ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 53556.1— 2012

Звуковое вещание цифровое

КОДИРОВАНИЕ СИГНАЛОВ ЗВУКОВОГО ВЕЩАНИЯ С СОКРАЩЕНИЕМ ИЗБЫТОЧНОСТИ ДЛЯ ПЕРЕДАЧИ ПО ЦИФРОВЫМ КАНАЛАМ СВЯЗИ. ЧАСТЬ 3 (MPEG–4 AUDIO)

Общие требования к кодированию

Издание официальное

Москва
Стандартинформ
2014

gost.e-ecolog.ru

Предисловие

Цели и принципы стандартизации в Российской Федерации установлены Федеральным законом от 27 декабря 2002 г. № 184-ФЗ «О техническом регулировании», а правила применения национальных стандартов Российской Федерации – ГОСТ Р 1.0 – 2004 «Стандартизация в Российской Федерации. Основные положения»

Сведения о стандарте

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

2 ВНЕСЕН Техническим комитетом по стандартизации № 480 «Связь»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 20 ноября 2012 г. № 940-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 (ИСО/МЭК14496-3:2009 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 ближайшем выпуске 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
2 Нормативные ссылки 1
3 Термины, определения, символы и сокращения 2
3.1 Термины и определения 2
3.2 Символы и сокращения 3
3.3 Метод описания синтаксиса потока битов 4
3.4 Арифметические типы данных 5
3.5 Технический обзор 5
3.6 Интерфейс *MPEG*–4 Системы 20
3.7 Транспортный поток *MPEG*–4 Аудио 37
3.8 Защита от ошибок 50
Приложение А (справочное) Форматы обмена аудиофайлами 68
Приложение Б (справочное) Инструмент защиты от ошибок 72
Библиография 91

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)

Общие требования к кодированию

Digital sound broadcasting.Coding of sound broadcasting signals with redundancy reduction for transfer on digital communication channels. Part III (MPEG-4 audio). General requirements for coding

Дата введения – 2013–09–01

1 Область применения

Для достижения лучшего качества звучания на низких скоростях передачи данных (менее 64 Кбит/с на канал) вводятся три дополнительных частоты дискретизации (*Fs*) - 16 кГц, 22,05 кГц и 24 кГц. Это позволяет поддерживать звуковые сигналы с полосами частот 7,5 кГц, 10,3 кГц и 11,25 кГц соответственно, и обеспечивать передачу сигналов звукового вещания по стандартным каналам (ГОСТ Р 52742–2007 и ГОСТ Р 53537–2009).

Синтаксис, семантика и методы кодирования ГОСТ Р 54711–2011 сохраняются в данном стандарте, за исключением определения поля частоты дискретизации, поля скорости передачи и таблиц распределения бит. Новые значения действительны, если бит *ID* в заголовке ГОСТ Р 54711–2011 равняется нулю. Для получения лучшей производительности звуковой системы параметры психоакустической модели, используемой в кодере, должны быть изменены соответственно.

Вход кодера и выход декодера совместимы с существующими стандартами ИКМ, такими как ГОСТ 27667–88, ГОСТ 28376–89.

Показатели, определенные настоящим документом, являются базовыми для профессиональной и бытовой аппаратуры – проигрывателей компакт-дисков, усилителей сигналов звуковой частоты и другого оборудования класса *Hi-Fi.*

Универсальная и совместимая многоканальная звуковая система применима для спутникового и наземного телевизионного вещания, цифрового звукового вещания (наземного и спутникового), так же как и для других носителей, например:

CATV Кабельное телевидение;
CDAD Кабельное цифровое звуковое вещание;
DAB Широковещательная передача цифрового звукового сигнала;
DVD Цифровой универсальный диск;
ENG Электронные новости (включая новости по спутнику);
HDTV Телевидение высокой четкости;
IPC Межличностное общение (видеоконференция, видеотелефон и т. д.);
ISM Интерактивные носители (оптические диски и т. д.).

2 Нормативные ссылки

В настоящем стандарте использованы нормативные ссылки на следующие стандарты:

ГОСТ Р 54711–2011 Звуковое вещание цифровое. Кодирование сигналов звукового вещания с сокращением избыточности для передачи по цифровым каналам связи. MPEG–1 часть III (MPEG–1 audio)

ГОСТ Р 54712–2011 Звуковое вещание цифровое. Кодирование сигналов звукового вещания с сокращением избыточности для передачи по цифровым каналам связи. MPEG–2, часть III (MPEG–2 audio)

Издание официальное

ГОСТ Р 54713–2011 Звуковое вещание цифровое. Кодирование сигналов звукового вещания с сокращением избыточности для передачи по цифровым каналам связи. MPEG–2, часть VII. Усовершенствованное кодирование звука (MPEG–2 AAC)

ГОСТ Р 53556.0–2009 Звуковое вещание цифровое. Кодирование сигналов звукового вещания с сокращением избыточности для передачи по цифровым каналам связи. MPEG–4, часть III (MPEG–4 audio). Основные положения

ГОСТ Р 52742–2007 Каналы и тракты звукового вещания. Типовые структуры. Основные параметры качества. Методы измерений

ГОСТ Р 53537–2009 Звуковое вещание. Основные электрические параметры каналов и трактов студийного качества (с полосой частот 20 ...20000 Гц)

ГОСТ 27667–88 Система цифровая звуковая «Компакт-диск». Параметры

ГОСТ 28376–89 Компакт-диск. Параметры и размеры

Примечание – При пользовании настоящим стандартом целесообразно проверить действие ссылочных стандартов и классификаторов в информационной системе общего пользования – на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет или по ежегодно издаваемому информационному указателю «Национальные стандарты», который опубликован по состоянию на 1 января текущего года, и по соответствующим ежемесячно издаваемым информационным указателям, опубликованным в текущем году. Если ссылочный документ заменен (изменен), то при пользовании настоящим стандартом следует руководствоваться замененным (измененным) документом. Если ссылочный документ отменен без замены, то положение, в котором дана ссылка на него, применяется в части, не затрагивающей эту ссылку.

3 Термины, определения, символы и сокращения

3.1 Термины и определения

В настоящем стандарте применены термины и сокращения с соответствующими определениями, по ГОСТ Р 53556.0–2009, а также следующие термины с соответствующими определениями, не вошедшие в указанный стандарт:

3.1.1 **буфер:** Устройство памяти большой емкости, позволяющее записывать и хранить большой объем данных.

3.1.2 **программа *AAC*:** Основные звуковые каналы, спаренные каналы, канал *lfe* и связанные потоки данных, которые должы быть декодированы и воспроизведены одновременно. Программа может быть задана по умолчанию или с помощью *program_config_element* (). Данные *single_channel_element* (), *channel_pair_element* (), *coupling_channel_element* (), *lfe_channel_element* () или *data_stream_element* () могут сопровождать одну или более программ в любом заданном потоке.

3.1.3 **звуковой блок доступа:** Часть звуковых данных в пределах элементарного потока с возможностью индивидуального доступа.

3.1.4 **звуковой композитный блок:** Часть выходных данных, которую звуковой декодер производит из звуковых блоков доступа.

3.1.5 **абсолютное время:** Время, которому соответствует тот или иной звук; реальное время. Время в партитуре.

3.1.6 **фактический параметр:** Параметр команды.

3.1.7 **адаптивная кодовая книга:** Способ кодирования длительной периодичности сигнала. Входными параметрами кодовой книги являются перекрывающиеся сегменты прошлых возбуждений.

3.1.8 ***API*:** Интерфейс прикладного программирования.

3.1.9 **масштабируемость полосы пропускания:** Возможность менять полосу пропускания сигнала во время передачи.

3.1.10 ***ELD*:** Расширенная низкая задержка.

3.1.11 ***EP*:** Защита от ошибок.

3.1.12 ***R*:** Способность системы противостоять ошибкам.

3.1.13 **возбуждение:** Сигнал возбуждения представляет вход модуля *LPC*.

3.1.14 **межфреймовое предсказание:** Метод предсказания значений в текущем фрейме по значениям в предыдущих фреймах. Используется в *VQ LSP*.

3.1.15 ***LTP*:** Долгосрочное предсказание.

3.1.16 **основные звуковые каналы:** Весь *single_channel_elements* или *channel_pair_elements* в одной программе.

3.1.17 ***MIDI*:** Стандарт цифрового интерфейса музыкальных инструментов.

3.1.18 **смешанный голосовой фрейм:** Речевой сегмент, в котором присутствуют как голосовые, так и неголосовые компоненты.

3.1.19 ***PS*:** Параметрическое стерео.

3.1.20 ***TTSI*:** Интерфейс преобразования текста в речь.

3.1.21 ***VQ*:** Векторное квантование.

3.1.22 ***VXC*:** Векторное кодирование возбуждения. Синоним *CELP*.

3.2 Символы и сокращения

3.2.1 Математические операторы

Математические операторы, используемые в настоящем стандарте, аналогичны используемым в языке программирования *C*. Однако целочисленное деление с усечением и округление определены особым образом. Побитные операторы определяются с учетом представления чисел в дополнительном коде. Нумерация и счетчики циклов обычно начинаются с нуля.

\+ Сложение.

\- Вычитание (как бинарный оператор) или отрицание (как унарный оператор).

++ Инкремент.

\- - Декремент.

* Умножение.

^ Возведение в степень.

/ Целочисленное деление с округлением к меньшему по модулю целому. Например, 7/4 и -7/4 округляются до одного, а -7/4 и 7/-4 округляются до минус одного.

// Целочисленное деление с округлением к ближайшему целому числу. Полуцелые числа округляются в сторону ближайшего большего по модулю числа, если не указано другое. Например 3//2 округляется до двух, а -3//2 округляется до минус двух.

DIV Целочисленное разделение с округлением результата в сторону $-\infty$.

| | Абсолютное значение. $|x| = x$, когда $x > 0$;
$|x| = 0$, когда $x == 0$;
$|x| = -x$, когда $x < 0$.

% Деление с остатком. Операция определена только для положительных чисел.

Sign ( ) Принимает следующие значения: *Sign* (x) = 1, когда $x > 0$;
Sign (x) = 0, когда $x = 0$;
Sign (x) = -1, когда $x < 0$.

NINT ( ) Округление до ближайшего целого. Возвращает самое близкое к вещественному аргументу целочисленное значение. Полуцелые числа округляются в сторону от нуля.

sin Синус.

cos Косинус.

exp Экспонента.

$\sqrt{}$ Квадратный корень.

$\log_{10}$ Логарифм по основанию 10.

$\log_e$ Натуральный логарифм.

$\log_2$ Логарифм по основанию 2.

3.2.2 Логические операторы

II Логическое ИЛИ.

&& Логическое И.

! Логическое НЕ.

3.2.3 Операторы сравнения

\> Больше.

\> = Больше или равно.

< Меньше.

<= Меньше или равно.

== Равно.

! = Не равно.

max [,...,] Максимальное значение.
min [,...,] Минимальное значение.

3.2.4 Побитные операторы

Использование побитных операций подразумевает представление чисел в дополнительном коде.
& Побитное И.
| Побитное ИЛИ.
>> Сдвиг вправо.
<< Сдвиг влево.

3.2.5 Оператор присвоения

= Оператор присвоения.

3.2.6 Мнемоники

Следующие мнемоники подлежат определению для описания различных типов данных, используемых в кодированном потоке битов.

bslbf	Битовая строка, младший бит слева, в соответствии с настоящим стандартом. Битовые строки пишутся, как строка единиц и нулей внутри одинарных кавычек, например, '1000 0001'. Пробелы внутри битовой строки вводятся для удобства чтения и не имеют никакого значения.
L, C, R, LS, RS	Левый, центральный, правый, левый окружения и правый окружения звуковые каналы объемного звучания.
rpchof	Коэффициенты остатка от деления на порождающий полином, сначала следует коэффициент высшего порядка.
uimsbf	Целое число без знака, старший бит первый.
vlclbf	Код с переменной длиной слова, левый бит первый, где «левый» относится к порядку, в котором пишутся коды с переменной длиной.
window	Номер текущего временного интервала в случае *block_type* == 2, $0 \le$ *window* ≤ 2.

В многобайтовых словах старший байт является первым.

3.2.7 Константы

π 3,14159265358...
e 2,71828182845...

3.3 Метод описания синтаксиса потока битов

Для выражения условий присутствия элементов данных используются следующие конструкции, указанные обычным шрифтом:

while (condition) {	Если «истина», то группа элементов данных появляется в потоке данных.
data_element	Это повторяется, пока условие не «ложь».
. . .	
}	
do {	Элемент данных всегда появляется, по крайней мере, один раз.
data_element	Элемент данных повторяется, пока условие не «ложь».
. . .	
} while (condition)	
if (condition) {	Если условие является «истиной», то первая группа элементов данных появляется в потоке данных.
data_element	
. . .	
}	
else {	Если условие не является «истиной», то вторая группа элементов данных появляется в потоке данных.
data_element	
. . .	
}	

for (*expr*1; *expr*2; *expr*3) {
    data_element
    . . .
}

*Expr*1 является инициализирующим выражением цикла. Обычно оно определяет начальное состояние счетчика. *Expr*2 является условием, определяющим проверку перед каждой итерацией цикла. Цикл завершается, когда условие не является «истиной». *Expr*3 является выражением, которое выполняется в конце каждой итерации цикла, обычно оно инкрементирует счетчик.

Группа элементов данных может содержать вложенные условные конструкции. Для компактности скобки {} могут быть опущены, когда следует только один элемент данных.

data_element [ ] — *data_element* [ ] является массивом данных. Количество элементов массива зависит от контекста.

data_element [*n*] — *data_element* [*n*] является (*n*+1)-ым элементом массива данных.

data_element [*m*] [*n*] — *data_element* [*m*] [*n*] является элементом (*m*+1)-ой строки (*n*+1)-го столбца двухмерного массива данных.

data_element [*l*] [*m*] [*n*] — *data_element* [*l*] [*m*] [*n*] является (*l*+1), (*m*+1), (*n*+1)-ым элементом трехмерного массива данных.

data_element [*m...n*] — *data_element* [*m...n*] содержит биты массива *data_element* с *m* по *n* включительно.

3.4 Арифметические типы данных

INT32 32-битное знаковое целое с дополнением к нулю.
INT64 64-битное знаковое целое с дополнением к нулю.

3.5 Технический обзор

3.5.1 Типы звуковых объектов MPEG–4

3.5.1.1 Определение типов звуковых объектов приведено в таблице 1.

Таблица 1 – Определение типов звуковых объектов на основе инструментов/модулей

ID типа объекта	Тип объекта *Audio*	*gain control*	*block switching*	*window shapes - standard*	*window shapes – AAC LD*	*Low Delay Window*	*filterbank - standard*	*filterbank - SSR*	*TNS*	*LTP*	*intensity*	*coupling*	*frequency deomain prediction*	*PNS*	*SIAQ*	*FSS*	*upsampling filter tool*	*quantisation&coding - AAC*	*quantisation&coding – TwinVQ*	*quantisation&coding - BSAC*	*AAC ER Tools*	*ER payload syntax*	*EP Tool 1*	*CELP*	*Silence Compression*	*HVXC*	*SA tools*	*SASBF*	*MIDI*	*HILN*	*TTSI*	*SBR*	*low delay SBR*	*Layer-1*	*Layer-2*	*Layer-3*	*SSC (Transient, Sinusoid)*	*Parametric stereo*				*Integer M/S*			Примечания
0	*Null*																																												
1	*AAC main*		X	X			X		X			X	X	X	X				X																											2
2	*AAC LC*		X	X			X		X			X	X		X				X																											
3	*AAC SSR*	X	X	X				X	X			X	X		X				X																											
4	*AAC LTP*		X	X			X		X	X		X	X		X				X																											2
5	*SBR*																															X													
6	*AAC Scalable*		X	X			X		X	X					X	X	X	X	X																											6
7	*TwinVQ*		X	X			X		X	X					X					X																										
8	*CELP*																								X																				
9	*HVXC*																									X																				
10	(зарезервировано)																																												
11	(зарезервировано)																																												
12	*TTSI*																														X													
13	*Main synthetic*																										X	X	X																3
14	*Wavetable synthesis*																											X	X																4
15	*General MIDI*																												X															

Окончание таблицы 1

ID типа объекта	Тип объекта *Audio*	*gain control*	*block switching*	*window shapes - standard*	*window shapes – AAC LD*	*Low Delay Window*	*filterbank - standard*	*filterbank - SSR*	*TNS*	*LTP*	*intensity*	*coupling*	*frequency deomain prediction*	*PNS*	*SIAQ*	*FSS*	*upsampling filter tool*	*quantisation&coding - AAC*	*quantisation&coding – TwinVQ*	*quantisation&coding - BSAC*	*AAC ER Tools*	*ER payload syntax*	*EP Tool 1*	*CELP*	*Silence Compression*	*HVXC*	*SA tools*	*SASBF*	*MIDI*	*HILN*	*TTSI*	*SBR*	*low delay SBR*	*Layer-1*	*Layer-2*	*Layer-3*	*SSC (Transient, Sinusoid)*	*Parametric stereo*			*Integer M/S*			Примечания
16	*Algorithmic Synthesis and AudioFX*																										X																	
17	*ER AAC LC*		X	X			X		X		X			X	X				X			X	X	X																				
18	(зарезервировано)																																											
19	*ER AAC LTP*		X	X			X		X	X	X			X	X				X			X	X	X																				5
20	*ER AAC scalable*		X	X			X		X		X			X	X	X	X	X	X			X	X	X																				6
21	*ER TwinVQ*		X	X			X		X						X					X			X	X																				
22	*ER BSAC*		X	X			X		X		X			X	X						X		X	X																				
23	*ER AAC LD*				X		X		X	X	X			X	X				X			X	X	X																				
24	*ER CELP*																						X	X	X	X																		
25	*ER HVXC*																						X	X			X	X																
26	*ER HILN*																						X	X						X														
27	*ER Parametric*																						X	X			X	X		X														
28	*SSC*																																				X	X							
29	*PS*																														X						X							
30	*MPEG Surround*																																												
31	*(escape)*																																												
32	*Layer-1*																																	X											
33	*Layer-2*																																		X										
34	*Layer-3*																																			X									
35	*DST*																																					X							
36	*ALS*																																						X						
37	*SLS*																																							X	X	X	X	X	
38	*SLS non-core*																																										X	X	
39	*ER AAC ELD*					X	X		X		X			X	X				X			X	X	X										X										
40	*SMR Simple*																																												
41	*SMR Main*																																												
42–95	(зарезервировано)																																												

Примечания

1 Функция битового поиска обязательна для декодера. Однако функции обнаружения и исправления ошибок являются необязательными.

2 Содержит *AAC LC*.

3 Содержит таблично-волновой синтез, алгоритмический синтез и звуковые эффекты.

4 Содержит спецификацию *General MIDI*.

5 Содержит *ER AAC LC*.

6 Инструмент передискретизации требуется только в комбинации с основным кодером.

3.5.1.2 Описание

3.5.1.2.1 Тип объекта *NULL*

Объект *NULL* обеспечивает возможность подать необработанные ИКМ отсчеты непосредственно в звуковой процессор. Декодирование не применяется, однако дескриптор звуковых объектов используется для определения частоты дискретизации и конфигурации звуковых каналов.

3.5.1.2.2 Тип объекта *AAC - Main*

Объект *AAC Main* очень схож с профилем *AAC Main*. Однако, дополнительно доступен инструмент *PNS*. Ограничения профиля *AAC Main* относительно различных программ и элементов сведения

также относятся к типу объекта *AAC Main*. Все многоканальные возможности *AAC MPEG–2* доступны. Декодер, способный декодировать поток основного объекта *MPEG–4*, также в состоянии анализировать и декодировать поток необработанных данных *AAC MPEG–2*. С другой стороны, не смотря на то, что кодер *AAC MPEG–2* может анализировать полезный поток битов *AAC MPEG–4 Main*, возможна ошибка декодирования из-за использования *PNS*.

3.5.1.2.3 Тип объекта *AAC – Low Complexity (LC)*

Тип объекта *AAC MPEG–4 Low Complexity* является копией профиля пониженной сложности *AAC MPEG–2* с теми же ограничениями, как и для типа объекта *AAC Main*.

3.5.1.2.4 Тип объекта *AAC – Scalable Sampling Rate (SSR)*

Тип объекта *AAC MPEG–4 Scalable* Sampling *Rate* является копией профиля масштабируемой частоты дискретизации *AAC MPEG–2* с теми же ограничениями, как и для типа объекта *AAC Main*.

3.5.1.2.5 Тип объекта *AAC – Long Term Predictor (LTP)*

Тип объекта *AAC MPEG–4 LTP* подобен типу объекта *AAC Main*, однако, предсказатель *AAC MPEG–2* заменен долговременным предсказателем. *LTP* достигает схожего усиления эффективности кодирования, но требует значительно более низкой сложности реализации. Полезный поток бит профиля *AAC MPEG–2 LC* может быть декодирован декодером объекта *LTP AAC MPEG–4* без ограничений.

3.5.1.2.6 Тип объекта *SBR*

Объект *SBR* содержит инструмент *SBR* и может быть объединен с типами звуковых объектов, обозначенными в таблице 2.

Таблица 2 – Типы звуковых объектов, которые могут быть объединены с инструментом *SBR*

Тип звукового объекта	*ID* типа объекта
AAC main	1
AAC LC	2
AAC SSR	3
AAC LTP	4
AAC Scalable	6
ER AAC LC	17
ER AAC LTP	19
ER AAC Scalable	20
ER BSAC	22

3.5.1.2.7 Тип объекта *AAC Scalable*

Объект *AAC Scalable* использует различный синтаксис полезного потока бит для реализации масштабируемости скорости передачи и полосы пропускания. Доступно большое количество масштабируемых комбинаций, включая комбинации с инструментами *TwinVQ* и *CELP*. Однако, поддерживаются только моно или 2-канальные стерео объекты.

3.5.1.2.8 Тип объекта *TwinVQ*

Объект *TwinVQ* принадлежит схеме кодирования *GA*, реализующей квантование коэффициентов МДКП. Эта схема кодирования основана на векторном квантовании с фиксированной скоростью вместо кода Хаффмана в *AAC*.

Доступны низкие скорости передачи для моно и стерео. Схемы масштабируемых скоростей также доступны в профиле *Scalable Audio*, объединенном с типом объекта *AAC Scalable*.

3.5.1.2.9 Тип объекта *CELP*

Объект *CELP* поддерживается инструментами кодирования речи *CELP*, которые обеспечивают кодирование на частотах дискретизации 8 кГц и 16 кГц при скоростях передачи данных в диапазоне 4–24 кбит/с. Дополнительно доступны масштабируемость скорости передачи и полосы пропускания для обеспечения масштабируемого декодирования потоков *CELP*. Объект *CELP* всегда содержит только один моно сигнал.

3.5.1.2.10 Тип объекта *HVXC*

Объект *HVXC* поддерживается инструментами параметрического кодирования речи (*HVXC*), которые обеспечивают режимы кодирования с фиксированной скоростью передачи данных (2,0 – 4,0 кбит/с) в масштабируемой и немасштабируемой схеме, режим с переменной скоростью передачи данных (<2,0 кбит/с) и функции изменения высоты и скорости воспроизведения. Поддерживаются только звуковые **сигналы** с частотой дискретизации 8 кГц, моно.

3.5.1.2.11 Тип объекта *TTSI*

Объект *TTSI* поддерживается инструментами *TTSI*. Это позволяет осуществить передачу фонемных дескрипторов речи на очень низких скоростях для дальнейшего синтеза. *MPEG*–4 не дает определения самому методу синтеза речи; инструменты *TTSI* скорее определяют интерфейс нестандартных методов синтеза. Этот метод позволяет получить скорость передачи 200 – 1200 бит/с.

3.5.1.2.12 Тип объекта *Main Synthetic*

Объект *Main Synthetic* позволяет использовать все инструменты *MPEG*–4 *Structured Audio*. Это обеспечивает гибкий, высококачественный алгоритмический синтез на основе языка синтеза музыки *SAOL*, эффективный таблично-волновой синтез с форматом банка сэмплов *SASBF* и реализует высококачественное сведение и пост-продакшн с набором инструментов *Systems AudioBIFS*. Описание звука может быть передано на скоростях от 0 кбит/с (не требуется непрерывная передача) до 3–4 кбит/с для чрезвычайно экспрессивных звуков в формате *Structured Audio MPEG*–4.

3.5.1.2.13 Тип объекта *Wavetable Synthesis*

Объект таблично-волнового синтеза поддерживается только форматом *SASBF* и инструментами *MIDI*. Это позволяет использовать простой сэмплерный синтез, когда качество и гибкость полного набора инструментов синтеза не требуются.

3.5.1.2.14 Тип объекта *General MIDI*

Объект General *MIDI* включен только для предоставления функциональной совместимости с существующим контентом. Стандартизованное качество звучания и свойства декодера объектом General *MIDI* не предоставляются.

3.5.1.2.15 Тип объекта *Algorithmic Synthesis* и *Audio FX*

Объект *Algorithmic Synthesis* обеспечивает синтез звука на основе *SAOL* при очень низких скоростях передачи. Он также используется для поддержки узла *AudioBIFS AudioFX* там, где не требуется синтезированный звук.

3.5.1.2.16 Тип объекта *Error Resilient (ER) AAC Low Complexity (LC)*

Тип объекта *Error Resilient (ER) AAC Low Complexity (LC)* является копией объекта *AAC MPEG*–4 *Low Complexity* с дополнительными функциональными возможностями.

3.5.1.2.17 Тип объекта *Error Resilient (ER) AAC Long Term Predictor (LTP)*

Тип объекта *Error Resilient (ER) AAC Long Term Predictor* (*LTP*) является копией объекта *AAC MPEG*–4 *LTP* с дополнительными функциональными возможностями.

3.5.1.2.18 Тип объекта *Error Resilient (ER) AAC scalable*

Тип объекта *Error Resilient (ER) AAC scalable* является копией объекта *AAC MPEG*–4 *scalable* с дополнительными функциональными возможностями.

3.5.1.2.19 Тип объекта *Error Resilient (ER) TwinVQ*

Тип объекта *Error Resilient (ER) AAC TwinVQ* является копией объекта *AAC MPEG*–4 *TwinVQ* с дополнительными функциональными возможностями.

3.5.1.2.20 Тип объекта *Error Resilient (ER) BSAC*

Объект *BSAC ER* поддерживается инструментом точного масштабирования (*BSAC*). Это позволяет обеспечить как устойчивость к ошибкам, так и точную масштабируемость шага квантования в кодере *MPEG*–4 *General Audio (GA).* Данный объект используется в комбинации с инструментами кодирования *AAC* и заменяет кодирование без потерь и форматирование полезного потока бит кодера *AAC MPEG*–4. Доступно большое количество масштабируемых уровней, что обеспечивает уровень расширения на скорости 1 кбит/с/канал, то есть 2 кбит/с шага для сигнала стерео.

3.5.1.2.21 Тип объекта *Error Resilient (ER) AAC LD*

Объект *AAC LD* поддерживается инструментом кодирования *AAC* с низкой задержкой. Он также допускает комбинации с инструментами *PNS* и *LTP*. Объект *AAC LD* расширяет использование универсального низкоскоростного кодирования аудио на приложения, требующие очень низкой задержки кодирования / декодирования (например дуплексная связь в режиме реального времени).

3.5.1.2.22 Тип объекта *Error Resilient (ER) CELP*

Объект *ER CELP* поддерживается инструментами сжатия тишины и *ER*. Это позволяет уменьшить среднюю скорость передачи благодаря более низкому сжатию для участков тишины с дополнительными функциональными возможностями.

3.5.1.2.23 Тип объекта *Error Resilient* (*ER*) *HVXC*

Объект *ER HVXC* поддерживается инструментами параметрического речевого кодирования (*HVXC*), которые обеспечивают режимы с фиксированной скоростью передачи данных (2,0–4,0 кбит/с)

и режимы с переменной скоростью передачи данных (<2,0 кбит/с и <4,0 кбит/с), как с использованием масштабирования, так и без него, с возможностью изменения высоты и скорости воспроизведения. Синтаксис инструмента *EP* и способность устранения ошибок поддерживаются для использования в каналах связи, подверженных ошибкам. Поддерживаются только звуковые сигналы с частотой дискретизации 8 кГц, моно.

3.5.1.2.24 Тип объекта *Error Resilient (ER) HILN*

Объект *ER HILN* поддерживается инструментами параметрического кодирования звука (*HILN*: гармонические и индивидуальные линии вместе с шумом), которые обеспечивают кодирование обычных звуковых сигналов при очень низких скоростях передачи в пределах от менее 4 кбит/с до более 16 кбит/с. Доступны масштабируемость скорости передачи с возможностью изменения высоты и скорости воспроизведения. Объект *ER HILN* поддерживает звуковые объекты моно в широком диапазоне частот дискретизации.

3.5.1.2.25 Тип объекта *Error Resilient (ER) Parametric*

Объект *ER Parametric* поддерживается инструментами параметрического кодирования звука и кодирования речи *HILN* и *HVXC*. Этот встроенный параметрический кодер комбинирует функциональные возможности объектов *ER HILN* и *ER HVXC*. Поддерживаются только звуковые сигналы моно с частотой дискретизации 8 кГц.

3.5.1.2.26 Тип объекта *SSC Audio*

Объект *SSC* (синусоидальное кодирование) комбинирует инструменты параметрического кодирования *SSC*: транзиенты, синусоиды, шум и параметрическое стерео. Поддерживаются сигналы моно, двойное моно и стерео (параметрическое) с частотой дискретизации 44,1 кГц.

3.5.1.2.27 Тип объекта *Layer*-1 *Audio*

Объект *Layer*-1 является копией алгоритма кодирования уровня I.

3.5.1.2.28 Тип объекта *Layer*-2 *Audio*

Объект *Layer*-2 является копией алгоритма кодирования уровня II.

3.5.1.2.29 Тип объекта *Layer*-3 *Audio*

Объект *Layer*-3 схож с алгоритмом кодирования уровня III.

3.5.1.2.30 Тип объекта *ALS Audio*

Тип объекта *ALS* копия алгоритма кодирования без потерь (*ALS*), содержит соответствующие инструменты *ALS*.

3.5.1.2.31 Тип объекта *SLS Audio*

Объект *SLS* поддерживается инструментом масштабируемого кодирования без потерь, который обеспечивает точное масштабирование расширения кодирования без потерь в перцепционных звуковых кодеках *MPEG*, таких как *AAC*, позволяя применять различные шаги по улучшению – от качества звучания основного алгоритма до кодирования без потерь и почти незаметного кодирования. Он также обеспечивает *stand-alone* кодирование звука без потерь, когда основной звуковой кодек отключен.

3.5.1.2.32 SLS Тип объекта *SLS Non-Core Audio*

Объект *SLS non-core* поддерживается инструментом масштабируемого кодирования без потерь. Он подобен типу объекта *SLS*, однако основной звуковой кодек отключен.

3.5.1.2.33 Тип объекта *PS*

Тип объекта *PS* содержит инструмент *PS* и может быть объединен с инструментом *SBR*.

3.5.1.2.34 Тип объекта *MPEG Surround*

Объект *MPEG Surround* содержит служебную информацию *MPEG Surround*.

3.5.1.2.35 Тип объекта *SMR Simple*

Тип объекта *SMR Simple* используется для передачи музыкальных партитур для их аудио и видео обработки. Кодированные данные содержат информацию относительно основной партитуры, частей (то есть партий отдельных инструментов), возможных многоязычных текстов, связанных с частями, правил визуального форматирования, которые будут использоваться при обработке видео, шрифтов для специальных музыкальных символов и информации синхронизации. Шрифты и информация синхронизации представлены как двоичные данные, остальные – как *XML* данные. Тип объекта *SMR Simple* может передать *XML* данные в виде простого *XML* текста или как *gzip XML*.

3.5.1.2.36 Тип объекта *SMR Main*

Тип объекта *SMR Main* может передавать музыкальные партитуры, как тип объекта *SMR Simple*, однако в этом случае *XML* данные могут быть закодированы инструментами *MPEG-B*.

3.5.1.2.37 Тип объекта *Error Resilient (ER) AAC ELD*

Тип объекта с улучшенной низкой задержкой (*ER AAC ELD*) идентичен типу объекта *ER AAC LD* с добавлением банка фильтров с низкой задержкой (*LDFB*) и улучшенного окна. Возможны комбинации с инструментом *PNS*, так же как и с инструментом низкой задержки *SBR*. Тип объекта *ER AAC ELD* расширяет применение универсального низкоскоростного кодирования звука для приложений, требующих очень низкой задержки кодирования/декодирования (например, дуплексная связь в режиме реального времени).

3.5.2 Звуковые профили и уровни

3.5.2.1 Профили

Определены следующие звуковые профили (см. таблицу 3):

1 Речевой профиль обеспечивает параметрический речевой кодер, речевой кодер *CELP* и интерфейс преобразования текста в речь.

2 Профиль синтезированного звука обеспечивает возможность генерировать звук и речь при очень низких скоростях передачи.

3 Масштабируемый профиль – надмножество речевого профиля, является подходящим для масштабируемого кодирования речи и музыки для Интернета и цифрового вещания.

4 Основной профиль – надмножество масштабируемого профиля, речевого профиля и профиля звукового синтеза, содержит инструменты для обычного и синтезированного звука.

5 Профиль высокого качества содержит речевой кодер *CELP* и кодер низкой сложности *AAC*, включающий долгосрочное предсказание. Масштабируемое кодирование может быть выполнено объектом типа *AAC Scalable*. Дополнительно может применяться новый *ER* синтаксис потока битов.

6 Профиль низкой задержки содержит речевые кодеры *HVXC* и *CELP* (дополнительно использующие *ER* синтаксис), кодер *AAC* низкой задержки и интерфейс преобразования текста в речь *TTSI*.

7 Профиль натурального звука содержит все доступные в *MPEG–4* инструменты для кодирования натурального звука.

8 Профиль звука для Интернета содержит типы объектов с низкой задержкой и масштабированием *AAC*, включая *TwinVQ* и *BSAC*. Эта конфигурация предназначена для расширения коммуникационных приложений при помощи не-*MPEG* алгоритмов кодирования речи с высоким качеством.

9 Профиль *AAC* содержит тип звукового объекта 2 (*AAC-LC*).

10 Профиль *AAC* высокой производительности содержит типы звуковых объектов 5 (*SBR*) и 2 (*AAC-LC*). Профиль *AAC* высокой производительности является надмножеством профиля *AAC*.

11 Профиль *AAC* высокой производительности версии 2 содержит типы звуковых объектов 5 (*SBR*), 29 (*PS*) и 2 (*AAC-LC*). Профиль *AAC* высокой производительности версии 2 является надмножеством профиля *AAC* высокой производительности.

12 Профиль *AAC* с низкой задержкой содержит тип звукового объекта 23 (*ER AAC LD*).

Таблица 3 – Определение звуковых профилей

ID типа объекта	Тип объекта *Audio*	*Speech Audio Profile*	*Synthetic Audio Profile*	*Scalable Audio Profile*	*Main Audio Profile*	*High Quality Audio Profile*	*Low Delay Audio Profile*	*Natural Audio Profile*	*Mobile Audio Internetworking Profile*	*AAC Profile*	*High Efficiency AAC Profile*	*High Efficiency AAC v2 Profile*	*Low Delay AAC Profile*
0	*Null*												
1	*AAC main*				X			X					
2	*AAC LC*			X	X	X		X		X	X	X	
3	*AAC SSR*				X			X					
4	*AAC LTP*			X	X	X		X					
5	*SBR*										X	X	
6	*AAC Scalable*			X	X	X		X					
7	*TwinVQ*			X	X			X					
8	*CELP*	X		X	X	X	X	X					
9	*HVXC*	X		X	X		X	X					

Окончание таблицы 3

ID типа объекта	Тип объекта *Audio*	*Speech Audio Profile*	*Synthetic Audio Profile*	*Scalable Audio Profile*	*Main Audio Profile*	*High Quality Audio Profile*	*Low Delay Audio Profile*	*Natural Audio Profile*	*Mobile Audio Internetworking Profile*	*AAC Profile*	*High Efficiency AAC Profile*	*High Efficiency AAC v2 Profile*	*Low Delay AAC Profile*
10	(зарезервировано)												
11	(зарезервировано)												
12	*TTSI*	X	X	X	X		X	X					
13	*Main synthetic*		X		X								
14	*Wavetable synthesis*		X*		X*								
15	*General MIDI*		X*		X*								
16	*Algorithmic Synthesis and Audio FX*		X*		X*								
17	*ER AAC LC*					X		X	X				
18	(зарезервировано)												
19	*ER AAC LTP*					X		X					
20	*ER AAC Scalable*					X		X	X				
21	*ER TwinVQ*							X	X				
22	*ER BSAC*							X	X				
23	*ER AAC LD*						X	X	X				X
24	*ER CELP*					X	X	X					
25	*ER HVXC*						X	X					
26	*ER HILN*							X					
27	*ER Parametric*							X					
28	*SSC*												
29	*PS*											X	
30	*MPEG Surround*												
31	*(escape)*												
32	*Layer-1*												
33	*Layer-2*												
34	*Layer-3*												
35	*DST*												
36	*ALS*												
37	*SLS*												
38	*SLS non-core*												
39	*ER AAC ELD*												
40	*SMR Simple*												
41	*SMR Main*												

В дополнение к описаниям профилей, данным выше, необходимо отметить, что объекты *AAC Scalable*, использующие широкополосный уровень ядра *CELP* (с или без синтаксиса полезного битового потока *ER*) не являются частью какого бы то ни было звукового профиля.

3.5.2.2 Единицы сложности

Единицы сложности введены для того, чтобы дать приблизительную оценку сложности декодера в терминах вычислительной мощности и использования *RAM*, необходимых для обработки полезной части потока бит *MPEG–4* Аудио в зависимости от определенных параметров.

Приблизительная вычислительная мощность дана в единицах вычислительной сложности (*PCU*), выраженных в *MOPS*. Приблизительное использование оперативной памяти дано в единицах использования памяти (*RCU*), выраженных в килословах (1000 слов). Количество *RCU* не включает в себя объем буфера, который может быть разделен между различными объектами и/или каналами.

Если уровень профиля определен максимальным числом единиц сложности, то гибкая конфигурация декодера, обрабатывающего различные типы объектов, допустима при условии, что оба значения сложности (*PCU* и *RCU*) для декодирования и преобразования частоты дискретизации (если требуется) не превышают этот предел.

Таблица 4 дает оценку сложности для различных типов объектов. Значения *PCU* даны в *MOPS* на канал, значения *RCU* – в килословах на канал (в *AAC* термин «канал» соответствует основному каналу, например, канал *SCE*, один канал *CPE* или канал независимо переключаемого *CCE*).

Таблица 4 – Сложность типов звуковых объектов и преобразования частоты дискретизации

Тип объекта	Параметры	*PCU (MOPS)*	*RCU*	Примечания
AAC Main	*fs* = 48 кГц	5	5	1
AAC LC	*fs* = 48 кГц	3	3	1
AAC SSR	*fs* = 48 кГц	4	3	1
AAC LTP	*fs* = 48 кГц	4	4	1
SBR	*fs* = 24/48 кГц (*in/out*) (*SBR tool*)	3	2.5	1
	fs = 24/48 кГц (*in/out*) (*Low Power SBR tool*)	2	1.5	1
	fs = 48/48 кГц (*in/out*) (*Down Sampled SBR tool*)	4.5	2.5	1
	fs = 48/48 кГц (*in/out*)			
(*Low Power Down Sampled)*				
SBR tool)	3	1.5	1	
AAC Scalable	*fs* = 48 кГц	5	4	1, 2
TwinVQ	*fs* = 24 кГц	2	3	1
CELP	*fs* = 8 кГц	1	1	
CELP	*fs* = 16 кГц	2	1	
CELP	*fs* = 8/16 кГц (*bandwidth scalable*)	3	1	
HVXC	*fs* = 8 кГц	2	1	
TTSI		-	-	4
General MIDI		4	1	
Wavetable				
Synthesis	*fs* = 22,05 кГц	Зависит от потока битов	Зависит от потока битов	
Main Synthetic		Зависит от потока битов	Зависит от потока битов	
Algorithmic Synthesis and AudioFX		Зависит от потока битов	Зависит от потока битов	
Sampling Rate Conversion	*rf* = 2, 3, 4, 6, 8, 12	2	0.5	3
ER AAC LC	*fs* = 48 кГц	3	3	1
ER AAC LTP	*fs* = 48 кГц	4	4	1
ER AAC Scalable	*fs* = 48 кГц	5	4	1, 2
ER TwinVQ	*fs* = 24 кГц	2	3	1
ER BSAC	*fs* = 48 кГц (*Размер входного буфера*=26000 битов)	4	4	1
	fs = 48 кГц (*Размер входного буфера* =106000 битов)	4	8	
ER AAC LD	*fs* = 48 кГц	3	2	1
ER CELP	*fs* = 8 кГц	2	1	
	fs = 16 кГц	3	1	
ER HVXC	*fs* = 8 кГц	2	1	
ER HILN	*fs* = 16 кГц, *ns*=93	15	2	6
	fs = 16 кГц, *ns*=47	8	2	

Окончание таблицы 4

Тип объекта	Параметры	*PCU (MOPS)*	*RCU*	Примечания
ER Parametric	*fs* = 8 кГц, *ns*=47	4	2	5,6
ER AAC ELD	*fs* = 48 кГц	3	2	1
ER AAC ELD, Low Delay SBR tool only	*fs* = 24/48 кГц (*in/out*) (*SBR tool*)	3	2.5	1
	fs = 24/48 *кГц* (*in/out*) (*Low Power* SBR *tool*)	2	1.5	1
	fs = 48/48 кГц (*in/out*) (*Down Sampled SBR tool*)	4.5	2.5	1
	fs = 48/48 кГц (*in/out*)			
(*Low Power Down Sampled*				
SBR tool)	3	1.5	1	

Определения:

fs = частота дискретизации

rf = отношение частот дискретизации

Примечания

1 *PCU* пропорционально частоте дискретизации.

2 Включает основной декодер.

3 Сложность для синтеза речи не учитывается.

4 Параметрический кодер в режиме *HILN*, для режима *HVXC* см. *ER HVXC*.

5 *PCU* зависит от *fs* и *ns*, см. ниже.

6 Преобразование частоты дискретизации необходимо, если объекты с различными частотами дискретизации объединены в сцене. Указанные значения должны быть добавлены для каждого необходимого преобразования.

PCU для *HILN*:

Вычислительная сложность *HILN* зависит от частоты дискретизации *fs* и максимального числа синусоид *ns*, которое должно быть синтезировано одновременно. Значение *ns* для фрейма является общим количеством гармонических и индивидуальных линий, синтезируемых в этом фрейме, то есть суммой начальных, промежуточных и конечных линий. Для *fs* в кГц *PCU* в *MOPS* вычисляется следующим образом:

PCU= (1 + 0,15**ns*) * *fs*/16

Типовые максимальные значения *ns* составляют 47 для *HILN* 6 кбит/с и 93 для потоков *HILN* 16 кбит/с.

PCU и *RCU* для *AAC*:

Для типов объектов *AAC PCU* и *RCU* зависят от частоты дисретизации и конфигурации каналов следующим образом:

PCU

PCU = (*fs* / *fs_ref*) * *PCU_ref* * (2 * *#CPE* + *#SCE* + #LFE + *#IndepCouplingCh* + 0.3 * *#DepCouplingCh*)

fs: фактическая частота дискретизации

fs_ref: эталонная частота дискретизации (частота дискретизации для данного *PCU_ref*)

PCU_ref: эталонное *PCU*, данное в таблице 4

#SCE: количество *SCE*

#CPE: ...

RCU

#CPE <2:

RCU = *RCU_ref* * [*#SCE* + 0,5 * *#LFE* + 0,5 * *#IndepCouplingCh* + 0,4 * *#DepCouplingCh*] + [*RCU_ref* + (*RCU_ref* - 1)] * *#CPE*

#CPE> = 2:

RCU = *RCU_ref* * [*#SCE* + 0,5 * *#LFE* + 0,5 * *#IndepCouplingCh* + 0,4 * *#DepCouplingCh*] + [*RCU_ref* + (*RCU_ref* - 1) * (2 * *#CPE* - 1)]

RCU_ref: эталонное *RCU*, данное в таблице 4

#SCE: количество *SCE*

#CPE: количество *CPE*

3.5.2.3 Уровни профилей

Под числом звуковых каналов понимается число основных звуковых каналов. На основании количества основных звуковых каналов (A) в таблице 5 указано число каналов *LFE* (*L*), число независимо переключаемых спаренных каналов (*I*) и число зависимо переключаемых спаренных каналов (*D*) для типов объектов, полученных из многоканального *AAC* в форме *A.L.I.D.*

Таблица 5 – Максимальное число индивидуальных типов каналов *AAC* в зависимости от указанного количества основных звуковых каналов

	Количество основных звуковых каналов		
AOT	1	2	3
1 (*AAC main*)	1.0.0.0	2.0.0.0	5.1.1.1
2 (*AAC LC*)	1.0.0.0	2.0.0.0	5.1.0.1
3 (*AAC SSR*)	1.0.0.0	2.0.0.0	5.1.0.0
4 (*AAC LTP*)	1.0.0.0	2,0.0.0	5.1.0.1
17 (*ER AAC LC*)	1.0.0.0	2.0.0.0	5.1.0.0
19 (*ER AAC LTP*)	1.0.0.0	2.0.0.0	5.1.0.0
23 (*ER AAC LD*)	1.0.0.0	2.0.0.0	5.1.0.0

Примечание – В случае масштабируемых схем кодирования для определения количества объектов, допустимых по сложности, учитывается только первая реализация каждого типа объекта. Например, в масштабируемом кодере, состоящем из основного кодера *CELP* и двух уровней расширения, реализованных средствами масштабируемых объектов *AAC*, считается один объект *CELP* и один масштабируемый объект *AAC*; учитываются их соответствующие показатели сложности, так как практически отсутствуют затраты, связанные со вторым (и выше) уровенем расширения *GA*.

Уровни профиля речи

Определены два уровня по числу объектов:

1. Один речевой объект.

2. До 20 речевых объектов.

Уровни профиля синтезированного звука

Определены три уровня:

1. Синтезированный звук 1: все элементы полезной части потока бит могут использоваться с:

- режимом пониженной производительности
- только основными частотами дискретизации
- только одним объектом *TTSI*

2. Синтезированный звук 2: все элементы полезной части потока бит могут использоваться с:

- режимом средней производительности.
- только основными частотами дискретизации
- максимум четырьмя объектами *TTSI*

3. Синтезированный звук 3: все элементы полезной части потока бит могут использоваться с:

- режимом высокой производительности
- максимум двенадцатью объектами *TTSI*
- уровнями масштабируемого профиля

Профилем определены четыре уровня; четвертый уровень определяется единицами сложности:

1. Максимальное значение частоты дискретизации – 24 кГц, один моно объект (все типы объектов).

2. Максимальное значение частоты дискретизации – 24 кГц, один стерео объект или два моно (все типы объектов).

3. Максимальное значение частоты дискретизации – 48 кГц, один стерео объект или два моно (все типы объектов).

4. Максимальное значение частоты дискретизации – 48 кГц, один 5-канальный объект или несколько объектов с одним целочисленным множителем частоты дискретизации максимум для двух каналов.

Разрешена гибкая конфигурация при *PCU*<30 и *RCU* <19.

Для типов звуковых объектов 2 (*AAC LC*) и 4 (*AAC LTP*) допускается длина фрейма только 1024 отсчета для уровней 1, 2, 3 и 4. Для типов звуковых объектов 2 (*AAC LC*) и 4 (*AAC LTP*) не допускаются моно или стерео элементы сведения для уровней 1, 2, 3 и 4. Для типа звуковых объектов

6 (*AAC Scalable*) применяются следующие ограничения. Число уровней *AAC* не должно превышать 8 для всех масштабируемых конфигураций. Если тип звукового объекта 8 (*CELP*) будет использоваться как основной кодер уровня, то число уровней *CELP* не должно превышать 2. Если тип звукового объекта 7 (*TwinVQ*) используется как основной кодер уровня, разрешен только один моно уровень *TwinVQ*.

Уровни основного профиля

Основной профиль содержит все натуральные и синтезированные типы объектов. Уровни определяются как комбинация двух различных типов уровней двух различных показателей, определенных для натуральных инструментов (показатели на основе вычислительной мощности) и инструментов синтеза (макропоказатели).

Для типов объектов, не принадлежащих профилю синтезированного звука определены четыре уровня:

- Натуральный звук 1: *PCU*<40, *RCU* <202.
- Натуральный звук 2: *PCU*<80, *RCU* <643.
- Натуральный звук 3: *PCU*<160, *RCU* <1284.
- Натуральный звук 4: *PCU*<320, *RCU* <256.

Для типов объектов, принадлежащих к профилю синтезированного звука, определены те же самые три уровня, то есть синтезированный звук 1, синтезированный звук 2 и синтезированный звук 3.

Четыре уровня определены для основного профиля:

- Натуральный звук 1 + синтезированный звук 1.
- Натуральный звук 2 + синтезированный звук 1.
- Натуральный звук 3 + синтезированный звук 2.
- Натуральный звук 4 + синтезированный звук 3.

Для типов звуковых объектов 1 (*AAC main*), 2 (*AAC LC*), 3 (*AAC SSR*) и 4 (*AAC LTP*) допустима длина фрейма только 1024 отсчета для уровней 1, 2, 3 и 4. Для типов звуковых объектов 1 (*AAC main*), 2 (*AAC LR*), 3 (*AAC SSR*) и 4 (*AAC LTP*) не разрешены моно или стерео элементы сведения для уровней 1, 2, 3 и 4. Для типа звуковых объектов 6 (*AAC Scalable*) применяются следующие ограничения. Число уровней *AAC* не должно превышать 8 в любой из масштабируемых конфигураций. Если тип звуковых объектов 8 (*CELP*) используется как основной кодер уровня, то число уровней *CELP* не должно превышать 2. Если тип звукового объекта 7 (*TwinVQ*) используется как основной кодер слоя, то разрешен только один моно уровень *TwinVQ*.

Уровни профиля высокого качества приведены в таблице 6.

Таблица 6 – Уровни профиля высокого качества

Уровень	Максимум каналов/ объект	Максимальная частота дискретизации, кГц	*Max PCU* **	*Max RCU* **	*EP-Tool*: Максимальная избыточность класса *FEC* *, %	*EP-Tool*: Максимальное число этапов чередования на объект
1	2	22,05	5	8	0	0
2	2	48	10	8	0	0
3	5	48	25	12 ***	0	0
4	5	48	100	42 ***	0	0
5	2	22,05	5	8	20	9
6	2	48	10	8	20	9
7	5	48	25	12 ***	20	22
8	5	48	100	42 ***	20	22

* Значение определяет максимальную избыточность на основе доступного звукового объекта с самой большой длиной фрейма. Избыточность может принимать большие значения в случае меньших длин фрейма. Однако использование любого *FEC* класса не разрешено при 0 %. Предел действителен для каждого звукового объекта. Так как это значение не учитывает ни заголовок *EP* и его биты защиты, ни любой *CRC*, 5 % всегда должны быть добавлены к этому значению для получения необходимого увеличения минимального входного буфера декодера. Это подразумевает, что не более, чем 5 % может быть потрачено для заголовка *EP* и его бит защиты или любого *CRC*.

** Уровни 5 – 8 не включают оперативную память и вычислительную сложность для инструмента *EP*.

*** Подразумевается совместное использование рабочих буферов для множественных объектов и элементов канальных пар.

Для типов звуковых объектов 2 (*AAC LC*), 4 (*AAC LTP*), 17 (*ER AAC LC*) и 19 (*ER AAC LTP*) допустима длина фрейма только 1024 отсчета для уровней 1, 2, 3, 4, 5, 6, 7 и 8. Для типов звуковых объектов 2 (*AAC LC*) и 4 (*AAC LTP*) не разрешены моно или стерео элементы сведения для уровней 1, 2, 3, 4, 5, 6, 7 и 8. Для типа звуковых объектов 6 и 20 (*ER*) *AAC Scalable*) применяются следующие ограничения. Число уровней *AAC* не должно превышать 8 для любой масштабируемой конфигурации. Если тип звукового объекта 8 или 24 (*ER*) *CELP*) будет использоваться как основной кодер уровня, то число уровней *CELP* не должно превышать 2.

Уровни профиля низкой задержки приведены в таблице 7.

Таблица 7 – Уровни профиля низкой задержки

Уровень	Максимум каналов/ объект	Максимальная частота дискретизации, кГц	*Max PCU* **	*Max RCU* **	*EP-Tool*: Максимальная избыточность класса *FEC* *, %	*EP-Tool*: Максимальное число этапов чередования на объект
1	1	8	2	1	0	0
2	1	16	3	1	0	0
3	1	48	3	2	0	0
4	2	48	24	12 ***	0	0
5	1	8	2	1	100	5
6	1	16	3	1	100	5
7	1	48	3	2	20	5
8	2	48	24	12 ***	20	9

Уровни профиля натурального звука приведены в таблице 8.

Таблица 8 – Уровни профиля натурального звука

Уровень	Максимальная частота дискретизации, кГц	*Max PCU* ****	*EP-Tool*: Максимальная избыточность класса *FEC* *, %	*EP-Tool*: Максимальное число этапов чередования на объект
1	48	20	0	0
2	96	100	0	0
3	48	20	20	9
4	96	100	20	22

Для данного профиля нет ограничений по *RCU*.

Для типов звуковых объектов 1 (*AAC main*), 2 (*AAC LC*), 3 (*AAC SSR*), 4 (*AAC LTP*), 17 (*ER AAC LC*) и 19 (*ER AAC LTP*) допустима длина фрейма только 1024 отсчета для уровней 1, 2, 3 и 4. Для типов звуковых объектов 1 (*AAC main*), 2 (*AAC LC*), 3 (*AAC SSR*) и 4 (*AAC LTP*) не разрешены моно или стерео элементы сведения для уровней 1, 2, 3 и 4. Для типа звуковых объектов 6 и 20 (*ER*) *AAC Scalable*) применяются следующие ограничения. Число уровней *AAC* не должно превышать 8 для любой масштабируемой конфигурации. Если тип звукового объекта 8 или 24 (*ER*) *CELP*) будет использоваться как основной кодер уровня, то число уровней *CELP* не должно превышать 2. Если тип звукового объекта 7 или 21 (*ER*) *TwinVQ*) будет использоваться как основной кодер уровня, то только один моно уровень *TwinVQ* допустим.

---

* Значение определяет максимальную избыточность на основе доступного звукового объекта с самой большой длиной фрейма. Избыточность может принимать большие значения в случае меньших длин фрейма. Однако использование любого *FEC* класса не разрешено при 0 %. Предел действителен для каждого звукового объекта. Так как это значение не учитывает ни заголовок *EP* и его биты защиты, ни любой *CRC*, 5 % всегда должны быть добавлены к этому значению для получения необходимого увеличения минимального входного буфера декодера. Это подразумевает, что не более, чем 5 % может быть потрачено для заголовка *EP* и его бит защиты или любого *CRC*.

** Уровни 5 – 8 не включают оперативную память и вычислительную сложность для инструмента *EP*.

*** Подразумевается совместное использование рабочих буферов для множественных объектов и элементов канальных пар.

**** Уровни 3 и 4 не включают оперативную память и вычислительную сложность для инструмента *EP*.

Уровни профиля звука для Интернета приведены в таблице 9.

Таблица 9 – Уровни профиля звука для Интернета

Уровень	Максимум каналов/ объект	Максимальная частота дискретизации, кГц	*Max PCU* **	*Max RCU* **, ***	Максимальное число объеков аудио	*EP-Tool*: Максимальная избыточность класса *FEC* *, %	*EP-Tool*: Максимальное число этапов чередования на объект
1	1	24	2.5	4	1	0	0
2	2	48	10	8	2	0	0
3	5	48	25	12 ****	-	0	0
4	1	24	2.5	4	1	20	5
5	2	48	10	8	2	20	9
6	5	48	25	12 ****	-	20	22

Для типа звуковых объектов 17 (*ER AAC LC*) допустима длина фрейма только 1024 отсчета для уровней 1, 2, 3, 4, 5 и 6. Для типа звуковых объектов 20 (*ER AAC Scalable*) применяются следующие ограничения. Число уровней *AAC* не должно превышать 8 для любой масштабируемой конфигурации. Если тип звукового объекта 21 (*ER TwinVQ*) будет использоваться как основной кодер уровня, то только один моно уровень *TwinVQ* допустим.

Уровни профиля *AAC* приведены в таблице 10.

Таблица 10 – Уровни профиля *AAC*

Уровень	Максимальное число каналов/ объект	Максимальная частота дискретизации, кГц	Max *PCU*	Max *RCU*
1	2	24	3	5
2	2	48	6	5
3	не применяется	не применяется	не применяется	не применяется
4	5	48	19	15
5	5	96	38	15

Для типа звукового объекта 2 (*AAC LC*) не разрешены моно или стерео элементы сведения.

Уровни с пометкой «не применяется» введены для сохранения иерархической структуры профиля *AAC* и профиля *AAC* высокой производительности. Следовательно, декодер, поддерживающий профиль *AAC* высокой производительности на данном уровне, может декодировать поток профиля *AAC* того же самого или более низкого уровня. Уровни с пометкой «не применяется» не обозначены в таблице *audioProfileLevelIndication* (таблица 14).

Уровни профиля *AAC* высокой производительности приведены в таблице 11.

Таблица 11 – Уровни профиля AAC высокой производительности

Уровень	Максимальное число каналов/ объект	Максимальная частота дискретизации *AAC*, *SBR* отсутствует, кГц	Максимальная частота дискретизации *AAC*, *SBR* присутствует, кГц	Максимальная частота дискретизации *SBR*, кГц (ввод/ вывод)	*Max PCU*	*Max RCU*	*Max PCU*, маломощный *SBR*	*Max RCU*, маломощный *SBR*
1	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
2	2	48	24	24/48	9	10	7	8
3	2	48	48	48/48 (примечание 1)	15	10	12	8

* Значение определяет максимальную избыточность на основе доступного звукового объекта с самой большой длиной фрейма. Избыточность может принимать большие значения в случае меньших длин фрейма. Однако использование любого *FEC* класса не разрешено при 0 %. Предел действителен для каждого звукового объекта. Так как это значение не учитывает ни заголовок *EP* и его биты защиты, ни любой *CRC*, 5 % всегда должны быть добавлены к этому значению для получения необходимого увеличения минимального входного буфера декодера. Это подразумевает, что не более, чем 5 % может быть потрачено для заголовка *EP* и его бит защиты или любого *CRC*.

** Максимальное значение *RCU* для одного канала в любом объекте этой конфигурации – 4. Для *ER BSAC* это является ограничением входного размера буфера. Максимальный возможный размер входного буфера в битах для этого случая дается *PCU*/*RCU* (таблица 4).

*** Уровни 4–6 не включают оперативную память и вычислительную сложность для инструмента *EP*.

**** Подразумевается совместное использование рабочих буферов для множественных объектов и элементов канальных пар.

Окончание таблицы 11

Уровень	Максимальное число каналов/ объект	Максимальная частота дискретизации *AAC*, *SBR* отсутствует, кГц	Максимальная частота дискретизации *AAC*, *SBR* присутствует, кГц	Максимальная частота дискретизации *SBR*, кГц (ввод/ вывод)	*Max PCU*	*Max RCU*	*Max PCU*, маломощный *SBR*	*Max RCU*, маломощный *SBR*
4	5	48	24/48 (примечание 2)	48/48 (примечание 1)	25	28	20	23
5	5	96	48	48/96	49	28	39	23

Примечание 1 – Для уровней 3 и 4 декодера является обязательным управление инструментом *SBR* в режиме децимации, если частота дискретизации ядра *AAC* превышает 24 кГц. Следовательно, если инструмент *SBR* обрабатывает сигнал *AAC* на частоте 48 кГц, то внутренняя частота дискретизации инструмента *SBR* составит 96 кГц, однако выходной сигнал будет подвергнут децимации инструментом *SBR* до 48 кГц.

Примечание 2 – Для одного или двух каналов максимальная частота дискретизации *AAC* с включенным *SBR* составляет 48 кГц. Для более чем двух каналов максимальная частота дискретизации *AAC* с включенным *SBR* составляет 24 кГц.

Для типа звукового объекта 2 (*AAC LC*) не разрешены моно или стерео элементы сведения. Уровни профиля *AAC* высокой производительности версии 2 приведены в таблице 12.

Таблица 12 – Уровни профиля *AAC* высокой производительности версии 2

Уровень (примечание 1)	Максимальное число каналов/ объект	Максимальная частота дискретизации *AAC*, *SBR* отсутствует, кГц	Максимальная частота дискретизации *AAC*, *SBR* присутствует, кГц	Максимальная частота дискретизации *SBR*, кГц (ввод/ вывод)	*Max PCU*	*Max RCU*	*Max PCU HQ/LP SBR* (примечание 5)	*Max RCU* HQ/LP SBR (примечание 5)
1	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*	*NA*
2	2	48	24	24/48	9	10	9	10
3	2	48	24/48 (примечание 3)	48/48 (примечание 2)	15	10	15	10
4	5	48	24/48 (примечание 4)	48/48 (примечание 2)	25	28	20	23
5	5	96	48	48/96	49	28	39	23

Примечание 1 – Декодеры уровней 2, 3 и 4 *HE AAC* профиля версии 2 реализуют базовую версию параметрического инструмента стерео. Декодер уровня 5 не должен быть ограничен базовой версией параметрического инструмента стерео.

Примечание 2 – Для уровней декодера 3 и 4 инструмент *SBR* должен использоваться в режиме децимации, если частота дискретизации ядра *AAC* выше, чем 24 кГц. Следовательно, если инструмент *SBR* будет обрабатывать сигнал *AAC* с частотой дискретизации 48 кГц, то внутренняя частота дискретизации инструмента *SBR* составит 96 кГц, однако при этом входной сигнал будет децимирован инструментом *SBR* до 48 кГц.

Примечание 3 – Если присутствуют параметрические стерео данные, то максимальная частота дискретизации *AAC* составляет 24 кГц, в противном случае максимальная частота дискретизации *AAC* составляет 48 кГц.

Примечание 4 – Для одного или двух каналов максимальная частота дискретизации *AAC* с включенным *SBR* составляет 48 кГц. Для большего числа каналов максимальная частота дискретизации *AAC* с включенным *SBR* составляет 24 кГц.

Примечание 5 – Количество *PCU*/*RCU* дается для декодера, использующего инструмент *SBR*, если необходимо.

Для типа звукового объекта 2 (*AAC LC*) не разрешены моно или стерео элементы сведения.

Декодер профиля *HE AAC v2* обрабатывает с помощью инструмента *HQ SBR* потоки бит, содержащие параметрические стерео данные. Для потоков бит, не содержащих параметрические стерео данные, декодер профиля *HE AAC v2* может использовать инструмент *HQ SBR* или инструмент *LP SBR*.

Только потоки бит, содержащие строго один отдельный канальный элемент *AAC*, могут содержать параметрические стерео данные. Потоки бит, содержащие больше одного канала в части *AAC*, не должны содержать параметрические стерео данные.

Уровни профиля *AAC* с низкой задержкой приведены в таблице 13.

Таблица 13 – Уровни профиля *AAC* с низкой задержкой

Уровень	Максимальное число каналов/объект	Максимальная частота дискретизации, кГц	*epConfig*
1	2	48	0

LTP не разрешен. Импульсные данные не разрешены.

3.5.2.4 *audioProfileLevelIndication*

audioProfileLevelIndication – элемент данных *InitialObjectDescriptor*. Он показывает профиль и уровень, требуемые для обработки контента, ассоциируемого с *InitialObjectDescriptor*, согласно таблице 14.

Таблица 14 – Значения *audioProfileLevelIndication*

Значение	Профиль	Уровень
0x00	Зарезервировано для ISO	-
0x01	*Main Audio Profile*	*L1*
0x02	*Main Audio Profile*	*L2*
0x03	*Main Audio Profile*	*L3*
0x04	*Main Audio Profile*	*L4*
0x05	*Scalable Audio Profile*	*L1*
0x06	*Scalable Audio Profile*	*L2*
0x07	*Scalable Audio Profile*	*L3*
0x08	*Scalable Audio Profile*	*L4*
0x09	*Speech Audio Profile*	*L1*
0x0*A*	*Speech Audio Profile*	*L2*
0x0*B*	*Synthetic Audio Profile*	*L1*
0x0*C*	*Synthetic Audio Profile*	*L2*
0x0*D*	*Synthetic Audio Profile*	*L3*
0x0*E*	*High Quality Audio Profile*	*L1*
0x0*F*	*High Quality Audio Profile*	*L2*
0x10	*High Quality Audio Profile*	*L3*
0x11	*High Quality Audio Profile*	*L4*
0x12	*High Quality Audio Profile*	*L5*
0x13	*High Quality Audio Profile*	*L6*
0x14	*High Quality Audio Profile*	*L7*
0x15	*High Quality Audio Profile*	*L8*
0x16	*Low Delay Audio Profile*	*L1*
0x17	*Low Delay Audio Profile*	*L2*
0x18	*Low Delay Audio Profile*	*L3*
0x19	*Low Delay Audio Profile*	*L4*
0x1*A*	*Low Delay Audio Profile*	*L5*
0x1*B*	*Low Delay Audio Profile*	*L6*
0x1*C*	*Low Delay Audio Profile*	*L7*
0x1*D*	*Low Delay Audio Profile*	*L8*
0x1*E*	*Natural Audio Profile*	*L1*
0x1*F*	*Natural Audio Profile*	*L2*
0x20	*Natural Audio Profile*	*L3*
0x21	*Natural Audio Profile*	*L4*
0x22	*Mobile Audio Internetworking Profile*	*L1*
0x23	*Mobile Audio Internetworking Profile*	*L2*
0x24	*Mobile Audio Internetworking Profile*	*L3*
0x25	*Mobile Audio Internetworking Profile*	*L4*
0x26	*Mobile Audio Internetworking Profile*	*L5*
0x27	*Mobile Audio Internetworking Profile*	*L6*
0x28	*AAC Profile*	*L1*
0x29	*AAC Profile*	*L2*
0x2*A*	*AAC Profile*	*L4*

Окончание таблицы 14

Значение	Профиль	Уровень
0x2*B*	*AAC Profile*	*L*5
0x2*C*	*High Efficiency AAC Profile*	*L*2
0x2*D*	*High Efficiency AAC Profile*	*L*3
0x2*E*	*High Efficiency AAC Profile*	*L*4
0x2*F*	*High Efficiency AAC Profile*	*L*5
0x30	*High Efficiency AAC v2 Profile*	*L*2
0x31	*High Efficiency AAC v2 Profile*	*L*3
0x32	*High Efficiency AAC v2 Profile*	*L*4
0x33	*High Efficiency AAC v2 Profile*	*L*5
0x34	*Low Delay AAC Profile*	*L*1
0x35	*Baseline MPEG Surround Profile*	*L*1
0x36	*Baseline MPEG Surround Profile*	*L*2
0x37	*Baseline MPEG Surround Profile*	*L*3
0x38	*Baseline MPEG Surround Profile (see)*	*L*4
0c39	*Baseline MPEG Surround Profile (see)*	*L*5
0x3*A*	*Baseline MPEG Surround Profile (see)*	*L*6
0x3*B* - 0x7*F*	*reserved for ISO use*	-
0x80 - 0x*FD*	*user private*	-
0x*FE*	*no audio profile specified*	-
0x*FF*	*no audio capability required*	-

Примечание — Использование значения 0x*FE* указывает, что описанный этим дескриптором *InitialObjectDescriptor* контент не соответствует никакому аудиопрофилю. Использование значения равного 0x*FF* указывает, что никакие возможности аудио профиля не требуются для этого контента.

3.6 Интерфейс *MPEG*–4 Системы

3.6.1 Введение

Потоки заголовка транспортируются через *MPEG*–4 Системы. Эти потоки содержат информацию о конфигурации, которая необходима для декодирования и анализа необработанных потоков данных. Однако обновление необходимо только при изменениях в конфигурации.

3.6.2 Синтаксис

3.6.2.1 *AudioSpecificConfig*

AudioSpecificConfig () расширяет абстрактный класс. В этом случае наличие *AudioSpecificConfig* () обязательно (см. таблицы 15, 16).

Таблица 15 – Синтаксис *AudioSpecificConfig* ()

Синтаксис	Количество битов	Мнемоника
AudioSpecificConfig () *{* *audioObjectType = GetAudioObjectType();* *samplingFrequencyIndex;* *if (samplingFrequencyIndex == 0xf) {* *samplingFrequency;* *}* *channelConfiguration;* *sbrPresentFlag = -1; psPresentFlag = -1;*	 4 24 4 	 *bslbf* *uimsbf* *bslbf*
if (audioObjectType == 5 || *audioObjectType == 29) {* *extensionAudioObjectType = 5;* *sbrPresentFlag = 1;* *if (audioObjectType == 29) {* *psPresentFlag = 1;* *}*		

Продолжение таблицы 15

Синтаксис	Количество битов	Мнемоника
extensionSamplingFrequencyIndex;	4	*uimsbf*
if (extensionSamplingFrequencyIndex == 0xf)		
extensionSamplingFrequency;	24	*uimsbf*
audioObjectType = GetAudioObjectType();		
if (audioObjectType == 22)		
extensionChannelConfiguration;	4	*uimsbf*
}		
else {		
extensionAudioObjectType = 0;		
}		
switch (audioObjectType) {		
case 1:		
case 2:		
case 3:		
case 4:		
case 6:		
case 7:		
case 17:		
case 19:		
case 20:		
case 21:		
case 22:		
case 23:		
GASpecificConfig();		
break;		
case 8:		
CelpSpecificConfig();		
break;		
case 9:		
HvxcSpecificConfig();		
break;		
case 12:		
TTSSpecificConfig();		
break;		
case 13:		
case 14:		
case 15:		
case 16:		
StructuredAudioSpecificConfig();		
break;		
case 24:		
ErrorResilientCelpSpecificConfig();		
break;		
case 25:		
ErrorResilientHvxcSpecificConfig();		
break;		
case 26:		
case 27:		
ParametricSpecificConfig();		
break;		
case 28:		
SSCSpecificConfig();		
break;		
case 30:		
sacPayloadEmbedding;	1	*uimsbf*
SpatialSpecificConfig();		
break;		
case 32:		
case 33:		

Продолжение таблицы 15

Синтаксис	Количество битов	Мнемоника
case 34*:* *MPEG_1_2_SpecificConfig();* *break;* *case* 35*:* *DSTSpecificConfig();* *break;* *case* 36*:* *fillBits;* *ALSSpecificConfig();* *break;* *case* 37*:* *case* 38*:* *SLSSpecificConfig();* *break;* *case* 39*:* *ELDSpecificConfig(channelConfiguration);* *break:* *case* 40*:* *case* 41*:* *SymbolicMusicSpecificConfig();* *break;* *default:* */* reserved */* *}*		
switch (audioObjectType) { *case* 17*:* *case* 19*:* *case* 20*:* *case* 21*:* *case* 22*:* *case* 23*:* *case* 24*:* *case* 25*:* *case* 26*:* *case* 27*:* *case* 39*:*		
epConfig; *if (epConfig == 2 || epConfig == 3) {* *ErrorProtectionSpecificConfig();* *}*	2	*bslbf*
if (epConfig == 3) { *directMapping;* *if (! directMapping) {* */* tbd */* *}* *}* *}*	1	*bslbf*
if (extensionAudioObjectType!=5 && bits_to_decode()>=16) { *syncExtensionType;* *if (syncExtensionType == 0x2b7) {* *extensionAudioObjectType = GetAudioObjectType();* *if (extensionAudioObjectType == 5) {* *sbrPresentFlag;* *if (sbrPresentFlag == 1) {* *extensionSamplingFrequencyIndex;* *if (extensionSamplingFrequencyIndex == 0xf) {* *extensionSamplingFrequency;* *}* *if (bits_to_decode() >= 12) {*	11 1 4 24	*bslbf* *uimsbf* *uimsbf* *uimsbf*

Окончание таблицы 15

Синтаксис	Количество битов	Мнемоника
syncExtensionType; *if (syncExtesionType == 0x548) {* *psPresentFlag;* *}* *}* *}* *}* *if (extensionAudioObjectType == 22) {* *sbrPresentFlag;* *if (sbrPresentFlag == 1) {* *extensionSamplingFrequencyIndex;* *if (extensionSamplingFrequencyIndex == 0xf) {* *extensionSamplingFrequency;* *}* *}* *extensionChannelConfiguration;* *}* *}* *}* *}*	 11 1 1 4 24 4	*bslbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf*

Т а б л и ц а 16 – Синтаксис *GetAudioObjectType* ()

Синтаксис	Количество битов	Мнемоника
GetAudioObjectType() *{* *audioObjectType;* *if (audioObjectType == 31) {* *audioObjectType = 32 + audioObjectTypeExt;* *}* *return audioObjectType;* *}*	 5 6 	 *uimsbf* *uimsbf*

3.6.2.2 Полезная нагрузка

3.6.2.2.1 Краткий обзор

Для объекта *NULL* размер полезной нагрузки должен быть целым 16-битным числом со знаком в диапазоне от -32768 до +32767. Размеры полезной нагрузки для всех других звуковых типов объекта определены в соответствующих частях. Полезные нагрузки являются основными объектами, которые будет переносить транспортный уровень систем. Для всех схем кодирования натурального звука выходной сигнал масштабируется до максимума 32767/-32768. Однако компоновщик систем *MPEG*–4 ожидает масштабирование.

Полезные нагрузки без побайтного выравнивания добавляются в конец потока, так как транспортные схемы требуют выравнивания.

3.6.2.2.2 Соответствие полезных звуковых нагрузок блокам доступа и элементарным потокам

3.6.2.2.2.1 *AAC Main*, *AAC LC*, *AAC SSR*, *AAC LTP*

Одной полезной нагрузке верхнего уровня (*raw_data_block* ()) соответствует один блок доступа. Последующие блоки доступа формируют один элементарный поток.

3.6.2.2.2.2 *AAC Scalable*

Одной полезной нагрузке верхнего уровня (*aac_scalable_main_element* (), *ASME*) соответствует один блок доступа. Последующие блоки доступа формируют один элементарный поток.

Одной полезной нагрузке верхнего уровня (*aac_scalable_extension_element* (), *ASEE*) соответствует один блок доступа. Последующие блоки доступа того же самого уровня расширения формируют один элементарный поток. Это приводит к индивидуальным элементарным потокам для каждого уровня.

Потоки последующих уровней зависят друг от друга.

3.6.2.2.2.3 *ER AAC LC*, *ER AAC SSR*, *ER AAC LTP*, *ER AAC*, *ER AAC LD*

Элементы данных соответствующих полезных нагрузкок верхнего уровня (*er_raw_data_block* (), *aac_scalable_main_element* (), *aac_scalable_extension_element* ()) подразделены на различные категории в зависимости от ошибочной чувствительности и собраны в экземплярах класса этих категорий. В зависимости от значения *epConfig* есть несколько способов поставить в соответствие эти экземпляры класса блокам доступа для формирования одного или нескольких элементарных потоков. Последующие элементарные потоки зависят друг от друга.

Примечание – Биты функции *byte_alignment* (), завершающей полезные нагрузки верхнего уровня *AAC,* могут быть исключены, если элементы данных согласно полезной нагрузке верхнего уровня не отображаются непосредственно в один блок доступа. Следовательно их можно исключить, если полезная нагрузка верхнего уровня разбита (например, в случае *epConfig*=1) или подвергнута пост-обработке (например, в случае *epConfig*=3).

3.6.2.2.2.4 *ER AAC ELD*

Полезная нагрузка верхнего уровня для *ER AAC ELD* определена в *er_raw_data_block_eld* (). Все определения, упомянутые в 3.6.2.2.2.3, также верны для этого *AOT*.

3.6.3 Семантика

3.6.3.1 *AudioObjectType*

Пять бит, указывающие тип звукового объекта. Это главный переключать, определяющий фактический синтаксис полезного битового потока звуковых данных. В целом различные типы объектов используют различный синтаксис полезного битового потока. Значения комбинаций битов даны в таблице1.

3.6.3.2 *AudioObjectTypeExt*

Этот элемент данных расширяет диапазон типов звуковых объектов.

3.6.3.3 *SamplingFrequency*

Частота дискретизации, используемая в этом звуковом объекте. Передается либо непосредственно значение, либо код *samplingFrequencyIndex*.

3.6.3.4 *SamplingFrequencyIndex*

Четыре бита, указывающие частоту дискретизации. Если *samplingFrequencyIndex* равно15, то фактическое значение частоты дискретизации содержится в *samplingFrequency*. Во всех других случаях *samplingFrequency* установлено в соответствии с таблицей17.

Таблица 17 – Индекс частоты дискретизации

samplingFrequencyIndex	Значение
0x0	96000
0x1	88200
0x2	64000
0x3	48000
0x4	44100
0x5	32000
0x6	24000
0x7	22050
0x8	16000
0x9	12000
0xa	11025
0xb	8000
0xc	7350
0xd	зарезервировано
0xe	зарезервировано
0xf	*escape value*

3.6.3.5 Конфигурация каналов

Четыре бита, указывающие конфигурацию выходных звуковых каналов.

Конфигурация каналов приведена в таблице 18.

Таблица 18 – Конфигурация каналов

Значе-ние	Количество каналов	Синтаксические элементы аудио, перечисленные в порядке приема	Отображение канала к динамику
0	–	-	определено в AOT *related SpecificConfig*
1	1	*single_channel_element*()	центральный фронтальный динамик
2	2	*channel_pair_element*()	левый, правый фронтальные динамики
3	3	*single_channel_element*(), *channel_pair_element*()	центральный фронтальный динамик, левый, правый фронтальные динамики
4	4	*single_channel_element*(), *channel_pair_element*(), *single_channel_element*()	центральный фронтальный динамик, левый, правый центральные фронтальные динамики, задние динамики объемного звучания
5	5	*single_channel_element*(), *channel_pair_element*(), *channel_pair_element*()	центральный фронтальный динамик, левый, правый фронтальные динамики, левый, правый задние динамики объемного звучания
6	5+1	*single_channel_element*(), *channel_pair_element*(), *channel_pair_element*(), *lfe _element*()	центральный фронтальный динамик, левый, правый фронтальные динамики, левый, правый задние динамики объемного звучания, фронтальный динамик низкочастотных эффектов
7	7+1	*single_channel_element*(), *channel_pair_element*(), *channel_pair_element*(), *channel_pair_element*(), *lfe_element*()	центральный фронтальный динамик, левый, правый центральные фронтальные динамики, левый, правый внешние фронтальные динамики, левый, правый задние динамики объемного звучания, фронтальный динамик низкочастотных эффектов
8–15	-	-	зарезервировано

3.6.3.6 Элемент данных *epConfig*

Этот элемент данных сообщает тип схемы обработки ошибок (таблица 19).

Таблица 19 – Элемент данных *epConfig*

epConfig	
0	Все экземпляры класса всех схем обработки ошибок, принадлежащих одному фрейму, расположены в пределах одного блока доступа. Существует один элементарный поток на масштабируемый уровень или один элементарный поток в случае немасштабируемых конфигураций
1	Каждый экземпляр класса всех схем обработки ошибок, принадлежащих одному фрейму, расположен отдельно в пределах отдельного блока доступа, то есть существует столько элементарных потоков, сколько экземпляров класса, определенных в пределах фрейма
2	Необходимо применение декодера защиты от ошибок. Определение классов *EP* нормативно не определяется, однако ему дается определение на прикладном уровне. Тем не менее, ограничения, введенные для классов *EP* в соответствии с 1.6.9, должны быть удовлетворены
3	Необходимо применение декодера защиты от ошибок. Соответствие между классами *EP* и экземплярами класса *ESC* определяется элементом данных *directMapping*

3.6.3.7 Этот элемент данных идентифицирует отображение между классами защиты от ошибок и экземплярами схем обработки ошибок (таблица 20).

Таблица 20 – Элемент данных *directMapping*

directMapping	
0	В резерве
1	Каждый класс защиты от ошибок интерпретируется как экземпляр категории чувствительности к ошибкам (соответствие один к одному), так что выход декодера защиты от ошибок эквивалентен данным *epConfig*=1

3.6.3.8 *extensionSamplingFrequencyIndex*

Четыре бита, указывающие выходную частоту дискретизации инструмента расширения, соответствующего *extensionAudioObjectType*, согласно таблице 18.

3.6.3.9 *extensionSamplingFrequency*

Выходная частота дискретизации инструмента расширения, соответствующего *extensionAudioObjectType*. Передается непосредственно ее значение или код в форме *extensionSamplingFrequencyIndex*.

3.6.3.10 *bits_to_decode*

Вспомогательная функция возвращает число битов, еще не декодированных в текущем *AudioSpecificConfig* (), если о длине этого элемента известно на системном/транспортном уровне. Если размер этого элемента неизвестен, то *bits_to_decode* () возвращает 0.

3.6.3.11 *syncExtensionType*

Синхропоследовательность, определяющая начало данных конфигурации расширения. Эти данные конфигурации соответствуют инструменту расширения, кодированные данные которого помещены (обратно совместимым способом) в *audioObjectType*. Если *syncExtensionType* присутствует, данные конфигурации инструмента расширения отделены от соответствующего *audioObjectType*, который учитывает обратно совместимое сообщение. Декодеры, которые не поддерживают инструмент расширения, могут проигнорировать данные конфигурации. Такая обратносовместимая индикация может использоваться только в системах *MPEG–4*, передающих длину *AudioSpecificConfig* ().

3.6.3.12 *sbrPresentFlag*

Флаг, указывающий присутствие или отсутствие данных *SBR* в случае *extensionAudioObjectType* == 5 (явная сигнализация *SBR*). Значение -1 указывает, что *sbrPresentFlag* не был передан в *AudioSpecificConfig* (). В этом случае декодер профиля *AAC* высокой производительности должен быть в состоянии обнаружить присутствие данных *SBR* в элементарном потоке.

3.6.3.13 *extensionAudioObjectType*

Пять битов, указывающие расширения типа звукового объекта. Этот тип объекта соответствует инструменту расширения, который используется для увеличения *audioObjectType*.

3.6.3.14 *psPresentFlag*

Бит, указывающий присутствие или отсутствие параметрических стерео данных. Значение -1 указывает, что *psPresentFlag* не был передан в *AudioSpecificConfig* (). В этом случае декодер профиля *AAC* высокой производительности *v2* должен поддерживать неявную сигнализацию.

3.6.3.15 *fillBits*

Биты заполнения для побайтного выравнивания *ALSSpecificConfig* () относительно начала *AudioSpecificConfig* ().

3.6.3.16 *extensionChannelConfiguration*

Четыре бита, указывающие конфигурацию каналов расширений *BSAC*. Эти данные конфигурации доступны в случае явной сигнализации расширений *BSAC*. Число выходных звуковых каналов определяется из таблицы 9 конфигурации каналов.

3.6.3.17 *sacPayloadEmbedding*

Флаг типа звукового объекта 30 *MPEG Surround*, используемый для передачи служебной информации кодированного пространственного звука для декодирования. В зависимости от этого флага полезная нагрузка *MPEG Surround*, например *SpatialFrame* (), доступна различными средствами (см. таблицу 21).

Таблица 21 – Элемент данных *sacPayloadEmbedding*

sacPayload-Embedding	
0	Один *SpatialFrame* () соответствует одному блоку доступа. Последующие блоки доступа формируют один элементарный поток, который всегда будет зависеть от другого элементарного потока, содержащего основные (сведенные) звуковые данные
1	Полезная нагрузка верхнего уровня мультиплексируется в основные (сведенные) звуковые данные. Фактические подробности мультиплексирования зависят от представления звуковых данных (то есть обычно от *AOT*). Это приводит к элементарному потоку без реальной полезной нагрузки, который всегда будет зависеть от другого элементарного потока, содержащего как основные (сведенные) звуковые данные, так и мультиплексированные пространственные звуковые данные

3.6.4 Восходящие потоки (потоки клиент-сервер)

3.6.4.1 Введение

Восходящие потоки данных вводятся для того, чтобы позволить удаленному пользователю динамически управлять потоками из сервера.

Сигнал запроса восходящего потока данных поступает на клиентский терминал, предоставляя дескриптор соответствующего элементарного потока с указанием его параметров. Клиентский терминал открывает этот канал восходящего потока данных аналогично открытию нисходящего потока. Объекты (например, кодеры и декодеры), которые подключены через канал восходящего потока данных, известны через параметры дескриптора элементарного потока и ассоциацию дескриптора элементарного потока с определенным дескриптором объекта.

Восходящий поток данных может быть связан с отдельным потоком или группой (нисходящих) потоков. Тип нисходящего потока, связанного с восходящим, определяет область видимости восходящего потока данных. Когда восходящий поток данных связан с отдельным нисходящим потоком, он несет сообщения об этом потоке. Синтаксис и семантика сообщений для *MPEG*–4 Аудио определены в следующем подзаголовке.

3.6.4.2 Синтаксис *AudioUpstreamPayload* () (таблица 22)

Таблица 22 – Синтаксис *AudioUpstreamPayload* ()

Синтаксис	Количество битов	Мнемоника
AudioUpstreamPayload()		
{		
upStreamType;	4	*uimsbf*
switch (upStreamType) {		
case 0: / scalability control */*		
numOfLayer;	6	*uimsbf*
for (layer = 0; layer < numOfLayer; layer++) {		
avgBitrate[layer];	24	*uimsbf*
}		
break;		
case 1: / BSAC frame interleaving */*		
numOfSubFrame;	5	*uimsbf*
break;		
case 2: / quality feedback */*		
multiLayOrSynEle;	1	*uimsbf*
if (multiLayOrSynEle) {		
layOrSynEle;	6	*uimsbf*
}		
else {		
layOrSynEle = 1;		
}		
numFrameExp[layOrSynEle];	4	*uimsbf*
lostFrames[layOrSynEle];	*numFrameExp [layOrSynEle]*	*uimsbf*
break;		
case 3: / bitrate control */*		
avgBitrate;	24	*uimsbf*
break;		
default: / reserved for future use */ break;*		
}		
}		

3.6.4.3 Определения

upStreamType – 4-битовое беззнаковое целое, представляющее тип восходящего потока данных, в соответствии с таблицей 23.

Таблица 23 – Определение *upStreamType*

UpStreamType	Тип восходящего звукового потока
0	контроль масштабируемости
1	объединение *BSAC* фрейма

Окончание таблицы 23

UpStreamType	Тип восходящего звукового потока
2	контроль качества
3	контроль скорости передачи
4–15	зарезервировано для будущего использования

avgBitrate [*level*] – среднее значение скорости передачи в битах в секунду для уровня с большим шагом, который запрашивает клиент для передачи с сервера.

numOfSubFrame – 5-битовое беззнаковое целое, представляющее количество фреймов, сгруппированных для уменьшения загрузки канала передачи. При этом задержка увеличивается пропорционально *numOfSubFrame*.

multiLayOrSynEle - бит сигнализирует об использовании многоканальной или многоуровневой конфигурации. В этом случае требуется передача номеров уровней или элементов.

layOrSynEle – 6-битовое беззнаковое целое, представляющее число синтаксических элементов (в случае многоканального режима) или число уровней (в случае многоуровнего режима), которым соответствует следующая информация контроля качества. Это число относится к одному из уровней или синтаксических элементов, содержащихся в пределах ассоциируемого звукового объекта. Если звуковой объект не поддерживает ни масштабируемость, ни многоканальные возможности, это значение равно 1.

numFrameExp [*layOrSynEle*] – это значение указывает количество последних переданных фреймов ($2^{\text{numFrameExp}} - 1$), которое учитывается в следующем значении *lostFrames*.

lostFrames [*layOrSynEle*] – эта область содержит число потерянных фреймов относительно обозначенного уровня или синтаксического элемента в пределах последних переданных фреймов, о которых сообщает *numFrameExp*.

avgBitrate – среднее значение скорости передачи в битах в секунду целого звукового объекта, который был запрошен клиентом с сервера.

3.6.4.4 Процесс декодирования

В первую очередь *upStreamType* анализируется для обнаружения его типа, от которого зависит остальное декодирование.

3.6.4.4.1 Декодирование управления масштабируемости

Далее декодируется значение *numOfLayer*. Это представляет число элементов данных *avgBitrate*, которое необходимо считать. После этого следует *avgBitratefollows*.

3.6.4.4.2 Декодирование чередования во фрейме *BSAC*

Считывается элемент данных *numOfSubFrame*. Он представляет собой число подфреймов, которые будут чередоваться в инструменте *BSAC*. *BSAC* позволяет корректировку в процессе работы. Далее контент восходящего потока данных передается от клиента на сервер для осуществления динамической и интерактивной передачи. Данные *BSAC* разбиваются и объединяются в сервере.

3.6.4.4.3 Декодирование контроля качества

Процент потерянных фреймов может быть получен, используя следующую формулу

$$frameLossRate[layOrSynEle] = \frac{lostFrames[layOrSynEle]}{2^{\text{numFrameExp[layOrSynEle]}} - 1} * 100\%.$$

3.6.4.4.4 Декодирование контроля скорости передачи

Происходит обнаружение *avgBitrate*.

3.6.5 Сигнализация *SBR*

3.6.5.1 Генерация и сигнализация контента *AAC+SBR*

Инструмент *SBR* в комбинации с кодером *AAC* обеспечивает существенное увеличение эффективности сжатия звука. В то же самое время обеспечивается совместимость с *AAC* декодерами. Однако качество звука декодеров без инструмента *SBR* будет значительно ниже по сравнению с теми, которые поддерживают инструмент *SBR*. Поэтому в зависимости от приложения поставщик контента или его создатель могут выбрать между этими двумя вариантами. Данные *SBR* всегда внедряются в поток *AAC* таким образом, чтобы была обеспечена совместимость с *AAC* (в *extension_payload*), а *SBR* относится

к пост-процессингу в декодере. Это позволяет достичь совместимости. Однако путем различной сигнализации создатель контента может выбрать между режимом полного качества и режимом обратной совместимости способом, указанным ниже.

Ни один из методов сигнализации *SBR*, описанных в этом подзаголовке, не является допустимым для *AAC ELD*. Вместо этого используется флаг *ldSbrPresentFlag* в *ELDSpecificConfig* () для сигнализации использования инструмента *Low Delay SBR*.

3.6.5.1.1 Обеспечение полного качества звучания *AAC+SBR* для слушателя

Чтобы гарантировать получение всеми слушателями полного звукового качества *AAC+SBR*, поток должен соответствовать профилю *HE ACC* и использовать явную сигнализацию, чтобы быть обработанным декодером профиля *HE ACC*. Указанный декодер в состоянии обработать все потоки профиля *AAC* соответствующего уровня, так как профиль *HE ACC* – надмножество профиля *AAC*.

3.6.5.1.2 Достижение обратной совместимости с существующими декодерами *AAC*

Цель этого режима состоит в том, чтобы обеспечить воспроизведение потока на любых *AAC* декодерах, даже если они не поддерживают инструмент *SBR*. Совместимые потоки могут быть созданы следующими двумя способами сигнализации:

а) индикация профиля, содержащего *AAC* данные (например, профиля *AAC*), кроме профиля *HE ACC*, и использование явной обратносовместимой сигнализации (как описано ниже). Этот метод рекомендуется для всех основных систем *MPEG–4*, в которых длина *AudioSpecificConfig* () известна декодеру. Из-за проблемы с *LATM* при *audioMuxVersion* == 0 (см. 1.7) этот метод не может использоваться для *LATM* с *audioMuxVersion* == 0. При явной обратносовместимой сигнализации данные *SBR* добавляются в конец *AudioSpecificConfig* (). Декодеры, которые не поддерживают *SBR*, проигнорируют эти части, в то время как декодеры профиля *HE AAC* обнаружат ее присутствие и сконфигурируют декодер соответствующим образом;

б) индикация профиля, содержащего *AAC*, кроме профиля *HE AAC*, и использование неявной сигнализации. В этом режиме нет никакой явной индикации присутствия *SBR* данных. Декодеры проверяют наличие данных при декодировании потока и используют инструмент *SBR*, если данные *SBR* найдены. Это возможно, потому что *SBR* может быть декодирован без конфигурационных данных *SBR* при определенном способе выбора выходной частоты дискретизации, как описано ниже для декодеров профиля *HE AAC*.

Оба метода приводят к тому, что часть *AAC* потоков *AAC+SBR* будет декодирована декодерами *AAC*. Декодеры *AAC+SBR* обнаружат присутствие *SBR* и декодируют полный качественный поток *AAC+SBR*.

3.6.5.2 Неявная и явная сигнализация *SBR*

В этом подразделе описаны различные методы сигнализации SBR и соответствующее поведение декодеров.

Есть несколько способов сигнализаци данных *SBR*:

1 Неявная сигнализация: Если *EXT_SBR_DATA* или *EXT_SBR_DATA_CRC* элементы *extension_payload* () обнаружены в полезном битовом потоке, это неявно сигнализирует о присутствии данных *SBR*. Способность обнаружить и декодировать *SBR* с неявной сигнализацией обязательна для всех декодеров профиля *High Efficiency AAC* (*HE AAC*).

2 Явная сигнализация: наличие данных *SBR* сигнализируется явным образом посредством типа звукового объекта *SBR* в *AudioSpecificConfig* (). Когда используется явная сигнализация, неявная не должна применяться. Допустимы два различных типа явной сигнализации:

а) иерархическая сигнализация: если первый сигнализируемый *audioObjectType* (*AOT*) является *SBR AOT*, то сигнализируется второй тип звукового объекта, который указывает на основной тип. Этот тип сигнализации не является обратносовместимым. Он может быть необходим в системах, которые не передают длину *AudioSpecificConfig* (), таких как *LATM* с *audioMuxVersion* == 0, где производители контента должны использовать это только по необходимости;

б) обратносовместимая сигнализация: *extensionAudioObjectType* сигнализируется в конце *AudioSpecificConfig* (). Этот метод должен использоваться в системах, которые передают длину *AudioSpecificConfig* (). Следовательно это не должно использоваться для *LATM* с *audioMuxVersion* == 0.

Таблица 24 отражает поведение декодера в зависимости от профиля и индикации типа звукового объекта, когда используется неявная или явная сигнализация.

Таблица 24 – Сигнализация *SBR* и соответствующее поведение декодера

Характеристики полезной нагрузки потока битов				Поведение декодера	
Индикация профиля	*AudioObjectType* расширения	Флаг *sbrPresent*	*raw_data_block*	Декодер профиля *AAC*	Профиль *HE AAC*
Поддержка профилей с AAC, иных, чем высокоэффективный профиль *AAC*	!= *SBR* (сигнализация 1)	-1	*AAC*	Работа *AAC*	Работа *AAC* (Примечание 1)
			AAC+SBR	Работа *AAC*	Работа, по крайней мере, *AAC* должно работать (Примечание 1)
	== *SBR* [сигнализация 2.б]	0	*AAC*	Работа *AAC*	Работа *AAC* (Примечание 2)
		1	*AAC+SBR*	Работа *AAC*	Работа, по крайней мере, *AAC* должно работать *AAC+SBR* (Примечание 3)
Высокоэффективный профиль *AAC*	== *SBR* [сигнализация 2.а или 2.б]	1	*AAC+SBR*	Не определено	Работа *AAC+SBR* (Примечание 3)

Примечание 1 – Неявная сигнализация: необходимо проверить полезную нагрузку для определения выходной частоты дискретизации или принять наличие данных *SBR* в полезной нагрузке, означающих, что выходная частота дискретизации равна удвоенной частоте дискретизации *samplingFrequency* в *AudioSpecificConfig* () (до тех пор, пока инструмент *SBR* не используется или двойная частота дискретизации, обозначенная как *samplingFrequency*, превышает максимальное позволенное для уровня значение, выходная частота дискретизации равна *samplingFrequency*).

Примечание 2 – Явная сигнализации указывает на отсутствие данных *SBR*, следовательно, отсутствие неявной сигнализации; выходная частота дискретизации равна *samplingFrequency* в *AudioSpecificConfig* ().

Примечание 3 – Выходная частота дискретизации равна *extensionSamplingFrequency* в *AudioSpecificConfig* ().

3.6.5.3 Поведение декодера профиля *HE AAC* в случае неявной сигнализации

Если присутствие данных *SBR* сигнализируется обратносовместимым методом [сигнализация 2.а], *extensionAudioObjectType* не является *SBR AOT*, и *sbrPresentFlag* установлен в -1, указывая, что неявная сигнализация может произойти.

Так как декодер профиля *HE AAC* представляет собой систему с двойной скоростью, с инструментом *SBR*, работающим на удвоенной частоте дискретизации основного декодера *AAC*, выходная частота дискретизации не может быть принята за частоту дискретизации декодера *AAC* из-за неявной сигнализации *SBR*. Декодер должен определить выходную частоту дискретизации любым из следующих двух методов:

- проверить присутствие данных *SBR* в полезном битовом потоке до декодирования. Если никакие данные *SBR* не найдены, выходная частота дискретизации равна *samplingFrequency* в *AudioSpecificConfig* (). Если данные *SBR* найдены, то частота равна удвоенной samplingFrequency в *AudioSpecificConfig;*

- предположить, что данные *SBR* присутствуют и принять выходную частоту дискретизации в два раза больше, чем в *AudioSpecificConfig* ().

Вышеупомянутое применяется только в том случае, если удвоенная частота дискретизации, сигнализируемая в *AudioSpecificConfig* (), не превышает максимальную выходную частоту дискретизации, учитывая текущее значение. Следовательно, для декодера профиля *HE AAC* уровней 2, 3 или 4 выходная частота дискретизации равна типовой, которая сигнализируется в *AudioSpecificConfig* (), если последняя превышает 24 кГц.

Инструмент *SBR* с децимацией должен использоваться, если нужно гарантировать, что выходная частота дискретизации не будет превышать максимальное допустимое значение для текущего уровня декодера профиля *AAC* высокой производительности.

3.6.5.4 Поведение декодера профиля *HE AAC* в случае явной сигнализации

Если присутствие данных *SBR* явно сигнализируется (сигнализация 2), то используется обратносовместимая явная сигнализация [сигнализация 2.б] или несовместимая явная сигнализация [сигнализация 2.а].

Для обратносовместимой явной сигнализации [сигнализация 2.б] сигнализируемый *extensionAudioObjectType* является *SBR AOT*. Для этого передается обратносовместимая явная сигнализация о *sbrPresentFlag*, который может быть нулем или единицей. Если *sbrPresentFlag* – нуль, это указывает, что данные *SBR* отсутствуют и, следовательно, декодер профиля *HE AAC* не должен проверять *Fill*-элемент на наличие данных *SBR* или делать предположение о выходной частоте дискретизации в ожидании данных *SBR*. Если *sbrPresentFlag* – один, данные SBR присутствуют и декодер профиля *HE AAC* должен управлять инструментом *SBR*.

Для несовместимой явной сигнализации *SBR* [сигнализация 2.а] сигнализируемый *extensionAudioObjectType* является *SBR AOT*. Для этой иерархической явной сигнализации *sbrPresentFlag* установлен, если *extensionAudioObjectType* – *SBR*. *sbrPresentFlag* не передается и, следовательно, невозможно явно сигнализировать об отсутствии неявной сигнализации. Следовательно при иерархической явной сигнализации данные *SBR* всегда присутствуют, и декодер профиля *HE AAC* должен управлять инструментом *SBR*.

Инструмент *SBR* с децимацией должен использоваться, если выходная частота дискретизации превысит максимально допустимую выходную частоту дискретизации для текущего уровня или если *extensionSamplingFrequency* равна *samplingFrequency*.

3.6.6 Сигнализация параметрического стерео (*PS*)

3.6.6.1 Генерация и сигнализация контента *HE AAC + PS*

Инструмент *PS* в комбинации с кодером *HE AAC* обеспечивает хорошее качество стерео при очень низких скоростях передачи. В то же самое время он позволяет обеспечить совместимость с существующими декодерами *HE AAC*. Однако выход декодера *HE AAC* будет только моно для потока *HE AAC v2*, содержащего данные *PS*.

Следовательно, в зависимости от приложения поставщик контента или создатель контента могут выбрать из двух вариантов, данных ниже. В целом данные *PS* всегда помещаются в поток *HE AAC* совместимым образом (в *sbr_extension* элемент), и *PS* полностью принадлежит к пост-обработке. Это позволяет достичь совместимости. Посредством различной сигнализации создатель контента может выбрать между режимом полного качества и режимом обратной совместимости, как показано в 3.6.6.1.1 и 3.6.6.1.2.

Для иерархических профилей более высокий профиль в иерархии в состоянии декодировать контент более низкого профиля иерархии. На рисунке 1 отображена иерархическая структура профилей *AAC*, *HE AAC* и *HE AAC v2*. Из рисунка видно, что декодер профиля *HE AAC* полностью способен декодировать любой поток *AAC*, при условии, что декодер профиля *HE AAC* имеет тот же или более высокий уровень. Так же декодер профиля *HE AAC v2* определенного уровня может обработать все потоки профиля *HE AAC* того же самого уровня или ниже, так же как и все потоки профиля *AAC* того же самого или более низкого уровня.

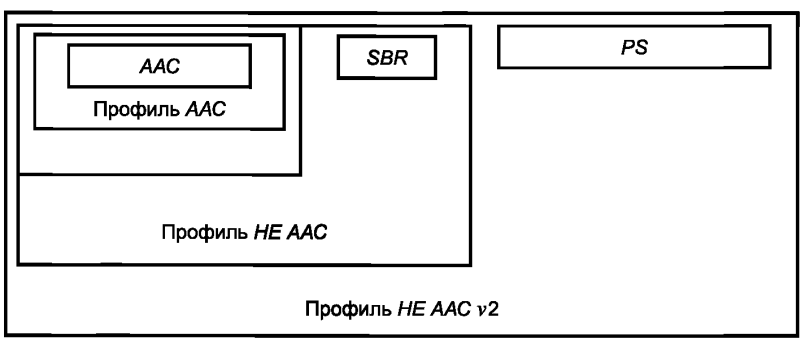


Рисунок 1 – Структура иерархии профилей *AAC*, *HE AAC* и *HE AAC v2* и совместимость между ними

3.6.6.1.1 Обеспечение полного звукового качества *AAC+SBR+PS* для слушателя

Чтобы гарантировать получение слушателями полного звукового качества *AAC+SBR+PS*, поток должен соответствовать профилю *HE AAC v2* и использовать явную иерархическую сигнализацию [сигнализация 2.а, как описано ниже] так, чтобы он был воспроизведен декодерами профиля *HE AAC v2*, то есть *PS* декодерами. Декодер профиля *HE AAC v2* будет декодировать все потоки профиля *HE AAC* и *AAC* соответствующего уровня, так как профиль *HE AAC v2* является надмножеством профилей *HE ACC* и *AAC*.

3.6.6.1.2 Достижение обратной совместимости с существующими *HE AAC* и *AAC*

Цель этого режима состоит в обеспечении возможности воспроизведения потока на всех декодерах профилей *AAC* и *HE AAC*, даже если они не поддерживают инструмент *PS*. Совместимые потоки могут быть созданы следующими двумя способами сигнализации:

а) индикация профиля, содержащего *SBR* данные (например, профиля *HE AAC*), кроме профиля *HE ACC v2*, и использование явной обратносовместимой сигнализации [2б, как описано ниже]. Этот метод рекомендуется для всех *MPEG–4* систем, в которых длина *AudioSpecificConfig* () известна декодеру. Из-за проблемы с *LATM* при *audioMuxVersion* == 0 (см. 1.7) этот метод не может использоваться для *LATM* с *audioMuxVersion* == 0. При явной обратносовместимой сигнализации данные *PS* добавляются в конец *AudioSpecificConfig* (). Декодеры, которые не поддерживают *PS*, проигнорируют эти части, в то время как декодеры профиля *HE AAC v2* обнаружат их присутствие и сконфигурируют декодер соответствующим образом;

б) индикация профиля, содержащего *SBR* (например, профиля *HE AAC*), кроме профиля *HE AAC v2*, и использование неявной сигнализации. В этом режиме нет никакой явной индикации присутствия *PS* данных.

Вместо этого декодеры профиля *HE AAC v2* открывают два выходных канала для потока, содержащего *SBR* данные с *channelConfiguration*==1, например моно поток с одним канальным элементом, проверяют наличие *PS* данных и применяют инструмент *PS*, если *PS* данные были обнаружены. Это возможно, так как *PS* может быть декодировано без конфигурационных данных, если соблюдается определенный способ декодирования каналов в соответствии с профилем *HE AAC v2*.

Оба метода приводят к тому, что часть *AAC+SBR* потоков *AAC+SBR+PS* будет декодирована декодерами *HE AAC*, а часть *AAC* потока *AAC+SBR+PS* будет декодирована декодерами *AAC*. Декодеры *HE AAC v2* обнаружат присутствие *PS* и декодируют полный качественный поток *AAC+SBR+PS*.

3.6.6.2 Неявная и явная сигнализация параметрического стерео

В этом подразделе описаны различные методы сигнализации *SBR* и соответствующее поведение декодеров.

Есть несколько способов сигнализации данных *PS*:

1 Неявная сигнализация: если *bs_extension_id* равен *EXTENSION_ID_PS*, данные *PS* представлены в элементе *sbr_extension*; это неявно сигнализирует о присутствии данных *PS*. Способность обнаружить и декодировать *PS* с неявной сигнализацией обязательна для всех декодеров профиля *High Efficiency AAC v2* (*HE AAC v2*).

2 Явная сигнализация: наличие данных *PS* сигнализируется явным образом посредством типа звукового объекта *PS* и флага *psPresentFlag* в *AudioSpecificConfig* (). Когда используется явная сигнализация *PS*, неявная не должна применяться. Допустимы два различных типа явной сигнализации:

а) иерархическая сигнализация: Если первый сигнализируемый *audioObjectType* (*AOT*) является *PS AOT*, то *extensionAudioObjectType* устанавливается в *SBR* и сигнализируется второй тип звукового объекта, который указывает на основной тип. Этот тип сигнализации не является обратно совместимым. Он может быть необходимым в системах, которые не передают длину *AudioSpecificConfig* (), таких как *LATM* с *audioMuxVersion* == 0, где производители контента должны использовать это только по необходимости;

б) обратносовместимая сигнализация: *extensionAudioObjectType* сигнализируется в конце *AudioSpecificConfig* (), а флаг *psPresentFlag* передается в конце обратносовместимой явной сигнализации *SBR*, указывая наличие или отсутствие данных *PS*. Этот метод должен использоваться в системах, которые передают длину *AudioSpecificConfig* (). Следовательно, это не должно использоваться для *LATM* с *audioMuxVersion* == 0.

Для всех типов сигнализации параметрического стерео *channelConfiguration* в *audioSpecificConfig* показывает количество каналов потока *AAC*. Таким образом, если данные параметрического стерео

присутствуют, *channelConfiguration* равно 1, т. е. один канальный элемент, а инструмент параметрического стерео произведет два выходных канала на основе одного канального элемента и данных параметрического стерео.

Таблица 25 отражает поведение декодера в зависимости от профиля и индикации типа звукового объекта, когда используется неявная или явная сигнализация.

Таблица 25 – Сигнализация *PS* и соответствующее поведение декодера

Характеристики потока битов				Поведение декодера	
Индикация профиля	Сигнализация *PS*	Флаг *psPresent*	*raw_data_block*	Декодеры профиля *HE AAC*	Декодеры профиля *HE AAC v2*
Профиль *AAC* высокоэффективной сети	Сигнализация 1, неявная сигнализация (первый *AOT* != *PS*)	-1	*AAC+SBR*	Работа *AAC+SBR*	Работа *AAC+SBR* (Примечание 1)
			AAC+SBR+PS	Работа *AAC+SBR*	Работа, по крайней мере *AAC+SBR* должно работать *play AAC+SBR+PS* (Примечание 1)
	Сигнализация 2.б, обратно-совместимая неявная сигнализация (второй *AOT* == *SBR*)	0	*AAC+SBR*	Работа *AAC+SBR*	Работа *AAC+SBR* (Примечание 2)
		1	*AAC+SBR+PS*	Работа *AAC+SBR*	Работа, по крайней мере *AAC+SBR* должно работать *AAC+SBR+PS* (Примечание 3)
Профиль высокоэффективный *AAC v2*	Сигнализация 2.а, обратно несовместимая сигнализация (первый *AOT* == *PS*)	1	*AAC+SBR+PS*	Не определено	Работа *AAC+SBR+PS* (Примечание 3)
	Сигнализация 2.б, обратносовместимая сигнализация (второй *AOT* == *SBR*)	1	*AAC+SBR+PS*	Не определено	Работа *AAC+SBR+PS* (Примечание 3)

Примечание 1 – Неявная сигнализация; принять наличие данных *PS* в полезной нагрузке, получив два выходных канала из одного элемента канала.

Примечание 2 – Явная сигнализация о том, что нет никаких данных *PS*, следовательно, отсутствует неявная сигнализация.

Примечание 3 – Число выходных каналов равно двум для одного элемента канала, содержащего *AAC+SBR+PS* данные.

3.6.6.3 Поведение декодера профиля *HE AAC v2* в случае неявной сигнализации

Если присутствие данных *PS* сигнализируется неявным обратносовместимым образом [сигнализация 1], то первый сигнализируемый *AudioObjectType* не является *PS AOT* и флаг *psPresentFlag* не считывается из *AudioSpecificConfig*(). Далее флаг *psPresentFlag* устанавливается в -1, показывая что неявная сигнализация параметрического стерео может применяться.

Так как полученный моно поток будет преобразован в стерео выход при наличии данных параметрического стерео в потоке, то декодер профиля *HE AAC v2* должен подразумевать наличие данных *PS* и принимать решение о количестве выходных каналов, равном 2 для одного канального элемента, содержащего *SBR* данные и, возможно, *PS* данные. Если данных *PS* не было обнаружено, моно выход должен быть поставлен в соответствие двум открытым каналам для каждого отдельного канального элемента.

3.6.6.4 Поведение декодера профиля *HE AAC v2* в случае явной сигнализации

Если присутствие данных *PS* сигнализируется явным образом (сигнализация 2), то для этого используется обратносовместимая явная сигнализация [сигнализация 2.б] или явная сигнализация без обратной совместимости [сигнализация 2.а].

При обратносовместимой явной сигнализации [сигнализация 2.б] сигнализируемый *extensionAudioObjectType* является *SBR AOT*. Явная сигнализация *PS* выполняется посредством *psPresentFlag*, который может быть нулем или единицей.

Если *psPresentFlag* равен нулю, это указывает, что данные *PS* отсутствуют и, следовательно, декодер профиля *HE AAC v2* не должен принимать решение о числе выходных каналов в ожидании данных *PS* (как в случае неявной сигнализации *PS*), и вместо этого использовать оригинальный *channelConfiguration*. Если *psPresentFlag* равен одному, данные *PS* присутствуют и декодер профиля *HE AAC v2* должен управлять инструментом *PS*.

Для явной сигнализации *PS* без обратной совместимости [сигнализация 2.a] первый сигнализируемый *AudioObjectType* является *PS AOT*. *extensionAudioObjectType* назначают *SBR AOT*. Для этой иерархической явной сигнализации флаг *psPresentFlag* устанавливается в единицу, если первый сигнализируемый *AOT – PS AOT*. *psPresentFlag* не передается и, следовательно, невозможно сигнализировать явным образом об отсутствии неявной сигнализации. Следовательно, для иерархической явной сигнализации параметрического стерео данные *PS* всегда присутствуют и декодер профиля *HE AAC v2* должен управлять инструментом *PS*.

3.6.7 Интерфейс между частями Аудио и Системами

3.6.7.1 Введение

В этой части описывается интерфейс между *MPEG–4* Аудио и *MPEG–4* Системами.

Каждый блок доступа, доставленный звуковому декодеру системным интерфейсом, должен привести к соответствующему композитному блоку, доставляемому от звукового декодера к системному интерфейсу. Это должно включать условия начала и завершения, то есть, когда блок доступа будет первым или последним в конечной последовательности блоков доступа.

3.6.7.2 Обработка временных меток композиции

Для блока звуковой композиции временная метка композиции (*CTS*) определяет, что время композиции относится к *n*-му звуковому отсчету в пределах блока композиции. Значение *n* принимается равным 1, если не указано другое.

Для сжатых данных, таких как кодированный звук *HE AAC*, которые могут быть декодированы декодерами различных профилей, декодирование может быть выполнено как обратносовместимым путем (только *AAC*), так и в расширенном виде (*AAC+SBR*). Чтобы обеспечить корректную обработку временных меток композиции (так, чтобы звук остался синхронизированным с другими медиа), применяется следующее:

если сжатые данные позволяют как обратносовместимое, так и расширенное декодирование, и если декодер работает обратносовместимым образом, то ему не нужно выполнять никаких специальных действий. В этом случае, значение *n* равно 1;

если сжатые данные позволяют как обратносовместимое, так и расширенное декодирование, и если декодер работает в расширенном режиме, используя постпроцессор, который вызывает некоторую дополнительную задержку (например, постпроцессор *SBR* в *HE AAC*), то необходимо учесть эту дополнительную задержку с помощью конкретного значения *n* из таблицы 26.

Таблица 26 – Обработка временной метки композиции для различных режимов декодера

Значение *n*	Дополнительная задержка (примечание)	Режим работы декодера
1	0	а) Все операционные режимы, не перечисленные в этой таблице
963	962	б1) Декодер *HE-AAC* или *HE-AAC v2* с *SBR*, работающем в режиме двойной скорости; декодируя компрессированное *HE-AAC* или *HE-AAC v2* аудио
482	481	б2) То же самое, как в б1), но с *SBR*, работающем в режиме субдискретизации
Значение *n*	Дополнительная задержка (Примечание)	Режим работы декодера

Примечание – Задержка из-за постпроцессинга дается в количестве отсчетов (на один звуковой канал) при выходной частоте дискретизации для данного режима декодера.

3.6.8 Сигнализация полезных нагрузок расширения *BSAC*

Метод неявной сигнализации полезных нагрузок расширения *BSAC* аналогичен тому, что применяется в инструменте *SBR*. Декодер *BSAC*, который может декодировать полезную нагрузку расширения *BSAC*, проверяет наличие расширения для инструмента *SBR* '*EXT_BSAC_SBR_DATA*' в *bsac_raw_data_block* (). Частота дискретизации должна быть обновлена при обнаружении расширения, а инструмент *SBR* должен управляться в режиме двойной скорости.

Декодер расширения *BSAC* проверяет наличие типа расширения для расширения канала *BSAC*, например *'EXT_BSAC_CHANNEL'* в *bsac_raw_data_block* (). В случае многоканального типа расширения номер канала из *AudioSpecificConfig* () для объекта типа *BSAC* Аудио обновляется в зависимости от *'channel_configuration_index'* каждого *extended_bsac_base_element* ().

Когда используется явная сигнализация, неявная сигнализация не должна использоваться. Доступны два различных метода явной сигнализации:

1 Метод явной сигнализации 1: иерархическая сигнализация

Если первым сигнализируемым *audioObjectType* (AOT) является *SBR AOT*, сигнализируется второй тип звукового объекта, который указывает *BSAC ER AOT*. *extensionChannelConfiguration* указывает общее количество каналов в *bsac_raw_data_block* ().

2 Метод явной сигнализации 2: обратносовместимая сигнализация

extensionAudioObjectType сигнализируется в конце *AudioSpecificConfig* (). Если *extensionAudioObjectType - BSAC ER AOT*, то *extensionChannelConfiguration* указывает общее количество каналов в *bsac_raw_data_block* (). Этот метод должен использоваться только в системах, которые передают длину *AudioSpecificConfig* (). Следовательно это не должно использоваться для *LATM* с *audioMuxVersion* == 0.

Таблица 27 объясняет поведение декодера с *SBR* и сигнализацией расширения канала *BSAC*.

Таблица 27 – Сигнализация *SBR* и расширения канала *BSAC* и соответствующее поведение декодера

Характеристики потока битов				Поведение декодера	
AudioObjectType расширения	*sbrPresentFlag*	*extensionChannelConfiguration*	*raw_data_block*	Декодер *BSAC*	Декодер расширений *BSAC*
!= *ER_BSAC* (Неявная сигнализация)	-1 (Примечание 1)	Не доступно	*BSAC*	Работа *BSAC*	Работа *BSAC*
			BSAC+SBR	Работа *BSAC*	Работа, по крайней мере, *BSAC* должно работать *BSAC+SBR*
			BSAC+MC	Работа *BSAC*	Работа, по крайней мере, *BSAC* должно работать *BSAC+MC*
			BSAC+SBR+ MC	Работа *BSAC*	Работа, по крайней мере, *BSAC* должно работать *BSAC+SBR +MC*
== *ER_BSAC* (Неявная сигнализация)	0 (Примечание 2)	== *channelConfiguration* (Примечание 4)	*BSAC*	Работа *BSAC*	Работа *BSAC*
		!= *channelConfiguration*	*BSAC+MC*	*Play BSAC*	*Play BSAC+MC*
	1 (Примечание 3)	== *channelConfiguration* (Примечание 4)	*BSAC+SBR*	*Play BSAC*	*Play BSAC+SBR*
		!= *channelConfiguration*	*BSAC+SBR+ MC*	*Play BSAC*	*Play BSAC+SBR+MC*

Примечание 1 – Неявная сигнализация: проверить полезную нагрузку, чтобы определить выходную частоту дискретизации или принять присутствие данных *SBR* в полезной нагрузке, выбирая выходную частоту дискретизации в два раза выше частоты дискретизации *samplingFrequency* в *AudioSpecificConfig* () (пока не используется инструмент *SBR* с децимацией или двойная частота дискретизации *samplingFrequency* не превышает максимально допустимое значение для текущего уровня, на котором выходная частота дискретизации равна *samplingFrequency*).

Примечание 2 – Явно сигнализируется отсутствие данных *SBR*, следовательно, нет неявной сигнализации и выходная частота дискретизации равна *samplingFrequency* в *AudioSpecificConfig* ().

Примечание 3 – Выходная частота дискретизации равна *extensionSamplingFrequency* в *AudioSpecificConfig* ().

Примечание 4 – Явно сигнализируется отсутствие данных расширения канала BSAC и количество выходных каналов задано *channelConfiguration в AudioSpecificConfig* ().

3.6.9 Информация о звуковом файле для формата файла основная медиа Международной организации по стандартизации

3.6.9.1 Введение

Блок информации о звуковом файле позволяет хранить вспомогательные (незвуковые) данные оригинального звукового файла в файле формата основная медиа Международной организации по стандартизации. Этот файл также обычно содержит сжатые данные звука. Этот блок особенно полезен в комбинации с кодированием звука без потерь (например, *MPEG–4 ALS*, *SLS*), когда представляет интерес восстановление оригинального входного звукового файла.

3.6.9.2 Определение

Контейнер: метаблок

Обязательность: нет

Количество: Ноль или один

Содержит информацию об оригинальном звуковом файле, включая тип файла, и предоставляет блоки *OriginalHeader* (), *OriginalTrailer* () и *AuxData* () через элементный блок.

Блок *OriginalHeader* () содержит часть заголовка оригинального звукового файла; на него делается ссылка из элементного блока. Часть заголовка включает все байты до первого звукового отсчета данных в оригинальном файле.

Блок *OriginalTrailer* () содержит трейловую часть оригинального звукового файла; на него делается ссылка из элементного блока. Часть трейла включает все байты после последнего звукового отсчета в оригинальном файле.

Блок *AuxData* () содержит дополнительную информацию, добавленную кодером, относящуюся к оригинальному звуковому файлу; на него делается ссылка элементного блока. Содержание *AuxData* () не используется при восстановлении оригинального файла.

Информация об оригинальном звуковом файле хранится как метаданные в *MetaBox*. Тип обработчика для этого *MetaBox* – '*oafi*', такой *MetaBox* должен иметь *DataBox*, содержащий *OrigAudioFileInfoRecord*, или идентифицировать первичный элемент, данные которого – *OrigAudioFileInfoRecord*.

3.6.9.3 Синтаксис

Синтаксис блока информации об оригинальном звуковом файле следующий:

```
aligned(8) class DataBox extends FullBox('data', version=0,0) {
}
aligned(8) class OrigAudioFileInfoRecord {
    unsigned int(4)  file_type;
    unsigned int(4)  reserved;
    unsigned int(16) header_item_ID;
    unsigned int(16) trailer_item_ID;
    unsigned int(16) aux_item_ID;
    if (file_type == "1111") string original_MIME_type;
}
```

Синтаксисы *OriginalHeader* (), *OriginalTrailer* () и *AuxData* () даны в таблице 28, таблице 29 и таблице 30. *original_MIME_type* – строка с нулевым символом в конце в символах *UTF*-8, идентифицирующих оригинальный тип файла.

Таблица 28 – Синтаксис *OriginalHeader* ()

Синтаксис	Количество битов	Мнемоника
OriginalHeader() *{* *header length* *orig_header[];* *}*	 64 *header_length* * 8	 *bslbf* *bslbf*

Таблица 29 – Синтаксис *OriginalTrailer* ()

Синтаксис	Количество битов	Мнемоника
OriginalTrailer() *{* *trailer length;* *orig_trailer[];* *}*	 64 *trailer_length* * 8	 *bslbf* *bslbf*

Таблица 30 – Синтаксис *AuxData* ()

Синтаксис	Количество битов	Мнемоника
AuxData() *{* *aux length;* *aux_data[];* *}*	 64 *aux_length* * 8 	 *bslbf* *bslbf*

3.6.9.4 Семантика

file_type описывает тип оригинального входного файла (см. таблицу 31 для описания возможных значений).

Таблица 31 – Значения *file_type*

Поле	Количество битов	Описание / Значение
file_type	4	0000 = *unknown / raw file* 0001 = *wave file* 0010 = *aiff file* 0011 = *bwf file* 0100 = *Sony Wave*64 *file (.w*64*)* 0101 = *bwf with RF*64 1111 = *"escape" for MIME type* *(other values are reserved)*

header_item_ID – идентификатор, который ссылается на *OriginalHeader* () в элементном блоке. Если это значение равно 0, *OriginalHeader* () отсутствует. Если *OriginalHeader* () пуст (нулевой длины), то пустой *OriginalHeader* () должен поставляться.

trailer_item_ID – идентификатор, который ссылается на *OriginalTrailer* () в элементном блоке. Если это значение равно 0, *OriginalTrailer* () отсутствует. Если *OriginalTrailer* () пуст (нулевой длины), то пустой *OriginalTrailer* () должен поставляться.

aux_item_ID – идентификатор, который ссылается на *AuxData* () в элементном блоке. Если это значение равно 0, *AuxData* () отсутствует. Если *AuxData* () пуст (нулевой длины), то пустой *AuxData* () должен поставляться.

Элементы *OriginalHeader*, *OriginalTrailer* и *AuxData* даны в таблице 32, таблице 33 и таблице 34.

Таблица 32 – Элементы *OriginalHeader* ()

Поле	Количество битов	Описание / Значения
header_length	64	Размер поля оригинального заголовка в байтах
orig_header[]	*header_length**8	Заголовок оригинального файла аудио

Таблица 33 – Элементы *OriginalTrailer* ()

Поле	Количество битов	Описание / Значение
trailer_length	64	Размер поля вспомогательных данных в байтах
orig_trailer[]	*trailer_length**8	Концевая часть оригинального аудиофайла

Таблица 34 – Элементы *AuxData* ()

Поле	Количество битов	Описание / Значение
aux_length	64	Размер поля вспомогательных данных в байтах
aux_data	*aux_length**8	Вспомогательные данные (для декодирования не требуется)

3.7 Транспортный поток *MPEG*–4 Аудио

3.7.1 Краткий обзор

Транспортный механизм использует двухуровневый подход, а именно: мультиплексный уровень и уровень синхронизации. Мультиплексный уровень (мультиплексный транспортный уровень *MPEG*–4 Аудио с низкой задержкой: *LATM*) управляет мультиплексированием нескольких полезных нагрузок

MPEG–4 Аудио и их элементов *AudioSpecificConfig* (). Уровень синхронизации определяет самосинхронизирующийся синтаксис транспортного потока *MPEG*–4 Аудио, который называют звуковым потоком с низкой задержкой (*LOAS*). Формат интерфейса уровня передачи зависит от основного уровня передачи следующим образом:

- *LOAS* должен использоваться для передачи по каналам, где не доступна синхронизация фреймов;
- *LOAS* может использоваться для передачи по каналам с фиксированной синхронизацией фрейма;
- мультиплексный элемент (*AudioMuxElement* () / *EPMuxElement* ()) без синхронизации будет использоваться только для каналов передачи, где основной транспортный уровень уже обеспечивает синхронизацию фрейма, которая может поддерживать произвольный размер фрейма.

Подробно о форматах *LOAS* и *LATM* описано в 1.7.2 и 1.7.3.

Механизм, определенный в этом параграфе, не должен использоваться для передачи объектов *TTSI*, объектов *Main Synthetic*, объектов *Wavetable Synthesis*, объектов *General MIDI* и объектов алгоритмического синтеза и *FX*. Это не должно использоваться для передачи какого-либо объекта с (*epConfig* == 1). Для таких объектов должны применяться другие мультиплексные и транспортные механизмы, например определенные в *MPEG*–4 Системы.

3.7.2 Уровень синхронизации

Уровень синхронизации предоставляет мультиплексному элементу механизм самосинхронизации для генерации *LOAS*. У *LOAS* есть три различных типа формата, а именно: *AudioSyncStream* (), *EPAudioSyncStream* () и *AudioPointerStream* (). Выбор одного из трех форматов зависит от основного уровня передачи.

AudioSyncStream ()

AudioSyncStream () состоит из синхрослова, мультиплексного элемента с побайтным выравниванием и его информации о длине. Максимальное расстояние между двумя синхрословами составляет 8192 байт. Этот самосинхронизирующийся поток должен использоваться для случая, когда основной уровень передачи следует без какой-либо синхронизации фреймов.

EPAudioSyncStream ()

Для каналов с ошибками предоставлена альтернативная версия *AudioSyncStream* (). Этот формат обладает теми же основными функциональными возможностями, как ранее описанный *AudioSyncStream* (). Однако это дополнительно обеспечивает более длинную синхропоследовательность и счетчик потерянных фреймов. Информация о длине и счетчик фреймов дополнительно защищены *FEC* кодом.

AudioPointerStream ()

AudioPointerStream () должен использоваться для приложений, использующих основной уровень передачи с фиксированной синхронизацией фреймов, где передача кадров не может быть синхронизирована с переменной длиной мультиплексных элементов. Этот формат использует указатель на начало следующего мультиплексного элемента для синхронизации полезной нагрузки переменной длины с постоянным фреймом передачи.

3.7.2.1 Синтаксис (таблицы 35, 36, 37, 38)

Таблица 35 – Синтаксис *AudioSyncStream* ()

Синтаксис	Количество битов	Мнемоника
AudioSyncStream() *{* *while (nextbits() == 0x2B7) { /* syncword */* *audioMuxLengthBytes;*	 11 13	 *bslbf* *uimsbf*

Таблица 36 – Синтаксис *EPAudioSyncStream* ()

Синтаксис	Количество битов	Мнемоника
EPAudioSyncStream() *{* *while (nextbits() == 0x4de1) { /* syncword */* *futureUse;* *audioMuxLengthBytes;*	 16 4 13	 *bslbf* *uimsbf* *uimsbf*

Окончание таблицы 36

Синтаксис	Количество битов	Мнемоника
frameCounter; *headerParity;* *EPMuxElement*(1, 1); } }	5 18	*uimsbf* *bslbf*

Таблица 37 Синтаксис *AudioPointerStream* ()

Синтаксис	Количество битов	Мнемоника

AudioPointerStream (syncFrameLength)
{
while (! EndOfStream) {
AudioPointerStreamFrame (syncFrameLength);
}
}		

Таблица 38 – Синтаксис *AudioPointerStreamFrame* ()

Синтаксис	Количество битов	Мнемоника

AudioPointerStreamFrame(length)
{
audioMuxElementStartPointer;
audioMuxElementChunk;
}
	
	ceil(log2(length))
	length–ceil(log2(length))
		
		uimsbf
		bslbf

3.7.2.2 Семантика

3.7.2.2.1 *AudioSyncStream* ()

audioMuxLengthBytes – 13-битный элемент данных, указывающий длину байта последующего *AudioMuxElement* () с побайтным выравниванием (*AudioSyncStream*) или последующего *EPMuxElement* () (*EPAudioSyncStream*).

AudioMuxElement () – мультиплексный элемент, как определено в 3.7.3.2.2.

3.7.2.2.2 *EPAudioSyncStream* ()

futureUse – 4-битный элемент данных для будущего использования, должно быть установлено в '0000'.

audioMuxLengthBytes (см. 3.7.2.2.1).

frameCounter – 5-битный элемент данных, который используется для обнаружения потерянных фреймов. Число непрерывно увеличивается для каждого мультиплексного элемента.

headerParity – 18-битный элемент данных, который содержит *BCH*(36,18) код, сокращенный от *BCH*(63,45) для элементов *audioMuxLengthBytes* и *frameCounter*. Полиномиальный генератор $x^{18}+x^{17}+x^{16}+x^{15}+x^9+x^7+x^6+x^3+x^2+x+1$. Значение вычисляется при помощи этого генератора, как описано в 3.8.4.3.

EPMuxElement () – эластичный мультиплексный элемент, как определено в 3.7.3.2.1.

3.7.2.2.3 *AudioPointerStream* ()

AudioPointerStreamFrame () – синхронизирующий фрейм фиксированной длины, предоставляемый основным уровнем передачи.

audioMuxElementStartPointer – элемент данных, указывающий начальную точку первого *AudioMuxElement* () в пределах текущего *AudioPointerStreamFrame* (). Число битов, требуемых для этого элемента данных, вычисляется как *ceil*(log2(*syncFrameLength*)). Длина фрейма передачи должна быть получена из основного уровня передачи. Максимально возможное значение этого элемента данных зарезервировано для сигнализации отсутствия начала *AudioMuxElement* () в этом синхронизирующем фрейме.

audioMuxElementChunk – часть конкатенации последующих *AudioMuxElement* ().

3.7.3 Мультиплексный уровень

Уровень *LATM* мультиплексирует несколько полезных нагрузок *MPEG–4* Аудио и *AudioSpecificConfig* () элементов синтаксиса в один мультиплексный элемент. Мультиплексированный формат элемента выбирается между *AudioMuxElement* () и *EPMuxElement* () в зависимости от того, требуется ли способность исправления ошибок в мультиплексном элементе или нет. *EPMuxElement* () является эластичной версией *AudioMuxElement* () и может использоваться для каналов с ошибками.

Мультиплексированные элементы могут быть непосредственно переданы на уровнях передачи с синхронизацией фрейма. В этом случае первый бит мультиплексного элемента должен соответствовать первому биту полезной нагрузки основного уровня передачи. Если полезная нагрузка передачи требует побайтного выравнивания, биты пэддинга должны следовать за мультиплексируемым элементом. Количество битов пэддинга должно быть меньше 8. Эти биты должны быть удалены при демультиплексации элемента в полезные нагрузки *MPEG–4* Аудио. Затем полезные нагрузки *MPEG–4* Аудио передаются соответствующему инструменту декодера *MPEG–4* Аудио.

Использование *LATM* в случае масштабируемых конфигураций с ядром *CELP* и уровнем (уровнями) расширения *AAC*:

- экземпляры класса *AudioMuxElement* () передаются равноотстоящим способом;
- представленный период одного *AudioMuxElement* () подобен суперфрейму;
- отношение числа битов определенного уровня в пределах любого *AudioMuxElement* () к общему количеству битов в пределах этого *AudioMuxElement* () равно отношению скорости передачи того уровня к скорости передачи всех слоев.

В случае, когда *coreFrameOffset* = 0 и *latmBufferFullness* = 0, все основные фреймы кодера и все фреймы *AAC* определенного суперфрейма сохраняются в пределах того же самого экземпляра класса *AudioMuxElement* ().

В случае *coreFrameOffset*> 0 несколько или все основные фреймы кодера сохраняются в пределах предыдущих экземпляров класса *ofAudioMuxElement* ().

Любая информация о конфигурации основного уровня относится к основным фреймам, переданным в пределах текущего экземпляра класса *AudioMuxElement* (), независимо от значения coreFrameOffset:

- указанный *latmBufferFullness* связан с первым фреймом AAC первого суперфрейма, сохраненного в пределах текущего *AudioMuxElement* ();
- значение *latmBufferFullness* может использоваться, чтобы определить местоположение первого бита первого фрейма AAC текущего уровня первого суперфрейма, сохраненного в пределах текущего *AudioMuxElement* () посредством *backpointer*:

backPointer = - *meanFrameLength* + *latmBufferFullness* + *currentFrameLength.*

Значение *backpointer* определяет позицию, как отрицательное смещение от текущего *AudioMuxElement* (), то есть указывает на начало фрейма *AAC*, расположенное в полученных данных. Любые данные, не принадлежащие полезной нагрузке текущего уровня *AAC*, не принимаются во внимание. Если (*latmBufferFullness* == '0'), то фрейм *AAC* начинается после текущего *AudioMuxElement* ().

Возможные конфигурации *LATM* ограничены из-за сигнализации определенных элементов данных следующим образом:

- число уровней: 8 (*numLayer* имеет 3 бита);
- число потоков: 16 (*streamIndx* имеет 4 бита);
- число участков памяти: 16 (*numChunk* имеет 4 бита).

3.7.3.1 Синтаксис (см. таблицы 39, 40, 41, 42, 43, 44).

Таблица 39 – Синтаксис *EPMuxElement* ()

Синтаксис	Количество битов	Мнемоника
EPMuxElement(epDataPresent, muxConfigPresent)		
{		
if (epDataPresent) {		
epUsePreviousMuxConfig;	1	*bslbf*
epUsePreviousMuxConfigParity;	2	*bslbf*
if (!epUsePreviousMuxConfig) {		
epSpecificConfigLength;	10	*bslbf*
epSpecificConfigLengthParity;	11	*bslbf*
ErrorProtectionSpecificConfig();		

Окончание таблицы 39

Синтаксис	Количество битов	Мнемоника
ErrorProtectionSpecificConfigParity(); *}* *ByteAlign();* *EPAudioMuxElement(muxConfigPresent);* *}* *else {* *AudioMuxElement(muxConfigPresent);* *}* *}*		

Таблица 40 – Синтаксис *AudioMuxElement* ()

Синтаксис	Количество битов	Мнемоника
AudioMuxElement(muxConfigPresent) *{* *if (muxConfigPresent) {* *useSameStreamMux;* *if (!useSameStreamMux)* *StreamMuxConfig();* *}* *if (audioMuxVersionA == 0) {* *for (i = 0; i <= numSubFrames; i++) {* *PayloadLengthInfo();* *PayloadMux();* *}* *if (otherDataPresent) {* *for(i = 0; i < otherDataLenBits; i++) {* *otherDataBit;* *}* *}* *}* *else {* */* tbd */* *}* *ByteAlign();* *}*	 1 1	 *bslbf* *bslbf*

Таблица 41 – Синтаксис *StreamMuxConfig* ()

Синтаксис	Количество битов	Мнемоника
StreamMuxConfig() *{* *audioMuxVersion;* *if (audioMuxVersion == 1) {* *audioMuxVersionA;* *else {* *audioMuxVersionA = 0;* *if (audioMuxVersionA == 0) {* *if (audioMuxVersion == 1) {* *taraBufferFullness = LatmGetValue();* *}* *streamCnt = 0;* *allStreamsSameTimeFraming;* *numSubFrames;* *numProgram;* *for (prog = 0; prog <= numProgram; prog++) {* *numLayer;* *for (lay = 0; lay <= numLayer; lay++) {* *progSIndx[streamCnt] = prog; laySIndx[streamCnt] = lay;* *streamID [prog][lay] = streamCnt++;*	 1 1 1 6 4 3	 *bslbf* *bslbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf*

Окончание таблицы 41

Синтаксис	Количество битов	Мнемоника
if (prog == 0 && lay == 0) { *useSameConfig = 0;* *} else {* *useSameConfig;* *}* *if (! useSameConfig)* *if (audioMuxVersion == 0) {* *AudioSpecificConfig();* *}* *else {* *ascLen = LatmGetValue();* *ascLen -= AudioSpecificConfig();* *fillBits;* *}* *}* *frameLengthType[streamID[prog][lay]];* *if (frameLengthType[streamID[prog][lay] == 0) {* *latmBufferFullness[streamID[prog][lay]];* *if (! allStreamsSameTimeFraming) {* *if ((AudioObjectType[lay] == 6 ||* *AudioObjectType[lay] == 20) &&* *(AudioObjectType[lay-1] == 8 ||* *AudioObjectType[lay-1] == 24)) {* *coreFrameOffset;* *}* *}* *} else if (frameLengthType[streamID[prog][lay]] == 1) {* *frameLength[streamID[prog][lay]];* *} else if (frameLengthType[streamID[prog][lay]] == 4 ||* *frameLengthType[streamID[prog][lay]] == 5 ||* *frameLengthType[streamID[prog][lay]] == 3){* *CELPframeLengthTableIndex[streamID[prog][lay]];* *} else if (frameLengthType[streamID[prog][lay]] == 6 ||* *frameLengthType[streamID[prog][lay]] == 7){* *HVXCframeLengthTableIndex[streamID[prog][lay]];* *}* *}* *}* *otherDataPresent;* *if (otherDataPresent) {* *if (audioMuxVersion == 1) {* *otherDataLenBits = LatmGetValue();* *}* *else {* *otherDataLenBits = 0; /* helper variable 32bit */* *do {* *otherDataLenBits *= 2^8;* *otherDataLenEsc;* *otherDataLenTmp;* *otherDataLenBits += otherDataLenTmp;* *} while (otherDataLenEsc);* *}* *}* *crcCheckPresent;* *if (crcCheckPresent) crcCheckSum;* *else {* */* tbd */* *}* *}*	1 3 8 6 9 6 1 1 1 8	*uimsbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf* *uimsbf*

Примечание 1 – *AudioSpecificConfig* () возвращает количество считанных битов.

Таблица 42 – Синтаксис *LatmGetValue* ()

Синтаксис	Количество битов	Мнемоника
LatmGetValue()		
{		
bytesForValue;	2	*uimsbf*
value = 0; / helper variable 32bit */*		
for (i = 0; i <= bytesForValue; i++) {		
*value *= 2^8;*		
valueTmp;	8	*uimsbf*
value += valueTmp;		
}		
return value;		
}		

Таблица 43 – Синтаксис *PayloadLengthInfo* ()

Синтаксис	Количество битов	Мнемоника
PayloadLengthInfo()		
if (allStreamsSameTimeFraming) {		
for (prog = 0; prog <= numProgram; prog++) {		
for (lay = 0; lay <= numLayer; lay++) {		
if (frameLengthType[streamID[prog][lay]] == 0) {		
MuxSlotLengthBytes[streamID[prog][lay]] = 0;		
do { / always one complete access unit */*		
tmp;	8	*uimsbf*
MuxSlotLengthBytes[streamID[prog][lay]] += tmp;		
} while(tmp == 255);		
} else {		
if (frameLengthType[streamID[prog][lay]] == 5 ||		
frameLengthType[streamID[prog][lay]] == 7 ||		
frameLengthType[streamID[prog][lay]] == 3) {		
MuxSlotLengthCoded[streamID[prog][lay]];	2	*uimsbf*
}		
}		
}		
}		
} else {		
numChunk;	4	*uimsbf*
for (chunkCnt = 0; chunkCnt <= numChunk; chunkCnt++) {		
streamIndx;	4	*uimsbf*
prog = progCIndx[chunkCnt] = progSIndx[streamIndx];		
lay = layCIndx[chunkCnt] = laySIndx [streamIndx];		
if (frameLengthType[streamID[prog][lay]] == 0) {		
MuxSlotLengthBytes[streamID[prog][lay]] = 0;		
do { / not necessarily a complete access unit */*		
tmp;	8	*uimsbf*
MuxSlotLengthBytes[streamID[prog][lay]] += tmp;		
} while (tmp == 255);		
AuEndFlag[streamID[prog][lay]];	1	*bslbf*
} else {		
if (frameLengthType[streamID[prog][lay]] == 5 ||		
frameLengthType[streamID[prog][lay]] == 7 ||		
frameLengthType[streamID[prog][lay]] == 3) {		
MuxSlotLengthCoded[streamID[prog][lay]];	2	*uimsbf*
}		
}		
}		
}		
}		

Таблица 44 – Синтаксис *PayloadMux* ()

Синтаксис	Количество битов	Мнемоника
PayloadMux() *if (allStreamsSameTimeFraming) {* *for (prog = 0; prog <= numProgram; prog++) {* *for (lay = 0; lay <= numLayer; lay++) {* *payload [streamID[prog][lay]];* *}* *}* *} else {* *for (chunkCnt = 0; chunkCnt <= numChunk; chunkCnt++) {* *prog = progCIndx[chunkCnt];* *lay = layCIndx [chunkCnt];* *payload [streamID[prog][lay]];* *}* *}* *}*		

3.7.3.2 Семантика

3.7.3.2.1 *EPMuxElement* ()

Для обнаружения *EPMuxElement* () флаг *epDataPresent* должен быть дополнительно установлен в основном уровне. Если *epDataPresent* установлен в 1, это указывает, что у *EPMuxElement* () есть защита от ошибок. В противном случае формат *EPMuxElement* () идентичен *AudioMuxElement* (). Значение по умолчанию для обоих флагов 1.

epDataPresent	Описание
0	*EPMuxElement()* идентично *AudioMuxElement()*
1	*EPMuxElement()* имеет защиту от ошибок

epUsePreviousMuxConfig флаг, указывающий применена ли конфигурация инструмента *EP MPEG–4* Аудио для предыдущего фрейма в текущем фрейме.

epUsePreviousMuxConfig Description	Описание
0	Конфигурация для инструмента *EP MPEG–4* Аудио присутствует
1	Конфигурация для инструмента *EP MPEG–4* Аудио отсутствует. Должна быть применена предыдущая конфигурация

epUsePreviousMuxConfigParity – 2-битный элемент, который содержит паритет для *epUsePreviousMuxConfig*.

Каждый бит – повторение *epUsePreviousMuxConfig*. Решение принимается по максимуму.

epSpecificConfigLength – 10-битный элемент данных, определяющий размер *ErrorProtectionSpecificConfig* ().

epSpecificConfigLengthParity – 11-битный элемент данных для *epHeaderLength*.

Примечание – Это означает, что используется укороченный код *Golay*(23,12).

ErrorProtectionSpecificConfig () – функция, содержащая данные конфигурации для инструмента *EP*, к которому относятся *AudioMuxElement* ().

ErrorProtectionSpecificConfigParity () – функция, содержащая биты четности для *ErrorProtectionSpecificConfig* ().

EPAudioMuxElement () – функция, содержащая эластичный мультиплексный элемент, генерируемый с использованием инструмента *EP* к *AudioMuxElement* (), как определено *ErrorProtectionSpecificConfig* (). Поэтому элементы данных в *AudioMuxElement* () подразделяются на различные категории в зависимости от их чувствительности к ошибкам и собираются в экземплярах класса этих категорий.

Определены следующие категории чувствительности:

Элементы	Категория чувствительности к ошибкам
useSameStreamMux + *StreamMuxConfig* ()	1
PayloadLengthInfo ()	2
PayloadMux ()	3
otherDataBits	4

Примечание 1 – Может быть больше чем один экземпляр класса категорий чувствительности к ошибкам 1 и 2 в зависимости от значения переменной *numSubFrames*, определенных в *StreamMuxConfig* (). Рисунок 2 показывает пример порядка экземпляров класса при numSubFrames = 1.

Примечание 2 – *EPAudioMuxElement* () должен быть побайтно выровненным, поэтому *bit_stuffing inErrorProtectionSpecificConfig* () должно быть всегда включено.

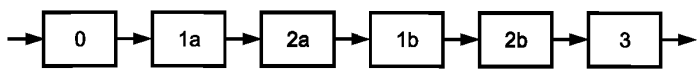


Рисунок 2 – Порядок следования экземпляров класса в *EPAudioMuxElement* ()

3.7.3.2.2 *AudioMuxElement* ()

Для обнаружения *AudioMuxElement* () флаг *muxConfigPresent* должен быть установлен в основном уровне. Если *muxConfigPresent* установлен в 1, это сигнализирует, что конфигурация мультиплексирования (*StreamMuxConfig* ()) является мультиплексируемой в *AudioMuxElement* (), то есть имеет место передача в полосе. В противном случае, *StreamMuxConfig* () должен быть передан вне полосы с использованием протоколов объявления/описания/управления сеансом.

muxConfigPresent	Описание
0	Передача вне полосы *StreamMuxConfig* ()
1	Передача в полосе *StreamMuxConfig* ()

useSameStreamMux – флаг применения конфигурации мультиплексирования предыдущего фрейма в текущем фрейме.

useSameStreamMux	Описание
0	Конфигурация мультиплексирования присутствует
1	Конфигурация мультиплексирования отсутствует. Предыдущая конфигурация должна быть применена

otherDataBit – 1-битовый элемент данных, указывающий наличие других данных.

3.7.3.2.3 *StreamMuxConfig* ()

AudioSpecificConfig () определен в 3.6.2.1. В таком случае это является автономным элементом.

audioMuxVersion – элемент данных для сигнализации об используемом мультиплексном синтаксисе.

Примечание – В дополнение к (*audioMuxVersion* == 0), (*audioMuxVersion* == 1) поддерживает передачу *taraBufferFullness* и передачу длин отдельных *AudioSpecificConfig* () функции данных.

audioMuxVersionA – элемент данных для сигнализации используемой версии синтаксиса. Возможные значения: 0 (значение по умолчанию), 1 (зарезервировано для будущих расширений).

taraBufferFullness – вспомогательная переменная, указывающая состояние резервуара битов в ходе кодирования информации о статусе *LATM*. Это передается как число доступных битов в резервуаре битов *tara*, деленное на 32 и округленное до целого значения. Максимальное значение, которое допустимо для сигнализации с использованием любой установки сигналов *bytesForValue*, таких, что отдельная программа и уровень имеют переменную скорость. Возможные варианты: *FF* (*bytesForValue* == 0), *FFFF* (*bytesForValue* == 1), *FFFFFF* (*bytesForValue* == 2) или *FFFFFFFF* (*bytesForValue* == 3). В этих случаях заполнение буфера не применимо. Информация о состоянии *LATM taraBufferFullness* включает любые данные *AudioMuxElement* (), кроме *PayloadMux* ().

allStreamsSameTimeFraming – элемент данных, указывающий, все ли полезные нагрузки, мультиплексированные в *inPayloadMux* (), используют общее время.

numSubFrames – элемент данных, указывающий, сколько фреймов *PayloadMux* () мультиплексируются (*numSubFrames*+1). Если мультиплексируется более чем один фрейм *PayloadMux* (), все *PayloadMux* () используют общий *StreamMuxConfig* (). Минимальное значение (0) соответствует 1 подфрейму.

numProgram – элемент данных, указывающий, сколько программ мультиплексируются (*numProgram*+1).

Минимальное значение (0) соответствует 1 программе.

numLayer – элемент данных, указывающий, сколько масштабируемых уровней мультиплексируются (*numLayer*+1).

Минимальное значение (0) соответствует 1 уровню.

useSameConfig – элемент данных, указывающий, что не передается *AudioSpecificConfig* (), однако должен быть применен последний принятый *AudioSpecificConfig* ().

useSameConfig	Описание
0	*AudioSpecificConfig* () присутствует
1	*AudioSpecificConfig* () отсутствует. Должен быть применен *AudioSpecificConfig* () предыдущего уровня или программы

ascLen [*prog*] [*lay*] – вспомогательная переменная, указывающая длину в битах *subsequentAudioSpecificConfig* (), включая возможные биты заполнения.

fillBits - биты заполнения.

frameLengthType – элемент данных, указывающий тип длины фрейма полезной нагрузки. Для объектов *CELP* и *HVXC* длина фрейма (бит/фрейм) сохранена в таблицах; передаются только индексы для указания длины фрейма текущей полезной нагрузки.

frameLengthType	Описание
0	Полезная нагрузка с переменной длиной фрейма. Длина полезной нагрузки в байтах непосредственно определена с 8-битовыми кодами в *PayloadLengthInfo* ()
1	Полезная нагрузка с фиксированной длиной фрейма. Длина полезной нагрузки в битах определена с *frameLength* в *StreamMuxConfig* ()
2	В резерве
3	Полезная нагрузка для *CELP* с одним из 2 видов длины фрейма. Длина полезной нагрузки определена двумя индексами таблиц, а именно, *CELPframeLengthTableIndex* и *MuxSlotLengthCoded*
4	Полезная нагрузка для объектов *CELP* или *ER_CELP* с фиксированной длиной фрейма. *CELPframeLengthTableIndex* определяет длину полезной нагрузки
5	Полезная нагрузка для объекта *ER_CELP* с одним из 4 видов длины фрейма. Длина полезной нагрузки определена двумя индексами таблиц, а именно: *CELPframeLengthTableIndex* и *MuxSlotLengthCoded*
6	Полезная нагрузка для объектов *HVXC* или *ER_HVXC* с фиксированной длиной фрейма. *HVXCframeLengthTableIndex* определяет длину полезной нагрузки
7	Полезная нагрузка для объектов *HVXC* или *ER_HVXC* с одним из 4 видов длины фрейма. Длина полезной нагрузки определена двумя индексами таблиц, а именно: *HVXCframeLengthTableIndex* и *MuxSlotLengthCoded*

latmBufferFullness [*streamID* [*prog*] [*lay*]] - элемент данных, указывающий состояние резервуара битов в ходе кодирования первого блока доступа отдельной программы и уровня в *AudioMuxElement* (). Передается как количество доступных битов в резервуаре битов, деленное на *NCC*, деленное на 32 и округленное до целого значения. Шестнадцатеричное значение *FF* сообщает о том, что отдельная программа и уровень имеют переменную скорость. В этом случае полнота буфера не применима.

В случае (*audioMuxVersion* == 0) биты, потраченные на данные, отличные от любой полезной нагрузки (например, мультиплексная информация или другие данные) определяются первым *latmBufferFullness* в *AudioMuxElement* (). Для *AAC* применяются ограничения, вызванные минимальным входным буфером декодера. В случае (*allStreamsSameTimeFraming* == 1) и присутствия только одной программы и одного уровня, это приводит к конфигурации *LATM*, подобной *ADTS*.

В случае (*audioMUxVersion* == 1) биты, потраченные на данные, отличные от любой полезной нагрузки, определяются *taraBufferFullness*.

coreFrameOffset идентифицирует первый фрейм *CELP* текущего суперфрейма. Это определено только в случае масштабируемых конфигураций с ядром *CELP* и уровнем (уровнями) расширения *AAC* и передается с первым уровнем расширения *AAC*. Значение 0 идентифицирует первый фрейм *CELP* после *StreamMuxConfig* () как первый фрейм *CELP* текущего суперфрейма. Значение > 0 сигнализирует число фреймов *CELP* таких, что первый фрейм *CELP* текущего суперфрейма передан ранее.

frameLength – элемент данных, указывающий длину фрейма полезной нагрузки с *frameLengthType* 1. Длина полезной нагрузки в битах определена как 8 * (*frameLength* + 20).

CELPframeLengthTableIndex – элемент данных, указывающий один из двух индексов, соответствующих длине фрейма для объектов *CELP* или *ER_CELP* (таблица 47 и таблица 48).

HVXCframeLengthTableIndex – элемент данных, указывающий один из двух индексов, соответствующих длине фрейма для объектов *HVXC* или *ER_HVXC* (таблица 46).

otherDataPresent – флаг, указывающий присутствие данных, отличных от звуковых полезных нагрузок.

otherDataPresent	Описание
0	данные не мультиплексированы
1	данные мультиплексированы

otherDataLenBits - вспомогательная переменная, указывающая длину в битах других данных.

crcCheckPresent - элемент данных, указывающий наличие битов *CRC* для *StreamMuxConfig* ().

crcCheckPresent	Описание
0	*CRC* биты не присутствуют
1	*CRC* биты присутствуют

crcCheckSum – проверочная сумма *CRC*. Используется образующий полином *CRC*8, как определено в 3.8.4.5. Охватывает весь *StreamMuxConfig* () до бита *crcCheckPresent* (без его включения в проверку).

3.7.3.2.4 *LatmGetValue* ()

bytesForValue – элемент данных, указывающий число элементов данных *valueTmp*.

valueTmp – элемент данных, используемый для вычисления вспомогательной переменной *value*.

value – вспомогательная переменная, содержащая значение, возвращенное функцией *LatmGetValue* ().

3.7.3.2.5 *PayloadLengthInfo* ()

tmp – элемент данных, указывающий длину полезной нагрузки с *frameLengthType* 0. Значение 255 используется как *escape* и указывает, что, по крайней мере, еще одно значение tmp следует. Полная длина переданной полезной нагрузки вычисляется путем суммирования значений частей.

MuxSlotLengthCoded – элемент данных, указывающий один из двух индексов, соответствующих длине полезной нагрузки для объектов *CELP*, *HVXC*, *ER_CELP* и *ER_HVXC*.

numChunk – элемент данных, указывающий число участков памяти полезной нагрузки (*numChunk*+1). Каждый участок памяти может принадлежать блоку доступа с различным временем; используется, только если *allStreamsSameTimeFraming* установлен на ноль. Минимальное значение (0) соответствует 1 участку памяти.

streamIndx – элемент данных, указывающий поток. Используется, если полезные нагрузки разбивают на участки памяти.

chunkCnt – вспомогательная переменная для подсчета числа участков памяти.

progSIndx, *laySIndx* – вспомогательные переменные для идентификации программы и числа уровня от *streamIndx*.

progCIndx, *layCIndx* – вспомогательные переменные для идентификации программы и числа уровня от *chunkCnt*.

AuEndFlag - флаг, указывающий, является ли полезная нагрузка последним фрагментом в случае, если блок доступа передан по частям.

AuEndFlag	Описание
0	фрагментированная часть не является последней
1	фрагментированная часть – последняя

3.7.3.2.6 *PayloadMux* ()

payload - фактическая звуковая полезная нагрузка любого блока доступа (*allStreamsSameTimeFraming* == 1) или части конкатенации последующих блоков доступа (*allStreamsSameTimeFraming* == 0).

3.7.3.3 Таблицы

Длины фреймов приведены в таблицах 45, 46, 47.

Таблица 45 – Длина фрейма *HVXC,* биты

Фрейм		*MuxSlotLengthCoded*			
frameLengthType[ ]	*HVXCframeLengthTableIndex[ ]*	00	01	10	11
6	0	40			
6	1	80			
7	0	40	28	2	0
7	1	80	40	25	3

Таблица 46 – Длина фрейма *CELP* уровня 0, биты

CELP frameLengh TableIndex	*Fixed-Rate frameLengthType[*	1-*of*-4 *Rates* (сжатие молчания) *frameLengthType[ ]*=5				1-*of*-2 *Rates (FRC) frameLengthType[ ]*=3	
		MuxSlotLengthCoded				*MuxSlotLengthCoded*	
		00	01	10	11	00	01
0	154	156	23	8	2	156	134
1	170	172	23	8	2	172	150
2	186	188	23	8	2	188	166
3	147	149	23	8	2	149	127
4	156	158	23	8	2	158	136
5	165	167	23	8	2	167	145
6	114	116	23	8	2	116	94
7	120	122	23	8	2	122	100
8	126	128	23	8	2	128	106
9	132	134	23	8	2	134	112
10	138	140	23	8	2	140	118
11	142	144	23	8	2	144	122
12	146	148	23	8	2	148	126
13	154	156	23	8	2	156	134
14	166	168	23	8	2	168	146
15	174	176	23	8	2	176	154
16	182	184	23	8	2	184	162
17	190	192	23	8	2	192	170
18	198	200	23	8	2	200	178
19	206	208	23	8	2	208	186
20	210	212	23	8	2	212	190
21	214	216	23	8	2	216	194
22	110	112	23	8	2	112	90
23	114	116	23	8	2	116	94
24	118	120	23	8	2	120	98
25	120	122	23	8	2	122	100
26	122	124	23	8	2	124	102
27	186	188	23	8	2	188	166
28	218	220	40	8	2	220	174
29	230	232	40	8	2	232	186
30	242	244	40	8	2	244	198
31	254	256	40	8	2	256	210
32	266	268	40	8	2	268	222
33	278	280	40	8	2	280	234
34	286	288	40	8	2	288	242
35	294	296	40	8	2	296	250
36	318	320	40	8	2	320	276
37	342	344	40	8	2	344	298
38	358	360	40	8	2	360	314

Окончание таблицы 46

CELP frameLengh TableIndex	*Fixed-Rate frameLengthType[ ]*=4	1-*of*-4 *Rates* (сжатие молчания) *frameLengthType[ ]*=5				1-*of*-2 *Rates (FRC) frameLengthType[ ]*=3	
		MuxSlotLengthCoded				*MuxSlotLengthCoded*	
		00	01	10	11	00	01
39	374	376	40	8	2	376	330
40	390	392	40	8	2	392	346
41	406	408	40	8	2	408	362
42	422	424	40	8	2	424	378
43	136	138	40	8	2	138	92
44	142	144	40	8	2	144	98
45	148	150	40	8	2	150	104
46	154	156	40	8	2	156	110
47	160	162	40	8	2	162	116
48	166	168	40	8	2	168	122
49	170	172	40	8	2	172	126
50	174	176	40	8	2	176	130
51	186	188	40	8	2	188	142
52	198	200	40	8	2	200	154
53	206	208	40	8	2	208	162
54	214	216	40	8	2	216	170
55	222	224	40	8	2	224	178
56	230	232	40	8	2	232	186
57	238	240	40	8	2	240	194
58	216	218	40	8	2	218	172
59	160	162	40	8	2	162	116
60	280	282	40	8	2	282	238
61	338	340	40	8	2	340	296
62–63	Зарезервировано						

Т а б л и ц а 47 – Длина фрейма *CELP* уровней 1–5, биты

CELPframeLenghTableIndex	*Fixed-Rate frameLengthType [ ]*=4	1-*of*-4 *Rates* (сжатие молчания) *frameLengthType[ ]*=5			
		MuxSlotLengthCoded			
		00	01	10	11
0	80	80	0	0	0
1	60	60	0	0	0
2	40	40	0	0	0
3	20	20	0	0	0
4	368	368	21	0	0
5	416	416	21	0	0
6	464	464	21	0	0
7	496	496	21	0	0
8	284	284	21	0	0
9	320	320	21	0	0
10	356	356	21	0	0
11	380	380	21	0	0
12	200	200	21	0	0
13	224	224	21	0	0
14	248	248	21	0	0
15	264	264	21	0	0
16	116	116	21	0	0
17	128	128	21	0	0
18	140	140	21	0	0
19	148	148	21	0	0
20–63	Зарезервировано				

3.8 Защита от ошибок

3.8.1 Краткий обзор инструментов

Для типов звуковых объектов, устойчивых к ошибкам, может быть применен инструмент защиты от ошибок (*EP*). Об использовании этого инструмента сигнализирует поле *epConfig*. Вход декодера инструмента *EP* состоит из блоков доступа с защитой от ошибок. В случае если об использовании декодера инструмента *EP* сигнализирует *epConfig*, применяются следующие ограничения:

- существует один элементарный поток на уровень масштабируемости, или только один элементарный поток в случае немасштабируемых конфигураций;

- на выходе декодера *EP* имеется набор нескольких классов *EP*. Конкатенация классов *EP* на выходе декодера *EP* идентична данным с *epConfig* = 0.

Определение класса *EP* зависит от *epConfig* и *directMapping*. Для *epConfig* = 2 классы *EP* не имеют строгого определения. Их точный контент должен быть определен на прикладном уровне, хотя вышеупомянутые ограничения должны быть выполнены. Для *epConfig* = 3, нормативно определено соответствие между классами *EP* и экземплярами класса категорий чувствительности к ошибкам (*ESCs*). В этом случае о соответствии сообщает *directMapping*. В случае, если *directMapping* = 1, каждый класс *EP* соответствует только одному экземпляру класса категорий чувствительности к ошибкам. Выход декодера *EP* в этом случае идентичен случаю *epConfig* = 1. Рисунок 3 подводит итог использования классов *EP*, в зависимости от значения *epConfig*.

Инструмент защиты от ошибок (инструмент *EP*) обеспечивает неравную защиту от ошибок (*UEP*) для кодеков ИСО/МЭК14496-3. Основные особенности инструмента *EP* следующие:

- обеспечение набора кодов с обнаружением/исправлением ошибок различного масштаба в основной части и в избыточной части;

- обеспечение универсальной и эффективной, с точки зрения канала передачи, защитой от ошибок, которая охватывает как потоки фреймов фиксированной длины, так и потоки фреймов переменной длины;

- обеспечение управления конфигурацией *UEP* с низкими задержками.

Основная идея *UEP* состоит в разделении фреймов на подфреймы согласно чувствительности к ошибкам в символе (эти подфреймы именуются классами в следующих подразделах) и защите этих подфреймов с соответствующим уровнем *FEC* и/или *CRC*. Без этого качество декодированного звука определяется степенью повреждения наиболее чувствительной части. Таким образом самое сильное *FEC*/*CRC* должно быть применено к целому фрейму, что требует гораздо большей избыточности.

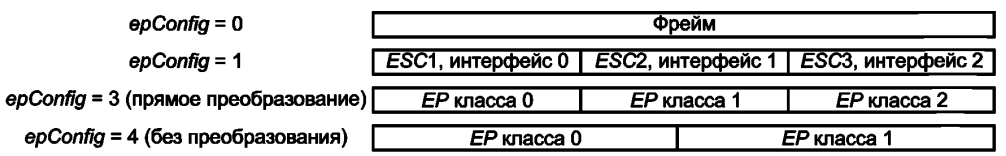


Рисунок 3 – Классы *EP* для различных значений *epConfig*

Чтобы применить *UEP* к звуковым фреймам, должна быть запрошена следующая информация:

1) число классов;

2) число битов, которое содержит каждый класс;

3) код *CRC*, который будет применяться для каждого класса. Представлен как *CRC*-биты;

4) код *FEC*, который будет применяться для каждого класса.

Эту информацию в следующих разделах называют “параметрами конфигурации фрейма”. Та же самая информация используется при декодировании фреймов *UEP*; следовательно они должны быть переданы. Чтобы передать их эффективно, учитываются структуры фреймов *MPEG*–4 Аудио.

Существует три различных подхода к организации структуры фрейма *MPEG*–4 Аудио с точки зрения *UEP*:

1) конфигурация всех фреймов остается постоянной во время передачи (как в *CELP*);

2) используется определенный набор конфигурации (как в *Twin-VQ*);

3) большинство параметров является постоянными при передаче, однако некоторые могут меняться от фрейма к фрейму (как в *AAC*).

Для реализации этих подходов инструмент *EP* использует два пути передачи параметров конфигурации фреймов. Первый представляет собой сигнализацию вне полосы, аналогично передаче параметров конфигурации кодека. Так передаются общие параметры фреймов. В случае, если есть несколько шаблонов конфигурации, эти шаблоны передаются с индексами. Другой путь заключается в передаче внутри полосы, при которой задается структура фрейма *EP* с заголовком. Только те параметры, которые не переданы вне полосы, передаются этим способом. Это позволяет минимизировать количество избыточной внутриполосной информации, произведенной инструментом *EP*.

С этими параметрами каждый класс кодируется и декодируется по *FEC/CRC*. Для увеличения производительности защиты от ошибок используется метод чередования. Цель чередования состоит в рандомизации пакетов ошибок в пределах фреймов, и это нежелательно для незащищенного класса из-за наличия других инструментов обработки ошибок, цель которых состоит в локализации действия ошибок; а рандомизация может оказывать негативное влияние на такую часть полезного битового потока.

Схемы кодера и декодера *EP* представлены на рисунках 4 и 5.

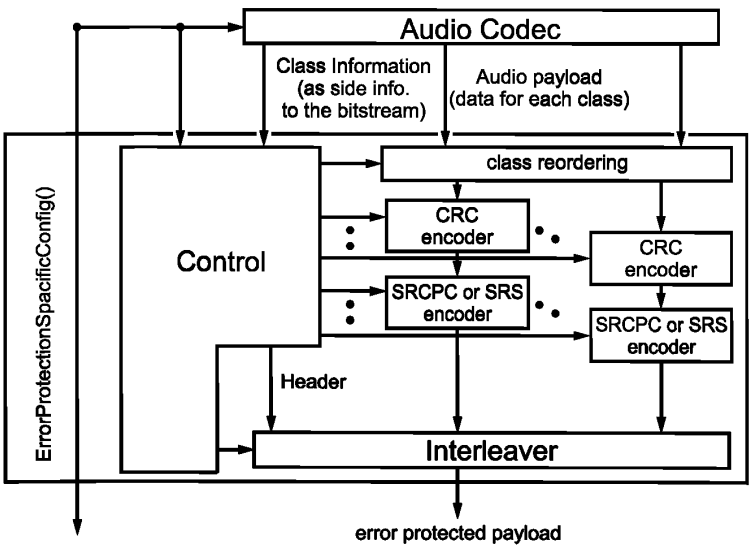


Рисунок 4 - Схема кодера *EP*

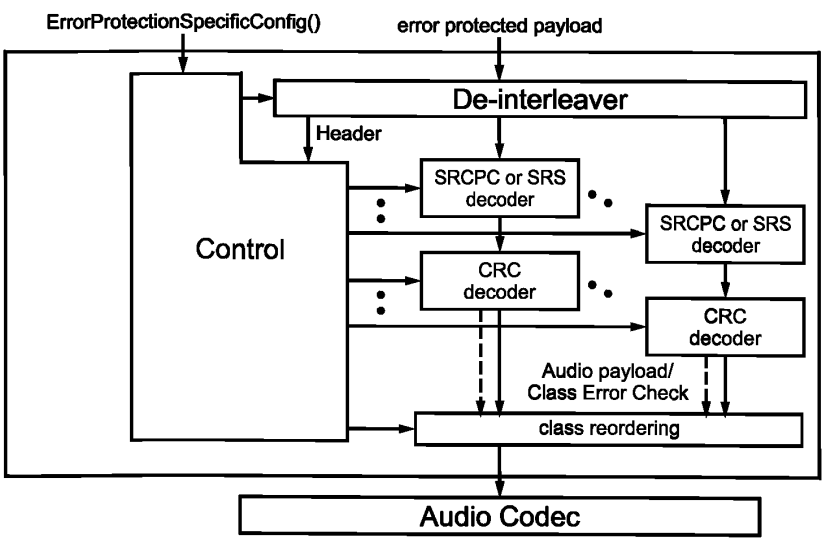


Рисунок 5 - Схема декодера *EP*

3.8.2 Синтаксис

3.8.2.1 Конфигурация защиты от ошибок

Эта часть определяет синтаксис конфигурации для защиты от ошибок (см. таблицу48).

Т а б л и ц а 48 – Синтаксис *ErrorProtectionSpecificConfig* ()

Синтаксис	Количество битов	Мнемоника
ErrorProtectionSpecificConfig()		
number_of_predefined_set;	8	*uimsbf*
interleave_type;	2	*uimsbf*
bit_stuffing;	3	*uimsbf*
number_of_concatenated_frame;	3	*uimsbf*
for (i = 0; i < number_of_predefined_set; i++) {		
number_of_class[i];	6	*uimsbf*
for (j = 0; j < number_of_class[i]; j++) {		
length escape[i][j];	1	*uimsbf*
rate_escape[i][j];	1	*uimsbf*
crclen_escape[i][j];	1	*uimsbf*
if (number_of_concatenated_frame != 1) {	1	*uimsbf*
concatenate flag[i][j];		
}	2	*uimsbf*
fec_type[i][j];		
if(fec_type[i][j] == 0) {	1	*uimsbf*
termination_switch[i][j];		
if (interleave_type == 2) {	2	*uimsbf*
interleave_switch[i][j];		
}	1	*uimsbf*
class optional;		
if (length_escape[i][j] == 1) { / ESC */*	4	*uimsbf*
number_of_bits_for_length[i][j];		
}		
else {	16	*uimsbf*
class length[i][j];		
}		
if (rate_escape[i][j] != 1) { / not ESC */*		
if(fec_type[i][j]){	7	*uimsbf*
class_rate[i][j]		
}else{	5	*uimsbf*
class_rate[i][j]		
}		
}		
if (crclen_escape[i][j] != 1) { / not ESC */*	5	*uimsbf*
class_crclen[i][j];		
}		
}	1	*uimsbf*
class_reordered_output;		
if (class_reordered_output == 1) {		
for (j = 0; j < number_of_class[i]; j++) {	6	*uimsbf*
class_output_order[i][j];		
}		
}		
}	1	*uimsbf*
header_protection;		
if (header_protection == 1) {	5	*uimsbf*
header_rate;	5	*uimsbf*
header_crclen;		
}		
}		

3.8.2.2 Полезные нагрузки защиты от ошибок

Эта часть определяет синтаксис защищенного звукового полезного битового потока. Этот вид синтаксиса может быть выбран установкой *epConfig*=2 или *epConfig*=3. Это характерно для всех типов звуковых объектов. Если используется *MPEG*–4, то один *ep_frame* () непосредственно соответствует одному блоку доступа (см. таблицы 49, 50, 51, 52).

Таблица 49 – Синтаксис *ep_frame* ()

Синтаксис	Количество битов	Мнемоника
ep_frame()		
{		
if (interleave_type == 0){		
ep_header();		
ep_encoded_classes();		
stuffing_bits;	*Nstuff*	*bslbf*
}		
if (interleave_type == 1){		
interleaved_frame_mode1;	1 -	*bslbf*
}		
if (interleave_type == 2){		
interleaved_frame_mode2;	1 -	*bslbf*
}		
}		

Таблица 50 – Синтаксис *ep_header* ()

Синтаксис	Количество битов	Мнемоника
ep_header()		
{		
choice_of_pred;	N_{pred}	*uimsbf*
choice_of_pred_parity;	N_{pred_parity}	*bslbf*
class_attrib();		
class_attrib_parity;	N_{attrib_parity}	*bslbf*
}		

Таблица 51 – Синтаксис *class_attrib* ()

Синтаксис	Количество битов	Мнемоника
class_*attrib()*		
{		
for(j=0; j< number_of_class[choice_of_pred]; j++){		
if(class_reordered_output == 1){		
k = class_output_order[choice_of_pred][j];		
} else {		
k = j;		
}		
if (length_escape[choice_of_pred][k] == 1){		
class_bit_count[k];	*Nbitcount*	*Uimsbf*
}		
if (rate_escape[choice_of_pred][k] == 1){		
class_code_rate[k];	3	*uimsbf*
}		
if (crclen_escape[choice_of_pred][k] == 1){		
class_crc_count[k];	3	*uimsbf*
}		
}		
if (bit_stuffing == 1){		
num_stuffing_bits;	3	*uimsbf*
}		
}		

Таблица 52 – Синтаксис ep_encoded_classes ()

Синтаксис	Количество битов	Мнемоника
ep_encoded_classes() *{* *for(j=0; j< number_of_class[choice_of_pred]; j++){* *if(class_reordered_output == 1){* *k = class_output_order[choice_of_pred][j];* *} else {* *k = j;* *}* *ep_encoded_class[k];* *}* *}*		*bslbf*

3.8.3 Общая информация

3.8.3.1 Определения

ErrorProtectionSpecificConfig () – конфигурация защиты от ошибок, которая является внеполосной информацией;

number_of_predefined_set – номер предопределенного набора;

interleave_type – это переменная определяющая тип чередования. (*interleave_type* == 0), означает «без чередования», (*interleave_type* == 1) означает «внутрикадровое чередование» и (*interleave_type* == 2) включает режим дополнительной настройки чередования для каждого класса. Подробнее дано в 3.8.4.8. (*interleave_type* == 3), зарезервировано;

bit_stuffing – сигнализация использования заполнения для обеспечения побайтного выравнивания:

1 – заполнение используется.

0 – заполнение не используется. Это подразумевает, что для данной конфигурации гарантировано побайтное выравнивание фрейма *EP*;

number_of_concatenated_frame – количество исходных фреймов кодера на один защищенный фрейм.

Объединения фреймов в зависимости от *number_of_concatenated_frame* приведены в таблице 53.

Таблица 53 – Объединение фреймов в зависимости от *number_of_concatenated_frame*

Кодовое слово	000	001	010	011	100	101	110	111
Количество объединяемых фреймов	резерв	1	2	3	4	5	6	7

number_of_class [*i*] – количество классов для *i*-го предопределенного набора;

length_escape [*i*] [*j*] – если 0, то длина *j*-го класса в *i*-ом предопределенном наборе имеет фиксированное значение; если 1, длина является переменной. В случае «*until the end*» это значение должно быть равно 1, и number*_of_bits_for_length* [*i*] [*j*] должно быть равно 0;

rate_escape [*i*] [*j*] – если 0, скорость кода *SRCPC j*-го класса в *i*-ом предопределенном наборе имеет фиксированное значение. Если 1, то о скорости кода сообщается в полосе;

crclen_escape [*i*] [*j*] – если 0, длина *CRC j*-го класса в *i*-ом предопределенном наборе имеет фиксированное значение; если 1, то о *CRClength* сообщается в полосе;

concatenate_flag [*i*][*j*] – параметр, определяющий, связан ли *j*-ый класс *i*-го предопределенного набора или нет. 0 означает “несвязанный”, 1 – «связанный» (см. 3.8.4.4);

fec_type [*i*][*j*] – параметр определяющий, используется ли *SRCPC* код (“0”) или *RS* код (“1” или “2”) для защиты *j*-го класса *i*-го предопределенного набора. Класс, защищенный кодом *RS*, должен быть побайтно выровненным. Если эта область установлена в “2”, это означает, что этот класс защищен *RS* и объединен со следующим классом, как один код *RS*. Если более чем два последующих класса имеют значение “2” для этой области, то эти классы объединены и кодированы как один код *RS*. Если эта область равна “1”, то этот класс не объединяется со следующим. Это означает, что текущий класс – последний, который будет связан перед кодированием *RS*, или этот класс – с независимым *RS* кодированием;

termination_switch [*i*][*j*] – параметр, определяющий, закончен ли *j*-ый класс *i*-го предопределенного набора или нет, при коде *SRCPC*. См. 3.8.4.6.2;

interleave_switch [*i*] [*j*] – параметр, определяющий, как чередовать *j*-ый класс *i*-го предопределенного набора;

0 – без чередования;

1 - чередование без чередования внутри класса: ширина чередования равна числу битов в пределах текущего класса, если (*fec_type* == 0), или числу байтов в пределах текущего класса, если (*fec_type* == 1 || *fec_type* == 2);

2 - чередование с чередованием внутри класса: ширина чередования = 28, если (*fec_type* == 0); это значение зарезервировано, если (*fec_type* == 1 || *fec_type* == 2);

3 - связанный (см. 3.8.4.8.2.2);

class_optional – флаг, сигнализирующий, является ли класс обязательным (*class_optional* == 0) или дополнительным (*class_optional* == 1). Этот флаг может использоваться, чтобы уменьшить избыточность в пределах *ErrorProtectionSpecificConfig*. Обычно это необходимо для определения 2^N наборов, где *N* равняется числу дополнительных классов (см. 3.8.4.2);

number_of_bits_for_length [*i*] [*j*] – область, существующая только когда *length_escape* [*i*] [*j*] = 1. Это значение показывает количество битов для длины класса при сигнализации внутри полосы. Это значение должно быть установлено, если подразумевается максимальная длина класса. Значение 0 соответствует режиму “до конца”;

class_length [*i*] [*j*] – область, существующая только, когда *length_escape* [*i*] [*j*] = 0. Это значение показывает длину *j*-го класса в *i*-ом предопределенном наборе, которая является фиксированной при передаче;

class_rate [*i*] [*j*] – эта область существует только, когда *rate_escape* [*i*][*j*]=0. В случае если *fec_type*[*i*][j] = 0, это значение соответствует скорости кода *SRCPC j*-го класса в *i*-ом предопределенном наборе, фиксированным во время передачи. Значение от 0 до 24 соответствует скорости кода от 8/8 до 8/32 соответственно. В случае если *fec_type* [*i*] [*j*] равен 1 или 2, это значение показывает количество ошибочных байтов, которое может быть исправлено кодом *RS* (см. 3.8.4.7). Все классы, для которых сигнализируется объединение с *fec_type* [*i*] [*j*], должны иметь то же самое значение *class_rate* [*i*] [*j*];

class_crclen [*i*] [*j*] – область, существующая только, когда *crclen_escape* [*i*] [*j*] = 0. Это значение показывает длину *CRC j*-го класса в *i*-ом предопределенном наборе, фиксированным во время передачи. Значение должно быть в диапазоне 0 – 18, что соответствует длине *CRC* 0, 1, 2, 3,4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 24 или 32;

class_reordered_output – если это значение “1”, то выход классов от декодера *EP* переупорядочен. Если “0”, никакой обработки не применяется;

class_output_order [*i*] [*j*] – область, существующая, когда *class_reordered_output* установлен в “1”, для сигнализации порядка класса после переупорядочения. *j*-ый класс *i*-го предопределенного набора выводится как *class_output_order* [*i*] [*j*]-ый класс от *EP* декодера;

header_protection – значение, указывающее режим защиты заголовка от ошибок. 0 указывает на использование основного набора *FEC*, а 1 – на использование расширенной защиты заголовка от ошибок, как определено в 3.8.4.3. Расширенная защита от ошибок заголовка применяется, только если длина заголовка превышает 16 битов;

header_rate, header_crclen – значения, имеющее ту же семантику, что и *class_rate* [*i*] [*j*] и *class_crclen* [*i*] [*j*] соответственно, в то время как эта защита от ошибок используется для защиты части заголовка;

ep_frame () – защищенный фрейм;

ep_header () – кодированная *EP* информация заголовка;

ep_encoded_classes () - *EP* кодированная *EP* звуковая информация;

*interleaved_frame_mode*1 – информационные биты после чередования в режиме 1;

*interleaved_frame_mode*2 – информационные биты после чередования в режиме 2;

stuffing_bits – биты наполнения для выравнивания октета фрейма *EP*. Число битов *Nstuff* сигнализируется в *class_attrib* () и должно быть в диапазоне 0... 7;

choice_of_pred - выбор предопределенного набора;

choice_of_pred_parity - биты четности для *choice_of_pred*;

class_attrib_parity - биты четности для *class_attrib* ();

class_attrib () - атрибутная информация для каждого класса;

class_bit_count [*j*] - число информационных битов в классе. Эта область существует только в случае, если *length_escape* в информации из полосы равен 1 (*Escape*). О количестве битов этого параметра *Nbitcount* сообщается внутри полосы;

class_code_rate [*j*] - скорость кодирования звуковых данных, принадлежащих классу в таблице 54. Эта область существует только в случае, если *rate_escape* в информации из полосы равно 1 (*Escape*).

Таблица 54 – Скорость кодирования для звуковых данных, принадлежащих классу

Кодовое слово	000	001	010	011	100	101	110	111
Скорость преобразования	8/8	8/11	8/12	8/14	8/16	8/20	8/24	8/32
Картина преобразования	FF, 00 00, 00	FF, A8 00, 00	FF, AA 00, 00	FF, EE 00, 00	FF, FF 00, 00	FF, FF AA, 00	FF, FF FF, 00	FF, FF FF, FF

class_crc_count [*j*] – число битов *CRC* для звуковых данных, принадлежащих классу в таблице 55. Эта область существует только в случае, если *crclen_escape* равен 1 (*Escape*).

Число битов *CRC* для звуковых данных, принадлежащих классу, приведены в таблица 55.

Таблица 55 – Число битов CRC для звуковых данных , принадлежащих классу

Кодовое слово	000	001	010	011	100	101	110	111
Число битов *CRC*	0	6	8	10	12	14	16	32

num_stuffing_bits - число битов стаффинга для выравнивания октета фрейма *EP*. Эта область существует только в случае, если *bit_stuffing* равен 1;

ep_encoded_class [*j*] *CRC/SRCPC* – закодированные звуковые данные *j*-го класса. Если *class_bit_count* [*j*] == 0, звуковые данные *j*-го класса не закодированы *CRC*/*SRCPC*/*SRS*.

3.8.4 Описание инструмента

3.8.4.1 Внеполосная информация

Контент внеполосной информации представлен посредством *ErrorProtectionSpecificConfig* (). Некоторые примеры конфигурации представлены в приложении Б.

Длина последнего класса, обработанного декодером *EP* (до любого последующего переупорядочения, как описано в 3.8.4.9), не должна быть передана явно, однако возможна сигнализация “до конца”. В *MPEG–4* Системы системный уровень гарантирует границу звукового фрейма, устанавливая в соответствие один звуковой фрейм одному блоку доступа. Поэтому длина класса «до конца» может быть вычислена по длине других классов и общей длине кодированных *EP* звуковых фреймов.

Флаг *class_optional* может использоваться для уменьшения избыточности в пределах *ErrorProtectionSpecificConfig* (). Однако инструмент *EP* так же работает с тем же самым числом предопределенных наборов. Если есть *N* классов с (*class_optional* == 1), этот предопределенный набор расширяется до 2^N предопределенных наборов. Разворачивание наборов описано в следующем подразделе.

3.8.4.2 Предопределенные наборы

В этом подразделе описывается постобработка, вход которой - *ErrorProtectionSpecificConfig* () с переключателем “*class_optional*”, а выход - предопределенные наборы, используемые для праметров *ep_frame* ().

Общая процедура

Каждый предопределенный набор расширяется до $2^{NCO[i]}$ предопределенных наборов, где *NCO*[*i*] - число классов с (*class_optional* == 1) в *i*-ом оригинальном предопределенном наборе. После этого любой класс с (*class_optional* == 1) упоминается как *optClass*.

Эти расширенные наборы начинаются с “все *optClasses* существуют” и заканчиваются «никаких *optClasses* не существует”. Алгоритм:

```
transPred = 0;
for (i = 0; i < nPred; i++){                    /*for all predefinition sets */
    for (j = 0; j < 2^NCO[i]; j++){                    /* unwraping */
        for (k = 0; k < NCO[i]; k++){                  /* for all optional classes */
            if (j & (0x01 << k)) {
                optClassExists[k] = 0;
```

```
            }
            else {
                optClassExists[k] = 1;
            }
        }
        DefineTransPred(transPred, i, optClassExists);
        transPred ++;
    }
}
```

где *optClassExists* [*k*] сообщает, существует ли *k*-ый *optClass* предопределенного набора (1) или нет (0) в определяющем новом предопределенном наборе.

DefineTransPred (*transPred*, *i*, *optClassExists*) определяет *transPred*-ый новый предопределенный набор, используемый для передачи. Этот новый предопределенный набор – копия *i*-го оригинального предопределенного набора, кроме того, что он не имеет *optClasses*, *optClassExists* которого равняется 0.

Пример

ErrorProtectionSpecificConfig () определяет предопределенные наборы следующим образом: После предварительной обработки, описанной выше, предопределенные наборы, используемые для ep_frame (), устанавливаются следующим образом:

3.8.4.3 Внутриполосная информация

Информация фрейма *EP*, которая не включена во внеполосную информацию, является внутриполосной информацией. Параметры, относящиеся к этой информации, передаются как заголовок фрейма *EP*. Существуют следующие параметры:

- выбор предопределенного набора;
- количество битов стаффинга для побайтного выравнивания;
- информация о классе, которая не включена во внеполосную информацию.

Декодер *EP* не может декодировать звуковую информацию фрейма без этих параметров, и, таким образом, они должны обладать защитой от ошибок, более сильной или равной защите других частей. С учетом этого выбор предопределенного набора должен быть обработан отдельно от других частей. Это следует из того, что длина информации о классе может быть изменена в соответствии с предопределенным набором. Поэтому этот параметр кодируется *FEC* независимо от других частей. На стороне декодера в первую очередь декодируется выбор предопределенного, а затем вычисляется длина оставшейся части заголовка и декодируется.

FEC применяется для этих частей следующим образом:

Основной набор кодов *FEC* приведен в таблице 56.

Таблица 56 – Основной набор кодов FEC для внутриполосной информации

Количество необходимых битов	Код *FEC*	Общее число битов	Длина кодового слова
1–2	Большинство (повторение 3 раза)	3–6	3
3–4	*BCH*(7,4)	6–7	6–7
5–7	*BCH*(15,7)	13–15	13–15
8–12	*Golay*(23,12)	19–23	19–23
13–16	*BCH*(31,16)	28–31	28–31
17	*RCPC* 8/16 + 4-*бит CRC*	50	-

Примечания

1 общее количество битов: число битов после *FEC* кодирования.

2 ширина чередования: ширина матрицы чередования, см. также 3.8.4.8.

3 *Npred_parity* (или *Nattrib_parity*) = общее количество битов минус число битов, которые будут защищены.

4 *SRCPC* завершен.

5 число битов, которые будут защищены, равно *Npred* (или общее количество битов для *class_attrib* ()).

Расширенное *FEC*

Если длина заголовка превышает 16 битов, этот заголовок защищен с помощью *CRC* и завершен *SRCPC*. Скорость *SRCPC* и количество битов *CRC* сигнализируются. Метод кодирования и декодирования такой же, как описано ниже для *CRC*/*SRCPC*.

Порождающие полиномы для каждого *FEC* следующие:

BCH(7,4): x^3+x+1

BCH (15,7): $x^8+x^7+x^6+x^4+1$

Golay (23,12): $x^{11}+x^9+x^7+x^6+x^5+x+1$

BCH (31,16): $x^{15}+x^{11}+x^{10}+x^9+x^8+x^7+x^5+x^3+x^2+x+1$

С этими полиномами *FEC*(*n*, *k*) для *I*-битного кодирования выполняется следующим образом:

Вычисляется полином *R*(*x*), который удовлетворяет

$M(x)\,x^{n-I} = Q(x)G(x) + R(x)$

M(*x*): Информационные биты. Самый высокий порядок соответствует первому биту, который будет передан.

G(*x*): образующий полином, определенный выше.

Этот полином *R* (*x*) производит контроль к *choice_of_pred* или *class_attrib* (), и устанавливается в *choice_of_pred_parity* или *class_attrib_parity* соответственно. Самый высокий порядок соответствует первому биту. Декодер может выполнить коррекцию ошибок, используя эти биты паритета, однако это дополнительная операция.

3.8.4.4 Функциональные возможности конкатенации

У инструмента *EP* есть функциональные возможности для объединения (конкатенации) нескольких исходных фреймов кодера, чтобы создать новый фрейм для инструмента *EP*. При этой конкатенации группы битов, принадлежащих тому же самому классу в различных исходных фреймах кодера, связанны класс с классом. Составные группы, принадлежащие тому же самому классу, либо обрабатываются как отдельный элемент, либо как независимый класс, так же как перед конкатенацией.

О числе фреймов, которые будут связаны, сообщается в *number_of_concatenated_frame* в *ErrorProtectionSpecificConfig (),* и выбор того, обрабатываются ли составные группы, принадлежащие тому же самому классу, как отдельный новый класс, или с независимым классом, с помощью сигнализации *concatenate_flag* [*i*] [*j*] (1 соответствует “отдельному новому одному классу”, а 0 – “независимому классу”).

Тот же самый предопределенный набор должен использоваться для всех составных фреймов. Никакой механизм escape не должен использоваться ни для какого параметра класса.

3.8.4.5 *CRC*

CRC обеспечивает возможность обнаружения ошибок. Информационные биты каждого класса кодируются *CRC*. В этом инструменте определен следующий набор *CRC*:

1-битовый *CRC*1: $x+1$

2-битовый *CRC*2: x^2+x+1

3-битовый *CRC*3: x^3+x+1

4-битовый *CRC*4: $x^4+x^3+x^2+1$

5-битовый *CRC*5: $x^5+x^4+x^2+1$

6-битовый *CRC*6: $x^6+x^5+x^3+x^2+x+1$

7-битовый *CRC*7: $x^7+x^6+x^2+1$

8-битовый *CRC*8: x^8+x^2+x+1

9-битовый *CRC*9: $x^9+x^8+x^5+x^2+x+1$

10-битовый *CRC*10: $x^{10}+x^9+x^5+x^4+x+1$

11-битовый *CRC*11: $x^{11}+x^{10}+x^4+x^3+x+1$

12-битовый *CRC*12: $x^{12}+x^{11}+x^3+x^2+x+1$

13-битовый *CRC*13: $x^{13}+x^{12}+x^7+x^6+x^5+x^4+x^2+1$

14-битовый *CRC*14: $x^{14}+x^{13}+x^5+x^3+x^2+1$

15-битовый *CRC*15: $x^{15}+x^{14}+x^{11}+x^{10}+x^7+x^6+x^2+1$

16-разрядный *CRC*16: $x^{16}+x^{12}+x^5+1$

24-битовый *CRC*24: $x^{24}+x^{23}+x^6+x^5+x+1$

32-разрядный *CRC*32: $x^{32}+x^{26}+x^{23}+x^{22}+x^{16}+x^{12}+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

С этими полиномами кодирование *CRC* выполняется следующим образом:

Вычисляется полином *R*(*x*), такой что

$M(x)x^k = Q(x)G(x) + R(x)$

M(*x*): Информационные биты. Самый высокий порядок соответствует первому биту, который будет передан

G(*x*): Образующий полином, определенный ранее

k: Число битов *CRC*.

С этим полиномом $R(x)$, *CRC* биты $W(x)$, представлены как:

$W(x) = M(x)x^k + R(x)$

Значение k должно быть выбрано так, чтобы число кодированных битов *CRC* не превышало $2k$-1.

Биты *CRC* записываются в обратном порядке, каждый бит инвертирован. Используя эти биты *CRC*, декодер может выполнить обнаружение ошибок. Когда ошибка обнаружена через *CRC*, может быть применена ошибочная маскировка для уменьшения ухудшения качества, вызванного ошибкой. Метод маскировки ошибок зависит от алгоритмов *MPEG*–4 Аудио.

3.8.4.6 Систематические сверточные коды (*SRCPC*)

После *CRC* кодирования выполняется *FEC* кодирование с *SRCPC* кодами. В этом подразделе описывается процесс кодирования *SRCPC*.

Кодер канала основан на систематическом рекурсивном сверточном кодирования (*SRC*) со скоростью R=1/4. Кодированные классы CRC связаны и поступают в этот кодер. Так образуется *RCPC* код, скорость которого изменяется для каждого класса согласно ошибочной чувствительности.

3.8.4.6.1 Генерация *SRC* кода

Код *SRC* генерируется из рациональной порождающей матрицы при использовании петли обратной связи. Реализация сдвигового регистра кодера показана на рисунке 6.

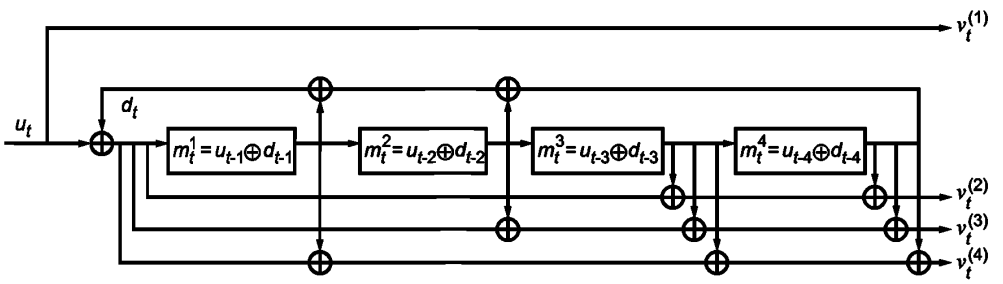


Рисунок 6 – Реализация сдвигового регистра для систематического рекурсивного сверточного кодера

Для получения выходных векторов v_t в момент времени t необходимо знать содержимое сдвиговых регистров m_t^1, m_t^2, m_t^3, m_t^4 (соответствует состоянию) и входной бит u_t во время t.

Получаем выход $v_t^{(2)}$, $v_t^{(3)}$ *и* $v_t^{(4)}$

$V_t^{(2)} = m_t^4 \oplus m_t^3 \oplus (u_t \oplus d_t)$

$V_t^{(3)} = m_t^4 \oplus m_t^3 \oplus m_t^2 \oplus (u_t \oplus d_t)$

$V_t^{(4)} = m_t^4 \oplus m_t^3 \oplus m_t^1 \oplus (u_t \oplus d_t)$

$d_t = m_t^4 \oplus m_t^2 \oplus m_t^1$, $m_t^4 = u_{t-4} \oplus d_{t-4}$, $m_t^3 = u_{t-3} \oplus d_{t-3}$, $m_t^2 = u_{t-2} \oplus d_{t-2}$, $m_t^1 = u_{t-1} \oplus d_{t-1}$

Наконец получаем для выходного вектора $v_t = (v_t^{(1)}, v_t^{(2)}, v_t^{(3)}, v_t^{(4)})$ во время t в зависимости от входного бита u_t и текущего состояния $m_t = (m_t^{(1)}, m_t^{(2)}, m_t^{(3)}, m_t^{(4)})$:

$V_t^{(1)} = u_t$

$V_t^{(2)} = m_t^4 \oplus m_t^3 \oplus (u_t \oplus d_t) = m_t^3 \oplus m_t^2 \oplus m_t^1 \oplus u_t$

$V_t^{(3)} = m_t^4 \oplus m_t^3 \oplus m_t^2 \oplus (u_t \oplus d_t) = m_t^3 \oplus m_t^1 \oplus u_t$

$V_t^{(4)} = m_t^4 \oplus m_t^3 \oplus m_t^1 \oplus (u_t \oplus d_t) = m_t^3 \oplus m_t^2 \oplus u_t$

с $m_1 = (m_1^1, m_1^2, m_1^3, m_1^4) = (0, 0, 0, 0) = 0$

Начальное состояние всегда 0, то есть каждая ячейка памяти содержит 0 перед входом первого информационного бита u_t.

3.8.4.6.2 Завершение кода *SRC*

В случае, если для кодированного *SRC* класс обозначен как окончание в *termination_switch* [*i*] в *inErrorProtectionSpecificConfig* (), или код *SRC* используется для защиты информации в полосе кодера *SRC* и должны добавить биты хвоста в конец этого класса и запустить следующее кодирование *SRC* с начальным состоянием, весь сдвиговый регистр кодера должен быть установлен в 0.

Хвостовые биты после информационной последовательности u для возвращения в состояние $m_n = 0$ (завершение) зависят от последнего состояния m_{n-3} (состояние после входа последнего информационного бита u_{n-4}). Завершающая последовательность для каждого состояния, описываемого m_{n-3}, дана в таблице 57. Приемник может использовать эти хвостовые биты (*TB*) для дополнительного обнаружения ошибок.

Дополнение (u_{n-3}, u_{n-2}, u_{n-1}, u_n) к информационной последовательности может быть вычислено из следующего условия:

для всех t от n-3<=t<=n: $u_t \oplus d_t = 0$

Далее, мы получаем для вектора хвостовых битов $u' = (u_{n-3}, u_{n-2}, u_{n-1}, u_n)$ в зависимости от состояния $m_{n-3} = (m_{n-3}^1, m_{n-3}^2, m_{n-3}^3, m_{n-3}^4)$

$u_{n-3} = d_{n-3} = m_{n-3}^4 \oplus m_{n-3}^2 \oplus m_{n-3}^1$

$u_{n-2} = d_{n-2} = m_{n-2}^4 \oplus m_{n-2}^2 \oplus m_{n-2}^1 = m_{n-3}^3 \oplus m_{n-3}^1 \oplus 0 = m_{n-3}^3 \oplus m_{n-3}^1$

$u_{n-1} = d_{n-1} = m_{n-1}^4 \oplus m_{n-1}^3 \oplus m_{n-1}^2 = m_{n-3}^2 \oplus 0 \oplus 0 = m_{n-3}^2$

$u_n = d_n = m_{n-3}^1 \oplus 0 \oplus 0 = m_{n-3}^1$

Хвостовые биты для систематического рекурсивного сверточного кода приведены в таблице 57.

Таблица 57 – Хвостовые биты для систематического рекурсивного сверточного кода

Состояние m_{n-3}	m_{n-3}^4	m_{n-3}^3	m_{n-3}^2	m_{n-3}^1	u_{n-3}	u_{n-2}	u_{n-1}	u_n
0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	0	1
2	0	0	1	0	1	0	1	0
3	0	0	1	1	0	1	1	1
4	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	0	1
6	0	1	1	0	1	1	1	0
7	0	1	1	1	0	0	1	1
8	1	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	1
10	1	0	1	0	0	0	1	0
11	1	0	1	1	1	1	1	1
12	1	1	0	0	1	1	0	0
13	1	1	0	1	0	0	0	1
14	1	1	1	0	0	1	1	0
15	1	1	1	1	1	0	1	1

3.8.4.6.3 Использование *SRC* для кода *SRCPC*

Прореживание выхода кодера *SRC* позволяет использовать различные скорости для передачи. Таблицы прореживания перечислены в таблице 58.

Таблица 58 – Прореживающие таблицы (все значения в шестнадцатеричном представлении)

Норма r	8/8	8/9	8/10	8/11	8/12	8/13	8/14	8/15	8/16	8/17	8/18	8/19	8/20	8/21	8/22	8/23	8/24	8/25	8/26	8/27	8/28	8/29	8/30	8/31	8/32
$P_r(0)$	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*
$P_r(1)$	00	80	88	*A8*	*AA*	*EA*	*EE*	*FE*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*
$P_r(2)$	00	00	00	00	00	00	00	00	00	80	88	*A8*	*AA*	*EA*	*EE*	*FE*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*	*FF*
$P_r(3)$	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	80	88	*A8*	*AA*	*EA*	*EE*	*FE*	*FF*

Шаблон прореживания, который применяется с периодом 8, зависит от *class_rate* (см. таблицу 58). Каждый бит *Pr*(*i*) указывает, прорежен ли соответствующий символ *vt*(*i*) кодера *SRC* (то есть не рассмотрен). Каждый бит *Pr* (*i*) используется от *MSB* к *LSB*, и 0/1 указывает режим *punctured/not-punctured* соответственно. Шаблон прореживания изменяется от класса к классу, но только в пунктах, где период из 8 закончен. После решения о том, какие биты из *vt*(*i*) учитываются, они выводятся в порядке от *vt*(0) до *vt*(3).

3.8.4.6.4 Декодирование кода *SRCPC*

В декодере может быть выполнена коррекция ошибок с помощью кода *SRCPC*, однако это дополнительная операция, и декодер может извлечь оригинальную информацию, игнорируя биты паритета.

Декодирование *SRCPC* может быть выполнено при помощи алгоритма Витерби для сверточных кодов.

3.8.4.7 Сокращенные коды Рида-Соломона

Сокращенные коды Рида-Соломона *SRS* (255-*I*, 255-2*k*-I), определенные на $GF(2^8)$, могут использоваться для защиты одного отдельного класса или нескольких составных классов. Составные классы впоследствии обрабатываются как один отдельный класс. Здесь *k* – число корректируемых ошибок в одном ключевом слове *SRS*. Значение *I* отражает сокращение.

Перед *SRS* кодированием, класс *EP* подразделяется на части таким образом, чтобы их длины были меньше или равны 255-2*k*. Длины частей вычисляются следующим образом:

I_i = 255-2*k*, при i <*N;*

I_i = *L mod* (255-2*k*), при i = *N;*

L: длина класса *EP* в октетах;

N: число частей;

I_i: длина i-ой части (0 < i <*N*+1).

Если длина *N*-ой части I_N меньше, чем 255-2*k* байт, добавляется так много битов со значением 0, как требуется для того, чтобы достигнуть длины 255-2*k* байт перед SRS кодированием/декодированием, и в обратном порядке.

На стороне декодера, если выполняется декодирование SRS, то же самое число нулевых бит должно быть добавлено перед процедурой *SRSdecoding*, и удалено после *SRS* декодирования.

SRS код, определенный в поле Галуа $GF(2^8)$, получен из образующего полинома $g(x) = (x-a)(x-a^2)\ldots(x-a^{2k})$, где *a* обозначает корень примитивного полинома $m(x) = x^8+x^4+x^3+x^2+1$. Двоичное представление a^i показано в таблице 59, где *MSB* октет следует первым.

Таблица 59 – Двоичное представление для a^i (0 <= i <= 254) на $GF(2^8)$

a^i	Двоичное представление	a^i	Двоичное представление	a^i	Двоичное представление	a^i	Двоичное представление
0	00000000	a^{63}	10100001	a^{127}	11001100	a^{191}	01000001
a^0	00000001	a^{64}	01011111	a^{128}	10000101	a^{192}	10000010
a^1	00000010	a^{65}	10111110	a^{129}	00010111	a^{193}	00011001
a^2	00000100	a^{66}	01100001	a^{130}	00101110	a^{194}	00110010
a^3	00001000	a^{67}	11000010	a^{131}	01011100	a^{195}	01100100
a^4	00010000	a^{68}	10011001	a^{132}	10111000	a^{196}	11001000
a^5	00100000	a^{69}	00101111	a^{133}	01101101	a^{197}	10001101
a^6	01000000	a^{70}	01011110	a^{134}	11011010	a^{198}	00000111
a^7	10000000	a^{71}	10111100	a^{135}	10101001	a^{199}	00001110
a^8	00011101	a^{72}	01100101	a^{136}	01001111	a^{200}	00011100
a^9	00111010	a^{73}	11001010	a^{137}	10011110	a^{201}	00111000
a^{10}	01110100	a^{74}	10001001	a^{138}	00100001	a^{202}	01110000
a^{11}	11101000	a^{75}	00001111	a^{139}	01000010	a^{203}	11100000
a^{12}	11001101	a^{76}	00011110	a^{140}	10000100	a^{204}	11011101
a^{13}	10000111	a^{77}	00111100	a^{141}	00010101	a^{205}	10100111
a^{14}	00010011	a^{78}	01111000	a^{142}	00101010	a^{206}	01010011
a^{15}	00100110	a^{79}	11110000	a^{143}	01010100	a^{207}	10100110
a^{16}	01001100	a^{80}	11111101	a^{144}	10101000	a^{208}	01010001
a^{17}	10011000	a^{81}	11100111	a^{145}	01001101	a^{209}	10100010
a^{18}	00101101	a^{82}	11010011	a^{146}	10011010	a^{210}	01011001
a^{19}	01011010	a^{83}	10111011	a^{147}	00101001	a^{211}	10110010
a^{20}	10110100	a^{84}	01101011	a^{148}	01010010	a^{212}	01111001

Окончание таблицы 59

a^i	Двоичное представление	a^i	Двоичное представление	a^i	Двоичное представление	a^i	Двоичное представление
a^{21}	01110101	a^{85}	11010110	a^{149}	10100100	a^{213}	11110010
a^{22}	11101010	a^{86}	10110001	a^{150}	01010101	a^{214}	11111001
a^{23}	11001001	a^{87}	01111111	a^{151}	10101010	a^{215}	11101111
a^{24}	10001111	a^{88}	11111110	a^{152}	01001001	a^{216}	11000011
a^{25}	00000011	a^{89}	11100001	a^{153}	10010010	a^{217}	10011011
a^{26}	00000110	a^{90}	11011111	a^{154}	00111001	a^{218}	00101011
a^{27}	00001100	a^{91}	10100011	a^{155}	01110010	a^{219}	01010110
a^{28}	00011000	a^{92}	01011011	a^{156}	11100100	a^{220}	10101100
a^{29}	00110000	a^{93}	10110110	a^{157}	11010101	a^{221}	01000101
a^{30}	01100000	a^{94}	01110001	a^{158}	10110111	a^{222}	10001010
a^{31}	11000000	a^{95}	11100010	a^{159}	01110011	a^{223}	00001001
a^{32}	10011101	a^{96}	11011001	a^{160}	11100110	a^{224}	00010010
a^{33}	00100111	a^{97}	10101111	a^{161}	11010001	a^{225}	00100100
a^{34}	01001110	a^{98}	01000011	a^{162}	10111111	a^{226}	01001000
a^{35}	10011100	a^{99}	10000110	a^{163}	01100011	a^{227}	10010000
a^{36}	00100101	a^{100}	00010001	a^{164}	11000110	a^{228}	00111101
a^{37}	01001010	a^{101}	00100010	a^{165}	10010001	a^{229}	01111010
a^{38}	10010100	a^{102}	01000100	a^{166}	00111111	a^{230}	11110100
a^{39}	00110101	a^{103}	10001000	a^{167}	01111110	a^{231}	11110101
a^{40}	01101010	a^{104}	00001101	a^{168}	11111100	a^{232}	11110111
a^{41}	11010100	a^{105}	00011010	a^{169}	11100101	a^{233}	11110011
a^{42}	10110101	a^{106}	00110100	a^{170}	11010111	a^{234}	11111011
a^{43}	01110111	a^{107}	01101000	a^{171}	10110011	a^{235}	11101011
a^{44}	11101110	a^{108}	11010000	a^{172}	01111011	a^{236}	11001011
a^{45}	11000001	a^{109}	10111101	a^{173}	11110110	a^{237}	10001011
a^{46}	10011111	a^{110}	01100111	a^{174}	11110001	a^{238}	00001011
a^{47}	00100011	a^{111}	11001110	a^{175}	11111111	a^{239}	00010110
a^{48}	01000110	a^{112}	10000001	a^{176}	11100011	a^{240}	00101100
a^{49}	10001100	a^{113}	00011111	a^{177}	11011011	a^{241}	01011000
a^{50}	00000101	a^{114}	00111110	a^{178}	10101011	a^{242}	10110000
a^{51}	00001010	a^{115}	01111100	a^{179}	01001011	a^{243}	01111101
a^{52}	00010100	a^{116}	11111000	a^{180}	10010110	a^{244}	11111010
a^{53}	00101000	a^{117}	11101101	a^{181}	00110001	a^{245}	11101001
a^{54}	01010000	a^{118}	11000111	a^{182}	01100010	a^{246}	11001111
a^{55}	10100000	a^{119}	10010011	a^{183}	11000100	a^{247}	10000011
a^{56}	01011101	a^{120}	00111011	a^{184}	10010101	a^{248}	00011011
a^{57}	10111010	a^{121}	01110110	a^{185}	00110111	a^{249}	00110110
a^{58}	01101001	a^{122}	11101100	a^{186}	01101110	a^{250}	01101100
a^{59}	11010010	a^{123}	11000101	a^{187}	11011100	a^{251}	11011000
a^{60}	10111001	a^{124}	10010111	a^{188}	10100101	a^{252}	10101101
a^{61}	01101111	a^{125}	00110011	a^{189}	01010111	a^{253}	01000111
a^{62}	11011110	a^{126}	01100110	a^{190}	10101110	a^{254}	10001110

Для каждой из частей паритет *SRS* с полной длиной октетов 2*k* вычисляется, используя *g*(*x*) следующим образом:

$p(x) = x^{2k}\, u(x) \bmod g(x)$

u (*x*): полиномиальный представитель части. Низший порядок соответствует первому октету;

p (*x*): полиномиальный представитель четности. Низший порядок соответствует первому октету.

Для хранения и передачи паритет добавляется в конце класса *EP*. Этот процесс показан на рисунке 7.

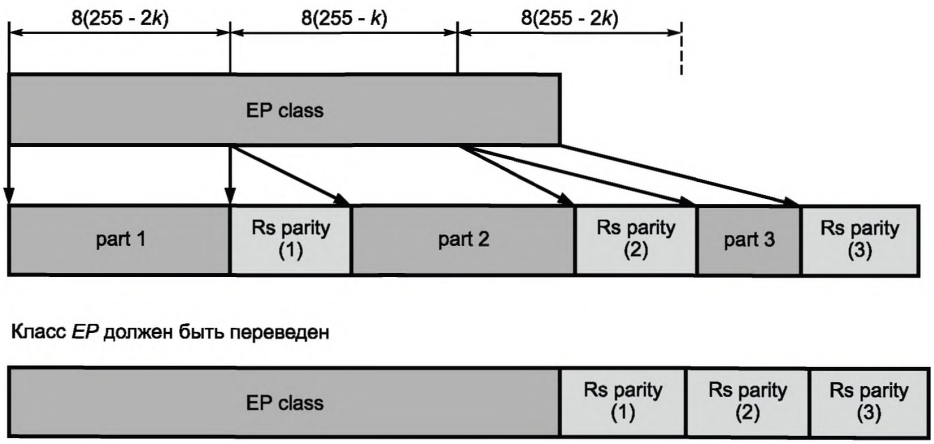


Рисунок 7 – Кодирование *RS* фрейма *EP*

3.8.4.8 Рекурсивное чередование

Чередование применяется многоступенчатым способом. Рисунок 8 показывает метод чередования.

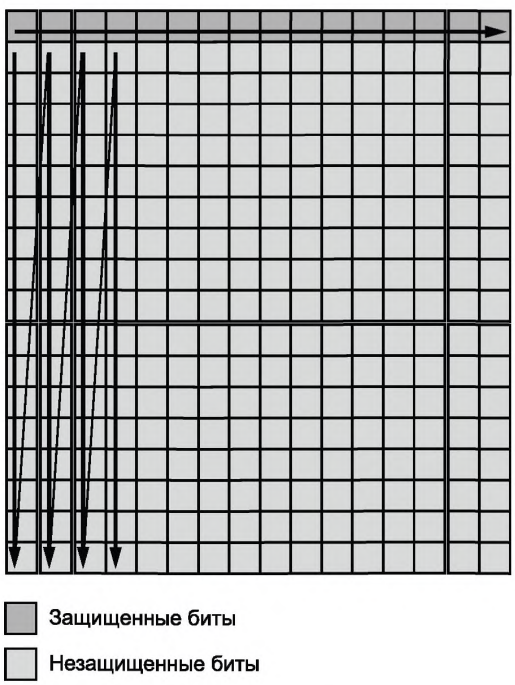


Рисунок 8 – Одна стадия чередования

В многоступенчатом чередовании выход этой стадии чередования обрабатывается как незащищенная часть в следующей стадии. Рисунок 9 показывает пример 2 стадии чередований.

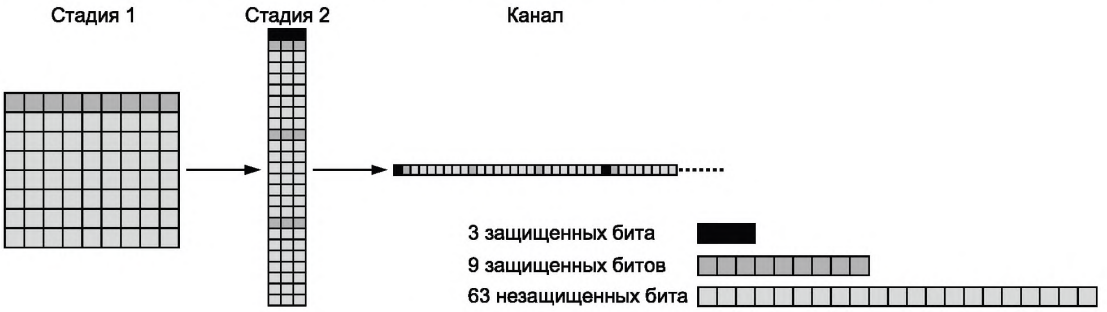


Рисунок 9 – Пример многоступенчатого чередования

Путем выбора ширины *W* матрицы чередования для получения одинаковой длины с длиной кода *FEC* (или значения 28 в случае кодов *SRCPC*) размер чередования может быть оптимизирован для всех *FEC* кодов.

В фактическом случае общее количество битов для чередования может не позволить использовать такой прямоугольник. В таком случае используется матрица, как показано на рисунке 10.

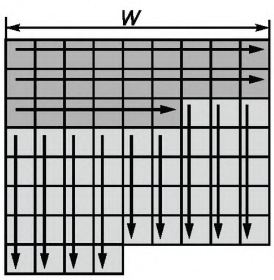


Рисунок 10 – Матрица чередования в непрямоугольном случае

3.8.4.8.1 Определение рекурсивного чередования

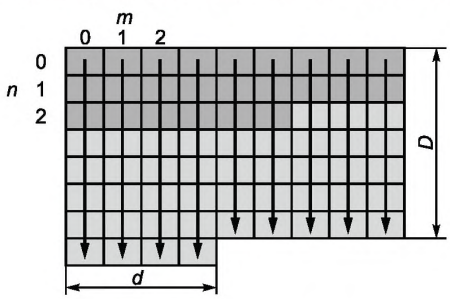
Два информационных потока X_i и Y_j являются входами:

X_i, $0 <= i < I_x$;

Y_j, $0 <= j < I_y$,

где I_x и I_y - число битов для каждого из входных потоков X_i и Y_j соответственно. X_i настроен на матрицу чередования сверху вниз и слева направо в горизонтальном направлении. Y_j настроен на остальные места в вертикальном направлении.

С шириной чередования *W* размер матрицы чередования показан на рисунке 11.



$D = (I_x + I_y) / W$, $d = I_x + I_y - D * W$, '/' указывает деление с округлением.

Рисунок 11 – Размер матрицы чередования

Полезный выходной битовый поток Z_k ($0 < k <= I_x + I_y$) считывается с этой матрицы сверху вниз и слева направо в горизонтальном направлении. Таким образом бит, расположенный в *m*-ом столбце и *n*-ой строке (*m* и *n* начинаются с 0) соответствует Z_k , где:

$k = m * D + \min(m, d) + n$

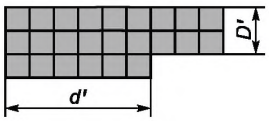
В матрице X_i установлен в

$m = i \bmod W,\ n = i / W,$

Таким образом Z_k, который установлен X_i, становится:

$Z_k = X_i$, где $k = (i \bmod W) * D + \min (i \bmod W, d) + i/W$

Биты, которые установлены с X_i в матрице чередования, показаны на рисунке 12:



$D' = I_x / W$
$d' = I_x - D' * W$

Рисунок 12 – Биты, установленные с X_i в матрице чередования

Таким образом в *m*-ой строке Y_j установлен из *n*-ой строки, где $n = D' + (m < d' ? 1 : 0)$ к основанию. Таким образом Z_k, установленный Y_j, представлен следующим образом:

```
Set j to 0;
for m = 0 to D-1 {
    for k = m * D+min(m, d)+D'+(m < d' ? 1: 0) to (m+1) * D+min (m+1, d) - 1 {
            Zk = Yj;
            j ++;
    }
}
```

3.8.4.8.2 Режимы чередования

Два режима чередования, режим 1 и режим 2, в соответствии с *interleave_type* 1 и 2, определены в следующих выражениях. Таблица 60 и таблица 61 дают краткий обзор доступных конфигураций.

Таблица 60 – Ширина матрицы чередования

interleave_type	*fec_type* == 0 (*SRCPC*)	*fec_type* ==1/2 (*SRS*)
0	Чередование отсутствует	
1	28 бит	Длина класса
2	Зависит от переключения чередования (см. таблицу 61)	
3	Зарезервировано	

Таблица 61 – Ширина матрицы чередования для *interleave_type* 2

interleave_switch	*fec_type* == 0 (*SRCPC*)	*fec_type* ==1/2 (*SRS*)
0	Чередование отсутствует	
1	Длина класса	Длина класса
2	28 бит	Не разрешено
3	Конкатенация	

В случае *fec_type*=0 (*SRCPC*) чередование выполняется побитно. В случае *fec_type* == 1 или *fec_type* == 2 (*SRS*) чередование выполняется побайтно.

3.8.4.8.2.1 Чередование в режиме 1

Многоступенчатое чередование используется для *ep_encoded_class* от последнего класса до первого класса, когда биты наполнения добавлены после чередованных классов. Процесс чередования продолжает атрибутивную часть класса *ep_header* () (который является *class_attrib* () + *class_attrib_parity*) и предопределенную часть *ep_header* () (который является *choice_of_pred* + *choice_of_pred_parity*), как показано на рисунке 13.

Ширина матрицы чередования выбирается согласно используемому *FEC*. В случае *SRCPC* кодирования (*fec_type* == 0) ширина матрицы чередования составляет 28 битов. В случае *SRS* кодирования (*fec_type* == 1 или 2) ширина матрицы чередования равна длине класса в байтах. Биты в классе записываются в матрицу чередований байтов для каждого столбца.

Ширина матрицы чередования для частей заголовка либо равна длине ключевого слова (в битах), предоставленной блочным кодом согласно таблице 61, или 28 битам, если используется *SRCPC*.

3.8.4.8.2.2 Чередование в режиме 2

В режиме 2 флаг указывает, обработан ли класс с чередованием, и как именно. Об этом флаге *interleave_switch* сообщается в пределах полосы. Значение 0 указывает, что класс не обработан с чередованием. Значение 1 указывает, что класс чередован рекурсивно и длина класса используется как ширина чередования (или длина в битах в случае *SRCPC*, или длина в байтах в случае *SRS*). Значение 2 указывает, что класс чередован рекурсивно и ширина должна быть равной 28 (разрешено только в случае *SRCPC*). Значение 3 указывает, что класс связан, но не чередован рекурсивно. Операция чередования для *ep_header* аналогична режиму 1.

Рисунок 13 показывает схему чередования для *fec_type* == 1 или 2 (*SRS*) и *interleave_switch* == 1. Ширина должна быть числом байтов в классе. Биты в классе записываются в матрицу чередований байтов для каждого столбца.

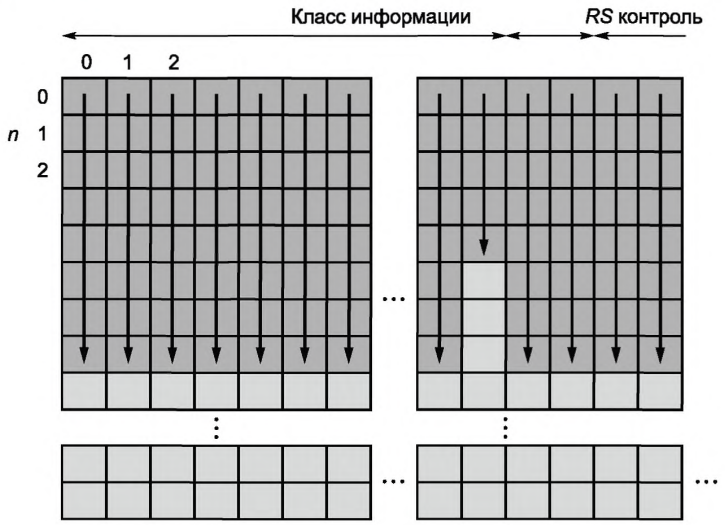


Рисунок 13 – Матрица чередования в случае *RS* класса

Процесс чередования для получения *interleaved_frame_mode*2 описывается следующим образом (*N*: число классов):

```
clear buffer BUF_NO /* Buffer for non-interleaved part. */
clear buffer BUF_Y /* Buffer for Y input in the next stage */
for( j = 0; j < N; j++ )
    if ( class_reordered_output == 1 ) {
        k = class_output_order[choice_of_pred][j];
    } else {
        k = j;
    }
    if ( interleave_switch[choice_of_pred][k] == 3) {
    add ep_encoded_class[k] to BUF_Y;
    }
    if ( interleave_switch[choice_of_pred][k] == 0 ) {
    add ep_encoded_class[k] to BUF_NO;
    }
}
```

```
for ( j = N-1; j >= 0; j-- ) {
    if ( class_reordered_output == 1 ) {
        k = class_output_order[choice_of_pred][j];
    } else {
        k = j;
    }
    if ((interleave_switch[choice_of_pred][k] != 0 )
            && ( interleave_switch[choice_of_pred][k] != 3 ) ){
        if ( interleave_switch[choice_of_pred][k] == 1 ) {
            set the size of the interleave window to be the length of ep encoded class[k];
        } else if ( interleave_switch[choice_of_pred][k] == 2 ) {
            set the size of the interleave window to be 28;
        }
        input ep_encoded_class[k] into the recursive interleaver as X input;
        input BUF_Y into the recursive interleaver as Y input;
        set the output of the interleaver into BUF_Y;
    }
}
add BUF_NO to BUF_Y;
if( bit_stuffing ) {
    add Nstuff stuffing-bits to BUF_Y;
}
input class_attrib() followed by class_attrib_parity into the recursive interleaver as X input;
input BUF_Y into the recursive interleaver as Y input;
set the output of the interleaver into BUF_Y;
input choice_of_pred followed by choice_of_pred_parity into the recursive interleaver as X input;
input BUF_Y into the recursive interleaver as Y input; set the output of the interleaver into BUF_Y; set BUF_Y
into interleaved_frame_mode2;
```

3.8.4.9 Упорядочивание классов

Инструмент *EP* позволяет переупорядочивать классы так, что нет необходимости привязываться к порядку предоставленному / требуемому звуковым кодеком. Порядок классов после переупорядочения сигнализируется как *class_output_order* [*i*] [*j*] в полосе. Декодер *EP* переупорядочивает классы во фрейме *EP*, используя *i*-ый предопределенный набор, так чтобы *j*-ый класс фрейма *EP* был направлен как (*class_output_order* [*i*] [*j*])-ый класс к звуковому декодеру.

Приложение А
(справочное)

Форматы обмена аудиофайлами

А.1 Введение

Полные возможности и гибкость *MPEG*–4 Аудио, такие как композиция звуковых сцен из множественных звуковых объектов, синтезированный звук и преобразование текста в речь, доступны, только если *MPEG*–4 Аудио используется вместе с *MPEG*–4 Системы. Форматы обмена, определенные здесь в приложении А, поддерживают лишь небольшое подмножество возможностей *MPEG*–4 Аудио, определяя форматы для хранения и передачи отдельных моно, стерео или многоканальных звуковых объектов, схожих с форматами, определенным в *MPEG*–1 и *MPEG*–2.

Нормативные элементы в *MPEG*–4 Аудио заканчиваются определением полезных нагрузок (примерные эквиваленты фреймам потока битов в *MPEG*–1 и *MPEG*–2) и структурами конфигурации кодера (напоминающими *MPEG*–1/2 информацию заголовка). Однако нет никакого нормативного определения *MPEG*–4 Аудио относительно того, как эти элементы мультиплексируются, поскольку это требуется только для ограниченного числа приложений. Однако это информативное приложение описывает такое мультиплексирование. Тем не менее декодеры *MPEG*–4 не обязательно должны иметь эти форматы интерфейса.

А.2 Форматы обмена AAC

А.2.1 Синтаксис

А.2.1.1 *AAC MPEG–2 Audio_Data_Interchange_Format*, *ADIF* (таблицы А.1, А.2, А.3, А.4, А.5).

Таблица А.1 – Синтаксис *adif_sequence*

Синтаксис	Количество битов	Мнемоника
adif_sequence() *{* *adif_header();* *byte_alignment();* *raw_data_stream();* *}*		

Таблица А.2 – Синтаксис *adif_header* ()

Синтаксис	Количество битов	Мнемоника
adif_header() *{*		
adif_id;	32	*bslbf*
copyright_id_present;	1	*bslbf*
if (copyright_id_present)		
copyright_id;	72	*bslbf*
original_copy;	1	*bslbf*
home;	1	*bslbf*
bitstream_type;	1	*bslbf*
bitrate;	23	*uimsbf*
num_program_config_elements;	4	*bslbf*
if (bitstream_type == '0') {		
adif_buffer_fullness;	20	*uimsbf*
for (i = 0; i < num_program_config_elements + 1; i++) {		
program_config_element();		
}		
}		

Таблица А.3 – Синтаксис *raw_data_stream* ()

Синтаксис	Количество битов	Мнемоника
raw_data_stream() *{* *while (data_available()) {* *raw_data_block();* *}* *}*		

Т а б л и ц а А.4 – Синтаксис *adts_sequence* ()

Синтаксис	Количество битов	Мнемоника
adts_sequence() *{* *while (nextbits() == syncword) {* *adts_frame();* *}* *}*		

Т а б л и ц а А.5 – Синтаксис *adts_frame* ()

Синтаксис	Количество битов	Мнемоника
adts_frame() *{* *adts_f ixed_header();* *adts_variable_header();* *if (number_of_raw_data_blocks_in_frame == 0) {* *adts_error_check();* *raw_data_block();* *}* *else {* *adts_header_error_check();* *for(i=0; i<=number_of_raw_data_blocks_in_frame; i++){* *raw_data_block();* *adts_raw_data_block_error_check();* *}* *}* *}*		

А.2.1.2 Фиксированный заголовок *ADTS* (таблица А.6).

Т а б л и ц а А.6 – Синтаксис *adts_fixed_header* ()

Синтаксис	Количество битов	Мнемоника
adts_fixed_header()		
{		
syncword;	12	*bslbf*
ID;	1	*bslbf*
layer;	2	*uimsbf*
protection_absent;	1	*bslbf*
profile_ObjectType;	2	*uimsbf*
sampling_frequency_index;	4	*uimsbf*
private_bit;	1	*bslbf*
channelconfiguration;	3	*uimsbf*
original_copy;	1	*bslbf*
home;	1	*bslbf*
}		

А.2.1.2.1 Переменный заголовок *ADTS* (таблица А.7).

Т а б л и ц а А.7 – Синтаксис *adts_variable_header* ()

Синтаксис	Количество битов	Мнемоника
adts variable header()		
{		
copyright_identification_bit;	1	*bslbf*
copyright_identification_start;	1	*bslbf*
aac_frame_length;	13	*bslbf*
adts_buffer_fullness;	11	*bslbf*
number_of_raw_data_blocks_in_frame;	2	*uimsbf*
}		

А.2.1.2.2 Обнаружение ошибок (таблицы А.8, А.9, А.10).

Таблица А.8 – Синтаксис *adts_error_check*

Синтаксис	Количество битов	Мнемоника
adts error check() *{* *if (protection_absent == '0')* *crc_check;* *}*	16	*Rpchof*

Таблица А.9 – Синтаксис *adts_header_error_check*

Синтаксис	Количество битов	Мнемоника
adts header error check () *{* *if (protection_absent == '0') {* *for (i=1; i<=number_of_raw_data_blocks_in_frame; i++){* *raw_data_block_position(i);* *}* *crc_check;* *}* *}*	16 16	*uimsbf* *rpchof*

Таблица А.10 – Синтаксис *adts_raw_data_block_error_check* ()

Синтаксис	Количество битов	Мнемоника
adts_raw_data_block_error_check(i) *{* *if (protection_absent == '0')* *crc_check;* *}*	16	*rpchof*

А.3 Семантика

А.3.1 Краткий обзор

raw_data_block () содержит все данные, которые относятся к звуку (включая вспомогательные данные). Кроме того, дополнительная информация, такая как *sampling_frequency,* необходима для полного описания звуковой последовательности. Формат обмена звуковыми данными (*ADIF*) содержит все элементы, которые необходимы для описания потока битов согласно этому стандарту.

Для определенных приложений некоторые или все элементы синтаксиса, определенные в заголовке *ADIF*, например *sampling_rate*, могут быть известны декодеру за счет других средств и, следовательно, не появляться в потоке битов.

Кроме того, может потребоваться дополнительная информация, которая изменяется от блока к блоку (например, чтобы увеличить читаемость данных или устойчивость к ошибкам). Следовательно, транспортные потоки могут быть разработаны для определенного приложения и не определены в этом стандарте. Однако один ненормативный транспортный поток, названный Транспортным Потоком Звуковых Данных (*ADTS*), описан. Он может использоваться для приложений, в которых декодер может анализировать этот поток.

А.3.2 Формат обмена звуковыми данными (*ADIF*)

raw_data_stream () – последовательность *raw_data_block* () блоков.

program_config_element () содержит информацию о конфигурации для одной программы.

А.3.3 Транспортный поток звуковых данных (*ADTS*)

ID – идентификатор *MPEG* установлен в '1', если звуковые данные в потоке *ADTS - AAC MPEG*–2 и в '0', если звуковые данные – *MPEG*–4.

profile_ObjectType – интерпретация этого элемента данных зависит от значения бита идентификатора. Если идентификатор равен '1', эта область содержит ту же самую информацию, что и область конфигурации в потоке *ADTS*. Если идентификатор равен '0', этот элемент обозначает тип объекта *MPEG*–4 Аудио (*profile_ObjectType*+1).

sampling_frequency_index указывает частоту дискретизации, используемую согласно таблице 17. Значение *escape* не разрешено.

channel_configuration указывает используемую конфигурацию каналов. В случае (*channel_configuration*> 0) конфигурация каналов дается в таблице 18. В случае (*channel_configuration* == 0) конфигурация каналов не определена в заголовке и задается следующим образом:

MPEG–2/4 *ADTS* – отдельный *program_config_element* (), являющийся первым синтаксическим элементом в первом *raw_data_block* () после заголовка; определяет конфигурацию канала. Элемент *program_config_element* () может не присутствовать во всех фреймах. Декодер *MPEG–4 ADTS* не должен генерировать выходные данные, пока он не получит *program_config_element* (), в то время как декодер *MPEG–2 ADTS* может принять неявную конфигурацию канала.

MPEG–2 ADTS: Помимо использования *program_config_element* (), конфигурация каналов может предполагаться неявной или может быть известна в приложении.

Приложение Б
(справочное)

Инструмент защиты от ошибок

Ниже представлены формат текстового файла внутриполосной информации и его пример для *AAC*, *Twin-VQ*, *CELP* и *HVXC*. Кроме того представлен пример маскировки ошибок.

Б.1 Пример внутриполосной информации

Б.1.1 Пример для *AAC*

Этот пример основан на категориях чувствительности к ошибкам, описанным в нормативной части. Нижеследующая защита от ошибок может использоваться при соответствии категорий чувствительности классам. Этот пример показывает использование простого режима с одним каналом и без *extension_payload* ():

Класс	Длина	Чередование	*SRCPC puncture rate*	Длина *CRC*
0	6 бит	Внутрикадровое	8/24	6
1	12 бит	Внутрикадровое	8/24	6
2	9 бит	Внутрикадровое	8/8	6
3	9 бит	-	8/8	4
4	До конца	Внутрикадровое	8/8	-

Б.1.2 Пример для *Twin-VQ*

Ниже описываются примеры назначения битов *UEP* профилю масштабируемого звука (объект TwinVQ).

Имеется два режима кодирования: с и без *PPC* (Периодический Пиковый Компонент). Обычно кодер может адаптивно выбирать *PPC*, но в данном случае необходимо всегда поддерживать режим «вкл/выкл». Если *PPC* включен, то 43 бита выделятся для квантования периодических пиковых компонент, и эти биты должны быть защищены как служебная информация.

Для каждого режима показано распределение четырех различных скоростей передачи, 16 кбит/с моно, 32 кбит/с стерео, 8 кбит/с + 8 кбит/с масштабируемого моно и 16 кбит/с + 16 кбит/с стерео для каждого режима.

Во всех случаях коррекция ошибок и инструменты обнаружения применяются только к 10 % битов для служебной информации. У оставшихся битов для индексов коэффициентов МДКП нет никакой защиты вообще. В результате такого распределения битов скорость передачи увеличивается по сравнению с оригинальной исходной скоростью приблизительно на 10 % в случае включения *PPC* и менее чем 10 % в случае выключения *PPC*.

а) *PPC* (Периодический Пиковый Компонент) включен

16 кбит/с моно:

Класс 1: 121 бит (фиксирован), код *SRCPC* 8/12, 8 бит *CRC*

Класс 2: 839 битов (фиксирован), код *SRCPC* 8/8, *CRC* нет

32 кбит/с стерео:

Класс 1: 238 битов (фиксирован), код *SRCPC* 8/12, 10 бит *CRC*

Класс 2: 1682 бита (фиксирован), код *SRCPC* 8/8, *CRC* нет

8 кбит/с + 8 кбит/с масштабируемое моно:

Класс 1: 121 бит (фиксирован), код *SRCPC* 8/12, 8 бит *CRC*

Класс 2: 359 битов (фиксирован), код *SRCPC* 8/8, *CRC* нет

Класс 3: 72 бита (фиксирован), код *SRCPC* 8/12, 8 бит *CRC*

Класс 4: 408 битов (фиксирован), код *SRCPC* 8/8, *CRC* нет

16 кбит/с + 16 кбит/с масштабируемое стерео

Класс 1: 238 битов (фиксирован), код *SRCPC* 8/12, 10 бит *CRC*

Класс 2: 722 бита (фиксирован), код *SRCPC* 8/8, *CRC* нет

Класс 3: 146 битов (фиксирован), код *SRCPC* 8/12, 10 бит *CRC*

Класс 4: 814 битов (фиксирован), код *SRCPC* 8/8, *CRC* нет

б) *PPC* отключен

16 кбит/с моно:

Класс 1: 78 битов (фиксирован), код *SRCPC* 8/12, 8 бит *CRC*

Класс 2: 882 бита (фиксирован), код *SRCPC* 8/8, *CRC* нет

32 кбит/с стерео:
Класс 1: 152 бита (фиксирован), код *SRCPC* 8/12, 10 бит *CRC*
Класс 2: 1768 битов (фиксирован), код *SRCPC* 8/8, *CRC* нет
8 кбит/с + 8 кбит/с масштабируемое моно:
Класс 1: 78 битов (фиксирован), код *SRCPC* 8/12, 8 бит *CRC*
Класс 2: 402 бита (фиксирован), код *SRCPC* 8/8, *CRC* нет
Класс 3: 72 бита (фиксирован), код *SRCPC* 8/12, 8 бит *CRC*
Класс 4: 408 битов (фиксирован), код *SRCPC* 8/8, *CRC* нет
16 кбит/с + 16 кбит/с масштабируемое стерео
Класс 1: 152 бита (фиксирован), код *SRCPC* 8/12, 10 бит *CRC*
Класс 2: 808 битов (фиксирован), код *SRCPC* 8/8, *CRC* нет
Класс 3: 146 битов (фиксирован), код *SRCPC* 8/12, 10 бит *CRC*
Класс 4: 814 битов (фиксирован), код *SRCPC* 8/8, *CRC* нет

Б.1.3 Пример для *CELP*

Следующие таблицы обеспечивают краткий обзор количества битов, выделяемых на каждую категорию чувствительности к ошибкам, зависящую от конфигурации.

Б.1.3.1 Режим узкополосный *MPE*

Краткий обзор выделения битов для узкополосного *MPE:*

Режим *MPE*	Подфреймы	Бит/Фрейм	Битовая скорость	*ECR*0	*ECR*1	*ECR*2	*ECR*3	*ECR*4
0	4	154	3850	6	13	20	37	78
1	4	170	4250	6	13	20	41	90
2	4	186	4650	6	13	20	45	102
3	3	147	4900	5	11	16	36	79
4	3	156	5200	5	11	16	39	85
5	3	165	5500	5	11	16	42	91
6	2	114	5700	4	9	12	29	60
7	2	120	6000	4	9	12	31	64
8	2	126	6300	4	9	12	33	68
9	2	132	6600	4	9	12	35	72
10	2	138	6900	4	9	12	37	76
11	2	142	7100	4	9	12	39	78
12	2	146	7300	4	9	12	41	80
13	4	154	7700	6	13	20	41	74
14	4	166	8300	6	13	20	45	82
15	4	174	8700	6	13	20	49	86
16	4	182	9100	6	13	20	53	90
17	4	190	9500	6	13	20	57	94
18	4	198	9900	6	13	20	61	98
19	4	206	10300	6	13	20	65	102
20	4	210	10500	6	13	20	69	102
21	4	214	10700	6	13	20	73	102
22	2	110	11000	4	9	12	33	52
23	2	114	11400	4	9	12	35	54
24	2	118	11800	4	9	12	37	56
25	2	120	12000	4	9	12	39	56
26	2	122	12200	4	9	12	41	56
27	4	186	6200	6	13	20	49	98
28	Зарезервировано							
29	Зарезервировано							
30	Зарезервировано							
31	Зарезервировано							

Б.1.3.2 Режим широкополосный *MPE*

Краткий обзор выделения битов для широкополосного *MPE*

Режим *MPE*	Подфреймы	Бит/Фрейм	Битовая скорость	*ECR*0	*ECR*1	*ECR*2	*ECR*3	*ECR*4
0	4	218	10900	17	20	27	45	109
1	4	230	11500	17	20	27	49	117
2	4	242	12100	17	20	27	53	125
3	4	254	12700	17	20	27	57	133
4	4	266	13300	17	20	27	61	141
5	4	278	13900	17	20	27	65	149
6	4	286	14300	17	20	27	69	153
7	Зарезервировано							
8	8	294	14700	17	32	43	61	141
9	8	318	15900	17	32	43	69	157
10	8	342	17100	17	32	43	77	173
11	8	358	17900	17	32	43	85	181
12	8	374	18700	17	32	43	93	189
13	8	390	19500	17	32	43	101	197
14	8	406	20300	17	32	43	109	205
15	8	422	21100	17	32	43	117	213
16	2	136	13600	17	14	19	29	57
17	2	142	14200	17	14	19	31	61
18	2	148	14800	17	14	19	33	65
19	2	154	15400	17	14	19	35	69
20	2	160	16000	17	14	19	37	73
21	2	166	16600	17	14	19	39	77
22	2	170	17000	17	14	19	41	79
23	Зарезервировано							
24	4	174	17400	17	20	27	37	73
25	4	186	18600	17	20	27	41	81
26	4	198	19800	17	20	27	45	89
27	4	206	20600	17	20	27	49	93
28	4	214	21400	17	20	27	53	97
29	4	222	22200	17	20	27	57	101
30	4	230	23000	17	20	27	61	105
31	4	238	23800	17	20	27	65	109

Б.1.3.3 Режим широкополосный *RPE*

Характеристические параметры для широкополосного *CELP* с *RPE*

Режим *MPE*	Подфреймы	Бит/Фрейм	Битовая скорость	*ECR*0	*ECR*1	*ECR*2	*ECR*3	*ECR*4
0	6	216	14400	40	24	34	25	93
1	4	160	16000	32	18	26	21	63
2	8	280	18667	48	30	42	29	131
3	10	338	22533	56	36	50	33	163

Б.1.4 Пример для *HVXC*

кодер 2 кбит/с:

Класс 1: 22 бита (фиксирован), код *SRCPC* 8/16, 6 битов *CRC*

Класс 2: 4 бита (фиксирован), код *SRCPC* 8/8, 1 бит *CRC*

Класс 3: 4 бита (фиксирован), код *SRCPC* 8/8, 1 бит *CRC*

Класс 4: 10 битов (фиксирован), код *SRCPC* 8/8, *CRC* нет

кодер 4 кбит/с:

Класс 1: 33 бита (фиксирован), код *SRCPC* 8/16, 6 битов *CRC*

Класс 2: 22 бита (фиксирован), код *SRCPC* 8/8, 6 битов *CRC*

Класс 3: 4 бита (фиксирован), код *SRCPC* 8/8, 1 бит *CRC*

Класс 4: 4 бита (фиксирован), код *SRCPC* 8/8, 1 бит *CRC*

Класс 4: 17 битов (фиксирован), код *SRCPC* 8/8, *CRC* нет

Б.1.5 Пример для *ER BSAC*

Этот подраздел описывает примеры распределения битов неравной защиты от ошибок (*UEP*) типу объекта *ER BSAC*.

Категория низкой чувствительности к ошибкам (*ESC*) указывает класс с более высокой чувствительностью к ошибкам, тогда как более высокий *ESC* указывает класс с меньшей чувствительностью. Следующий пример основан на категориях чувствительности к ошибкам *BSAC*. Этот пример соответствует простой настройке, где категории чувствительности соответствуют классам.

Класс	Категория	Длина, бит	Чередование	*SRCPC puncture rate*	Длина *CRC*
0	0	9	Внутрикадровое	8/24	6
1	Другое	11	-	8/8	-

В этом примере инструменты коррекции и обнаружения ошибок применяются только к общей служебной информации. У оставшихся битов для индекса коэффициентов МДКП нет никакой защиты. В результате такого распределения битов скорость передачи увеличивается приблизительно на 10% по сравнению с оригинальной исходной скоростью.

Б.2 Пример маскировки ошибок

Инструмент маскировки ошибок - дополнительный инструмент декодера для уменьшения ухудшения качества декодированных сигналов, когда полезный битовый поток ввода декодера затронут ошибками, такими как ошибка передачи полезного битового потока. Это особенно эффективно в случае использования инструментов *MPEG*–4 Аудио в радио приложениях. Обнаружение ошибок и решающий метод для замены фрейма не определены в этом подразделе и зависят от конкретного приложения.

Б.2.1 Пример для *CELP*

Б.2.1.1 Краткий обзор инструмента маскировки ошибок

Инструмент маскировки ошибок используется с декодером *MPEG*–4 *CELP*. Этот инструмент уменьшает неприятный шум, возникающий в результате обработки декодером *MPEG*–4 *CELP* ошибочных данных фрейма. Также *CELP MPEG*–4 используется для декодирования речи даже в том случае, если входные данные фрейма потеряны.

У инструмента есть два операционных режима: режим битовых ошибок (*BE*) и режим стирания фрейма (*FE*). Режимы переключаются на основании пригодности данных фрейма в декодере. Когда данные фрейма доступны (режим *BE*), декодирование выполняется с использованием данных прошлых фреймов и годных данных текущего фрейма. Когда данные фрейма не доступны (режим *FE*), декодер генерирует речь, используя только данные прошлых фреймов.

Этот инструмент работает в режиме кодирования II, который использует режимы узкой полосы, широкой полосы и масштабируемые режимы, которые поддерживают *MPE* на частотах дискретизации 8 и 16 кГц.

Б.2.1.2 Определения

BE: Ошибочный бит

BWS: Масштабируемая ширина полосы

FE: Стирание фрейма

LP: Линейное предсказание

LSP: Линейная спектральная пара

MPE: Мультиимпульсное возбуждение

NB: Узкая полоса

RMS: Среднеквадратичное значение (энергия фрейма)

WB: Широкая полоса

Б.2.1.3 Вспомогательные переменные

frame_size: количество отсчетов во фрейме

g_ac: адаптивная кодовая книга

g_ec: уровень *MPE*

lpc_order: порядок *LP*

signal_mode: речевой режим

signal_mode_pre: речевой режим предыдущего фрейма

Б.2.1.4 Спецификация инструмента маскировки ошибок

Инструмент маскировки ошибок основан на модели перехода с шестью состояниями, изображенными на рисунке Б.1. Состояние соответствует качеству канала передачи. Чем больше номер состояния, тем хуже качество канала передачи. В каждом состоянии используется своя маскировка. Начальное состояние при декодировании - 0, передача состояния происходит с помощью флага *BF_flag*. Каждая операция маскировки описана в следующих подразделах.

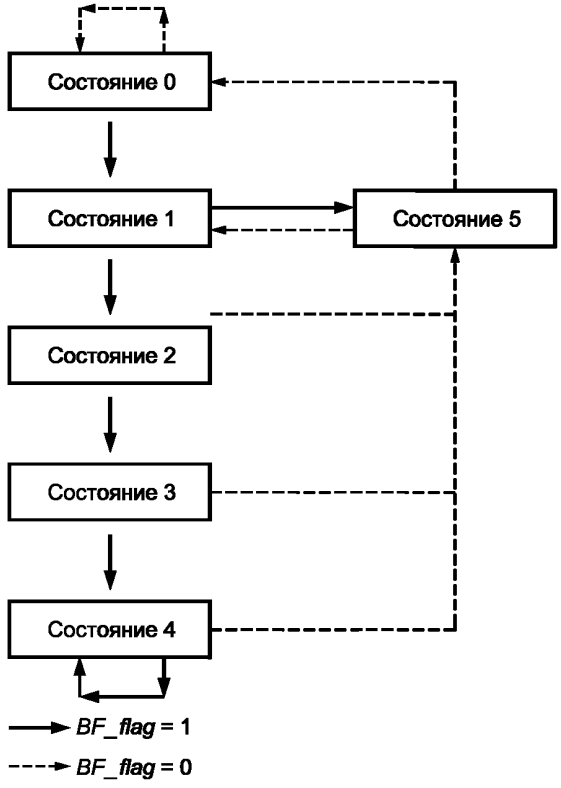


Рисунок Б.1 – Модель перехода между состояниями для управления маскировкой ошибок

Б.2.1.4.1 Операции в состояниях 0 и 5

Процесс декодирования идентичен тому, что используется в декодере *MPEG–4 CELP* за следующими исключениями относительно адаптивной кодовой книги:

В состоянии 0 после состояния 5, и в состоянии 5:

1) для первых 80 отсчетов во фрейме (и после), где *BF_flag* сменился от 1 к 0, уровни *g_ac* и *g_ec* вычисляются по уровням *g_ac'* и *g_ec'*, декодированных из текущих данных фрейма следующим образом:

```
if (g_ac> 1,0) {
   g_ec = g_ec '/g_ac';
   g_ac=1,0;
}
```

2) для 160 отсчетов, которые следуют за первыми 80 отсчетами, уровни вычисляются как:

```
if (g_ac> 1,0) {
   g_ec = g _ec' * (g_ac' +1,0)/g_ac '/2,0;
   g_ac = (g_ac' +1,0)/2,0;
},
```

где эти операции продолжаются самое большее четыре подфрейма.

Б.2.1.4.2 Операции в состояниях 1, 2, 3 и 4

Процесс декодирования идентичен тому, который применяется в декодере *MPEG–4 CELP* за исключениями, описанными в следующих подразделах.

Б.2.1.4.2.1 Режим Речь

Б.2.1.4.2.1.1 Режим *FE*

Речевой режим (*signal_mode*) декодируется по данным предыдущего фрейма.

Б.2.1.4.2.1.2 Режим *BE*

Речевой режим (*singal_mode*) декодируется по данным текущего фрейма для сигнала *_mode_pre*=0 или 1. Иначе используется режим декодирования по данным предыдущего фрейма.

Б.2.1.4.2.2 Режим *MPE*

Б.2.1.4.2.2.1 Режим *FE*

MPE декодируется по данным фрейма, сгенерированным случайным образом.

Б.2.1.4.2.2.2 Режим *BE*

MPE декодируется по данным текущего фрейма.

Б.2.1.4.2.3 *RMS*

Для состояний от 1 до *K*, *RMS* в последнем подфрейме предыдущего фрейма используется после аттенюации в первом подфрейме текущего фрейма. В последующих подфреймах *RMS* в предыдущем подфрейме используется после аттенюации. Уровень аттенюации P_{att} зависит от состояния следующим образом:

P_{att} = 0,4 дБ, для состояний 1 … *K*

P_{att} =1,2 дБ, для состояний *K*+1, …,

где *K* – наименьшее значение из 4 и K_0, а K_0 – наибольшее целое, удовлетворяющее *frame_size* x K_0<= 320.

Б.2.1.4.2.4 *LSP*

Используются *LSP*, декодированные в предыдущем фрейме. В режиме *BWS* векторы *LSP* должны быть буферизованы для межфреймового предсказания. Однако, когда данные фреймы повреждены или потеряны, правильный вектор не может быть получен. Поэтому буферизированный вектор (*blsp* [0] []) оценивается по *LSP* (*qlsp_pre* []) предыдущего фрейма, а предсказанный *LSP* (*vec_hat* []) и коэффициент предсказания (*cb*[0] []) в текущем фрейме следующие:

```
for (i = 0; i <lpc_order; i ++) {
  blsp [0] [i] = (qlsp_pre [i] - vec_hat [i]) / cb [0] [i];
}
```

Б.2.1.4.2.5 Задержка адаптивной кодовой книги

Б.2.1.4.2.5.1 Режим *FE*

Все задержки адаптивной кодовой книги декодируются по индексу задержки, полученному в последнем подфрейме предыдущего фрейма.

Б.2.1.4.2.5.2 Режим *BE*

1) когда *signal_mode_pre*=0, задержки декодируются по данным текущего фрейма.

2) когда *signal_mode_pre*=1 и максимальное различие между индексами задержки смежных подфреймов во фрейме меньше чем 10, задержки декодируются по индексам задержки в текущем фрейме. В каждом подфрейме, где различие в индексах задержки между текущим и предыдущими подфреймами равно или больше 10, задержка декодируется по индексу предыдущего подфрейма.

3) когда *signal_mode_pre* = 2 или 3, задержка декодируется по индексу последнего подфрейма предыдущего фрейма.

Б.2.1.4.2.6 Усиления

Б.2.1.4.2.6.1 Операция индексирования

Б.2.1.4.2.6.1.1 Режим *FE*

Все усиления имеют одинаковое значение, которое декодируется по индексу усиления, полученному в последнем подфрейме предыдущего фрейма.

Б.2.1.4.2.6.1.2 Режим *BE*

Усиление декодируется по данным текущего фрейма.

Б.2.1.4.2.6.2 Операция подстройки

Б.2.1.4.2.6.2.1 Режим *FE*

1) когда *signal_mode_pre*=0, усиления *g_ac*' и *g_ec'* декодируются по данным текущего фрейма. Усиления *g_ac* и *g_ec* получаются умножением *g_ac*' на 0,5 и *g_ec*' на *X* соответственно. *X* удовлетворяет следующему уравнению и вычисляется в каждом подфрейме:

$(g\_ac'*g\_ac') + (g\_ec'*g\_ec')\ B = ((0{,}5*g\_ac') * (0{,}5*g\_ac')) + ((X*g\_ec') * (X*g\_ec'))\ B,$

где $A = norm_{ac} \times norm_{ac}$

$B = norm_{ec} \times norm_{ec}$

$norm_{ac}$ *и* $norm_{ec}$ - соответствующие *RMS* значения адаптивного вектора и вектора возбуждений.

2) Когда *signal_mode_pre*=1, усиление декодируется по данным текущего фрейма.

3) Когда *signal_mode_pre*=2 или 3, усиление для первых 320 отсчетов в или после фрейма, где *BF_flag* изменен от 0 к 1, вычисляются как:

$g\_ac = 0{,}95 \times 10^{(-0{,}4/20)}$;

$g\_ec = X \times 10^{(-0{,}4/20)}$.

После первых 320 отсчетов:

$g\_ac = 0{,}95 \times 10^{(-1{,}2/20)}$;

$g\_ec = X \times 10^{(-1{,}2/20)}$,

где $X = 0{,}05 \times norm_{ac} / norm_{ec}$

Б.2.1.4.2.6.2.2 Режим *BE*

1) Когда *signal_mode_pre*=0 или 1, усиления вычисляются, используя усиления *g_ac*' и g_ec', декодированные по данным текущего фрейма и усилениям *g_ac_pre* и *g_ec_pre* предыдущего подфрейма так, чтобы расчетное значение усилений было в нормальном диапазоне и не генерировало неприятного шума следующим образом:

```
if (g_ac> 1,2589) {
  g_ec = g_ec' * 1,2589/g_ac';
  g_ac = 1,2589;
}
```

```
if (g_ec> 1,2589*g_ec_pre) {
   g_a c = g_ac '* 1,2589*g_ec_pre/g_ec';
    g_ec = g_ec_pre*1,2589;
}
if (signal_mode = 1 & g_ac_pre <1,2589 & g_ac <g_ac_pre*0,7943) {
   g_ac = g_ac_pre*0,7943;
   g_ec = g_ec_pre*0,7943;
}.
```

2) Когда *signal_mode_pre*=2 или 3, операция идентична той, что используется для *signal_mode_pre*=2 или 3 в режиме *FE*.

Б.2.2 Маскировка ошибок для инструмента сжатия тишины

Во фреймах, где полезный битовый поток, полученный в декодере, поврежден или потерян из-за ошибок при передаче, выполняется маскировка ошибок. Когда получен *TX_flag* 1, процесс декодирования идентичен тому, который применяется для *CELP MPEG*–4. Для *TX_flag*=0, 2 или 3 используется процесс декодирования для *TX_flag*=0.

Б.3 Пример настройки инструмента *EP* и маскировки ошибок для *HVXC*

Этот подраздел описывает один пример реализации инструмента защиты от ошибок (*EP*) и метода маскировки ошибок для *HVXC*. Некоторые из перцепционно важных битов защищены *FEC* схемой, а некоторые проверены с помощью *CRC* для принятия решения о том, включены ли ошибочные биты. Когда возникает ошибка *CRC*, выполняется маскировка ошибок для уменьшения заметного ухудшения звучания.

Метод исправления ошибок и настройка инструмента *EP*, а также алгоритм маскировки ошибок, описанные ниже, являются лишь одним примером, и они должны быть изменены в зависимости от фактических условий канала.

Б.3.1 Определения

2/4 кбит/с общие параметры:

*LSP*1	Индекс 1 *LSP*	(5 битов)
*LSP*2	Индекс 2 *LSP*	(7 битов)
*LSP*3	Индекс 3 *LSP*	(5 битов)
*LSP*4	Индекс 4 *LSP*	(1 бит)
VUV	Флаг речевой, неречевой	(2 бита)
Pitch	Параметр шага	(7 битов)
*SE_shape*1	Индекс спектра 0	(4 бита)
*SE_shape*2	Индекс спектра 1	(4 бита)
SE_gain	Индекс усиления спектра	(5 битов)
*VX_shape*1[0]	Индекс 0 стохастической книги шифров	(6 битов)
*VX_shape*1[1]	Индекс 1 стохастической книги шифров	(6 битов)
*VX_gain*1[0]	Индекс 0 книги шифров усиления	(4 бита)
*VX_gain*1[1]	Индекс 1 книги шифров усиления	(4 бита)

параметры 4 кбит/с только:

*LSP*5	Индекс 5 *LSP*	(8 битов)
*SE_shape*3	Индекс 0 спектра 4*k*	(7 битов)
*SE_shape*4	Индекс 1 спектра 4*k*	(10 битов)
*SE_shape*5	Индекс 2 спектра 4*k*	(9 битов)
*SE_shape*6	Индекс 3 спектра 4*k*	(6 битов)
*VX_shape*2[0]	Индекс 0 стохастической книги шифров 4*k*	(5 битов)
*VX_shape*2[1]	Индекс 1 стохастической книги шифров 4*k*	(5 битов)
*VX_shape*2[2]	Индекс 2 стохастической книги шифров 4*k*	(5 битов)
*VX_shape*2[3]	Индекс 3 стохастической книги шифров 4*k*	(5 битов)
*VX_gain*2[0]	Индекс 0 книги шифров усиления 4*k*	(3 бита)
*VX_gain*2[1]	Индекс 1 книги шифров усиления 4*k*	(3 бита)
*VX_gain*2[2]	Индекс 2 книги шифров усиления 4*k*	(3 бита)
*VX_gain*2[3]	Индекс 3 книги шифров усиления 4*k*	(3 бита)

Б.3.2 Канальное кодирование

Б.3.2.1 Выбор защищаемых битов

Согласно чувствительности битов, кодированные биты относятся к нескольким классам. Число битов для каждого класса дано в таблице.Б.1, таблице Б.2 (2 кбит/с), таблице Б.3 и таблице Б.4 (4 кбит/с). Как пример, показаны режимы для скоростей 3,5 кбит/с (для 2 кбит/с) и 6,2 кбит/с (для 4 кбит/с). В этих случаях два исходных фрейма кодера обрабатываются как один набор. Суффикс *"p"* означает предыдущий фрейм, а *"c"* означает текущий.

Для режима 3,5 кбит/с используются 6 классов. Проверка *CRC* применяется для битов классов I, II, III, IV и V. Биты класса VI не проверяются *CRC*.

Таблица Б.1 дает распределение защищенных/незащищенных битов (классы I ... VI) в случае, когда предыдущие и текущие фреймы известны.

Таблица Б.1 – Количество защищенных/незащищенных битов при 3,5 кбит/с

Параметры	Речевой фрейм						
	Битов класса I	Битов класса II	Битов класса III	Битов класса IV	Битов класса V	Битов класса VI	Итого
LSP1p/c	5/5	-	-	-	-	-	10
LSP2p/c	2/2	-	-	-	-	5/5	14
LSP3p/c	1/1	-	-	-	-	4/4	10
LSP4p/c	1/1	-	-	-	-	-	2
VUVp/c	2/2	-	-	-	-	-	4
Pitchp/c	6/6	-	-	-	-	1/1	14
SE_gainp/c	5/5	-	-	-	-	-	10
SE_shape1p	-	4	-	-	-	-	4
SE_shape1c	-	-	-	4	-	-	4
SE_shape2p	-	-	4	-	-	-	4
SE_shape2c	-	-	-	-	4	-	4
Итого	44	4	4	4	4	20	80

Таблица Б.2 дает распределение защищенных/незащищеных битов (классы I ... VI) в случае, когда предыдущий и текущий фреймы неизвестны.

Таблица Б.2 – Количество защищенных/незащищенных битов при 3,5 кбит/с

Параметры	Неречевой фрейм						
	Битов класса I	Битов класса II	Битов класса III	Битов класса IV	Битов класса V	Битов класса VI	Итого
LSP1p/c	5/5	-	-	-	-	-	10
LSP2p/c	4/4	-	-	-	-	3/3	14
LSP3p/c	2/2	-	-	-	-	3/3	10
LSP4p/c	1/1	-	-	-	-	-	2
VUVp/c	2/2	-	-	-	-	-	4
VX_gain1[0]*p/c*	4/4	-	-	-	-	-	8
VX_gain1[1]*p/c*	4/4	-	-	-	-	-	8
VX_shape1[0]*p/*	-	-	-	-	-	6/6	12
VX_shape1[1]*p/*	-	-	-	-	-	6/6	12
Итого	44	0	0	0	0	36	80

Когда предыдущий фрейм неизвестен, а текущий фрейм известен, или когда предыдущий фрейм известен, а текущий фрейм неизвестен, то же самое распределение защищенных/незащищенных битов используется, как показано выше.

Для режима 6,2 кбит/с используются 7 классов. Проверка *CRC* выполняется для битов классов I, II, III, IV, V и VI. Биты класса VII не проверяются *CRC*.

Таблица Б.3 - Количество защищенных/незащищенных битов при 6,2 кбит/с.

Параметры	Речевой фрейм							
	Битов класса I	Битов класса II	Битов класса III	Битов класса IV	Битов класса V	Битов класса VI	Битов класса VII	Итого
LSP1p/c	5/5	-	-	-	-	-	-	10
LSP2p/c	4/4	-	-	-	-	-	3/3	14
LSP3p/c	1/1	-	-	-	-	-	4/4	10
LSP4p/c	1/1	-	-	-	-	-	-	2
LSP5p/c	1/1	-	-	-	-	-	7/7	16
VUVp/c	2/2	-	-	-	-	-	-	4
Pitchp/c	6/6	-	-	-	-	-	1/1	14
SE_gainp/c	5/5	-	-	-	-	-	-	10
SE_shape1p	-	-	4	-	-	-	-	4

Окончание таблицы Б.3

Параметры	Речевой фрейм							
	Битов класса I	Битов класса II	Битов класса III	Битов класса IV	Битов класса V	Битов класса VI	Битов класса VII	Итого
SE_shape1c	-	-	-	-	4	-	-	4
SE_shape2p	-	-	-	4	-	-	-	4
SE_shape2c	-	-	-	-	-	4	-	4
SE_shape3p/c	5/5	-	-	-	-	-	2/2	14
SE_shape4p/c	1/1	9/9	-	-	-	-	-	20
SE_shape5p/c	1/1	8/8	-	-	-	-	-	18
SE_shape6p/c	1/1	5/5	-	-	-	-	-	12
Итого	66	44	4	4	4	4	34	160

Таблица Б.4 дает распределение защищенных/незащищеных битов (классы I … VII) в случае, когда предыдущий и текущий фреймы неизвестны.

Т а б л и ц а Б.4 – Количество защищенных/незащищенных битов при 6,2 кбит/с.

Параметры	Неречевой фрейм							
	Битов класса I	Битов класса II	Битов класса III	Битов класса IV	Битов класса V	Битов класса VI	Битов класса VII	Итого
LSP1p/c	5/5	-	-	-	-	-	-	10
LSP2p/c	4/4	-	-	-	-	-	3/3	14
LSP3p/c	1/1	-	-	-	-	-	4/4	10
LSP4p/c	1/1	-	-	-	-	-	-	2
LSP5p/c	1/1	-	-	-	-	-	7/7	16
VUVp/c	2/2	-	-	-	-	-	-	4
VX_gain1[0]p/c	4/4	-	-	-	-	-	-	8
VX_gain1[1]p/c	4/4	-	-	-	-	-	-	8
VX_shape1[0]p/	-	-	-	-	-	-	6/6	12
VX_shape1[1]p/	-	-	-	-	-	-	6/6	12
VX_gain2[0]p/c	3/3	-	-	-	-	-	-	6
VX_gain2[1]p/c	3/3	-	-	-	-	-	-	6
VX_gain2[2]p/c	3/3	-	-	-	-	-	-	6
VX_gain2[3]p/c	2/2	-	-	-	-	-	1/1	6
VX_shape2[0]p/	-	-	-	-	-	-	5/5	10
VX_shape2[1]p/	-	-	-	-	-	-	5/5	10
VX_shape2[2]p/	-	-	-	-	-	-	5/5	10
VX_shape2[3]p/	-	-	-	-	-	-	5/5	10
Итого	66	0	0	0	0	0	94	160

Когда предыдущий фрейм неизвестен, а текущий фрейм известен, или когда предыдущий фрейм известен, а текущий фрейм неизвестен, то же самое распределение защищенных/незащищенных битов используется, как показано выше.

Порядок битов для входа *UEP* показан в таблицах Б.5 – Б.8 (для 2 кбит/с) и Б.9 – Б.12 (для 4 кбит/с). Эти таблицы показывают порядок битов относительно каждой из комбинаций условия *V/UV* для 2 фреймов. Например, если предыдущий фрейм известен, а текущий фрейм находится в режиме 2 кбит/с, используется таблица Б.8. Порядок битов организован согласно чувствительности к ошибкам. Столбец «Биты» обозначает индекс бита параметра. “0” - младший бит.

Т а б л и ц а Б.5 – Порядок бит 2 кбит/с (известный фрейм – известный фрейм)

Речевой фрейм – речевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
Class I Bit			28	*SE_gainc*	1	54	*SE_shape1c*	1
0	*VUVp*	1	29	*SE_gainc*	0	55	*SE_shape1c*	0
1	*VUVp*	0	30	*LSP1c*	4	*Class V Bit*		
2	*LSP4p*	0	31	*LSP1c*	3	56	*SE_shape2c*	3
3	*SE_gainp*	4	32	*LSP1c*	2	57	*SE_shape2c*	2
4	*SE_gainp*	3	33	*LSP1c*	1	58	*SE_shape2c*	1

Окончание таблицы Б.5

Речевой фрейм – речевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
5	*SE_gainp*	2	34	*LSP1c*	0	59	*SE_shape2c*	0
6	*SE_gainp*	1	35	*Pitchc*	6		*Class VI Bit*	
7	*SE_gainp*	0	36	*Pitchc*	5	60	*LSP2p*	4
8	*LSP1p*	4	37	*Pitchc*	4	61	*LSP2p*	3
9	*LSP1p*	3	38	*Pitchc*	3	62	*LSP2p*	2
10	*LSP1p*	2	39	*Pitchc*	2	63	*LSP2p*	1
11	*LSP1p*	1	40	*Pitchc*	1	64	*LSP2p*	0
12	*LSP1p*	0	41	*LSP2c*	6	65	*LSP3p*	3
13	*Pitchp*	6	42	*LSP3c*	4	66	*LSP3p*	2
14	*Pitchp*	5	43	*LSP2c*	5	67	*LSP3p*	1
15	*Pitchp*	4		*Class II Bit*		68	*LSP3p*	0
16	*Pitchp*	3	44	*SE_shape1p*	3	69	*Pitchp*	0
17	*Pitchp*	2	45	*SE_shape1p*	2	70	*LSP2c*	4
18	*Pitchp*	1	46	*SE_shape1p*	1	71	*LSP2c*	3
19	*LSP2p*	6	47	*SE_shape1p*	0	72	*LSP2c*	2
20	*LSP3p*	4		*Class III Bit*		73	*LSP2c*	1
21	*LSP2p*	5	48	*SE_shape2p*	3	74	*LSP2c*	0
22	*VUVc*	1	49	*SE_shape2p*	2	75	*LSP3c*	3
23	*VUVc*	0	50	*SE_shape2p*	1	76	*LSP3c*	2
24	*LSP4c*	0	51	*SE_shape2p*	0	77	*LSP3c*	1
25	*SE_gainc*	4		*Class IV Bit*		78	*LSP3c*	0
26	*SE_gainc*	3	52	*SE_shape1c*	3	79	*Pitchc*	0
27	*SE_gainc*	2	53	*SE_shape1c*	2			

Таблица Б.6 – Порядок бит 2 кбит/с (известный фрейм – неизвестный фрейм)

Речевой фрейм – неречевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
	Класс I		28	*VX_gain*1[0]*c*	0	54	*LSP2c*	0
0	*VUVp*	1	29	*VX_gain*1[1]c	3	55	*LSP3c*	2
1	*VUVp*	0	30	*VX_gain*1[1]c	2		Класс V	
2	*LSP4p*	0	31	*VX_gain*1[1]c	1	56	*LSP3c*	1
3	*SE_gainp*	4	32	*VX_gain*1[1]c	0	57	*LSP3c*	0
4	*SE_gainp*	3	33	*LSP1c*	4	58	*VX_shape*1[0]*c*	5
5	*SE_gainp*	2	34	*LSP1c*	3	59	*VX_shape*1[0]*c*	4
6	*SE_gainp*	1	35	*LSP1c*	2		Класс VI	
7	*SE_gainp*	0	36	*LSP1c*	1	60	*LSP2p*	4
8	*LSP1p*	4	37	*LSP1c*	0	61	*LSP2p*	3
9	*LSP1p*	3	38	*LSP2c*	6	62	*LSP2p*	2
10	*LSP1p*	2	39	*LSP2c*	5	63	*LSP2p*	1
11	*LSP1p*	1	40	*LSP2c*	4	64	*LSP2p*	0
12	*LSP1p*	0	41	*LSP2c*	3	65	*LSP3p*	3
13	*Pitchp*	6	42	LSP3c	4	66	*LSP3p*	2
14	*Pitchp*	5	43	*LSP3c*	3	67	*LSP3p*	1
15	*Pitchp*	4		Класс II		68	*LSP3p*	0
16	*Pitchp*	3	44	*SE_shape1p*	3	69	*Pitchp*	0
17	*Pitchp*	2	45	*SE_shape1p*	2	70	*VX_shape*1[0]*c*	3
18	*Pitchp*	1	46	*SE_shape1p*	1	71	*VX_shape*1[0]*c*	2
19	*LSP2p*	6	47	*SE_shape1p*	0	72	*VX_shape*1[0]*c*	1
20	*LSP3p*	4		Класс III		73	*VX_shape*1[0]*c*	0
21	*LSP2p*	5	48	*SE_shape2p*	3	74	*VX_shape*1[1]*c*	5
22	*VUVc*	1	49	*SE_shape2p*	2	75	*VX_shape*1[1]*c*	4

Окончание таблицы Б.6

Речевой фрейм – неречевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
23	*VUVc*	0	50	*SE_shape2p*	1	76	*VX_shape*1[1]*c*	3
24	*LSP*4*c*	0	51	*SE_shape2p*	0	77	*VX_shape*1[1]*c*	2
25	*VX_gain*1[0]*c*	3	Класс IV			78	*VX_shape*1[1]*c*	1
26	*VX_gain*1[0]*c*	2	52	*LSP*2*c*	2	79	*VX_shape*1[1]*c*	0
27	*VX_gain*1[0]*c*	1	53	*LSP*2*c*	1			

Таблица Б.7 – Порядок бит 2 кбит/с (неизвестный фрейм – известный фрейм)

Неречевой фрейм – речевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
Класс I			28	*SE_gainc*	1	54	*SE_shape*1*c*	1
0	*VUVp*	1	29	*SE_gainc*	0	55	*SE_shape*1*c*	0
1	*VUVp*	0	30	*LSP*1*c*	4	Класс V		
2	*LSP*4*p*	0	31	*LSP*1*c*	3	56	*SE_shape*2*c*	3
3	*VX_gain*1[0]*p*	3	32	*LSP*1*c*	2	57	*SE_shape*2*c*	2
4	*VX_gain*1[0]*p*	2	33	*LSP*1*c*	1	58	*SE_shape*2*c*	1
5	*VX_gain*1[0]*p*	1	34	LSP1*c*	0	59	*SE_shape*2*c*	0
6	*VX_gain*1[0]*p*	0	35	*Pitchc*	6	Класс VI		
7	*VX_gain*1[0]*p*	3	36	*Pitchc*	5	60	*VX_shape*1[0]*p*	3
8	*VX_gain*1[0]*p*	2	37	*Pitchc*	4	61	*VX_shape*1[0]*p*	2
9	*VX_gain*1[0]*p*	1	38	*Pitchc*	3	62	*VX_shape*1[0]*p*	1
10	*VX_gain*1[0]*p*	0	39	*Pitchc*	2	63	*VX_shape*1[0]*p*	0
11	*LSP*1*p*	4	40	*Pitchc*	1	64	*VX_shape*1[0]*p*	5
12	*LSP*1*p*	3	41	*LSP*2*c*	6	65	*VX_shape*1[0]*p*	4
13	*LSP*1*p*	2	42	*LSP*3*c*	4	66	*VX_shape*1[0]*p*	3
14	*LSP*1*p*	1	43	*LSP*2*c*	5	67	*VX_shape*1[0]*p*	2
15	*LSP*1*p*	0	Класс II			68	*VX_shape*1[0]*p*	1
16	*LSP*2*p*	6	44	*LSP*2*p*	2	69	*VX_shape*1[0]*p*	0
17	*LSP*2*p*	5	45	*LSP*2*p*	1	70	*LSP*2*c*	4
18	*LSP*2*p*	4	46	*LSP*2*p*	0	71	*LSP*2*c*	3
19	*LSP*2*p*	3	47	*LSP*3*p*	2	72	*LSP*2*c*	2
20	*LSP*3*p*	4	Класс III			73	*LSP*2*c*	1
21	*LSP*3*p*	3	48	*LSP*3*p*	1	74	*LSP*2*c*	0
22	*VUVc*	1	49	*LSP*3*p*	0	75	*LSP*3*c*	3
23	*VUVc*	0	50	*VX_shape*1[0]*p*	5	76	*LSP*3*c*	2
24	*LSP*4*c*	0	51	*VX_shape*1[0]*p*	4	77	*LSP*3*c*	1
25	*SE_gainc*	4	Класс IV			78	*LSP*3*c*	0
26	*SE_gainc*	3	52	*VX_shape*1[0]*p*	3	79	*Pitchc*	0
27	*SE_gainc*	2	53	*VX_shape*1[0]*p*	2			

Таблица Б.8 - Порядок бит 2 кбит/с (неизвестный фрейм – неизвестный фрейм)

Неречевой фрейм – неречевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
Класс I			28	*VX_gain*1[0]*c*	0	54	*LSP*2*c*	0
0	*VUVp*	1	29	*VX_gain*1[1]*c*	3	55	*LSP*3*c*	2
1	*VUVp*	0	30	*VX_gain*1[1]*c*	2	Класс V		
2	*LSP*4*p*	0	31	*VX_gain*1[1]*c*	1	56	*LSP*3*c*	1
3	*VX_gain*1[0]*p*	3	32	*VX_gain*1[1]*c*	0	57	*LSP*3*c*	0
4	*VX_gain*1[0]*p*	2	33	*LSP*1*c*	4	58	*VX_shape*1[0]*c*	5
5	*VX_gain*1[0]*p*	1	34	*LSP*1*c*	3	59	*VX_shape*1[0]*c*	4
6	*VX_gain*1[0]*p*	0	35	*LSP*1*c*	2	Класс VI		

Окончание таблицы Б.8

Неречевой фрейм – неречевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
7	*VX_gain*1[0]*p*	3	36	*LSP*1*c*	1	60	*VX_shape*1[0]*p*	3
8	*VX_gain*1[0]*p*	2	37	*LSP*1*c*	0	61	*VX_shape*1[0]*p*	2
9	*VX_gain*1[0]*p*	1	38	*LSP*2*c*	6	62	*VX_shape*1[0]*p*	1
10	*VX_gain*1[0]*p*	0	39	*LSP*2*c*	5	63	*VX_shape*1[0]*p*	0
11	*LSP*1*p*	4	40	*LSP*2*c*	4	64	*VX_shape*1[0]*p*	5
12	*LSP*1*p*	3	41	*LSP*2*c*	3	65	*VX_shape*1[0]*p*	4
13	*LSP*1*p*	2	42	*LSP*3*c*	4	66	*VX_shape*1[0]*p*	3
14	*LSP*1*p*	1	43	*LSP*3*c*	3	67	*VX_shape*1[0]*p*	2
15	*LSP*1*p*	0	Класс II			68	*VX_shape*1[0]*p*	1
16	*LSP*2*p*	6	44	*LSP*2*p*	2	69	*VX_shape*1[0]*p*	0
17	*LSP*2*p*	5	45	*LSP*2*p*	1	70	*VX_shape*1[0]*p*	3
18	*LSP*2*p*	4	46	*LSP*2*p*	0	71	*VX_shape*1[0]*p*	2
19	*LSP*2*p*	3	47	*LSP*3*p*	2	72	*VX_shape*1[0]*p*	1
20	LSP3p	4	Класс III			73	*VX_shape*1[0]*p*	0
21	*LSP*3*p*	3	48	*LSP*3*p*	1	74	*VX_shape*1[0]*p*	5
22	*VUVc*	1	49	*LSP*3*p*	0	75	*VX_shape*1[0]*p*	4
23	*VUVc*	0	50	*VX_shape*1[0]*p*	5	76	*VX_shape*1[0]*p*	3
24	*LSP*4*c*	0	51	*VX_shape*1[0]*p*	4	77	*VX_shape*1[0]*p*	2
25	*VX_gain*1[0]*c*	3	Класс IV			78	*VX_shape*1[0]*p*	1
26	*VX_gain*1[0]*c*	2	52	*LSP*2*c*	2	79	*VX_shape*1[0]*p*	0
27	*VX_gain*1[0]*c*	1	53	*LSP*2*c*	1			

Таблица Б.9 - Порядок бит 4 кбит/с (известный фрейм – известный фрейм)

Речевой фрейм – речевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
Класс I			55	*LSP*2*c*	3	Класс III		
0	*VUVp*	1	56	*SE_shape*3*c*	6	110	*SE_shape*1*p*	3
1	*VUVp*	0	57	*SE_shape*3*c*	5	111	*SE_shape*1*p*	2
2	*LSP*4*p*	0	58	*SE_shape*3*c*	4	112	*SE_shape*1*p*	1
3	*SE_gainp*	4	59	*SE_shape*3*c*	3	113	*SE_shape*1*p*	0
4	*SE_gainp*	3	60	*SE_shape*3*c*	2	Класс IV		
5	*SE_gainp*	2	61	*LSP*3*c*	4	114	*SE_shape*2*p*	3
6	*SE_gainp*	1	62	*LSP*5*c*	7	115	*SE_shape*2*p*	2
7	*SE_gainp*	0	63	*SE_shape*4*c*	9	116	*SE_shape*2*p*	1
8	*LSP*1*p*	4	64	*SE_shape*5*c*	8	117	*SE_shape*2*p*	0
9	*LSP*1*p*	3	65	*SE_shape*6*c*	5	Класс V		
10	*LSP*1*p*	2	Класс II			118	*SE_shape*1*c*	3
11	*LSP*1*p*	1	66	*SE_shape*4*p*	8	119	*SE_shape*1*c*	2
12	*LSP*1*p*	0	67	*SE_shape*4*p*	7	120	*SE_shape*1*c*	1
13	*Pitchp*	6	68	*SE_shape*4*p*	6	121	*SE_shape*1*c*	0
14	*Pitchp*	5	69	*SE_shape*4*p*	5	Класс VI		
15	*Pitchp*	4	70	*SE_shape*4*p*	4	122	*SE_shape*2*c*	3
16	*Pitchp*	3	71	*SE_shape*4*p*	3	123	*SE_shape*2*c*	2
17	*Pitchp*	2	72	*SE_shape*4*p*	2	124	*SE_shape*2*c*	1
18	*Pitchp*	1	73	*SE_shape*4*p*	1	125	*SE_shape*2*c*	0
19	*LSP*2*p*	6	74	*SE_shape*4*p*	0	Класс VII		
20	*LSP*2*p*	5	75	*SE_shape*5*p*	7	126	*LSP*2*p*	2
21	*LSP*2*p*	4	76	*SE_shape*5*p*	6	127	*LSP*2*p*	1
22	*LSP*2*p*	3	77	*SE_shape*5*p*	5	128	*LSP*2*p*	0
23	*SE_shape*3*p*	6	78	*SE_shape*5*p*	4	129	*LSP*3*p*	3
24	*SE_shape*3*p*	5	79	*SE_shape*5*p*	3	130	*LSP*3*p*	2

Окончание таблицы Б.9

Речевой фрейм – речевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
25	*SE_shape3p*	4	80	*SE_shape5p*	2	131	*LSP3p*	1
26	*SE_shape3p*	3	81	*SE_shape5p*	1	132	*LSP3p*	0
27	*SE_shape3p*	2	82	*SE_shape5p*	0	133	*LSP5p*	6
28	*LSP3p*	4	83	*SE_shape6p*	4	134	*LSP5p*	5
29	*LSP5p*	7	84	*SE_shape6p*	3	135	*LSP5p*	4
30	*SE_shape4p*	9	85	*SE_shape6p*	2	136	*LSP5p*	3
31	*SE_shape5p*	8	86	*SE_shape6p*	1	137	*LSP5p*	2
32	*SE_shape6p*	5	87	*SE_shape6p*	0	138	*LSP5p*	1
33	*VUVc*	1	88	*SE_shape4c*	8	139	*LSP5p*	0
34	*VUVc*	0	89	*SE_shape4c*	7	140	*Pitchp*	0
35	*LSP4c*	0	90	*SE_shape4c*	6	141	*SE_shape3p*	1
36	*SE_gainc*	4	91	*SE_shape4c*	5	142	*SE_shape3p*	0
37	*SE_gainc*	3	92	*SE_shape4c*	4	143	*LSP2c*	2
38	*SE_gainc*	2	93	*SE_shape4c*	3	144	*LSP2c*	1
39	*SE_gainc*	1	94	*SE_shape4c*	2	145	*LSP2c*	0
40	*SE_gainc*	0	95	*SE_shape4c*	1	146	*LSP3c*	3
41	*LSP1c*	4	96	*SE_shape4c*	0	147	*LSP3c*	2
42	*LSP1c*	3	97	*SE_shape5c*	7	148	*LSP3c*	1
43	*LSP1c*	2	98	*SE_shape5c*	6	149	*LSP3c*	0
44	*LSP1c*	1	99	*SE_shape5c*	5	150	*LSP5c*	6
45	*LSP1c*	0	100	*SE_shape5c*	4	151	*LSP5c*	5
46	*Pitchc*	6	101	*SE_shape5c*	3	152	*LSP5c*	4
47	*Pitchc*	5	102	*SE_shape5c*	2	153	*LSP5c*	3
48	*Pitchc*	4	103	*SE_shape5c*	1	154	*LSP5c*	2
49	*Pitchc*	3	104	*SE_shape5c*	0	155	*LSP5c*	1
50	*Pitchc*	2	105	*SE_shape6c*	4	156	*LSP5c*	0
51	*Pitchc*	1	106	*SE_shape6c*	3	157	*Pitchp*	0
52	*LSP2c*	6	107	*SE_shape6c*	2	158	*SE_shape3c*	1
53	*LSP2c*	5	108	*SE_shape6c*	1	159	*SE_shape3c*	0
54	*LSP2c*	4	109	*SE_shape6c*	0			

Таблица Б.10 – Порядок бит 4 кбит/с (известный фрейм – неизвестный фрейм)

Речевой фрейм– неречевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
Класс I			55	*VX_gain2[0]c*	2	Класс III		
0	*VUVp*	1	56	*VX_gain2[0]c*	1	110	*SE_shape1p*	3
1	*VUVp*	0	57	*VX_gain2[0]c*	0	111	*SE_shape1p*	2
2	*LSP4p*	0	58	*VX_gain2[1]c*	2	112	*SE_shape1p*	1
3	*SE_gainp*	4	59	*VX_gain2[1]c*	1	113	*SE_shape1p*	0
4	*SE_gainp*	3	60	*VX_gain2[1]c*	0	Класс IV		
5	*SE_gainp*	2	61	*VX_gain2[2]c*	2	114	*SE_shape2p*	3
6	*SE_gainp*	1	62	*VX_gain2[2]c*	1	115	*SE_shape2p*	2
7	*SE_gainp*	0	63	*VX_gain2[2]c*	0	116	*SE_shape2p*	1
8	*LSP1p*	4	64	*VX_gain2[3]c*	2	117	*SE_shape2p*	0
9	*LSP1p*	3	65	*VX_gain2[3]c*	1	Класс V		
10	*LSP1p*	2	Класс II			118	*VX_shape1[1]c*	4
11	*LSP1p*	1	66	*SE_shape4p*	8	119	*VX_shape1[1]c*	3
12	*LSP1p*	0	67	*SE_shape4p*	7	120	*VX_shape1[1]c*	2
13	*Pitchp*	6	68	*SE_shape4p*	6	121	*VX_shape1[1]c*	1
14	*Pitchp*	5	69	*SE_shape4p*	5	Класс VI		
15	*Pitchp*	4	70	*SE_shape4p*	4	122	*VX_shape1[1]c*	0

Окончание таблицы Б.10

Речевой фрейм– неречевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
16	*Pitchp*	3	71	*SE_shape4p*	3	123	*VX_shape2[0]c*	4
17	*Pitchp*	2	72	*SE_shape4p*	2	124	*VX_shape2[0]c*	3
18	*Pitchp*	1	73	*SE_shape4p*	1	125	*VX_shape2[0]c*	2
19	*LSP2p*	6	74	*SE_shape4p*	0	Класс VII		
20	*LSP2p*	5	75	*SE_shape5p*	7	126	*LSP2p*	2
21	*LSP2p*	4	76	*SE_shape5p*	6	127	*LSP2p*	1
22	*LSP2p*	3	77	*SE_shape5p*	5	128	LSP2p	0
23	*SE_shape3p*	6	78	*SE_shape5p*	4	129	*LSP3p*	3
24	*SE_shape3p*	5	79	*SE_shape5p*	3	130	*LSP3p*	2
25	*SE_shape3p*	4	80	*SE_shape5p*	2	131	*LSP3p*	1
26	*SE_shape3p*	3	81	*SE_shape5p*	1	132	*LSP3p*	0
27	*SE_shape3p*	2	82	*SE_shape5p*	0	133	*LSP5p*	6
28	*LSP3p*	4	83	*SE_shape6p*	4	134	*LSP5p*	5
29	*LSP5p*	7	84	*SE_shape6p*	3	135	*LSP5p*	4
30	*SE_shape4p*	9	85	*SE_shape6p*	2	136	*LSP5p*	3
31	*SE_shape5p*	8	86	*SE_shape6p*	1	137	*LSP5p*	2
32	*SE_shape6p*	5	87	*SE_shape6p*	0	138	*LSP5p*	1
33	*VUVc*	1	88	*VX_gain2[3]c*	0	139	*LSP5p*	0
34	*VUVc*	0	89	*LSP2c*	2	140	*Pitchp*	0
35	*LSP4c*	0	90	*LSP2c*	1	141	*SE_shape3p*	1
36	*VX_gain1[0]c*	3	91	*LSP2c*	0	142	*SE_shape3p*	0
37	*VX_gain1[0]c*	2	92	*LSP3c*	3	143	*VX_shape2[0]c*	1
38	*VX_gain1[0]c*	1	93	*LSP3c*	2	144	*VX_shape2[0]c*	0
39	*VX_gain1[0]c*	0	94	*LSP3c*	1	145	*VX_shape2[1]c*	4
40	*VX_gain1[1]c*	3	95	*LSP3c*	0	146	*VX_shape2[1]c*	3
41	*VX_gain1[1]c*	2	96	*LSP5c*	6	147	*VX_shape2[1]c*	2
42	*VX_gain1[1]c*	1	97	*LSP5c*	5	148	*VX_shape2[1]c*	1
43	*VX_gain1[1]c*	0	98	*LSP5c*	4	149	*VX_shape2[1]c*	0
44	*LSP1c*	4	99	*LSP5c*	3	150	*VX_shape2[2]c*	4
45	*LSP1c*	3	100	*LSP5c*	2	151	*VX_shape2[2]c*	3
46	*LSP1c*	2	101	*LSP5c*	1	152	*VX_shape2[2]c*	2
47	*LSP1c*	1	102	*LSP5c*	0	153	*VX_shape2[2]c*	1
48	*LSP1c*	0	103	*VX_shape1[0]c*	5	154	*VX_shape2[2]c*	0
49	*LSP2c*	6	104	*VX_shape1[0]c*	4	155	*VX_shape2[3]c*	4
50	*LSP2c*	5	105	*VX_shape1[0]c*	3	156	*VX_shape2[3]c*	3
51	*LSP2c*	4	106	*VX_shape1[0]c*	2	157	*VX_shape2[3]c*	2
52	*LSP2c*	3	107	*VX_shape1[0]c*	1	158	*VX_shape2[3]c*	1
53	*LSP3c*	4	108	*VX_shape1[0]c*	0	159	*VX_shape2[3]c*	0
54	*LSP5c*	7	109	*VX_shape1[1]c*	5			

Т а б л и ц а Б.11 - Порядок бит 4 кбит/с (неизвестный фрейм – известный фрейм)

Неречевой фрейм– речевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
Класс I			55	*LSP2c*	3	Класс III		
0	*VUVp*	1	56	*SE_shape3c*	6	110	*VX_shape1[1]p*	4
1	*VUVp*	0	57	*SE_shape3c*	5	111	*VX_shape1[1]p*	3
2	*LSP4p*	0	58	*SE_shape3c*	4	112	*VX_shape1[1]p*	2
3	*VX_gain1[0]p*	3	59	*SE_shape3c*	3	113	*VX_shape1[1]p*	1
4	*VX_gain1[0]p*	2	60	*SE_shape3c*	2	Класс IV		
5	*VX_gain1[0]p*	1	61	*LSP3c*	4	114	*VX_shape1[1]p*	0
6	*VX_gain1[0]p*	0	62	*LSP5c*	7	115	*VX_shape2[0]p*	4

Окончание таблицы Б.11

Неречевой фрейм– речевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
7	*VX_gain1[1]p*	3	63	*SE_shape4c*	9	116	*VX_shape2[0]p*	3
8	*VX_gain1[1]p*	2	64	*SE_shape5c*	8	117	*VX_shape2[0]p*	2
9	*VX_gain1[1]p*	1	65	*SE_shape6c*	5	Класс V		
10	*VX_gain1[1]p*	0	Класс II			118	*SE_shape1c*	3
11	*LSP1p*	4	66	*VX_gain2[3]p*	0	119	*SE_shape1c*	2
12	*LSP1p*	3	67	*LSP2p*	2	120	*SE_shape1c*	1
13	*LSP1p*	2	68	*LSP2p*	1	121	*SE_shape1c*	0
14	*LSP1p*	1	69	*LSP2p*	0	Класс VI		
15	*LSP1p*	0	70	*LSP3p*	3	122	*SE_shape2c*	3
16	*LSP2p*	6	71	*LSP3p*	2	123	*SE_shape2c*	2
17	*LSP2p*	5	72	*LSP3p*	1	124	*SE_shape2c*	1
18	*LSP2p*	4	73	*LSP3p*	0	125	*SE_shape2c*	0
19	*LSP2p*	3	74	*LSP5p*	6	Класс VII		
20	*LSP3p*	4	75	*LSP5p*	5	126	*VX_shape2[0]p*	1
21	*LSP5p*	7	76	*LSP5p*	4	127	*VX_shape2[0]p*	0
22	*VX_gain2[0]c*	2	77	*LSP5p*	3	128	*VX_shape2[1]p*	4
23	*VX_gain2[0]c*	1	78	*LSP5p*	2	129	*VX_shape2[1]p*	3
24	*VX_gain2[0]c*	0	79	*LSP5p*	1	130	*VX_shape2[1]p*	2
25	*VX_gain2[1]c*	2	80	*LSP5c*	0	131	*VX_shape2[1]p*	1
26	*VX_gain2[1]c*	1	81	*VX_shape1[0]p*	5	132	*VX_shape2[1]p*	0
27	*VX_gain2[1]c*	0	82	*VX_shape1[0]p*	4	133	*VX_shape2[2]p*	4
28	*VX_gain2[2]c*	2	83	*VX_shape1[0]p*	3	134	*VX_shape2[2]p*	3
29	*VX_gain2[2]c*	1	84	*VX_shape1[0]p*	2	135	*VX_shape2[2]p*	2
30	*VX_gain2[2]c*	0	85	*VX_shape1[0]p*	1	136	*VX_shape2[2]p*	1
31	*VX_gain2[3]c*	2	86	*VX_shape1[0]p*	0	137	*VX_shape2[2]p*	0
32	*VX_gain2[3]c*	1	87	*VX_shape1[1]p*	5	138	*VX_shape2[3]p*	4
33	*VUVc*	1	88	*SE_shape4c*	8	139	*VX_shape2[3]p*	3
34	*VUVc*	0	89	*SE_shape4c*	7	140	*VX_shape2[3]p*	2
35	*LSP4c*	0	90	*SE_shape4c*	6	141	*VX_shape2[3]p*	1
36	*SE_gainc*	4	91	*SE_shape4c*	5	142	*VX_shape2[3]p*	0
37	*SE_gainc*	3	92	*SE_shape4c*	4	143	*LSP2c*	2
38	*SE_gainc*	2	93	*SE_shape4c*	3	144	*LSP2c*	1
39	*SE_gainc*	1	94	*SE_shape4c*	2	145	*LSP2c*	0
40	*SE_gainc*	0	95	*SE_shape4c*	1	146	*LSP3c*	3
41	*LSP1c*	4	96	*SE_shape4c*	0	147	*LSP3c*	2
42	*LSP1c*	3	97	*SE_shape5c*	7	148	*LSP3c*	1
43	*LSP1c*	2	98c	*SE_shape5c*	6	149	*LSP3c*	0
44	*LSP1c*	1	99c	*SE_shape5c*	5	150	*LSP5c*	6
45	*LSP1c*	0	100	*SE_shape5c*	4	151	*LSP5c*	5
46	*Pitchc*	6	101	*SE_shape5c*	3	152	*LSP5c*	4
47	*Pitchc*	5	102	*SE_shape5c*	2	153	*LSP5c*	3
48	*Pitchc*	4	103	*SE_shape5c*	1	154	*LSP5c*	2
49	*Pitchc*	3	104	*SE_shape5c*	0	155	*LSP5c*	1
50	*Pitchc*	2	105	*SE_shape6c*	4	156	*LSP5c*	0
51	*Pitchc*	1	106	*SE_shape6c*	3	157	*Pitchc*	0
52	*LSP2c*	6	107	*SE_shape6c*	2	158	*SE_shape3c*	1
53	*LSP2c*	5	108	*SE_shape6c*	1	159	*SE_shape3c*	0
54	*LSP2c*	4	109	*SE_shape6c*	0			

Таблица Б.12 – Порядок бит 4 кбит/с (неизвестный фрейм – неизвестный фрейм)

Неречевой фрейм – неречевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
Класс I			55	*VX_gain2[0]c*	2	Класс III		
0	*VUVp*	1	56	*VX_gain2[0]c*	1	110	*VX_shape1[1]p*	4
1	*VUVp*	0	57	*VX_gain2[0]c*	0	111	*VX_shape1[1]p*	3
2	*LSP4p*	0	58	*VX_gain2[1]c*	2	112	*VX_shape1[1]p*	2
3	*VX_gain1[0]p*	3	59	*VX_gain2[1]c*	1	113	*VX_shape1[1]p*	1
4	*VX_gain1[0]p*	2	60	*VX_gain2[1]c*	0	Класс IV		
5	*VX_gain1[0]p*	1	61	*VX_gain2[2]c*	2	114	*VX_shape1[1]p*	0
6	*VX_gain1[0]p*	0	62	*VX_gain2[2]c*	1	115	*VX_shape2[0]p*	4
7	*VX_gain1[1]p*	3	63	*VX_gain2[2]c*	0	116	*VX_shape2[0]p*	3
8	*VX_gain1[1]p*	2	64	*VX_gain2[3]c*	2	117	*VX_shape2[0]p*	2
9	*VX_gain1[1]p*	1	65	*VX_gain2[3]c*	1	Класс V		
10	*VX_gain1[1]p*	0	Класс II			118	*VX_shape1[1]c*	4
11	*LSP1p*	4	66	*VX_gain2[3]p*	0	119	*VX_shape1[1]c*	3
12	*LSP1p*	3	67	*LSP2p*	2	120	*VX_shape1[1]c*	2
13	*LSP1p*	2	68	*LSP2p*	1	121	*VX_shape1[1]c*	1
14	*LSP1p*	1	69	*LSP2p*	0	Класс VI		
15	*LSP1p*	0	70	*LSP3p*	3	122	*VX_shape1[1]c*	0
16	*LSP2p*	6	71	*LSP3p*	2	123	*VX_shape2[0]c*	4
17	*LSP2p*	5	72	*LSP3p*	1	124	*VX_shape2[0]c*	3
18	*LSP2p*	4	73	*LSP3p*	0	125	*VX_shape2[0]c*	2
19	*LSP2p*	3	74	*LSP5p*	6	Класс VII		
20	*LSP3p*	4	75	*LSP5p*	5	126	*VX_shape2[0]p*	1
21	*LSP5p*	7	76	*LSP5p*	4	127	*VX_shape2[0]p*	0
22	*VX_gain2[0]c*	2	77	*LSP5p*	3	128	*VX_shape2[1]p*	4
23	*VX_gain2[0]c*	1	78	*LSP5p*	2	129	*VX_shape2[1]p*	3
24	*VX_gain2[0]c*	0	79	*LSP5p*	1	130	*VX_shape2[1]p*	2
25	*VX_gain2[1]c*	2	80	*LSP5c*	0	131	*VX_shape2[1]p*	1
26	*VX_gain2[1]c*	1	81	*VX_shape1[0]p*	5	132	*VX_shape2[1]p*	0
27	*VX_gain2[1]p*	0	82	*VX_shape1[0]p*	4	133	*VX_shape2[2]p*	4
28	*VX_gain2[2]p*	2	83	*VX_shape1[0]p*	3	134	*VX_shape2[2]p*	3
29	*VX_gain2[2]p*	1	84	*VX_shape1[0]p*	2	135	*VX_shape2[2]p*	2
30	*VX_gain2[2]p*	0	85	*VX_shape1[0]p*	1	136	*VX_shape2[2]p*	1
31	*VX_gain2[3]p*	2	86	*VX_shape1[0]p*	0	137	*VX_shape2[2]p*	0
32	*VX_gain2[3]p*	1	87	*VX_shape1[1]p*	5	138	*VX_shape2[3]p*	4
33	*VUVc*	1	88	*VX_gain2[3]c*	0	139	*VX_shape2[3]p*	3
34	*VUVc*	0	89	*LSP2c*	2	140	*VX_shape2[3]p*	2
35	*LSP4c*	0	90	*LSP2c*	1	141	*VX_shape2[3]p*	1
36	*VX_gain1[0]c*	3	91	*LSP2c*	0	142	*VX_shape2[3]p*	0
37	*VX_gain1[0]c*	2	92	*LSP3c*	3	143	*VX_shape2[0]c*	1
38	*VX_gain1[0]c*	1	93	*LSP3c*	2	144	*VX_shape2[0]c*	0
39	*VX_gain1[0]c*	0	94	*LSP3c*	1	145	*VX_shape2[1]c*	4
40	*VX_gain1[1]c*	3	95	*LSP3c*	0	146	*VX_shape2[1]c*	3
41	*VX_gain1[1]c*	2	LSP1c	*LSP5c*	6	147	*VX_shape2[1]c*	2
42	*VX_gain1[1]c*	1	LSP1c	*LSP5c*	5	148	*VX_shape2[1]c*	1
43	*VX_gain1[1]c*	0	LSP1c	*LSP5c*	4	149	*VX_shape2[1]c*	0
44	*LSP1c*	4	LSP1c	*LSP5c*	3	150	*VX_shape2[2]c*	4
45	*LSP1c*	3	LSP2c	*LSP5c*	2	151	*VX_shape2[2]c*	3
46	*LSP1c*	2	101	*LSP5c*	1	152	*VX_shape2[2]c*	2
47	*LSP1c*	1	102	*LSP5c*	0	153	*VX_shape2[2]c*	1
48	*LSP1c*	0	103	*VX_shape1[0]c*	5	154	*VX_shape2[2]c*	0
49	*LSP2c*	6	104	*VX_shape1[0]c*	4	155	*VX_shape2[3]c*	4

Окончание таблицы Б.12

Неречевой фрейм – неречевой фрейм								
Номер	Пункт	Биты	Номер	Пункт	Биты	Номер	Пункт	Биты
50	*LSP2c*	5	105	*VX_shape1[0]c*	3	156	*VX_shape2[3]c*	3
51	*LSP2c*	4	106	*VX_shape1[0]c*	2	157	*VX_shape2[3]c*	2
52	*LSP2c*	3	107	*VX_shape1[0]c*	1	158	*VX_shape2[3]c*	1
53	*LSP3c*	4	108	*VX_shape1[0]c*	0	159	*VX_shape2[3]c*	0
54	*LSP5c*	7	109	*VX_shape1[1]c*	5			

Б.3.3 Настройка инструмента EP

Б.3.3.1 Назначение битов

Таблица Б.13 показывает пример назначения битов для использования инструментом *EP*. В этой таблице описано назначение битов для кодеров 4 кбит/с и 2 кбит/с.

Т а б л и ц а Б.13 – Назначение битов для использования инструментом *EP*

	Кодер источника 2 кбит/с	Кодер источника 4 кбит/с
Класс I		
Биты кодера источника	44	66
Четность CRC	6	6
Кодовая скорость	8/16	8/16
Класс I всего	100	144
Класс II		
Биты кодера источника	4	44
Четность CRC	1	6
Кодовая скорость	8/8	8/8
Класс II всего	5	50
Класс III		
Биты кодера источника	4	4
Четность CRC	1	1
Кодовая скорость	8/8	8/8
Класс III всего	5	5
Класс IV		
Биты кодера источника	4	4
Четность CRC	1	1
Кодовая скорость	8/8	8/8
Класс IV всего	5	5
Класс V		
Биты кодера источника	4	4
Четность CRC	1	1
Кодовая скорость	8/8	8/8
Класс V всего	5	5
Класс VI		
Биты кодера источника	20	4
Четность CRC	0	1
Кодовая скорость	8/8	8/8
Класс VI всего	20	5
Класс VII		
Биты кодера источника		34
Четность CRC		0
Кодовая скорость		8/8
Класс VII всего		34
Суммарно битов всех классов	140	248
Битовая скорость	3,5 кбит/с	6,2 кбит/с

Класс I:

CRC охватывает все биты Класса I, включая *CRC*.

Класс II–V (2 кбит/с), II–VI (4 кбит/с):

По крайней мере одни биты *CRC* охватывают исходные биты кодера этих классов.

Класс VI (2 кбит/с), VII (4 кбит/с):

Исходные биты кодера не проверяются *CRC* и не защищаются какой-либо схемой коррекции.

Б.3.4 Маскировка ошибок

Когда обнаружена ошибка *CRC*, выполняется маскировка ошибок (маскировка «плохого» фрейма). Пример метода маскировки описан ниже.

Состояние маскировки текущего фрейма обновляется на основании результата декодирования *CRC* Класса I. Диаграмма переходов показана на рисунке Б.2. Начальное состояние = 0. Стрелка с символом «1» обозначает переход для плохого фрейма, а с символом «0» – для хорошего фрейма.

Б.3.4.1 Замена параметров

Согласно текущему состоянию выполняется следующая замена параметров. При условии отсутствия ошибок, состояние равно 0, а полученные исходные биты кодера используются без маскировки.

Б.3.4.1.1 Параметры *LSP*

В *state* = 1…6, параметры *LSP* заменяются таковыми из предыдущих.

Когда state=7, при *LSP*4=0 (режим квантования *LSP* без межфреймового предсказания), параметры *LSP* вычисляются по всем индексам *LSP*, полученным в текущем фрейме. Если *LSP*4=1 (режим квантования *LSP* с межфреймовым кодированием), параметры *LSP* вычисляются следующим методом.

В этом режиме параметры *LSP* индекса *LSP*1 интерполируются с предыдущими *LSP*.

$LSP_{base}(n) = p \times LSP_{prev}(n) + (1 - p)\, LSP_{1st}(n)$ для n=1.. 10 (1)

$LSP_{base}(n)$ - параметры *LSP* базового слоя, $LSP_{prev}(n)$ - предыдущие *LSP*, $LSP_{1st}(n)$ – декодированные *LSP* текущего индекса *LSP*1, p - коэффициент интерполяции. p изменяется согласно числу предыдущих ошибочных фреймов *CRC* битов Класса I, как показано в таблице Б.14. Индексы *LSP*, *LSP*2, *LSP*3 и *LSP*5 не используются, $LSP_{base}(n)$ используются как текущие параметры *LSP*.

Таблица Б.14 – Коэффициент *p*

Фрейм	p
0	0,7
1	0,6
2	0,5
3	0,4
4	0,3
5	0,2
6	0,1
=>7	0,0

Б.3.4.1.2 Переменная *Mute*

Согласно значению *state*, переменная *mute* управляет выходным уровнем речи.

При *state* = 7, используется среднее между 1,0 и значением *mute* предыдущего фрейма (= 0,5 (1,0 + предыдущее значение *mute*)), но, когда это значение больше, чем 0,8, значение *mute* заменяется на 0,8.

Значения *mute* приведены в таблице Б.15.

Таблица Б.15 – Значения *mute*

Состояние	mute
0	1,000
1	0,800
2	0,700
3	0,500
4	0,250
5	0,125
6	0,000
7	Среднее/0,800

Б.3.4.1.3 Замена и контроль усиления «голосовых» параметров

В *state*=1.. 6, параметр спектра *SE_shape*1, *SE_shape*2, параметр *SE_gain* усиления спектра, параметр спектра для кодека 4 кбит/с *SE_shape*3.. *SE_shape*6 заменяются соответствующими параметрами предыдущего фрейма. Кроме того, для управления выходной громкостью речи, параметры амплитуд гармоник *LPC Am* [0 … 127] усиливаются, как показано в (Б.1). В этом уравнении, $Am_{(org)}[i]$ вычисляется из последнего фрейма, свободного от ошибок.

$$Am[i] = mute * Am_{(org)}[i] \text{ для } i=0..\ 127 \quad (Б.1)$$

Если предыдущий фрейм не содержит речи, и текущее состояние - *state*=7, (Б.1) заменяется на (Б.2).

$$Am[i] = 0{,}6 * mute * Am_{(org)}[i] \text{ для } i=0..\ 127 \quad (Б.2)$$

Как описано прежде, *SE_shape*1 и *SE_shape*2 индивидуально защищены 1 битом *CRC*. В *state*=0 или 7, когда ошибки *CRC* этих классов обнаружены в то же самое время, когда квантованные значения амплитуд Am_{qnt} [1.. 44] ослабляются, как показано в (Б.3).

$$Am[i] = s[i] * Am_{qnt(org)}[i] \text{ для } i=1..\ 44 \quad (Б.3)$$

s[*i*] - коэффициент ослабления.

Таблица Б.16 – Коэффициент ослабления для *s*[0.. 44]

i	1	2	3	4	5	6	7...44
s[*i*]	0,10	0,25	0,40	0,55	0,70	0,85	1,00

На 4 кбит/с *SE_shape*4, *SE_shape*5, и *SE_shape*6 проверяются *CRC* как биты Класса II. Когда ошибка *CRC* обнаружена, спектральный параметр расширения не используется.

Б.3.4.1.4 Замена и контроль усиления над «неголосовыми» параметрами

В *state*=1.. 6, стохастический параметр усиления кодовой книги *VX_gain*1[0] и *VX_gain*1[1] заменяются на *VX_gain*1[1] от последнего фрейма, свободного от ошибок. Также стохастические параметры усиления кодовой книги для кодека 4 кбит/с *VX_gain*2[0]... *VX_gain*2[3] заменяются *VX_gain*2[3] от последнего фрейма, свободного от ошибок.

Стохастические параметры *VX_shape*1[0], *VX_shape*1[1] и стохастический параметр кодека 4 кбит/с генерируются из произвольных индексных значений.

Кроме того, для управления выходой громкостью речи остаточный сигнал *LPC res* [0 … 159] усиливается, как показано в (Б.4). В этом уравнении, $res_{(org)}[i]$ вычисляется по стохастическим параметрам кодовой книги.

$$res[i] = mute * res_{(org)}[i] \text{ для } i=0..\ 159 \quad (Б.4)$$

Б.3.4.1.5 Переходы состояний маскирования фреймов (рисунок Б.2)

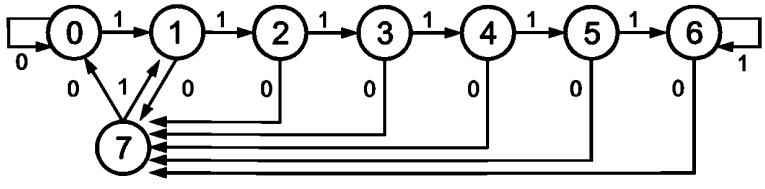


Рисунок Б.2 – Переходы состояний маскирования фреймов

Библиография

[1] ИСО/МЭК 14496-3:2009 Информационные технологии. Кодирование аудиовизуальных объектов. Часть 3. Аудио (ИСО/МЭК14496–3:2009 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 *А.Б. Заварзина*
Корректор *В.Г. Смолин*
Компьютерная верстка *Д.Е. Першин*

Сдано в набор 20.12.2013. Подписано в печать 5.02.2014. Формат 60x841/8. Гарнитура Ариал.
Усл. печ. л. 11,16. Уч.-изд. л. 9,15. Тираж 56 экз. Зак. 2138.

Набрано в ООО «Академиздат».
gost.e-ecolog.ru gost.e-ecolog.ru
Издано и отпечатано
во ФГУП «СТАНДАРТИНФОРМ», 123995 Москва, Гранатный пер., 4.
gost.e-ecolog.ru
gost.e-ecolog.ru gost.e-ecolog.ru