noiseLAR [i];	2.. 17	*LARN*1
}		
}		

Т а б л и ц а 34 — Синтаксис *NOISEbasicParaESC*3 *()*

Синтаксис	Количество битов	Мнемоника
NOISEbasicParaESC3 ()		
{		
for (i = 2; i <numNoisePara; i ++) {		
noiseLAR [i];	1.. 18	*LARN*2
}		
}		

Т а б л и ц а 35 — Синтаксис *NOISEbasicParaESC*4 *()*

Синтаксис	Количество битов	Мнемоника
*NOISEbasicParaESC*4 *()*		
{		
if (noiseEnvFlag) {		
noiseEnvTmax;	*tmbits*	*uimsbf*
noiseEnvRatk;	*atkbits*	*uimsbf*
noiseEnvRdec;	*decbits*	*uimsbf*
}		
}		

Т а б л и ц а 36 — *numNoiseParaTable*

i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
numNoiseParaTable [i]	1	2	3	4	5	6	7	9	11	13	15	17	19	21	23	25

Т а б л и ц а 37 — Синтаксис *INDIbasicParaESC*1 *()*

Синтаксис	Количество битов	Мнемоника
*INDIbasicParaESC*1 *()*		
{		
for (k = 0; k <prevNumLine; k ++) {		
prevLineContFlag [k];	1	*uimsbf*
}		
i = 0;		
for (k = 0; k <prevNumLine; k ++) {		
if (prevLineContFlag [k]) {		

Окончание таблицы 37

Синтаксис	Количество битов	Мнемоника
linePred [i] = k; *lineContFlag [i ++] = 1;* *}* *}* *while (i <numLine) {* *lineContFlag [i ++] = 0;* *}* *}*		

Т а б л и ц а 38 — Синтаксис *INDlbasicParaESC2* ()

Синтаксис	Количество битов	Мнемоника
INDlbasicParaESC2() *{* *lastNLFreq* = 0*;* *for (i* = 0*; i <prevNumLine; i ++) {* *prevILFreqIndex [i] = ILFreqIndex [i];* *prevILAmplIndex [i] = ILAmplIndex [i];* *}* *for (i* = 0*; i <numLine; i ++) {* *if (envFlag) {* *lineEnvFlag [i];* *}* *}* *for (i = 0; i <numLine; i ++) {* *if (! lineContFlag [i]) {* *if (numLine-1-i <7) {* *ILFreqInc [i];* */* SDCdecode (maxFindexlastNLFreq, */* */* sdcILFTable [numLine-1-i]) */* *}* *else {* *ILFreqInc [i];* */* SDCdecode (maxFindexlastNLFreq, */* */* sdcILFTable [7]) */* *}* *ILFreqIndex [i] = lastNLFreq + ILFreqInc [i];* *lastNLFreq = ILFreqIndex [i];* *если (HILNquantMode) {* *ILAmplRel [i];* */* SDCdecode (50, sdcILATable) */* *ILAmplIndex [i] = maxAmplIndex + ILAmplRel [i];*	 1 0.. 14 0.. 14 4.. 10 	 *uimsbf* *SDC* *SDC* *SDC*

Окончание таблицы 38

Синтаксис	Количество битов	Мнемоника

}
else {
ILAmplRel [i];
/* SDCdecode (25, sdcILATable) */
ILAmplIndex [i] = maxAmplIndex +
2*ILAmplRel [i];
}
}
}
}
	3.. 9	SDC

Таблица 39 — Синтаксис *INDIbasicParaESC3 ()*

Синтаксис	Количество битов	Мнемоника

INDIbasicParaESC3 ()
{
for (i = 0; i <numLine; i ++) {
if (lineContFlag [i]) {
DILFreq [i];
ILFreqIndex [i] = prevILFreqIndex [linePred [i]] +
DILFreq [i];
DILAmpl [i];
ILAmplIndex [i] = prevILAmplIndex [linePred [i]] +
DILAmpl [i];
}
}
}
	2.. 10
	
	3.. 8
		DIF
		
		DIA

Таблица 40 — Синтаксис *INDIbasicParaESC4 ()*

Синтаксис	Количество битов	Мнемоника

INDIbasicParaESC4 ()
{
if (phaseFlag) {
numLinePhase;
}
else {
numLinePhase = 0;
}
j = 0;
for (i = 0; i <numLine; i ++) {
if (! linePred [i] && j <numLinePhase) {
	linebits	uimsbf

Окончание таблицы 40

Синтаксис	Количество битов	Мнемоника
linePhase [i]; *linePhaseAvail [i] = 1;* *j ++;* *}* *else {* *linePhaseAvail [i] = 0;* *}* *}* *}*	*phasebits*	*uimsbf*

Т а б л и ц а 41 — Синтаксис *HILNenexFrame ()*

Синтаксис	Количество битов	Мнемоника
HILNenexFrame () *{* *HILNenhaLayer value in в ParametricSpecificConfig () */* */* thisElementary Stream must be used here! */* *if (HILNenhaLayer) {* *HILNenhaFrame ();* *}* *else {* *numLayer ++;* *HILNextFrame (numLayer);* *}* *}*		

Т а б л и ц а 42 — Синтаксис *HILNenhaFrame ()*

Синтаксис	Количество битов	Мнемоника
HILNenhaFrame () *{* *if (envFlag) {* *envTmaxEnha;* *envRatkEnha;* *envRdecEnha;* *}* *if (harmFlag) {* *HARMenhaPara ();* *}* *INDlenhaPara ();* *}*	 *tmEnhbits* *atkEnhbits* *decEnhbits*	 *uimsbf* *uimsbf* *uimsbf*

Т а б л и ц а 43 — Синтаксис *HARMenhaPara ()*

Синтаксис	Количество битов	Мнемоника
HARMenhaPara () *{* *for (i = 0; i <min (numHarmLine, 10); i ++) {* *harmFreqEnha [i];* *harmPhase [i];* *}* *}*	 *fEnhbits [i]* *phasebits* 	 *uimsbf* *uimsbf*

Т а б л и ц а 44 — Синтаксис *INDlenhaPara ()*

Синтаксис	Количество битов	Мнемоника
INDlenhaPara () *{* *for (i = 0; i <numLine; i ++) {* *lineFreqEnha [i];* *linePhase [i];* *}* *}*	 *fEnhbits [i]* *phasebits* 	 *uimsbf* *uimsbf*

Т а б л и ц а 45 — Синтаксис *HILNextFrame ()*

Синтаксис	Количество битов	Мнемоника
HILNextFrame (numLayer) *{* *layPrevNumLine [numLayer] = layNumLine [numLayer];* */* layPrevNumLine [numLayer] = 0 for the */* */* first bitstream frame */* *addNumLine [numLayer];* *if (phaseFlag) {* *layNumLinePhase [numLayer];* *}* *layNumLine [numLayer] = layNumLine [numLayer-1] +* *addNumLine [numLayer];* *for (k = 0; k <layPrevNumLine [numLayer-1]; k ++) {* *if (layPrevLineContFlag [numLayer-1] [k]) {* *layPrevLineContFlag [numLayer] [k] = 1;* *}* *else {* *layPrevLineContFlag [numLayer] [k];* *}* *}*	 *linebits* *linebits* 1 	 *uimsbf* *uimsbf* *uimsbf*

Окончание таблицы 45

Синтаксис	Количество битов	Мнемоника
for (k = layPrevNumLine [numLayer-1]; *k <layPrevNumLine [numLayer]; k ++) {* *layPrevLineContFlag [numLayer] [k];* *}* *i = layNumLine [numLayer-1];* *for (k = 0; k <layPrevNumLine [numLayer-1]; k ++) {* *if (! layPrevLineContFlag [numLayer-1] [k] &&* *layPrevLineContFlag [numLayer] [k]) {* *linePred [i] = k;* *lineContFlag [i ++] = 1* *}* *}* *for (k = layPrevNumLine [numLayer-1];* *k <layPrevNumLine [numLayer]; k ++) {* *if (layPrevLineContFlag [numLayer] [k]) {* *linePred [i] = k;* *lineContFlag [i ++] = 1;* *}* *}* *while (i <layNumLine [numLayer]) {* *lineContFlag [i ++] = 0;* *}* *INDlextPara (numLayer);* *if (phaseFlag) {* *INDlextPhasePara (numLayer);* *}* *}	1	*uimsbf*

Т а б л и ц а 46 — Синтаксис *INDlextPara ()*

Синтаксис	Количество битов	Мнемоника
INDlextPara (numLayer) *{* *lastNLFreq = 0;* *for (i = layPrevNumLine [numLayer-1];* *i <layPrevNumLine [numLayer]; i ++) {* *prevILFreqIndex [i] = ILFreqIndex [i];* *prevILAmplIndex [i] = ILAmplIndex [i];* *}* *for (i = layNumLine [numLayer-1];* *i <layNumLine [numLayer]; i ++) {* *if (envFlag) {*		

Окончание таблицы 46

Синтаксис	Количество битов	Мнемоника
lineEnvFlag [i];	1	*uimsbf*
}		
if (lineContFlag [i]) {		
DILFreq [i];	2.. 10	*DIF*
ILFreqIndex [i] = prevILFreqIndex [linePred [i]] + DILFreq [i];		
DILAmpl [i];	3.. 8	*DIA*
ILAmplIndex [i] = prevILAmplIndex [linePred [i]] +		
DILAmpl [i];		
}		
else {		
if (layNumLine [numLayer] - 1-i <7) {		
ILFreqInc [i];	0.. 14	*SDC*
/ SDCdecode (maxFindexlastNLFreq, */*		
/ sdcILFTable [layNumLine [numLayer] - 1-i]) */*		
}		
else {		
ILFreqInc [i];	0.. 14	*SDC*
/ SDCdecode (maxFindexlastNLFreq, */*		
/ sdcILFTable [7]) */*		
}		
ILFreqIndex [i]= lastNLFreq + ILFreqInc i];		
lastNLFreq = ILFreqIndex [i];		
if (HILNquantMode) {		
ILAmplRel [i];	4.. 10	*SDC*
/ SDCdecode (50, sdcILATable) */*		
ILAmplIndex [i] = maxAmplIndex + ILAmplRel [i];		
}		
else {		
ILAmplRel [i];	3.. 9	*SDC*
/ SDCdecode (25, sdcILATable) */*		
*ILAmplIndex [i] = maxAmplIndex + 2*ILAmplRel [i];*		
}		
}		
}		
}		

Т а б л и ц а 47 — Синтаксис *INDIextPhasePara ()*

Синтаксис	Количество битов	Мнемоника
INDIextPhasePara(numLayer)		
{		
j = 0;		

Окончание таблицы 47

Синтаксис	Количество битов	Мнемоника
for (i = layNumLine[numLayer-1]; i < layNumLine[numLayer]; *i++) {* *if (! linePred[i] && j < layNumLinePhase[numLayer]) {* *linePhase[i];* *linePhaseAvail[i] = 1;* *j++;* *}* *else {* *linePhaseAvail[i] = 0;* *}* *}* *}*	*phasebits*	*uimsbf*

3.2.3 Сборники кодов *HILN*

Т а б л и ц а 48 — Код LARH1 (harmLAR [0.. 1])

Кодовая комбинация	*harmLAR [i]*	Кодовая комбинация	*harmLAR [i]*
1000000000000000100	-6,350	0100	0,050
1000000000000000101	-6,250	0101	0,150
1000000000000000110	-6,150	0110	0,250
1000000000000000111	-6,050	0111	0,350
100000000000000100	-5,950	00100	0,450
100000000000000101	-5,850	00101	0,550
100000000000000110	-5,750	00110	0,650
100000000000000111	-5,650	00111	0,750
10000000000000100	-5,550	000100	0,850
10000000000000101	-5,450	000101	0,950
10000000000000110	-5,350	000110	1,050
10000000000000111	-5,250	000111	1,150
1000000000000100	-5,150	0000100	1,250
1000000000000101	-5,050	0000101	1,350
1000000000000110	-4,950	0000110	1,450
1000000000000111	-4,850	0000111	1,550
100000000000100	-4,750	00000100	1,650
100000000000101	-4,650	00000101	1,750
100000000000110	-4,550	00000110	1,850
100000000000111	-4,450	00000111	1,950
10000000000100	-4,350	000000100	2,050
10000000000101	-4,250	000000101	2,150
10000000000110	-4,150	000000110	2,250
10000000000111	-4,050	000000111	2,350

Окончание таблицы 48

Кодовая комбинация	*harmLAR [i]*	Кодовая комбинация	*harmLAR [i]*
1000000000100	-3,950	0000000100	2,450
1000000000101	-3,850	0000000101	2,550
1000000000110	-3,750	0000000110	2,650
1000000000111	-3,650	0000000111	2,750
100000000100	-3,550	00000000100	2,850
100000000101	-3,450	00000000101	2,950
100000000110	-3,350	00000000110	3,050
100000000111	-3,250	00000000111	3,150
10000000100	-3,150	000000000100	3,250
10000000101	-3,050	000000000101	3,350
10000000110	-2,950	000000000110	3,450
10000000111	-2,850	000000000111	3,550
1000000100	-2,750	0000000000100	3,650
1000000101	-2,650	0000000000101	3,750
1000000110	-2,550	0000000000110	3,850
1000000111	-2,450	0000000000111	3,950
100000100	-2,350	00000000000100	4,050
100000101	-2,250	00000000000101	4,150
100000110	-2,150	00000000000110	4,250
100000111	-2,050	00000000000111	4,350
10000100	-1,950	000000000000100	4,450
10000101	-1,850	000000000000101	4,550
10000110	-1,750	000000000000110	4,650
10000111	-1,650	000000000000111	4,750
1000100	-1,550	0000000000000100	4,850
1000101	-1,450	0000000000000101	4,950
1000110	-1,350	0000000000000110	5,050
1000111	-1,250	0000000000000111	5,150
100100	-1,150	00000000000000100	5,250
100101	-1,050	00000000000000101	5,350
100110	-0,950	00000000000000110	5,450
100111	-0,850	00000000000000111	5,550
10100	-0,750	000000000000000100	5,650
10101	-0,650	000000000000000101	5,750
10110	-0,550	000000000000000110	5,850
10111	-0,450	000000000000000111	5,950
1100	-0,350	0000000000000000100	6,050
1101	-0,250	0000000000000000101	6,150
1110	-0,150	0000000000000000110	6,250
1111	-0,050	0000000000000000111	6,350

Т а б л и ц а 49 — Код *LARH*2 *(harmLAR* [2.. 6])

Кодовая комбинация	*harmLAR[.]*	Кодовая комбинация	*harmLAR[.]*
100000000000000010	-4,725	010	0,075
100000000000000011	-4,575	011	0,225
10000000000000010	-4,425	0010	0,375
10000000000000011	-4,275	0011	0,525
1000000000000010	-4,125	00010	0,675
1000000000000011	-3,975	00011	0,825
100000000000010	-3,825	000010	0,975
100000000000011	-3,675	000011	1,125
10000000000010	-3,525	0000010	1,275
10000000000011	-3,375	0000011	1,425
1000000000010	-3,225	00000010	1,575
1000000000011	-3,075	00000011	1,725
100000000010	-2,925	000000010	1,875
100000000011	-2,775	000000011	2,025
10000000010	-2,625	0000000010	2,175
10000000011	-2,475	0000000011	2,325
1000000010	-2,325	00000000010	2,475
1000000011	-2,175	00000000011	2,625
100000010	-2,025	000000000010	2,775
100000011	-1,875	000000000011	2,925
10000010	-1,725	0000000000010	3,075
10000011	-1,575	0000000000011	3,225
1000010	-1,425	00000000000010	3,375
1000011	-1,275	00000000000011	3,525
100010	-1,125	000000000000010	3,675
100011	-0,975	000000000000011	3,825
10010	-0,825	0000000000000010	3,975
10011	-0,675	0000000000000011	4,125
1010	-0,525	00000000000000010	4,275
1011	-0,375	00000000000000011	4,425
110	-0,225	000000000000000010	4,575
111	-0,075	000000000000000011	4,725

Т а б л и ц а 50 — Код *LARH*3 *(harmLAR* [7.. 24])

Кодовая комбинация	*harmLAR[.]*	Кодовая комбинация	*harmLAR[.]*
10000000000000001	-2,325	01	0,075
1000000000000001	-2,175	001	0,225
100000000000001	-2,025	0001	0,375
10000000000001	-1,875	00001	0,525
1000000000001	-1,725	000001	0,675
100000000001	-1,575	0000001	0,825

Окончание таблицы 50

Кодовая комбинация	*harmLAR[.]*	Кодовая комбинация	*harmLAR[.]*
10000000001	-1,425	00000001	0,975
1000000001	-1,275	000000001	1,125
100000001	-1,125	0000000001	1,275
10000001	-0,975	00000000001	1,425
1000001	-0,825	000000000001	1,575
100001	-0,675	0000000000001	1,725
10001	-0,525	00000000000001	1,875
1001	-0,375	000000000000001	2,025
101	-0,225	0000000000000001	2,175
11	-0,075	00000000000000001	2,325

Т а б л и ц а 51 — Код *LARN1 (noiseLAR* [0,1])

Кодовая комбинация	*noiseLAR[.]*	Кодовая комбинация	*noiseLAR[.]*
10000000000000001	-4,65	01	0,15
1000000000000001	-4,35	001	0,45
100000000000001	-4,05	0001	0,75
10000000000001	-3,75	00001	1,05
1000000000001	-3,45	000001	1,35
100000000001	-3,15	0000001	1,65
10000000001	-2,85	00000001	1,95
1000000001	-2,55	000000001	2,25
100000001	-2,25	0000000001	2,55
10000001	-1,95	00000000001	2,85
1000001	-1,65	000000000001	3,15
100001	-1,35	0000000000001	3,45
10001	-1,05	00000000000001	3,75
1001	-0,75	000000000000001	4,05
101	-0,45	0000000000000001	4,35
11	-0,15	00000000000000001	4,65

Т а б л и ц а 52 — Код *LARN2 (noiseLAR* [2.. 24])

Кодовая комбинация	*noiseLAR[.]*	Кодовая комбинация	*noiseLAR[.]*
110000000000000001	-6,35	101	0,35
11000000000000001	-5,95	1001	0,75
1100000000000001	-5,55	10001	1,15
110000000000001	-5,15	100001	1,55
11000000000001	-4,75	1000001	1,95
1100000000001	-4,35	10000001	2,35
110000000001	-3,95	100000001	2,75
11000000001	-3,55	1000000001	3,15
1100000001	-3,15	10000000001	3,55

Окончание таблицы 52

Кодовая комбинация	*noiseLAR[.]*	Кодовая комбинация	*noiseLAR[.]*
110000001	-2,75	100000000001	3,95
11000001	-2,35	1000000000001	4,35
1100001	-1,95	10000000000001	4,75
110001	-1,55	100000000000001	5,15
11001	-1,15	1000000000000001	5,55
1101	-0,75	10000000000000001	5,95
111	-0,35	100000000000000001	6,35
0	0,00		

Т а б л и ц а 53 — Код *DIA*

Кодовая комбинация	Значение	Кодовая комбинация	Значение
111 1 1111	-25	001	1
111 1 1110	-24	011 0	2
111 1 1101	-23	100 0	3
111 1 *xxxx*	-*y*	101 0 0	4
111 1 0001	-11	101 0 1	5
111 1 0000	-10	110 0 00	6
110 1 11	-9	110 0 01	7
110 1 10	-8	110 0 10	8
110 1 01	-7	110 0 11	9
110 1 00	-6	111 0 0000	10
101 1 1	-5	111 0 0001	11
101 1 0	-4	111 0 xxxx	*y*
100 1	-3	111 0 1101	23
011 1	-2	111 0 1110	24
010	-1	111 0 1111	25
000	0		

Т а б л и ц а 54 — Код *DIF*

Кодовая комбинация	Значение	Кодовая комбинация	Значение
11 11 1 11111	-42	01 0	1
11 11 1 11110	-41	10 0 0	2
11 11 1 11101	-40	10 0 1	3
11 11 1 xxxxx	-y	11 00 0	4
11 11 1 00001	-12	11 01 0 0	5
11 11 1 00000	-11	11 01 0 1	6
11 10 1 11	-10	11 10 0 00	7
11 10 1 10	-9	11 10 0 01	8
11 10 1 01	-8	11 10 0 10	9
11 10 1 00	-7	11 10 0 11	10
11 01 1 1	-6	11 11 0 00000	11

Окончание таблицы 54

Кодовая комбинация	Значение	Кодовая комбинация	Значение
11 01 1 0	-5	11 11 0 00001	12
11 00 1	-4	11 11 0 xxxxx	*y*
10 1 1	-3	11 11 0 11101	40
10 1 0	-2	11 11 0 11110	41
01 1	-1	11 11 0 11111	42
00	0		

Таблица 55 — Код *DHF*

Кодовая комбинация	Значение	Кодовая комбинация	Значение
11 1 111111	-69	01 0	1
11 1 111110	-68	10 0 00	2
11 1 111101	-67	10 0 01	3
11 1 xxxxxx	-*y*	10 0 10	4
11 1 000001	-7	10 0 11	5
11 1 000000	-6	11 0 000000	6
10 1 11	-5	11 0 000001	7
10 1 10	-4	11 0 xxxxxx	*y*
10 1 01	-3	11 0 111101	67
10 1 00	-2	11 0 111110	68
01 1	-1	11 0 111111	69
00	0		

Таблица 56 — Код *HFS*

Кодовая комбинация	Значение	Кодовая комбинация	Значение
1 1 1 1111	-17	1 0 0	1
1 1 1 1110	-16	1 0 1 0000	2
1 1 1 1101	-15	1 0 1 0001	3
1 1 1 xxxx	-*y*	1 0 1 xxxx	*y*
1 1 1 0001	-3	1 0 1 1101	15
1 1 1 0000	-2	1 0 1 1110	16
1 1 0	-1	1 0 1 1111	17
0	0		

Примечания к таблицам 53—56 — Группировка битов в пределах кодовой комбинации (например, “1 1 1 1111”) обеспечивается только для лучшей удобочитаемости. Кодовые комбинации, не перечисленные явно в сборниках кодов (например, “1 1 1 xxxx”), определяются постепенным увеличением неявной части кодовой комбинации “xxxx” (*uimsbf*) и величины “*y*” соответствующего значения. Во всех случаях кодовые комбинации и значения для двух самых маленьких и трех самых больших величин перечисляются явно.

3.2.4 *HILN SubDivisionCode (SDC)*

Код *SubDivisionCode (SDC)* является алгоритмически сгенерированным кодом изменяемой длины, основанным на данной таблице и данном числе различных кодовых комбинаций.

Идея этой схемы кодирования заключается в разделении функции плотности вероятности на две части, которые представляют равную вероятность. Один бит передается, чтобы определять расположение части, значение которой будет кодировано. Это разделение повторяется, пока ширина части не единица, и затем ее позиция равна позиции кодируемого значения. Позиции границ извлекаются из таблицы 32 квантованных значений с фиксированной точкой. Помимо этой таблицы (параметр *«tab»*) также необходимо число различных кодовых комбинаций (параметр *"k"*).

Следующая функция *C SDCDecode (k, tab)* вместе с этими 9 таблицами *sdclLATable* [32] и *sdclLFTable* [8] [32] описывает декодирование. Функция *GetBit ()* возвращает следующий бит в потоке.

```
int sdclLATable [32] = {
      0, 13, 27, 41, 54, 68, 82, 96, 110, 124, 138, 152, 166, 180, 195, 210, 225, 240, 255, 271, 288, 305, 323,
      342, 361, 383, 406, 431, 460, 494, 538, 602
};
   int sdclLFTable [8] [32] = {
      {   0, 53, 87, 118, 150, 181, 212, 243, 275, 306, 337, 368, 399, 431, 462, 493, 524, 555, 587, 618, 649,
          680, 711, 743, 774, 805, 836, 867, 899, 930, 961, 992},
      {   0, 34, 53, 71, 89, 106, 123, 141, 159, 177, 195, 214, 234, 254, 274, 296, 317, 340, 363, 387, 412,
          438, 465, 494, 524, 556, 591, 629, 670, 718, 774, 847},
      {   0, 26, 41, 54, 66, 78, 91, 103, 116, 128, 142, 155, 169, 184, 199, 214, 231, 247, 265, 284, 303, 324,
          346, 369, 394, 422, 452, 485, 524, 570, 627, 709},
      {   0, 23, 35, 45, 55, 65, 75, 85, 96, 106, 117, 128, 139, 151, 164, 177, 190, 204, 219, 235, 252, 270,
          290, 311, 334, 360, 389, 422, 461, 508, 571, 665},
      {   0, 20, 30, 39, 48, 56, 64, 73, 81, 90, 99, 108, 118, 127, 138, 149, 160, 172, 185, 198, 213, 228, 245,
          263, 284, 306, 332, 362, 398, 444, 507, 608},
      {   0, 18, 27, 35, 43, 50, 57, 65, 72, 79, 87, 95, 104, 112, 121, 131, 141, 151, 162, 174, 187, 201, 216,
          233, 251, 272, 296, 324, 357, 401, 460, 558},
      {   0, 16, 24, 31, 38, 45, 51, 57, 64, 70, 77, 84, 91, 99, 107, 115, 123, 132, 142, 152, 163, 175, 188, 203,
          219, 237, 257, 282, 311, 349, 403, 493},
      {   0, 12, 19, 25, 30, 35, 41, 46, 51, 56, 62, 67, 73, 79, 85, 92, 99, 106, 114, 122, 132, 142, 153, 165,
          179, 195, 213, 236, 264, 301, 355, 452}
   };
   int SDCDecode (int k, int * tab)
   {
      {int *pp;
      int g, dp, min, max;
      min = 0;
      max = k-1;
      pp = tab+16;
      dp = 16;
      while (min ! = max);
      {
         If (dp) g = (k * ( * pp)) > > 10; else g = (max +min) >> 1;
         dp >> = 1;
         if (GetBit()  = = 0) { pp - = dp; max = g;} else {pp+ = dp;min = g + 1;}
      }
      return max;
   }
```

4 Семантика потока битов

4.1 Конфигурация декодера (ParametricSpecificConfig)

Элементы потока битов:

isBaseLayer — Одноразрядный идентификатор, представляющий, является ли соответствующий уровень базовым уровнем (1) или уровнем улучшения или расширения (0).

4.1.1 Конфигурация параметрического декодера аудио

Элементы потока битов:

PARAmode — 2-битовое поле, указывающее режим работы параметрического кодера.

PARAextensionFlag — Флаг, указывающий на присутствие данных *MPEG* 4 версии 3 (для будущего использования).

4.1.2 Конфигурация декодера *HILN*

Элементы потока битов:

HILNquantMode — 1-битовое поле, указывающее на режим квантователя с отдельной линией.

HILNmaxNumLine — 8-битовое поле, указывающее максимальное количество отдельных линий в фрейме потока битов. Это также определяет размер поля linebits.

HILNsampleRateCode — 4-битовое поле, указывающее частоту дискретизации, используемую для декодирования параметра *HILN.*

HILNframeLength — 12-битовое поле, указывающее длину фрейма *HILN*, в выборках при частоте дискретизации, указанной *HILNsampleRateCode*.

HILNcontMode — 2-битовое поле, указывающее режим продолжения дополнительной линии декодера.

HILNenhaLayer — Флаг, указывающий, содержит ли этот поток *Elementaru Stream* уровень улучшения или уровень расширения.

HILNenhaQuantMode — 2-битовое поле, указывающее режим квантователя с улучшением частоты.

4.2 Фрейм потока битов *(slPacketPayload)*

4.2.1 Фрейм параметрического потока битов аудио

Элементы потока битов:

PARAswitchMode — Флаг, указывающий, какой инструмент кодирования используется в текущем фрейме потока битов с переключением *HVXC/HILN.*

PARAmixMode — 2-битовое поле, указывающее, какие инструменты кодирования используются в текущем фрейме потока битов со смешиванием *HVXC/HILN*.

4.2.2 Фрейм потока битов *HILN*

Элементы потока битов:

numLine — Поле, указывающее число отдельных линий в текущем фрейме.

harmFlag — Флаг, указывающий на присутствие данных гармонических линий в текущем фрейме.

noiseFlag — Флаг, указывающий на присутствие данных шумовых компонентов в текущем фрейме.

phaseFlag — Флаг, указывающий на присутствие данных фазы начала линии в текущем фрейме.

numLinePhase — Поле, указывающее число отдельных линий с фазой запуска в текущем фрейме.

maxAmplndexCoded — Поле, указывающее максимальную амплитуду нового компонента сигнала в текущем фрейме.

envFlag — Флаг, указывающий на присутствие данных огибающей в текущем фрейме.

envTmax — Кодированный параметр огибающей: время максимума.

envRatk — Кодированный параметр огибающей: скорость атаки.

envRdec — Кодированный параметр огибающей: скорость затухания.

prevLineContFlag [k] — Флаг, указывающий, что k-я отдельная линия предыдущего фрейма продолжается в текущем фрейме.

numHarmParaIndex — Поле, указывающее на число параметров *LPC* гармонических линий в текущем фрейме.

numHarmLineIndex — Поле, указывающее на число гармонических линий в текущем фрейме.

harmContFlag — Флаг, указывающий, что гармонические линии продолжаются из предыдущего фрейма.

numHarmPhase	Поле, указывающее число гармонических линий с фазой старта в текущем фрейме.
harmEnvFlag	Флаг, указывающий, что к гармоническим линиям применяется огибающая амплитуды.
contHarmAmpl	Кодированное изменение амплитуды гармонических линий.
contHarmFreq	Кодированное изменение основной частоты гармонических линий.
harmAmplRel	Кодированная относительная амплитуда гармонических линий.
harmFreqIndex	Кодированная основная частота гармонических линий.
harmFreqStretch	Кодированный параметр растяжения частоты гармонических линий.
harmLAR [i]	Кодированные параметры *LAR LPC* гармонических линий.
harmPhase [i]	Кодированная фаза *i*-й гармонической линии.
numNoiseParaIndex	Поле, указывающее на число шумовых параметров LPC в текущем фрейме.
noiseContFlag	Флаг, указывающий, что шум продолжается из предыдущего фрейма.
noiseEnvFlag	Флаг, указывающий, что данные огибающей шума присутствуют в текущем фрейме.
contNoiseAmpl	Кодированное изменение амплитуды шума.
noiseAmplRel	Кодированная относительная амплитуда шума.
noiseEnvTmax	Кодированный параметр огибающей шума: время максимума
noiseEnvRatk	Кодированный параметр огибающей шума: темп нарастания.
noiseEnvRdec	Кодированный параметр огибающей шума: темп спада.
noiseLAR [i]	Кодированные параметры *LAR LPC* шума.
lineEnvFlag [i]	Флаг, указывающий, что огибающая амплитуды применяется к *i*-й отдельной линии.
DILFreq [i]	Кодированное изменение частоты *i*-й отдельной линии.
DILAmpl [i]	Кодированное изменение амплитуды *i*-й отдельной линии.
ILFreqInc [i]	Кодированный инкремент частоты *i*-й отдельной линии.
ILAmplRel [i]	Кодированная относительная амплитуда *i*-й отдельной линии.
linePhase [i]	Кодированная фаза *i*-й отдельной линии.
envTmaxEnha	Кодированный параметр улучшения огибающей: время максимума.
envRatkEnha	Кодированный параметр улучшения огибающей: темп нарастания.
envRdecEnha	Кодированный параметр улучшения огибающей: темп затухания.
harmFreqEnha [i]	Кодированное улучшение частоты линии *i*-й гармоники.
lineFreqEnha [i]	Кодированное улучшение частоты *i*-й отдельной линии.
addNumLine [i]	Поле, указывающее число отдельных линий в уровне расширения *i* текущего фрейма.
layNumLinePhase [i]	Поле, указывающее число отдельных линий с фазой старта в уровне расширения *i* текущего фрейма.
layPrevLineContFlag [i] [k]	Флаг, указывающий, что *k*-я отдельная линия предыдущего фрейма продолжается в уровне расширения *i* текущего фрейма.

Элементы справки:

numLayer	Число доступных уровней расширения (0, если доступен только базовый уровень).
layNumLine [i]	Общее количество отдельных линий в текущем фрейме, которые переданы в базовом уровне и первых *i* уровнях расширения.
prevNumLine	Число отдельных линий в предыдущем фрейме.
layPrevNumLine [i]	Общее количество отдельных линий в предыдущем фрейме, переданных в базовом уровне и первых *i* уровнях расширения.
maxAmplIndex	Максимальная амплитуда нового компонента сигнала в текущем фрейме.
linePred [i]	Индекс предшествующего элемента в предыдущем фрейме для *i*-й отдельной линии в текущем фрейме.

lineContFlag [i]	Флаг, указывающий, что линия i в текущем фрейме продолжается из предыдущего фрейма.
numHarmPara	Число параметров *LPC* гармонической линии в текущем фрейме.
numHarmLine	Число гармонических линий в текущем фрейме.
harmAmpIIndex	Индекс амплитуды гармонических линий в текущем фрейме.
harmFreqIndex	Индекс основной частоты гармонических линий в текущем фрейме.
prevHarmAmpIIndex	Индекс амплитуды гармонических линий в предыдущем фрейме.
prevHarmFreqIndex	Индекс основной частоты гармонических линий в предыдущем фрейме.
harmPhaseAvail [i]	Флаг, указывающий, что доступна информация о стартовой фазе для *i*-й гармонической линии.
numNoisePara	Число параметров *LPC* шума в текущем фрейме.
noiseAmpIIndex	Индекс амплитуды шума в текущем фрейме.
prevNoiseAmpIIndex	Индекс амплитуды шума в предыдущем фрейме.
lastNLFreq	Аккумулятор инкремента частоты отдельной линии.
ILFreqIndex [i]	Индекс частоты *i*-й отдельной линии в текущем фрейме.
ILAmpIIndex [i]	Индекс амплитуды *i*-й отдельной линии в текущем фрейме.
prevILFreqIndex [i]	Индекс частоты *i*-й отдельной линии в предыдущем фрейме.
prevILAmpIIndex [i]	Индекс амплитуды *i*-й отдельной линии в предыдущем фрейме.
linePhaseAvail [i]	Флаг, указывающий, что доступна информация о стартовой фазе для *i*-й отдельной линии.
linebits	Число битов для *numLine.*
tmbits	Число битов для *envTmax.*
atkbits	Число битов для *encRatk.*
decbits	Число битов для *envRdec.*
tmEnhbits	Число битов для *envTmaxEnha.*
atkEnhbits	Число битов для *encRatkEnha.*
decEnhbits	Число битов для *envRdecEnha.*
fEnhbits [i]	Число битов для *lineFreqEnha [i]* и *harmFreqEnha [i]*
phasebits	Число битов для *linePhase* и *harmPhase.*

5 Инструменты параметрического декодера

5.1 Инструменты декодера HILN

Декодер гармонических и отдельных линий и шум (*HILN*) используют ряд параметров, которые кодируются в потоке битов, чтобы описать аудиосигнал.

Поддерживаются три различных модели сигнала (таблица 57).

Таблица 57 — Модели сигнала *HILN*

Модель сигнала	Описание	Существенные параметры
Гармонические линии	Группа синусоидальных сигналов с общей основной частотой	Основная частота и амплитуды линий спектра
Отдельные линии	Синусоидальные сигналы	Частота и амплитуда отдельных линий спектра
Шум	Шумовой сигнал спектральной формы	Форма спектра и энергия шума

Декодер *HILN* сначала восстанавливает эти параметры из потока битов с помощью ряда инструментов декодирования, а затем синтезирует аудиосигнал на базе этих параметров, используя ряд инструментов синтезатора:

декодер гармонической линии;
декодер отдельной линии;
декодер шума;
синтезатор гармонических и отдельных линий;
синтезатор шума.

Инструменты декодера *HILN* восстанавливают из потока битов параметры гармонических и отдельных линий (частота, амплитуда) и шум (форма спектра), а также возможные параметры огибающей.

Инструменты синтезатора *HILN* воссоздают один фрейм аудиосигнала, основываясь на параметрах, декодированных инструментами декодера *HIL*N для текущего фрейма потока битов.

Выборки декодируемого аудиосигнала имеют полномасштабный диапазон [–32768, 32767], и возможные выбросы должны быть ограничены до этих значений.

Декодер *HILN* поддерживает широкий диапазон длин фрейма и частот дискретизации. Масштабируя длину фрейма синтезатора с произвольным коэффициентом, в декодере достигается доступность функциональности с изменением скорости. Масштабируя частоты линий и передискретизируя шумовой сигнал с произвольным коэффициентом, в декодере обеспечивается доступность функциональности изменения шага.

Декодер *HILN* может работать в двух различных режимах, как основной декодер и как улучшенный декодер. Основной декодер, который используется для нормальной работы, только оценивает информацию, доступную в элементах потока битов *HILNbasicFrame ()*, чтобы восстановить аудиосигнал. Чтобы позволить большую масштабируемость шага в комбинации с другими инструментами кодера, должны быть переданы дополнительные элементы потока битов *HILNenhaFrame ()*, и декодер *HILN* должен работать в расширенном режиме, который использует информацию как *HILNbasicFrame ()*, так и *HILNenhaFrame ()*. Этот режим восстанавливает аудиосигнал с четко определенными фазовыми соотношениями, который может быть объединен с остаточным сигналом, кодированным при более высоких скоростях передачи, используя кодер улучшения. Если декодер *HILN* используется таким образом в качестве ядра для масштабируемого кодера, никакой шумовой сигнал не должен быть синтезирован для сигнала, который подается декодеру улучшения.

В силу представления параметрического сигнала, используемого параметрическим кодером *HILN*, это хорошо подходит для приложений, требующих кодирование с масштабируемой скоростью передачи. Кодирование *HILN* с масштабируемой скоростью передачи выполняется путем добавления к данным, закодированным в *HILNbasicFrame ()* основного потока битов, данных, закодированных в одном или более *HILNextFrame ()* одного или нескольких потоков битов расширения, переданных, как дополнительные элементарные потоки.

5.1.1 Декодер гармонический линии

5.1.1.1 Описание инструмента

Этот инструмент декодирует параметры гармонических линий, переданных в потоке битов.

5.1.1.2 Определения

prevNumHarmPara	Число параметров *LPC* гармонической линии в предыдущем фрейме.
harmLPCPara [i]	Параметр *LPC* гармонической линии *i* в текущем фрейме (*LARs* для спектра гармонического тона).
prevHarmLPCPara [i]	Параметр *LPC* гармонической линии *i* в предыдущем фрейме (*LARs* для спектра гармонического тона).
hFreq	Основная частота гармонических линий.
hStretch	Растяжение частоты гармонических линий.
harmAmpl	Амплитуда гармонического тона.
harmPwr	Мощность гармонического тона.
hLineAmpl [i]	Амплитуда *i*-й гармонической линии.
hLineFreq [i]	Частота *i*-й гармонической линии, Гц.
hLineAmplEnh [i]	Улучшенная амплитуда i-й гармонической линии.
hLineFreqEnh [i]	Улучшенная частота *i*-й гармонической линии, Гц.
hLinePhaseEnh [i]	Фаза *i*-й гармонической линии (в радианах).

ha [i]	Немасштабированная амплитуда *i*-й гармонической линии.
r [i]	Коэффициенты отражения *LPC*.
h [i]	Импульсная характеристика *LPC*.
H (i)	Системная функция *LPC*.

5.1.1.3. Процесс декодирования

Если *harmFlag* устанавливается и, таким образом данные *HARMbasicPara ()*, а в режиме улучшения и данные *HARMenhaPara ()*, доступны в текущем фрейме, параметры гармонических линий декодируются и деквантуются следующим образом.

5.1.1.3.1 Основной декодер

Гармонический тон представляется его основной частотой, его энергией и рядом *LPC*-параметров.

Сначала восстанавливаются параметры *harmNumPara LAR*. Когда устанавливается *harmContFlag*, используется прогноз из предыдущего фрейма.

```
Float harmLPCMean [25] = {5,0,-1,5 , 0,0 , 0,0 , 0,0..., 0,0};
Float harmPredCoeff [25] = {0,75 , 0,75 , 0,5 , 0,5 , 0,5..., 0,5};
for (i = 0; i <numHarmPara; i ++) {
    if (i <prevNumHarmPara && harmContFlag)
    pred = harmLPCMean [i] +
    (prevHarmLPCPara [i]-harmLPCMean [i]) *harmPredCoeff [i];
    else
    pred = harmLPCMean [i];
    harmLPCPara [i] = pred + harmLAR [i];
}
```

Параметры, необходимые в следующем фрейме, сохраняются в межфреймовой памяти:

```
prevNumHarmPara = numHarmPara;
for (i = 0; i <numHarmPara; i ++)
prevHarmLPCPara [i] = harmLPCPara [i];
```

Основная частота и протяжение гармонических линий являются деквантованными:

```
hFreq = 20 * exp (log (4000./20) * (harmFreqIndex+0,5) / 2048,0);
hStretch = harmFreqStretch / 16000,0;
```

Амплитуда и мощность гармонического тона деквантуются следующим образом:

```
harmAmpl = 32768 * pow (10,-1,5 * (harmAmplIndex+0,5)/20);
harmPwr = harmAmpl*harmAmpl;
```

Флаги *harmEnvFlag* и *harmContFlag* не требуют дальнейшей деквантизации; они непосредственно передаются на инструмент синтезатора.

Параметры *LPC* передаются в форме “Логарифмических коэффициентов области” (*LAR*). После декодирования параметров частоты и амплитуды частей *harmNumLine* гармонического тона вычисляются следующим образом.

Частоты гармонических линий вычисляются так:

```
for (i = 0; i <numHarmLine; i ++)
hLineFreq [i] = hFreq * (i+1) * (1 + hStretch * (i+1)).
```

Параметры *LPC* представляют *IIR*-фильтр. Амплитуды синусоид получаются вычислением абсолютного значения системной функции этого фильтра *H(z)* на соответствующих частотах.

```
for (i = 0; i <numHarmLine; i ++)
ha [i] = abs (H (exp (j *pi * (i+1) / (numHarmLine+1)));
c j = sqrt (-1) и
H (z) = 1 / (1 - h [0] *pow (z,-1) - h [1] *pow (z,-2)-... - h [numHarmPara-1] *pow (z, - numHarmPara))
```

Импульсная характеристика *h [i]* вычисляется из *LARs* по следующему алгоритму.

В первом шаге *LARs* преобразовываются в коэффициенты отражения:

```
for (i = 0; i <numHarmPara; i ++)
r [i] = (exp (harmLPCPara [i]) - 1) / (exp (harmLPCPara [i]) + 1).
```

После этого коэффициенты отражения преобразовываются в характеристику времени. Функция *C* делает это преобразование по месту (вызов *c x [i] =r [i]* и N=*numHarmPara*; возвраты с *x [i] =h [i]):*

```
void Convert_k_to_h (float *x, int N)
{
    int i, j;
    float a, b, c;
    for (i = 1; i <N; i ++)
    {
        c = x [i];
        for (j = 0; j <i-j-1; j ++)
        {
            a = x [j];
            b = x [i-j-1];
            x [j] = a-c*b;
            x [i-j-1] = b-c*a;
        }
        if (j == i-j-1)
        x [j] - = c*x [j];
    }
}
```

После вычисления амплитуд *ha [i]* они должны быть нормализованы и умножены на *harmAmp*l, чтобы найти амплитуды гармонической линии, удовлетворяющие условию:

*sum (hLineAmpl [i] *hLineAmpl [i]) = power of harmonic tone*

Это реализуется следующим образом:

```
p = 0,0
for (i = 0; i <numHarmLine; i ++)
p + = ha [i] *ha [i];
s = sqrt (harmPwr / p);
for (i = 0; i <numHarmLine; i ++)
hLineAmpl [i] = ha [i] * s;
```

Дополнительная информация о фазе декодируется следующим образом:

```
for (i = 0; i <numHarmLine; i ++) {
        if (harmPhaseAvail [i]) {
            hStartPhase [i] = 2*pi * (harmPhase [i] +0,5 / (1 <<phasebits) - pi;
            hStartPhaseAvail [i] = 1;
        }
        else
        hStartPhaseAvail [i] = 0;
}
```

5.1.1.3.2 Декодер улучшения

В этом режиме параметры гармонической линии, декодированные основным декодером, уточняются, а также декодируются фазы линии, используя информацию, содержавшуюся в *HARMenhaPara ()* следующим образом:

Для первых максимум 10 гармонических линий *i*

i = 0….. min *(numHarmLine,* 10*)*-1

вычисляются улучшенные параметры гармонической линии, используя базовые параметры гармонической линии и данные в потоке битов улучшения:

hLineAmplEnh [i] = hLineAmpl [i];
*hLineFreqEnh [i] = hLineFreq [i] **
*(1 + ((harmFreqEnh [i] +0,5 / (1 <<fEnhbits [ij])- 0,5) * (hFreqRelStep-1));*

где *hFreqRelStep* является отношением двух соседних шагов квантователя основной частоты:
*hFreqRelStep = exp (log (*4000/20)/2048*)).*

Для обоих типов линии фаза декодируется из потока битов улучшения:
hLinePhaseEnh [i] = 2**pi * (harmPhase [i]* +0,5*) / (*1 *<<phasebits) - pi*

5.1.2 Декодер отдельной линии

5.1.2.1 Описание инструмента

Основной декодер потока битов отдельной линии восстанавливает параметры линии, частоту, амплитуду, и огибающую из потока битов. Декодер улучшения потока битов восстанавливает параметры линии, частоту, амплитуду, и огибающую с более тонким квантованием и дополнительно восстанавливает фазу параметров линии.

5.1.2.2 Определения

t_max	Параметр огибающей: время максимума.
r_atk	Параметр огибающей: темп нарастания.
r_dec	Параметр огибающей: темп спада.
ampl [i]	Амплитуда *i*-й отдельной линии.
частота [i]	Частота *i*-й отдельной линии, Гц.
startPhase [i]	Стартовая фаза *i*-й отдельной линии.
startPhaseAvail [i]	Флаг, указывающий, что доступна информация о стартовой фазе для *i*-й отдельной линии.
t_maxEnh	Параметр улучшенной огибающей: время максимума.
r_atkEnh	Параметр улучшенной огибающей: темп нарастания.
r_decEnh	Параметр улучшенной огибающей: темп спада.
amplEnh [i]	Улучшенная амплитуда *i*-й отдельной линии.
freqEnh [i]	Улучшенная частота *i*-й отдельной линии, Гц.
phaseEnh [i]	Фаза *i*-й отдельной линии (в радианах).

5.1.2.3 Процесс декодирования

5.1.2.3.1 Основной декодер

Основной декодер восстанавливает параметры линии из данных, содержавшихся в *HILNbasicFrame ()* и *INDIbasicPara ()* следующим образом.

Для каждого фрейма сначала из *HILNbasicFrame ()* читается число отдельных линий, закодированных в этом фрейме:

numLine.

Затем из *HILNbasicFrame ()* читается флаг огибающей фрейма:

envFlag.

Если *envFlag* = 1, тогда из *HILNbasicFrame ()* декодируются 3 параметра огибающей, *t_max, r_atk* и *r_dec*:

t_max = (envTmax+0,5*) / (*1 *<<tmbits);*
*r_atk = tan (pi/*2**max (*0*, envRatk*-0,5*) / ((*1 *<<atkbits)*-1*))/*0,2

*r_dec = tan (pi/2*max (0, envRdec*-0,5*) / ((1 <<decbits)*-1*))*/0,2

Эти параметры огибающей действительны для гармонических линий, а также для отдельных линий. Таким образом, параметры огибающей *envTmax, envRatk, envRdec* должны быть деквантованы, даже если *numLine* == 0.

Для каждой линии *k* предыдущего фрейма

k = 0 .. *prevNumLine-1*

флаг продолжения предыдущей линии читается из *HILNbasicFrame ()*:

prevLineContFlag [k]

Если *prevLineContFlag [k]* == 1, тогда линия *k* предыдущего фрейма продолжается в текущем фрейме. Если *prevLineContFlag [k]* == 0, тогда линия *k* предыдущего фрейма не продолжается.

В текущем фрейме сначала параметры всех продолжающихся линий кодируются, сопровождаемые параметрами новых линий. Поэтому флаг продолжения линии и предшественник линии определяются прежде, чем декодировать параметры линии:

```
i = 0;
for (k = 0; k <prevNumLine; k ++)
if (prevLineContFlag [k]) {
    linePred [i] = k;
    lineContFlag [i ++] = 1;
}
while (i <numLine)
lineContFlag [i ++] = 0;
```

Параметры новых линий кодируются с увеличивающимся индексом частоты, используя схему дифференциального кодирования. Поэтому единожды для каждого фрейма требуется следующая инициализация:

lastNLFreq = 0.

Для каждой линии i текущего фрейма

i = 0. *numLine*-1

параметры линии теперь декодируются из *INDIbasicPara ()*.

Если *envFlag* = = 1, тогда флаг огибающей линии читается из *INDIbasicPara ()*:

lineEnvFlag [i]

Если *lineContFlag [i]* == 1, тогда параметры продолжающейся линии декодируются из *INDIbasicPara ()* на базе амплитуды и индекса частоты ее предшественника в предыдущем фрейме:

ILFreqIndex [i] = prevILFreqIndex [linePred [i]] + DILFreq [i];

ILAmplIndex [i] = prevILAmplIndex [linePred [i]] + DILAmpl [i];

Если *lineContFlag [i]* == 0, тогда параметры новой линии декодируются из *INDIbasicPara ()*:

```
if (numLine-1-i <7)
ILFreqInc [i] = SDCdecode (maxFindex-lastNLFreq, sdclLFTable [numLine-1-i]);
else
ILFreqInc [i] = SDCdecode (maxFindex-lastNLFreq, sdclLFTable [7]);
ILFreqIndex [i] = lastNLFreq + ILFreqInc [i];
lastNLFreq = ILFreqIndex [я];
if (HILNquantMode) {
    ILAmplRel [i] = SDCdecode (50, sdclLATable);
    ILAmplIndex [i] = maxAmplIndex + ILAmplRel [i];
}
else {
    ILAmplRel [i] = SDCdecode (25, sdclLATable);
    ILAmplIndex [i] = maxAmplIndex + 2*ILAmplRel [i];
}
```

Индексы параметров линии сохраняются для декодирования параметров линии следующего фрейма:

```
prevNumLine = numLine;
for (i = 0; i <prevNumLine; i ++) {
    prevILFreqIndex [i] = ILFreqIndex [i];
    prevILAmplIndex [i] = ILAmplIndex [i];
}
```

Основной декодер также обрабатывает комбинации основного уровня и одного или более уровней расширения. Если данные из всех уровней расширения *numLayer* доступны базовому декодеру, значения *layNumLine [numLayer]* и *layPrevNumLine [numLayer]* должны использоваться вместо *numLine* и *prevNumLine*, соответственно. Значения *ILAmplIndex [i]*, *ILFreqIndex [i], lineContFlag [i]* и *linePred [i]*, как определено описанием синтаксиса потока битов, должны использоваться.

Амплитуды и частоты отдельных линий теперь деквантованы из индексов:

```
for (i = 0; i <numLine; i ++) {
    ampl [i] = 32768 * pow (10,-1,5 * (ILAmplIndex+0,5)/20);
    if (ILFreqIndex <160)
    freq [i] = (ILFreqIndex+0,5) * 3,125;
    else
    freq [] = 500 * exp (0,00625 * (ILFreqIndex+0,5-160));
}
```

Дополнительная информация о фазе запуска декодируется следующим образом:

```
for (i = 0; i <numLine; i ++) {
    if (linePhaseAvail [i]) {
        startPhase [i] = 2*pi * (linePhase [i] +0,5) / (1 <<phasebits) - pi;
        startPhaseAvail [i] = 1;
    }
    else
    startPhaseAvail [i] = 0;
}
```

Если процесс декодирования стартует с произвольного фрейма потока битов, все отдельные линии, которые отмечаются в потоке битов как продолжающиеся из предыдущих фреймов и не декодировались, должны быть обеззвучены.

5.1.2.3.2 Декодер улучшения

Декодер улучшения уточняет параметры линии, полученные из основного декодера, а также декодирует фазы линии. Дополнительная информация содержится в элементе потока битов *INDIenhaPara ()* и оценивается следующим образом:

Все операции базового декодера должны быть выполнены, чтобы можно было корректно декодировать параметры для продолжающихся линий.

Если *envFlag* == 1, тогда улучшенные параметры *t_maxEnh, r_atkEnh* и *r_decEnh* декодируются с использованием данных огибающей, переданных в *HILNbasicFrame ()* и *HILNenhaFrame ()*:

```
t_maxEnh = (envTmax + (envTmaxEnh+0,5) / (1 <<tmEnhbits)) / (1 <<tmbits);
if (envRatk == 0)
r_atkEnh = 0;
else
r_atkEnh = tan (pi/2 * (envRatk-1 + (envRatkEnh+0,5) / (1 <<atkEnhbits)) /
((1 <<atkbits)-1))/0,2;
if (envRdec == 0)
r_decEnh = 0;
else
r_decEnh = tan (pi/2 * (envRdec-1 + (envRdecEnh+0,5) / (1 <<decEnhbits)) /
 ((1 <<decbits)-1))/0,2.
```

Для каждой линии i текущего фрейма

i = 0…… *numLine-1*

улучшенные параметры линии получаются путем уточнения параметров из базового декодера данными в *INDlenhaPara* ():

```
amplEnh [i] = ampl [i];
if (fEnhbits [i]! = 0) {
    if (ILFreqIndex <160)
    freqEnh [i] = (ILFreqIndex+0,5 + ((lineFreqEnh [i] +0,5) / (1 <<fEnhbits [i])- 0,5)) * 3,125;
    else
    freqEnh [i] = 500 * exp (0,00625 * (ILFreqIndex+0,5 -160 +
    ((lineFreqEnh [i] +0,5) / (1 <<fEnhbits [i])- 0,5)));
}
else
freqEnh [i] = частота [i].
```

Для обоих типов линии фаза декодируется из потока битов улучшения:

*phaseEnh [i] = 2*pi * (linePhase [i] +0,5) / (1 <<phasebits) – pi.*

5.1.3 Декодер шума

5.1.3.1 Описание инструмента

Этот инструмент декодирует параметры шума, переданные в потоке битов.

5.1.3.2 Определения

prevNumNoisePara	Число параметров *LPC* шума в предыдущем фрейме.
noiseLPCPara [i]	Параметр *LPC* шума *i* в текущем фрейме (*LARs* для спектра шума).
prevNoiseLPCPara [i]	Параметр *LPC* шума *i* в предыдущем фрейме (*LARs* для спектра шума).
noiseAmpl	Амплитуда шума.
noisePwr	Мощность шума.
noiseT_max	Параметр огибающей шума: время максимума.
noiseR_atk	Параметр огибающей шума: темп нарастания.
noiseR_dec	Параметр огибающей шума: темп спада.

5.1.3.3 Процесс декодирования

5.1.3.3.1 Базовый декодер

Если *noiseFlag* установлено и, таким образом, данные *NOISEbasicPara ()* доступны в текущем фрейме, параметры компонента сигнала шума декодируются и деквантуются следующим образом.

Шум представляется его энергией и рядом *LPC*-параметров.

Сначала восстанавливаются LAR параметры *noiseNumPara*. Прогноз из предыдущего фрейма используется, когда установлен *noiseContFlag*.

```
float noiseLPCMean [25] = {2,0 ,-0,75 , 0,0 , 0,0 , 0,0.. ., 0,0};
for  (i = 0; i <numNoisePara; i ++) {
    if (i <prevNumNoisePara && noiseContFlag)
    pred = noiseLPCMean [i]+ (prevNoiseLPCPara [i]-noiseLPCMean [i]) *0,75;
    else
    pred = noiseLPCMean [i];
    noiseLPCPara [i] = pred + noiseLAR [i];
}
```

Параметры, необходимые в следующем фрейме, сохраняются в межфреймовой памяти:

```
prevNumNoisePara = numNoisePara;
for (i = 0; i <numNoisePara; i ++) {
    prevNoiseLPCPara [i] = noiseLPCPara [i];
```

Амплитуда и мощность шума деквантуются следующим образом:

noiseAmpl = 32768 * *pow (*10,-1,5 * *(noiseAmplIndex*+0,5*)/*20*);*
*noisePwr = noiseAmpl*noiseAmpl.*

Если *noiseEnvFlag* == 1, тогда шумовые параметры огибающей *noiseEnvTmax, noiseEnvRatk* и *noiseEnvRdec* деквантуются в *noiseT_max, noiseR_atk* и *noiseR_dec* таким же образом, как описано для декодера отдельной линии.

5.1.3.3.2 Декодер улучшения

Поскольку для шумовых компонентов нет никаких данных улучшения, для шумовых параметров нет специального режима улучшенного декодирования. Если шум должен быть синтезирован с данными улучшения для других компонентов, может использоваться базовый декодер параметров шума. Если декодер *HILN* используется в качестве ядра в масштабируемом кодере, никакой сигнал шума не должен синтезироваться для сигнала, который подается на декодер улучшения.

5.1.4 Синтезатор гармонической и отдельной линии

5.1.4.1 Описание инструмента

Этот инструмент синтезирует аудиосигнал согласно параметрам гармонической и отдельной линии, декодируемым соответствующими инструментами декодера. Это включает комбинацию гармонических и отдельных линий, базовый синтезатор и синтезатора улучшения. Чтобы получить полный декодированный аудиосигнал, выходной сигнал этого инструмента добавляется к выходному сигналу синтезатора шума.

5.1.4.2 Определения

totalNumLine	Общее количество линий в текущем фрейме, который будет синтезирован (отдельные плюс гармонические).
sampleRate	Частота дискретизации, Гц, как обозначено. *HILNsampleRateCode* (см. таблицу 7).
synthSampleRate	Частота дискретизации синтезируемого выходного сигнала *x[n],* Гц.
speedFactor	Коэффициент изменения скорости синтеза (> 1 для большей, чем исходная скорость воспроизведения).
pitchFactor	Коэффициент изменения шага синтеза (> 1 для большего, чем исходный шаг воспроизведения).
T	Длина фрейма синтеза в секундах.
N	Длина фрейма синтеза в выборках.
env(t)	Амплитудная функция огибающей в текущем фрейме.
a (t)	Мгновенная амплитуда синтезируемой линии.
p (t)	Мгновенная фаза синтезируемой линии.
x (t)	Синтезируемый выходной сигнал.
x [n]	Дискретный синтезируемый выходной сигнал.
startPhi [i]	Стартовая фаза *i*-й линии в текущем фрейме (в радианах).
phi [i]	Фаза окончания *i*-й линии в текущем фрейме (в радианах).
previousEnvFlag	Флаг огибающей в предыдущем фрейме.
previousT_max	Параметр огибающей *t_max* в предыдущем фрейме.
previousR_atk	Параметр огибающей *r_atk* в предыдущем фрейме.
previousR_dec	Параметр огибающей *r_dec* в предыдущем фрейме.
previousEnv (t)	Функция огибающей амплитуды в предыдущем фрейме.
previousTotalNumLine	Общее количество линий в предыдущем фрейме.
previousAmpl [k]	Амплитуда *k*-й линии в предыдущем фрейме.
previousFreq [k]	Частота *k*-й линии в предыдущем фрейме, Гц.
previousPhi [k]	Фаза окончания *k*-й линии в предыдущем фрейме (в радианах).
previousLineEnvFlag [k]	Флаг, указывающий, что предыдущая огибающая амплитуды применяется к *k*-й линии в предыдущем фрейме.

previousT_maxEnh	Улучшенный параметр огибающей *t_max* в предыдущем фрейме.
previousR_atkEnh	Улучшенный параметр огибающей *r_atk* в предыдущем фрейме.
previousR_decEnh	Улучшенный параметр огибающей *r_dec* в предыдущем фрейме.
previousAmplEnh [k]	Улучшенная амплитуда *k*-й линии в предыдущем фрейме.
previousFreqEnh [k]	Улучшенная частота *k*-й линии в предыдущем фрейме, Гц.
previousPhaseEnh [k]	Фаза *k*-й линии в предыдущем фрейме (в радианах).

5.1.4.3 Процесс синтеза

5.1.4.3.1 Комбинация гармонических и отдельных линий

Для синтеза гармонических линий используется тот же самый метод синтеза, что и для отдельных линий.

Если никакая гармоническая составляющая не декодируется для следующих шагов, *numHarmLine* должен быть обнулен.

Иначе параметры гармонических линий добавляются к списку параметров отдельных линий как декодируемые декодером отдельных линий:

```
for (i=0; i <numHarmLine; i ++) {
    freq [numLine+i] = hLineFreq [i];
    ampl [numLine+i] = hLineAmpl [i];
    if (harmContFlag && prevNumLine+i <previousTotalNumLine) {
        lineContFlag [numLine+i] = 1;
        linePred [numLine+i] = prevNumLine+i;
    }
    else
    lineContFlag [numLine+i] = 0;
    lineEnvFlag [numLine+i] = harmEnvFlag;
    startPhase [numLine+i] = hStartPhase [i];
    startPhaseAvail [numLine+i] = hStartPhaseAvail [i];
}
```

Таким образом, общее количество параметров линий, поступивших на синтезатор гармонических и отдельных линий равно:

totalNumLine = numLine + numHarmLine.

В зависимости от значения *HILNcontMode* возможно соединить линии в смежных фреймах, чтобы избежать разрывов фазы в случае переходов с гармонических линий (*HILNcontMode* == 0) или дополнительно с отдельных линий на отдельные линии, для которых бит продолжения *lineContFlag* в потоке битов не был установлен кодером (*HILNcontMode* == 1). Это дополнительное продолжение линии также может быть полностью отключено (*HILNcontMode* == 2).

Для каждой линии *i* = 0.. *totalNumLine*-1 текущего фрейма, у которой нет никакого предшественника (то есть *lineContFlag [i]* == 0), линия оптимальной подгонки *j* предыдущего фрейма, не имеющая преемника и с комбинацией, удовлетворяющей требованиям, определенным *HILNcontMode* определяется, максимизируя следующую меру *q*:

```
df = freq [i] / previousFreq [j];
df = max (df, 1/df);
da = ampl [i] / previousAmpl [j];
da = max (da, 1/da);
q = (1 - (df-1) / (dfCont-1)) * (1 - (da-1) / (daCont-1)),
```

где *dfCont* = 1,05 и *daCont* = 4 являются максимальными разрешенными относительными изменениями частоты и амплитуды. Если имеется больше одного кандидата в предшественники, достигающего максимума *q*, то выбирается кандидат с самым маленьким индексом *j*. Для дополнительных продолжений линии, определенных таким образом, информация о предшественнике строки обновляется:

lineContFlag [i] = 1;
linePred [i] = j.

Если нет по крайней мере одного возможного предшественника с *df <dfCont* и *da <daCont, lineContFlag [i]* и *linePred [i]* остаются неизменными.

Для улучшенного синтезатора параметры улучшенной гармонической (максимум до 10) и отдельной линии объединяются следующим образом:

```
for (i = 0; i <min (10, numHarmLine); i ++) {
    freqEnh [numLine+i] = hLineFreqEnh [i];
    amplEnh [numLine+i] = hLineAmplEnh [i];
    if (harmContFlag && prevNumLine+i <previousTotalNumLine) {
        lineContFlag [numLine+i] = 1;
        linePred [numLine+i] = prevNumLine+i;
    }
    else
    lineContFlag [numLine+i] = 0;
    lineEnvFlag [numLine+i] = harmEnvFlag;
    phaseEnh [numLine+i] = hLinePhaseEnh [i];
}
```

Таким образом, общее количество параметров линии, переданных на синтезатор улучшенной гармонической и отдельной линии, если декодер *HILN* используется в качестве ядра в масштабируемом кодере, равно:

*totalNumLine = numLine + min (*10, *numHarmLine).*

Так как информация о фазе доступна для всех этих линий, никакое продолжение линии не вводится для улучшенного синтезатора.

5.1.4.3.2 Изменение скорости и шага

Благодаря используемому кодером *HILN* параметрическому представлению сигнала и продолжению фазы, обеспечиваемому базовым синтезатором линии, скорость воспроизведения и шаг легко могут быть изменены во время синтеза сигнала в декодере. Если требуется воспроизведение на исходной скорости и шаге, соответствующие факторы управления устанавливаются в их значения по умолчанию:

speedFactor = 1,0;
pitchFactor = 1,0;

Если скоростью управляет масштабный коэффициент времени в поле *speed* узла *AudioSource BSF*, фактор изменения скорости будет:

speedFactor = 1 / speed;

Если шагом управляет поле *pitch* узла *AudioSource BSF*, фактор изменения шага будет:

pitchFactor = pitch;

Когда вместо базового синтезатора используется синтезатор улучшения, *speedFactor* и *pitchFactor* должны всегда устанавливаться в их значение по умолчанию 1,0.

Изменение скорости реализуется изменением длины фрейма синтеза согласно требующемуся *speedFactor*.

Изменение шага реализуется изменением параметров частоты гармонических и отдельных линий следующим образом:

```
for (i = 0; i <totalNumLine; i ++) {
freq [i] * = pitchFactor;
}
```

Синтезатор шума также поддерживает изменение скорости и шага.

5.1.4.3.3 Структурирование синтеза

Создание фреймов при синтезе

Синтезатор гармонический и отдельной линии восстанавливает один фрейм аудиосигнала. Так как параметры линии, закодированные во фрейме потока битов, действительны для центра соответствую-

щего фрейма аудиосигнала, синтезатор генерирует секцию аудиосигнала *x (t)* длиной в один фрейм, которая стартует в центре предыдущего фрейма (*t* = 0), и заканчивается в центре текущего фрейма (*t* = *T*).

По умолчанию синтезатор *HILN* работает при частоте дискретизации *synthSampleRate,* как указано *samplingFrequency*, передающимся в *AudioSpecificInfo ()*:

synthSampleRate = samplingFrequency.

Фрейм синтеза содержит *N* выборок:

*N = (int) (HILNframeLength * synthSampleRate / sampleRate / speedFactor* + 0,5*).*

Таким образом продолжительность *T* фрейма синтеза равна:

T = N / synthSampleRate.

В дальнейшем описывается вычисление синтезируемого выходного сигнала *x (t)*

для 0 <= *t* < *T*. Вариант интервала времени (то есть фактический фрейм выходных выборок) определяется как

x [n] = x (t) c t = (n+0,5*) * (T/N)*

для 0 <= n <N.

Синтезатор шума использует тот же синтез создания фреймов, как синтезатор гармонической и отдельной линии.

5.1.4.3.4 Базовый синтезатор

Некоторые параметры предыдущего фрейма (имена, начинающиеся со слова «предыдущий»), извлекаются из межфреймовой памяти, которая должна быть сброшена перед декодированием первого фрейма потока битов.

Сначала вычисляются функции огибающей *previousEnv (t)* и *env(t)* предыдущего и текущего фреймов согласно следующим правилам:

Если *envFlag* == 1, тогда функция огибающей *env(t)* получается из параметров огибающей *t_max, r_atk* и *r_dec*. При *T,* являющемся длиной фрейма, *env(t)* вычисляется для -*T*/2 <= *t* <3/2**T*:

```
if (-1/2 <= t/T && t/T <t_max)
env (t) = max (0,1-(t_max-t/T) *r_atk);
if (t_max <= t/T && t/T <3/2)
env (t) = max (0,1-(t/T-t_max) *r_dec).
```

Если e*nvFlag* == 0, то используется постоянная функции огибающей *env(t)*:

env (t) = 1.

Соответственно, *previousEnv(t)* вычисляется исходя из параметров *previousT_max, previousR_atk, previousR_dec* и *previousEnvFlag*.

Параметры огибающей, переданные в случае *envFlag* == 1, справедливы для гармонических линий, а также для отдельных линий. Таким образом, функции огибающей должны генерироваться всегда, даже если все *lineEnvFlag [i]* == 0.

Прежде, чем выполняется синтез, очищается аккумулятор *x (t)* для синтезируемого аудиосигнала для 0 <= *t* <*T:*

x (t) = 0*;*

Линии I, продолжающиеся из предыдущего фрейма в текущем фрейме

all i = 0*.. totalNumLine*-1*, that have lineContFlag [i]* == 1

синтезируются для 0 <= *t* <*T* следующим образом:

```
k = linePred [i];
ap (t) = previousAmpl [k];
if (previousLineEnvFlag [k] == 1)
ap (t) * = previousEnv (t+T/2);
ac (t) = ampl [k];
if (lineEnvFlag [i] == 1)
ac (t) * = env (t-T/2);
short_x_fade = (previousLineEnvFlag [k] &&! (previousR_dec <5 &&
(previousT_max <0,5 || previousR_atk <5))) ||
(lineEnvFlag [i] &&! (r_atk <5 && (t_max> 0,5 || r_dec <5)));
if (short_x_fade == 1) {
```

```
    if (0 <= t && t <7/16*T)
    a(t) = ap (t);
    if (7/16*T <= t && t <9/16*T)
    a(t) = ap (t) + (ac (t)-ap (t)) * (t/T-7/16) *8;
    if (9/16*T <= t&& t <T)
    a(t) = ac (t);
}
else
a(t) = ap (t) + (ac (t)-ap (t)) *t/T;
p (t) = previousPhi [k] +2*pi*previousFreq [k] *t +
2*pi * (частота [i]-previousFreq [k]) / (2*T) *t*t;
x (t) + = (t) *sin (p (t));
phi [i] = p (T).
```

Линии *i*, стартующие в текущем фрейме
все i = 0.. *totalNumLine-1, у которых lineContFlag [i]* == 0
синтезируются для 0 <= *t* < *T* следующим образом:

```
if (lineEnvFlag [i]&&! (r_atk <5 && (t_max> 0,5 || r_dec <5))) {
    if (0 <= t&& t <7/16*T)
    fade_in (t) = 0;
    if (7/16*T <= t&& t <9/16*T)
    fade_in (t) = 0,5 - 0,5*cos ((8*t/T-7/2) *pi);
    if (9/16*T <= t&& t <T)
    fade_in (t) = 1;
}
else
fade_in (t) = 0,5-0,5*cos (t/T*pi);
a(t) = fade_in (t) *ampl [i];
if (lineEnvFlag [i] == 1)
a(t) * = env (t-T/2);
if (startPhaseAvail [i])
startPhi [i] = startPhase [i];
else
startPhi [i] = random (2*pi);
p (t) = startPhi [i] + 2*pi*freq [i] * (t-T);
x (t) + = (t) *sin (p (t));
phi [i] = p (T)
```

random(x) является функцией, возвращающей случайное число с универсальным распределением в интервале
0 <= *random (x)* <*x*
Линиии k, оканчивающиеся в предыдущем фрейме
all k = 0.. previousTotalNumLine-1, that have prevLineContFlag [k] == 0
синтезируются для 0 <= *t* < *T* следующим образом:

```
if (previousLineEnvFlag [k] &&! (previousR_dec <5 &&
(previousT_max <0,5 || previousR_atk <5))) {
    if (0 <= t&& t <7/16*T)
    fade_out (t) = 1;
    if (7/16*T <= t&& t <9/16*T)
    fade_out (t) = 0,5 + 0,5*cos ((8*t/T-7/2) *pi);
    if (9/16*T <= t&& t <T)
    fade_out (t) = 0;
}
else
```

fade_out (t) = 0,5+0,5*cos *(t/T*pi);*
*a(t) = fade_out (t) *previousAmpl [k];*
if (previousLineEnvFlag [k] == 1)
*a(t) * = previousEnv (t+T/2);*
*p (t) = previousPhi [k] +2*pi*previousFreq [k] *t;*
*x (t) + = (t) *sin (p (t)).*

Чтобы избежать искажений из-за наложения спектров, синтезируемые линии заглушаются (то есть *a(t)* = 0), пока их мгновенная частота выше или равна половине частоты дискретизации, то есть
*d phi (t) / dt> = pi*N/T.*
Параметры, необходимые в следующем фрейме, сохраняются в межфреймовой памяти:

```
previousEnvFlag = envFlag;
previousT_max = t_max;
previousR_atk = r_atk;
previousR_dec = r_dec;
previousTotalNumLine = totalNumLine;
for (i=0; i <totalNumLine; i ++) {
    previousFreq [i] = freq [i];
    previousAmpl [i] = ampl [i];
    previousPhi [i] = fmod (p [i], 2*pi);
    previousLineEnvFlag [i] = lineEnvFlag [i];
}
```

fmod (x, 2**pi)* является функцией, возвращающей модуль 2**pi* для *x*.
5.1.4.3.5 Синтезатор улучшения
Синтезатор улучшения основан на базовом синтезаторе, но оценивает также фазы линии при восстановлении одного фрейма аудиосигнала. Так как параметры линии, закодированные во фрейме потока битов и соответствующем фрейме улучшения, имеют силу в середине соответствующего фрейма аудиосигнала, синтезатор гармонической и отдельной линии генерирует секцию аудиосигнала длиной в один фрейм, который стартует в середине предыдущего фрейма и заканчивается в середине текущего фрейма.
Некоторые параметры предыдущего фрейма (имена, начинающиеся со слова «*previous*»), извлекаются из межфреймовой памяти, которая должна быть сброшена прежде, чем будет декодироваться первый фрейм потока битов.
Сначала вычисляются функции огибающей *previousEnv (t)* и *env(t)* предыдущего и текущего фреймов согласно следующим правилам.
Если *envFlag* == 1, тогда функция огибающей *env(t)* получается из параметров огибающей *t_maxEnh, r_atkEnh* и *r_decEnh.* При *T,* являющемся длиной фрейма, *env(t)* вычисляется для-*T*/2 <= *t* <3/2**T*:

if (-1/2 <= t/T && t/T <t_maxEnh)
*env (t) = max (0,1-(t_maxEnh-t/T) *r_atkEnh).*
if (t_maxEnh <= t/T && t/T <3/2)
*env (t) = max (0,1-(t/T-t_maxEnh) *r_decEnh);*

Если *envFlag* == 0, тогда используется постоянная функция огибающей *env(t)*:
env (t) = 1.
Соответственно *previousEnv (t)* вычисляется, исходя из параметров *previousT_maxEnh, previousR_atkEnh, previousR_decEnh* и *previousEnvFlag.*
Параметры огибающей, переданные в случае *envFlag* == 1, справедливы для гармонических линий, а также для отдельных линий. Таким образом, функции огибающей должны быть сгенерированы всегда, даже если все *lineEnvFlag [i]* == 0.
Прежде, чем выполняется синтез, аккумулятор *x (t)* для синтезируемого аудиосигнала очищается для 0 <= *t* <*T*:
x (t) = 0.

Все линии *i* в текущем фрейме
all i =0.. *totalNumLine*-1
синтезируются следующим образом для 0 <= *t* <*T*:

```
if (envFlag &&! (r_decEnh <5 && (t_maxEnh <0,5 || r_atkEnh <5))) {
    if (0 <= t&& t <7/16*T)
    fade_in (t) = 0;
    if (7/16*T <= t&& t <9/16*T)
    fade_in (t) = 0,5 - 0,5*cos ((8*t/T-7/2) *pi);
    if (9/16*T <= t&& t <T)
    fade_in (t) = 1;
}
else
fade_in (t) = 0,5-0,5*cos (t/T*pi);
a(t) = fade_in (t) *amplEnh [i];
if (envFlag [i] == 1)
a(t) * = env (t-T/2);
phi (t) = 2*pi*freqEnh [i] * (t-T) +phaseEnh [i];
x (t) + = (t) *sin (phi (t)).
```

Линии *k* в предыдущем фрейме
all k=0.. *previousTotalNumLine*-1
синтезируются следующим образом для 0 <= *t* <*T*:

```
if (previousEnvFlag &&! (previousR_atkEnh <5 &&
(previousT_maxEnh> 0,5 || previousR_decEnh <5))) {
    if (0 <= t&& t <7/16*T)
    fade_out (t) = 1;
    if (7/16*T <= t&& t <9/16*T)
    fade_out (t) = 0,5 + 0,5*cos ((8*t/T-7/2) *pi);
    if (9/16*T <= t&& t <T)
    fade_out (t) = 0;
}
else
fade_out (t) = 0,5+0,5*cos (t/T*pi);
a(t) = fade_out (t) *previousAmplEnh [k];
if (previousEnvFlag [k] == 1)
a(t) * = previousEnv (t+T/2);
phi (t) = 2*pi*previousFreqEnh [k] *t+previousPhaseEnh [i];
x (t) + = (t) *sin (phi (t)).
```

Если мгновенная частота линии выше или равная половине частоты дискретизации, то есть
d phi (t) / dt> = *pi*N/T*,
она не синтезируется, чтобы избежать искажения из-за наложения спектров.
Параметры, необходимые в следующем фрейме, сохраняются в межфреймовой памяти:

```
previousEnvFlag = envFlag;
previousT_maxEnh = t_maxEnh;
previousR_atkEnh = r_atkEnh;
previousR_decEnh = r_decEnh;
previousTotalNumLine = totalNumLine;
для (i=0; i <totalNumLine; i ++) {
    previousFreqEnh [i]= freqEnh [i];
    previousAmplEnh [i]= amplEnh [i];
    previousPhaseEnh [i]= phaseEnh [i];
}
```

5.1.5 Синтезатор шума

5.1.5.1 Описание инструмента

Этот инструмент синтезирует шумовую часть выходного сигнала, основанного на параметрах шума, декодируемых декодером шума. Шумовой сигнал добавляется к выходному сигналу синтезатора гармонической и отдельной линии, чтобы получить полный декодированный аудиосигнал

5.1.5.2 Определения

noiseWin [n]	Окно для наложения-добавления шума.
noiseEnv [n]	Огибающая для шумового компонента в текущем фрейме.
M	Длина фрейма в выборках перед передискретизацией.
w [m]	Белый шум с мощностью *pw*.
xf [m.]	Фильтруемый сигнал шума в текущем фрейме.
xn [n]	Синтезируемый сигнал шума в текущем фрейме.
previousXn [n]	Синтезируемый сигнал шума в предыдущем фрейме.
previousNoiseWin [n]	Окно и огибающая для шумового компонента в предыдущем фрейме.
r [i]	Коэффициенты отражения *LPC*.
h [i]	Импульсная характеристика *LPC*.
hlp [i]	Импульсная характеристика фильтра передискретизации низких частот

5.1.5.3 Процесс синтеза

5.1.5.3.1 Базовый синтезатор

Если для текущего фрейма передаются шумовые параметры, сигнал шума со спектральной формой, которая описана шумовыми параметрами, декодируемыми из потока битов, синтезируется и добавляется к аудиосигналу, сгенерированному синтезатором гармонической и отдельной линии.

Шум представляется его энергией и рядом *LPC*-параметров. Шумовые параметры *LPC* преобразовываются в коэффициенты отражения *r[i]* и во временную характеристику *h [i]*:

for (i = 0; i <numNoisePara; i ++)
r [i] = (exp (noiseLPCPara [i]) - 1) / (exp (noiseLPCPara [i]) + 1).

После этого коэффициенты отражения *r [i]* преобразовываются во временную характеристику *h [i],* используя функцию *C*

*void Convert_k_to_h (float *x, int N),*

вызов с *x [i] = r [i]* и *N = numNoisePara* возвращает с *x [i] = h [i]).*

Фильтруемый сигнал шума *xf [m]* генерируется, применяя фильтр *IIR* синтеза *LPC* к белому шуму, представленному случайными числами *w [m].* Энергия этого белого шума с нулевым средним обозначается *pw*. Для шума с равномерным распределением в [-1,1] энергия будет

pw = 1/3.

Чтобы достигнуть необходимой энергии шумового сигнала, требуется следующий масштабный коэффициент *s*:

ss = 1,0;
for (i = 0*; i <numNoisePara; i ++)*
*ss * = 1-r [i] *r [i];*
*s = noiseAmpl * sqrt (ss/pw).*

Затем белый шум w [m] IIR-фильтруется, чтобы получить синтезируемый сигнал шума *xf* [*m*]

```
for (m = startup; m <2*M; m ++) {
    xf [m] = s * w [m];
    for (i = 0; i <min (m-startup, numNoisePara); i ++)
    xf [m] + = h [i] *xf [m-i-1];
}
```

Чтобы гарантировать, что фильтр *IIR* может достигнуть достаточно устойчивого состояния, используется фаза запуска:

startup = - numNoisePara

Если производится декодирование с изменением шага (то *есть pitchPactor* ! = 1,0) или с другой частотой дискретизации, чем у кодера (то есть *synthSampleRate ! = sampleRate*), к сигналу *xf [m]* должна быть применена операция передискретизации с использованием коэффициента передискретизации

*resampleFactor = (sampleRate * pitchFactor) / synthSampleRate;*

где, например, *pitchFactor*2 указывает, что этот сигнал синтезируется при его удвоенном исходном шаге. Иначе *resampleFactor* устанавливается в 1,0. На базе *resampleFactor* перед передискретизацией определяется длина фрейма *M:*

*M = N * resampleFactor.*

Передискретизация может быть реализована, применяя две операции фильтра нижних частот *FIR* к сигналу *xf [m]* и линейно интерполируя между этими двумя значениями, чтобы получить заключительный сигнал шума *xn [n].*

```
if (resampleFactor <1)
fc = 1;
else
fc = 1/resampleFactor.
```

Следующая функция вычисляет временную характеристику *hlp*[0.. 31] соответствующего фильтра нижних частот *FIR* с 16 отводами и коэффициентом передискретизации 4. Частота среза будет *fc*.

```
void GenLPFilter (float, *hlp, double  fc),
{
    double  x, f;
    int i;
    hlp [0] = (float) fc;
    for (i = 1; i <32; i ++)
    {
        x = i*pi/4,0;
        hlp [i] = (float) ((0,54+0,46*cos (0,125*x)) *sin (fc*x)/x);
    }
}
```

Чтобы выполнить работу фильтра *FIR*, может использоваться следующая функция *C*. Параметрами являются сигнал, временная характеристика (как результат функции выше) и позиция точки выборки. Позиция дается как разница между ближайшей позицией выборки до требующейся позиции выборки (*x* [7]) и требующейся позицией выборки. Поэтому 0 <= *pos* <1. Интерполяция выполняется между *x*[7] и *x*[8], полученное значение представляет позицию выборки 7+*pos*.

```
float LPInterpolate (float *x, float *hlp, double fc),
{
    long j;
    double s, t;
    pos * = 4,0;
    j = (long) pos;
    pos - = (double) j;
    s = t = 0,0;
    j = 32-j;
    if (j == 32)
    {
        t = h [31] * (*x);
        x ++;
         j - = 4;
```

```
    }
    while (j> 0)
    {
        s + = h [j] * (*x);
        t + = h [j-1] * (*x);
        x ++;
        j - = 4;
    }
        j =-j;
        while (j <32-1)
    {
        s + = h [j] * (*x);
        t + = h [j+1] * (*x);
        x ++;
         j + = 4;
    }
    if (j <32)
    s + = h [j] * (*x);
    return (float) (s+pos * (t-s));
}
```

Используя функции *GenLPFilter ()* и *LPInterpolate (),* производится передискретизация, как описано ниже. *xf [m]* устанавливается в 0,0 для *m <startup* и m> = 2**M*.

```
GenLPFilter (hlp, fc);
for (n = 0; n <2*N; n ++)
xn [n] = LPInterpolate (xf + (int) n*resampleFactor)-7, hlp, frac (n*resampleFactor)).
```

Если *resampleFactor* == 1,0, *xf [m]* просто копируется в *xn [n]* без передискретизации:

```
for (n = 0; n <2*N; n ++)
xn [n] = xf [n].
```

Для гладкого затухания-перекрытия сигнала шума на границе между двумя смежными фреймами для этой операции перекрытия-добавления используется следующее окно:

```
for (n = 0; n <N; n ++) {
    if (n <N*3/8)
     noiseWin [n] = 0;
    if (N*3/8 <= n && n <N*5/8)
    noiseWin [n] = epex (пи/2 * (n-N*3/8+0,5) / (N*2/8));
    if (N*5/8 <= n)
    noiseWin [n] = 1;
    noiseWin [2*N-1-n] = noiseWin [n];
}
```

Теперь вычисляется функция огибающей *noiseEnv [n]*. Если *noiseEnvFlag* == 1, тогда функция огибающей

*noiseEnv [n] = noiseEnv (t) c t = (n+0,5) * (T/N)-0,5*

получается из параметров огибающей *noiseT_max, noiseR_atk u noiseR_dec* для-*T*/2 <= *t* <3/2**T* (то есть 0 <= *n* <2**N*):

```
если (-1/2 <= t/T && t/T <noiseT_max)
noiseEnv (t) = max (0,1-(noiseT_max-t/T) *noiseR_atk);
если (noiseT_max <= t/T && t/T <3/2)
noiseEnv (t) = max (0,1-(t/T-noiseT_max) *noiseR_dec).
```

Если *noiseEnvFlag* == 0, то используется постоянная функция огибающей *noiseEnv (t):*

noiseEnv [n] = 1*;*

Сигнал шума *xn [n]* является оконным для перекрытия-добавления и умножается на огибающую *noiseEnv [n].* Затем этот сигнал и шум из предыдущего фрейма *previousXn [n]* добавляются к сигналу *x [n]* из синтезатора гармонической и отдельной линии, чтобы создать полный синтезируемый *signalx [n]:*

for (n = 0; n <N; n ++)
*x [n] + = xn [n] *noiseWin [n] *noiseEnv [n] + previousXn [n].*

Вторая половина сгенерированного *signalxn [n]* шума сохраняется в межфреймовой памяти *previousXn [n]* для перекрытия-добавления:

for (n = 0; n <N; n ++)
*previousXn [n] = xn [N+n] *noiseWin [N+n] *noiseEnv [N+n].*

Память *previousXn [n]* должна быть сброшена в 0,0 прежде, чем будет декодироваться первый фрейм.

5.1.5.3.2 Синтезатор улучшения

Когда нет никаких данных улучшения для шумовых компонентов, нет и никакого определенного режима синтезатора улучшения для шумовых компонентов. Если должен быть синтезирован шум и имеются данные улучшения для других компонентов, может использоваться базовый декодер синтезатора шума. Если декодер *HILN* используется в масштабируемом кодере в качестве ядра, никакой шумовой сигнал не должен синтезироваться для сигнала, который подается декодеру улучшения.

5.2 Интегрированный параметрический кодер

Интегрированный параметрический кодер может работать в четырех различных режимах. *PARAmodes* 0 и 1 представляют режимы фиксированных *HVXC* и *HILN*. *PARAmode* 2 разрешает автоматическое переключение между *HVXC* и *HILN* в зависимости от текущего типа входного сигнала. В *PARAmode* 3 кодеры *HVXC* и *HILN* могут использоваться одновременно, и их выходные сигналы добавляются (смешиваются) в декодере.

Интегрированный параметрический кодер использует длину фрейма 40 мс и частоту дискретизации 8 кГц и может работать со скоростью передачи 2025 бит/с или любой более высокой.

5.2.1 Интегрированный параметрический декодер

Для режимов "*HVXC only*" и "*HILN only* " параметрический декодер не изменяется.

В режимах "*switched HVXC/HILN*" и "*mixed HVXC/HILN*" управление инструментами декодера *HVXC* и *HILN* происходит альтернативно или одновременно согласно *PARAswitchMode* или *PARAmixMode* текущего фрейма. Чтобы получить надлежащее выравнивание по времени выходных сигналов декодера *HVXC* и *HILN* прежде, чем они будут добавлены, различие между задержкой декодера *HVXC* и *HILN* нужно компенсировать с помощью буфера *FIFO*:

если *HVXC* используется в режиме декодера с низкой задержкой, его выход должен быть задержан на 100 выборок (то есть 12,5 мс);

если *HVXC* используется в режиме декодера с нормальной задержкой, его выход должен быть задержан на 80 выборок (то есть 10 мс).

Чтобы избежать трудных переходов на границах фрейма, когда включаются или выключаются декодеры *HVXC* или *HILN,* соответствующие выходные сигналы декодера нарастают и спадают гладко. Для декодера *HVXC* применяется линейное нарастание или спад 20 мс, когда он включается или выключается. Декодер *HILN* не требует дополнительного нарастания и исчезновения по причине гладких окон синтеза, используемых в синтезаторе *HILN*.

6 Устойчивые к ошибкам полезные нагрузки потока битов

6.1 Обзор инструментов

Устойчивые к ошибкам полезные нагрузки потока битов позволяют эффективно использовать усовершенствованные методы кодирования канала вроде неравномерной защиты от ошибок (*UEP*), которые могут быть отлично адаптированы к потребностям различных инструментов кодирования. Основная идея состоит в том, чтобы перестроить стандартную полезную нагрузку потока битов в зависимости

от ее чувствительности к ошибкам в одном или более случаях, принадлежащих различным категориям чувствительности к ошибкам (*ESC*). Эта перестановка работает с данными или поэлементно, или даже порязрядно. Устойчивая к ошибкам полезная нагрузка потока битов создается, связывая эти случаи.

Поток битов переупорядочивается согласно чувствительности к ошибкам единичных элементов потока битов или даже единичных битов. Этот поток битов с новым расположением канально кодируется, передается и канально декодируется. Перед декодированием поток битов перестраивается к своему первоначальному порядку. Вместо того, чтобы выполнить переупорядочение описанным способом, в этом перестроении определяется переупорядоченный синтаксис, который является порядком потока битов, до форматизатора потока битов в месте расположения декодера.

6.2 *ER HILN*

Категории чувствительности к ошибкам (*ESC*) определяются в параметрическом потоке битов. Ниже описывается упорядочивание различных *ESC* для четырех различных режимов *PARAmode* == 0, 1, 2, 3.

PARAmode == 0 (только *HVXC*)
*HVXC: ESC*0 *ESC*1 *ESC*2 *ESC*3 *ESC*4

PARAmode == 1 (только *HILN*)
*PARA/HILN: ESC*0 *ESC*1 *ESC*2 *ESC*3 *ESC*4

PARAmode == 2 (переключение *HVXC / HILN),*
*PARA/HILN: ESC*0 *[ESC*1 *ESC*2 *ESC*3 *ESC*4*]*
HVXC 1/двойной*: [ESC*0 *ESC*1 *ESC*2 *ESC*3 *ESC*4*]*
HVXC 2/двойной*: [ESC0 ESC1 ESC2 ESC3 ESC4]*

PARAmode == 3 (смешение *HVXC / HILN),*
*PARA/HILN: ESC*0 *[ESC*1 *ESC*2 *ESC*3 *ESC*4*]*
HVXC 1/двойной*: [ESC*0 *ESC*1 *ESC*2 *ESC*3 *ESC*4*]*
HVXC 2/двойной*: [ESC0 ESC1 ESC2 ESC3 ESC4]*

*ESC*0 для "*PARA/HILN*" состоит из элемента потока битов *PARAswitchMode* или *PARAmixMode* в *PARAframe()*, сопровождаемого элементами потока битов в *HILNbasicFrameESC0()*. Фактическое присутствие этих элементов потока битов может зависеть от текущих значений *PARAmode, PARAswitchMode* и *PARAmixMode*. "*HVXC* 1/двойной" и "*HVXC* 2/двойной" обозначают первый и второй *ErHVXCfixfram ()* в пределах *ErHVXCdoubleframe()*. Присутствие *ESC* в квадратных скобках зависит от значения *PARAswitchMode* или *PARAmixMode* в текущем фрейме.

Приложение А
(справочное)

Параметрический аудиокодер

А.1 Краткий обзор инструментов кодера

В параметрическом кодере входной сигнал разделяется на две части, которые кодируются *HVXC* и инструментами *HILN*. Это может быть сделано вручную или автоматически. Автоматическое переключение между речью и музыкальными сигналами поддерживается *HVXC* для речи и *HILN* для музыки. Общее средство форматирования потока битов позволяет работу только в *HVXC* или только в *HILN*, или также в объединенных режимах, то есть переключенном или смешанном режиме.

Следующее описание параметрического кодера *HILN* информативно, и также альтернативные методы для сигнального разделения и оценки параметра могут использоваться в кодере.

А.2 Кодер *HILN* инструменты

Основной принцип *HILN*: кодер должен проанализировать входной сигнал, чтобы извлечь параметры, описывающие сигнал. Эти параметры кодируются и передаются как поток битов. В декодере выходной сигнал синтезируется, основанный на параметрах, извлеченных и переданных кодером.

Кодер состоит из двух основных частей: “экстракция параметра” и “кодирование параметра”. В кодере входной сигнал делится на последовательные кадры, и для каждого фрейма ряд параметров, описывающих сигнал в этом фрейме, извлекается и кодируется. Из-за этого параметрического описания возможен широкий диапазон скоростей передачи, частот дискретизации и длин фрейма. Обычно используется длина фрейма 32 мс. Для входных сигналов с частотой дискретизации на 8—16 кГц обычно используется скорость передачи 6—16 Кбит/с.

А.2.1 Экстракция параметра *HILN*

На экстракции параметра входной сигнал разделяется на три различные части: “гармонические строки”, “отдельные строки” и «шум».

Из этих параметров частей, описывающих сигнал, извлекаются:

гармонические строки: основная частота и амплитуды гармонических составляющих;

отдельные строки: частота и амплитуда каждой отдельной строки;

шум: спектральная форма шума.

Дополнительно параметры для амплитудных конвертов и для продолжения линий спектра от одного фрейма до следующего могут быть определены.

Сигнальная оценка разделения и параметра реализуется в трех шагах: сначала оценивается основная частота гармонической части сигнала, затем оцениваются параметры соответствующих линий спектра, и эти строки классифицируются как “отдельные строки” или “гармонические строки” в зависимости от частоты относительно основной частоты. После того, как все соответствующие линии спектра извлекаются, остающийся остаточный сигнал подобен шуму, и его спектральная форма описывается рядом параметров.

Гармоническая экстракция строки инструментов *HILN* может быть использована в интегрированном параметрическом кодере, использующем инструменты кодирования речи *HVXC* и кодирование инструментов *HILN* одновременно.

А.2.1.1 Оценка основной частоты

Инструментами *HILN* используется метод оценки основной частоты “*Cepstrum*”. Сначала входной сигнал *Hanning* центрируется вокруг текущего фрейма. Для оконного сигнала вычисляется спектр:

$$w(f) = (1+\cos(2*pi*f/fs))/2 \qquad 0 <= f <= fs/2$$

Определяются локальные максимумы в *cepstrum*, и идентифицируется самый большой максимум в пределах разрешенного “диапазона поиска” задержки подачи. Вычисляется основная частота от “задержки подачи” (период основной частоты) самого большого максимума.

Основная частота, определенная на основе метода *cepstrum,* используется в качестве начальной (грубой) оценки для следующей оценки параметра строки.

А.2.1.2 Гармоническая и отдельная оценка параметра строки

Оценка гармонических и отдельных параметров строки основана на “Цикле Анализа/Синтеза”.

В первом шаге оцениваются параметры всех гармонических строк. Вычисляется оценка основной частоты *hFreq* и «протяжения» *hStretch,* который минимизирует полную ошибку между реальными гармоническими строчными частотами и вычисленными согласно

$$hLinefreq[i] = hFreq * (i+1) * (1 + hStretch * (i+1)) \qquad i = 0..\ harmNumLine-1,$$

где общее количество гармонических строк определяется пропускной способностью *w* сигнала и текущей основной частоты *hFreq*:

harmNumLine = floor (w/hFreq)

Гармонический флаг конвертера устанавливается, используя конвертер амплитуды тока для всех гармонических результатов строк по более низкой остаточной ошибке.

Во втором шаге извлекаются соответствующие линии спектра из входного сигнала посредством цикла анализа/синтеза. Этот цикл использует психоакустическую модель, чтобы извлечь линии спектра в порядке их субъективной уместности. Если частота извлеченной линии спектра близка к частоте гармонической строки, она классифицируется как гармоническая строка. Иначе она классифицируется как отдельная строка. Цикл анализа/синтеза завершается, если требуемое число отдельных строк было извлечено или если остающиеся сигнальные компоненты не могут быть должным образом смоделированы линиями спектра. Отношение между числом гармонических извлеченных строк и полными извлеченными строками передается кодеру как мера «уместности» гармонических строк.

А.2.1.2.1 Предварительный анализ

Преданалитический модуль определяет конвертер амплитуды сигнала, который используется в цикле анализа/синтеза.

А.2.1.2.2 Анализ/синтез, основанный на единственных линиях спектра

Отдельный кодер строки основан на модели единственных линий спектра, которые могут быть сгенерированы генератором синусоидальных колебаний. Для *i*-й строки цикл состоит из следующих шагов:

вычисление отклонения между спектрами *FFT* ввода и синтезируемых сигналов;

выбор соответствующей строки *FFT* с центральной частотой *fi*;

оценка частоты высокого разрешения окружения *fi;*

выбор информации о конверторе амплитуды и фазовая оценка;

синтез с решительными параметрами;

вычисление остаточной ошибки синтезируемого сигнала от входного сигнала.

Строка *FFT* определяется, вычисляя отклонение между входным спектром и синтезируемым спектром и находя максимальное отношение квадрата этого отклонения и порога маскирования, полученного из сигнала, синтезируемого от линий спектра.

Чтобы получить параметр частоты более высокой точности, чем разрешение *FFT*, используется центральная частота *fi* выбранной строки *FFT*.

А.2.1.2.3 Психоакустическая модель

Психоакустическая модель вычисляет порог для синтезируемых сигнальных компонентов в цикле анализа/синтеза.

А.2.1.3 Шумовая оценка параметра

Шумовые параметры используются, чтобы смоделировать спектральную форму остаточного сигнала. Сначала вычисляется энергетический спектр *Hanning* оконного остаточного сигнала. Затем этот спектр преобразовывается в автокорреляционную функцию. Вычисляются параметры *LPC*, используя алгоритм Дербина. Параметры *LPC* преобразовываются в коэффициенты отражения.

А.2.2 Кодер параметра *HILN*

Чтобы генерировать вывод потока битов кодера *HILN,* извлеченные параметры гармоники отдельной строки и шумовых частей сигнала квантуются и кодируются.

А.2.2.1 Гармоническое квантование параметра строки

Число битов, доступных для гармонических параметров строки, зависит от величины гармонического сигнального компонента. Если она низка, то число гармонических закодированных строк может быть меньше, чем число извлеченных строк. Это соответствует ограничению пропускной способности гармонического сигнала.

Основная частота квантуется с 2048 шагами по логарифмической шкале в пределах от 20 Гц к 4 кГц.

Для описания спектра гармонического тона вычисляется автокорреляционная функция гармонического сигнала. Из нее получают коэффициенты *LPC*. Этот процесс подобен *LPC specral* моделированию, используемому для шумового сигнала.

А.2.2.2 Отдельное квантование параметра строки

В модуле квантования и кодирования параметры обрабатываются в порядке поступления из цикла анализа/синтеза. Этот модуль в состоянии генерировать два потока битов, один основной поток битов, который позволяет генерацию основного качественного аудиосигнала, и поток битов улучшения, который может использоваться в приложениях. Основной поток битов содержит частоту и амплитудные параметры, в то время как поток битов улучшения содержит фазовые параметры и информацию для квантования параметров конверта и частоты.

Для каждого фрейма входного сигнала передаются биты согласно требуемой скорости передачи. В каждом фрейме передается бит, который указывает, используются ли параметры конверта или нет.

Так как человеческая слуховая система не чувствительна к изменениям фазы, то кодируются и передаются в основном потоке битов только частота и информация об амплитуде линий спектра. В этом случае необходимо предоставить информацию для декодера, который позволяет генерировать сигнал, свободный от фазовых разрывов на границах фрейма. Первый этап обработки обнаруживает строки, которые продолжаются от одного фрейма

до другого. Если строка должна продолжаться от предыдущего фрейма, квантуются только частота и амплитудные изменения и передаются вместо абсолютной частоты и амплитудных значений. Продолжение строки используется, если относительное изменение частоты

$$g_f(i,k) = \frac{|f_i(m) - f_k(m-1)|}{f_i(m)}$$

не превышает порог $q_{f,\,max}$ и если отношение амплитуд

$$g_a(i,k) = \begin{cases} a_i(m)/a_k(m-1), \text{ если } a_i(m) \geq a_k(m-1) \\ a_k(m-1)/a_i(m), \text{ если } a_i(m) \leq a_k(m-1) \end{cases}$$

находится в пределах интервала [1... $q_{f,\,max}$]. Если есть более одной возможности продолжать строку от предыдущего фрейма, выбирается та строка в предыдущем фрейме, для которого следующий критерий достигает своего максимума:

$$Q = \frac{q_{f,max} - q_f(i,k)}{q_{f,max}} \cdot \frac{q_{a,max} - q_a(i,k)}{(q_{a,max} - 1)q_a(i,k)}.$$

Частоты и амплитуды отдельных строк квантуются согласно шкале частот и логарифмической шкале амплитуд. Для каждой строки предыдущего фрейма бит продолжения передается в потоке битов, который указывает, продолжается ли строка в текущем фрейме или нет. Для новых строк индексы для квантованной частоты и амплитуды кодируются, используя *SubDivisionCode (SDC)*. Для всех строк, продолжаемых от предыдущего фрейма, индексные различия частоты и амплитуды, кодируются с кодом энтропии.

Так как основной поток битов не содержит фазовую информацию, нет необходимости вычислять остаточный ошибочный сигнал, вычитая соответствующий выходной сигнал декодера входного сигнала. Чтобы включить режимы масштабируемости, в которых остаточный сигнал передается в потоке битов более высокой скорости передачи, генерируют дополнительный поток битов улучшения. Это создается следующим образом.

Если параметры конверта передаются в основном потоке битов, для лучшего квантования передаются дополнительные биты 3 параметров конверта.

Если строка запускается, то есть не продолжаемая от предыдущего фрейма, и его частота превышает данный порог, для лучшего квантования передаются дополнительные биты абсолютной частоты.

Для каждой строки фазовый параметр передается после универсального квантования.

Число битов на фрейм в потоке битов улучшения может измениться, это должно быть принято во внимание в вычислении доступных битов для кодирования остаточной ошибки.

Так как позиция продолжительной строки в текущем фрейме зависит от позиции его предшественника в предыдущем фрейме, используется алгоритм выделения, который гарантирует, что строки *N*, переданные в текущем фрейме, всегда являются *N* большинством соответствующих строк, найденных циклом анализа/синтеза.

Системная задержка кодера равна 1,5 длины фрейма. Эта задержка следует из длины фрейма непосредственно и дополнительной задержки (0,5) времени длины фрейма, вызванная смещенным накладывающимся окном, используемым для оценки частоты.

SDC-кодирование:

k: число кодовых комбинаций (0... *k*-1)

i: значение для кодирования

tab: таблица, содержащая доменные пределы

```
void  SDCEncode (int k, i, int *tab)
{
int *pp;
int g, dp, min, max, cwl;
long cw;
cw=cwl=0;
min=0
max=k-1;
pp=tab+16;
dp=16;
while (min ! = max)
{
if (dp)g=(k*(*pp))>>10;  else g = (max+min)>>1;
dp>> = 1;
cw<< = 1;
cwl++;
if (i<=g){pp -=dp; max=g;}  else {cw |=1; pp+=dp; min=g+1;}
```

```
}
 PutBits (cw, cwl);
}
```

PutBits () пишет кодовую комбинацию в поток битов, где *LSB* по часовой стрелке являются *"vlclbf"* кодовой комбинацией, и *cwl* определяет число битов, которые будут переданы.

А.2.2.3 Шумовое квантование параметра

Число шумовых параметров, которые квантуются и кодируются, зависит от размера компонента шумового сигнала. Если число шумовых параметров мало, никакие шумовые параметры не передаются. Для более высоких значений числа шумовых параметров соответствующие числа параметров *LAR* квантуются и кодируются. Из-за свойств коэффициентов отражения число переданных параметров *LAR* может быть решено во время разрядного выделения в кодере, и никакой перерасчет этих параметров не требуется.

Если устанавливается *noiseEnvFlag,* тогда дополнительный набор шумовых параметров конверта квантуется и кодируется.

А.2.3 Масштабируемость скорости передачи *HILN*

Параметрические сигнальные представления, используемые параметрическим кодером *HILN*, хорошо подходят для приложений, требующих скорости передачи масштабируемого кодирования. В таком приложении скорость передачи, полученная декодером, может быть динамически адаптирована к свойствам ссылки передачи или выбрана согласно некоторым другим правилам. В случае потока битов льготного тарифа передаются только параметры перцепционно соответствующих сигнальных компонентов (отдельные строки, гармонический тон, шум). В случае потока битов полного тарифа также передаются параметры дополнительных сигнальных компонентов (например, отдельные строки).

Эта масштабируемость скорости передачи для потоков битов *HILN* может быть реализована, используя основу и потоки битов уровня расширения или динамически управляемый параметр, кодирующий, как описано ниже.

А.2.3.1 Масштабируемость скорости передачи *HILN* динамически управляемым кодированием параметра

Чтобы реализовать масштабируемость скорости передачи посредством динамически управляемого кодирования параметра, используется то, что экстракцией параметра *HILN* и кодером параметра *HILN* можно управлять независимо. Параметры, сгенерированные инструментом экстракции параметра, могут обращаться к инструментам кодера параметра, каждый из которых генерирует поток битов с различной скоростью передачи. Также можно сохранить неквантованные параметры, сгенерированные инструментом экстракции параметра в файле. Затем инструмент кодера параметра может использоваться, чтобы генерировать поток битов с требуемой в настоящий момент скоростью передачи от параметров, сохраненных в этом файле.

А.3 Музыка/речь — смешанный инструмент кодера

Аудиопараметрический кодер используется для того, чтобы кодировать естественные аудиосигналы в очень низких скоростях передачи в пределах от 2 Кбит/с до 16 Кбит/с. Параметрический кодер обеспечивает два набора инструментов для того, чтобы кодировать речевые и неречевые аудиосигналы соответственно:

гармоническое векторное возбуждение (*HVXC*) инструментов подходит для того, чтобы кодировать речевые сигналы от 2 Кбит/с до 4 Кбит/с;

гармонические и отдельные строки плюс шум (*HILN*) инструментов подходят для того, чтобы кодировать неречевые аудиосигналы в скоростях передачи от 4 Кбит/с и выше.

В режиме только *HVXC* или только *HILN*, режим кодирования выбирается вручную во время кодирования, и выбранный режим используется для всего закодированного аудиосигнала.

Интегрированный параметрический кодер автоматически выбирает инструменты кодирования, которые подходят лучше всего для фактических характеристик входного сигнала. В случае речевого сигнала используются инструменты *HVXC*, а для музыки используются инструменты *HILN*. Этот выбор делается на основании решения об автоматическом инструменте классификации речи/музыки. Для сигналов, которые являются смесью речи и музыки, также возможно использовать инструменты *HVXC* и *HILN* одновременно.

А.3.1 Инструмент классификации музыки/речи

Это инструмент для параметрического речевого кодера, который включается автоматической идентификацией музыки/речи для параметрического кодера речи/аудио (*HVXC* и *HILN*). Инструмент принимает решения, использует внутренние параметры *HVXC*.

Этот инструмент классификации музыки/речи может быть применен двумя способами:

первые 5 секунд сигнала, который будет закодирован, анализируются инструментом классификации, и затем выбираются *HVXC* или *HILN*, чтобы кодировать сообщение согласно решению речи/музыки;

инструментом классификации управляют непрерывно, и его текущее решение речи/музыки используется, чтобы выбрать *HVXC* или *HILN* для текущего фрейма. В этом приложении должна быть принята во внимание задержка решения с 5 секундами.

А.3.1.1 Энергия фрейма

Энергия фрейма *P* вычисляется как

$$P = \sum_{n=0}^{159} s(n)^2,$$

где s (n) является входным сигналом.

В этом случае фреймы с энергетическими уровнями выше, чем предопределенный минимальный уровень, используются (исключая>-78 дБ). Краткосрочная средняя энергия фрейма определяется как

$$Pav = \sum_{t=0}^{3} P\{t\}/4,$$

который вычисляется из последних четырех энергий фрейма.

Различие между энергией фрейма и краткосрочной средней энергией фрейма вычисляется как:

$$Pd[frm] = |P - Pav| / Pav,$$

Pd[*frm*] сохраняется приблизительно для 250 фреймов (5 секунд).

А.3.1.2 Сила подачи

В *HVXC* максимальная автокорреляция остатка *LPC (r0r)* вычисляется во время процесса обнаружения подачи. *r0r* сохраняются приблизительно для 250 фреймов.

А.3.1.3 Решение музыки/речи

Среднее значение и различие энергий фрейма и *r0r* вычисляются соответственно как:

$$Pd(an) = \sum_{frm=0}^{249} Pd[frm]/250$$

$$Pd(na) = \sqrt{\sum_{frm=0}^{249} (Pd[frm]) - Pd(an)^2} \Big/ 250$$

$$r0r(an) = \sum_{frm=0}^{249} r0r[frm] \Big/ 250$$

$$r0r(na) = \sqrt{\sum_{frm=0}^{249} (r0r[frm]) - r0r(an)^2} \Big/ 250.$$

В том же самом диапазоне среднего значения *r0r* у речевых данных есть более высокие различия, чем музыкальные данные. Матрица классифицируется в трех областях.

(1) *speech* *r0r (va)* ≥ 0,153 *r0r (av)* + 0,113

(2) *unknown* 0,07 *r0r (av)* = 0,137 < *r0r (va)* < 0,153 *r0r (av)* + 0,113

(3) *music* 0,07 *r0r (av)* + 0,137 ≥ *r0r (va)*

Если среднее значение и различие включаются в область (1), данные классифицируются как речь. Если они находятся в области (3), данные классифицируются как музыка.

Если среднее значение и различие существуют в области (2), среднее значение и различие (дифференциального) энергетического *Pd* фрейма используются дополнительно. У речевых данных есть более выразительные средства и различия *Pd,* чем музыкальные данные. Речевые и музыкальные данные разделяются на следующие две области.

(1) speech *Pd (va)* ≥ - 0,5 *Pd r0r (av)* + 0,8

(3) music *Pd (va)* < - 0,5 *Pd (av)* + 0,8

Используя выше упомянутые два критерия, разделяются речь и музыка.

А.3.2 Интегрированный параметрический кодер

Интегрированный параметрический кодер может работать в следующих режимах:

PARAmode	Описание
0	Только *HVXC*
1	Только *HILN*
2	Переключенный *HVXC/HILN*
3	Смешанный *HVXC/HILN*

PARAmodes 0 и 1 представляют фиксированный *HVXC* и режимы *HILN. PARAmode* 2 разрешения автоматическое переключение между *HVXC и HILN* в зависимости от текущего типа входного сигнала. В *PARAmode* 3 *HVXC* и кодеры *HILN* могут использоваться одновременно, и их выходные сигналы добавляются (смешанные) в декодере.

Интегрированный параметрический кодер использует длину фрейма 40 мс и частоту дискретизации 8 кГц и может работать с 2025 бит/с или любой более высокой скоростью передачи.

А.3.2.1 Интегрированный параметрический кодер

Для режима "только *HVXC*" и "только *HILN*" параметрический кодер не изменяется. "Коммутируемые *HVXC / HILN*" и "смешанный *HVXC/HILN*" режимы описываются ниже.

А.3.2.2 Коммутируемый режим *HVXC/HILN*

Поскольку инструмент классификации речи/музыки основан на кодере *HVXC*, кодером *HVXC* управляют непрерывно для каждого фрейма. Фрейм потока битов, сгенерированный кодером *HVXC* и входным аудиосигналом, сохранен в двух буферах *FIFO*, чтобы компенсировать задержку с 5 секундами решения речи/музыки. Если фрейм классифицируется как «речь», тогда *PARAswitchMode* устанавливается в 0 и фрейм потока битов *HVXC*, доступный в *FIFO* потока битов, передается. В случае «музыкального» решения *PARAswitchMode* устанавливается в 1, и вывод сигнального буфера *FIFO* кодируется кодером *HILN*, и этот кадр потока битов *HILN* передается. Если *HVXC* используется для фрейма, кодер *HILN* сбрасывается (*prevNumLine* = 0).

А.3.2.3 Смешанный режим *HVXC/HILN*

Чтобы управлять параметрическим кодеком в "смешанном *HVXC/HILN*" режиме, речь и музыкальные компоненты входного сигнала должны быть разделены. Если оба компонента уже доступны отдельно (например, речь и фоновая музыка), кодирование является прямым.

Библиография

[1] ИСО/МЭК 14496-3:2009 Информационные технологии. Кодирование аудиовизуальных объектов. Часть 3. Аудио (ISO/IEC 14496-3:2009 *Information technology — Coding of audio-visual objects — Part* 3: *Audio*)

УДК 621.396 : 006.354 ОКС 33.170

Ключевые слова: звуковое вещание, электрические параметры, каналы и тракты, технологии MPEG-кодирования, синтетический звук, масштабирование, защита от ошибок, поток битов расширения, психоакустическая модель

Редактор *Н.А. Аргунова*
Технический редактор *В.Н. Прусакова*
Корректор *М.В. Бучная*
Компьютерная верстка *Е.О. Асташина*

Сдано в набор 22.08.2014. Подписано в печать 31.10.2014. Формат 60×84⅛. Гарнитура Ариал.
Усл. печ. л. 6,51. Уч.-изд. л. 5,21. Тираж 39 экз. Зак. 4418.

Издано и отпечатано во ФГУП «СТАНДАРТИНФОРМ», 123995 Москва, Гранатный пер., 4.
www.gostinfo.ru gost.e-ecolog.ru

gost.e-ecolog.ru