ФЕДЕРАЛЬНОЕ АГЕНТСТВО
ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 53556.3— 2012

Звуковое вещание цифровое

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

Часть 3

(MPEG-4 AUDIO)

Кодирование речевых сигналов с использованием линейного предсказания — CELP

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 20 ноября 2012 г. № 942-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») [1]

5 ВВЕДЕН ВПЕРВЫЕ

Правила применения настоящего стандарта установлены в ГОСТ Р 1.0—2012 (раздел 8). Информация об изменениях к настоящему стандарту публикуется в ежегодном (по состоянию на 1 января текущего года) информационном указателе «Национальные стандарты», а официальный текст изменений и поправок — в ежемесячном информационном указателе «Национальные стандарты». В случае пересмотра (замены) или отмены настоящего стандарта соответствующее уведомление будет опубликовано в ежемесячно издаваемом информационном указателе «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 применения . . . 1
1.1 Общее описание декодера *CELP* . . . 1
1.2 Функциональные возможности *MPEG-4 CELP* . . . 1
2 Термины и определения . . . 4
3 Синтаксис потока бит . . . 5
3.1 Тип объекта *CELP* . . . 5
3.2 Тип объекта *ER-CELP* . . . 9
4 Семантики . . . 20
4.1 Семантики заголовка . . . 20
4.2 Семантика фрейма . . . 23
5 Инструменты декодера *MPEG-4 CELP* . . . 28
5.1 Введение в набор инструментов декодера *CELP MPEG-4* . . . 28
5.2 Конфигурация масштабируемого *AAC/CELP* . . . 29
5.3 Переменные помощи . . . 29
5.4 Элементы потока бит для набора инструментов *MPEG-4 CELP* . . . 30
5.5 Демультиплексор потока бит *CELP* . . . 30
5.6 Декодер *CELP LPC* и интерполятор . . . 30
5.7 Генератор возбуждения *CELP* . . . 45
5.8 Фильтр синтеза *CELP LPC* . . . 62
5.9 Инструмент сжатия тишины *CELP* . . . 63
Приложение А (справочное) Инструменты декодера *MPEG*-4 *CELP* . . . 69
Приложение В (справочное) Инструменты кодера *MPEG*-4 *CELP* . . . 71
Библиография . . . 96

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Звуковое вещание цифровое

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

Часть 3

(MPEG-4 AUDIO)

Кодирование речевых сигналов с использованием линейного предсказания — CELP

Sound broadcasting digital. Coding of signals of sound broadcasting with reduction of redundancy for transfer on digital communication channels. Part 3 (MPEG-4 audio). Code excited linear prediction

Дата введения — 2013—09—01

1 Область применения

1.1 Общее описание декодера *CELP*

Здесь дается краткий обзор декодера *CELP (Code Excited Linear Prediction* (Линейное предсказание с кодированием)).

Декодер *CELP* прежде всего состоит из генератора возбуждения и фильтра синтеза. Дополнительно декодеры *CELP* часто включают в свой состав постфильтр. У генератора возбуждения имеется адаптивная книга шифров для моделирования периодических компонент, фиксированные книги шифров для моделирования случайных компонент и декодер усиления, чтобы представлять уровень речевого сигнала. Индексы для книг шифров и коэффициентов усиления предоставляются кодером. Индексы книги шифров (индекс задержки шага для адаптивной книги шифров и индекс формы для фиксированной книги шифров) и индексы усиления (коэффициенты усиления адаптивной и фиксированной книг шифров) используются, чтобы генерировать сигнал возбуждения. Затем он фильтруется фильтром линейного прогнозирующего синтеза (фильтр синтеза *LP*). Коэффициенты фильтра реконструируются, используя индексы *LPC*, затем интерполируются коэффициентами фильтра последовательных фреймов анализа. И наконец, опционно может быть применен постфильтр, чтобы улучшить качество речи.

1.2 Функциональные возможности *MPEG-4 CELP*

MPEG-4 CELP представляет собой универсальный алгоритм кодирования с новыми функциональными возможностями. Обычные кодеры *CELP* предлагают сжатие при одной битовой скорости и оптимизированы для определенных приложений. Сжатие является одной из функций, предоставленных *CELP MPEG-4*, позволяющей использовать один базовый кодер для различных приложений. Это обеспечивает масштабируемость по битовой скорости и полосе пропускания, а также возможность генерировать потоки бит с произвольной битовой скоростью. Кодер *CELP MPEG-4* поддерживает две частоты дискретизации, а именно 8 и 16 кГц. Соответствующие полосы пропускания равны 100—3400 Гц для частоты дискретизации 8 кГц и 50—7000 Гц для частоты дискретизации 16 кГц. Кроме того, заново приняты сжатие молчание и переупорядочение эластичного потока бит ошибок.

1.2.1 Конфигурация кодера *CELP MPEG-4*

Чтобы генерировать сигнал возбуждения, могут использоваться два различных инструмента. Это инструмент *Multi-Pulse Excitation* (Мультиимпульсное возбуждение) (*MPE*) или инструмент *Regular-Pulse Excitation* (Возбуждение регулярным импульсом) (*RPE*). *MPE* используется для дискретизации речи на частотах 8 кГц или 16 кГц. *RPE* используется только для дискретизации на частоте 16 кГц. Два возможных режима кодирования сведены в таблице 1.

Издание официальное

Таблица 1 — Режимы кодирования в кодере *CELP MPEG-4*

Режим кодирования	Инструмент возбуждения	Частота дискретизации
I	*RPE*	16 кГц
II	*MPE*	8,16 кГц

1.2.2 Особенности кодера *CELP MPEG-4*

Кодер *CELP MPEG-4* предлагает следующие функциональные возможности в зависимости от режима кодирования (см. таблицу 2).

Таблица 2 — Функциональные возможности кодера *CELP MPEG-4*

Режим кодирования	Функциональность
I	Несколько битовых скоростей, управление FineRate
II	Несколько битовых скоростей, масштабируемость битовой скорости, масштабируемость полосы пропускания, управление FineRate

Для обоих режимов кодирования доступны сжатие молчания и переупорядочение эластичного потока бит ошибок.

Доступные битовые скорости зависят от режима кодирования и частоты дискретизации. Поддерживаются следующие фиксированные битовые скорости (см. таблицы 3, 4).

Таблица 3 — Фиксированные битовые скорости для режима I кодера

Битовые скорости для частоты дискретизации 16 кГц, бит/с
14400, 16000, 18667, 22533

Таблица 4 — Фиксированные битовые скорости для режима II кодера

Битовые скорости для частоты дискретизации 8 кГц (бит/с)	Битовые скорости для частоты дискретизации 16 кГц (бит/с)
3850, 4250, 4650, 4900, 5200, 5500, 5700, 6000, 6200, 6300, 6600, 6900, 7100, 7300, 7700, 8300, 8700, 9100, 9500, 9900, 10300, 10500, 10700, 11000, 11400, 11800, 12000, 12200	10900, 11500, 12100, 12700, 13300, 13900, 14300, 14700, 15900, 17100, 17900, 18700, 19500, 20300, 21100, 13600, 14200, 14800, 15400, 16000, 16600, 17000, 17400, 18600, 19800, 20600, 21400, 22200, 23000, 23800

Во время неактивных фреймов используется инструмент сжатия тишины и кодер *CELP* работает на битовых скоростях, показанных в таблице 5. Битовая скорость зависит от режима кодирования, частоты дискретизации и длины фрейма.

Таблица 5 — Битовые скорости для инструмента сжатия тишины

Режим кодирования	Частота взятия выборок, кГц	Масштабируемость ширины полосы	Длина фрейма	Битовая скорость, бит/с		
				TX_flag	*HD-SID*	*LR-SID*
I (*RPE*)	16	—	15 10	133 200	2533 3800	400 600
II (*MPE*)	8	*On, Off*	40 30 20 10	50 67 100 200	525 700 1050 2100	150 200 300 600

Окончание таблицы 5

Режим кодирования	Частота взятия выборок, кГц	Масштабируемость ширины полосы	Длина фрейма	Битовая скорость, бит/с		
				TX_flag	*HD-SID*	*LR-SID*
	16	*Off*	20 10	100 200	1900 3800	300 600
		On	40 30 20 10	50 67 100 200	1050 1400 2100 4200	150 200 300 600

Управление скоростью: обеспечивает управление битовой скоростью малыми шажками (давая возможность работать на варьируемой битовой скорости). Это достигается только за счет управления скоростью передачи параметров *LPC*, используя комбинации двух элементов потока бит *interpolation_flag* и флажок *LPC_present*. Для изменения отношения фреймов *LPC* к общему числу фреймов между 50 % и 100 % можно использовать управление *FineRate*. Это позволяет уменьшить битовую скорость относительно битовой скорости привязки, как определено в семантике.

Масштабируемость битовой скорости обеспечивается добавлением уровней расширения. Уровни расширения могут быть добавлены с шагом 2000 бит/с для сигналов, дискретизированных с частотой 8 кГц или 4000 бит/с для сигналов, дискретизированных на 16 кГц. С любой битовой скоростью, выбранной из таблицы 4, можно объединить максимум три уровня расширения.

Масштабируемость полосы пропускания с охватом обеих частот дискретизации достигнута включением инструмента расширения полосы пропускания в кодере *CELP*. Это инструмент расширения, поддерживаемый в Режиме II, который может быть добавлен, если требуется масштабирование с переходом от частоты дискретизации 8 кГц к частоте дискретизации 16 кГц. Полный кодер с масштабируемостью полосы пропускания состоит из основного кодера *CELP* для частоты дискретизации 8 кГц и инструмента расширения полосы пропускания для обеспечения одного уровня масштабируемости. Основной кодер *CELP* для частоты дискретизации 8 кГц может включить несколько уровней. Кодер частоты дискретизации 8 кГц с этим инструментом отличается от кодера частоты дискретизации 16 кГц. Обе конфигурации (кодер частоты дискретизации 8 кГц с масштабируемостью полосы пропускания и кодер частоты дискретизации 16 кГц) предлагают большую ясность и естественность декодированной речи, чем дает один только кодер 8 кГц, потому что они разворачивают полосу пропускания до 7 кГц. Дополнительная битовая скорость, требующаяся для инструмента масштабируемости полосы пропускания, может быть выбрана из четырех дискретных шагов для каждой битовой скорости основного уровня, как показано в таблице 6.

Таблица 6 — Битовые скорости для режима масштабируемой полосы пропускания

Битовая скорость основного уровня, бит/с	Дополнительная битовая скорость, бит/с
3850 — 4650	+9200, +10400, +11600, +12400
4900 — 5500	+9467, +10667, +11867, +12667
5700 — 10700	+10000, +11200, +12400, +13200
11000 — 12200	+11600, +12800, +14000, +14800

Инструмент сжатия тишины может использоваться, чтобы уменьшить битовую скорость для входных сигналов с небольшой голосовой активностью. В течение таких неактивных периодов декодер заменяет регулярный сигнал возбуждения искусственно сгенерированным шумом. Для периодов голосовой активности всегда используется регулярный процесс синтеза речи. Инструмент сжатия тишины доступен, когда используется тип объекта *ER-CELP*.

Переупорядочение эластичного потока бит ошибок позволяет эффективно использовать усовершенствованные техники кодирования канала как неравномерная защита от ошибок (*UEP*). Основная идея состоит в том, чтобы перестроить контент звукового фрейма в зависимости от его чувствительности к ошиб-

кам в одном или более случаях, принадлежащих различным категориям чувствительности к ошибкам (*ESC*). Эта перестановка воздействует на данные поэлементно или даже поразрядно. Фрейм эластичного потока бит ошибок строится, связывая эти случаи. Эти функциональные возможности доступны, когда используется тип объекта *ER-CELP*.

1.2.3 Алгоритмическая задержка режимов *CELP MPEG-4*

Алгоритмическая задержка кодера *CELP* исходит из длины фрейма и длины дополнительного предвидения. Длина фрейма зависит от режима кодирования и битовой скорости. Длина предвидения, которая является информативным параметром, также зависит от режима кодирования. Задержки, представленные ниже, применимы к режимам, где управление *FineRate Control* выключено (см. таблицы 7, 8, 9). Когда управление *FineRate Control* включено, вносится дополнительная задержка на один фрейм. Масштабируемость полосы пропускания в кодере режима II требует дополнительного предвидения на 5 мс из-за субдискретизации.

Таблица 7 — Задержка и длина фрейма для кодера режима I частоты дискретизации 16 кГц

Битовая скорость для режима I, бит/с	Задержка, мс	Длина фрейма, мс
14400	26,25	15
16000	18,75	10
18667	26,56	15
22533	26,75	15

Таблица 8 — Задержка и длина фрейма для кодера режима II частоты дискретизации 8 кГц

Битовая скорость для режима II, бит/с	Задержка, мс	Длина фрейма, мс
3850, 4250, 4650	45	40
4900, 5200, 5500, 6200	35	30
5700, 6000, 6300, 6600, 6900, 7100, 7300, 7700, 8300, 8700, 9100, 9500, 9900, 10300, 10500, 10700	25	20
11000, 11400, 11800, 12000, 12200	15	10

Таблица 9 — Задержка и длина фрейма для кодера режима II частоты дискретизации

Битовая скорость для режима II, бит/с	Задержка, мс	Длина фрейма, мс
10900, 11500, 12100, 12700, 13300, 13900, 14300,14700, 15900, 17100, 17900, 18700, 19500, 20300, 21100	25	20
13600, 14200, 14800, 15400, 16000, 16600, 17000,17400, 18600, 19800, 20600, 21400, 22200, 23000, 23800	15	10

В случае, если сжатие тишины используется, алгоритмическая задержка такая же, как без сжатия тишины, поскольку используются такая же длина фрейма и такая же дистанция дополнительного упреждающего просмотра.

2 Термины и определения

Термины и определения в соответствии с ГОСТ Р 53556.0–2009

3 Синтаксис потока бит

3.1 Тип объекта *CELP*

3.1.1 Синтаксис заголовка

CelpSpecificConfig ()

Для типа объекта *CELP* требуется следующий *CelpSpecificConfig* () (см. таблицы 10, 11, 12)

Таблица 10 — Синтаксис *CelpSpecificConfig* ()

Синтаксис	Количество битов	Мнемосхема
CelpSpecificConfig (uint(4) samplingFrequencyIndex) *{* *isBaseLayer* *if (isBaseLayer)* *{* *CelpHeader (samplingFrequencyIndex);* *}* *else* *{* *isBWSLayer;* *if (isBWSLayer)* *{* *CelpBWSenhHeader ();* *}* *else* *{* *CELP-BRS-id;uimcbf* *}* *}* *}*	 1 2	 *uimsbf* *uimsbf*

Таблица 11 — Синтаксис *CelpHeader* ()

Синтаксис	Количество битов	Мнемосхема
CelpHeader (samplingFrequencyIndex) *{* *ExcitationMode;* *SampleRateMode;* *FineRateControl;* *if (ExcitationMode == RPE) {* *RPE_Configuration;* *}* *if (ExcitationMode == MPE) {* *MPE_Configuration;* *NumEnhLayers;* *BandwidthScalabilityMode;* *}* *}*	 1 1 1 3 5 2 1 	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 12 — Синтаксис *CelpBWSenhHeader* ()

Синтаксис	Количество битов	Мнемосхема
CelpBWSenhHeader () *{* *BWS_configuration;*	 2	 *uimcbf*

3.1.2 Синтаксис фрейма

Передача потоков бит *CELP*

Каждый уровень потока бит аудио *CELP MPEG–4* передается в элементарном потоке. В *slPacketPayload* должны быть включены следующие динамические данные для Аудио CELP:

Базовый уровень *CELP* — полезная нагрузка модуля доступа

```
slPacketPayload
{
    CelpBaseFrame ();
}
```

Уровень расширения *CELP* — полезная нагрузка модуля доступа

Чтобы анализировать и декодировать уровень расширения *CELP*, требуется информация, декодированная из базового уровня *CELP*. Для масштабируемого режима битовой скорости должны быть включены следующие данные для уровня расширения *CELP*:

```
slPacketPayload
{
    CelpBRSenhFrame ();
}
```

Для масштабируемого режима полосы пропускания должны быть включены следующие данные для уровня расширения *CELP*:

```
slPacketPayload
{
    CelpBWSenhFrame ();
}
```

В случае, если масштабирование битовой скорости и масштабирование полосы пропускания используются одновременно, сначала должны быть переданы все уровни расширения битовой скорости, а затем уровень масштабируемости полосы пропускания (см. таблицы 13, 14, 15, 16, 17, 18, 19, 20)

Таблица 13 — Синтаксис *CelpBaseFrame* ()

Синтаксис	Количество битов	Мнемосхема
CelpBaseFrame() *{* *Celp_LPC();* *if (ExcitationMode == MPE) {* *MPE_frame();* *}* *if ((ExcitationMode==RPE) && (SampleRateMode==16кГц)) {* *RPE_frame();* *}* *}*		

Таблица 14 — Синтаксис *CelpBRSenhFrame* ()

Синтаксис	Количество битов	Мнемосхема
CelpBRSenhFrame() *{* *for (subframe = 0; subframe < nrof_subframes; subframe++) {* *shape_enh_positions [subframe][enh_layer];* *shape_enh_signs [subframe][enh_layer];* *gain_enh_index [subframe][enh_layer];* *}* *}*	 4, 12 2, 4 4	 *uimcbf* *uimcbf* *uimcbf*

Таблица 15 — Синтаксис *CelpBWSenhFrame* ()

Синтаксис	Количество битов	Мнемосхема
CelpBWSenhFrame() *{* *BandScalable_LSP()* *for (subframe=0; subframe<nrof_subframe_bws; subframe++)* *{* *shape_bws_delay [subframe];* *shape_bws_positions [subframe];* *shape_bws_signs [subframe];* *gain_bws_index [subframe];* *}* *}*	 3 22, 26, 30, 32 6, 8, 10, 12 11	 *uimcbf* *uimcbf* *uimcbf* *uimcbf*

3.1.2.1 Синтаксис *LPC*

Таблица 16 — Синтаксис *Celp_LPC* ()

Синтаксис	Количество битов	Мнемосхема
Celp_LPC() *{* *if (FineRateControl == ON){* *interpolation_flag;* *LPC_Present;* *if (LPC_Present == YES) {* *LSP_VQ();* *}* *} else {* *LSP_VQ()* *}* *}*	 1 1	 *uimcbf* *uimcbf*

Таблица 17 — Синтаксис *LSP_VQ* ()

Синтаксис	Количество битов	Мнемосхема
LSP_ VQ() *{* *if (SampleRateMode == 8кГц) {* *NarrowBand_ LSP();* *} else {* *WideBand_LSP();* *}* *}*		

Таблица 18 — Синтаксис *NarrowBand_LSP* ()

Синтаксис	Количество битов	Мнемосхема
NarrowBand_LSP() *{* *lpc_indices [0];* *lpc_indices [1];* *lpc_indices [2];* *lpc_indices [3];* *lpc_indices [4];* *}*	 4 4 7 6 1	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 19 — Синтаксис *BandScalable_LSP* ()

Синтаксис	Количество битов	Мнемосхема
BandScalable_LSP()		
{		
lpc_indices [5];	4	*uimcbf*
lpc_indices [6];	7	*uimcbf*
lpc_indices [7];	4	*uimcbf*
lpc_indices [8];	6	*uimcbf*
lpc_indices [9];	7	*uimcbf*
lpc_indices [10];	4	*uimcbf*
}		

Таблица 20 — Синтаксис *WideBand_LSP* ()

Синтаксис	Количество битов	Мнемосхема
WideBand_LSP()		
{		
lpc_indices [0];	5	*uimcbf*
lpc_indices [1];	5	*uimcbf*
lpc_indices [2];	7	*uimcbf*
lpc_indices [3];	7	*uimcbf*
lpc_indices [4];	1	*uimcbf*
lpc_indices [5];	4	*uimcbf*
lpc_indices [6];	4	*uimcbf*
lpc_indices [7];	7	*uimcbf*
lpc_indices [8];	5	*uimcbf*
lpc_indices [9];	1	*uimcbf*
}		

3.1.2.2 Синтаксис возбуждения (см. таблицы 21, 22)

Таблица 21 — Синтаксис *RPE_frame* ()

Синтаксис	Количество битов	Мнемосхема
RPE_frame()		
{		
for (subframe = 0; subframe < nrof_subframes; subframe++)		
{		
shape_delay [subframe];	8	*uimcbf*
shape_index [subframe];	11	*uimcbf*
gainindices [0][subframe];	12	*uimcbf*
gainindices [1][subframe];	6, 3, 5	*uimcbf*
}		

Таблица 22 — Синтаксис *MPE_frame* ()

Синтаксис	Количество битов	Мнемосхема
MPE frame()		
{		
signal_mode;	2	*uimcbf*
rмс_index;	6	*uimcbf*
for (subframe = 0; subframe < nrof_subframes; subframe++)		
{	8, 9	*uimcbf*

Окончание таблицы 22

Синтаксис	Количество битов	Мнемосхема
shape_delay [subframe]; *shapepositions [subframe];* *shapesigns [subframe];* *gainindex [subframe];* *}* *}*	14 ... 32 3 ... 12 6, 7	*uimcbf* *uimcbf* *uimcbf*

3.2 Тип объекта *ER-CELP*

3.2.1 Синтаксис заголовка

ErrorResilientCelpSpecificConfig ()

Для типа объекта *ER-CELP* требуется следующий *ErrorResilientCelpSpecificConfig* () (см. таблицы 23, 24).

Таблица 23 — Синтаксис *ErrorResilientCelpSpecificConfig* ()

Синтаксис	Количество битов	Мнемосхема

ErrorResilientCelpSpecificConfig (uint(4) samplingFrequencyIndex)
{
isBaseLayer;
if (isBaseLayer)
{
ER_SC_CelpHeader (samplingFrequencyIndex);
}
Else
{
isBWSLayer;
if (isBWSLayer)
{
CelpBWSenhHeader ();
}
else
{
CELP-BRS-id;
}
}
}
	
	1
	
	1
	
	2
	
		uimcbf
		
		uimcbf
		
		uimcbf

Таблица 24 — Синтаксис *ER_SC_CelpHeader* ()

Синтаксис	Количество битов	Мнемосхема
ER_SC_CelpHeader (samplingFrequencyIndex) *{* *ExcitationMode;* *SampleRateMode;* *FineRateControl;* *SilenceCompression;* *if (ExcitationMode == RPE) {* *RPE_Configuration;* *}* *if (ExcitationMode == MPE) {* *MPE_Configuration;* *NumEnhLayers;* *BandwidthScalabilityMode;* *}* *}*	 1 1 1 1 3 5 2 1 	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

3.2.2 Синтаксис фрейма

Чтобы описать чувствительность к ошибке в символе элементов потока бит, введены категории чувствительности к ошибкам (*ESC*). Чтобы описать отдельные биты элементов, используется следующая система обозначений.

Усиление, *x — y*

Обозначает усиление элемента от бита x до бита y, в силу чего сначала передается x. Младший бит *LSB* — нулевой бит и *MCB* элемента, который состоит из *N* бит равен *N-1*. *MCB* всегда первый бит в потоке бит.

Следующий синтаксис является заменой для *CelpBaseFrame*. Синтаксис для уровня расширения для масштабируемости битовой скорости и полосы пропускания не затронут.

Передача потоков бит *CELP*

Данные полезной нагрузки для объекта *CELP ER* передаются как полезная нагрузка *slPacketPayload* в базовом уровне и опционном уровне расширения *Elementary Stream* (элементарный поток).

Базовый уровень эластичного *CELP* ошибок — полезная нагрузка модуля доступа

```
slPacketPayload
{
    ER_SC_CelpBaseFrame ();
}
```

Уровень расширения эластичного *CELP* ошибок — полезная нагрузка модуля доступа.

Чтобы анализировать и декодировать уровни расширения эластичного *CELP* ошибок, требуется декодированная информация из базового уровня эластичного *CELP* ошибок. Для режима масштабируемой битовой скорости должны быть включены следующие данные для уровней расширения эластичного *CELP* ошибок:

```
slPacketPayload
{
    ER_SC_CelpBRSenhFrame ();
}
```

Для режима масштабируемой полосы пропускания должны быть включены следующие данные для уровня расширения эластичного *CELP* ошибок:

```
slPacketPayload
{
    ER_SC_CelpBWSenhFrame ();
}
```

3.2.2.1 Базовый уровень *CELP* (см. таблицы 25, 26, 27)

Таблица 25 — Синтаксис *ER_SC_CelpBaseFrame* ()

Синтаксис	Количество битов	Мнемосхема
ER_SC_CelpBaseFrame() *{* *if (SilenceCompression == OFF) {* *ER_CelpBaseFrame();* *} else {* *SC_VoiceActivity_ESC0();* *if (TX_flag == 1) {* *ER_CelpBaseFrame();* *} else if (TX_flag == 2) {* *SID_LSP_VQ_ESC0();* *SID_Frame_ESC0();* *} else if (TX_flag == 3) {* *SID_Frame_ESC0();* *}* *}* *}*		

Таблица 26 — Синтаксис *SC_VoiceActivity_ESC0* ()

Синтаксис	Количество битов	Мнемосхема
SC_VoiceActivity_ESC0 () *{* *TX_flag;* *}*	2	*uimcbf*

Таблица 27 — Синтаксис *ER_CelpBaseFrame* ()

Синтаксис	Количество битов	Мнемосхема
ER_CelpBaseFrame() *{* *if (ExcitationMode == MPE) {* *if (SampleRateMode == 8кГц) {* *MPE_NarrowBand_ESC0();* *MPE_NarrowBand_ESC1();* *MPE_NarrowBand_ESC2();* *MPE_NarrowBand_ESC3();* *MPE_NarrowBand_ESC4();* *}* *if (SampleRateMode == 16кГц) {* *MPE_WideBand_ESC0();* *MPE_WideBand_ESC1();* *MPE_WideBand_ESC2();* *MPE_WideBand_ESC3();* *MPE_WideBand_ESC4();* *}* *}* *if ((ExcitationMode == RPE) && (SampleRateMode == 16кГц)) {* *RPE_WideBand_ESC0();* *RPE_WideBand_ESC1();* *RPE_WideBand_ESC2();* *RPE_WideBand_ESC3();* *RPE_WideBand_ESC4();* *};* *}*		

3.2.2.1.1 Синтаксис узкополосного *MPE* (см. таблицы 28, 29, 30, 31, 32)

Таблица 28 — Синтаксис *MPE_NarrowBand_ESC0* ()

Синтаксис	Количество битов	Мнемосхема
MPE_NarrowBand_ESC0() *{* *if (FineRateControl == ON) {* *interpolation_flag;* *LPC_Present;* *}* *rmc_index, 5-4;* *for (subframe = 0; subframe < nrof_subframes;* *subframe++) {* *shape_delay[subframe], 7;* *}* *}*	 1 1 2 1	 *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 29 — Синтаксис *MPE_NarrowBand_ESC1* ()

Синтаксис	Количество битов	Мнемосхема
MPE_NarrowBand_ESC1 () *{* *if (FineRateControl == ON) {* *if (LPC_Present == YES) {* *lpc_indices [0], 1-0;* *lpc_indices [1], 0;* *}* *} else {* *lpc_indices [0], 1-0;* *lpc_indices [1], 0;* *}* *signal mode;* *for (subframe = 0; subframe < nrof_subframes; subframe++) {* *shape_delay[subframe], 6-5;* *}* *}*	 2 1 2 1 2 2	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 30 — Синтаксис *MPE_NarrowBand_ESC2* ()

Синтаксис	Количество битов	Мнемосхема
MPE_NarrowBand_ESC2() *{* *if (FineRateControl == ON) {* *if (LPC_Present == YES) {* *lpc_indices [2], 6;* *lpc_indices [2], 0;* *lpc_indices [4];* *}* *} else {* *lpc_indices [2], 6;* *lpc_indices [2], 0;* *lpc_indices [4];* *}* *гмс_index, 3* *for (subframe = 0; subframe < nrof_subframes; subframe++)* *{* *shape_delay[subframe], 4-3;* *gain_index[subframe], 1-0;* *}* *}*	 1 1 1 1 1 1 1 2 2	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 31 — Синтаксис *MPE_NarrowBand_ESC3* ()

Синтаксис	Количество битов	Мнемосхема
MPE_NarrowBand_ESC3() *{* *if (FineRateControl == ON) {* *if (LPC_Present == YES) {* *lpc_indices [0], 3-2;* *lpc_indices [1], 2-1;* *lpc_indices [2], 5-1;* *}* *} else {*	 2 2 5	 *uimcbf* *uimcbf* *uimcbf*

Окончание таблицы 31

Синтаксис	Количество битов	Мнемосхема
lpc_indices [0], 3-2;	2	*uimcbf*
lpc_indices [1], 2-1;	2	*uimcbf*
lpc_indices [2], 5-1;	5	*uimcbf*
}		
for (subframe = 0; subframe < nrof_subframes;		
subframe++) {	3	*uimcbf*
shape_delay[subframe], 2-0;	3 ... 12	*uimcbf*
shape signs[subframe];	1	*uimcbf*
gain_index[subframe], 2;		
}		
}		

Таблица 32 — Синтаксис *MPE_NarrowBand_ESC4* ()

Синтаксис	Количество битов	Мнемосхема
MPE_NarrowBand_ESC4()		
{		
if (FineRateControl == ON) {		
if (LPC_Present == YES) {		
lpc_indices [1], 3;	1	*uimcbf*
lpc_indices [3];	6	*uimcbf*
}		
} else {		
lpc_indices [1], 3;	1	*uimcbf*
lpc_indices [3],	6	*uimcbf*
}		
гмс_index, 2-0	3	*uimcbf*
for (subframe = 0; subframe < nrof_subframes; subframe++) {		
shape positions[subframe];	13 ... 32	*uimcbf*
gain_index[subframe], 5-3;	3	*uimcbf*
}		
}		

3.2.2.1.2 Широкополосный синтаксис *MPE* (см. таблицы 33, 34, 35, 36, 37)

Таблица 33 — Синтаксис *MPE_WideBand_ESC0* ()

Синтаксис	Количество битов	Мнемосхема
MPE_WideBand_ESC0()		
{		
if (FineRateControl == ON) {		
interpolation flag;	1	*uimcbf*
LPC_Present;	1	*uimcbf*
if (LPC_Present == YES) {		
lpc_indices [0];	5	*uimcbf*
lpc_indices [1], 1-0;	2	*uimcbf*
lpc_indices [2], 6;	1	*uimcbf*
lpc_indices [2], 4-0;	5	*uimcbf*
lpc_indices [4];	1	*uimcbf*
lpc_indices [5], 0;	1	*uimcbf*
}		
} else {		
lpc_indices [0];	5	*uimcbf*

Окончание таблицы 33

Синтаксис	Количество битов	Мнемосхема
lpc_indices [1], 1-0;	2	*uimcbf*
lpc_indices [2], 6;	1	*uimcbf*
lpc_indices [2], 4-0;	5	*uimcbf*
lpc_indices [4];	1	*uimcbf*
lpc_indices [5], 0;	1	*uimcbf*
}		
гмс_index, 4-5;	2	*uimcbf*
}		

Таблица 34 — Синтаксис *MPE_WideBand_ESC*1 ()

Синтаксис	Количество битов	Мнемосхема
MPE_WideBand_ESC1()		
{		
if (FineRateControl == ON) {		
if (LPC_Present == YES) {		
lpc_indices [1], 3-2;	2	*uimcbf*
lpc_indices [2], 5;	1	*uimcbf*
lpc_indices [5], 1;	1	*uimcbf*
lpc_indices [6], 1-0;	2	*uimcbf*
}		
} else {		
lpc_indices [1], 3-2;	2	*uimcbf*
lpc_indices [2], 5;	1	*uimcbf*
lpc_indices [5], 1;	1	*uimcbf*
lpc_indices [6], 1-0;	2	*uimcbf*
}		
signal mode;	2	*uimcbf*
for (subframe = 0; subframe < nrof_subframes; subframe++) {		
shape_delay[subframe], 8-6;	3	*uimcbf*
}		
}		

Таблица 35 — Синтаксис *MPE_WideBand_ESC2* ()

Синтаксис	Количество битов	Мнемосхема
MPE_WideBand_ESC2()		
{		
if (FineRateControl == ON) {		
if (LPC_Present == YES) {		
lpc_indices [1], 4;	1	*uimcbf*
lpc_indices [3], 6;	1	*uimcbf*
lpc_indices [3], 1;	1	*uimcbf*
lpc_indices [5], 2;	1	*uimcbf*
lpc_indices [6], 3;	1	*uimcbf*
lpc_indices [7], 6;	1	*uimcbf*
lpc_indices [7], 4;	1	*uimcbf*
lpc_indices [7], 1-0;	2	*uimcbf*
lpc_indices [9];	1	*uimcbf*
}		
} else {		
lpc_indices [1], 4;	1	*uimcbf*
lpc_indices [3], 6;	1	*uimcbf*

Окончание таблицы 35

Синтаксис	Количество битов	Мнемосхема
lpc_indices [3], 1;	1	*uimcbf*
lpc_indices [5], 2;	1	*uimcbf*
lpc_indices [6], 3;	1	*uimcbf*
lpc_indices [7], 6;	1	*uimcbf*
lpc_indices [7], 4;	1	*uimcbf*
lpc_indices [7], 1-0;	2	*uimcbf*
lpc_indices [9];	1	*uimcbf*
}		
гмс_index, 3;	1	*uimcbf*
for (subframe = 0; subframe < nrof_subframes;		
subframe++) {	2	*uimcbf*
shape_delay[subframe], 5-4;	2	*uimcbf*
gain_index[subframe], 1-0;		
}		
}		

Т а б л и ц а 36 — Синтаксис *MPE_WideBand_ESC*3 ()

Синтаксис	Количество битов	Мнемосхема
MPE_WideBand_ESC3()		
{		
if (FineRateControl == ON) {		
if (LPC_Present == YES) {		
lpc_indices [3], 4-2;	3	*uimcbf*
lpc_indices [3], 0;	1	*uimcbf*
lpc_indices [5], 3;	1	*uimcbf*
lpc_indices [6], 2;	1	*uimcbf*
lpc_indices [7], 5;	1	*uimcbf*
lpc_indices [7], 3-2;	2	*uimcbf*
lpc_indices [8], 4-1;	4	*uimcbf*
}		
} else {		
lpc_indices [3], 4-2;	3	*uimcbf*
lpc_indices [3], 0;	1	*uimcbf*
lpc_indices [5], 3;	1	*uimcbf*
lpc_indices [6], 2;	1	*uimcbf*
lpc_indices [7], 5;	1	*uimcbf*
lpc_indices [7], 3-2;	2	*uimcbf*
lpc_indices [8], 4-1;	4	*uimcbf*
}		
for (subframe = 0; subframe < nrof_subframes; subframe++) {		
shape_delay[subframe], 3-2;	2	*uimcbf*
shape signs[subframe];	3 ... 12	*uimcbf*
gain_index[subframe], 2;	1	*uimcbf*
}		
}		

Таблица 37 — Синтаксис *MPE_WideBand_ESC*4 ()

Синтаксис	Количество битов	Мнемосхема
MPE_WideBand_ESC4()		
{		
if (FineRateControl == ON) {		
if (LPC_Present == YES) {		
lpc_indices [3], 5;	1	*uimcbf*
lpc_indices [8], 0;	1	*uimcbf*
}		
} else {		
lpc_indices [3], 5;	1	*uimcbf*
lpc_indices [8], 0;	1	*uimcbf*
}		
гмс_index, 2-0;	3	*uimcbf*
for (subframe = 0; subframe < nrof_subframes; subframe++) {		
shape_delay[subframe], 1-0;	2	*uimcbf*
shape positions[subframe];	14 ... 32	*uimcbf*
gain_index[subframe], 6-3;	4	*uimcbf*
}		
}		

3.2.2.1.3 Широкополосный синтаксис *RPE* (см. таблицы 38, 39, 40, 41, 42)

Таблица 38 — Синтаксис *RPE_WideBand_ESC*0 ()

Синтаксис	Количество битов	Мнемосхема
RPE_WideBand_ESC0()		
{		
if (FineRateControl == ON){		
interpolation flag;	1	*uimcbf*
LPC_Present;	1	*uimcbf*
if (LPC_Present == YES) {		
Lpc_indices [0];	5	*uimcbf*
lpc_indices [1], 1-0;	2	*uimcbf*
lpc_indices [2], 6;	1	*uimcbf*
lpc_indices [2], 4-0;	5	*uimcbf*
lpc_indices [4];	1	*uimcbf*
lpc_indices [5], 0;	1	*uimcbf*
}		
} else {		
lpc_indices [0];	5	*uimcbf*
lpc_indices [1], 1-0;	2	*uimcbf*
lpc_indices [2], 6;	1	*uimcbf*
lpc_indices [2], 4-0;	5	*uimcbf*
lpc_indices [4];	1	*uimcbf*
lpc_indices [5], 0;	1	*uimcbf*
}		
for (subframe = 0; subframe < nrof_subframes; subframe++) {		
gain_indices[0][subframe], 5-3;	3	*uimcbf*
if (subframe == 0) {		
gain_indices[1][subframe], 4-3;	2	*uimcbf*
} else {		
gain_indices[1][subframe], 2;	1	*uimcbf*
}		
}		
}		

Таблица 39 — Синтаксис *RPE_WideBand_ESC*1 ()

Синтаксис	Количество битов	Мнемосхема
RPE_WideBand_ESC1 ()		
{		
if (FineRateControl == ON) {		
if (LPC_Present == YES) {		
lpc_indices [1], 3-2;	2	*uimcbf*
lpc_indices [2], 5;	1	*uimcbf*
lpc_indices [5], 1;	1	*uimcbf*
lpc_indices [6], 1-0;	2	*uimcbf*
}		
} else {		
lpc_indices [1], 3-2;	2	*uimcbf*
lpc_indices [2], 5;	1	*uimcbf*
lpc_indices [5], 1;	1	*uimcbf*
lpc_indices [6], 1-0;	2	*uimcbf*
{		
for (subframe = 0; subframe < nrof_subframes; subframe++)	3	*uimcbf*
{		
shape_delay[subframe], 7-5;		
}		
}		

Таблица 40 — Синтаксис *RPE_WideBand_ESC*2 ()

Синтаксис	Количество битов	Мнемосхема
RPE_WideBand_ESC2()		
{		
if (FineRateControl == ON) {		
if (LPC_Present == YES) {		
lpc_indices [1], 4;	1	*uimcbf*
lpc_indices [3], 6;	1	*uimcbf*
lpc_indices [3], 1;	1	*uimcbf*
lpc_indices [5], 2;	1	*uimcbf*
lpc_indices [6], 3;	1	*uimcbf*
lpc_indices [7], 6;	1	*uimcbf*
lpc_indices [7], 4;	1	*uimcbf*
lpc_indices [7], 1-0;	2	*uimcbf*
lpc_indices [9];	1	*uimcbf*
}		
} else {		
lpc_indices [1], 4;	1	*uimcbf*
lpc_indices [3], 6;	1	*uimcbf*
lpc_indices [3], 1;	1	*uimcbf*
lpc_indices [5], 2;	1	*uimcbf*
lpc_indices [6], 3;	1	*uimcbf*
lpc_indices [7], 6;	1	*uimcbf*
lpc_indices [7], 4;	1	*uimcbf*
lpc_indices [7], 1-0;	2	*uimcbf*
lpc_indices [9];	1	*uimcbf*
}		
for (subframe = 0; subframe < nrof_subframes; subframe++)		
{	2	*uimcbf*
shape_delay[subframe], 4-3;	1	*uimcbf*
gain_indices[0][subframe], 2;		
if (subframe == 0) {	1	*uimcbf*

Окончание таблицы 40

Синтаксис	Количество битов	Мнемосхема
gain_indices[1][subframe], 2; *} else {* *gain_indices[1][subframe], 1;* *}* *}* *}*	1	*uimcbf*

Таблица 41 — Синтаксис *RPE_WideBand_ESC3* ()

Синтаксис	Количество битов	Мнемосхема
RPE_WideBand_ESC3() *{* *if (FineRateControl == ON) {* *if (LPC_Present == YES) {* *lpc_indices [3], 4-2;* *lpc_indices [3], 0;* *lpc_indices [5], 3;* *lpc_indices [6], 2;* *lpc_indices [7], 5;* *lpc_indices [7], 3-2;* *lpc_indices [8], 4-1;* *}* *} else {* *lpc_indices [3], 4-2;* *lpc_indices [3], 0;* *lpc_indices [5], 3;* *lpc_indices [6], 2;* *lpc_indices [7], 5;* *lpc_indices [7], 3-2;* *lpc_indices [8], 4-1;* *}* *for (subframe = 0; subframe < nrof_subframes; subframe++)* *{* *shape_delay[subframe], 2-1;* *}* *}*	 3 1 1 1 1 2 4 3 1 1 1 1 2 4 2	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 42 — Синтаксис *RPE_WideBand_ESC4* ()

Синтаксис	Количество битов	Мнемосхема
RPE_WideBand_ESC4() *{* *if (FineRateControl == ON) {* *if (LPC_Present == YES) {* *lpc_indices [3], 5;* *lpc_indices [8], 0;* *}* *} else {* *lpc_indices [3], 5;* *lpc_indices [8], 0;* *}* *for (subframe = 0; subframe < nrof_subframes; subframe++) {* *shape_delay[subframe], 0;*	 1 1 1 1 1	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Окончание таблицы 42

Синтаксис	Количество битов	Мнемосхема
shape index[subframe]; *gain_indices[0][subframe], 1-0;* *if (subframe == 0) {* *gain_indices[1][subframe], 1-0;* *} else {* *gain_indices[1][subframe], 0;* *}* *}* *}*	11, 12 2 2 1	*uimcbf* *uimcbf* *uimcbf* *uimcbf*

3.2.2.2 Уровни расширения *CELP* (см. таблицы 43, 44)

Таблица 43 — Синтаксис *ER_SC_CelpBRSenhFrame* ()

Синтаксис	Количество битов	Мнемосхема
ER_SC_CelpBRSenhFrame() *{* *if (SilenceCompression == OFF) {* *CelpBRSenhFrame();* *}else if (TX_flag == 1) {* *CelpBRSenhFrame();* *}* *}*		

Таблица 44 — Синтаксис *ER_SC_CelpBWSenhFrame* ()

Синтаксис	Количество битов	Мнемосхема
ER_SC_CelpBWSenhFrame() *{* *if (SilenceCompression == OFF) {* *CelpBWSenhFrame();* *} else {* *if (TX_flag == 1) {* *CelpBWSenhFrame();* *}* *if (TX_flag == 2) {* *SID_BandScalable_LSP();* *}* *}* *}*		

3.2.2.3 Элементы синтаксиса для неактивных фреймов (см. таблицы 45, 46, 47, 48, 49)

Таблица 45 — Синтаксис *SID_LSP_VQ_ESC0* ()

Синтаксис	Количество битов	Мнемосхема
SID_LSP_VQ_ESC0() *{* *if (SampleRateMode == 8кГц) {* *SID_NarrowBand_LSP();* *} else {* *SID_WideBand_LSP(); }* *}* *}*		

Таблица 46 — Синтаксис *SID_NarrowBand_LSP* ()

Синтаксис	Количество битов	Мнемосхема
SID_NarrowBand_LSP() *{* *SID_lpc_indices [0];* *SID_lpc_indices [1];* *SID_lpc_indices [2]* *}*	 4 4 7 	 *Uimcbf* *uimcbf* *uimcbf*

Таблица 47 — Синтаксис *SID_BandScalable_LSP* ()

Синтаксис	Количество битов	Мнемосхема
SID_BandScalable_LSP() *{* *SID_lpc_indices [3];* *SID_lpc_indices [4];* *SID_lpc_indices [5];* *SID_lpc_indices [6];* *}*	 4 7 4 6 	 *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 48 — Синтаксис *SID_WideBand_LSP* ()

Синтаксис	Количество битов	Мнемосхема
SID_WideBand_LSP() *{* *SID_lpc_indices [0];* *SID_lpc_indices [1];* *SID_lpc_indices [2];* *SID_lpc_indices [3];* *SID_lpc_indices [4];* *SID_lpc_indices [5];* *}*	 5 5 7 7 4 4 	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 49 — Синтаксис *SID_Frame_ESC0* ()

Синтаксис	Количество битов	Мнемосхема
SID_Frame_ESC0() *{* *SID_rmc_index;* *}*	 6 	 *uimcbf*

4 Семантики

Здесь описана семантика синтаксических элементов.

4.1 Семантики заголовка

isBaseLayer — однобитовый идентификатор, показывающий является ли соответствующий уровень базовым (1), либо уровенем расширения масштабируемой полосы пропускания или масштабируемой битовой скорости (0).

isBWSLayer — однобитовый идентификатор, показывающий является ли соответствующий уровень уровенем расширения масштабируемой полосы пропускания (1) или уровенем расширения масштабируемой битовой скорости (0).

CELP-BRS-id — двухбитовый идентификатор, показывающий порядок уровней расширения масштабируемой битовой скорости, где у первого уровня расширения значение '1'. Значение '0' не должно использоваться.

ExcitationMode — однобитовый идентификатор, показывающий используется ли инструмент *Multi-Pulse Excitation* или инструмент *Regular-Pulse Excitation* (см. таблицу 50).

Таблица 50 — Описание режима *ExcitationMode*

Режим возбуждения	*ID* возбуждения	Описание
0	*MPE*	Инструмент *MPE* используется
1	*RPE*	Инструмент *RPE* используется

SampleRateMode — однобитовый идентификатор, показывающий частоту дискретизации. Поддерживаются две частоты дискретизации (см. таблицу 51).

Таблица 51 — Описание режима *SampleRateMode*

Режим *SampleRateMode*	Идентификатор *SampleRateID*	Описание
0	8 кГц	8 кГц Частота дискретизации
1	16 кГц	16 кГц Частота дискретизации

FineRateControl — однобитовый флажок, показывающий, запущено ли точное управление скоростью очень мелкими шагами или заблокировано (см. таблицу 52).

Таблица 52 — Описание *FineRateControl*

Точное управление скоростью *FineRateControl*	Идентификатор управления скоростью *RateControlID*	Описание
0	*OFF*	Точное управление скоростью отключено
1	*ON*	Точное управление скоростью запущено

Управление *FineRate Control* позволяет уменьшать битовую скорость относительно ее битовой скорости привязки. Когда параметры *LPC* передаются в каждом фрейме, будет получена битовая скорость привязки. Самая низкая возможная битовая скорость для каждой конфигурации может быть получена, передавая параметры *LPC* в 50 % фреймов.

SilenceCompression — однобитовый идентификатор, показывающий, используется ли *Silence Compression* или нет.

SilenceCompression	SilenceCompressionID	Описание
0	SC_OFF	*SilenceCompression* отключено
1	SC_ON	*SilenceCompression* задействовано

RPE_Configuration — это 3-битовый идентификатор, который конфигурирует кодер *MPEG–4 CELP*, используя инструмент *Regular-Pulse Excitation*. Этот параметр непосредственно определяет набор разрешенных битовых скоростей (таблица 53) и число подфреймов во фрейме *CELP* (таблица 54).

Таблица 53 — Назначение скорости для кодера режима I на 16 кГц

RPE_Configuration	Фиксированная битовая скорость Точная регулировка скорости выключена, бит/с	Минимальная битовая скорость Точная регулировка скорости включена, 50% *LPC*, бит/с	Максимальная битовая скорость Точная регулировка скорости включена, 100% *LPC*, бит/с
0	14400	13000	14533
1	16000	13900	16200
2	18667	17267	18800
3	22533	21133	22667
4 ... 7	Зарезервировано		

MPE_Configuration — это 5-битовое поле, которое конфигурирует кодер *MPEG-4 CELP*, используя инструмент *Multi-Pulse Excitation*. Этот параметр определяет переменные *nrof_subframes* и *nrof_subframes_bws*. Этот параметр также определяет число битов для *shape_positions[i], shape_signs[i], shape_enh_positions[i][j]* и *shape_enh_signs[i][j]*.

nrof_subframes — вспомогательный параметр, определяющий число подфреймов во фрейме *CELP*, и использующийся, чтобы сообщить, сколько раз должны быть прочитаны параметры возбуждения. Для инструмента *Regular-Pulse Excitation*, работающего при частоте дискретизации 16 кГц, эта переменная зависит от *RPE_Configuration* следующим образом.

Таблица 54 — Определение *nrof_subframes* для кодера режима I на 16 кГц

RPE_Configuration	*Nrof_subframes*
0	6
1	4
2	8
3	10
4 ... 7	Зарезервировано

Для инструмента *Multi-Pulse Excitation* это получают из *MPE_Configuration* в зависимости от частоты дискретизации (см. таблицы 55, 56).

Таблица 55 — Определение *nrof_subframes* для кодера режима II на 8 кГц

MPE_Configuration	*nrof_subframes*
0, 1, 2	4
3, 4, 5	3
6 ... 12	2
13 ... 21	4
22 ... 26	2
27	4
28 ... 31	зарезервировано

Таблица 56 — Определение *nrof_subframes* для кодера режима II на 16 кГц

MPE_Configuration	*nrof_subframes*
0 6	4
8 ... 15	8
16 ... 22	2
24 ... 31	4
7, 23	зарезервировано

NumEnhLayers — это двухбитовое поле, определяющее число уровней расширения, которые используются (см. таблицу 57).

Таблица 57 — Определение *nrof_enh_layers*

NumEnhLayers	*nrof_enh_layers*
0	0
1	1
2	2
3	3

BandwidthScalabilityMode — это однобитовый идентификатор, который указывает, запущена ли масштабируемость полосы пропускания. Этот режим имеет силу только, когда *ExcitationMode* = *MPE* (см. таблицу 58).

Таблица 58 — Описание режима *BandwidthScalabilityMode*

BandwidthScalabilityMode	*ScalableID*	Описание
0	*OFF*	Масштабируемость полосы отключена
1	*ON*	Масштабируемость полосы задействована

BWS_Configuration — это двухбитовое поле, которое конфигурирует инструмент расширения полосы пропускания. Этот идентификатор имеет силу только, когда *BandwidthScalabilityMode* = *ON*. Этот параметр определяет число битов для *shape_bws_positions [i], shape_bws_signs [i].*

nrof_subframes_bws — этот параметр, который является вспомогательной переменной, показывает число подфреймов в инструменте расширения полосы пропускания, и его получают из *MPE_Configuration* (см. таблицу 59).

Таблица 59 — Определение *nrof_subframes_bws*

MPE_Configuration	*nrof_subframes_bws*
0, 1, 2, 3	8
3, 4, 5	6
6 ... 12	4
13 ... 21	4
22 ... 26	2
27	Не имеет силы
28, 31	Зарезервировано

4.2 Семантика фрейма

interpolation_flag — это однобитовый флажок. Когда он установлен, параметры *LPC* для текущего фрейма должны быть получены, используя интерполяцию (см. таблицу 60).

Таблица 60 — Описание флажка *interpolation_flag*

interpolation_flag	*InterpolationID*	Описание
0	*OFF*	Коэффициенты *LPC* фрейма не должны интерполироваться
1	*ON*	Коэффициенты *LPC* фрейма должны быть найдены путем интерполяции

LPC_Present — этот бит указывает, присоединены ли параметры *LPC* к текущему фрейму. Эти параметры *LPC* относятся или к текущему фрейму или следующему фрейму (см. таблицу 61).

Таблица 61 — Описание *LPC_Present*

LPC_Present	*LPCID*	Описание
0	*NO*	Фрейм не содержит данные *LPC*
1	*YES*	Фрейм содержит данные *LPC*

Вместе *interpolation_flag* и флажок *LPC_Present* описывают, как параметры *LPC* должны быть получены (см. таблицу 62).

Таблица 62 — Процесс декодирования *LPC*, описанный *interpolation_flag* и флажком *LPC_Present flag*

interpolation_flag	*LPC_Present*	Описание
1	1	Параметры *LPC* текущего фрейма должны быть извлечены, используя интерполяцию. Текущий фрейм несет параметры *LPC*, принадлежащие следующему фрейму
1	0	ЗАРЕЗЕРВИРОВАНО
0	1	Параметры *LPC* текущего фрейма присутствуют в текущем фрейме
0	0	Параметры *LPC* предыдущего фрейма должны использоваться в текущем фрейме

lpc_indices [] — это мультибитовые поля, представляющие коэффициенты *LPC*. Они содержат информацию необходимую, чтобы извлечь коэффициенты *LSP*. Точная процедура извлечения описана в процессе декодирования.

shape_delay [*subframe*] — это битовое поле представляет адаптивную задержку книги шифров. Декодирование этого поля зависит от *ExcitationMode* и *SampleRateMode* (см. таблицу 63).

Таблица 63 — Число битов для *shape_delay [ ]*

ExcitationMode	*SampleRateMode*	*shape_delay* [] (биты)
RPE	16 кГц	8
MPE	8 кГц	8
MPE	16 кГц	9

shape_index [*subframe*] — этот индекс содержит информацию, необходимую чтобы извлечь фиксированное вложение книги шифров из книги шифров регулярного импульса. Число битов, занимаемых этим полем, зависит от битовой скорости (полученной из *RPE_configuration*) (см. таблицу 64).

Таблица 64 — Число битов для *shape_index [ ]*

RPE_Configuration	Число битов, представляющих *shape_index* [ ]
0	11
1	11
2	12
3	12
4 ... 7	Зарезервировано

gain_indices [0] [*subframe*] — эти битовые поля определяют адаптивное усиление книги шифров в инструменте *RPE*, используя 6 битов. Это читается из потока бит для каждого подфрейма.

gain_indices [1] [*subframe*] — эти битовые поля определяют фиксированное усиление книги шифров в инструменте *RPE*. Это читается из потока бит для каждого подфрейма. Число читаемых битов для представления этой области зависит от номера подфрейма. Для первого подфрейма это 5 битов, в то время как для остальных подфреймов это 3 бита.

gain_index [*subframe*] — это 6 или 7-битовое поле представляет усиления для адаптивной книги шифров и мультиимпульсного возбуждения для частоты дискретизации 8 кГц или 16 кГц соответственно.

gain_enh_index [*subframe*] — это 4-битовое поле представляет усиление для мультиимпульсного возбуждения расширения в кодере *CELP* на 8 кГц.

gain_bws_index [*subframe*] — это 11-битовое поле представляет усиления для адаптивной книги шифров и двух возбуждений мультиимпульса в инструменте расширения полосы пропускания.

signal_mode — это 2-битовое поле представляет тип сигнала. Эта информация используется в инструменте *MPE*. Книги шифров усиления переключаются в зависимости от этой информации (см. таблицу 65).

Таблица 65 — Описание *signal_mode*

signal_mode	Описание
0	Неголосовой
1, 2, 3	Голосовой

rmc_index — этот параметр указывает среднеквадратический уровень фрейма. Эта информация используется только в инструменте *MPE*.

shape_positions [*subframe*], *shape_signs* [*subframe*] — эти битовые поля представляют позиции импульсов и знаки импульсов для мультиимпульсного возбуждения. Длина битового поля зависит от *MPE_Configurations* (см. таблицы 66, 67).

Таблица 66 — Определения *shape_positions*[] и *shape_signs*[] для дискретизированной на 8 кГц речи

MPE_Configuration	*shape_positions*[] (битов)	*shape_signs*[] (битов)
0	14	3
1	17	4
2	20	5
3	20	5
4	22	6
5	24	7
6	22	6
7	24	7
8	26	8
9	28	9
10	30	10
11	31	11
12	32	12
13	13	4
14	15	5
15	16	6
16	17	7
17	18	8
18	19	9

Окончание таблицы 66

MPE_Configuration	*shape_positions*[] (битов)	*shape_signs*[] (битов)
19	20	10
20	20	11
21	20	12
22	18	8
23	19	9
24	20	10
25	20	11
26	20	12
27	19	6
28 … 31	Зарезервировано	

Таблица 67 — Определения *shape_positions*[] и *shape_signs*[] для дискретизированной на 16 кГц речи

MPE_Configuration	*shape_positions*[] (битов)	*shape_signs*[] (битов)
0, 16	20	5
1, 17	22	6
2, 18	24	7
3, 19	26	8
4, 20	28	9
5, 21	30	10
6, 22	31	11
7, 23	Зарезервировано	Зарезервировано
8, 24	11	3
9, 25	13	4
10, 26	15	5
11, 27	16	6
12, 28	17	7
13, 29	18	8
14, 30	19	9
15, 31	20	10

shape_enh_positions[*subframe*][], *shape_enh_signs*[*subframe*][] — эти битовые поля представляют позиции импульсов, и знаки импульсов для мультиимпульсного возбуждения в каждом уровне расширения. Длина битового поля зависит от *MPE_Configuration* (см. таблицы 68, 69).

Таблица 68 — Определение *shape_enh_positions*[][] и *shape_enh_signs*[][] для дискретизированной на 8 кГц речи

MPE_Configuration	*shape_enh_positions*[][] (битов)	*shape_enh_signs*[][] (битов)
0 ... 12	12	4
13 ... 26	4	2
27	не имеет силы	
28 ... 31	зарезервировано	

Таблица 69 — Определение *shape_enh_positions*[][] и *shape_enh_signs*[][] для дискретизированной на 16 кГц речи

MPE_Configuration	*shape_enh_positions*[][] (битов)	*shape_enh_signs*[][] (битов)
0 ... 6, 16 ... 22	12	4
8 ... 15, 24 ... 31	4	2
7, 23	зарезервировано	

shape_bws_delay[*subframe*] — это 3-битовое поле используется в декодировании адаптивной книги шифров для инструмента расширения полосы пропускания. Это значение указывает задержку, отличающуюся от задержки, описанной в *shape_delay* [].

shape_bws_positions[*subframe*], *shape_bws_signs* [*subframe*] — эти поля представляют позиции импульсов и знаки импульсов для мультиимпульсного возбуждения в инструменте расширения полосы пропускания. Длина битового поля зависит от *BWS_Configuration* (см. таблицу 70).

Таблица 70 — Определение *shape_bws_positions* [] и *shape_bws_signs* []

BWS_Configuration	*shape_bws_positions*[] (битов)	*shape_bws_signs*[] (битов)
0	22	6
1	26	8
2	30	10
3	32	12

TX_flag — двухбитовая область, указывающая режим передачи (см. таблицу 71).

Таблица 71 — Определение *TX_flag*

TX_flag	Режим передачи
0	Неактивный фрейм. Энергия фрейма или индексы *LPC* не передаются
1	Активный фрейм
2	Неактивный фрейм. Энергия фрейма или индексы *LPC* передаются
3	Неактивный фрейм. Передается только энергия фрейма, индексы *LPC* не передаются

SID_rmc_index — 6-битовое поле, указывающее энергию фрейма.

SID_lpc_indices — это мультибитовые поля, представляющие коэффициенты *LPC* для неактивных фреймов *LPC* (*TX_flag* = 2). Они содержат информацию, необходимую для извлечения коэффициентов *LSP*. Семантики потока бит для *SID_lpc_indices* показаны в таблице 72.

Таблица 72 — Семантики потока бит для *SID_lpc_indices*

Режим кодирования	Частота дискретизации, кГц	Масштабируемость полосы	Параметр	Описание
I	16	*Off*	*SID_lpc_indices[0]*	0-4th *LSPs of the 1st stage VQ*
			SID_lpc_indices[1]	5-9th *LSPs of the 1st stage VQ*
			SID_lpc_indices[2]	10-14th *LSPs of the 1st stage VQ*
			SID_lpc_indices[3]	15-19th *LSPs of the 1st stage VQ*
			SID_lpc_indices[4]	0-4th *LSPs of the 2nd stage VQ*
			SID_lpc_indices[5]	5-9th *LSPs of the 2nd stage VQ*
II	8	*On, Off*	*SID_lpc_indices[0]*	0-4th *LSPs of the 1st stage VQ*
			SID_lpc_indices[1]	5-9th *LSPs of the 1st stage VQ*
			SID_lpc_indices[2]	0-4th *LSPs of the 2nd stage VQ*
	16	*On*	*SID_lpc_indices[3]*	0-9th *LSPs of 1st stage VQ*
			SID_lpc_indices[4]	10-19th *LSPs of 1st stage VQ*
			SID_lpc_indices[5]	0-4th *LSPs of 2nd stage VQ*
			SID_lpc_indices[6]	5-9th *LSPs of 2nd stage VQ*
		Off	*SID_lpc_indices[0]*	0-4th *LSPs of 1st stage VQ*
			SID_lpc_indices[1]	5-9th *LSPs of 1st stage VQ*
			SID_lpc_indices[2]	10-14th *LSPs of 1st stage VQ*
			SID_lpc_indices[3]	15-19th *LSPs of 1st stage VQ*
			SID_lpc_indices[4]	0-4th *LSPs of 2nd stage VQ*
			SID_lpc_indices[5]	5-9th *LSPs of 2nd stage VQ*

5 Инструменты декодера *MPEG-4 CELP*

Здесь дается краткое описание функциональных возможностей, определение параметра и процессов декодирования инструментов, поддержанных ядром *MPEG-4 CELP*. Описание каждого инструмента включает три части и дополнительную часть, содержащую таблицы, используемые инструментом:

1 Описание инструмента: короткое описание функциональных возможностей инструмента дается вместе с его интерфейсом.

2 Определения: здесь описаны параметры ввода и вывода, так же как элементы справки инструмента. Каждый элемент дается полужирным шрифтом или курсивным. Полужирные названия указывают, что элемент читается из потока битов, курсивные названия указывают вспомогательные элементы. Если элементы уже используются другим инструментом, дается ссылка на предыдущее определение.

3 Процесс декодирования: процесс декодирования объяснен здесь подробно при помощи математических уравнений и кода pseudo-C.

4 Таблицы: эта дополнительная четвертая часть содержит таблицы, которые используются инструментом.

5.1 Введение в набор инструментов декодера *CELP MPEG-4*

Декодер *MPEG–4 CELP* работает при частоте дискретизации 8 или 16 кГц. Один или несколько индивидуальных блоков сгруппированы, формируя инструменты, доступные для декодирования *MPEG-4 CELP*. Поддерживаются следующие инструменты:

Демультиплексор потока бит *CELP*

Декодер *CELP LPC* и интерполятор

Узкополосный инструмент декодирования *LSP-VQ*

Широкополосный инструмент декодирования *LSP-VQ*

Инструмент декодирования *LSP-VQ* с масшабируемой полосой пропускания

Генератор возбуждения *CELP*

Инструмент генерации возбуждения регулярным импульсом
Инструмент генерации мультиимпульсного возбуждения
Инструмент генерации мультиимпульсного возбуждения масштабируемой битовой скорости
Инструмент генерации мультиимпульсного возбуждения масштабируемой полосы пропускания
Фильтр синтеза *CELP LPC*
Постпроцессор *CELP* (информативный инструмент)

Декодирование выполняется на основе фрейма, и каждый фрейм разделен на подфреймы. Фрейм в потоке бит демультиплексируется модулем демультиплексора потока бит *CELP*. Параметры, которые извлечены из потока бит, являются информацией заголовка, кодами, представляющими коэффициенты *LPC* фрейма и параметрами возбуждения для каждого подфрейма. Эти коды декодируются и интерполируются для каждого подфрейма модулем декодера и интерполятора *CELP LPC*. Для каждого подфрейма используются параметры возбуждения, чтобы генерировать сигнал возбуждения, используя модуль генератора возбуждения *CELP*. Модуль фильтра синтеза *CELP LPC* восстанавливает речевой сигнал на основании подфрейма, исходя из интерполированных коэффициентов *LPC* и сгенерированного сигнала возбуждения. Расширение синтезируемого сигнала получено дополнительным модулем постпроцессора *CELP*.

Чтобы реализовать масштабируемость битовой скорости, для генерации сигнала возбуждения используется инструмент генерации мультиимпульсного возбуждения масштабируемой битовой скорости (*MPE*). Инструмент генерации *MPE* масштабируемой битовой скорости реализуется путем добавления инструмента декодирования возбуждения расширения к инструменту генерации *MPE*, чтобы повысить качество сигнала возбуждения.

Декодер *CELP* масштабируемой полосы пропускания реализуется с использованием как инструмента *LSP-VQ* масштабируемой полосы пропускания, так и инструмента генерации *MPE* масштабируемой битовой скорости. Эти инструменты масштабирования используются, чтобы расширить полосу пропускания декодированного сигнала от 3,4 кГц до 7 кГц.

5.2 Конфигурация масштабируемого *AAC/CELP*

Когда декодер узкополосного *CELP* используется как "основной кодер" в масштабируемой конфигурации *AAC/CELP* (см. часть *T/F*) для масштабируемости большого шага, постфильтр выключен. В случае, если декодер узкополосного *CELP* работает как основной декодер в масштабируемой конфигурации, этот декодер может декодировать объект аудио *CELP* при частоте дискретизации, отличающейся от 8 кГц. Частота дискретизации объекта аудио *CELP* определяется *samplingFrequencyIndex* в *AudioSpecificInfo* (см. подраздел 1). Флажок *SampleRateMode* в заголовке *CELP*, управляющий работой декодера, должен указывать 8 кГц. В случае, если *samplingFrequencyIndex* указывает, что частота дискретизации отличается от 8000 Гц, битовая скорость, длина фрейма и задержка изменяются соответственно.

5.3 Переменные помощи

Хотя каждый инструмент имеет описание переменных, которые он использует, в этом подпункте приводятся обычно используемые многими инструментами переменные.

frame_size: Это поле указывает число отсчетов во фрейме. Декодер выводит фрейм с *frame_size* отсчетов.

nrof_subframes: Фрейм состоит из ряда подфреймов. Число подфреймов определено в этом поле.

sbfrm_size: Подфрейм состоит из ряда отсчетов, количество которых обозначено этим полем. Число отсчетов во фрейме всегда должно быть равным сумме количеств отсчетов в подфреймах. Соответственно всегда должно соблюдаться следующее соотношение

*frame_size = nrof_subframes * sbfrm_size.*

Эти три параметра зависят от настройки параметров режима кодирования и битовой скорости, как указано в таблице 73 для режима I, таблице 74 и таблице 75 для режима II.

Таблица 73 — Конфигурация кодера CELP (режим I) для частоты дискретизации 16 кГц

RPE_Configuration	*Frame_size* (#samples)	*nrof_subframes*	*sbfrm_size* (#samples)
0	240	6	40
1	160	4	40
2	240	8	30
3	240	10	24
4 ... 7	Зарезервировано		

Таблица 74 — Конфигурация кодера CELP (режим II) для частоты дискретизации 8 кГц

MPE_Configuration	*Frame_size* (*#samples*)	*nrof_subframes*	*sbfrm_size* (*#samples*)
0,1,2	320	4	80
3,4,5	240	3	80
6 ... 12	160	2	80
13 ... 21	160	4	40
22 ... 26	80	2	40
27	240	4	60
28 ... 31	Зарезервировано		

Таблица 75 — Конфигурация кодера CELP (режим II) для частоты дискретизации 16 кГц

MPE_Configuration	*Frame_size* (*#samples*)	*nrof_subframes*	*sbfrm_size* (*#samples*)
0, ..., 6	320	4	80
8, ..., 15	320	8	40
16, ..., 22	160	2	80
24, ..., 31	160	4	40
7, 23	Зарезервировано		

lpc_order: Это поле указывает число коэффициентов, используемых для линейного предсказания. Значение этого поля равно 20 для частоты дискретизации 16 кГц и 10 для 8 кГц.

num_lpc_indices. Этот параметр определяет число индексов, содержащих информацию *LPC*, которая должна читаться из потока бит. Оно не равно порядку *LPC*. *num_lpc_indices* равно 5 в режиме 8 кГц и дополнительно 6 для инструмента расширения полосы пропускания. Для частоты дискретизации 16 кГц в векторном квантователе это значение равно 10.

5.4 Элементы потока бит для набора инструментов *MPEG–4 CELP*

Описания всех переменных потока бит перечислены в разделе 4.

5.5 Демультиплексор потока бит *CELP*

5.5.1 Описание инструмента

Инструмент демультиплексора потока бит *CELP* извлекает фрейм *CELP* из полученного потока бит.

5.5.2 Определения

Все элементы потока бит и связанные переменные справки были определены в разделе 4.

5.5.3 Процесс декодирования

Декодирование элементов потока бит производится в соответствии с синтаксисом, опописанным в разделе 3.

5.6 Декодер *CELP LPC* и интерполятор

У декодера *CELP LPC* и интерполятора есть две функции:

1 Восстанавливать коэффициенты *LPC* из *lpc_indices* [].

2 Интерполировать восстановленные коэффициенты *LPC* для каждого подфрейма.

В зависимости от режима квантования *lpc_indices*[] содержат информацию, необходимую чтобы восстановить коэффициенты *LSP*. Есть три типа процессов декодирования, а именно, процесс декодирования узкополосного *LSP*, процесс декодирования широкополосного *LSP*, и процесс декодирования *LSP* с масштабируемой полосой пропускания. Выходы этого процесса являются коэффициентами фильтра, названные коэффициентами *LPC* или параметрами, которые могут использоваться в фильтре прямой формы (см. таблицу 76).

Таблица 76 — Инструменты декодирования *LPC*

Режим кодирования	Частота дискретизации	Инструмент
Mode I	16 кГц	*Wideband LSP-VQ*
Mode II	8 кГц 16 кГц 8/16 кГц (BWS)	*Narrowband LSP-VQ* *Wideband LSP-VQ* *Bandwidth Scalable* (*BWS*) *LSP-VQ*

5.6.1 Инструмент декодирования узкополосного *LSP-VQ*

5.6.1.1 Описание инструмента

Инструмент декодирования узкополосного *LSP* имеет двухступенчатую структуру *VQ*. Квантованные *LSPs* воспроизводятся добавлением декодированных в первой и во второй стадиях *LSPs*. Во второй стадии есть два процесса декодирования, с или без межфреймового предсказания, процесс декодирования выбирается согласно флажку *lpc_indices*. Затем декодированные *LSPs* интерполируются и преобразуются в коэффициенты *LPC*.

5.6.1.2 Определения

Вход

lpc_indices []: Размерность этого массива *num_lpc_indices* содержит упакованные индексы *lpc*.

Выход

int_Qlpc_coefficients[]: Этот массив содержит коэффициенты *LPC* для каждого подфрейма. Коэффициенты *LPC* декодированы и интерполированы, как описано в процессе декодирования. Коэффициенты *LPC* расположены в стеке один за другим в блоках *lpc_order*. Таким образом размерность массива равна *lpc_order* * *nrof_subframes*.

Конфигурация

lpc_order: Это поле указывает порядок используемого *LPC*.

num_lpc_indices: Это поле содержит ряд упакованных кодов *LPC*. Для узкополосного *LSP*, *num_lpc_indices* процесса декодирования установлен 5.

nrof_subframes: Это поле содержит ряд подфреймов.

Элементы справки, используемые в процессе декодирования узкополосного *LSP*:

lsp_tbl[][][]	таблицы поиска для первой стадии процесса декодирования
d_tbl[][][]	таблицы поиска для второй стадии процесса декодирования *VQ* без межфреймового предсказания
pd_tbl[][][]	таблицы поиска для второй стадии процесса декодирования *VQ* с межфреймовым предсказанием
dim[][]	размерности для квантования расщепленного вектора
sign	знак вектора кода для второй стадии процесса декодирования
idx	распакованный индекс для второй стадии процесса декодирования
lsp_first[],	*LSPs*, декодированные на первой стадии процесса декодирования
lsp_previous[]	*LSPs*, декодированные в предыдущем фрейме
lsp_predict[]	*LSPs*, предсказанные из *lsp_previous[]*и *lsp_first[]*
lsp_current[]	*LSPs*, декодированные в текущем фрейме
lsp_subframe[][]	*LSPs*, интерполированные в каждом подфрейме
ratio_predict	коэффициент для предсказания *lsp_predict[]*
ratio_sub	коэффициент интерполяции для вычисления *lsp_subframe[][]*
min_gap	минимальное расстояние между соседними *LSPs*
Convert2lpc ()	функция для преобразования *LSPs* в *LPCs*

5.6.1.3 Процесс декодирования

Процесс декодирования *LSP* для восстановления коэффициентов интерполированного *LPC* для каждого подфрейма описан ниже.

5.6.1.3.1 Преобразование индексов в *LSPs*

LSPs текущего фрейма (*lsp_current[]*), которые кодируются разбиением и двухступенчатым векторным квантованием, декодируются двухступенчатым процессом декодирования. Размерность каждого вектора описана в таблице 77 и таблице 78. *lpc_indices*[0],[1] и *lpc_indices*[2],[3] представляют индексы для первой и второй стадии, соответственно.

Таблица 77 — Размерность вектора первой стадии *LSP*

Индекс вектора разбиения: *i*	Размерность вектора: *dim*[0][*i*]
0	5
1	5

Таблица 78 — Размерность вектора второй стадии *LSP*

Индекс вектора разбиения: *i*	Размерность вектора: *dim*[1][*i*]
0	5
1	5

В первой стадии вектор *LSP* первой стадии *lsp_first[]* декодируется путем просмотра таблицы *lsp_tbl[][][]*.

```
for(i = 0; i < dim[0][0]; i++)
{
   lsp_first[i] = lsp_tbl[0][lpc_indices[0]][i];
}
for(i = 0; i < dim[0][1]; i++)
{
   lsp_first[dim[0][0]+i] = lsp_tbl[1][lpc_indices[1]][i];
}
```

Во второй стадии есть два типа процессов декодирования, а именно, процесс декодирования *VQ* без межфреймового предсказания и *VQ* — с межфреймовым предсказанием. *lpc_indices*[4] указывает, какой процесс должен быть выбран (см. таблицу 79).

Таблица 79 — Процесс декодирования для второй стадии

Индекс *LPC: lpc_indices* [4]	Процесс декодирования
0	*VQ* без межфреймового предсказания
1	*VQ* с межфреймовым предсказанием

Процесс декодирования *VQ* без межфреймового предсказания

Чтобы получить *LSPs* текущего фрейма *lsp_current[]*, декодированные векторы во второй стадии добавляются к декодированному вектору *LSP* первой стадии *lsp_first[]*. MCB для *lpc_indices*[] представляет знак декодированного вектора, а остающиеся биты представляют индекс для таблицы *d_tbl[][][]*.

```
sign = lpc_indices[2] >> 6;
idx = lpc_indices[2] & 0x3f;
if (sign==0)
{
   for (i = 0; i < dim[1][0]; i++)
   {
      lsp_current[i] = lsp_first[i] + d_tbl[0][idx][i];
   }
}
else
{
   for (i = 0; i < dim[1][0]; i++)
   {
      lsp_current[i] = lsp_first[i] – d_tbl[0][idx][i];
   }
}
sign = lpc_indices[3] >> 5;
idx = lpc_indices[3] & 0x1f;
if (sign==0)
{
   for(i = 0; i < dim[1][1]; i++)
   {
      lsp_current[dim[1][0]+i] = lsp_first[dim[1][0]+i] +
d_tbl[1][idx][i];
   }
```

```
    }
    else
    {
      for(i = 0; i < dim[1][1]; i++)
      {
        lsp_current[dim[1][0]+i] = lsp_first[dim[1][0]+i] –
d_tbl[1][idx][i];
      }
    }
```

Процесс декодирования *VQ* с межфреймовым предсказанием

Чтобы получить *LSPs* текущего фрейма *lsp_current[]*, декодированные векторы второй стадии добавляются к вектору *LSP - - lsp_predict[]*, которые предсказаны из декодированных *LSPs* предыдущего фрейма *lsp_previous[]* и декодированного вектора *LSP* первой стадии *lsp_first[]*. Таким же образом как процесс декодирования *VQ* без межфреймового предсказания, *LSP* для *lpc_indices*[] представляет знак декодированного вектора, а остающиеся биты представляют индекс для таблицы *pd_tbl[][][][]*.

```
    for (i = 0; i < lpc_order; i++)
    {
      lsp_predict[i]=(1–
ratio_predict)*lsp_first[i]
                +ratio_predict*lsp_pr
evious[i]
    }
```

где *ratio_predict = 0,5*

```
    sign = lpc_indices[2] >> 6;
    idx = lpc_indices[2] & 0x3f;
    if (sign==0)
    {
      for (i = 0; i < dim[1][0]; i++)
      {
        lsp_current[i] = lsp_predict[i] +
pd_tbl[0][idx][i];
      }
    }
    else
    {
      for (i = 0; i < dim[1][0]; i++)
      {
        lsp_current[i] = lsp_predict –
pd_tbl[0][idx][i];
      }
    }
    sign = lpc_indices[3] >> 5;
    idx = lpc_indices[3] & 0x1f;
    if(sign==0)
    {
      for(i = 0; i < dim[1][1]; i++)
      {
        lsp_current[dim[1][0]+i] = lsp_predict[dim[1][0]+i] +
pd_tbl[1][idx][i];
      }
    }
    else
    {
      for (i = 0; i < dim[1][1]; i++)
      {
```

```
            lsp_current[dim[1][0]+i] = lsp_predict[dim[1][0]+i] –
pd_tbl[1][idx][i];
        }
    }
```

5.6.1.3.2 Стабилизация *LSPs*

Декодированные *LSPs lsp_current[]* стабилизируются, чтобы гарантировать стабильность фильтра синтеза *LPC*, который получен из декодированных *LSPs*. Декодированные *LSPs* упорядочиваются в порядке возрастания, имея расстояние между соседними коэффициентами, по крайней мере, *min_gap*.

```
    for(i = 0; i < lpc_order; i++)
    {
        if (lsp_current[i] < min_gap)
        {
            lsp_current[i] = min_gap;
        }
    }
    for (i = 0; i < lpc_order-1; i++)
    {
        if (lsp_current[i+1]–lsp_current[i] < min_gap)
        {
            lsp_current[i+1] = lsp_current[i]+min_gap;
        }
    }
    for (i = 0; i < lpc_order; i++)
    {
        if (lsp_current[i] > 1–min_gap)
        {
            lsp_current[i] = 1–min_gap;
        }
    }
    for (i = lpc_order–1; i > 0; i- -)
    {
        if (lsp_current[i]–lsp_current[i–1] < min_gap)
        {
            lsp_current[i–1] = lsp_current[i]–min_gap;
        }
    }
```

где *min_gap* = 2,0/256,0

5.6.1.3.3 Интерполяция *LSPs*

Декодированные *LSPs lsp_current[]* линейно интерполируются в каждом подфрейме, используя *LSPs* предыдущего фрейма *lsp_previous[]*.

```
    for (n = 0; n < nrof_subframes; n++)
    {
        ratio_sub=(n+1)/nrof_subframes
        for(i = 0; i < lpc_order; i++)
        {
            lsp_subframe[n][i]=((1–
ratio_sub)*lsp_previous[i]+ratio_sub*lsp_current[i]));
        }
    }
```

5.6.1.3.4 Преобразование *LSP* в *LPC*

Интерполированные *LSPs* преобразуются в *LPCs*, используя вспомогательную функцию *Convert2lpc()*.

```
    for (n = 0; n < nrof_subframes; n++)
    {
        Convert2lpc (lpc_order, lsp_subframe[n],
    int_Qlpc_coefficients + n*lpc_order]);
    }
```

Преобразование *LSP* в *LPC* описано ниже. Входные *LSPs* должны быть нормализованы в диапазоне от нуля до π.

```
tmp_lpc[0] = 1,0;
for (i = 1; i < lpc_order + 1; i++) tmp_lpc[i] = 0,0;
for (i = 0; i < (lpc_order+1)*2; i++) w[i] = 0,0;
for (j = 0; j < lpc_order + 1; j++)
{
  xin1 = tmp_lpc[j];
  xin2 = tmp_lpc[j];
  for (i = 0; i < (lpc_order >> 1); i++)
  {
    n1 = i*4;
    n2 = n1+1;
    n3 = n2+1;
    n4 = n3+1;
    xout1 = –2. * cos(lsp_coefficients[i*2+0]) * w[n1] + w[n2]
+ xin1;
    xout2 = –2. * cos(lsp_coefficients[i*2+1]) * w[n3] + w[n4] +
xin2;
    w[n2] = w[n1];
    w[n1] = xin1;
    w[n4] = w[n3];
    w[n3] = xin2;
    xin1 = xout1;
  xin2 = xout2;
  }
  xout1 = xin1 + w[n4+1];
  xout2 = xin2 – w[n4+2];
  tmp_lpc[j] = 0,5 * (xout1 + xout2);
  w[n4+1] = xin1;
  w[n4+2] = xin2;
}
for (i = 0; i < lpc_order; i++)
{
  lpc_coefficients[i] = tmp_lpc[i+1];
}
```

5.6.1.3.5 Сохранение коэффициентов

После вычисления коэффициентов *LPC*, текущие *LSPs* должны быть сохранены в памяти, так как они используются для интерполяции в следующем фрейме.

```
for (i = 0; i < lpc_order; i++)
{
   lsp_previous[i] = lsp_current[i];
}
```

Сохраненные lsp_previous[] LSPs должны быть инициализированы как описано ниже, когда инициализируется весь декодер.

```
for (i = 0; i < lpc_order; i++)
{
   lsp_previous[i] = (i+1) / (lpc_order+1);
}
```

5.6.2 Инструмент декодирования широкополосного *LSP-VQ*

5.6.2.1 Описание инструмента

Процесс декодирования широкополосного *LSP* основан на процессе декодирования узкополосного *LSP*. Процесс декодирования широкополосного *LSP* состоит из двух блоков декодирования *LSP*, соединенных параллельно. Каждый из блоков декодирования идентичен процессу декодирования узкополосного *LSP* и декодирует нижнюю и верхнюю части *LSPs* соответственно. Декодированные *LSPs* объединяются и выводятся как один набор параметров.

5.6.2.2 Определения

Вход

lpc_indices[]: Размерность этого массива — *num_lpc_indices*, он содержит упакованные индексы *lpc*.

Выход

int_Qlpc_coefficients[]: Этот массив содержит коэффициенты *LPC* для каждого подфрейма. Коэффициенты *LPC* декодированы и интерполированы, как описано в процессе декодирования. Коэффициенты *LPC* расположены в стеке один за другим в блоках *lpc_order*. Таким образом, размерность массива равна *lpc_order* * *nrof_subframes*.

Конфигурация

lpc_order: Это поле указывает порядок используемого *LPC*.

num_lpc_indices: Это поле содержит число кодов упакованного *LPC*. Для процесса декодирования широкополосного *LSP num_lpc_indices* устанавливается 10.

nrof_subframes: Это поле содержит число подфреймов.

Элементы справки, используемые в процессе декодирования широкополосного *LSP*:

lsp_tbl[][][][]	таблицы поиска для первой стадии процесса декодирования
d_tbl[][][][]	таблицы поиска для второй стадии процесса декодирования *VQ* без межфреймового предсказания
pd_tbl[][][][]	таблицы поиска для второй стадии процесса декодирования *VQ* с межфреймовым предсказанием
dim[][]	размерности для квантования расщепленного вектора
sign	знак кодового вектора для второй стадии процесса декодирования
idx	распакованный индекс для второй стадии процесса декодирования
lsp_first[]	*LSPs*, декодированные на первой стадии процесса декодирования
lsp_previous[]	*LSPs*, декодированные в предыдущем фрейме
lsp_predict[]	*LSPs*, предсказанные из *lsp_previous[]* и *lsp_first[]*
lsp_current[]	*LSPs*, декодированные в текущем фрейме
lsp_current_lower[]	нижняя часть текущих *LSPs*
lsp_current_upper[]	верхняя часть текущих *LSPs*
lsp_subframe[][]	*LSPs*, интерполированные в каждом подфрейме
ratio_predict	коэффициент для предсказания *lsp_predict[]*
ratio_sub	коэффициент интерполяции для вычисления *lsp_subframe[][]*
min_gap	минимальное расстояние между смежными *LSPs*
Convert2lpc()	функция для преобразования *LSPs* в *LPCs*

5.6.2.3 Процесс декодирования

Процесс декодирования *LSP* для поиска интерполированных коэффициентов *LPC* для каждого подфрейма описан ниже.

5.6.2.3.1 Преобразование индексов в *LSPs*

Используя тот же способ, что и в процессе декодирования узкополосного *LSP*, декодируются двухступенчатым процессом декодирования *LSPs* текущего фрейма (*lsp_current[]*), которые закодированы разбиением и двухступенчатым векторным квантованием.

Сначала декодируется нижняя часть текущих *LSPs lsp_current_lower*[]. Размерность каждого вектора описана в таблице 80 и таблице 81. *lpc_indices* [0], [1] и *lpc_indices*[2],[3] представляют индексы для первой и второй стадии, соответственно.

Таблица 80 — Размерность вектора *LSP* первой стадии

Индекс вектора разбиения: *i*	Размерность вектора: *dim*[0][*i*]
0	5
1	5

Таблица 81 — Размерность вектора *LSP* второй стадии

Индекс вектора разбиения: *i*	Размерность вектора: *dim*[1][*i*]
0	5
1	5

В первой стадии вектор *LSP* первой стадии *lsp_first[]* декодируется путем просмотра таблицы *lsp_tbl[][][][]*.

```
for (i = 0; i < dim[0][0]; i++)
{
  lsp_first[i] = lsp_tbl[0][lpc_indices[0]][i];
```

```
}
for (i = 0; i < dim[0][1]; i++)
{
   lsp_first[dim[0][0]+i] =
lsp_tbl[1][lpc_indices[1]][i];
}
```

Во второй стадии *lpc_indices* [4] указывает, какой процесс должен быть выбран (см. таблицу 82).

Таблица 82 — Процесс декодирования для второй стадии

Индекс *LPC*: *lpc_indices* [4]	Процесс декодирования
0	*VQ* без межфреймового предсказания
1	*VQ* с межфреймовым предсказанием

Процесс декодирования *VQ* без межфреймового предсказания

Чтобы получить *LSPs* текущего фрейма *lsp_current_lower[]*, декодированные векторы второй стадии добавляются к декодированному вектору *LSP*, *lsp_first_lower[]*. *MCB lpc_indices[]* представляет знак декодированного вектора, а остальные биты представляют индекс для таблицы *d_tbl[][][][]*.

```
sign = lpc_indices[2] >> 6;
idx = lpc_indices[2] & 0x3f;
if (sign == 0)
{
   for (i = 0; i < dim[1][0]; i++)
   {
      lsp_current_lower[i] = lsp_first[i] + d_tbl[0][idx][i];
   }
}
else
{
   for (i = 0; i < dim[1][0]; i++)
   {
      lsp_current_lower[i] = lsp_first[i] – d_tbl[0][idx][i];
   }
}
sign = lpc_indices[3] >> 6;
idx = lpc_indices[3] & 0x3f;
if (sign == 0)
{
   for (i = 0; i < dim[1][1]; i++)
   {
      lsp_current_lower[dim[1][0]+i] = lsp_first[dim[1][0]+i] +
d_tbl[1][idx][i];
   }
}
else
{
   for (i = 0; i < dim[1][1]; i++)
   {
      lsp_current_lower[dim[1][0]+i] = lsp_first[dim[1][0]+i] –
d_tbl[1][idx][i];
   }
}
```

Процесс декодирования *VQ* с межфреймовым предсказанием

Чтобы получить *LSPs* текущего фрейма *lsp_current_lower*[], к вектору *LSP lsp_predict*[] добавляются декодированные векторы второй стадии, которые предсказаны из декодированных *LSPs* предыдущего

фрейма *lsp_previous*[] и декодированного вектора *LSP* первой стадии *lsp_first*[]. Таким же образом как в процессе декодирования *VQ* без межфреймового предсказания, *MCB lpc_indices*[] представляет знак декодированного вектора, а остающиеся биты представляют индекс для таблицы *pd_tbl*[][][].

```
for (i = 0; i < lpc_order/2; i++)
{
   lsp_predict[i]=(1–ratio_predict)*lsp_first[i] +
ratio_predict*lsp_previous[i];
}
```

где *ratio_predict* = 0.5

```
sign = lpc_indices[2] >> 6;
idx = lpc_indices[2] & 0x3f;
if (sign == 0)
{
   for (i = 0; i < dim[1][0]; i++)
   {
      lsp_current_lower[i] = lsp_predict[i] + pd_tbl[0][idx][i];
   }
}
else
{
   for (i = 0; i < dim[1][0]; i++)
   {
      lsp_current_lower[i] = lsp_predict[i] – pd_tbl[0][idx][i];
   }
}
sign = lpc_indices[3] >> 6;
idx = lpc_indices[3] & 0x3f;
if (sign == 0)
{
   for (i = 0; i < dim[1][1]; i++)
   {
      lsp_current_lower[dim[1][0]+i]=lsp_predict[dim[1][0]+i]+pd_
tbl[1][idx][i];
   }
}
else
{
   for (i = 0; i < dim[1][1]; i++)
   {
      lsp_current_lower[dim[1][0]+i]=lsp_predict[dim[1][0]+i]–
pd_tbl[1][idx][i];
   }
}
```

Затем, декодируется верхняя часть текущих *LSPs* тем же способом как в процессе декодирования нижней части. Размерность каждого вектора описана в таблице 83 и таблице 84. *lpc_indices*[5],[6] и *lpc_indices*[7],[8] представляют индексы для первой и второй стадии соответственно.

Таблица 83 — Размерность вектора *LSP* первой стадии

Индекс вектора разбиения: *i*	Размерность вектора: *dim*[0][*i*]
0	5
1	5

Таблица 84 — Размерность вектора *LSP* второй стадии

Индекс вектора разбиения: *i*	Размерность вектора: *dim*[1][*i*]
0	5
1	5

```
For (i = 0; i < dim[0][0]; i++)
{
   lsp_first[i] = lsp_tbl[0][lpc_indices[5]][i];
}
for (i = 0; i < dim[0][1]; i++)
{
   lsp_first[dim[0][0]+i] =
lsp_tbl[1][lpc_indices[6]][i];
}
```

Во второй стадии *lpc_indices* [9] указывает, какой процесс должен быть выбран (см. таблицу 85).

Т а б л и ц а 85 — Процесс декодирования для второй стадии

Индекс *LPC*: *lpc_indices* [9]	Процесс декодирования
0	*VQ* без межфреймового предсказания
1	*VQ* с межфреймовым предсказанием

Процесс декодирования *VQ* без межфреймового предсказания

```
sign = lpc_indices[7] >> 6;
idx = lpc_indices[7] & 0x3f;
if (sign == 0)
{
   for (i = 0; i < dim[1][0]; i++)
   {
      lsp_current_upper[i] = lsp_first[i] + d_tbl[0][idx][i];
   }
}
Else
{
   for (i = 0; i < dim[1][0]; i++)
   {
      lsp_current_upper[i] = lsp_first[i] – d_tbl[0][idx][i];
   }
}
sign = lpc_indices[8] >> 4;
idx = lpc_indices[8] & 0x0f;
if (sign == 0)
{
   for (i = 0; i < dim[1][1]; i++)
   {
      lsp_current_upper[dim[1][0]+i] = lsp_first[dim[1][0]+i] +
d_tbl[1][idx][i];
   }
}
Else
{
   for (i = 0; i < dim[1][1]; i++)
   {
      lsp_current_upper[dim[1][0]+i] = lsp_first[dim[1][0]+i] –
d_tbl[1][idx][i];
   }
}
```

Процесс декодирования *VQ* с межфреймовым предсказанием

```
for (i = 0; i < lpc_order/2; i++)
{
   lsp_predict[i]=(1–
ratio_predict)*lsp_first[i]+ratio_predict*lsp_previous [lpc_orde
```

```
r/2+i];
}
```

где *ratio_predict = 0.5*

```
sign = lpc_indices[7] >> 6;
idx = lpc_indices[7] & 0x3f;
if (sign == 0)
{
   for (i = 0; i < dim[1][0]; i++)
   {
      lsp_current_upper[i] = lsp_predict[i] + pd_tbl[0][idx][i];
   }
}
else
{
   for (i = 0; i < dim[1][0]; i++)
   {
      lsp_current_upper[i] = lsp_predict[i] – pd_tbl[0][idx][i];
   }
}
sign = lpc_indices[8] >> 4;
idx = lpc_indices[8] & 0x0f;
if (sign == 0)
{
   for (i = 0; i < dim[1][1]; i++)
   {
      lsp_current_upper[dim[1][0]+i] = lsp_predict[dim[1][0]+i] +
pd_tbl[1][idx][i];
   }
}
else
{
   for (i = 0; i < dim[1][1]; i++)
   {
      lsp_current_upper[dim[1][0]+i] = lsp_predict[dim[1][0]+i] –
pd_tbl[1][idx][i];
   }
}
```

Наконец декодированные *LSPs lsp_current_lower[]* и *lsp_current_upper[]* объединяются и сохраняются в массиве *lsp_current[]*.

```
for (i = 0; i < lpc_order/2; i++)
{
   lsp_current[i] = lsp_current_lower[i]; }
for (i = 0; i < lpc_order/2; i++)
{
   lsp_current[lpc_order/2 + i] = lsp_current_upper[i];
}
```

5.6.2.4 Стабилизация *LSPs*

Декодированные *LSPs lsp_current[]* стабилизируются, чтобы гарантировать стабильность фильтра синтеза *LPC*, который получен из декодированных *LSPs*. Декодированные *LSPs* упорядочены в порядке возрастания, имея расстояние между смежными коэффициентами, по крайней мере, *min_gap*.

```
for (i = 0; i < lpc_order; i++)
{
   if (lsp_current[i] < min_gap)
   {
      lsp_current[i] = min_gap;
   }
```

```
    }
    for (i = 0; i < lpc_order – 1; i++)
    {
      if (lsp_current[i+1]–lsp_current[i] < min_gap)
      {
        lsp_current[i+1] = lsp_current[i]+min_gap;
      }
    }
    for (i = 0; i < lpc_order; i++)
    {
      if (lsp_current[i] > 1–min_gap)
      {
        lsp_current[i] = 1–min_gap;
      }
    }
    for (i = lpc_order - 1; i > 0; i - -)
    {
      if (lsp_current[i]–lsp_current[i–1] < min_gap)
      {
        lsp_current[i–1] = lsp_current[i]-min_gap;
      }
    }
```

где *lpc_order* = 20 и *min_gap* = 1.0/256.0

5.6.2.5 Интерполяция *LSPs*

Декодированные *LSPs lsp_current[]* интерполированы линейно в каждом подфрейме, используя *LSPs* предыдущего фрейма *lsp_previous[]*.

```
    for (n = 0; n < nrof_subframes; n++)
    {
      ratio_sub=(n+1)/nrof_subframes
      for (i = 0; i < lpc_order; i++)
      {
        lsp_subframe[n][i] = ((1–
    ratio_sub)*lsp_previous[i]+ratio_sub*lsp_current[i]));
      }
    }
```

5.6.2.6 Преобразование *LSP* в *LPC*

Интерполированные *LSPs* преобразуются в *LPCs* с использованием вспомогательной функции *Convert2lpc()*.

```
    for (n = 0; n < nrof_subframes; n++)
    {
      Convert2lpc (lpc_order, lsp_subframe[n],
    int_Qlpc_coefficients + n*lpc_order]);
    }
```

5.6.2.7 Сохранение коэффициентов

После вычисления коэффициентов *LPC* текущие *LSPs* должны быть сохранены в памяти, так как они используются для интерполяции в следующем фрейме.

```
    for (i = 0; i < lpc_order; i++)
    {
      lsp_previous[i] = lsp_current[i];
    }
```

Сохраненные *LSPs lsp_previous[]* должны быть инициализированы как описано ниже, когда инициализируется весь декодер.

```
    for (i = 0; i < lpc_order; i++)
    {
      lsp_previous[i] = (i+1) / (lpc_order+1);
    }
```

5.6.3 Инструмент декодирования масштабируемого по полосе пропускания *LSP-VQ*

5.6.3.1 Описание инструмента

Инструмент декодирования масштабируемого по полосе пропускания *LSP-VQ* используется, чтобы добавить масштабируемость полосы пропускания к кодеру режима II. В инструменте декодирования масштабируемого по полосе пропускания *LSP-VQ* декодирующий инструмент, инструмент расширения полосы пропускания соединен с инструментом декодирования узкополосного *LSP-VQ*. Коэффициенты *LPC* для каждого подфрейма восстанавливаются путем поиска *lpc_indices*[] и преобразовывая декодированные *LSPs* в узкополосном режиме.

5.6.3.2 Определения

Вход

lpc_indices []: Размерность этого массива равна *num_lpc_indices*, содержит упакованные индексы lpc.

lsp_current []: Этот массив содержит декодированные параметры *LSP*, которые нормализованы в диапазоне от нуля до *PI*, в инструменте декодирования узкополосного *LSP*. Эти параметры получены как промежуточные параметры в процессе декодирования узкополосного *LSP*, описанном в 5.6.1 и переданным в инструмент декодирования масштабируемого по полосе пропускания *LSP*.

Выход

int_Qlpc_coefficients[]: Этот массив содержит коэффициенты *LPC* для каждого подфрейма. Коэффициенты *LPC* декодируются и интерполируются, как описано в процессе декодирования. Коэффициенты *LPC* расположены в стеке один за другим в блоках *lpc_order*. Таким образом, размерность массива — *lpc_order* * *nrof_subframes*.

Конфигурация

lpc_order: Это поле указывает порядок используемого *LPC*.

num_lpc_indices: Это поле содержит число кодов упакованного *LPC*. Для процесса декодирования *LSP* с масштабируемой полосой пропускания *num_lpc_indices* устанавливается в 11.

nrof_subframes: Это поле содержит число подфреймов.

nrof_subframes_bws: Этот параметр, который является переменной справки, представляет число подфреймов в инструменте расширения полосы пропускания.

Следующие элементы справки используются в процедуре декодирования

lsp_bws_tbl[][][][][]	таблицы поиска для процесса декодирования *LSP* с масштабируемой полосой пропускания
lsp_bws_bu[][]	буфер, содержащий остаток предсказания *LSP*
bws_ma_prdct[][][]	коэффициенты для предсказания скользящего среднего значения
bws_nw_prdct[]	коэффициенты предсказания для преобразования для *LSPs* из узкополосных в широкополосные
lsp_current[]	декодированные *LSPs* в узкополосном кодере *CELP*
lsp_bws_current[]	декодированные *LSPs* в текущем фрейме в инструменте расширения полосы пропускания
lsp_bws_previous[]	декодированные *LSPs* в предыдущем фрейме в инструменте расширения полосы пропускания
lpc_order_bws	порядок *LPC* в инструменте расширения полосы пропускания (=20). Это удвоенный *lpc_order* в узкополосном *CELP*.

5.6.3.3 Процесс декодирования

Коэффициенты *LPC* в каждом подфрейме воспроизводятся, используя *lpc_indices* [5],..., *lpc_indices*[10] и текущие *LSPs* (*lsp_current[]*) для узкополосного кодера *CELP*. Процедура декодирования состоит из трех шагов, декодирования *LSPs* в последнем подфрейме, интерполяции для других подфреймов и преобразования *LSPs* в коэффициенты *LPC*.

Процесс декодирования инструмента расширения полосы пропускания для параметров *LSP*.

Модуль межфреймового предсказания производит оценку параметров *LSP*, преобразовывая декодированные параметры *LSP* (*lsp_current[]*) в инструменте декодирования узкополосного *LSP-VQ*. Оцениваемые остаточные параметры *LSP* декодируются из *lpc_indices*[] посредством межфреймового предсказания скользящего среднего значения (*MA*).

LSPs в последнем подфрейме (*lsp_bws_current[]*) для инструмента расширения полосы пропускания восстановливаются просмотром таблицы и предсказанием следующим образом:

```
for (i = 0; i < 5; i++)
{
  lsp_bws_buf[0][i]=lsp_bws_tbl[0][lpc_indices[5]][i]+lsp_bws_tbl[
2][lpc_indices[7]][i];
}
for (i = 5; i < 10; i++)
{
  lsp_bws_buf[0][i]=lsp_bws_tbl[0][lpc_indices[5]][i]+lsp_bws_tbl[
3][lpc_indices[8]][i–5];
}
for (i = 10; i < 15; i++)
{
  lsp_bws_buf[0][i]=lsp_bws_tbl[1][lpc_indices[6]][i–
10]+lsp_bws_tbl[4][lpc_indices[9]][i–10];
}
for (i = 15; i < 20; i++)
{
  lsp_bws_buf[0][i]=lsp_bws_tbl[1][lpc_indices[6]][i-
10]+lsp_bws_tbl[5][lpc_indices[10]][i-15];
}
for (n = 0; n <= 2; n++)
{
  for (i = 0; i < 20; i++)
  {
    lsp_bws_current[i] += bws_ma_prdct[n][i]*lsp_bws_buf[n][i];
  }
}
for (i = 0; i < 10; i++)
{
  lsp_bws_current[i] += bws_nw_prdct[i]*lsp_current[i];
}
```

где *lsp_bws_tbl[][][]* являются книгами шифров *LSP*, *bws_ma_prdct[][]* и *bws_nw_prdct[]* являются коэффициентами предсказания для скользящего среднего значения межфреймового предсказания и внутрифреймового предсказания, соответственно. *lsp_bws_buf[][]* является буфером, содержащим остаток предсказания *LSP* в текущем фрейме и предыдущих двух. Этот буфер сдвигается для работы следующего фрейма следующим образом:

```
for (n = 2; n > 0; n- -)
{
  for (i = 0; i < 20; i++)
  {
    lsp_bws_buf[n][i] = lsp_bws_buf[n–1][i];
  }
}
```

Декодированные *LSPs lsp_bws_current[]* стабилизируются, чтобы гарантировать стабильность фильтра синтеза *LPC*, который получен из декодированных *LSPs*. Декодированные *LSPs* упорядочены в порядке возрастания, имея расстояние между смежными коэффициентами, по крайней мере, *min_gap*.

```
for (i = 0; i < lpc_order_bws; i++)
{
  if (lsp_bws_current[i] < 0.0 || lsp_bws_current[i] > PAI)
  {
    lsp_bws_current[i] = 0.05 + PAI * i / lpc_order_bws;
  }
}
for (i = (lpc_order_bws–1); i > 0; i- -)
{
```

```
        for (j = 0; j < i; j++)
        {
          if (lsp_bws_current[j] + min_gap > lsp_bws_current[j+1])
          {
            tmp = 0.5 * (lsp_bws_current[j] + lsp_bws_current[j+1]);
            lsp_bws_current[j] = tmp – 0.51 * min_gap;
            lsp_bws_current[j+1] = tmp + 0.51 * min_gap;
          }
        }
      }
```

где *PAI* = 3,141592 и *min_gap* = 0,028.

Интерполяция параметров *LSP*

Декодированные *LSPs* линейно интерполируются в каждом подфрейме.

```
for (n = 0; n < nrof_subframes_bws; n++)
{
  ratio_sub = (n+1)/nrof_subframes_bws;
  for (i = 0; i < 2*lpc_order; i++)
  {
    lsp_bws_subframe[n][i]=((1–ratio_sub)*lsp_bws_previous[i]
                            + ratio_sub*lsp_bws_current[i]));
  }
}
for (i = 0; i < 2*lpc_order; i++)
{
    lsp_bws_previous[i] = lsp_bws_subframe[nrof_subframes_bws–
1][i];
}
```

Преобразование *LSPs* в коэффициенты *LPC*

Интерполированные *LSPs* преобразуются в коэффициенты *LPC* в каждом подфрейме.

```
for (n = 0; n < nrof_subframes_bws; n++)
{
  Convert2lpc (lpc_order_bws,lsp_bws_subframe[n],
                        &int_Qlpc_coefficients[n*lpc_order_bws]);
}
```

5.6.4 Управление скоростью в инструменте декодирования *LSP*

5.6.4.1 Описание инструмента

Управление скоростью возможно с помощью инструмента декодирования *LSP*. Если используется *FRC*, во входные параметры включаются два дополнительных параметра *interpolation_flag* и *LPC_present*. Процесс декодирования узкополосного *LSP* выполняется, если в текущем фрейме присутствуют *lpc_indices*[] (*LPC_present* = *YES*).

5.6.4.2 Определения

Вход

lpc_indices[]: Размерность этого массива равна *num_lpc_indices* и содержит индексы *LPC*.

interpolation_flag: Это однобитовый флажок. Когда он установлен, флажок указывает, что рассматриваемый фрейм является неполным фреймом, то есть фрейм не несет коэффициенты *LPC* текущего речевого фрейма, а только его параметры возбуждения (параметры адаптивной и фиксированной книги шифров). Коэффициенты *LPC* для рассматриваемого речевого фрейма должны быть получены, используя интерполяцию коэффициентов *LPC* смежных фреймов.

1 коэффициенты *LPC* фрейма должны быть найдены интерполяцией

0 коэффициенты *LPC* фрейма не нужно интерполировать.

Для того чтобы поддержать хорошее субъективное качество, никогда не должно быть более одного фрейма подряд без информации *LPC*, то есть, *interpolation_flag* не может иметь значения 1 в двух последовательных *CelpFrames*.

LPC_Present — Это поле указывает присутствие параметров *LPC* в рассматриваемом речевом фрейме. Эти коэффициенты *LPC* принадлежат текущему речевому фрейму или последующему фрейму. Когда используются в комбинации с *interpolation_flag* эти два параметра полностью описывают, как получены коэффициенты *LPC* текущего фрейма. Если установлен флажок интерполяции, коэффициенты *LPC* текущего фрейма вычисляются при использовании коэффициентов *LPC* предыдущего и следующего фрейма. Это означает, что декодирование текущего фрейма должно быть отсрочено на один фрейм. Чтобы избежать этой дополнительной задержки в декодере, коэффициенты *LPC* следующего фрейма включены в текущий фрейм. В этом случае установлен флажок *LPC_Present*. Так как коэффициенты *LPC* следующего фрейма уже присутствуют в текущем фрейме, следующий фрейм не будет содержать информацию *LPC*.

Если *interpolation_flag* равен "1" и *LPC_Present* равно "1", то (*a*) параметры *LPC* текущего фрейма получают, используя параметры *LPC* предыдущего фрейма и параметры следующего фрейма, и (*b*) текущий фрейм (фрейм на рассмотрении) переносит параметры *LPC* последующего фрейма, но не такие, как для рассматриваемого фрейма. Коэффициенты *LPC* рассматриваемого фрейма получаются интерполяцией ранее полученных параметров *LPC* и параметров *LPC*, полученных в рассматриваемом фрейме.

Если *interpolation_flag равен* "0" и *LPC_Present* равно "0", параметры *LPC*, которые будут использоваться с рассматриваемым фреймом, получены в предыдущем фрейме.

Когда *interpolation_flag* "0" и *LPC_Present* равно "1", тогда текущий фрейм является полным фреймом, и параметры *LPC*, полученные в текущем фрейме, принадлежат текущему фрейму.

Такая конструкция выбрана для того, чтобы минимизмровать задержку, когда декодер начинает восстанавливать фрейм, коэффициенты *LPC* которого получены, используя интерполяцию, не имея необходимости ждать прибытия следующего фрейма. Такая комбинация позволяет декодировать поток бит от любого пункта (произвольный доступ). В конфигурации с фиксированной битовой скоростью эти два флажка показывают фиксированный шаблон

01, 01, 01, 01, 01, 01, ... строка 01 повторяется

11, 00, 11, 00, 11, 00, ... строка 11, 00 повторяется (фиксированная битовая скорость достигается на протяжении двух фреймов)

Для переменной битовой скорости (когда *FineRateControl* = *ON*) строка обычно не будет содержать фиксированный шаблон.

Выход

int_Qlpc_coefficients[]: Этот массив содержит коэффициенты *LPC* для каждого подфрейма. Коэффициенты *LPC* декодируются и интерполируются, как описано в процессе декодирования.

5.6.4.3 Процесс декодирования

Если *interpolation_flag* установлен, декодированные в текущем фрейме *LSPs* принадлежат следующему фрейму, а *LSPs* для текущего фрейма вычисляются линейной интерполяцией *LSPs* соседних фреймов.

Если *interpolation_flag* не установлен и *lpc_indices*[] присутствуют в текущем фрейме, вычисленные коэффициенты принадлежат текущему фрейму, и интерполяцию выполнять не нужно.

Если *interpolation_flag* не установлен и *lpc_indices*[] отсутствует в текущем фрейме, *LSPs* для текущего фрейма уже получены в предыдущем фрейме. Поэтому *LSPs*, полученные в предыдущем фрейме, копируются и используются в текущем фрейме.

5.7 Генератор возбуждения *CELP*

Генератор возбуждения *CELP* генерирует сигнал возбуждения для одного подфрейма из полученных индексов, используя процесс возбуждения регулярным импульсом (*RPE*) или процесс мультиимпульсного возбуждения (*MPE*) в зависимости от режима кодирования (см. таблицу 86).

Таблица 86 — Инструменты генерации возбуждения

Режим кодирования	Частота дискретизации	Инструмент
Mode I	16 кГц	*RPE*
Mode II	8,16 кГц 8,16 кГц 8/16 кГц (BWS)	*MPE* *MPE* с масштабируемой битовой скоростью *MPE* с масштабируемой полосой

5.7.1 Инструмент возбуждения регулярным импульсом

5.7.1.1 Описание инструмента

Сигнал возбуждения создается из периодического компонента (вклад адаптивной книги шифров) и непериодического компонента (вклад *RPE*), масштабированных их соответствующими усилениями. Используя *shape_delay*[*sub_frame*] и *gain_indices*[0][*sub_frame*], вычисляется вклад адаптивной книги шифров. Вклад *RPE* вычисляется путем использования *shape_index*[*sub_frame*] и *gain_indices*[1][*sub_frame*]. Для ясности индексация, основанная на *sub_frame*, опущена. Процесс генерации возбуждения повторяется каждый подфрейм.

5.7.1.2 Определения

Вход

shape_delay[]: Этот массив имеет размерность *nrof_subframes* и содержит задержку адаптивной книги шифров.

shape_index[]: Этот массив имеет размерность *nrof_subframes* и содержит индекс книги шифров *RPE*.

gain_indices[0][]: Этот массив имеет размерность *nrof_subframes* и содержит индекс усиления адаптивной книги шифров.

gain_indices[1][]: Этот массив имеет размерность *nrof_subframes* и содержит усиление книги шифров *RPE*.

Выход

excitation[]: Этот массив имеет размерность *sbfrm_size* и содержит сигнал возбуждения. Этот сигнал восстановлен из векторов формы и усиления, используя адаптивные и фиксированные книги шифров.

lag: Это поле содержит декодированную задержку (период шага) для адаптивного codevector.

adaptive_gain: Эта область содержит декодированное усиление для адаптивного codevector.

Конфигурация

sbfrm_size: Это поле указывает число отсчетов в подфрейме.

nrof_subframes: Это поле указывает число подфреймов.

Дополнительные элементы, используемые в режиме возбуждения *RPE*, следующие:

tbl_cba_gain[]	таблица поиска для усиления адаптивной книги шифров
tbl_cbf_gain[]	таблица поиска для усиления фиксированной книги шифров
tbl_cbf_gain_dif[]	таблица поиска для разницы усиления фиксированной книги шифров
cba[]	адаптивная книга шифров
prev_Gf	усиление фиксированной книги шифров предыдущего подфрейма

5.7.1.3 Процесс декодирования

5.7.1.3.1 Декодер формы

Этот блок описывает извлечение запаздывания адаптивной книги шифров и параметров *RPE*. Задержка адаптивной книги шифров получается из *shape_delay* следующим образом:

lag = *Lmax* — *Lmin* — *shape_delay*,

где *Lmax* и *Lmin* — максимальная и минимальная задержка, соответственно, равные 295 и 40. Коэффициент уменьшения *D* и число импульсов Np сведены в таблицу.

Параметры *RPE*, а именно, фаза *RPE* (*rpe_phase*) и амплитуды *RPE* (*rpe_amps*) получают следующим образом:

```
rpe_index = shape_index;
rpe_phase = rpe_index MOD D
rpe_index = rpe_index / D;
for (n = Np – 1; n> = 0; n–)
{
   rpe_amps [n] = (rpe_index MOD 3) – 1;
   rpe_index = rpe_index / 3;
}
```

5.7.1.3.2 Декодер усиления

Этот блок извлекает скалярное квантованное усиление адаптивной и фиксированной книг шифров из индексов усиления. Усиление адаптивной книги шифров Ga определяется поиском *cba_gain* [] в таблице (см. таблицу 88):

```
if (gain_indices[0] > 31)
{
   Ga = –1 * cba_gain[((64 – gain_indices[0]) – 1)];
```

```
}
Else
{
   Ga = cba_gain[gain_indices[0]];
}
```

Декодирование усиления фиксированной книги шифров зависит от рассматриваемого подфрейма. Для поиска усиления фиксированной книги шифров *Gf* используется вектор усиления. Для первого подфрейма во фрейме усиление декодируется с:

Gf = cbf_gain[gain_indices[1]],

где *cbf_gain*[] дается в таблице 89. Для всех последующих подфреймов усиление декодируется, используя *cbf_gain_dif*[], представленный в таблице 90:

*Gf = cbf_gain_dif[gain_indices[1]] * prev_Gf,*

где *prev_Gf* — декодированное усиление предыдущих подфреймов.

5.7.1.3.3 Генерация адаптивной книги шифров

Сначала будет описана генерация сигнала возбуждения, обусловленная адаптивной книгой шифров. Возбуждение для адаптивной книги шифров *ya*[*n*] вычисляется, используя вектор формы:

```
for (n = 0; n < sbfrm_size; n ++)
{
   ya[n] = cba[lag + n];
}
```

5.7.1.3.4 Генерация фиксированной книги шифров

Возбуждение для фиксированной книги шифров вычисляется в два шага. Чтобы генерировать фиксированную книгу шифров, также нужны два дополнительных параметра, а именно, *D* (интервал импульса или коэффициент уменьшения) и *Np* (число импульсов).

Эти значения зависят от битовой скорости и сведены в таблицу 87.

Таблица 87 — Распределение интервала импульса и числа импульсов в *RPE*

RPE_Configuration	*D*	*Np*
0	8	5
1	8	5
2	5	6
3	4	6
4 7	Зарезервировано	

Когда фаза и амплитуды известны, амплитуды помещают в регулярную сетку. У амплитуд может быть 3 различных значения:–1, 0 и 1. Возбуждение для фиксированной книги шифров, *yf*, вычисляется, используя *RPE*-формулу:

```
for (n = 0; n < sbfrm_size; n++)
{
   yf[n] = 0;
}
for (n = 0; n < Np; n++)
{
   yf[phase + D*n] = amp[n];
}
```

5.7.1.3.5 Генерация возбуждения

Возбуждение является суммой адаптивного и фиксированного возбуждения, умноженных на соответствующие усиления:

```
for (n = 0; n < sbfrm_size; n++)
{
   excitation[n] = Ga * ya[n] + Gf * yf[n];
}
```

Должна быть обновлена адаптивная книга шифров, используя возбуждение. Это делается путем сдвига адаптивной книги шифров на *sbfrm_size* входов, заполняя пустые входы вычисленным возбуждением.

```
for (n = sbfrm_size; n < Lmax; n++)
{
   cba[n–sbfrm_size] = cba[n];
}
for (n = 0; n < sbfrm_size; n++)
{
   cba[Lmax–1–n] = excitation[sbfrm_size – 1 – n];
}
```

Адаптивная книга шифров инициализируется заполнением ее нулями.

Представления для усиления книги шифров даны в таблице 88, таблице 89 и таблице 90.

Т а б л и ц а 88 — Таблица представления для усиления адаптивной книги шифров

Масштабный коэффициент: 2^17

Индекс	*cba_gain*
0	12386
1	27800
2	38483

Индекс	*cba_gain*
3	471471
4	548272
5	61669

Масштабный коэффициент: 2^15

Индекс	*cba_gain*
6	16993
7	18520
8	20021
9	21493
10	22938
11	24396
12	25834
13	27292
14	28767

Индекс	*cba_gain*
15	318778
16	302847
17	336079
18	3555010
19	3780811
20	4061312
21	4412213
22	4884714
23	55745

Масштабный коэффициент: 2^8

Индекс	*cba_gain*
24	528
25	727
26	1388
27	2777

Индекс	*cba_gain*
28	5553
29	11107
30	22214
31	44426

Т а б л и ц а 89 — Таблица представления для усиления фиксированной книги шифров

Масштабный коэффициент: 2^12

Индекс	*cba_gain*
0	8192
1	11578
2	14877
3	199909

Индекс	*cba_gain*
4	265731
5	343242
6	441603
7	55564

Масштабный коэффициент: 2^9

Индекс	*cba_gain*
8	8739
9	10823
10	13458
11	16451
12	19978
13	24039

Индекс	*cba_gain*
14	28624
15	33682
16	39118
17	45372
18	52626
19	60243

Масштабный коэффициент: 2^5

Индекс	*cba_gain*
20	4327
21	4954
22	5700
23	6551
24	7649
25	8977

Индекс	*cba_gain*
26	10689
27	13066
28	16183
29	21291
30	32832
31	Зарезервирован

Т а б л и ц а 90 — Таблица представления для дифференциала усиления фиксированной книги шифров

Масштабный коэффициент: 2^13

Индекс	*cba_gain*
0	819
1	3205
2	5418
3	7335

Индекс	*cba_gain*
4	9544
5	12763
6	22118
7	53248

5.7.2 Инструмент генерации мультиимпульсного возбуждения

5.7.2.1 Описание инструмента

Процесс генерации возбуждения повторяется каждый подфрейм. Сигнал возбуждения создается из периодического компонента (вектор адаптивной книги шифров) и непериодического компонента (вектор фиксированной книг шифров), масштабированных их соответствующими усилениями. Векторы адаптивной и фиксированной книги шифров декодированы из *shape_delay*, *shape_positions* и *shape_signs*. Усиления декодируются исходя из трех типов индексов *signal_mode*, *rms_index* и *gain_index*.

5.7.2.2 Определения

Вход

shape_delay[]: Этот массив имеет размерность *nrof_subframes* и содержит задержку адаптивной книги шифров

shape_positions[]: Этот массив имеет размерность *nrof_subframes* и содержит позиции импульса

shape_signs[]: Этот массив имеет размерность *nrof_subframes* и содержит знаки импульса

gain_index[]: Этот массив имеет размерность *nrof_subframes* и содержит индекс усиления адаптивной книги шифров и индекс усиления фиксированной книги шифров

rms_index: Это поле определяет индекс для мощности сигнала

signal_mode: Это поле содержит флажок голосовой/неголосовой

int_Qlpc_coefficients[]: Это — массив размерности *lpc_order*, содержит коэффициенты квантованного и интерполированного *LPC* одного подфрейма.

Выход

excitation[]: Этот массив имеет размерность *sbfrm_size* и содержит сигнал возбуждения. Этот сигнал реконструируется из векторов формы и усиления, используя адаптивные и фиксированные книги шифров

acb_delay: Это поле содержит декодированную задержку для адаптивного *codevector*
adaptive_gain: Это поле содержит декодированное усиление для адаптивного *codevector*.
Конфигурация
lpc_order Это поле указывает порядок *LPC*, который используется
sbfrm_size Это поле указывает число отсчетов в подфрейме
nrof_subframes Это поле указывает число подфреймов.

Дополнительные элементы, используемые в инструменте *MPE*, следующие:

pacb[] таблица поиска возбуждения для периодического компонента
pos_tbl[][][] таблица поиска возбуждения для непериодического компонента
acb[] декодированный сигнал возбуждения как периодический компонент
fcb[] декодированный сигнал возбуждения как непериодический компонент
ga декодированное усиление для периодического компонента
gf декодированное усиление для непериодического компонента
qxnorm[] декодированные среднеквадратические значения речевого сигнала
par[] коэффициенты отражения, преобразованные из *int_Qlpc_coefficients[]*
acb_energy энергия *acb[]*
fcb_energy энергия *fcb[]*
acb_delay целая часть задержки шага
acb_frac дробная часть задержки шага

5.7.2.3 Процесс декодирования

5.7.2.3.1 Декодирование *signal_mode*

Signal_mode (режим сигнала) представляет собой один из четырех режимов для каждого фрейма. Режимы 0 и 1 соответствуют неголосовому фрейму и фрейму перехода. Режимы 2 и 3 соответствуют голосовым фреймам, а последний указывает более высокую периодичность шага, чем первый. Эта информация используется в декодировании энергии фрейма, мультиимпульсного возбуждения и усилений.

5.7.2.3.2 Декодирование энергии фрейма

Среднеквадратическое значение (*rms*) в последнем подфрейме восстанавливается, используя *signal_mode* и *rmc_index*. Значение *rms* декодировано в шкале μ — характеристики. Параметры μ — характеристики зависят от *signal_mode*. Среднеквадратические значения других подфреймов получают линейной интерполяцией декодированных среднеквадратических значений в последнем подфрейме текущего и предыдущего фреймов. Квантованные значения *rms* используются для процесса декодирования усиления.

```
delt = 1.0 / 64;
aa = 1.0 / log10(1.0 + mu_law);
bb = rms_max / mu_law;
pwk = aa * log10(1.0 + pqxnorm / bb);
qwk = delt*(rms_index+1);
for (i = 0; i < n_subframes; i++ )
{
   nwk = (qwk–pwk)*(i+1)/n_subframes + pwk;
   qxnorm[i] = bb * (pow((double)10.0,(nwk/aa)) – 1.0);
}
pqxnorm = qxnorm [n_subframes–1];
```

Значение *rmc_max* и *mu_law* показаны в таблице 91.

Таблица 91 — Значение *rmc_max* и *mu_law*

Signal_mode	*rmc_max*	*mu_law*
0	7932	1024
1, 2, 3	15864	512

5.7.2.3.3 Декодирование вектора адаптивной книги шифров

Целую и дробную части задержки шага получают из *shape_delay*. таблицы отображения между *shape_delay* и двумя частями задержки шага показаны в таблице 92 для частоты дискретизации 8 кГц и в таблице 93 для частоты дискретизации 16 кГц. Вектор адаптивной книги шифров *acb[n]* вычисляется интер-

поляцией прошлого сигнала возбуждения *pacb[n]* в декодированной целой части задержки *acb_delay* и дробной части *acb_frac*. Интерполяция производится, используя *FIR* фильтр *int_fil[k]*, основанный на функции синхронизации с окном Хемминга. Если значение *shape_delay* равно 255 для частоты дискретизации 8 кГц или 511 для частоты дискретизации 16 кГц, выход *acb[]* состоит из отсчетов со всеми нулевыми значениями. Для других индексов выход *acb[]* вырабатывается следующей процедурой:

```
for (n = 0; n < sbfrm_size;)
{
  tt += acb_frac;
  kt = acb_delay + tt / 6;
  tt = tt % 6;
  for (i = 0; i < kt && n < sbfrm_size; i++, n++)
  {
    for ( k = –iftap; k <= iftap; k++)
    {
      kk = (k+1) * 6 – tt;
      acb[n] += int_fil[abs(kk)] * pacb[pacb_size–(kt–i+k+1)];
      pacb[pacb_size+n] = acb[n];
    }
  }
}
```

где параметры справки *iftap* и *pacb_size* зависят от частоты дискретизации (см. таблицы 92, 93, 94).

Таблица 92 — Отображение между *shape_delay* и задержкой шага для частоты дискретизации 8 кГц

shape_delay	*acb_delay*	*acb_frac*
0 – 161	*shape_delay*/3+17	(2**shape_delay*)%6
162 – 199	(*shape_delay*- 162)/2+71	(3*(*shape_delay*-162))%6
200 – 254	*shape_delay*-200+90	0
255	0	0

Таблица 93 — Отображение между *shape_delay* и задержкой шага для частоты дискретизации 16 кГц

shape_delay	*acb_delay*	*acb_frac*
0 – 215	*shape_delay*/3+20	(2**shape_delay*)%6
216 – 397	(*shape_delay*- 216)/2+92	(3*(*shape_delay*-216))%6
398 – 510	*shape_delay*-398+183	0
511	0	0

Таблица 94 — Число отводов фильра интерполяции и размер адаптивной книги шифров

Частота дискретизации	Число отводов	*pacb_size*
8 кГц	6	150
16 кГц	11	306

5.7.2.3.4 Декодирование вектора фиксированной книги шифров

Вектор фиксированной книги шифров содержит несколько ненулевых импульсов и представлен позицией импульса и амплитудами импульса. Позиции импульса *pul_pos[i]* извлекаются из *shape_positions*. Амплитуды импульса *pul_amp[i]* получают из *shape_signs*.

```
for (i = num_pulse–1, k = 0; i >= 0; i- -)
{
  for (j = 0; j < num_bit_pos[i]; j++)
  {
```

```
            pos_idx[i] |= ((shape_positions>>k) & 0x1) <<j;
            k++;
          }
          pul_amp[i] = 1.0;
          if (((shape_signs >> (num_pulse-1-i)) & 0x1) == 1)
          {
            pul_amp[i] = –1.0;
          }
          pul_pos[i] = pos_tbl[i][pos_idx[i]];
        }
```

где *num_pulse* — число импульсов, устанавливается из *MPE_Configuration* в зависимости от частоты дискретизации. *num_bit_pos[i]* — число битов для кодирования позиции *i*-го импульса. *pos_tbl[i][j]* является таблицей ограничения, которая указывает возможные позиции для каждого импульса. Таблица 95 указывает возможные позиции для каждого импульса. Таблица 96 устанавливается в соответствии с комбинацией *subfrm_size*, *num_pulse* и *num_bit_pos[]* следующим образом:

```
        step = subfrm_size / min_num_bit_pos;
        for (i = 0; i < num_pulse; i++)
        {
          m = 1 << (num_bit_pos[i]–min_num_bit_pos);
        for ( j = 0, k = 0; k < m)
        {
          ch[j] = i;
          k++;
          j += (long)((float)step/m + 0.5);
          j = j % step;
        }
      }
      for (i = 0; i < num_pulse; i++)
      {
        for (l = 0, k = 0; k < step; k++)
        {
          if (i == ch[k])
          {
            for (j = 0; j < min_num_bit_pos; j++)
            {
              pos_tbl[i][l++] = k + step * j;
            }
          }
        }
      }
```

Т а б л и ц а 95 — Число импульсов для частоты дискретизации 8 кГц

MPE_Configuration	*num_pulse*
0	3
1	4
2	5
3	5
4	6
5	7
6	6
7	7
8	8
9	9

MPE_Configuration	*num_pulse*
10	10
11	11
12	12
13	4
14	5
15	6
16	7
17	8
18	9
19	10

MPE_Configuration	*num_pulse*
20	11
21	12
22	8
23	9
24	10
25	11
26	12
27 ... 31	Зарезервировано

Таблица 96 — Число импульсов для частоты дискретизации 16 кГц

MPE_Configuration	*num_pulse*
0, 16	5
1, 17	6
2, 18	7
3, 19	8
4, 20	9
5, 21	10
6, 22	11
7, 23	Зарезервировано

MPE_Configuration	*num_pulse*
8, 24	3
9, 25	4
10, 26	5
11, 27	6
12, 28	7
13, 29	8
14, 30	9
15, 31	10

Позиции импульса, используемые в кодере режима II на 6 кбит/с для частоты дискретизации 8 кГц показаны в таблице 97.

Таблица 97 — Позиции импульса для кодера на 6 кбит/с

Pulse number: I	*Num_bit_pos[i]*	Позиции импульса: *pos_tbl[i][j]*
0	4	0,5,10,15,20,25,30,35, 40,45,50,55,60,65,70,75
1	4	1,6,11,16,21,26,31,36, 41,46,51,56,61,66,71,76
2	4	2,7,12,17,22,27,32,37 42,47,52,57,62,67,72,77
3	3	3,13,23,33,43,53,63,73
4	3	4,14,24,34,44,54,64,74
5	3	8,18,28,38,48,58,68,78
6	3	9,19,29,39,49,59,69,79

Вектор фиксированной книги шифров *fcb[n]* подсчитывается из *pul_pos[i]* и *pul_amp[i]* следующим образом:

```
for (n = 0; n < sbfrm_size; n++)
{
   fcb[n] = 0.0; }
for (i = 0; i < num_pulse; i++)
{
   fcb[pul_pos[i]] = pul_amp[i];
}
```

Если целочисленная задержка *acb_delay* меньше, чем размер подфрейма *sbfrm_size*, векторный сигнал фиксированной книги шифров *fcb[n]* измененяется гребенчатой фильтрацией *zero-state-combfiltering* следующим образом:

```
for (n = 0; n < sbfrm_size; n++)
{
   if (n - acb_delayt >= 0)
   {
      ix = fcb[n – acb_delay];
   }
   else
   {
      ix = 0.0;
```

```
    }
    fcb[n] += cga[signal_mode] * ix;
  }
```

где *cga* [4] = {0.0, 0.0, 0.6, 0.8} являются усилением гребенчатого фильтра.

5.7.2.3.5 Декодирование усилений адаптивной и фиксированной книг шифров

gain_index преобразуется в нормализованные усиления *nga, ngf* для векторов адаптивной и фиксированной книг шифров. Таблица усиления изменяется в соответствии с *signal_mode*, *sbfrm_size* и частотой дискретизации. Усиление адаптивной книги шифров *ga* и усиление фиксированной книги шифров *gf* вычисляются следующим образом:

*ga = nga * sqrt (norm / acb_energy);*

*gf = ngf * sqrt (norm / fcb_energy),*

где *acb_energy* и *fcb_energy* являются энергиями для векторов адаптивной книги шифров и фиксированной книги шифров соответственно. *norm* — коэффициент нормализации.

```
  norm = (qxnorm*subfrm_size)*(qxnorm*subfrm_size);
  for(i = 0; i < lpc_order; i++)
  {
    norm *= (1 – par[i] * par[i]);
  }
```

где *par[]* являются коэффициентами отражения и подсчитываются из *LP* коэффициентов *int_Qlpc_coefficients[]*. При этом *qxnorm* является квантованной энергией подфрейма, которая декодируется из *rmc_index* (см. 5.7.2.3.2).

5.7.2.3.6 Генерация сигнала возбуждения

Сигнал возбуждения (*excitation[]*) вычисляется суммированием *acb[]* и *fcb[]* масштабированных *ga* и *gf* соответственно.

```
  for (i = 0; i < sbfrm_size; i++)
  {
    excitation[i] = ga*acb[i] + gf*fcb[i];
  }
```

5.7.2.3.7 Обновление адаптивной книги шрифтов

Адаптивная книга шифров обновляется для процесса декодирования в следующем фрейме сгенерированным сигналом возбуждения *excitation[]* следующим образом:

```
  for (i = 0; i < pacb_size – sbfrm_size; i++)
  {
   pacb[i] = pacb[sbfrm_size+i];
  }
  for (i = 0; i < sbfrm_size; i++)
  {
   pacb[pacb_size–sbfrm_size+i] = excitation[i];
  }
```

5.7.3 Инструмент генерации мультиимпульсного возбуждения с масштабируемой битовой скоростью

5.7.3.1 Описание инструмента

Декодер масштабируемой битовой скорости реализуется, используя инструмент мультиимпульсного возбуждения с масштабируемой битовой скоростью, который состоит из инструмента мультиимпульсного возбуждения и инструмента декодирования возбуждения расширения. Это расширение масштабируемости разрешено только для кодера режима II. Сигнал возбуждения расширения восстанавливается путем поиска *shape_enh_positions*, *shape_enh_signs*, *gain_enh_index* и использования сигнала декодированного мультиимпульсного возбуждения в инструменте мультиимпульсного возбуждения.

5.7.3.2 Определения

Вход

shape_delay[]: Этот массив имеет размерность *nrof_subframes* и содержит задержку адаптивной книги шифров

shape_positions[]: Этот массив имеет размерность *nrof_subframes* и содержит позиции импульса

shape_signs[]: Этот массив имеет размерность *nrof_subframes* и содержит знаки импульса

shape_enh_positions []: Этот массив имеет размерность *nrof_subframes* и содержит позиции импульса

shape_enh_signs[]: Этот массив имеет размерность *nrof_subframes* и содержит знаки импульса

gain_enh_index[]: Этот массив имеет размерность *nrof_subframes* и содержит индекс усиления адаптивной книги шифров и индекс усиления фиксированной книги шифров

int_Qlpc_coefficients[]: Это массив размерности *lpc_order* содержит квантованные и интерполированные коэффициенты *LPC* одного подфрейма

Выход

enh_excitation[]: Этот массив имеет размерность *sbfrm_size* и содержит сигнал возбуждения расширения. Этот сигнал реконструируется из векторов формы и усиления, используя фиксированную книгу шифров.

Конфигурация

lpc_order: Это поле указывает порядок *LPC*, который используется

sbfrm_size: Это поле указывает число отсчетов в подфрейме

nrof_subframes: Это поле указывает число подфреймов

Дополнительные элементы, используемые в инструменте *MPE* с масштабируемой битовой скоростью, следующие:

pos_tbl[][]	таблица поиска возбуждения для непериодического компонента
enh_fcb[]	декодированный сигнал возбуждения как непериодический компонент
ge	декодированное усиление для непериодического компонента
enh_fcb_energy	энергия *enh_fcb[]*

5.7.3.3 Процесс декодирования

5.7.3.3.1 Декодирование вектора фиксированной книги шифров расширения

Вектор фиксированной книги шифров расширения также состоит из нескольких ненулевых импульсов. Позиции импульса и амплитуды извлекаются из *shape_enh_positions*, *shape_enh_signs* тем же алгоритмом декодирования, что из фиксированной книги шифров. Векторы фиксированной книги шифров расширения *enh_fcb[n]* вычисляются из *pul_pos [i]* и *pul_amp [i]* следующим образом:

```
for (i = num_pulse_enh–1, k = 0; i >= 0; i- -)
{
   for (j = 0; j < num_bit_pos[i]; j++) {
      pos_idx[i] |= ((shape_enh_positions>>k)&0x1) <<j;
      k++;
   }
   pul_amp[i] = 1.0;
   if (((shape_enh_signs >> (num_pulse_enh–1–i)) & 0x1) == 1)
   {
      pul_amp[i] = –1.0;
   }
   pul_pos[i] = pos_tbl[i][pos_idx[i]];
}

for (n = 0; n < sbfrm_size; n++)
{
   enh_fcb[n] = 0.0;
}
for (i = 0; i < num_pulse_enh; i++)
{
   enh_fcb[pul_pos[i]] = pul_amp[i];
}
```

Таблица 98 — Определение *num_pulse_enh*

sbfrm_size	*num_pulse_enh*
40	2
80	4

Таблица 98 позиций импульсов временно генерируется тем же самым алгоритмом декодирования, что и фиксированная книга шифров. Временная таблица позиций импульса изменяется следующим образом:

```
for (n = 0; n < num_enh; n++)
{
   for (i = num[n]–1, k = 0; i >= 0; i- -)
   {
      pul_loc = 0;
      for (j = 0; j < bit_pos[i]; j++)
   {
      pul_loc |= ((idx[n]>>k)&0x1)<<j;
      k++;
   }
   pul_loc = chn_pos[i*len+pul_loc];
   for (l = 0; l < 10; l++)
   {
      for (m = 0; m < (1 << bit_pos_org[l]); m++)
      {
        if (pul_loc == chn_pos_org[l*len+m])
          chn_ctr[l]++;
          break;
        }
      }
   }
}

for (i = 0; i < 10; i++)
{
   ctr_tmp[i] = chn_ctr[i];
}
for (i = 0; i < num[n+1]; i++)
{
   min_ctr = len;
   for (j = 0; j < 10; j++)
   {
      if (ctr_tmp[j] < min_ctr)
      {
        min_ctr = ctr_tmp[j];
        min_chn = j;
      }
   }
   ctr_tmp[min_chn] = len;
   bit_pos[i] = bit_pos_org[min_chn];
   for (j = 0; j < (1<<bit_pos_org[min_chn]); j++)
   {
       chn_pos[i*len+j] = chn_pos_org[min_chn*len+j];
      }
   }
}

for (i = 0; i < num[num_enh]; i++)
{
   bit[i] = bit_pos[i];
   for (j = 0; j < (1<<bit[i]); j++)
   {
```

```
        pos_tbl[i*len+j] = chn_pos[i*len+j];
    }
}
```

Позиции импульса, используемые в инструменте возбуждения расширения, показаны в таблице 99 и таблице 100. Таблица изменяется в зависимости от длины подфрейма.

Таблица 99 — Временная таблица позиций импульса для подфрейма с 80 отсчетами

Номер импульса: *i*	*bit_pos_org[i]*	Позиции импульса: *chn_pos_org[i][j]*
0	3	0,10,20,30,40,50,60,70
1	3	1,11,21,31,41,51,61,71
2	3	2,12,22,32,42,52,62,72
3	3	3,13,23,33,43,53,63,73
4	3	4,14,24,34,44,54,64,74
5	3	5,15,25,35,45,55,65,75
6	3	6,16,26,36,46,56,66,76
7	3	7,17,27,37,47,57,67,77
8	3	8,18,28,38,48,58,68,78
9	3	9,19,29,39,49,59,69,79

Таблица 100 — Временная таблица позиций импульса для подфрейма с 40 отсчетами

Номер импульса: *i*	*bit_pos_org[i]*	Позиции импульса: *chn_pos_org[i][j]*
0	2	0,10,20,30
1	2	1,11,21,31
2	2	2,12,22,32
3	2	3,13,23,33
4	2	4,14,24,34
5	2	5,15,25,35
6	2	6,16,26,36
7	2	7,17,27,37
8	2	8,18,28,38
9	2	9,19,29,39

5.7.3.3.2 Декодирование усиления фиксированной книги шифров расширения

gain_enh_index преобразуется в нормализованное усиление *nge* для вектора фиксированной книги шифров расширения путем просмотра таблицы усиления. Таблица усиления меняется в соответствии с *signal_mode*. Усиление фиксированной книги шифров расширения *ge* вычисляется следующим образом:

*ge = nge * sqrt (norm / enh_fcb_energy)*;

где *enh_fcb_energy* — энергия для вектора фиксированной книги шифров расширения. *norm* — коэффициент нормализации.

5.7.3.3.3 Генерация расширенного сигнала возбуждения

Расширенный сигнал возбуждения (*enh_excitation[]*) вычисляется, добавляя вектор книги шифров расширения к сигналам возбуждения.

```
for (i = 0; i < sbfrm_size; i++)
{
  enh_excitation[i] = excitation[i] + ge*enh_fcb[i];
}
```

5.7.4 Инструмент генерации мультиимпульсного возбуждения с масштабируемой полосой пропускания

5.7.4.1 Описание инструмента

Процесс декодирования возбуждения в декодере с масштабируемой полосой пропускания достигается с помощью инструмента мультиимпульсного возбуждения с масштабируемой полосой пропускания, который состоит из инструмента мультиимпульсного возбуждения и инструмента расширения полосы пропускания. Это расширение масштабируемости позволено только для кодера режима II. Сигнал масштабируемого возбуждения восстанавливается поиском *shape_bws_positions*, *shape_bws_ signs* и *gain_bws_index* и использованием декодированных выходов для частоты дискретизации 8 кГц, а именно, целочисленной и дробной части задержки шага и вектора фиксированной книги шифров. Эти выходы генерируются инструментом декодирования *MPE* (5.7.2) или инструментом декодирования *MPE* с масштабируемой битовой скоростью (5.7.3).

5.7.4.2 Определения

Вход

shape_bws_delay[]: Этот массив имеет размерность *nrof_subframes_bws* и содержит задержку адаптивной книги шифров

shape_bws_positions[]: Этот массив имеет размерность *nrof_subframes_bws* и содержит позиции импульса

shape_bws_signs[]: Этот массив имеет размерность *nrof_subframes_bws* и содержит знаки импульса

gain_bws_index[]: Этот массив имеет размерность *nrof_subframes_bws* и содержит индекс усиления адаптивной книги шифров и индекс усиления фиксированной книги шифров

rмс_index: Это поле определяет индекс для мощности сигнала

signal_mode: Это поле содержит флажок голосовой/неголосовой

int_Qlpc_coefficients[]: Это массив размерности *lpc_order* содержит коэффициенты квантованного и интерполированного *LPC* одного подфрейма

Выход

excitation[]: Этот массив имеет размерность *sbfrm_size* и содержит сигнал возбуждения. Этот сигнал восстановлен из векторов формы и усиления, используя адаптивные и фиксированные книги шифров

acb_delay: Это поле содержит декодированную задержку для адаптивной книги шифров

adaptive_gain: Это поле содержит декодированное усиление для адаптивной книги шифров

Конфигурация

llpc_order: Это поле указывает порядок *LPC*, который используется

sbfrm_size: Это поле указывает число отсчетов в подфрейме в инструменте расширения полосы пропускания

nrof_subframes_bws: Это поле указывает число подфреймов в инструменте расширения полосы пропускания

Дополнительные элементы, используемые в инструменте *MPE* с масштабируемой полосой пропускания, следующие:

pacb[]	таблица поиска возбуждения для периодического компонента
pos_tbl[][]	таблица поиска возбуждения для непериодического компонента
acb[]	декодированный сигнал возбуждения как периодический компонент
fcb1[]	декодированный сигнал возбуждения как непериодический компонент
fcb2[]	декодированный сигнал возбуждения как непериодический компонент
ga	декодированное усиление для периодического компонента
gn	декодированное усиление для непериодического компонента
gf	декодированное усиление для непериодического компонента
qxnorm[]	декодированные среднеквадратические значения речевого сигнала
par[]	коэффициенты отражения, конвертированные из *int_Qlpc_ coefficients []*
acb_energy	энергия *acb[]*
fcb1_energy	энергия *fcb1[]*
fcb2_energy	энергия *fcb2[]*
acb_delay_wb	целая часть задержки шага
acb_frac_wb	дробная часть задержки шага

5.7.4.3 Процесс декодирования

Для декодера режима II с масштабируемостью полосы пропускания сигнал возбуждения при частоте выборки 16 кГц создается из периодического компонента (вектор адаптивный книги шифров) и двух непериодических компонентов (вектор фиксированной книги шифров 1 и 2) масштабированных соответствующими усилениями.

5.7.4.3.1 Декодирование *signal_mode*

Signal_mode также используется в декодировании энергии фрейма, мультиимпульсного возбуждения и усилений в этом процессе декодирования.

5.7.4.3.2 Энергия декодирования фрейма

Процедура декодирования является такой же, как инструмент декодирования *MPE*.

5.7.4.3.3 Декодирование вектора адаптивной книги шифров

Целую и дробную части задержки шага получают из *shape_delay* и *shape_bws_delay*.

acb_delay и *acb_frac* в частоте дискретизации 8 кГц декодируются в инструменте *MPE* и подаются на инструмент расширения полосы пропускания для сигнала возбуждения. Параметры частоты дискретизации 8 кГц конвертируются в параметры частоты дискретизации 16 кГц *acb_delay_wb*, *acb_frac_wb* в частоте дискретизации 16 кГц следующим образом:

```
op_delay_wb = 2 * acb_delay
if (acb_frac != 0)
{
   op_delay_wb++;
}
if (op_delay_ wb == 0)
{
   op_idx_wb = 778;
}
else
{
   op_idx_wb = (op_delay_wb – 32) * 3 + 2;
}
st_idx_wb = op_idx_wb – 4;
if (st_idx_wb < 0)
{
   st_idx_wb = 0;
}
if ((st_idx_wb + 7) >= 778)
{
   st_idx_wb = 778 – 8;
}
if (op_idx_wb == 778)
{
   acb_idx_wb = 778;
}
else
{
   acb_idx_wb = st_idx_wb + shape_bws_delay;
}
```

Отображение между *acb_idx_wb* и параметрами задержки шага *acb_delay_wb*, *acb_frac_wb* показано в таблице 101. Вектор адаптивной книги шифров *acb[n]* вычисляется, интерполируя прошлый сигнал возбуждения *pacb[n]* при декодированной целочисленной задержке *acb_delay_wb* и дробной *acb_frac_wb*. Интерполяция выполняется, используя фильтр *FIR int_fil[k], k=0..., 66*, основанный на функции синхронизации с окном Хемминга. Если значение *shape_delay* равно 255 или значение *shape_bws_delay* равно 768, выход *acb[]* состоит из всех нулевых отсчетов. Для других комбинаций индексов выход *acb[]* получают следующей процедурой:

```
for (n = 0; n < sbfrm_size;)
{
```

```
      tt += acb_frac_wb;
      kt = acb_delay_wb + tt / 6;
      tt = tt % 6;
    for (i = 0; i < kt && n < sbfrm_size; i++, n++)
    {
      for (k = –11; k <= 11; k++ )
      {
        kk = (k+1) * 6 – tt;
        acb[n] += int_fil [abs(kk)] * pacb [306-(kt-i+k)];
        pacb[306+n] = acb[n];
      }
    }
  }
```

Таблица 101 — Отображение между *shape_delay* и задержкой шага

acb_idx_wb	*acb_delay_wb*	*acb_frac_wb*
0, 1	32	(2*(*acb_idx_wb*+1)%6
2 – 777	32+2*(*acb_idx_wb*–2)/6	(2*(*acb_idx_wb*–2))%6
778	0	0

5.7.4.3.4 Декодирование вектора фиксированной книги шифров 1

Вектор фиксированной книги шифров 1 *fcb1*[*n*] получен преобразованием частоты дискретизации вектора фиксированной книги шифров *nb_fcb*[*n*], используемой в инструменте *MPE* или инструменте *MPE* с масштабируемой битовой скоростью следующим образом:

```
for (n = 0; n < sbfrm_size/2; n++)
{
  fcb1[2*n] = nb_fcb[n];
  fcb1[2*n+1] = 0;
}
```

5.7.4.3.5 Декодирование вектора фиксированной книги шифров 2

Вектор фиксированной книги шифров 2 содержит несколько ненулевых импульсов и представлен позицией импульса и амплитудами импульса. Позиции импульса *pul_pos[i]* получают из *shape_bws_positions*. Амплитуды импульса *pul_amp[i]* получают из *shape_bws_signs*.

```
for (i = num_pulse_bws – 1, k = 0; i >= 0; i- -)
{
  for (j = 0; j < num_bit_pos[i]; j++)
  {
    pos_idx[i] |= ((shape_bws_positions >> k) & 0x1) <<j;
    k++;
  }
  pul_amp[i] = 1.0;
  if (((shape_bws_signs >> (num_pulse_bws–1–i)) & 0x1) == 1)
  {
    pul_amp[i] = – 1.0;
  }
  pul_pos[i] = pos_tbl[i][pos_idx[i]];
}
```

где *num_pulse_bws* — число импульсов равно одному из вариантов 6, 8, 10, 12. Выбор зависит от *BWS_configuration*. *num_bit_pos[i]* является числом битов для кодирования позиции *i*-го импульса. *pos_tbl[i][j]* является таблицей ограничения, которая указывает возможные позиции для каждого импульса. Таблица 102 указывает возможные позиции для каждого импульса.

Таблица 102 — Определение *num_pulse_bws*

BWS_configuration	*num_pulse_bws*
0	6
1	8
2	10
3	12

Таблицы позиции импульса для каждого числа импульсов также устанавливаются в соответствии с комбинацией *subfrm_size*, *num_pulse* и *num_bit_pos[]* той же самой процедурой, что инструмент *MPE*.

Вектор фиксированной книги шифров *fcb2[n]* вычисляется из *pul_pos[i]* и *pul_amp[i]* следующим образом:

```
for (n = 0; n < sbfrm_size; n++)
{
  fcb2[n] = 0.0;
}
for (i = 0; i < num_pulse_bws; i++)

  fcb2[pul_pos[i]] = pul_amp[i];
}
```

Если целочисленная задержка *acb_delay_wb* меньше, чем размер подфрейма *sbrm_size*, сигнал вектора фиксированной книги шифров *fcb2[n]* изменяется фильтрацией *zero-state-comb* следующим образом:

```
for (n = 0; n < sbfrm_size; n++)
{
  if (n – acb_delayt >= 0)
  {
    ix = fcb2[n – acb_delay];
  }
  else
  {
    ix = 0.0;
  }
  fcb2[n] += cga[signal_mode] * ix;
}
```

где *cga* [*4*] = {0.0 , 0.0 , 0.6 , 0.8} являются усилениями фильтра гребенки *comb-filter*.

5.7.4.3.6 Декодирование усилений адаптивной и фиксированной книг шифров

gain_bws_index преобразуется в два индекса.

```
qga_idx = gain_bws_index >> 7;
qgc_idx = gain_bws_index – (gain_bws_index << 7);
```

qga_idx преобразуется к нормализованным усилениям *nga*, *ngf* для векора 2 адаптивной и фиксированной книг шифров путем просмотра таблицы усиления. *qgc_idx* преобразуется в нормализованноое усиление *ngn* для вектора 1 фиксированной книги шифров просмотром таблицы усиления. Таблица усиления изменяется в соответствии с *signal_mode*. Усиление адаптивной книги шифров *ga*, усиление фиксированной книга шифров 1 *gn* и усиление фиксированной книги шифров 2 *gf* вычисляются следующим образом:

```
ga = nga * sqrt( norm / acb_energy );
gn = ngn * sqrt( norm / fcb1_energy );
gf = ngf * sqrt( norm / fcb2_energy );
```

где *acb_energy*, *fcb2_energy* и *fcb1_energy* являются энергиями для векторов адаптивной книги шифров и двух фиксированных книг шифров. *norm* — коэффициент нормализации.

```
norm = (qxnorm*subfrm_size) * (qxnorm*subfrm_size);
for (i = 0; i < lpc_order; i++)
{
  norm *= (1 – par[i] * par[i]);
}
```

где *par[]* являются коэффициентами отражения и вычисляются из коэффициентов *LP int_ Qlpc_coefficients[]*. *qxnorm* — квантованная энергия подфрейма, декодируется из *гмс_indx*.

5.7.4.3.7 Генерация сигнала возбуждения

Сигнал возбуждения (*excitation[]*) подсчитывается, суммируя *acb[]*, *fcb1[]* и *fcb2[n]* масштабированные *ga*, *gn* и *gf* соответственно.

```
for (i = 0; i < sbfrm_size; i++)
{
  excitation[i] = ga*acb[i] + gn*fcb1[i] + gf*fcb2[i];
}
```

5.7.4.3.8 Обновление адаптивной книги шифров

Адаптивная книга шифров обновляется для процесса декодирования в следующем фрейме генерированным сигналом возбуждения *excitation[]* следующим образом:

```
for (i = 0; i < 306–sbfrm_size; i++)
{
  pacb[i] = pacb[sbfrm_size+i];
}
for (i = 0; i < sbfrm_size; i++)
{
  pacb[306–sbfrm_size+i] = excitation[i];
}
```

5.8 Фильтр синтеза *CELP LPC*

5.8.1 Описание инструмента

Фильтр синтеза *CELP LPC* создает синтезируемый сигнал из коэффициентов *LPC* и сигнала возбуждения для каждого подфрейма.

5.8.2 Определения

Вход

excitation[]: Этот массив содержит сигнал возбуждения для одного подфрейма

int_Qlpc_coefficients[]: Этот массив размерности *lpc_order* содержит квантованные и интерполированные коэффициенты *LPC*.

Выход

synth_signal[]: Сигнал возбуждения, *exitation []* подается через фильтр синтеза, используя коэффициенты *LPC* из *int_Qlpc_coefficients*[]. Размерность этого массива равна *lpc_order*.

Конфигурация

lpc_order: Это поле содержит порядок используемого *LPC*

sbfrm_size: Это поле содержит число отсчетов в подфрейме

5.8.3 Процесс декодирования

Используя интерполированные коэффициенты *LPC* одного подфрейма, сигнал возбуждения подается через следующий фильтр

$$H_s(z) = \frac{1}{\hat{A}(z)} = \frac{1}{1 - \sum_{k=1}^{lpc_order} \hat{a}_k \cdot z^{-k}},$$

где $\hat{A}(z)$ — фильтр инверсии *LPC*, использующий квантованные коэффициенты *LPC*. Коэффициент $\hat{a}_k$ является *k*-ым коэффициентом *LPC (int_Qlpc_coefficients[k-1])*. Вывод фильтра инверсии является реконструированной речью. Прядок *LPC* устанавливается в 10 и 20 для частоты дискретизации 8 кГц и 16 кГц соответственно.

Следующий алгоритм представляет собой реализацию вышеупомянутого фильтра:

```
for (n = 0; n < sbfrm_size; n++)
{
  tmp = excitation[n];
  for (k = 0; k < lpc_order; k++)
  {
    tmp = tmp + Filter_states[k] * int_Qlpc_coefficients[k];
  {
```

```
    synth_signal[n] = tmp;
    for (k = lpc_order–1; k > 0; k- -)
    {
      Filter_states[k] = Filter_states[k-1];
    }
      Filter_states[0] = synth_signal[n];
}
```

Массив *Filter_states* первоначально установлен в нуль.

5.9 Инструмент сжатия тишины *CELP*

5.9.1 Описание инструмента

Инструмент сжатия тишины содержит модуль *voice activity detection* (*VAD*), блок *discontinuous transmission* (*DTX*) и модуль *comfort noise generator* (*CNG*). Инструмент кодирует/декодирует входной сигнал на нижних битовых скоростях во время неактивных (тишина) фреймов. Во время вокально активных (речевых) фреймов используются, кодирование *MPEG-4 CELP* и декодирование.

На стороне передачи модуль *DTX* кодирует входную речь во время неактивных фреймов. Во время разговорных фреймов используется кодер *MPEG-4 CELP*. Флажок речевой активности (*VAD_flag*), указывающий неактивный фрейм (*VAD_flag*=0) или речевой фрейм (*VAD_flag*=1), определяется исходя из входной речи модулем *VAD*. Во время неактивных фреймов модуль *DTX* обнаруживает фреймы, где входные характеристики изменяются (*DTX_flag*=1 и 2: Изменение, *DTX_flag*=0: Никакого изменения). Когда обнаружено изменение, модуль *DTX* кодирует входную речь, чтобы генерировать информацию *SID* (описатель вставки тишины). *VAD_flag* и *DTX_flag* передаются совместно на декодер как флаг *TX_flag*, чтобы сохранить синхронизацию между кодером и декодером.

На стороне приема модуль *CNG* генерирует комфортный шум, основанный на информации *SID*, во время неактивных фреймов. Во время фреймов речевой активности вместо этого используется декодер *MPEG-4 CELP*. Модуль *CNG* или декодер *MPEG-4 CELP* выбираются согласно флажку *TX_flag*.

Информация *SID* и *TX_flag* передаются, только когда обнаружено изменение входных характеристик. Иначе во время неактивных фреймов передается только *TX_flag*.

5.9.2 Описания

CNG: генератор комфортного шума

Coding mode: "I" для *RPE* и "II" для *MPE* (см. таблицу 1)

DTX: прерывающаяся передача

LP: линейное прогнозирование

LPCs: коэффициенты LP

MPE: мультиимпульсное возбуждение

MPE_Configuration: см. раздел 4

RMS: среднеквадратичный

RPE: возбуждение регулярным импульсом

RPE_Configuration: см. раздел 4

SID: описатель вставки тишины

SID frame: фрейм, в котором передается/принимается информация *SID*

signal_mode: режим, определяющийся на основе среднего усиления предсказания шага (см. раздел 4)

VAD: опознавание речевой активности

5.9.3 Процесс декодирования

5.9.3.1 Полезная нагрузка передачи

Имеется четыре типа полезных нагрузок передачи в зависимости от решения *VAD/DTX*. *TX_flag* указывает тип полезной нагрузки передачи и определяется флажками *VAD_flag* и *DTX_flag*, как показано в таблице 103. Когда *TX_flag* указывает речевой фрейм (*TX_flag* = 1), передаются информация, сгенерированная кодером *MPEG-4 CELP*, и флаг *TX_flag*. Когда *TX_flag* указывает фрейм перехода между речевым фреймом и неактивным фреймом или неактивный фрейм, в котором меняются спектральные характеристики входного сигнала (*TX_flag* = 2), передаются информация *High-Rate* (*HR*) *SID* и флаг *TX_flag*, чтобы обновить параметры *CNG*. Когда *TX_flag* указывает неактивный фрейм, в котором мощность фрейма входного сигнала измененяется (*TX_flag* = 3), передаются информация *Low-Rate* (*LR*) *SID* и *TX_flag*. Другие неактивные фреймы категоризируются в четвертый тип *TX_flag* (*TX_flag* = 0). В этом случае передается только *TX_flag*. Примеры изменения *TX_flag* согласно *VAD_flag* и *DTX_flag* показаны в таблице 104.

Таблица 103 — Соотношение между флажками для инструмента сжатия тишины

Флаги	Речевая активность	Неактивность		
VAD_flag	1	0		
DTX_flag	—	0	1	2
TX_flag	1	0	2	3

Таблица 104 — Примеры изменения *TX_flag* согласно *VAD_flag* и *DTX_flag*

Фрейм #	...k-5	k-4	k-3	k-2	k-1	k	k+1	k+2	k+3	k+4	k+5	k+6	k+7
VAD_flag	...1	1	1	1	0	0	0	0	0	0	0	0	0...
DTX_flag	...—	—	—	—	1	0	0	0	1	0	0	0	2...
TX_flag	...1	1	1	1	2	0	0	0	2	0	0	0	3...
	активный период разговора				активный период разговора								

5.9.3.2 Передача *LSP*

В случае, когда инструмент сжатия тишины используется в комбинации с запущенным *FineRate Control*, фрейм *CELP* с *LPC_present* = 1 и *interpolation_flag* = 0 должен быть передан в первом фрейме с речевой активностью после фрейма неактивной речи. Фреймы с речевой активностью отмечаются *TX_flag* = 1, фреймы с неактивной речью отмечаются *TX_flag* = 0, 2 или 3.

5.9.3.3 Модуль *CNG*

Комфортный шум генерируется путем фильтрации возбуждения с помощью фильтра синтеза *LP*, аналогичным способом как сигналы вокализованной речи. Чтобы улучшить качество кодирования, может использоваться постфильтр. Возбуждение дается, добавляя мультиимпульсное возбуждение или возбуждение регулярным импульсом и случайное возбуждение, масштабированные их соответствующими усилениями. Возбуждения генерируются на базе случайной последовательности, независимой от информации *SID*. Коэффициенты для фильтра синтеза *LP* и усилений вычисляются из *LSPs* и значения *RMC* (энергия фрейма), соответственно, которые получены как информация *SID*. *LSPs* и *RMC* сглажены, чтобы улучшить качество кодирования для речи с шумящим входом. Модуль *CNG* использует те же самые размеры фрейма и подфрейма, как в активных речевых фреймах. Обработка в каждой части описана в следующих подпунктах.

5.9.3.3.1 Определения

Вход

TX_flag: Это поле содержит режим передачи

SID_lpc_indices[]: Этот массив содержит упакованные индексы *LP*. Размерность 3, 5 или 6 (см. таблицу 72)

SID_rmc_index: Это поле содержит индекс *RMC*

Выход

PP_synth_signal[]: Этот массив содержит постфильтрованный (улучшенный) речевой сигнал. Размерность — *sbfrm_size*. Элементы справки, используемые в модуле *CNG*:

lpc_order: порядок *LP*

sbfrm_size: число отсчетов в подфрейме

n_subframe: число подфреймов во фрейме

int_Q_lpc_coefficients []: интерполированные *LPCs* (см. 5.8.2).

5.9.3.3.2 Декодер *LSP*

LSP lpc_current[] декодированные из *LSP* индексы *SID_lpc_indices[]*. Процесс декодирования идентичен описанному в 5.6 со следующими исключениями:

1) Подмножество *lpc_indices[]* передается на декодер. Соотношение между передаваемыми индексами *LSP, SID_lpc_indices[]* и индексами *LSP* для *MPEG-4 CELP, lpc_indices[]*, показано в таблице 105.

2) Процесс декодирования для непереданных индексов не выполняется.

Таблица 105 — Соотношение индекса *LSP* между инструментом сжатия тишины и *MPEG-4 CELP*

Режим кодирования	Частота дискретизации, кГц	Масштабируемость полосы	Инструмент сжатия тишины	*MPEG-4 CELP*
I (*RPE*)	16	*Off*	*SID_lpc_indices[0]*	*lpc_indices[0]*
			SID_lpc_indices[1]	*lpc_indices[1]*
			SID_lpc_indices[2]	*lpc_indices[2]*
			SID_lpc_indices[3]	*lpc_indices[3]*
			SID_lpc_indices[4]	*lpc_indices[5]*
			SID_lpc_indices[5]	*lpc_indices[6]*
II (*MPE*)	8	*On, Off*	*SID_lpc_indices[0]*	*lpc_indices[0]*
			SID_lpc_indices[1]	*lpc_indices[1]*
			SID_lpc_indices[2]	*lpc_indices[2]*
	16	*On*	*SID_lpc_indices[3]*	*lpc_indices[5]*
			SID_lpc_indices[4]	*lpc_indices[6]*
			SID_lpc_indices[5]	*lpc_indices[7]*
			SID_lpc_indices[6]	*lpc_indices[8]*
		Off	*SID_lpc_indices[0]*	*lpc_indices[0]*
			SID_lpc_indices[1]	*lpc_indices[1]*
			SID_lpc_indices[2]	*lpc_indices[2]*
			SID_lpc_indices[3]	*lpc_indices[3]*
			SID_lpc_indices[4]	*lpc_indices[5]*
			SID_lpc_indices[5]	*lpc_indices[6]*

5.9.3.3.3 Сглаживатель *LSP*

Сглаженные *LSPs, lsp_current_sm*[] обновляются, используя декодированные *LSPs lsp_current*[] в каждом фрейме как:

$$lsp_current_sm[i] = \begin{cases} 0{,}875\, lsp_current_sm[i] + 0{,}125\, lsp_current_sm[i], \; TX_flag = 2 \\ 0{,}875\, lsp_current_sm[i] + 0{,}125\, lsp_current_sid[i], \; TX_flag = 0 \; or \; 3, \end{cases}$$

где *i*=0..., *lpc_order-1* и *lsp_current_sid[]* являются *lsp_current[]* в последнем фрейме *SID*. В начале каждого неактивного периода *lsp_current_sm[]* инициализируется *lsp_current[]*, в конце предыдущего речевого периода.

5.9.3.3.4 Интерполяция *LSP* и преобрзование *LSP-LPC*

LPCs для синтеза *LP int_Q_lpc_coefficients*[] подсчитываются из сглаженных *LSPs lpc_current_sm*[], используя интерполяцию *LSP* со стабилизацией и преобразование *LSP* в *LPC*. Эти процессы описаны в 5.6. Общий буфер для предыдущего фрейма *lsp_previous*[] используется и в инструменте сжатия тишины, и в кодировании *CELP* для фреймов с речевой активностью.

5.9.3.3.5 Декодер *RMC*

RMC входной речи *qxnorm* в каждом подфрейме восстанавливается, используя *SID_rms_index* в том же процессе, как описано в 5.7.2.3.2 за исключением того, что параметры μ-закона независимы от режима сигнала и установлены, как *rms_max*=7932 и *mu_law* = 1024.

Востановленное *RMC* входной речи преобразуется в *RMC* сигнала возбуждения (*norm*), используя коэффициенты отражения *par*[] следующим образом и используется для вычисления усиления:

```
norm = (qxnorm*subfrm_size)*(qxnorm*subfrm_size);
for (i = 0; i < lpc_order; i++)
{
```

```
    norm *= (1 – par[i] * par[i])* αs;
  }
```

где *par*[] вычисляется из *LPCs int_Qlpc_coefficients*[], и коэффициент масштабирования α_s равен 0,8.

5.9.3.3.6 Сглаживатель *RMC*

Сглаженное *RMC norm_sm* обновляется, используя norm в каждом подфрейме следующим образом:

$$norm_sm = \begin{cases} 0{,}875\, norm_sm + 0{,}125\, norm[subnum] & for\ TX_flag = 2\ or\ 3 \\ 0{,}875\, norm_sm + 0{,}125\, norm_sid & for\ TX_flag = 0\ , \end{cases}$$

где *subnum* номер текущего подфрейма в диапазоне от 0 до *n_subframe*-1 и *norm_sid* является *norm[n_subframe*- 1] в последнем фрейме *SID*. В первом фрейме каждого неактивного периода *norm_sm* устанавливается в *norm*. Во время первых 40 мс неактивного периода *norm_sm* инициализируется *norm[subnum]*, когда *TX_flag* = 2 или 3 и

$$\left| 20\log_{10} norm_sid - 20\log_{10} norm[n_subframe - 1] \right| > 6db$$

5.9.3.3.7 Генерация возбуждения *CNG*

Сигнал возбуждения *CNG excitation[]* подсчитывается из сигнала мультиимпульсного возбуждения и случайного сигнала возбуждения следующим образом:

```
  for (i = 0; i < sbfrm_size; i++)
  {
    excitation[i] = gf * fcb_cng[i] + gr * excg[i];
  }
```

где *fcb_cng*[] и *excg*[] являются, соответственно, сигналом мультиимпульсного (*MP*) возбуждения или возбуждения регулярным импульсом (*RP*) и случайным сигналом возбуждения. *gf* и *gr* являются их соответствующими усилениями. Случайный сигнал возбуждения, позиции и знаки импульсов для возбуждения *MP/RP* последовательно производятся из случайной последовательности в каждом подфрейме. Чтобы синхронизировать генератор случайных чисел *CNG* в кодере и декодере, случайный сигнал возбуждения *excg*[] для данного подфрейма должен быть вычислен до генерации возбуждения *MP/RP fcb_cng*[] для этого подфрейма.

5.9.3.3.7.1 Случайная последовательность

Случайная последовательность генерируется следующей функцией и используется для генерации сигналов мультиимпульсного и случайного возбуждения:

```
  short Random (*seed)
  {
    *seed = (short) ((int)(*seed * 31821 + 13849));
    return (*seed);
  }
```

со значением начального числа 21845 и обычно используемого для обоих возбуждений. Этот генератор имеет периодический цикл 16 бит. Начальное число инициализируется значением 21845 в начале каждого неактивного периода.

5.9.3.3.7.2 Генерация случайного возбуждения

Сигнал случайного возбуждения каждого подфрейма является Гауссовской случайной последовательностью, которая генерируется следующим образом:

```
  for (i = 0; i < sbfrm_size; i++)
  {
    excg[i] = Gauss (seed);
  }
где
  float Gauss (short *seed)
  {
    temp = 0;
    for (i = 0; i < 12; i++)
    {
      temp += (float) Random (seed);
    }
```

```
    temp /= (2 * 32768);
    return (temp);
}
```

5.9.3.3.7.3 Генерация мультиимпульсного возбуждения

В случае, когда для кодирования голосовых фреймов используется *MPE*, сигнал мультиимпульсного возбуждения генерируется для каждого подфрейма случайным выбором позиций и знаков импульсов. Мультиимпульсные структуры *MPEG-4* версии 1 *CELP* с *MPE_Configuration*=24 и 31 используются для частоты дискретизации 8 и 16 кбит/с соответственно. Позиции и знаки 10 импульсов генерируются в векторе с 40 отсчетами. Для размера подфрейма 80 отсчетов дважды генерируется 20 импульсов в векторе с 80 отсчетами. Индексы позиций и знаков *mp_pos_idx* и *mp_sign_idx* генерируются в каждом подфрейме следующим образом:

```
if (subframe size is 40 samples)
{
  setRandomBits (&mp_pos_idx, 20, seed);
  setRandomBits
(&mp_sgn_idx, 10, seed);
}
if (subframe size is 80 samples)
{
  setRandomBits (&mp_pos_idx_1st_half, 20, seed);
  setRandomBits (&mp_sgn_idx_1st_half, 10, seed);
  setRandomBits (&mp_pos_idx_2nd_half, 20, seed);
  setRandomBits (&mp_sgn_idx_2nd_half, 10, seed);
}
```

где *mp_pos_idx_1st_half*, и *mp_sgn_idx_1st_half* — индексы позиций и знаков первой половины подфрейма и *mp_pos_idx_2nd_half*, и 20 *mp_sgn_idx_2nd_half* — индексы для второй половины. Функция *setRandomBits*() определяется в 5.9.3.3.7.5.

5.9.3.3.7.4 Возбуждение регулярным импульсом

В случае, когда *RPE* используется для кодирования речевых фреймов, сигнал возбуждения регулярным импульсом генерируется для каждого подфрейма. В случае неактивных фреймов контент адаптивной книги шифров инициализируется нулем. Для неактивных фреймов используется только фиксированная книга шифров. Сигнал возбуждения фиксированной книги шифров генерируется, используя случайный *shape_index*, как вход для процесса декодирования *RPE*.

```
setRandomBits (&shape_index, n_bits, seed);
rpe_index = shape_index;
rpe_phase = rpe_index % D;
rpe_index = rpe_index / D;
for (n = Np – 1; n >= 0; n- -)
{
  rpe_amps [n] = (rpe_index % 3) – 1;
  rpe_index = rpe_index / 3;
}
for (n = 0; n < sbfrm_size; n++)
{
  fcb_cng[n] = 0.0F;
}
for (n = 0; n < Np; n++)
{
  fcb_cng[rpe_phase + D*n] = gn[RPE_Configuration] *
(float)(rpe_amps [n]);
}
```

nbits устанавливается в 11 для *RPE_Configurations* 0 и 1 и устанавливается в 12 для *RPE_configurations* 2 и 3. *D* является коэффициентом децимации (прореживания), *Np* — число импульсов в подфрейме и *sbfrm_size* — число отсчетов в подфрейме, как определено в *MPEG-4 CELP* версии 1. Коэффициент нормализации *gn* определяется в таблице 106.

Таблица 106 — Коэффициент нормализации gn для RPE

RPE_Configuration	*gn* [ ]
0	56756 / 32768
1	56756 / 32738
2	44869 / 32768
3	40132 / 32768

5.9.3.3.7.5 Функция генератора случайного индекса

Функция генератора случайного индекса *setRandomBits* () определяется для *MPE* и *RPE* следующим образом:

```
void setRandomBits (long *l,
int n, short *seed)
{
  *l = 0xffff & Random(seed);
  if (n > 16)
  {
    *l |= (0xffff & Random(seed)) << 16;
  }
  if (n < 32)
  {
    *l &= ((unsigned long)1 << n) – 1;
  }
}
```

5.9.3.3.7.6 Подсчет усиления

Усиления *gf* и *gr* вычисляются из сглаженного *RMC* возбуждения, *norm_sm* следующим образом:

$$gf = \alpha \cdot norm_sm / \sqrt{\sum_{i-0}^{sbfrm_size-1} fcb_cng(i)^2 / sbfrm_size}$$

$$gr = [-\alpha \cdot A_3 + \sqrt{\alpha^2 \cdot A_3^2 - (\alpha^2 - 1) A_1 A_2}] / A_2$$

где $\alpha = 0{,}6$ и

$$A_1 = \sum_{i-0}^{sbfrm_size-1} gf^2 fcb_cng(i)^2$$

$$A_2 = \sum_{i-0}^{sbfrm_size-1} excg[i]^2$$

$$A_3 = \sum_{i-0}^{sbfrm_size-1} gf^2 \cdot fcb_cng(i) \cdot excg[i]$$

5.9.3.3.8 Фильтр синтеза *LP*

Фильтр синтеза идентичен фильтру синтеза *LP* в *MPEG-4 CELP*, описанному в 5.8.

5.9.3.3.9 Обновление памяти

Поскольку кодер и декодер должны быть сохранены синхронизированными во время неактивных периодов, генерация возбуждения выполняется на сторонах кодера и декодера, чтобы обновить соответствующие буферы для синтеза *LP*. Во время неактивных фреймов адаптивная книга шифров не используется и инициализируется нулем.

Приложение А
(справочное)

Инструменты декодера *MPEG-4 CELP*

А.1 Постпроцессор *CELP*

А.1.1 Описание инструмента

Постпроцессор *CELP* улучшает восстановленный речевой сигнал, сгенерированный фильтром синтеза для одного подфрейма. Инструменты постфильтрации включают постфильтр форманты и постфильтр компенсации наклона.

А.1.2 Определения

Вход

synth_signal[]: Этот массив содержит восстановленный речевой сигнал

int_Qlpc_coefficients[]: Этот массив содержит коэффициенты *LPC* для каждого подфрейма

acb_delay: Это поле указывает задержку шага, которая используется для постфильтра шага. Если постфильтр шага не нужен, *acb_delay* должен быть установлен в значение, меньшее чем 10.

adaptive_gain: Это поле указывает коэффициент усиления для периодического компонента сигнала возбуждения. Этот коэффициент усиления используется для постфильтра шага. Если постфильтр шага не нужен, *adaptive_gain* должен быть установлен в значение, меньшее чем 0,4

Выход

PP_synth_signal[]: Этот массив содержит постфильтрованный (улучшенный) речевой сигнал. Размерность этого массива — *sbfrm_size*

Конфигурация

lpc_order: Это поле указывает порядок *LPC*, который используется

sbfrm_size: Это поле указывает число отсчетов в подфрейме

А.1.3 Процесс декодирования

Процедура декодирования состоит из постфильтрации и адаптивного управления усилением. Постфильтр $H_{fpt}(z)$ является каскадным соединением трех фильтров: постфильтра форманты $H_f(z)$, дополнительного постфильтра шага $H_p(z)$ и фильтра компенсации наклона $H_t(z)$:

$$H_{fpt}(z) = H_f(z) \cdot H_p(z) \cdot H_t(z) .$$

Постфильтр форманты дается выражением

$$H_f(z) = \frac{\hat{A}(z/\gamma_n)}{\hat{A}(z/\gamma_d)} = \frac{1 - \sum_{i=1}^{lpc_order} \gamma_n^1 \hat{a}_i z^{-1}}{1 - \sum_{i=1}^{lpc_order} \gamma_n^1 \hat{a}_i z^{-1}},$$

где $\hat{A}z$ — фильтр инверсии *LPC*, коэффициенты γ_n и γ_d управляют степенью постфильтрации форманты. γ_n γ_d устанавливаются в 0,65 и 0,75, соответственно.

Постфильтр шага дается выражением

$$H_p(z) = \frac{1}{1 + \gamma_p g_a} (1 + \gamma_p g_a z^{acb_delay}),$$

$$g_a = \frac{\sum_{n=0}^{sbfm_size-1} s_r(n) s_r(n - acb_delay)}{\sum_{n=0}^{sbfm_size-1} s_r^2(n - acb_delay)},$$

где $s_r(n)$ является остаточным сигналом, произведенным фильтрацией входного сигнала через подсистему нумератора постфильтра форманты. Коэффициент усиления g_a ограничен 1. Коэффициент γ_p управляет степенью постфильтрации шага и имеет значение 0,5. Постфильтр шага применяется, только если усиление больше 0,4 и задержка шага больше 10. В частоте дискретизации 16 кГц для кодера режима I постфильтр шага не применяется.

Фильтр $H_t(z)$ компенсирует высокочастотный наклон и дается выражением

$$H_t(z) = 1 - \gamma_t z^{-1} ,$$

где γ_t — коэффициент наклона 0,3.

Синтезируемый сигнал *(synth_signal[])* фильтруется через подсистему нумератора постфильта форманты $\hat{A}(z/\gamma_n)$, чтобы произвести остаточный сигнал. В кодере режима II остаточный сигнал затем фильтруется через постфильтр шага $H_p(z)$, тогда как в кодере режима I никакой постфильтр шага не используется. Остаток постфильтрации шага подается через подсистему деноминатора постфильтра форманты $\hat{A}(z/\gamma_d)$. Выходной сигнал постфильтров форманты и шага передается через фильтр компенсации наклона $H_t(z)$, чтобы генерировать постфильтрованный синтезируемый сигнал, который еще не компенсирован по усилению.

Адаптивное управление усилением компенсирует различия усиления между синтезируемым сигналом $s(n)$ и постфильтрованным сигналом $s_p(n)$. Коэффициент управления усилением G для текущего подфрейма подсчитывается из

$$G = \sqrt{\frac{\sum_{n=0}^{sbfm_size-1} S^2(n)}{\sum_{n=0}^{sbfm_size-1} S_p^2(n)}}.$$

Масштабированный по усилению постфильтрованный сигнал $s_p(n)$ дается выражением

$$s(n) = g(n)s_p(n), \quad 0 \le n \le sbfrm_size - 1,$$

где $g(n)$ обновляется на поотсчетной основе и дается как

$$g(n) = 0{,}95\, g(n-1) + 0{,}05\, G, \quad 0 \le n \le sbfrm_size - 1.$$

Используется начальное значение $g(-1) = 0{,}0$. Тогда для каждого нового подфрейма $g(-1)$ устанавливается равным $g(N-1)$ предыдущего подфрейма.

Приложение В
(справочное)

Инструменты кодера *MPEG-4 CELP*

В.1 Введение в набор инструментов кодера *MPEG-4 CELP*

Это приложение дает краткое описание функциональных возможностей, определение параметров и процессов кодирования инструментов, поддерживаемых ядром *MPEG-4 CELP*. Описание каждого инструмента включает до четырех частей: описание инструмента, определения, процесс кодирования и таблицы.

Поддерживаются следующие инструменты кодера:

предварительная обработка *CELP*

анализ *CELP LPC*

квантизатор *CELP LPC* и интерполятор

векторный квантователь

кодер с масштабируемой полосой пропускания

фильтр анализа *CELP LPC*

модуль взвешивания *CELP*

анализ возбуждения *CELP*

возбуждение регулярным импульсом

мультиимпульсное возбуждение

мультиплексор потока бит *CELP*

Кодирование выполнено на основе фрейма, и каждый фрейм разделен на подфреймы. Инструмент анализа возбуждения *CELP* использует каждый подфрейм, в то время как другие инструменты используют каждый фрейм.

В.2. Переменные справки

Для каждого инструмента кодера дается описание переменных, которые он использует. В этом подпункте предоставлены переменные, которые используются совместно многими инструментами.

frame_size: Это поле указывает число отсчетов во фрейме. Декодер выводит фрейм с *frame_size* выборками.

nrof_subframes: Фрейм построен из ряда подфреймов. Число подфреймов определяется в этом поле.

sbfrm_size: Подфрейм состоит из ряда отсчетов, их число определяется в этом поле. Число отсчетов во фрейме всегда должно быть равным сумме чисел отсчетов в подфреймах. Так, что всегда должно соблюдаться следующее соотношение

*frame_size = nrof_subframes * sbfrm_size.*

Эти три параметра зависят от настройки параметров частоты дискретизации и битовой скорости, как представлено в таблице 73 для кодера режима I и в таблице 74 для кодера режима II.

lpc_order: Это поле указывает число коэффициентов, используемых для линейного предсказания. По умолчанию значение этого поля равно 20 для частоты дискретизации 16 кГц и 10 для 8 кГц.

num_lpc_indices. Этот параметр определяет число индексов, содержащих информацию *LPC*, которая должна быть записана в поток бит. Оно не равно порядку *LPC*. *num_lpc_indices* равно 5 в режиме 8 кГц и дополнительно 6 для уровня с масштабируемой полосой пропускания.

n_lpc_analysis: Это поле указывает, как часто во фрейме выполняется анализ *LPC*. Существует возможность выполнять несколько анализов *LPC* во фрейме с вариациями размера окна и смещения. Для частоты дискретизации 16 кГц значение этого поля равно 1, указывая, что анализ *LPC* выполняется только однажды. Для частоты дискретизации 8 кГц значение этого поля определяется отношением *sbfrm_size*/80.

window_offsets[]: Этот массив содержит смещения окон анализа *LPC*, и его размерность равна *n_lpc_analysis*.

window_sizes[]: Этот массив содержит размеры окна для анализа *LPC*. Так как анализ *LPC* выполняется *n_lpc_analysis* раз, размерность этого массива равна *n_lpc_analysis*.

Размер окна и смещение для кодера представлены в таблицах В.1, В.2, В.3, а параметры окна для кодера в таблице В.4.

Таблица В.1 — Размер окна и смещение для кодера режима I на 16 кГц

RPE_Configuration	*Window_sizes[]* (отсчетов)	*Window_offsets[]* (отсчетов)
0	400	280
1	320	160
2	400	280
3	400	280
4 ... 7	Зарезервировано	

Таблица В.2 — Размер окна и смещение для кодера режима II на 8 кГц

MPE_Configuration	*Window_sizes*[] (отсчетов)	*Window_offsets*[] (отсчетов)
0, 1, 2	200	0, 80, 160, 240
3, 4, 5	200	0, 80, 160
6 ... 12	200	0, 80
13 ... 21	200	0, 80
22 ... 26	200	0

Таблица В.3 — Размер окна и смещение для кодера режима II на 16 кГц

MPE_Configuration	*Window_sizes[]* (отсчетов)	*Window_offsets[]* (отсчетов)
0 ... 6, 8 ... 15	320	0, 80, 160, 240
16 ... 22, 24 ... 31	320	0, 80

windows[]: Этот массив содержит окно для каждого анализа, таким образом длина этого массива равна сумме *window_sizes* раз *n_lpc_analysis*. Для кодера режима I используется прямоугольное окно Хэмминга:

```
for (x = 0; x < window_sizes[i]; x++)
{
  window[i][x] = (0.54 – 0.46 * cos(2 * pi *
  (x/window_sizes[i])));
  window[i][x] = window[i][x] * window[i][x];
}
```

Для кодера режима II используется гибридное окно. Окно состоит из двух частей; половины окна Хэмминга и четверти косинусоидальной функции, дающихся выражением:

```
for (n = 0; n < nlb+len_lpcana/2; n++)
{
  Hw(n) = 0.54–0.46* cos(2* PI* n / (2*(nlb+len_lpcana/2) –
1));
}
for (n = nlb+len_lpcana/2; n < window_sizes[i]; n++)
{
  Hw(n) = cos (2 * PI * (n – (nlb+len_lpcana/2)) /
(4*(nla+len_lpcana/2)–1));
}
```

nlb отсчеты анализа прошлого фрейма, *len_lpcana* отсчеты анализа текущего фрейма, и *nla* отсчеты будущего фрейма формируют один блок для работы с окнами.

Таблица В.4 — Параметры окна для кодера режима II

Частота дискретизации, кГц	*nla*	*nlb*	*Len_lpc_ana*
8	40	80	80
16	80	160	80

gamma_be[]: Этот массив имеет размер *lpc_order,* чтобы применить расширение полосы пропускания к коэффициентам *LPC*. Эта информация используется только для кодера режима I.

```
gamma_be[0] = GAMMA;
for (x = 1; x < lpc_order; x++)
{
  gamma_be[x] = GAMMA * gamma_be[x–1];
}
```

Значение *GAMMA* равно 0,9883 для инструмента *RPE*.

n_lag_candidates: Эти поля содержат число кандидатов шага. Данная информация используется только для частоты дискретизации 16 кГц, значение этого поля равно 15.

max_pitch_frequency: Это поле содержит максимальную частоту шага (задержка). Для частоты дискретизации 16 кГц это поле имеет значение 0,025, поскольку минимальная задержка равна 40. Для частоты дискретизации 8 кГц это поле имеет значение 0,05882, так как минимальная задержка равна 17.

min_pitch_frequency: Это поле содержит минимальную частоту шага (задержка). Значение этого поля для частоты дискретизации 16 кГц равно $3{,}3898e^{-3}$, так как максимальная задержка равна 295. Для частоты дискретизации 8 кГц это поле имеет значение $6{,}944e^{-3}$, так как максимальная задержка равна 144.

B.3 Элементы потока бит для набора инструментов кодера *MPEG-4 CELP*

См. 5.4.

B.4. Предварительная обработка *CELP*

B.4.1 Описание инструмента

Инструмент предварительной обработки *CELP* производит речевой сигнал, свободный от составляющей постоянного тока.

B.4.2 Определения

Вход

s[]: Это — массив размерности *frame_size*, содержит входные речевые отсчеты.

Выход

pp_s []: Это массив длины *frame_size*, содержит речевые отсчеты, свободные от постоянного тока.

Вход/Выход

prev_x, prev_y: память фильтра предварительной обработки.

Конфигурация

frame_size: Это поле указывает число отсчетов во входном сигнале.

B.4.3 Процесс кодирования

Этот блок удаляет элемент постоянного тока из входного сигнала *s*[*n*]. Это рекурсивный фильтр первого порядка, его форма:

$$H_{pre}(z) = \frac{1 - z^{-1}}{1 - 0.99 \cdot z^{-1}}.$$

Реализация этого фильтра:

```
for (n = 0; n < frame_size; n++)
{
  pp_s[n] = s[n] – prev_x + 0.99 * prev_y;
  prev_x = s[n];
  prev_y = pp_s[n];
}
```

Состояния входа/выхода фильтра *prev_x* и *prev_y* инициализируются в нуль (обнуляются).

B.5 Анализ *CELP LPC*

B.5.1 Описание инструмента

Инструмент Анализа *CELP LPC* оценивает краткосрочный спектр. Анализ *LPC* выполняется на предобработанном речевом сигнале *pp_s* []. Порядок линейного предсказания определен параметром *lpc_order*. Чтобы взвешивать предобработанную речь, используется окно с размером, заданным в *window_size* []. Чтобы определить смещение для каждого окна, задается параметр *window_offset* [].

B.5.2 Определения

Вход

PP_InputSignal []: Этот массив содержит предобработанный речевой сигнал. Его размерность равна *frame_size*.

Выход

lpc_coefficients[]: Этот массив содержит вычисленные коэффициенты *LPC* и имеет размер *lpc_order*.

first_order_lpc_par: Это поле выхода содержит коэффициент *LPC* для согласования 1-го порядка. Этот параметр используется для предварительной выборки при поиске в адаптивной книге шифров.

Конфигурация

frame_size: Это поле обозначает число отсчетов во фрейме.

window_offset []: Этот массив содержит смещение окна.

window_size []: Этот массив содержит размер окна анализа *LPC*.

windows []: Этот массив содержит окна, используемые для взвешивания речевого сигнала.

gamma_be []: Этот массив содержит гамма-коэффициенты, которые используются для расширения полосы пропускания коэффициентов *LPC*.

lpc_order: Это поле указывает порядок *LPC*.

n_lpc_analysis: Это поле обозначает номер анализа *LPC*.

В.5.3 Процесс кодирования

Анализ линейного предсказания выполняется *n_lpc_analysis* раз, каждый раз с различным размером окна и смещением, как определено в массивах *window_size* и *window_offset*. Каждый раз входной сигнал *PP_InputSignal* взвешивается *w*[*n*].

Посредством взвешенного сигнала получают коэффициенты автокорреляции, используя:

$$acf[k] = \sum_{n=0}^{window\ size-k-1} sw[n] \cdot sw[n+k],\ 0 \le k \le lpc_order.$$

Имеется *lpc_order*+1 коэффициентов автокорреляции. Для кодера режима II расширение полосы пропускания и коррекция белого шума применяются путем модификации коэффициентов автокорреляции следующим образом:

```
for (k = 0; k < lpc_order; k++)
{
  acf[k] *= lag_win[k];
}
```

где *lag_win[]* является коэффициентами для расширения полосы пропускания.

Коэффициенты *LPC* вычисляются с помощью рекурсии *Levinson-Durbin* (Левинсона-Дурбина). Первый коэффициент *LPC* назначается для *first_order_lpc_par*. Для кодера режима I расширение полосы пропускания применяется к коэффициентам LPC, *u*используя массив *gamma_be*. Для каждого анализа *LPC* рассчетные *lpc_coefficients* располагают в стеке, приводя к коэффициентам *n_lpc_analysis* * *lpc_order*.

В.6 Квантизатор *LPC CELP* и интерполятор

Коэффициенты *LPC* квантуются при использовании одного из трех квантователей, инструмента узкополосного квантования *LSP*, инструмента широкополосного квантования *LSP* или инструмента квантования *LSP* с масштабируемой полосой пропускания.

В.6.1 Инструмент узкополосного квантования *LSP*

В.6.1.1 Описание инструмента

Инструмент узкополосного квантования *LSP* квантует коэффициенты *LPC* как параметры *LSP*, используя двухступенчатую и с разбиением вектора методику квантования.

В.6.1.2 Определения

Вход

lpc_coefficients[]: Это массив размерности *lpc_order*, содержит текущие неквантованные коэффициенты *LPC*.

Выход

int_Qlpc_coefficients[]: Это массив длиной *nrof_subframes* * *lpc_order*, содержит интерполированные и квантованные коэффициенты *LPC* для каждого подфрейма. Коэффициенты *LPC* для каждого подфрейма располагают в стеке один за другим, образуя массив *nrof_subframes* * *lpc_order*.

lpc_indices []: Это массив размерности *num_lpc_indices*, содержит упакованные индексы *lpc*, которые приписаны потоку бит.

Конфигурация

lpc_order: Это поле содержит порядок *LPC*.

num_lpc_indices: Эти поля указывают число упакованных кодов *LPC*.

num _lpc_analysis: Это поле содержит число параметров *LPC*.

nrof_subframes: Это поле содержит число подфреймов.

В.6.1.3 Процесс кодирования

LPCs конвертируются в *LSPs* и квантуются. Как описано в процессе декодирования, есть два метода квантования *LSPs*; двухступенчатый *VQ* без межфреймового предсказания, и комбинация *VQ* и *VQ* с межфреймовым предсказанием. В ходе процесса кодирования пытаются применить оба метода, чтобы квантовать *LSPs*, и какой метод должен быть применен, определяется сравнением ошибки квантования. Ошибка квантования вычисляется как взвешенное Евклидово расстояние. Коэффициенты взвешивания *w*[] следующие

$$w[i] = \begin{cases} \frac{1}{lsp[0]} + \frac{1}{lsp[1] - lsp[0]} & (i = 0) \\ \frac{1}{lsp[i] - lsp[i-1]} + \frac{1}{lsp[i+1] - lsp[i]} & (0 < i < N_p - 1) \\ \frac{1}{lsp[N_p - 1] - lsp[N_p - 2]} + \frac{1}{1{,}0 - lsp[N_p - 1]} & (i = N_p - 1) \end{cases},$$

где N_p — порядок анализа *LP lpc_order* и *lsp[]* являются *LSPs*, преобразованными из *LPCs*.

Квантователь первой стадии — для каждого метода квантования. *LSPs* квантуются при использовании квантователя разбитого надвое вектора, и соответствующие индексы сохраняются в *lpc_indices* [0] и *lpc_indices* [1]. Чтобы выполнить отсроченное решение, два индекса сохраняются как кандидаты на вторую стадию. Ошибка квантования в первой стадии *err1[]* дается выражением

$$err1[n] = \sum_{i=0}^{dim-1} \left\{ \left(lsp[sp+i] - lsp_tbl[n][m][i] \right)^2 \cdot w[sp+i] \right\} \quad n = 0,1,$$

где *n* — число вектора разбиения, *m* является индексом вектора разбиения кандидата, *sp* — порядок стартового *LSP n*-го вектора разбиения, и *dim* является размерностью *n*-го вектора разбиения (см. таблицу В.5).

Т а б л и ц а В.5 — Порядок запуска и размерность вектора *LSP* первой стадии

Номер вектора расщепления: *n*	Стартовый порядок *LSP*: *sp*	Размерность вектора: *dim*
0	0	5
1	5	5

Во второй стадии вышеупомянутые два метода квантования, которые являются также квантователями вектора с разбиением надвое, применены соответственно. Полные ошибки квантования во второй стадии подсчитываются для всех комбинаций кандидатов первой стадии и кандидатов второй стадии, и выбирается тот, у которого минимальная ошибка. В результате определяются индексы первой стадии и соответствующие индексы, и знаки для второй стадии сохраняются в *lpc_indices*[2] и *lpc_indices*[3]. Флажок, который указывает выбранный метод квантования, также сохраняется в *lpc_indices*[4]. Ошибка квантования во второй стадии *err2_total* дается выражением:

VQ без межфреймового предсказания:

err2_total = *err* 2[0] + *err*2[1]

$$err2[n] = \sum_{i=0}^{dim-1} \left\{ \left(lsp_res[sp+i] - sign[n] \cdot d_tbl[n][m][i] \right)^2 \cdot w[sp+i] \right\} \quad n = 0,1$$

lsp_res[*sp* + *i*] = *lsp*[*sp* + *i*] - *lsp_first*[*sp* + *i*],

где *lsp_first[]* является квантованным вектором *LSP* первой стадии, *n* — число разбиения вектора, *m* является индексом вектора разбиения кандидата, *sp* — стартовый порядок *LSP n*-го вектора разбиения и *dim* является размерностью *n*-го вектора разбиения,

VQ с межфреймовым предсказанием:

err2_total = *err*2[0] + *err*2[1]

$$err2[n] = \sum_{i=0}^{dim-1} \left\{ \left(lsp_pres[sp+i] - sign[n] \cdot pd_tbl[n][m][i] \right)^2 \cdot w[sp+i] \right\} \quad n = 0,1$$

lsp_pres[*sp* + *i*] = *lsp*[*sp* + *i*]–

{(1 – *ratio_predict*) · *lsp_first*[*sp* + *i*] + *ratio_predict* · *lsp_previous*[*sp* + *i*]},

где *lsp_first[]* является квантованным вектором *LSP* первой стадии, *n* — номер вектора разбиения, *m* является индексом вектора разбиения кандидата, *sp* — стартовый порядок *LSP n*-го вектора разбиения, *dim* является размерностью *n*-го вектора разбиения и *ratio_predict*=0,5 (см. таблицу В.6).

Таблица В.6 — Порядок запуска и размерность вектора *LSP* второй стадии

Номер вектора расщепления: *n*	Стартовый порядок *LSP*: *sp*	Размерность вектора: *dim*
0	0	5
1	5	5

Квантованные *LSPs lsp_current[]* стабилизированы, чтобы гарантировать стабильность фильтра синтеза *LPC*, который получен из квантованных *LSPs*. Квантованные *LSPs* упорядочены, имея расстояние, по крайней мере, *min_gap* между смежными коэффициентами.

```
for (i = 0; i < lpc_order; i++)
{
  if (lsp_current[i] < min_gap)
  {
    lsp_current[i] = min_gap;
  }
}
for (i = 0; i < lpc_order - 1; i++)
{
  if (lsp_current[i+1] – lsp_current[i] < min_gap)
  {
    lsp_current[i+1] = lsp_current[i]+min_gap;
  }
}
for (i = 0; i < lpc_order; i++)
{
  if (lsp_current[i] > 1–min_gap)
  {
    lsp_current[i] = 1–min_gap;
  }
}
for (i = lpc_order – 1; i > 0; i- -)
  if (lsp_current[i]–lsp_current[i–1] < min_gap)
  {
    lsp_current[i–-1] = lsp_current[i]–min_gap;
  }
}
```

где *min_gap* = 2,0/256,0

После процесса квантования квантованные *LSPs* в каждом подфрейме линейно интерполируются.

```
for (n = 0; n < nrof_subframes; n++)
{
  ratio_sub=(n+1)/nrof_subframes;
  for (i = 0; i < lpc_order; i++)
  {
    lsp_subframe[n][i]=((1–
    ratio_sub)*lsp_previous[i]+ratio_sub*lsp_current[i]));
  }
}
```

Интерполированные *LSPs* конвертируются в *LPCs*, используя вспомогательную функцию *Convert2lpc()*.

```
for (n = 0; n < nrof_subframes; n++)
{
  Convert2lpc(lpc_order, lsp_subframe[n], int_Qlpc_coefficients +
n*lpc_order]);
}
```

После вычисления коэффициентов *LPC* текущие *LSPs* должны быть сохранены в памяти, так как они используются для интерполяции в следующем фрейме.

```
for (i = 0; i < lpc_order; i++)
{
  lsp_previous[i] = lsp_current[i];
}
```

Сохраненные *LSPs lsp_previous[]* должны быть инициализированы, как описано ниже, когда инициализируется весь кодер.

```
for (i = 0; i < lpc_order; i++)
{
  lsp_previous[i] = (i+1) / (lpc_order+1);
}
```

В.6.2 Инструмент квантования широкополосного *LSP*

В.6.2.1 Описание инструмента

Инструмент квантования широкополосный *LSP* квантует коэффициенты *LPC* как параметры *LSP*, используя методику двухступенчатого и с разбиением вектора квантования.

В.6.2.2 Определения

Вход

lpc_coefficients[]: Это массив размерности *lpc_order*, содержит коэффициенты текущего неквантованного *LPC*.

Выход

int_Qlpc_coefficients[]: Это массив длины *nrof_subframes * lpc_order*, содержит интерполированные и квантованные коэффициенты *LPC* для каждого подфрейма. Коэффициенты *LPC* для каждого подфрейма расположены в стеке друг за другом, образуя массив *nrof_subframes * lpc_order*.

lpc_indices[]: Это массив размерности *num_lpc_indices*, содержит упакованные индексы *lpc*, которые приписаны потоку бит.

Конфигурация

lpc_order: Это поле содержит порядок *LPC*.

num_lpc_indices: Эти поля указывают число упакованных кодов *LPC*

n_lpc_analysis: Это поле содержит число параметров *LPC*

nrof_subframes: Это поле содержит число подфреймов

В.6.2.3 Процесс кодирования

Схема квантования основана на узкополосном квантовании *LSP*. Квантователь состоит из двух блоков квантования, соединенных параллельно, каждый из которых идентичен инструменту узкополосного квантования *LSP*. Входные *LSPs* делятся на две части, а именно, нижняя часть и верхняя часть, затем разделенные *LSPs* вводятся в блоки квантования соответственно.

Во-первых, нижняя часть квантуется тем же самым способом как узкополосное квантование *LSP*. Стартовый порядок и размерность векторов *LSP* описаны в таблицах В.7 и В.8. Квантованные *LSPs* сохраняются в массиве *lsp_current_lower[]*.

Таблица В.7 — Стартовый порядок и размерность вектора *LSP* первой стадии

Номер вектора расщепления: *n*	Стартовый порядок нижнего *LSP*: *sp*	Размерность вектора: *dim*
0	0	5
1	5	5

Таблица В.8 — Стартовый порядок и размерность вектора *LSP* второй стадии

Номер вектора расщепления: *n*	Стартовый порядок нижнего *LSP*: *sp*	Размерность вектора: *dim*
0	0	5
1	5	5

Затем, тем же самым способом квантуется верхняя часть. Стартовый порядок и размерность векторов *LSP* описаны в таблице В.9 и таблице В.10. Квантованные *LSPs* сохраняются в массиве *lsp_current_upper []*. В квантовании верхней части индексы первой стадии сохраняются в *lpc_indices* [5] и *lpc_indices* [6], а индексы и знаки для второй стадии сохраняются в *lpc_indices* [7] и *lpc_indices* [8]. Флажок, который указывает выбранный метод квантования, также сохраняется в *lpc_indices* [9].

Таблица В.9 — Стартовый порядок и размерность вектора *LSP* первой стадии

Номер вектора расщепления: *n*	Стартовый порядок верхнего *LSP*: *sp*	Размерность вектора: *dim*
0	0	5
1	5	5

Таблица В.10 — Стартовый порядок и размерность вектора *LSP* второй стадии

Номер вектора расщепления: *n*	Стартовый порядок верхнего *LSP*: *sp*	Размерность вектора: *dim*
0	0	5
1	5	5

Наконец, декодированные *LSPs lsp_current_lower[]* и *lsp_current_upper[]* объединяются и сохраняются в массиве *lsp_current[].*

```
for (i = 0; i < lpc_order/2; i++)
{
  lsp_current[i] = lsp_current_lower[i];
}
for (i = 0; i < lpc_order/2; i++)
{
  lsp_current[lpc_order/2+i] = lsp_current_upper[i];
}
```

Квантованные *LSPs, lsp_current[]* стабилизируются, чтобы гарантировать стабильность фильтра синтеза *LPC*, который получают из квантованных *LSPs*. Квантованные *LSPs* упорядочиваются в порядке возрастания, имея расстояние, по крайней мере, *min_gap* между смежными коэффициентами

```
for (i = 0; i < lpc_order; i++)
{
  if (lsp_current[i] < min_gap)
  {
    lsp_current[i] = min_gap;
  }
}
for (i = 0; i < lpc_order – 1; i++)
{
  if (lsp_current[i+1] – lsp_current[i] < min_gap)
  {
    lsp_current[i+1] = lsp_current[i]+min_gap;
  }
}
for(i = 0; i < lpc_order; i++)
{
  if (lsp_current[i] > 1–min_gap)
  {
    lsp_current[i] = 1–min_gap;
  }
for (i = lpc_order – 1; i > 0; i- -)
{
  if (lsp_current[i]–lsp_current[i–1] < min_gap)
  {
    lsp_current[i–1] = lsp_current[i]–min_gap;
  }
}
```

где *lpc_order* = 20 и *min_gap* = 1,0/256,0

После процесса квантования квантованные *LSPs* линейно интерполируются в каждом подфрейме.

```
for (n = 0; n < nrof_subframes; n++)
{
  ratio_sub=(n+1)/nrof_subframes;
  for (i = 0; i < lpc_order; i++)
  {
    lsp_subframe[n][i]=((1–
ratio_sub)*lsp_previous[i]+ratio_sub*lsp_current[i]));
  }
}
```

Интерполированные *LSPs* преобразуются в *LPCs*, используя вспомогательную функцию *Convert2lpc ()*.

```
for (n = 0; n < nrof_subframes; n++)
{
```

```
  Convert2lpc(lpc_order, lsp_subframe[n], int_Qlpc_coefficients
+ n*lpc_order]);
}
```

После вычисления коэффициентов *LPC* текущие *LSPs* должны быть сохранены в памяти, так как они используются для интерполяции в следующем фрейме.

```
for (i = 0; i < lpc_order; i++)
{
  lsp_previous[i] = lsp_current[i];
}
```

Сохраненные *LSPs, lsp_previous[]*, должны быть инициализированы как описано ниже, когда инициализируется весь кодер.

```
for (i = 0; i < lpc_order; i++)
{
  lsp_previous[i] = (i+1) / (lpc_order+1);
}
```

B.6.3 Инструмент квантования *LSP* с масштабируемой полосой пропускания

B.6.3.1 Описание инструмента

Инструмент квантования *LSP* с масштабируемой полосой пропускания квантует входные *LSPs* с частотой дискретизации 16 кГц, используя схему векторного квантования с внутрифреймовым и межфреймовым предсказанием.

B.6.3.2 Определения

Вход

lpc_coefficients[]: Это — массив размерности *lpc_order*, содержит текущие неквантованные коэффициенты *LPC*.

lsp_current[]: Этот массив содержит декодированные параметры *LSP*, которые нормализованы в диапазоне от нуля до *PI*, в инструменте узкополосного квантования *LSP*. Эти параметры получают как промежуточные параметры в процессе узкополосного квантования *LSP* и отправляют на инструмент декодирования *LSP* с масштабируемой полосой пропускания

Выход

int_Qlpc_coefficients[]: Это массив длины *nrof_subframes * lpc_order*, содержит интерполированные и квантованные коэффициенты *LPC* для каждого подфрейма. Коэффициенты *LPC* для каждого подфрейма расположены в стеке один за другим, образуя массив *nrof_subframes * lpc_order*.

lpc_indices[]: Это массив размерности *num_lpc_indices*, содержит упакованные индексы *lpc,* которые приписаны потоку бит.

Конфигурация

lpc_order: Это поле содержит порядок *LPC*

num_lpc_indices: Эти поля указывают число упакованных кодов *LPC*

n_lpc_analysis: Это поле содержит число параметров *LPC*

nrof_subframes: Это поле содержит число подфреймов

nrof_subframes_bws: Этот параметр является переменной справки, представляет число подфреймов в уровне масштабируемой полосы пропускания.

B.6.3.3 Процесс кодирования

Входные *LSPs* с частотой дискретизации 16 кГц (*input_lsp[]*) являются векторно-квантованными методом внутрифреймового и межфреймового предсказания. Модуль внутрифреймового предсказания производит оценочные *LSPs*, преобразовывая квантованные *LSPs*, полученные в кодере *CELP* с частотой дискретизации 8 кГц. Кроме того, для более точного предсказания также используется предиктивный *VQ* межфреймового скользящего среднего. Остаточные *LSPs* предсказания (*err_lsp[]*) вычисляются следующим образом:

```
for (i = 0; i < 20; i++)
{
  err_lsp[i] = (input_lsp[i] – pred_lsp[i]) /
bws_ma_prdct[0][i];
}
for (n = 1; n <= 2; n++)
{
  for (i = 0; i < 20; i++)
  {
    pred_lsp[i] += bws_ma_prdct[n][i]*lsp_bws_buf[n][i];
  }
}
for (i = 0; i < 10; i++)
{
```

```
    pred_lsp[i] += bws_nw_prdct[i]*lsp_current[i];
  }
```

где *pred_lsp[]* содержат предсказанные *LSPs*. *bws_ma_prdct[][]* и *bws_nw_prdct[]* являются коэффициентами предсказания для межфреймового предсказания скользящего среднего и внутрифреймового предсказания соответственно. *lsp_bws_buf[][]* является буфером, содержащим остаток предсказания *LSP* в предыдущих двух фреймах.

Тогда, *err_lsp[]* являются векторно-квантованными, используя схему двухступенчатого и с разбиением вектора квантования, и соответствующие индексы сохраняются в *lpc_indices*[5]..., *lpc_indices*[10]. Буфер *lsp_bws_buf[][]* сдвигается для операции следующего фрейма следующим образом:

```
for (i = 0; i < 20; i++)
{
  lsp_bws_buf[0][i] = err_lsp[i];
}
for (n = 2; n > 0; n- -)
{
  for (i = 0; i < 20; i++)
  {
    lsp_bws_buf[n][i] = lsp_bws_buf[n–1][i];
  }
}
```

После процесса квантования квантованные *LSPs* (*lsp_bws_current[]*) линейно интерполируются в каждом подфрейме.

```
for (n = 0; n < nrof_subframes_bws; n++)
{
  ratio_sub = (n+1)/nrof_subframes_bws;
  for(i = 0; i < 2*lpc_order; i++)
  {
  lsp_bws_subframe[n][i]=((1–ratio_sub)*lsp_bws_previous[i]
                          + ratio_sub*lsp_bws_current[i]));
  }
}
for (i = 0; i < 2*lpc_order; i++)
{
  lsp_bws_previous[i] = lsp_bws_subframe[nrof_subframes_bws–1][i];
}
```

Интерполированные *LSPs* конвертируются в коэффициенты *LPC* в каждом подфрейме.

```
for (n = 0; n < nrof_subframes_bws; n++)
{
  Convert2lpc (lpc_order_bws, lsp_bws_subframe[n],
                          &int_Qlpc_coefficients[n*lpc_order_bws]);
}
```

B.6.4 Тонкое управление скоростью в инструменте квантования *LSP*

B.6.4.1 Описание инструмента

Тонкое управление скоростью (*FRC*) доступно с применением инструмента квантования *LSP*. Если используется *FRC*, массив *lpc_coefficients[]* содержит коэффициенты *LPC* следующего фрейма, который является фреймом, следующим после обрабатываемого в данное время фрейма. Принимается решение, передавать ли *LSPs* обрабатываемого в настоящее время фрейма в декодер. Согласно решению, *interpolation_flag* и *LPC_present* устанавливаются в 1 или 0.

B.6.4.2 Определения

Вход

lpc_coefficients[]: Это массив размерности *lpc_order*, который содержит коэффициенты текущего неквантованного *LPC*

Выход

int_Qlpc_coefficients[]: Это массив длины *nrof_subframes * lpc_order*, который содержит интерполированные и квантованные коэффициенты *LPC* для каждого подфрейма. Коэффициенты *LPC* для каждого подфрейма расположены в стеке один за другим, образуя массив *nrof_subframes * lpc_order*

lpc_indices[]: Это массив размерности *num_lpc_indices*, содержит упакованные индексы *lpc*, которые записаны в потоке бит

interpolation_flag: Это поле указывает, выполнена ли интерполяция коэффициентов *LPC* между фреймами. Если флажок интерполяции установлен, текущие коэффициенты *LPC* вычислены из предыдущих и следующих коэффициентов *LPC*

LPC_Present: Этот флажок указывает, закончен ли текущий фрейм или неполон

B.6.4.3 Процесс кодирования

Решение о том, передавать ли коэффициенты *LPC* анализируемого речевого фрейма на декодер, зависит от количества различий *d* между спектром текущего фрейма и спектрами смежных фреймов. Если *d* больше, чем заданный порог, тогда коэффициенты передаются в декодер. Этот порог далее сделан зависящим от установки желательной битовой скорости следующим образом: порог поднят, если фактическая битовая скорость выше, чем установка желательной битовой скорости, а иначе — понижен.

B.7 Фильтр анализа *CELP LPC*

B.7.1 Описание инструмента

Инструмент фильтра анализа *CELP LPC* подает входные сигналы через фильтр с коэффициентами *LPC* и возвращает остаточный сигнал.

B.7.2 Определения

Вход

PP_InputSignal[]: Этот массив имеет размерность *sbfrm_size* и содержит входной сигнал

int_Qlpc_coefficients[]: Этот массив имеет размерность *lpc_order* и содержит коэффициенты *LPC*

Выход

lpc_residual[]: Этот массив имеет размерность *sbfrm_size* и содержит фильтрованный остаточный сигнал *LPC*

Конфигурация

lpc_order: Это поле указывает порядок *LPC*

sbfrm_size: Это поле указывает число отсчетов в одном подфрейме

B.7.3 Процесс кодирования

Входной сигнал фильтруется, используя коэффициенты фильтра.

```
for (k = 0; k < sbfrm_size; k++)
{
  tmp = PP_InputSignal[k];
  for (j = 0; j < lpc_order; j++)
  {
    tmp =  tmp – int_Qlpc_coefficients[j] * Filter_States[j];
  }
  lpc_residual[k] = tmp;
  update_Filter_States;
}
```

Начальные состояния фильтра установлены на нуль.

B.8 Модуль взвешивания *CELP*

B.8.1 Описание инструмента

Модуль взвешивания *CELP* вычисляет значения, которые будут применены к коэффициентам *LPC*.

B.8.2 Определения

Вход

lpc_coefficients[]: Это массив размерности *lpc_order*, содержащий коэффициенты *LPC*

gamma_num: Это поле содержит коэффициент взвешивания числителя

gamma_den: Это поле содержит коэффициент взвешивания знаменателя

Выход

Wnum_coeff[]: Это массив размерности *Wnum_order*, содержащий взвешенные коэффициенты числителя

Wden_coeff[]: Этот массив имеет размерность *Wden_order* и содержит взвешенные коэффициенты знаменателя

Конфигурация

lpc_order: Это поле содержит порядок *LPC*

B.8.3 Процесс кодирования

Взвешенные коэффициенты числителя вычисляются согласно выражению:

```
for (k = 0; k < Wnum_order; k++)
{
  Wnum_coeff[k] = lpc_coefficients[k] * gamma_num^(k+1);
}
```

Взвешенные коэффициенты знаменателя вычисляются согласно выражению:

```
for (k = 0; k < Wden_order; k++)
{
  Wden_coeff[k] = lpc_coefficients[k] * gamma_den^(k+1);
}
```

B.9 Анализ возбуждения *CELP*

Анализ возбуждения *CELP* вычисляет векторы формы и усиления, а также декодированный синтезированный речевой сигнал. Для модулей анализа возбуждения определены возбуждение регулярным импульсом (*RPE*) и мультиимпульсное возбуждение (*MPE*).

В.9.1 Возбуждение регулярным импульсом

В.9.1.1 Описание инструмента

Для возбуждения регулярным импульсом выходом анализа возбуждения являются *shape_delay*[], *shape_index*[] и *gain_indices*[]. Индексы формы и усиления генерируют каждый подфрейм. Вектор *shape_delay*[] содержит задержку адаптивной книги шифров для каждого подфрейма, в то время как вектор *shape_index*[] содержит индекс фиксированной книги шифров. Вектор *gain_indices*[0][] содержит усиление адаптивной книги шифров для каждого подфрейма. Усиление фиксированной книги шифров для каждого подфрейма сохраняется в векторе *gain_indices*[1][].

В.9.1.2 Определения

Вход

PP_inputSignal[]: Этот массив содержит предобработанный входной сигнал и имеет размерность *sfrm_size*

lpc_residual[]: Этот массив содержит остаточный сигнал *lpc*

int_Qlpc_coefficients[]: Этот массив содержит коэффициенты интерполированного и квантованного *LPC*

Wnum_coeff[]: Этот массив содержит коэффициенты фильтра взвешивания числителя

Wden_coeff[]: Этот массив содержит коэффициенты фильтра взвешивания знаменателя

first_order_lpc_par: Это поле указывает первый коэффициент *LPC*

lag_candidates[]: Этот массив содержит кандидатов задержки

signal_mode: Это поле содержит флажок речевой/неречевой

rmc_index: Это поле определяет индекс для мощности сигнала

Выход

shape_delay[]: Этот массив имеет размерность *nrof_subframes*. Он содержит задержку книги шифров для адаптивных книг шифров

shape_index[]: Этот массив имеет размерность *nrof_subframes*. Он содержит индексы формы для фиксированных книг шифров

gain_indices[][]: Этот массив имеет размерность *2 * nrof_subframes*. Он содержит индексы усиления для адаптивной и фиксированной книг шифров

decoded_excitation[]: Этот массив имеет длину *sbfrm_size* и содержит синтезируемый речевой сигнал

Конфигурация

n_lag_candidates: Это поле указывает число кандидатов задержки

frame_size: Это поле указывает количество отсчетов в одном фрейме

sbfrm_size: Это поле указывает количество отсчетов в одном подфрейме

nrof_subframes: Это поле указывает число подфреймов в одном фрейме

lpc_order: Это поле указывает порядок *LPC*

В.9.1.3 Процесс кодирования

Все блоки выполнены на основе единой для подфрейма базы. Для каждого подфрейма выполняются следующие шаги:

перцепционное взвешивание,

предварительный выбор поиска адаптивной книги шифров,

поиск адаптивной книги шифров,

предварительный выбор поиска фиксированной книги шифров,

поиск фиксированной книги шифров,

моделирование декодера.

Затем каждый шаг будет описан подробно. Для удобства вместо *int_Qlpc_coefficients*[] используется *aq*[], таким образом *aq*[] содержит квантованные коэффициенты *LPC* для рассматриваемого подфрейма.

Перцепционное взвешивание выполняется фильтрацией входного сигнала *PP_InputSignal[]* следующим фильтром:

$$W(z) = \frac{A(z)}{A(z/\gamma)} = \frac{1 - \sum_{k=0}^{lpc\,order-1} aq[k] \cdot z^{-k-1}}{1 - \sum_{k=0}^{lpc\,order-1} aq[k] \cdot \gamma^{k+1} \cdot z^{-k-1}}$$

при $\gamma = 0{,}8$. Получающийся сигнал называют *ws[n]*.

Отклик на нулевой вход *z*[*n*] определяется вычислением отклика *S(z)* на входной сигнал с нулевым значением.

$$S(z) = \frac{1}{A(z/\gamma)} = \frac{1}{1 - \sum_{k=0}^{lpc\,order-1} aq[k] \cdot \gamma^{k+1} \cdot z^{-k-1}},$$

где $\gamma = 0{,}8$.

Взвешенный целевой сигнал *t[n]* получают вычитанием *z[n]* из *ws[n]*:

```
for (n = 0; n < Nm; n++)
{
  t[n] = ws[n] - z[n];
}
```

Импульсная передаточная функция *h[n]* вычисляется следующим образом:

```
for (k = 0; k < lpc_order; k++)
{
  tmp_states[k] = 0;
}
 tmp = 1.0;
for (n = 0; n < sbfrm_size; n++)
{
  g = 0.8;
  for (k = 0; k < lpc_order; k++)
  {
    tmp = tmp + aq[k][s] * g * tmp_states[k];
    g = 0.8 * g;
  }
  h[n] = tmp;
  for (k = lpc_order–1; k > 0; k- -)
  {
    tmp_states[k] = tmp_states[k–1];
  }
  tmp_states[0] = tmp;
}
```

После вышеупомянутых вычислений выполняется предварительная выборка в адаптивной книге шифров. Адаптивная книга шифров содержит 256 последовательностей книги шифров, из которых 5 предвыбраны. Предварительная выборка достигается, оценивая элемент (для 0 <= *l* <*Lm* и 0 <= *i* <*nrof_subframes*):

$$rap[i \cdot Lm + l] = \frac{\left(\sum_{n=0}^{sbfrm_size-1} ca \cdot [L\min + i \cdot sbfrm_size + l \cdot 3 - n - 1] \cdot ta[n]\right)^2}{Ea[i \cdot Lm + l]},$$

где

$$Lm = 1 + \left[\frac{sbfrm_size - 1}{3}\right]$$

и *ca*[Lmin + *i sbfrm_size* +3*l* – *n*–1] и *ta*[*n*], представляющих последовательность книги шифров при задержке (Lmin+*i sbfrm_size*+3*l*) и «фильтрованном в обратном направлении» целевом сигнале *t[n]*, соответственно.

Обратная фильтрация включает обращение времени *t[n]*, фильтрацию *S(z)* и снова обращение времени. *Lmin* — минимальная задержка в отсчетах. Значение *Lmin* равно 40.

Ea[i, Lm+l] является энергией той последовательности книги шифров, фильтрованной фильтром синтеза уменьшенной сложности $S_p(z)$, которая является оценкой первого порядка фильтра синтеза *LPC* *S*(*z*):

$$S_p(z) = \frac{1}{A_p}(z/\gamma) = \frac{1}{1 - \alpha \cdot \gamma \cdot z^{-1}}.$$

Первый коэффициент *LPC* фрейма берется для a. В зависимости от номера подфрейма, который рассматривается, первый коэффициент *LPC* текущего или предыдущего берется, используя следующее:

```
if (subframe_number < nr_subframes/2)
  a = prev_a;
else
{
  a = cur_a;
}
```

После оценки *rap[]* выбираются максимально 5 последовательностей совместно с их 2 соседними последовательностями, приводя к 15 кандидатам, к которым применяется полный поиск. Индексы предвыбранных последовательностей и их соседей сохраняются в *ia[r]* (0 <= *r* <15).

Поиск адаптивный книги шифров минимизирует среднеквадратическую взвешенную ошибку между оригиналом и восстановленный речью. Это достигается поиском индекса *r,* максимизирующего элемент

$$ra(r) = \frac{\left(\sum_{n=0}^{sbfrm_size-1} t[n] \cdot y[r][n]\right)^2}{\left(\sum_{n=0}^{sbfrm_size-1} y^2[r][n]\right)}.$$

Сигнал *t[n]* представляет взвешенный целевой сигнал, *y[r][n]* — свертка последовательности *ca[ia[r]–n–1]* с импульсной передаточной функцией *h[n]*. Индекс *(ia[r]–Lmin)* максимума обозначен как *shape_delay* [*subframe*].

После определения индекса коэффициент усиления вычисляется согласно

$$g = \frac{\sum_{n=0}^{sbfrm_size-1} t[n] \cdot y'[n]}{\sum_{n=0}^{sbfrm_size-1} y'^2[n]}$$

при *y'* [*n*] равном свертке *ca [l+Lmin–n–1]* с *h[n]*. Коэффициент усиления квантуется неоднородным квантователем:

```
for (j = 0; abs(g) > cba_gain_quant[j] && j < 31; j ++);
if (g < 0)
{
  Ga = –cba_gain[j];
  gain_indices[0] = (((–j – 1)) & 63);
}
Else
{
  Ga = cba_gain[j];
  Gain_indices[0][subframe] = j;
}
```

Квантованный коэффициент усиления называют *Ga*. С помощью *shape_delay*[] и *Ga* вычисляется вклад *p[n]* адаптивной книги шифров согласно

$$p[n] = Ga \cdot y'[n].$$

Вклад *p[n]* адаптивной книги шифров вычитается из взвешенного целевого сигнала *t[n]*, чтобы получить остаточный сигнал *e[n]*:

$$e[n] = t[n] — p[n],\ 0 \le n < sbfrm_size.$$

Остаточный сигнал *e[n]* «фильтруется в обратном направлении», чтобы получить *tf[n]*.

Поиск фиксированный книги шифров также состоит из предварительного выбора и фазы выбора. Для возбуждения фиксированной книги шифров используется *RPE*-книга шифров. Каждый вектор книги шифров имеет *sbfrm_size* импульсов, из которых *Np* импульсов могут быть с амплитудой +1, 0 или –1. Эти *Np* импульсов позиционируются в регулярную сетку, определенную фазой *p* и междуимпульсным промежутком *D* таким образом, что позиции сетки равны *p* + *Dl*, где *l* между 0 и *Np*. Оставшиеся (*sbfrm_size–Np*) импульсов являются нулевыми. *D* и *Np* зависят от установки битовой скорости, как дано в таблице 87.

Чтобы уменьшить сложность, локальная *RPE*-книга шифров генерируется для каждого подфрейма, содержащего подмножество 16 входов. У всех векторов этой локальной *RPE*-книги шифров одна и та же фаза *P*, которая вычисляется следующим образом:

```
max = 0;
for (p = 0; p < D; p++)
{
  tmp_max = 0;
  for (l = 0; l < Np–1; l++)
  {
    tmp_max += abs(tf[p+D * l]);
  {
  if (tmp_max > max)
  {
    max = tmp_max;
    P = p;
  }
}
```

Определив фазу *P*, нужно чтобы амплитуда импульса 1, 0 <= *l* <*Np* была нулем, или равной знаку соответствующего отсчета *tf* [*n*]. Знак сохраняется в *amp*[*l*] следующим образом:

```
for (l = 0; l < Np; l++)
{
```

```
  amp[l] = sign (tf [P + D * l]);
}
```

где знак равен +1 для значений, больших 0 и –1 — в ином случае.

Для импульсов *Np*-4 возможность нулевой амплитуды исключена априорно в тех позициях, где *tf* [*n*] максимален. Поэтому используется массив *pos*[], который имеет следующую семантику:

pos[l] = 0 указывает, что возможность нулевой амплитуды включена для импульса l.

pos[l] = 1 указывает, что возможность нулевой амплитуды исключена.

Чтобы определить массив *pos*[], используется следующий алгоритм:

1. Инициализировать *pos*[] нулями,

2. Определить *n* таким образом, чтобы выполнялось *abs*(*tf* [*P* + *D* * *n*])> = *abs* (*tf* [*P* + *D* * *i*]) для всех *i* не равных *n*,

3. *pos*[*n*] = 1,

4. *tf* [*P* + *D* * *n*] = 0,

5. Продолжить с шагом 2, пока не фиксируются *Np*-4 позиций.

На основе *P amp*[] и *pos* генерируется локальная *RPE*-книга шифров *cf*[*k*][*n*], используя следующий алгоритм:

```
for (k = 0; k < 16; k++)
{
  for (n = 0; n < sbfrm_size}; n++)
  {
    cf[k][n] = 0;
  }
}
for (l = 0; l < Np; l++)
{
  cf[0][P+D*l]=amp[l];
}
m = 1;
for (l = 0; l < Np; l++)
{
  if (pos[l]==0)
  {
    c = m;
    for (q = 0; q < c; q++)
    {
      for (n = 0; n < Np; n++)
      {
        cf[m][P+D*n] = cf[q][P+D*n];
      }
      cf[m++][P+D*l] = 0;
    }
  }
}
```

На основе локальной *RPE*-книги шифров предварительно выбираются 5 векторов из 16 для поиска с обратной связью. Предварительная выборка достигается путем оценки элемента для 0 <= *k* <16:

$$rfp(k) = \frac{\left(\sum_{n=0}^{sbfrm_size-1} cf[k][n] \cdot tf[n]\right)^2}{Ef[l]}$$

с *cf* [*k*][*n*] представлением вектора локальной *RPE*-книги шифров. *Ef* [*k*] является энергией этого вектора книги шифров, фильтрованного фильтром синтеза уменьшенной сложности $S_p(z)$. Индексы предвыбранных векторов сохраняются в *if*[*r*].

Используя эти предвыбранные индексы, выполняется поиск фиксированный книги шифров с обратной связью. С помощью поиска фиксированной книги шифров с обратной связью разыскивается индекс *r*, максимизирующий элемент

$$rf[r] = 2 \cdot Gf \cdot \sum_{n=0}^{sbfrm_size-1} e[n] \cdot y[r][n] - Gf^2 \cdot \sum_{n=0}^{sbfrm_size-1} y^2[r][n].$$

Сигнал *e[n]* представляет остаточный сигнал поиска адаптивной книги шифров, а *y[r][n]* — свертка *cf[if[r]][n]* с импульсной реакцией *h[n]*. *Gf* — квантованный коэффициент усиления *g*, который определяется согласно

$$g = \frac{\sum_{n=0}^{sbfrm_size-1} e[n] \cdot y[r][n]}{\sum_{n=0}^{sbfrm_size-1} y^2[r][n]}.$$

Процесс квантования усиления фиксированной книги шифров дается ниже, где *Gp* — предыдущее квантованное усиление фиксированной книги шифров.

```
if (first subframe)
{
for (m = 0; g > cbf_gain_quant[m] && m < 30; m ++);
   Gf = cbf_gain[m];
}
else
{
g = g / Gp;
for (m = 0; g > cbf_gain_quant_dif[m]  && m < 7; m ++);
   g = Gp * cbf_gain_diff[m]; }
}
   gain_indices[0] = m.
```

Представления *cbf_gain* и *cbf_gain_dif* даются в таблице 89 и таблице 90; таблицы квантования *cbf_gain_quant* и *cbf_gain_quant_dif* даются ниже.

Наконец, декодер моделируется, выполняя следующие шаги:

подсчитывается возбуждение фиксированной книги шифров,

суммируются возбуждение фиксированной и адаптивной книг шифров,

обновляется память адаптивной книги шифров,

обновляется память фильтров.

Детальное описание этих шагов приводится в декодере.

Таблицы B.11, B.12, B.13 используются для квантования усиления

Т а б л и ц а B.11 — Таблица квантования для усиления адаптивной книги шифров

Индекс	*Cba_gain_quant*
0	0.1622
1	0.2542
2	0.3285
3	0.3900
4	0.4457
5	0.4952
6	0.5425
7	0.5887
8	0.6341
9	0.6783
10	0.7227
11	0.7664
12	0.8104
13	0.8556
14	0.9005
15	0.9487

Индекс	*Cba_gain_quant*
16	0.9989
17	1.0539
18	1.1183
19	1.1933
20	1.2877
21	1.4136
22	1.5842
23	1.8559
24	2.3603
25	3.8348
26	7.6697
27	15.339
28	30.679
29	61.357
30	122.71
31	245.43

Таблица В.12 — Таблица квантования для усиления фиксированной книги шифров

Индекс	*Cbf_gain_quant*	Индекс	*Cbf_gain_quant*
0	2.4726	16	82.3374
1	3.1895	17	95.3755
2	4.2182	18	109.8997
3	5.6228	19	126.3037
4	7.3781	20	144.3995
5	9.5300	21	165.5142
6	12.1013	22	190.9742
7	15.2262	23	220.6299
8	19.0319	24	258.2699
9	23.6342	25	305.5086
10	29.1562	26	368.5894
11	35.3606	27	453.5156
12	42.8301	28	573.6164
13	51.1987	29	801.6422
14	60.6440	30	9999.9
15	70.9884	31	.....

Таблица В.13 — Таблица квантования для разностностного усиления фиксированной книги шифров

Индекс	*Cbf_gain_quant*	Индекс	*Cbf_gain_quant*
0	0.2500	16	1.3356
1	0.5378	17	1.8869
2	0.7795	18	4.2000
3	1.0230	19	9999.9

В.9.2 Мультиимпульсное возбуждение

В.9.2.1 Описание инструмента

Для кодера *CELP*, базирующегося на мультиимпульсном возбуждении, выходом анализа возбуждения являются *signal_mode*, *rmc_index*, *shape_delay*[], *shape_positions*[], *shape_signs*[] и *gain_index*[]. Коэффициенты формы и усиления генерируют каждый подфрейм. Вектор *shape_delay*[] содержит задержку адаптивной книги шифров для каждого подфрейма, в то время как векторы *shape_positions*[] и *shape_signs*[] содержат позиции импульса и знаки соответственно. Усиление адаптивной книги шифров и мультиимпульсное усиление для каждого подфрейма представляют собой вектор, квантованный и сохраненный в векторе *gain_index*[].

В.9.2.2 Определения

Вход

PP_inputSignal[]: Этот массив содержит предобработанный входной сигнал и имеет размерность *sfrm_size*

int_Qlpc_coefficients[]: Этот массив содержит коэффициенты интерполированного и квантованного *LPC*

Wnum_coeff[]: Этот массив содержит взвешенные коэффициенты фильтра числителя

Wden_coeff[]: Этот массив содержит взвешенные коэффициенты фильтра знаменателя

Выход

signal_mode: Это поле содержит флажок голосовой/неголосовой

rmc_index: Это поле определяет индекс для мощности сигнала

shape_delay[]: Этот массив имеет размерность *nrof_subframes*. Он содержит задержку книги шифров для адаптивной и фиксированной книг шифров

shape_positions[]: Этот массив имеет размерность *nrof_subframes*. Он содержит позиции импульса

shape_signs[]: Этот массив имеет размерность *nrof_subframes*. Он содержит знаки импульса

gain_index[]: Этот массив имеет размерность *nrof_subframes*. Он содержит квантованное усиление вектора для адаптивной книги шифров и мультиимпульса

Конфигурация

frame_size: Это поле указывает число отсчетов в одном фрейме

sbfrm_size: Это поле указывает число отсчетов в одном подфрейме

nrof_subframes: Это поле указывает число подфреймов в одном фрейме

lpc_order: Это поле указывает порядок *LPC*

В.9.2.3 Процесс кодирования

Для кодера режима II сигнал возбуждения извлекается и кодируется на основе анализа через синтез с тремя шагами, поиска адаптивной книги шифров для периодического компонента, поиска фиксированной книги шифров для непериодического компонента и квантования усиления для каждого компонента. Целевой сигнал получают, вычитая реакцию на нулевой вход фильтров синтеза и перцепционного взвешивания из взвешенного входного речевого сигнала.

Квантование энергии фрейма

Среднеквадратичное (*rms*) значение отсчетов входа подфрейма вычисляется в последнем подфрейме. Значение *rms* скалярно-квантованное по шкале *μ-law*. Значения *rms* других подфреймов получают линейной интерполяцией квантованных значений *rms* в последнем подфрейме текущего и предыдущего фреймов. Квантованные значения *rms* используются для нормализации усиления.

Оценка шага без обратной связи и решение о режиме

Процедуры разомкнутой (без обратной связи) оценки шага и решения о режиме выполняются совместно каждый интервал анализа 10 мс, используя перцепционный взвешенный сигнал. Оценка шага выбирается, чтобы минимизировать квадратичную ошибку между текущим и предыдущим блоками взвешенных входных отсчетов. Усиление предсказания шага вычисляется для каждого интервала анализа. Каждый фрейм классифицируется в один из четырех режимов, основываясь на среднем усилении предсказания шага. Режимы 0 и 1 соответствуют неголосовому и переходному фреймам соответственно. Режимы 2 и 3 соответствуют голосовым фреймам, и у последнего более высокая периодичность, чем у предыдущего. Если длина подфрейма равна 5 мс, тот же самый шаг оценки используется в двух подфреймах для поиска шага с обратной связью.

Кодирование сигнала возбуждения

Сигнал возбуждения представлен линейной комбинацией адаптивного вектора кода и фиксированного вектора кода, масштабированных их соответствующими усилениями. Каждый компонент сигнала возбуждения последовательно выбирается процедурой поиска «анализа через синтез» так, чтобы перцепционная взвешенная ошибка между входным сигналом и восстановленным сигналом была минимальна. Параметрами адаптивного вектора кода являются задержка с обратной связью и усиление. Задержка с обратной связью выбирается в диапазоне, близком оценочной задержке без обратной связи. Адаптивный вектор кода генерируется из блока прошлых отсчетов сигнала возбуждения с выбранной задержкой с обратной связью. Фиксированный вектор кода содержит несколько ненулевых импульсов. Позиции импульсов ограничены в алгебраической структуре. Таблица ограничения позиций импульса устанавливается исходя из параметров. Чтобы улучшить производительность, после определения многочисленных наборов кандидатов позиций импульса выполняется комбинаторный поиск между кандидатами позиции импульса и индексом амплитуды импульса. Усиления для адаптивного и для мультиимпульсного вектора кода нормализуются и векторно квантуются. Коэффициент нормализации вычисляется из квантованных коэффициентов *LP*, квантованного *rms* подфрейма и *rms* сигнала возбуждения. Книга кодов усиления изменяется в соответствии с режимом.

Поиск адаптивной книги шифров

Для каждого подфрейма оптимальная задержка определяется через анализ с обратной связью, так что среднеквадратичная ошибка между целевым сигналом $x(n)$ и взвешенным синтезируемым сигналом $y_t(n)$ адаптивного вектора кода минимальна

$$E_t = \sum_{n=0}^{N-1} (x(n) - g_t\, y_t(n))^2, \qquad t_{op} - 8 < t < t_{op} + 8,$$

где t_{op} — задержка без обратной связи, определенная в анализе шага без обратной связи.

g_t является оптимальным усилением следующим образом

$$g_t = \frac{\sum_{n=0}^{N-1} x(n)\, y_t(n)}{\sum_{n=0}^{N-1} y_t^2(n)}.$$

Оптимальная задержка шага закодирована 8 битами, основываясь на соотношении между *shape_indices*[0] и задержкой (см. процесс декодирования).

Поиск фиксированной книги шифров

Вектор фиксированной книги шифров представлен позицией импульса и амплитудами импульса. Позиции импульса и амплитуды разыскиваются, чтобы минимизировать среднеквадратичную ошибку между целевым сигналом и взвешенным синтезируемым сигналом.

$$E_k = \sum_{n=0}^{N-1} (x(n) - g_t\, y_t(n) - g_k\, z_k(n))^2,$$

где g_t является оптимальным усилением следующим образом:

$$g_k = \frac{\sum_{n=0}^{N-1} (x(n) - g_t\, y_t(n))\, z_k(n)}{\sum_{n=0}^{N-1} z_k^2(n)},$$

где k указывает возможную комбинацию *shape_indices*[1] и *shape_indices*[2] (см. 4.2). Вектор фиксированной книги шифров создается из позиций импульса и амплитуд импульса (см. 4.2). Взвешенный сигнал $z_k(n)$ вычисляется, пропуская вектор фиксированной книги шифров через фильтр синтеза *LP 1/A(z)* и фильтр перцепционного взвешивания *W(z)*.

Квантование усиления

Усиления для адаптивного вектора кода и фиксированного вектора кода нормализуются остаточной энергией предсказания *rs* и скалярно квантуются. Остаточная энергии *rs* вычисляется на основе энергии фрейма и коэффициентов отражения. Энергия фрейма вычисляется каждый подфрейм как среднеквадратичное (*RMC*) значение и квантуется в домене *μ-law*. Коэффициенты отражения *k(i)* конвертируются из интерполированных коэффициентов *LPCs int_Qlpc_coefficients[]*. Следовательно остаточная энергия предсказания (*rs*) подфрейма равна

$$rs = sbfrm_len \cdot q_amp^2 \cdot \prod_{i=1}^{N_p} (1 - k^2 i)),$$

где N_p — порядок анализа *LP*, *q_amp* — квантованная амплитуда *RMC*, а *sbfrm_len* — длина подфрейма.

Квантование усиления выполняется путем поиска пары коэффициентов усиления, которые минимизируют ошибку между целевым и взвешенным синтезируемым сигналом. Ошибка err_g, обусловленная квантованием усиления дается выражением:

$$err_g = \sum_{i=0}^{sbfrm_size-1} \left(\sqrt{\frac{rs}{pow_{AC}}} \cdot AdaptGainCB[m] \cdot ac_syn(i) \right.$$

$$\left. + \sqrt{\frac{rs}{pow_{SC}}} \cdot FixedGainCB[n] \cdot sc_syn(i) - t\arg(i) \right)^2$$

$$pow_ac = \sum_{i=0}^{sbfrm_len-1} ac_ex^2(i)$$

$$pow_sc = \sum_{i=0}^{sbfrm_len-1} sc_ex^2(i),$$

где *m* и *n* — индексы, *ac_ex(i)* — выбранный адаптивный вектор кода, *ac_syn(i)* — сигнал, который синтезируется из *ac_ex(i)* и перцепционно взвешивается, *sc_ex(i)* — выбранный стохастический вектор кода, и *sc_syn(i)* — сигнал, который синтезируется из *sc_ex(i)* и перцепционно взвешивается.

Пара индексов *n* и *m*, которые минимизируют ошибку err_g, выбирается и сохраняется в *gain_indices[]*. Затем квантованное адаптивное усиление *qacg*, квантованное стохастическое *qscg* и сигнал возбуждения *decoded_excitation(i)* вычисляются следующим образом.

$$decoded_excitation(i) = qacg \cdot ac_ex(i) + gscg \cdot sc_ex(i) \qquad (0 \le i \le sbfrm_Len - 1)$$

$$qacg = \sqrt{\frac{rs}{pow_ac}} \cdot AdaptGainCB[m]$$

$$qacg = \sqrt{\frac{rs}{pow_sc}} \cdot FixedGainCB[n]$$

В.9.3 Мультиимпульсное возбуждение с масштабируемой битовой скоростью

В.9.3.1 Описание инструмента

Выходом анализа возбуждения расширения являются *shape_enh_positions*[], *shape_enh_signs*[] и *gain_enh_index*[]. Индексы формы и усиления генерируются каждый подфрейм. Векторы *shape_enh_positions*[] и *shape_enh_signs*[] содержат позиции импульса расширения и знаки соответственно. Мультиимпульсное усиление расширения для каждого подфрейма является скалярной величиной, квантованной и сохраненной в векторе *gain_index*[].

В.9.3.2 Определения

Вход

PP_inputSignal[]: Этот массив содержит предобработанный входной сигнал и имеет размерность *sfrm_size*

int_Qlpc_coefficients[]: Этот массив содержит интерполированные и квантованные коэффициенты *LPC*

Wnum_coeff[]: Этот массив содержит взвешенные коэффициенты фильтра числителя

Wden_coeff[]: Этот массив содержит взвешенные коэффициенты фильтра знаменателя

Выход

shape_enh_positions[]: Этот массив имеет размерность *nrof_subframes*. Он содержит позиции импульса

shape_enh_signs[]: Этот массив имеет размерность *nrof_subframes*. Он содержит знаки импульса

gain_enh_index[]: Этот массив имеет размерность *nrof_subframes*. Он содержит квантованные усиления вектора для адаптивной книги шифров и мультиимпульса

Конфигурация

n_lag_candidates: Это поле указывает число кандидатов задержки

frame_size: Это поле указывает число отсчетов в одном фрейме

sbfrm_size: Это поле указывает число отсчетов в одном подфрейме

nrof_subframes: Это поле указывает число подфреймов в одном фрейме.

lpc_order: Это поле указывает порядок *LPC*

В.9.3.3 Процесс кодирования

Вектор фиксированной книги шифров представлен позицией импульса и амплитудами импульса. Позиции и амплитуды импульсов разыскиваются, чтобы минимизировать среднеквадратичную ошибку между целевым сигналом и взвешенным синтезированным сигналом.

$$E_k = \sum_{n=0}^{N-1} (x(n) - g_t\, y_t(n) - g_k\, z_k(n))^2,$$

где g_t является оптимальным усилением следующим образом:

$$g_k = \frac{\sum_{n=0}^{N-1} (x(n) - g_t\, y_t(n))\, z_k(n)}{\sum_{n=0}^{N-1} z_k^2(n)},$$

где k указывает возможную комбинацию *shape_enh_positions* и *shape_enh_signs* (см. 4.2). Вектор фиксированной книги шифров создается из позиций импульса и амплитуд импульса (см. 4.2). Взвешенный сигнал $z_k(n)$ вычисляется путем фильтрации, пропуская вектор фиксированной книги шифров через фильтр синтеза *LP* $1/A(z)$ и фильтр перцепционного взвешивания $W(z)$.

В.9.4 Мультиимпульсное возбуждение для инструмента расширения полосы пропускания

В.9.4.1 Описание инструмента

Выходом анализа возбуждения в инструменте расширения полосы пропускания являются *shape_bws_delay*[], *shape_bws_positions*[], *shape_bws_signs*[] и *gain_bws_index*[]. Индексы формы и усиления генерируются каждый подфрейм. Вектор *shape_bws_delay*[] содержит задержку адаптивной книги шифров для каждого подфрейма, в то время как векторы *shape_bws_positions*[] и *shape_bws_signs*[] содержат позиции и знак импульса соответственно. Усиление адаптивной книги шифров и мультиимпульсное усиление для каждого подфрейма являются вектором, квантованным и сохраненным в векторе *gain_bws_index*[].

В.9.4.2 Определения

Вход

PP_inputSignal[]: Этот массив содержит предобработанный входной сигнал и имеет размерность *sfrm_size*

int_Qlpc_coefficients[]: Этот массив содержит интерполированные и квантованные коэффициенты *LPC*

Wnum_coeff[]: Этот массив содержит коэффициенты фильтра взвешивания числителя

Wden_coeff[]: Этот массив содержит коэффициенты фильтра взвешивания знаменателя

Выход

shape_bws_delay[]: Этот массив имеет размерность *nrof_subframes_bws*. Он содержит задержку книги шифров для адаптивных и фиксированных книг шифров

shape_bws_positions[]: Этот массив имеет размерность *nrof_subframes_bws*. Он содержит позиции импульса

shape_bws_signs[]: Этот массив имеет размерность *nrof_subframes_bws*, содержит знаки импульса

gain_bws_index[]: Этот массив имеет размерность *nrof_subframes_bws*. Он содержит квантованное усиление вектора для адаптивной книги шифров и мультиимпульсной

Конфигурация

n_lag_candidates: Это поле указывает число кандидатов задержки

frame_size: Это поле указывает число отсчетов в одном фрейме

sbfrm_size: Это поле указывает число отсчетов в одном подфрейме в инструменте расширения полосы пропускания

nrof_subframes_bws: Это поле указывает число подфреймов в инструменте расширения полосы пропускания

lpc_order: Это поле указывает порядок *LPC*

В.9.4.3 Процесс кодирования

Для инструмента расширения полосы пропускания сигнал возбуждения извлекается и кодируется на основе анализа через синтез с тремя шагами поиском адаптивной книги шифров для периодического компонента, поиском фиксированной книги шифров для непериодического компонента и квантованием усиления для каждого компонента. Целевой сигнал получают, вычитая реакцию фильтров синтеза и перцепционного взвешивания на нулевой входной сигнал из взвешенного входного речевого сигнала.

Квантование энергии фрейма

Используется то же самое среднеквадратичное (*rms*) значение, как для *CELP* с частотой дискретизации 8 кГц.

Оценка шага без обратной связи и решение о режиме

Оценка шага без обратной связи выполняется путем преобразования задержки шага с частотой дискретизации 8 кГц (см. процесс декодирования). Используется тот же самый режим, как у *CELP* частоты дискретизации 8 кГц.

Кодирование сигнала возбуждения

Сигнал возбуждения представлен линейной комбинацией адаптивного вектора кода и двух фиксированных векторов кода, масштабированных их соответствующими усилениями. Задержка шага выбирается в диапазоне около предполагаемой задержки шага без обратной связи. Один из двух фиксированных векторов кода получают путем расширения частоты дискретизации фиксированного вектора кода, используемой в кодере частоты дискретизации 8 кГц. Другой фиксированный вектор кода определяется процедурой поиска «анализ через синтез».

Поиск адаптивной книги шифров

Для каждого подфрейма оптимальная задержка определяется посредством анализа с обратной связью так, что среднеквадратичная ошибка между целевым сигналом $x(n)$ и взвешенным синтезированным сигналом $y_t(n)$ адаптивного вектора кода минимизируется.

$$E_t = \sum_{n=0}^{N-1} (x(n) - g_t\, y_t(n))^2, \quad t_{op} - 8 < t < t_{op} + 8,$$

где t_{op} — задержка без обратной связи, определенная в анализе шага без обратной связи. g_t является оптимальным усилением следующим образом:

$$g_t = \frac{\sum_{n=0}^{N-1} (x(n)_t\, y_t(n)}{\sum_{n=0}^{N-1} y_t^2(n)}.$$

Различие между оптимальной задержкой шага и задержкой шага без обратной связи кодируется 3 битами, основываясь на соотношении между *shape_bws_delay* и дифференциальной задержкой (см. процесс декодирования).

Поиск фиксированной книги шифров 1

Фиксированный вектор кода 1 получают увеличением частоты дискретизации фиксированного вектора кода, используемого в кодере с частотой дискретизации 8 кГц (см. 5.7.4.3.4).

Поиск фиксированной книги шифров 2

Фиксированный вектор книги шифров представлен позицией импульса и амплитудами импульса. Позиции и амплитуды импульса разыскиваются так, чтобы минимизировать среднеквадратичную ошибку между целевым сигналом и взвешенным синтезированным сигналом

$$E_k = \sum_{n=0}^{N-1} (x(n) - g_t\, y_t(n) - g_k\, z_k(n))^2,$$

где g_t является оптимальным усилением следующим образом:

$$g_k = \frac{\sum_{n=0}^{N-1} (x(n) - g_t\, y_t(n))\, z_k(n)}{\sum_{n=0}^{N-1} z_k^2(n)},$$

где k указывает возможную комбинацию *shape_bws_positions* и *shape_bws_signs* (см. 4.2). Вектор фиксированной книги шифров создается из позиций импульса и амплитуд импульса (см. 4.2). Взвешенный сигнал $z_k(n)$ подсчитывается, пропуская вектор фиксированной книги шифров через фильтр синтеза *LP* 1/*A (z)*, и перцепционный взвешивающий фильтр *W(z)*.

Квантование усиления

Усиления для адаптивного вектора кода и двух фиксированных векторов кода нормализуются остаточной энергией предсказания *rs* и квантуются. Остаточная энергия *rs* вычисляется на основании энергии фрейма и коэффициентов отражения. Энергия фрейма подсчитывается каждый подфрейм как среднеквадратичное (*RMC*) значение и квантуется в домене закона *μ-law*. Коэффициенты отражения *k(i)* преобразуются из интерполированных *LPCs int_Qlpc_coefficients*[]. Следовательно, остаточная энергия предсказания (*rs*) подфрейма равна

$$rs = sbfrm_len \cdot q_amp^2 \cdot \prod_{i=1}^{N_p} (1 - k^2 i)),$$

где N_p является порядком анализа *LP*, *q_amp* — квантованная амплитуда *RMC* и *sbfrm_len* — длина подфрейма.

Усиления для адаптивного вектора кода и фиксированного вектора кода 2 являются векторно квантованными. Усиление для фиксированного вектора кода 1 является скалярно квантованным. Эти операции квантования достигаются минимизацией перцепционно взвешенного искажения (с обратной связью).

B.10 Мультиплексор потока бит *CELP*

B.10.1 Описание инструмента

Инструмент мультиплексор потока бит *CELP* мультиплексирует фрейм в поток бит.

B.10.2 Определения

Все элементы потока бит и переменные поддержки были определены в 5.3 и 5.4.

B.10.3 Процесс кодирования

Параметры кодируются в поток бит согласно синтаксису, описанному в разделе 3.

B.11 Инструмент сжатия тишины *CELP*

B.11.1 Модуль *VAD*

Модуль *VAD* принимает решение о том, является ли фрейм фреймом паузы в разговоре или фреймом разговора, основываясь на том, насколько изменяются характеристики входного сигнала. Характеристики представлены четырьмя параметрами: энергиями полной полосы и нижней полосы, *LSPs* и частотой перехода через нуль фрейма входного сигнала. Временное решение принимается через каждые 80 отсчетов, и конечное решение для фрейма принимается на основе временных решений с ограничением затягивания. Для широкополосного режима параметры характеристик вычисляются исходя из входной речи с пониженной дискретизацией от 16 кГц до 8 кГц.

B.11.1.1 Определения

Вход

PP_InputSignal[] Этот массив содержит предобработанный речевой сигнал. Размерность равна *frame_size*

Выход

VAD_flag: Это поле содержит флажок *VAD* (см. таблицу 103).

Элементы справки, используемые в модуле VAD:

lpc_order: порядок *LP*

sbfrm_size: число отсчетов в подфрейме

frame_size: число отсчетов во фрейме

n_subframe: число подфреймов во фрейме

B.11.1.2 Уменьшение дискретизации для широкополосного случая

Сигнал *s_vad*[], используемый в модуле *VAD*, генерируется предварительной обработкой входного сигнала тем же способом, как это описано в B.4. Когда частота дискретизации составляет 16 кГц, частота дискретизации входного сигнала понижается до 8 кГц после предварительной обработки.

B.11.1.3 Вычисление параметров

LSPs входного сигнала, *lsp*[] вычисляются из коэффициентов *LPCs lpc_coefficients*[], которые даются модулем взвешивания *MPEG-4 CELP*, описанным в B.8. Энергия полной полосы *P*, энергия нижней полосы P_l между 0 и 1 кГц и частота перехода через нуль *Z* вычисляются следующим образом:

$P = 10log_{10}R\ [0]$

$P_l = 10log_{10}h^T\ Rh$

$$Z = \frac{1}{80}\sum_{i=0}^{80}\left|sign\left[s_vad[i]\right] - sign[s_vad[i-1]\right|,$$

где h — вектор импульсной передаточной функции фильтра *FIR* с частотой среза 1 кГц. $R[0]$ — первый коэффициент автокорреляции и R — матрица автокорреляции *Toeplitz* (Теплица) с коэффициентами автокорреляции в каждой диагонали. Эти параметры подсчитываются каждые 10 мс. Их средние значения обновляются следующим образом:

$\overline{P} = a\overline{P} + (1-a)P$

$\overline{P} = b\overline{P} + (1-b)P_l$

$\overline{Z} = \tilde{n}\overline{Z} + (1-\tilde{n})Z$

$\overline{L}sp[i] = d\overline{L}sp[i] + (1-d)Lsp[i],\ \ i = 1,....lpc_order,$

где $a = 0{,}995$, $b = 0{,}995$, $c = 0{,}998$ и $d = 0{,}75$. Каждые 80 отсчетов оцениваются следующие дифференциальные параметры, чтобы принять временное решение, характеризуется ли входной сигнал как разговорный или как пауза в разговоре:

$\Delta P = \overline{P} - P$

$\Delta P_l = \overline{P}_l - P_l$

$\Delta Z = \overline{Z} - Z$

$$\Delta Lsp = \sum_{i=1}^{lpc_onder}\left[\overline{L}sp[i] - Lsp[i]\right]^2,\quad i = 1,...lpc_order$$

В.11.1.4 Временное решение о речевой активности

Если удовлетворяется любое из следующих неравенств, флажок временной голосовой активности, *vad_flag_sub[i]* для

$$i = 0,...,\frac{frm_size[samples]}{80[samples]}$$ устанавливается в 1.

if (

$\Delta Lsp > 0{,}0009$ *or*

$\Delta Lsp > 0{,}00175 * \Delta Z + 0{,}00085$ *or*

$\Delta Lsp > -0{,}00455 * \Delta Z + 0{,}00116$ *or*

$\Delta P < -0{,}47$ *or*

$\Delta P < -2{,}5 * \Delta Z - 0{,}5$ *or*

$\Delta P < 2{,}0 * \Delta Z - 0{,}6$ *or*

$\Delta P < 2{,}5 * \Delta Z - 0{,}7$ *or*

$\Delta P < -2{,}91 * \Delta Z - 0{,}482$ *or*

$\Delta P < 880{,}0 * \Delta Lsp - 1{,}22$ *or*

$\Delta P_l < 1400{,}0 * \Delta Lsp - 1{,}55$ *or*

$\Delta P_l > 0{,}929 * \Delta P + 0{,}114$ *or*

$\Delta P_l < -1{,}5 * \Delta P - 0{,}9$ *or*

$\Delta P_l < 0{,}714 * \Delta P - 0{,}214$

)

{

vad_flag_sub[i] = 1;

} else {

vad_flag_sub[i] = 0;

}

В.11.1.5 Решение о речевой активности фрейма

Флажок голосовой активности фрейма *vad_flag* определяется на основании временного флажка *vad_flag_sub[]*, что выполняется в соответствующем фрейме следующим образом:

vad_flag = vad_flag_sub [0] ∪ *vad_flag_sub [1]* ∪ ...∪ *vad_flag_sub [L–1]*,

где «∪» замещает логическое ИЛИ.

В.11.1.6 Затягивание

Затягивание применяется к конечному решению *VAD*, *VAD_flag*, после переключения из речевого фрейма в фрейм с отсутствием речи:

$$VAD_flag = \begin{cases} 1, first\ 80\ m\ \text{sec}\ after\ the\ active - voice\ period\ (vad_flag = 1) \\ vad_flag, \qquad otherwise \end{cases}.$$

Когда период речевой активности (*vad_flag*=1) короче, чем 80 мс, *VAD_flag* всегда равен *vad_flag*.

В.11.2 Модуль *DTX*

Модуль *DTX* обнаруживает фреймы, в которых входные характеристики изменяются во время фреймов паузы в разговоре. В первом фрейме во время каждого периода паузы в разговоре и во фрейме, где обнаружено изменение, модуль *DTX* извлекает параметры; энергию фрейма и *LSPs* речевого входа и кодирует эти параметры. Есть три режима *DTX*; *DTX_flag* = 0, 1 и 2 в зависимости от того, какая информация передается. Когда обнаружено изменение в *LSPs* (*DTX_flag* = 1), закодированные параметры *LSP* и *RMC* а также *TX_flag* передаются как информация *HR-SID*. Когда обнаружено изменение среднеквадратичного значения *RMS* (энергия фрейма) (*DTX_flag* = 2), передаются только закодированный параметр *RMS* и *TX_flag* как информация *LR-SID*. Иначе передается только *TX_flag*.

В.11.2.1 Определения

Вход

PP_InputSignal[] Этот массив содержит предобработанный речевой сигнал. Размерность равна *frame_size*

TX_flag Это поле содержит режим передачи (см. таблицу 103)

Выход

VAD_flag Это поле содержит флажок *VAD* (см. таблицу 103).

Элементы справки, используемые в модуле *DTX*:

lpc_order: порядок *LP*

frame_size: число отсчетов во фрейме

n_subframe: число подфреймов во фрейме

В.11.2.2 Анализ *LP*

Используются тот же самый анализ *LP* и преобразование *LPC*-to-*LSP*, как описанные в подпункте В.5. Это дает неквантованные *LSPs lsp*[][].

В.11.2.3 Усреднение *LSP*

Среднее *LSP lsp_av*[] вычисляется из неквантованных *LSPs lsp*[][] следующим образом:

$$lsp_av[i] = (1/L)\sum_{j=0}^{L-1} lsp[j][i], \quad i = 0,\ldots lpc_order - 1,$$

где *lsp*[*j*][] являются неквантованными *LSPs* в *j-th* новом фрейме, которые вычислены из неквантованных *LPCs lpc_coefficients*[], описанных в В.6. *L* — число фреймов за 80 мс.

В.11.2.4 Вычисление *RMS*

RMS входного сигнала вычисляется способом, идентичным описанному в В.9. Это вычисление дает неквантованное *RMS xnorm*[] в каждом подфрейме.

В.11.2.5 Усреднение *RMS*

Среднее *RMS* входного сигнала *xnorm_av*[] вычисляется из неквантованного *RMC xnorm*[] следующим образом:

$$xnorm_av = (1/M)\sum_{j=0}^{M-1} xnorm[j],$$

где *M* является числом подфреймов за 80 мс. Кроме того *xnorm_av_end* является *xnorm_av[n_subframe–1]*, где *n_subframe* — это число подфреймов во фрейме.

В.11.2.6 Обнаружение изменения характеристик

Любое изменение в характеристиках обнаруживается на основе изменения энергии фрейма и спектра, вычисленного исходя из входного сигнала следующим образом:

DTX_flag = 0

$$f(\left|20\log_{10} xnorm_sid - 29\log_{10} xnorm_end\right| > D_{xnorm}\ dB)\ DTX_flag = 2$$

$$f(\sum_{i=1}^{lpc_onder} [lsp_sid[i] - lsp_av[i]]^2 > D_{lsp})\ DTX_flag = 1,$$

где *LSPs* нормализованы к диапазону от 0 до 1. Порог D_{xnorm} переключается согласно *xnorm_sid*, как показано в таблице В.14. Порог D_{lsp} изменяется согласно частоте дискретизации; 0,002 для 8 кГц и 0,0015 для 16 кГц.

lsp_av_sid и *xnorm_sid* являются *lsp_av* и *xnorm_av_end* последнего фрейма *SID* соответственно. Есть минимальный период, где обнаружение не производится. Длина этого периода обычно 20 мс, но во время первых 40 мс периода паузы в разговоре — это 0 мс.

Таблица В.14 — Соотношения между D_{xnorm} и *xnorm_sid*

$20log_{10}$ *xnorm_sid, dB*	D_{xnorm}, *dB*
< 1,0:	6,0
1,0 ~ 5,0	4,0
5,0 ~ 9,0	3,0
9,0 ~ 12,0	2,5
12,0 <	2,0

В.11.2.7 Кодирование параметров

Среднее значение *LSP lsp_av*[] кодируется, используя тот же процесс, который описан в В.6 с исключениями, описанными в 5.9.3. Кодирование среднего *RMS xnorm_av*[] идентично кодированию, описанному в В.9, за исключением того что параметры *μ-law* независимы от режима сигнала и установлены как *rms_max* = 7932 и *mu_law* = 1024.

В.11.2.8 Локальный декодер *CNG*

Локальное декодирование *CNG* выполняется для того, чтобы обновить буфера для фильтра синтеза *LP*. Обработка идентична процессу декодирования в декодере.

Библиография

[1] ИСО/МЭК 14496–3:2009 Информационные технологии. Кодирование аудиовизуальных объектов. Часть 3. Аудио (ИСО/МЭК 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 *Е. В. Вахрушева*
Технический редактор *В. Н. Прусакова*
Корректор *С. И. Фирсова*
Компьютерная верстка *З. И. Мартыновой*

Сдано в набор 03.10.2013. Подписано в печать 05.03.2014. Формат 60×84 1/8. Бумага офсетная. Гарнитура Ариал. Печать офсетная. Усл. печ. л. 11,63. Уч.-изд. л. 11,35. Тираж 58 экз. Зак. 1440

ФГУП «СТАНДАРТИНФОРМ», 123995 Москва, Гранатный пер., 4.
gost.e-ecolog.ru gost.e-ecolog.ru
Набрано и отпечатано в Калужской типографии стандартов, 248021 Калуга, ул. Московская, 256.

gost.e-ecolog.ru