ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 53556.11— 2014

Звуковое вещание цифровое

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

Часть III

(MPEG-4 AUDIO)

Аудиокодирование без потерь

(ISO/IEC 14496-3:2009, NEQ)

Издание официальное

Москва
Стандартинформ
2020

gost.e-ecolog.ru

Предисловие

1 РАЗРАБОТАН Техническим комитетом по стандартизации ТК 480 «Связь»

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 480 «Связь»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 17 марта 2014 г. № 148-ст

4 Настоящий стандарт разработан с учетом основных нормативных положений международного стандарта ИСО/МЭК 14496-3:2009 «Информационные технологии. Кодирование аудиовизуальных объектов. Часть 3. Аудио» (ISO/IEC 14496-3:2009 «Information technology — Coding of audio-visual objects — Part 3: Audio», NEQ) [1]

5 ВВЕДЕН ВПЕРВЫЕ

6 ПЕРЕИЗДАНИЕ. Июль 2020 г.

*Правила применения настоящего стандарта установлены в статье 26 Федерального закона от 29 июня 2015 г. № 162-ФЗ «О стандартизации в Российской Федерации». 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 ежемесячного информационного указателя «Национальные стандарты». Соответствующая информация, уведомление и тексты размещаются также в информационной системе общего пользования — на официальном сайте Федерального агентства по техническому регулированию и метрологии в сети Интернет (*gost.e-ecolog.ru

© Стандартинформ, оформление, 2015, 2020

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 Технический обзор . . . 1
2.1 Структура кодера и декодера . . . 1
2.2 Расширения с плавающей точкой . . . 1
3 Термины и определения . . . 2
3.1 Определения . . . 2
3.2 Мнемоника . . . 2
3.3 Типы данных . . . 2
3.4 Замечания по реализации . . . 3
4 Синтаксис . . . 3
4.1 Конфигурация декодера . . . 3
4.2 Полезные нагрузки потока битов . . . 4
4.3 Полезные нагрузки для данных с плавающей точкой . . . 8
5 Семантика . . . 10
5.1 Общая семантика . . . 10
5.2 Семантика для данных с плавающей точкой . . . 15
6 Инструменты *ALS* . . . 16
6.1 Краткий обзор . . . 16
6.2 Переключение блоков . . . 19
6.3 Прогноз . . . 20
6.4 Долгосрочный прогноз (*LTP*) . . . 26
6.5 Прогнозирующее устройство *RLS-LMS* . . . 28
6.6 Кодированный остаток . . . 39
6.7 Объединенное кодирование пар каналов . . . 56
6.8 Многоканальное кодирование (*MCC*) . . . 56
6.9 Расширение для данных с плавающей точкой . . . 59
Библиография . . . 65

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Звуковое вещание цифровое

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

Часть III

(MPEG-4 AUDIO)

Аудиокодирование без потерь

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

Дата введения — 2015—01—01

1 Область применения

Этот стандарт описывает алгоритм кодирования аудиосигналов без потерь: аудиокодирование без потерь *MPEG*-4 (*ALS*).

MPEG-4 *ALS* являются схемой сжатия данных цифрового аудио без потерь, то есть декодируемые данные являются разрядно-идентичной реконструкцией исходных входных данных. Входные сигналы могут быть целочисленными данными *PCM* от 8 до 32-разрядной длины слова или 32-разрядными данными *IEEE* с плавающей точкой. *MPEG*-4 *ALS* обеспечивает широкий диапазон гибкости с точки зрения компромисса сжатия — сложности, поскольку комбинация нескольких инструментов позволяет определить уровень компрессии с различными степенями сложности.

2 Технический обзор

2.1 Структура кодера и декодера

Входные аудиоданные делятся на фреймы (кадры). В пределах фрейма каждый канал может быть дополнительно разделен на блоки аудиосэмплов (выборок аудио) для дальнейшей обработки. Для каждого блока вычисляется ошибка прогноза, используя краткосрочный прогноз и дополнительно долгосрочный прогноз (*LTP*). Межканальная избыточность может быть удалена объединенным кодированием каналов, используя либо дифференциальное кодирование пар каналов, либо многоканальное кодирование (*MCC*). Остающаяся ошибка прогноза является в итоге кодированной энтропией.

Кодер генерирует информацию о потоке битов, позволяющую произвольный доступ с промежутками в несколько фреймов. Кодер может также обеспечить контрольную сумму *CRC*, которую может использовать декодер, чтобы проверить декодируемые данные. Декодер применяет инверсные операции кодера в обратном порядке. Его вывод является разрядно-идентичной версией исходных входных аудиоданных.

2.2 Расширения с плавающей точкой

В дополнение к целочисленным аудиосигналам *MPEG*-4 *ALS* также поддерживает сжатие аудиосигналов без потерь в 32-разрядном формате с плавающей точкой *IEEE*. Последовательность с плавающей точкой моделируется суммой целочисленной последовательности, умноженной на константу (*ACF*: *Approximate Common Factor*), и остаточной последовательностью. Целочисленная последовательность сжимается, используя основные инструменты *ALS* для целочисленных данных, в то время как остаточная последовательность отдельно сжимается замаскированным инструментом *Lempel-Ziv*.

Издание официальное

3 Термины и определения

3.1 Определения

В стандарте используются следующие определения и сокращения.

Frame	Сегмент аудиосигнала (содержащий все каналы).
Block	Сегмент одного звукового канала.
Sub-block	Часть блока, которая использует те же параметры кодирования энтропии.
Random Access Frame	Фрейм, который может декодироваться без декодирования предыдущих фреймов.
Residual	Ошибка прогноза, то есть истинный минус предсказанного сигнала.
Predictor/Prediction Filter	Линейный фильтр *FIR*, который вычисляет оценку входного сигнала, используя предыдущие выборки.
Prediction order	Порядок фильтра прогноза (число коэффициентов прогнозирующего устройства).
LPC coefficients	Коэффициенты фильтра прогноза прямой формы.
Parcor coefficients	Представление *parcor* коэффициентов прогнозирующего устройства.
Quantized coefficients	Квантованные коэффициенты *parcor*.
LTP	Долгосрочный прогноз.
Rice code	Также известно как код *Golomb-Rice*. В этом документе используется краткая форма.
BGMC	Блоковый код *Block Gilbert-Moore Code* (также известен как код *Elias-Shannon-Fano*).
CRC	Контроль циклически избыточным кодом.
LPC	Кодирование с линейным предсказанием.
PCM	Импульсно-кодовая модуляция.
Mantissa	Дробная часть данных с плавающей точкой.
Exponent	Экспоненциальная часть данных с плавающей точкой.
ACFC	Кодирование с приближенным общим множителем.
Masked-LZ	Замаскированное кодирование *Lempel-Ziv*.
MCC	Многоканальное кодирование.
MSB	Старший значащий бит.
LSB	Младший значащий бит.

3.2 Мнемоника

uimsbf	Целое число без знака, старший значащий бит сначала.
simsbf	Целое число со знаком, старший значащий бит сначала.
bslbf	Битовая строка, левый бит сначала, где «левый» является порядком, в котором пишутся биты.
EEE32	32-разрядные данные с плавающей точкой *IEEE* (4 байта), старший значащий бит сначала.

Мнемонический код Райса и *BGMC* указывают, что используются кодовые комбинации переменной длины.

3.3 Типы данных

В разделах псевдокода используются следующие типы данных:

INT64	64-разрядное целое число со знаком (дополнение до двух).
long	32-разрядное целое число со знаком (дополнение до двух).
short	16-разрядное целое число со знаком (дополнение до двух).
x. y	Дробное представление с фиксированной точкой со знаком, где *x* является числом битов слева от точки в двоичном числе и *y* является числом битов справа от точки в двоичном числе (представление знака дополнения до двух). 64-разрядное целое число со знаком (дополнение до двух).

Если перед типом данных добавляется «без знака», то типом является тип без знака вместо типа со знаком.

3.4 Замечания по реализации

В этом документе есть несколько логарифмических арифметических вычислений в форме '*ceil*(log2(…))', которые определяют целочисленное значение, которое описывает число необходимых битов для определенного параметра.

Чтобы избежать непоследовательных результатов и реализовать *ceil()* и log2(), не должны использоваться никакие функции с плавающей точкой. Чтобы получить математически корректные целочисленные результаты, должны использоваться реализации с фиксированной точкой (например, применение сдвигов битов).

4 Синтаксис

4.1 Конфигурация декодера

Таблица 1 — Синтаксис *ALSSpecificConfig*

Синтаксис	Количество битов	Мнемоника
ALSSpecificConfig()		
{		
als_id;	32	*uimsbf*
samp_freq;	32	*uimsbf*
samples;	32	*uimsbf*
channels;	16	*uimsbf*
file_type;	3	*uimsbf*
resolution;	3	*uimsbf*
floating;	1	*uimsbf*
msb_first;	1	*uimsbf*
frame_length;	16	*uimsbf*
random_access;	8	*uimsbf*
ra_flag;	2	*uimsbf*
adapt_order;	1	*uimsbf*
coef_table;	2	*uimsbf*
long_term_prediction;	1	*uimsbf*
max_order;	10	*uimsbf*
block_switching;	2	*uimsbf*
bgmc_mode;	1	*uimsbf*
sb_part;	1	*uimsbf*
joint_stereo;	1	*uimsbf*
mc_coding;	1	*uimsbf*
chan_config;	1	*uimsbf*
chan_sort;	1	*uimsbf*
crc_enabled;	1	*uimsbf*
RLSLMS	1	*uimsbf*
(reserved)	5	
aux_data_enabled;	1	*uimsbf*
if (chan_config)		
{chan_config_info;	16	*uimsbf*
}		
if (chan_sort) {		
for (c = 0; c <= channels; c++)		
chan_pos[c];	1..16	*uimsbf*
}		
byte_align;	0..7	*bslbf*
header_size;	32	*uimsbf*
trailer_size;	32	*uimsbf*
orig_header[];	*header_size* * 8	*bslbf*
orig_trailer[];	*trailer_size* * 8	*bslbf*
if (crc_enabled) {		
crc;	32	*uimsbf*
}		

Окончание таблицы 1

Синтаксис	Количество битов	Мнемоника
if ((ra_flag == 2) && (random_access > 0)) { *for (f = 0; f < ((samples-1) / (frame_length+1)) + 1; f++) {* *ra_unit_size[f]* *}* *}* *if (aux_data_enabled) {* *aux_size;* *aux_data[];* *}* *}*	 32 32 *aux_size* * 8	 *uimsbf* *uimsbf* *bslbf*
Примечание — «*byte_align*» обозначает выравнивание байта последующих данных относительно начала *ALSSpecificConfig* ().		

4.2 Полезные нагрузки потока битов

Таблица 2 — Синтаксис высокоуровневой полезной нагрузки (*frame_data*)

Синтаксис	Количество битов	Мнемоника
frame_data() *{* *if ((ra_flag == 1) && (frame_id % random_access == 0))* *{* *ra_unit_size* *}* *if (mc_coding && joint_stereo) {* *js_switch;* *byte_align;* *}* *if (!mc_coding || js_switch) {* *for (c = 0; c <= channels; c++) {* *if (block_switching) {* *bs_info;* *}* *if (independent_bs) {* *for (b = 0; b < blocks; b++) {* *block_data(c);* *}* *}* *else{* *for (b = 0; b < blocks; b++) { block_data(c); block_data(c+1);* *}* *c++;* *}* *}* *}* *else{* *if (block_switching) {* *bs_info;* *}* *for (b = 0; b < blocks; b++) {* *for (c = 0; c <= channels; c++) { block_data(c); channel_data(c);* *}* *}* *}* *if (floating)* *{* *num_bytes_diff_float;* *diff_float_data();* *}* *}*	 32 1 8, 16, 32 <

Окончание таблицы 2

Примечание — Если *joint_stereo* выключено или если *c* является последним каналом, *independent_bs* является истиной по умолчанию. Если *joint_stereo* включено, *independent_bs* является ложью по умолчанию, но если *block switching* также включено, флаг *independent_bs* сообщается, как первый бит поля *bs info* пары каналов. Поле *frame id* указывает последовательный номер фрейма, начиная с 0 для первого фрейма.

Таблица 3 — Синтаксис *block_data*

Синтаксис		Количество битов	Мнемоника
block_data()			
{			
block_type;		1	*uimsbf*
if (block_type == 0) {			
const_block;		1	*uimsbf*
js_block;			
(reserved)		1	*uimsbf*
if (const_block == 1) {		5	
if (resolution == 0) {	*// 8 bits*	8	*simsbf*
const_val;			
}			
else if (resolution == 1) {	*// 16 bits*	16	*simsbf*
const_val;			
}			
else if (resolution == 2 || floating == 1) {	*// 24 bits*	24	*simsbf*
const_val;			
}			
else {	*// 32 bits*	32	*simsbf*
const_val;			
}			
}			
}			
else {			
js_block;		1	*uimsbf*
if ((bgmc_mode == 0) && (sb_part == 0) {			
sub_blocks = 1;			
}			
else if ((bgmc_mode == 1) && (sb_part ==1) {		2	*uimsbf*
ec_sub;			
sub_blocks = 1 << ec_sub;			
}			
else {			
ec_sub;		1	*uimsbf*
sub_blocks = (ec_sub == 1) ? 4 : 1;			
}			
if (bgmc_mode == 0) {			
for (k = 0; k < sub_blocks; k++) {		Изменяется	*uimsbf*
s[k];			
}			
}			
else {		Изменяется	*Rice code*
for (k = 0; k < sub_blocks; k++) {			
s[k],sx[k];			
}			
}			
sb_length = block_length / sub_blocks;		1	*Rice code*
shift_lsbs;			
if (shift_lsbs == 1) {		4	*uimsbf*
shift_pos;			
}			

Продолжение таблицы 3

Синтаксис	Количество битов	Мнемоника
if (!RLSLMS) { *if (adapt_order == 1) {* *opt_order;* *}* *else {* *opt_order = max_order;* *}* *for (p = 0; p < opt_order; p++) {* *quant_cof[p];* *}* *}*	1..10 Изменяется 1	*uimsbf* *Rice code* *uimsbf*
if (long_term_prediction) { *LTPenable;* *if (LTPenable) {* *for (i = -2; i <= 2; i++) {* *LTPgain[i];* *}* *LTPlag;* *}* *}*	 Изменяется 8, 9, 10	 *Rice code* *uimsbf*
start = 0; *if (random_access_block) {* *if (opt_order > 0) {* *smp_val[0];* *}* *if (opt_order > 1) {* *res[1];* *}* *if (opt_order > 2) {* *res[2];* *}* *if (opt_order < 3) {* *start = opt_order;* *}* *else {* *start = 3;* *}* *}*	 Изменяется Изменяется Изменяется	 *Rice code* *Rice code* *Rice code*
if (bgmc_mode) { *for (n = start; n < sb_length; n++) {* *msb[n];* *}* *for (k=1; k < sub_blocks; k++) {* *for (n = k * sb_length; n < (k+1) * sb_length; n++)* *{* *msb[n];* *}* *}* *for (n = start; n < sb_length; n++) {* *if (msb[n] != tail_code) {* *lsb[n];* *}* *else {* *tail[n];* *}* *}* *for (k=1; k < sub_blocks; k++) {* *for (n = k * sb_length; n < (k+1) * sb_length; n++)* *{*	 Изменяется Изменяется Изменяется Изменяется	 *BGMC* *BGMC* *uimsbf* *Rice code*

Окончание таблицы 3

Синтаксис	Количество битов	Мнемоника
if (msb[n] != tail_code) { *lsb[n];* *}* *else {* *tail[n];* *}* *}* *}* *}* *else* *{* *for (n = start; n < block_length; n++) {* *res[n];* *}* *}* *}* *if (RLSLMS) {* *RLSLMS_extension_data()* *}* *if (!mc_coding || js_switch) {* *byte_align;* *}* *}*	Изменяется Изменяется Изменяется 0..7	*uimsbf* *Rice code* *Rice code* *bslbf*
Примечание — *random_access_block* является истиной, если текущий блок принадлежит фрейму произвольного доступа (*frame_id* % *random_access* = = 0) и является первым (или только) блоком канала в этом фрейме.		

Таблица 4 — Синтаксис *channel_data*

Синтаксис	Количество битов	Мнемоника
channel_data(c) *{* *for(;;) {* *stop_flag;* *if (stop_flag == 1) {* *break;* *}* *master_channel_index;* *if (c != master_channel_index) {* *time_difference_flag* *if (time_difference_flag == 0) {* *weighting_factor [0]* *weighting_factor [1]* *weighting_factor [2]* *}* *else {* *weighting_factor [0]* *weighting_factor [1]* *weighting_factor [2]* *weighting_factor [3]* *weighting_factor [4]* *weighting_factor [5]* *time_difference_sign* *time_difference_index* *}* *}* *}* *byte_align;* *}*	1 1..16 1 Изменяется Изменяется Изменяется Изменяется Изменяется Изменяется Изменяется Изменяется Изменяется 1 5,6,7 0..7	*uimsbf* *uimsbf* *uimsbf* *Rice code* *Rice code* *Rice code* *Rice code* *Rice code* *Rice code* *Rice code* *Rice code* *Rice code* *uimsbf* *uimsbf* *bslbf*

Таблица 5 — Синтаксис *RLSLMS_extension_data*

Синтаксис	Количество битов	Мнемоника
RLSLMS_extension()		
{		
mono_block	1	*uimsbf*
ext_mode	1	*uimsbf*
if (ext_mode) {		
extension_bits	3	*uimsbf*
if (extension_bits&0x01) {		
RLS_order	4	*uimsbf*
LMS_stage	3	*uimsbf*
for(i=0; i<LMS_stage;i++){		
LMS_order[i]	5	*uimsbf*
}		
}		
if (extension_bits&0x02) {		
if (RLS_order) {		
RLS_lambda	10	*uimsbf*
if (RA)		
RLS_lambda_ra	10	*uimsbf*
}		
}		
if (extension_bits&04) {		
for(i=0; i<LMS_stage;i++) {		
LMS_mu[i]	5	*uimsbf*
}		
LMS_stepsize	3	*uimsbf*
}		
}		
}		

4.3 Полезные нагрузки для данных с плавающей точкой

Таблица 6 — Синтаксис *diff_float_data*

Синтаксис	Количество битов	Мнемоника
diff_float_data()		
{		
use_acf;	1	*uimsbf*
if (random_access_block) {		
if (c=0; c <= channels; c++) {		
last_acf_mantissa[c] = 0;		
last_shift_value[c] = 0;		
FlushDict();		
}		
for (c = 0; c <= channels; c++) {		
if (use_acf == 1) {		
}		
acf_flag[c];	1	*uimsbf*
if (acf_flag[c] == 1) {		
acf_mantissa[c];	23	*uimsbf*
last_acf_mantissa[c] = acf_mantissa[c];		
}		
else {		
acf_mantissa[c] = last_acf_mantissa[c];		
}		
}		
else {		
acf_mantissa[c] = last_acf_mantissa[c] = 0;		
}		

Окончание таблицы 6

Синтаксис	Количество битов	Мнемоника
highest_byte[c];	2	*uimsbf*
shift_amp[c];	1	*uimsbf*
partA_flag[c];	1	*uimsbf*
if (shift_amp[c] == 1) {		
shift_value[c];	8	*uimsbf*
last_shift_value[c] = shift_value[c];		
}		
else {		
shift_value[c] = last_shift_value[c];		
}		
diff_mantissa();		
byte_align;	0..7	*bslbf*
}		
}		

Примечание — «*byte_align*» обозначает дополнение битов до следующей границы байта. «*FlushDict*()» является функцией, которая очищает и инициализирует словарь и переменные модуля распаковки *Masked-LZ*.

Таблица 7 — Синтаксис *diff_mantissa*

Синтаксис	Количество битов	Мнемоника
diff_mantissa()		
{		
if (partA_flag[c] != 0) {		
compressed_flag[c];	1	*uimsbf*
if (compressed_flag[c] == 0) {		
for (n = 0; n < frame_length; n++) {		
if (int_zero[c][n]) {		
float_data[c][n];	32	*IEEE32*
}		
}		
}		
else {		
nchars = 0;		
for (n = 0; n < frame_length; n++) {		
if (int_zero[c][n])		
nchars += 4;		
}		
Masked_LZ_decompression(nchars);		
}		
}		
if (highest_byte[c] != 0) {		
compressed_flag[c];	1	*uimsbf*
if (compressed_flag[c][n] == 0) {		
for (n = 0; n < frame_length; n++) {		
if (!int_zero[c][n]) {		
mantissa[c][n];	*nbits[c][n]*	*uimsbf*
}		
}		
}		
else {		
nchars = 0;		
for (n = 0; n < frame_length; n++) {		
if (!int_zero[c][n]) {		
nchars += (int)nbits[c][n]/8;		
if ((nbits[c][n] % 8) > 0)		
nchars++;		
}		
}		

Окончание таблицы 7

Синтаксис	Количество битов	Мнемоника
Masked_LZ_decompression(nchars); *}* *}* *}*		
Примечание — *«int_zero»* является истиной, если соответствующее округленное целое число равно 0. *«nbits [c] [n]»* является необходимой длиной слова для различия мантиссы.		

Таблица 8 — Синтаксис *Masked_LZ_decompression*

Синтаксис	Количество битов	Мнемоника
Masked_LZ_decompression (nchars) *{* *for (dec_chars = 0; dec_chars < nchars;) {* *string_code;* *}* *}*	9..14	*uimsbf*
Примечание — *«nchars»* является числом символов, которые должны декодироваться.		

5 Семантика

5.1 Общая семантика

5.1.1 *ALSSpecificConfig*

ALSSpecificConfig содержит общие данные конфигурации. Дополнительно могут быть встроены заголовок и концевик исходного аудиофайла, чтобы восстановить эту информацию в дополнение к фактическим аудиоданным. Синтаксис *ALSSpecificConfig* определяется в таблице 1, ее элементы описываются в таблице 9.

Таблица 9 — Элементы *ALSSpecificConfig*

Поле	Количество битов	Описание/Значения
als_id	32	Идентификатор *ALS* Фиксированное значение = 1095521024 = 0×414C5300 (*Hex*)
samp_freq	32	Частота дискретизации, Гц
samples	32	Число выборок (на канал) Если *samples* = 0×*FFFFFFFF* (*Hex*), число выборок не определяется
channels	16	Число каналов — 1 (0 — моно, 1 — стерео, …)
file_type	3	000 — неизвестно/неотработанный файл; 001 — волновой файл; 010 — файл *aiff;* 011 — файл *bwf*; (другие значения зарезервированы)
resolution	3	000 — 8-бит; 001 — 16-бит; 010 — 24-бит; 011 — 32-бит; (другие значения зарезервированы)
floating	1	1 — 32-битовый с плавающей точкой *IEEE*, 0 — целый
msb_first	1	Исходный порядок байта входных аудиоданных: 0 — младший значащий байт первый (прямой порядок) 1 — старший значащий байт первый (обратный порядок) если *resolution* = 0 (8-битовые данные), *msb_first* = 0 указывает данные без знака (0…255), тогда как *msb_first* = 1 указывает данные со знаком (– 128…127)

Продолжение таблицы 9

Поле	Количество битов	Описание/Значения
Frame_length	16	Длина фрейма — 1 (например, *frame_length* = 0×1*FFF* сигнализирует о длине *N* = 8192)
random_access	8	Расстояние между фреймами *RA frames* (во фреймах, 0…255). Если *RA* не используется, величина равна 0. Если каждый фрейм является фреймом *RA*, величина равна 1
ra_flag	2	Указывает, где хранится размер блоков случайного доступа (*ra_unit_size*) 00 — не хранится
adapt_order	1	Адаптивный порядок: 1 — вкл, 0 — выкл
coef_table	2	Табличный индекс (00, 01 или 10 параметров кода *Rice* для кодирования энтропии коэффициентов прогнозатора, 11 — кодирование энтропии отсутствует)
long_term_prediction	1	Долгосрочный прогноз (*LTP*): 1 — вкл, 0 — выкл
max_order	10	Максимальный порядок прогноза (0..1023)
bgmc_mode	1	Режим *BGMC*: 1 — вкл, 0 — выкл (кодирование *Rice* только)
joint_stereo	1	*Joint Stereo*: 1 — вкл, 0 — выкл, если каналы — 0 (моно), *joint_stereo* — 0
block_switching	2	Число уровней коммутации блока: 00 — коммутация блока отсутствует 01 — вплоть до 3 уровней 10 — 4 уровня 11 — 5 уровней
mc_coding	1	Расширенное межканальное кодирование: 1 — вкл, 0 — выкл, если каналы — 0 (моно), *mc_coding* — 0
sb_part	1	Расчленение субблока для кодирования энтропии остатка, если *bgmc_mode* = 0: 0 — расчленение отсутствует, бит *ec_sub bit* в *block_data* отсутствует 1 — расчленение 1:4, один бит *ec_sub bit* в *block_data*, если *bgmc_mode* = 1: 0 — расчленение 1:4, один бит *ec_sub* в *block_data* 1 — расчленение 1:2, 4:8, два бита *ec_sub* в *block_data*
chan_config	1	Указывает, что поле *chan_config_info* присутствует
chan_sort	1	Перестановка каналов: 1 — вкл, 0 — выкл, если каналы — 0 (моно), *chan_sort* — 0
crc_enabled	1	Указывает, что поле *crc* присутствует
RLSLMS	1	Использование предсказателя *RLS-LMS*: 1 — вкл, 0 — выкл
Зарезервировано	5	
aux_data_enabled	1	Указывает, что вспомогательные данные присутствуют (поля *aux_size* и *aux_data*)
chan_config_info	16	Отображение каналов на местоположение громкоговорителя. Каждый бит указывает, существует ли канал для определенного местоположения

Окончание таблицы 9

Поле	Количество битов	Описание/Значения
chan_pos[]	(*channels*+1)* *ChBits*	Если перестановка каналов включена (*chan_sort* — 1), имеет место исходное расположение каналов. Число битов на канал *ChBits* = *ceil*[log2(*channels*+1)] = 1..16, где *channels*+1 является числом каналов
header_size	32	Размер заголовка исходного аудиофайла в байтах Если *header_size* = 0x*FFFFFFFF* (*Hex*), поле *orig_header[]* отсутствует, но исходный заголовок может быть сохранен в другом месте, например в метаданных файла *MPEG*-4
trailer_size	32	Размер концевика исходного аудиофайла в байтах Если *trailer_size* = 0x*FFFFFFFF* (*Hex*), поле *orig_trailer[]* отсутствует, но исходный заголовок может быть сохранен в другом месте, например в метаданных файла *MPEG*-4
orig_header[]	*header_size**8	Заголовок исходного аудиофайла
orig_trailer[]	*trailer_size**8	Концевик исходного аудиофайла
crc	32	32-битовый контрольный код *CCITT*-32 *CRC* байтов исходных аудиоданных (полином: $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$)
ra_unit_size[]	*#frames**32	Расстояние (в байтах) между фреймами произвольного доступа, то есть размеры блоков произвольного доступа, где количество фреймов *#frames*=((*samples*-1)/(*frame_length*+1))+1. В *ALSSpecificConfig*() это поле появляется, только когда *ra_flag* = 2
aux_size	32	Размер поля *aux_data* в байтах Если *aux_size* = 0x*FFFFFFFF* (*Hex*), поле *aux_data[]* отсутствует, но вспомогательные данные могут быть сохранены в другом месте, например в метаданных файла *MPEG*-4
aux_data	*aux_size**8	Вспомогательные данные (для декодирования не требуется)

5.1.2 *frame_data*

Это высокоуровневая полезная нагрузка *ALS*. Если *random_access* > 0, число полезных нагрузок, отображенных в одном устройстве доступа, равняется значению *random_access* (1 … 255). В этом случае размер каждого блока доступа может быть сохранен в *ra_unit_size*. Если *random_access* = 0, все полезные нагрузки отображаются в тот же самый блок доступа.

Поле *bs_info* содержит информацию о переключении блока для канала или пары каналов. Синтаксис *frame_data* определяется в таблице 2, его элементы описываются в таблице 10.

Таблица 10 — Элементы *frame_data*

Поле	Количество битов	Описание/Значения
ra_unit_size	32	Расстояние (в байтах) до следущего фрейма произвольного доступа, то есть размер блока случайного доступа. В *frame_data*() это поле появляется, только когда *ra_flag* = 1
bs_info	8, 16, 32	Информация о переключении блока. Если *block_switching* = 0, поле *bs_info* не передается, иначе количество битов зависит от величины *block_switching*: *block_switching* = 1: 8 битов; *block_switching* = 2: 16 битов; *block_switching* = 3: 32 бита
js_switch	1	Если *js_switch* = 1, выбирается *Joint Stereo* (разница каналов), даже если задействовано *MCC* (*mc_coding*)
num_bytes_diff_float	32	Присутствует, только когда *floating* = 1: Число байтов для *diff_float_data*

5.1.3 *block_data*

Блочные данные определяют тип блока (нормальный, постоянный, тишина) и в основном содержат индексы кода, порядок прогнозирующего устройства, коэффициенты прогнозирующего устройства и кодированные остаточные значения. Синтаксис *block data* определяется в таблице 3, его элементы описываются в таблице 11.

Таблица 11 — Элементы *block_data*

Поле	Количество битов	Описание/Значения
block_type	1	1 — нормальный блок 0 — нуль/постоянный блок
const_block	1	Только если *block type* = 0: 1 — постоянный блок 0 — нулевой блок (тишина)
js_block	1	Блок содержит разностный сигнал объединенного стерео
const_val	8, 16, 24, 32	Постоянная величина выборки этого блока
ec_sub	0..2	Число сублоков для кодирования энтропии. Количество битов = *bgmc_mode* + *sb_part* Если количество битов — 0: 1 субблок. Если количество битов — 1: 0 — 1 субблок; 1 — 4 субблока. Если количество битов = 2: 00 — 1 субблок; 01 — 2 субблока; 10 — 4 субблока; 11 — 8 субблоков
s[],sx[]	Изменяется	До 8 индексов кода *Rice* (*s*) или *BGMC* (*s,sx*) для кодирования энтропии субблоков (число дается *ec sub*). Разностные значения кодируются по *Rice*
shift_lsbs	1	Указывает, что все исходные значения входных выборок блока были сдвинуты вправо перед дальнейшей обработкой, чтобы удалить пустые *LSBs*
quant_cof[]	Изменяется	Кодированные по *Rice* квантованные коэффициенты
LTPenable	1	Переключение *LTP*: 1 — вкл, 0 — выкл
LTPgain[]	Изменяется	Кодированные по *Rice* величины усиления (*5-tap*)
res[]	Изменяется	Кодированные по *Rice* остаточные величины
shift_pos	4	Число позиций — 1, на которое были сдвинуты вправо значения выборок этого блока: 0000 — 1 позиция … 1111 — 16 позиций
opt_order	1..10	Порядок прогнозатора для этого блока (длиной *NB*): *#Bits* = min{*ceil*(log2(*max_order*+1)); max[*ceil*(log2((*NB* >> 3)-1)),1]}. Количество битов ограничено как максимальным порядком (*max_order*), так и длиной блока *NB*
LTPlag	8, 9, 10	Величины задержки *LTP* *Freg* < 96000, диапазон — 0..255, битов — 8 96000 ≤ *Freg* < 192000, диапазон — 0..511, битов — 9 *Freg* ≥ 192000, диапазон — 0..1023, битов — 10
smp_val[0]	Изменяется	Кодированная по *Rice* величина выборки в начале блока произвольного доступа

Окончание таблицы 11

Поле	Количество битов	Описание/Значения
msb[]	Изменяется	*BGMC*-кодированные старшие значащие биты остатков. Для остатков вне центральной области передается спектральное "*tail_code*"
lsb[]	Изменяется	Прямо передаваемые младшие значащие биты остатков
tail[]	Изменяется	Кодированные по *Rice* остаточные величины вне центральной области (концевики)

5.1.4 *channel_data*

Синтаксис *channel_data* определяется в таблице 4, его элементы описываются в таблице 12.

Таблица 12 — Элементы *channel_data*

Поле	Количество битов	Описание/Значения
stop_flag	1	0 — Продолжение описания межканальных связей 1 — Прекращение описания
master_channel_index	1..16	Индекс мастер-канала. *#Bits = ceil*[log2(*channels*+1)], где *channels*+1 является количеством каналов
time_difference_flag	1	0 — Трехотводный без задержки разновременности 1 — Шестиотводный с задержкой разновременности
weighting factor	Изменяется	Индексы коэффициента межканального взвешивания
time_difference_sign	1	0 — Положительный, 1 — отрицательный. "Положительный" означает, что эталонный канал задержан относительно канала кодирования
time_difference_value	5, 6, 7	Канал кодирования. Абсолютное значение задержки разницы времени *Freq* < 96000, диапазон — 3..34, битов — 5 96000 ≤ *Freq* < 192000, диапазон — 3..66, битов — 6 *Freq* ≥ 192000, диапазон — 3..130, битов — 7

5.1.5 *RLSLMS_extension_data*

Синтаксис *RLSLMS_extension_data* определяется в таблице 5, его элементы описываются в таблице 13.

Таблица 13 — Элементы *RLSLMS_extension_data*

Поле	Количество битов	Описание/Значения
mono_block	1	*mono_frame* — 0: *CPE* кодировано с *joint-stereo RLS* *mono_frame* — 1: *CPE* кодировано с моно *RLS*
ext_mode	1	Параметры прогнозатора *RLS-LMS* обновляются в блоке расширения. 1 = = блок расширения 0 = = блок без расширения
extension_bits	3	Тип параметров *RLS-LMS*, переносимых в блоке расширения: *xtension*&01 — порядки прогнозаторов *RLS-LMS;* *extension*&02 — *RLS_lambda* и *RLS_lambda_ra;* *extension*&04 — *LMS_mu* и *LMS_stepsize*
RLS_order	4	Порядок прогнозатора *RLS*
LMS_stage	3	Количество прогнозаторов *LMS* в каскаде
LMS_order[]	5**LMS_stage*	Порядок прогнозатора *LMS*

Окончание таблицы 13

Поле	Количество битов	Описание/Значения
RLS lambda	10	Лямбда-параметр прогнозатора *RLS*
RLS_lambda_ra	10	Лямбда-параметр прогнозатора *RLS* для фрейма произвольного доступа
LMS mu[]	5*LMS stage	Параметр прогнозатора *LMS* — длина шага *NLMS*
LMS stepsize	3	Параметр линейного объединителя — длина шага *Sign Sign LMS*

5.2 Семантика для данных с плавающей точкой

5.2.1 *diff_float_data*

Синтаксис *diff_float_data* определяется в таблице 6, его элементы описываются в таблице 14.

Таблица 14 — Элементы *diff_float_data*

Поле	Количество битов	Описание/Значения
use_acf	1	1 — *acf_flag*[*c*] присутствует 0 — *acf flag*[*c*] отсутствует
acf_flag[c]	1	1 — *acf mantissa*[*c*] присутствует 0 — *acf_mantissa*[*c*] отсутствует
acf_mantissa[c]	23	Полные данные о мантиссе общего множителя
highest_byte[c]	2	Старшие ненулевые байты мантиссы во фрейме
partA_flag[c]	1	1 — в *PartA* существуют выборки 0 — в *PartA* не существует выборок или все они нулевые
shift_amp[c]	1	1 — *shift_value*[*c*] присутствует 0 — *shift value*[*c*] отсутствует
shift_value[c]	8	Величина сдвига: эта величина добавляется к экспоненте величин с плавающей точкой канала *c* после преобразования декодированного целого в величины с плавающей точкой и перед добавлением целого и данных разницы

5.2.2 *diff_mantissa*

Синтаксис *diff mantissa* определяется в таблице 7, его элементы описываются в таблице 15.

Таблица 15 — Элементы *diff_mantissa*

Поле	Количество битов	Описание/Значения
int_zero[c][n]	Изменяется	*int zero* для *n*-й выборки и *c*-го канала устанавливается в округленное целое равное "0". Эта величина не является синтаксическим элементом, но может быть определена из ассоциированного целого значения, которое доступно как в кодере, так и в декодере
mantissa[c][n]	*nbits[c][n]*	Полные данные о мантиссе
compresed_flag[c]	1	1 — Выборки упакованы 2 — Выборки распакованы
nchars	Изменяется	Количество символов для декодирования
float_data[c][n]	32	32-битовая величина с плавающей точкой *IEEE*
nbits[c][n]		Эта величина не является синтаксическим элементом. Она может быть определена из целой величины *acf mantissa*[*c*] и *highest byte*[*c*].

5.2.3 *Masked_LZ_decompression*

Синтаксис *Masked_LZ_decompression* определяется в таблице 8, его элементы описываются в таблице 16.

Таблица 16 — Элементы *Masked LZ decompression*

Поле	Количество битов	Описание/Значения
string_code	code_bits	Кодекс индекса словаря
code_bits	Изменяется	*code_bits* изменяется от 9 до 15 битов в зависимости от количества записей, хранящихся в словаре

6 Инструменты *ALS*

В стандартах кодирования *MPEG* с наибольшими потерями подробно определяется только декодер. Однако схема кодирования без потерь обычно требует спецификации некоторых (но не всех) частей кодера. Так как процесс кодирования должен быть совершенно обратимым без потери информации, несколько частей кодера и декодера должны быть определены детерминированным способом.

6.1 Краткий обзор

6.1.1 Структура потока битов

Каждый фрейм (*frame_data*) состоит из $B = 1 \ldots 32$ блоков выборок (*block_data*) для каждого канала. Помимо общей информация о блоке [например, блок тишины, блок различия объединенного стерео (*joint stereo*) и т. д.] каждый блок обычно содержит индексы кода, порядок прогнозирующего устройства *K*, коэффициенты прогнозирующего устройства и остаточные значения, кодированные по Райсу или *BGMC*. Если используется объединенное кодирование между парами каналов, часть блока идентична для обоих каналов, и блоки сохраняются перемежающимся способом. Иначе эта часть блока для каждого канала независима.

Если вводятся данные с плавающей точкой, дополнительные элементы потока битов для дифференциальных значений мантиссы вставляются после потока битов каждого целочисленного фрейма.

6.1.2 Декодирование *ALSSpecificConfig*

ALSSpecificConfig содержит информацию об исходных данных (например, «*samp freq*», «*channels*», «*resolution*»), а также глобальные параметры, которые не изменяются от фрейма к фрейму (например, «*frame_length*», «*max_order*»). Наиболее важные параметры описываются далее.

Идентификатор *ALS* — это поле должно содержать значение 1095521024 = 0×414C5300 (шестнадцатеричное). Используя побайтовое чтение, первые три байта эквивалентны кодам *ASCII* для *'ALS'*.

Частота дискретизации — частота дискретизации исходных аудиоданных сохраняется, например для прямого воспроизведения сжатого файла.

Выборки — общее количество аудиовыборок на канал.

Число каналов — 1 (моно), 2 (стерео) или более (многоканальный).

Разрешение — 8-битовое, 16-битовое, 24-битовое или 32-битовое. Если разрешение исходных аудиоданных находится в промежутке (например, 20-битовое), для представления выборок используется более высокое разрешение.

Плавающая точка — указывает формат аудиоданных. Если этот флаг установлен, аудиоданные представлены в 32-битовом формате с плавающей точкой *IEEE*, иначе аудиоданные являются целочисленными.

Порядок байтов — указывает на порядок байтов исходного аудиофайла, либо старший значащий байт сначала (например, *aiff*), либо младший значащий байт сначала (например, *wave*).

Длина фрейма — число выборок в каждом фрейме (на канал).

Произвольный доступ — расстояние (во фреймах) между теми фреймами, которые могут декодироваться независимо от предыдущих фреймов (фреймы произвольного доступа). Перед каждым фреймом произвольного доступа есть поле «*ra_unit_size*», которое определяет это расстояние в байтах.

Адаптивный порядок — у каждого блока может быть индивидуальный порядок прогнозирующего устройства.

Таблица коэффициентов — таблица, содержащая параметры, которые используются для кодирования энтропии коэффициентов прогнозирующего устройства.

Максимальный порядок — максимальный порядок фильтра прогноза. Если «*adapt_order*» выключается, этот порядок используется для всех блоков.

Переключение блоков — вместо одного блока на канал может быть до 32 более коротких блоков. Если переключение блоков не используется, размер блока идентичен длине фрейма.

Режим *BGMC* — указывает, что для остатка прогноза используются коды *BGMC*. Если этот флаг устанавливается в 0, для остатка прогноза используются более простые коды Райса.

Раздел подблока — раздел подблока для кодирования энтропии остатка.

Объединенное стерео — в каждом блоке может быть закодирован разностный сигнал вместо сигнала левого или правого канала (или один из двух каналов пары каналов соответственно).

Многоканальное кодирование — расширенное межканальное кодирование.

Вид канала — перегруппировка каналов, используемая для создания выделенных пар каналов.

Позиции канала — исходные позиции канала, используемые только если включается *channel_sort*.

Размер заголовка — размер заголовка исходного аудиофайла в байтах.

Размер концевика — размер оконечной неаудиоинформации в исходном аудиофайле в байтах.

Исходный заголовок — встроенный заголовок исходного аудиофайла.

Исходный концевик — встроенная оконечная часть исходного аудиофайла.

CRC — контрольная сумма циклической избыточности (*CCITT*-32) байтов исходных аудиоданных (то есть в их оригинальном порядке, включая чередование каналов).

6.1.3 Число фреймов

Число фреймов для декодирования зависит от фактической длины фрейма (*N* = *frame_length* + 1) и числа выборок. Это может быть определено следующим образом:

```
N = frame_length + 1.
frames = samples / N;
remainder = samples % N;
if (remainder)
{
    frames++;
        N_last = remainder;
}
    else
        N_last = N;
```

Если число выборок не является кратным длине фрейма *N*, длина последнего фрейма соответственно уменьшается (*N last* = остаток).

Если значение выборок является (шестнадцатеричным) 0×*FFFFFFFF*, число выборок не определяется. Если полезная нагрузка *ALS* сохраняется, используя формат файла *MPEG*-4, число выборок может быть получено из метаданных файла.

Если число выборок недоступно, число фреймов не определено, и считается, что у всех фреймов одна и та же длина *N*. В этом случае размеры блоков произвольного доступа не должны сохраняться в *ALSSpecifcConfig* (то есть должны использоваться только *ra flag* = 0 или *ra flag* = 1), так как число блоков произвольного доступа тоже не определено.

6.1.4 Объединенное кодирование каналов

Чтобы использовать дублирование между каналами, кодер может применить простой подход, состоящий из пар каналов и одиночных каналов. Два канала пары каналов могут быть закодированы, используя кодирование различия, тогда как одиночные каналы кодируются независимо.

Об общем использовании объединенного кодирования сообщается флагом *joint stereo* в заголовке *ALS*. Если *joint stereo* выключен, каждый канал является одиночным каналом и кодируется независимо от других каналов. Если *joint_stereo* включен, в каждом случае два соседних канала расцениваются как пара каналов. Если число каналов нечетно, то один канал остается одиночным.

Определение пар каналов не означает, что должно использоваться объединенное кодирование. Если *joint_stereo* будет установлен, то декодер будет считать комбинации двух каналов парами каналов, даже если кодер фактически никогда не использовал объединенное кодирование. В этом случае декодер просто не будет обнаруживать *block data* с установленным флагом *js_block*.

Если выбирается *MCC* (многоканальное кодирование), то декодируется информация об отношении между каналами (ведущий или ведомый). Декодируемые остаточные значения ведомого канала изменяются добавлением значений ведущего канала, умноженных на декодированные коэффициенты взвешивания. Другие процессы реконструкции для сигналов всех каналов, которые включают декодирование параметров, декодирование остатка прогноза, фильтрацию синтеза долгосрочного и краткосрочного прогноза, идентичны процессам для декодирования независимых каналов. Инструменты кодирования двух объединенных каналов, объединенное стерео и *MCC*, могут быть адаптивно выбраны на пофреймовой основе.

6.1.5 Конфигурация и перестановка каналов

Поле *chan_config_info* определяет отображение канал — динамик, указывая, существует ли канал для определенного местоположения. Существующие каналы должны быть расположены в предопределенном порядке (таблица 17). Если определенный канал присутствует, устанавливается соответствующий бит в поле *chan_config_info*.

Таблица 17 — Конфигурация канала

Положение динамика	Сокращение	Позиция бита в *chan_config_info*
Слева	*L*	1
Справа	*R*	2
Слева сзади	*Lr*	3
Справа сзади	*Rr*	4
Левый боковой	*Ls*	5
Правый боковой	*Rs*	6
В центре	*C*	7
Центральный сзади/Окружение	*S*	8
Низкочастотные эффекты	*LFE*	9
Левый *Downmix*	*L*0	10
Правый *Downmix*	*R*0	11
Моно *Downmix*	*M*	12
(Зарезервировано)		13—16

Решение о том, какие каналы группируются, может быть принято автоматически кодером или вручную пользователем. Если конфигурация каналов указывается в исходном файле, кодер может сделать подходящую перестановку. Если формат файла не имеет конфигурации каналов по умолчанию, но пользователь знает отображение канал — динамик в этом конкретном случае, он может сообщить кодеру, как сгруппировать каналы.

Декодер должен инвертировать возможную перестановку каналов (флаг *chan_sort*), присваивая каждому каналу его исходную позицию, которая хранится в *chan_pos* [].

6.1.6 Декодирование фреймов

Фрейм составляет высокоуровневую полезную нагрузку (*frame_data*), то есть основной блок аудиоданных (см. в таблице 2 о синтаксисе и таблице 10 о семантике). Если используется переключение блоков, каждый канал фрейма может быть подразделен на 32 блока. Иначе блок состоит из всех выборок канала фрейма.

6.1.7 Декодирование блоков

Структура *block_data* () содержит информацию об одном блоке (то есть сегмент аудиоданных из одного канала). Она определяет, является ли блок «нормальным» блоком (то есть содержащим закодированные аудиовыборки), постоянным блоком (все аудиовыборки являются одними и теми же) или блоком тишины (все аудиовыборки являются нулем). Кроме того, поле «*joint_stereo*» указывает, содержит ли блок разностный сигнал (правый канал минус левый). Либо левый, либо правый канал может

быть заменен этим разностным сигналом. Структура в случае блочного переключения также содержит информацию, когда длина блока может быть короче, чем длина фрейма.

Для «нормальных» блоков блочные данные включают:

- индексы кода;
- порядок прогнозирующего устройства *K*;
- квантованные и закодированные коэффициенты прогнозирующего устройства (или параметры прогнозирующего устройства *RLS-LMS* в случае режима *RLSLMS*);
- параметры *LTP* в случае режима *LTP*;
- кодированные остаточные значения по Райсу или *BGMC*.

Если блок дополнительно подразделяется на подблоки для кодирования энтропии (обозначенный как *ec sub*), параметры кода *s* и *sx* передаются для каждого подблока.

В случае адаптивного порядка прогнозирующего устройства (*adapt_order*) указывается порядок для блока (*opt order*). Имеется также флаг (*shift lsbs*), определяющий, есть ли у всех аудиовыборок в текущем блоке некоторые *LSB*, которые являются перманентно нулевыми. В этом случае число пустых *LSB* дается в другом поле (*shift_pos*). Это означает, что кодер сместил все значения выборок вправо на *shift pos*+1 позиций до выполнения прогноза. Таким образом, декодер должен сместить выходные значения выборок влево на *shift pos*+1 позиций после того, как был применен инверсный фильтр прогноза. Если процесс прогноза использует выборки из предыдущего блока, смещенная версия этих выборок должна использоваться в качестве ввода как в фильтр прогноза, так и в инверсный фильтр прогноза (то есть как в кодере, так и в декодере), даже если *LSB* не являются нулем в предыдущем блоке. Это необходимо, чтобы выровнять амплитудный диапазон входных выборок прогнозирующего устройства с выборками, которые будут спрогнозированы.

6.1.8 Чередование

Наиболее несжатые форматы аудиофайла хранят два канала стереосигнала как последовательность чередующихся выборок (*L*1, *R*1, *L*2, *R*2, *L*3, *R*3, …). Для многоканальных данных с *M* каналами каждый шаг выборки включает *M* чередующихся выборок. Так как кодер создает блоки выборок для каждого канала, декодируемые выборки всех каналов, вероятно, придется снова чередовать, прежде чем записать их в выходной аудиофайл.

6.2 Переключение блоков

Если включено *block switching*, каждый канал фрейма может быть иерархически подразделен на блоки вплоть до 32 блоков.

Произвольные комбинации блоков с $N_B = N$, $N/2$, $N/4$, $N/8$, $N/16$ и $N/32$ возможны в пределах фрейма до тех пор, пока каждый блок получается из подразделения вышестоящего блока двойной длины.

О фактическом разделении сообщается в дополнительном поле *bs_info*, длина которого зависит от числа уровней переключения блоков (таблица 18). Фрейм длиной *N* должен быть разделен на 2^{levels} без остатка, чтобы получить целочисленные длины блока N_B.

Таблица 18 — Уровни переключения блоков

Максимальное количество уровней	Минимальное N_B	Количествово байтов для *bs_info*
0	*N*	0
1	*N*/2	1
2	*N*/4	1
3	*N*/8	1
4	*N*/16	2
5	*N*/32	4

Поле *bs info* состоит из 4 байтов, где отображение битов относительно уровней от 1 до 5 имеет вид [(0) 1223333 44444444 55555555 55555555]. Первый бит используется, только чтобы сигнализировать о независимом переключении блоков.

В каждом фрейме передаются поля *bs info* для всех пар каналов и всех одиночных каналов, задействуя переключение независимого блока для различных каналов. В то время как длина фрейма

идентична для всех каналов, переключение блока может быть выполнено индивидуально для каждого канала. Если используется различное кодирование, оба канала пары каналов должны быть переключены синхронно, но другие пары каналов все еще могут использовать различное переключение блоков.

Если два канала пары каналов не коррелированы друг с другом, то кодирование различия не будет окупаться, и не будет никакой необходимости переключать оба канала синхронно. Вместо этого имеет смысл переключать каналы независимо.

Как правило поле *bs info* будет для каждой пары каналов и во фрейме одиночного канала, то есть два канала пары каналов переключаются синхронно. Если они переключаются независимо, первый бит *bs_info* устанавливается в 1, и информация применяется к первому каналу пары каналов. В этом случае становится необходимым другое поле *bs info* для второго канала.

6.3 Прогноз

В этой главе описывается прямая адаптивнаая схема прогноза.

Кодер состоит из нескольких стандартных блоков. Буфер хранит один блок входных выборок, и соответствующий набор коэффициентов *parcor* вычисляется для каждого блока. Число коэффициентов, то есть порядок прогнозирующего устройства, может быть также адаптировано. Квантованные значения *parcor* являются кодированными для передачи энтропией и преобразованными в коэффициенты *LPC* для фильтра прогноза, который вычисляет остаток прогноза.

Декодер значительно менее сложен, чем кодер, так как никакая адаптация не должна выполняться. Переданные значения *parcor* декодируются, преобразовываются в коэффициенты *LPC* и используются инверсным фильтром прогноза для вычисления сигнала реконструкции без потерь. Вычислительная работа декодера зависит от порядка прогнозирующего устройства, выбранного кодером.

Если порядок прогноза *K* выбирается адаптивно (*adapt_order* = 1), число битов, используемых для сигнализации о фактическом порядке (*opt_order* = *K*) в каждом блоке, ограничивается в зависимости как от глобального максимального порядка (*max order*), так и в зависимости от размера блока N_B.

Bits = min*{ceil*[log2*(max_order*+1)*]*, max*[ceil*([log2*(*N_B>>3*))*, 1*]}*.

Максимальный порядок K_{max} = min(2^{Bits} – 1, *max_order*) ограничивается в зависимости от значения *max order* и длины блока (таблица 19).

Таблица 19 — Примеры максимальных порядков прогноза в зависимости от длины блока и *max_order*

N_B	*max_order* = 1023		*max_order* = 100	
	Количество битов для *opt_order*	K_{max}	Количество битов для *opt_order*	K_{max}
8192	10	1023	7	100
4096	9	511	7	100
2048	8	255	7	100
1024	7	127	7	100
512	6	63	6	63
256	5	31	5	31
128	4	15	4	15
64	3	7	3	7
32	2	3	2	3
16	1	1	1	1

Базовый (краткосрочный) прогноз может быть объединен с долгосрочным прогнозом (*LTP*).

6.3.1 Коэффициенты прогнозирующего устройства

Передача коэффициентов фильтра прогноза выполняется путем использования коэффициентов *parcor* γ_k, *k* = 1…*K* (где *K* является порядком фильтра), которые могут быть получены при использовании алгоритма Левинсона — Дарбина.

6.3.1.1 Квантование и кодирование коэффициентов *parcor*

Первые два коэффициента *parcor* (γ_1 и γ_2, соответственно) квантуются при использовании следующих функций компандирования:

$$\alpha_1 = \left[64\left(-1+\sqrt{2}\sqrt{\gamma_1+1}\right)\right];$$

$$\alpha_2 = \left[64\left(-1+\sqrt{2}\sqrt{-\gamma_2+1}\right)\right].$$

Остающиеся коэффициенты квантуются, используя простые 7-битовые универсальные квантователи:

$$\alpha_k = \left[64\gamma_k\right], \text{ где } k > 2.$$

Во всех случаях получающиеся квантованные значения α_k ограничиваются диапазоном [– 64, 63].

Передача квантованных коэффициентов α_k выполняется созданием остаточных значений $\delta_k = \alpha_k - offset_k$, которые кодируются с использованием кодов Райса. Соответствующие смещения и параметры кодов Райса, используемых в этом процессе, могут быть выбраны из одного из наборов в таблице 20, где табличный индекс (*coef_table*) указывается в *ALSSpecificConfig*. Если *coef_table* = 11, то кодирование энтропии не применяется и квантованные коэффициенты передаются с 7 битами каждый. В этом случае смещение всегда – 64, чтобы получить значения без знака $\delta_k = \alpha_k + 64$, которые ограничиваются диапазоном [0, 127].

Таблица 20 — Параметры кода Райса, используемые для кодирования коэффициентов *parcor*

Номер коэффициента	*coef_table* = 00		*coef_table* = 01		*coef_table* = 10	
	Смещение	Параметр Райса	Смещение	Параметр Райса	Смещение	Параметр Райса
1	– 52	4	– 58	3	– 59	3
2	– 29	5	– 42	4	– 45	5
3	– 31	4	– 46	4	– 50	4
4	19	4	37	5	38	4
5	– 16	4	– 36	4	– 39	4
6	12	3	29	4	32	4
7	– 7	3	– 29	4	– 30	4
8	9	3	25	4	25	3
9	– 5	3	– 23	4	– 23	3
10	6	3	20	4	20	3
11	– 4	3	– 17	4	– 20	3
12	3	3	16	4	16	3
13	– 3	2	– 12	4	– 13	3
14	3	2	12	3	10	3
15	– 2	2	– 10	4	– 7	3
16	3	2	7	3	3	3
17	– 1	2	– 4	4	0	3

Окончание таблицы 20

Номер коэффициента	*coef_table* = 00		*coef_table* = 01		*coef_table* = 10	
	Смещение	Параметр Райса	Смещение	Параметр Райса	Смещение	Параметр Райса
18	2	2	3	3	– 1	3
19	– 1	2	– 1	3	2	3
20	2	2	1	3	– 1	2
$2k - 1$, $10 < k < 65$	0	2	0	2	0	2
$2k$, $10 < k < 64$	1	2	1	2	1	2
$k > 127$	0	1	0	1	0	1

6.3.1.2 Реконструкция коэффициентов *parcor*

Кодированные по Райсу остаточные значения δ_k декодируются и объединяются со смещениями (таблица 20), чтобы произвести квантованные индексы коэффициентов *parcor* α_k:

$$\alpha_k = \delta_k + offset_k .$$

Затем производится реконструкция первых двух коэффициентов, используя:

$$par_1 = \left\lfloor \gamma_1 2^Q \right\rfloor = \Gamma(\alpha_1);$$

$$par_2 = \left\lfloor \gamma_2 2^Q \right\rfloor = \Gamma(\alpha_2),$$

где 2^Q представляет постоянный масштабный коэффициент ($Q = 20$), требующийся для целочисленного представления восстановленных коэффициентов, и $\Gamma(i)$ является отображением, описанным таблице 21.

Таблица 21 — Индексы i и соответствующие масштабные значения *parcor* $\Gamma(i)$ для $i = -64 \ldots 63$

i	$\Gamma(i)$	i	$\Gamma(i)$	i	$\Gamma(i)$	i	$\Gamma(i)$
– 64	– 1048544	– 50	– 1021664	– 36	– 944608	– 22	– 817376
– 63	– 1048288	– 49	– 1017824	– 35	– 937184	– 21	– 806368
– 62	– 1047776	– 48	– 1013728	– 34	– 929504	– 20	– 795104
– 61	– 1047008	– 47	– 1009376	– 33	– 921568	– 19	– 783584
– 60	– 1045984	– 46	– 1004768	– 32	– 913376	– 18	– 771808
– 59	– 1044704	– 45	– 999904	– 31	– 904928	– 17	– 759776
– 58	– 1043168	– 44	– 994784	– 30	– 896224	– 16	– 747488
– 57	– 1041376	– 43	– 989408	– 29	– 887264	– 15	– 734944
– 56	– 1039328	– 42	– 983776	– 28	– 878048	– 14	– 722144
– 55	– 1037024	– 41	– 977888	– 27	– 868576	– 13	– 709088
– 54	– 1034464	– 40	– 971744	– 26	– 858848	– 12	– 695776
– 53	– 1031648	– 39	– 965344	– 25	– 848864	– 11	– 682208
– 52	– 1028576	– 38	– 958688	– 24	– 838624	– 10	– 668384
– 51	– 1025248	– 37	– 951776	– 23	– 828128	– 9	– 654304

Окончание таблицы 21

i	$\Gamma(i)$	i	$\Gamma(i)$	i	$\Gamma(i)$	i	$\Gamma(i)$
– 8	– 639968	10	– 338144	28	46624	46	514336
– 7	– 625376	11	– 318944	29	70432	47	542752
– 6	– 610528	12	– 299488	30	94496	48	571424
– 5	– 595424	13	– 279776	31	118816	49	600352
– 4	– 580064	14	– 259808	32	143392	50	629536
– 3	– 564448	15	– 239584	33	168224	51	658976
– 2	– 548576	16	– 219104	34	193312	52	688672
– 1	– 532448	17	– 198368	35	218656	53	718624
0	– 516064	18	– 177376	36	244256	54	748832
1	– 499424	19	– 156128	37	270112	55	779296
2	– 482528	20	– 134624	38	296224	56	810016
3	– 465376	21	– 112864	39	322592	57	840992
4	– 447968	22	– 90848	40	349216	58	872224
5	– 430304	23	– 68576	41	376096	59	903712
6	– 412384	24	– 46048	42	403232	60	935456
7	– 394208	25	– 23264	43	430624	61	967456
8	– 375776	26	– 224	44	458272	62	999712
9	– 357088	27	23072	45	486176	63	1032224

Реконструкция коэффициентов 3-го и более высоких порядков производится, используя формулу

$$par_1 = \left[\gamma_1 2^Q\right] = \alpha_k 2^{Q-6} + 2^{Q-7};\ (k > 2).$$

6.3.1.3 Преобразование восстановленных коэффициентов *parcor* в коэффициенты прямого фильтра

Масштабированные коэффициенты *parcor* затем преобразовываются в коэффициенты *LPC*, используя следующий алгоритм:

```
short m, i, K, Q = 20;
long *cof, *par, corr = 1 << (Q – 1);
INT64 temp, temp2;
for (m = 1; m <= K; m++)
{
    for (i = 1; i <= m/2; i++)
    {
        temp = cof[i] + ((((INT64)par[m] * cof[m-i]) + corr) >> Q);
        if ((temp > LONG_MAX) || (temp < LONG_MIN))          // Overflow: use different
            coefficients return(1);
        temp2 = cof[m-i] + ((((INT64)par[m] * cof[i]) + corr) >> Q);
        if ((temp2 > LONG_MAX) || (temp2 < LONG_MIN)) // Overflow: use different
            coefficients return(1);
        cof[m-i] = (long)temp2;
        cof[i] = (long)temp;
    }
    cof[m] = par[m];
}
```

Здесь *LONG_MAX* = $2^{31} - 1$ и *LONG_MIN* = $-(2^{31})$. Получающиеся коэффициенты *LPC cof* также масштабируются с 2^{20}. Масштабирование будет учтено во время процесса фильтрации.

6.3.2 Фильтр прогноза

Вычисление спрогнозированного сигнала должно быть выполнено детерминированным способом, чтобы включить идентичное вычисление и в кодере, и в декодере, поэтому невозможно использовать коэффициенты с плавающей запятой. Вместо этого используют целочисленное представление с увеличением масштаба. Так как коэффициенты увеличиваются с множителем $2^Q = 2^{20}$, предсказанный сигнал также будет увеличен тем же самым множителем. Таким образом, в конце процесса фильтрации масштаб каждой выборки предсказанного сигнала должен быть уменьшен.

6.3.2.1 Кодер

Следующий алгоритм описывает вычисление остатка *d* для входного сигнала *x*, длины блока *N*, порядка прогнозирующего устройства *K* и коэффициентов *LPC cof*:

```
short n, N, k, K, Q = 20;
long *x, *d, *cof, corr = 1 << (Q – 1);
INT64 y;
for (n = 0; n < N; n++)
{
    y = corr;
    for (k = 1; k <= K; k++)
        y += (INT64)cof[k-1] * x[n-k];
    d[n] = x[n] + (long)(y >> Q);
}
```

Чтобы предсказать первую выборку текущего блока, прогнозирующее устройство использует последние *K* выборок из предыдущего блока.

Если текущий блок (или подблок) является первым блоком канала во фрейме с произвольным доступом, никакие выборки из предыдущего блока не могут использоваться. В этом случае используется прогноз с прогрессивным порядком, где масштабированные коэффициенты *parcor* прогрессивно конвертируются в коэффициенты *LPC cof* в фильтре прогноза. В каждой рекурсии вычисляются величина текущего остатка $d(n)$ и новый набор $n+1$ коэффициентов *LPC* (первый цикл). После того как вычисляются первые значения остатка *K* и все *K* коэффициенты, используется прогноз полного порядка (второй цикл). Индексы для *par* и *cof* в этой реализации начинаются с 1.

```
short m, n, N, i, k, K, Q = 20;
long *x, *d, *cof, corr = 1 << (Q – 1);
INT64 y, temp, temp2;
for (n = 0; n < min(K, N); n++)
{
    y = corr;
    for (k = 1; k <= n; k++)
        y += (INT64)cof[k] * x[n-k];
    d[n] = x[n] + (long)(y >> Q);
    m = n + 1;
    for (i = 1; i <= m/2; i++)
    {
        temp = cof[i] + ((((INT64)par[m] * cof[m-i]) + corr) >> Q);
        if ((temp > LONG_MAX) || (temp < LONG_MIN))          // Overflow: use different
            coefficients return(1);
        temp2 = cof[m-i] + ((((INT64)par[m] * cof[i]) + corr) >> Q);
        if ((temp2 > LONG_MAX) || (temp2 < LONG_MIN)) // Overflow: use different
            coefficients return(1);
        cof[m-i] = (long)temp2,
        cof[i] = (long)temp;
    }
    cof[m] = par[m];
}
for (n = K; n < N; n++)
{
```

```
        y = corr;
        for (k = 1; k <= K; k++)
            y += (INT64)cof[k] * x[n-k];
        d[n] = x[n] + (long)(y >> Q);
    }
```

Только первая выборка *x*(0) передается непосредственно, используя код Райса с *s* = *resolution* – 4 (то есть *s* = 12 для 16-битового и *s* = 20 для 24-битового). Следующие две величины остатка *d*(1) и *d*(2) кодируются кодами Райса, которые связываются с первым параметром Райса блока *s*[0]. В зависимости от кодера энтропии остающиеся величины остатка от *d*(3) до *d*(*K*) являются или кодированными по Райсу с *s*[0], или *BGMC*-кодированными с *s*[0] и *sx*[0]. Сводка всех кодов дается в таблице 22.

Таблица 22 — Параметры кода для различных позиций выборки

Выборка/Остаток	Параметр кода
x(0)	Разрешающая способность — 4
d(1)	*s*[0] + 3
d(2)	*s*[0] + 1
d(3) ... *d*(K)	*s*[0] (*BGMC*: *sx*[0])

6.3.2.2 Декодер

Алгоритм для вычисления исходного сигнала в декодере почти идентичен алгоритму кодера, за исключением последней инструкции:

```
short n, N, k, K, Q = 20;
long *x, *d, corr = 1 << (Q – 1);
INT64 y;
for (n = 0; n < N; n++)
{
        y = corr;
        for (k = 1; k <= K; k++)
            y += (INT64)cof[k-1] * x[n-k];
        x[n] = d[n] – (long)(y >> Q);
}
```

В случае произвольного доступа используется прогноз с прогрессивным порядком. Алгоритм для вычисления также почти идентичен алгоритму кодера, за исключением двух строк, где вычисляется *x*. Индексы для *par* и *cof* начинаются с 1.

```
short m, n, N, i, k, K, Q = 20;
long *x, *d, *cof, corr = 1 << (Q – 1);
INT64 y, temp, temp2;
for (n = 0; n < min(K, N); n++)
{
        y = corr;
        for (k = 1; k <= n; k++)
            y += (INT64)cof[k] * x[n-k];
        x[n] = d[n] – (long)(y >> Q);
        m = n + 1;
        for (i = 1; i <= m/2; i++)
        {
            temp = cof[i] + ((((INT64)par[m] * cof[m-i]) + corr) >> Q);
            temp2 = cof[m-i] + ((((INT64)par[m] * cof[i]) + corr) >> Q);
            cof[m-i] = (long)temp2;
            cof[i] = (long)temp;
        }
```

```
        cof[m] = par[m];
    }
    for (n = K; n < N; n++)
    {
        y = corr;
        for (k = 1; k <= K; k++)
            y += (INT64)cof[k] * x[n-k];
        x[n] = d[n] – (long)(y >> Q);
    }
```

Если кодером использовалось кодирование объединенных каналов, декодируемый сигнал *x* может быть разностным сигналом. В этом случае должна быть произведена дальнейшая обработка, чтобы получить исходный сигнал.

6.4 Долгосрочный прогноз (*LTP*)

6.4.1 Усиление и задержка *LTP*

Если *LTPenable* включено, декодируются величины усиления $\rho(l)$ и значение задержки τ. Величины усиления $\rho(i)$ восстанавливаются из кодированных по Райсу индексов, перечисленных в таблицах 23, 24 и 25.

Таблица 23 — Значения реконструкции и код Райса для усиления $\rho(0)$

Величины усиления $\rho(0)$ *128	Индекс	Префикс	Субкод
0	0	0	00
8	1	0	01
16	2	0	10
24	3	0	11
32	4	10	00
40	5	10	01
48	6	10	10
56	7	10	11
64	8	110	00
70	9	110	01
76	10	110	10
82	11	110	11
88	12	1110	00
92	13	1110	01
96	14	1110	10
100	15	1110	11

Таблица 24 — Значения реконструкции и код Райса для усиления $\rho(\pm1)$

Величины усиления $\rho(\pm1)$*128	Индекс	Префикс	Субкод
0	0	0	00
– 8	1	0	01
8	2	0	10
– 16	3	0	11

Окончание таблицы 24

Величины усиления ρ(±1)*128	Индекс	Префикс	Субкод
16	4	10	00
– 24	5	10	01
24	6	10	10
– 32	7	10	11
32	8	110	00
– 40	9	110	01
40	10	110	10
– 48	11	110	11
48	12	1110	00
– 56	13	1110	01
56	14	1110	10
– 64	15	1110	11
64	16	11110	00

Таблица 25 — Значения реконструкции и код Райса для усиления ρ(±2)

Величины усиления ρ(±2)*128	Индекс	Префикс	Субкод
0	0	0	0
– 8	1	0	1
8	2	10	0
– 16	3	10	1
16	4	110	0
– 24	5	110	1
24	6	1110	0
– 32	7	1110	1
32	8	11110	0
– 40	9	11110	1
40	10	111110	0
– 48	11	111110	1
48	12	1111110	0
– 56	13	1111110	1
56	14	11111110	0
– 64	15	11111110	1
64	16	111111110	0

Переданное значение относительной задержки является фактическим значением, исключая стартовое значение задержки. Оно непосредственно кодируется естественным двоичным кодированием от 8 до 10 битов в зависимости от частот дискретизации. Фактические значения задержки показаны в таблице 26, где «*optP*» обозначает фактический порядок на краткосрочный прогноз.

Таблица 26 — Поисковый диапазон задержки τ

Поисковый диапазон τ (i)	Начало	Конец
Freq < 96 кГц	max(*optP*,*3*)+1	max(*optP*,*3*)+25
Freq >= 96 кГц	max(*optP*,*3*)+1	max(*optP*,*3*)+51
Freq >= 192 кГц	max(*optP*,*3*)+1	max(*optP*,*3*)+102

6.4.2 Процедура синтеза *LTP*

Если декодируются параметры задержки и усиления, выполняется операция следующей рекурсивной фильтрации $d(i) = d(i) + \sum_{j=-2}^{2} \rho(j) d(i - \tau + j)$.

Для того чтобы обеспечить совершенную реконструкцию, процесс должен быть строго определен. Псевдокод для этого фильтра в декодере следующий:

```
INT64 u;
for (smpl=0 ;smpl<end; smpl++)
{
    for (u=1<<6, tap=-2; tap<=2; tap++)
    {
        u += (INT64)LTPgain[tap]*d[smpl-lag+tap];
    }
    d[smpl] += (long)(u>>7);
}
```

Здесь *d* является остаточным сигналом (который позже подается в фильтр синтеза). *LTPgain* является величиной усиления *ρ*(*i*)*128. Задержка имеет величину τ.

Для простой комбинации с адаптивным переключением блока все значения остаточного сигнала и *d*(*i*) в предыдущем блоке равны «0». Процесс фильтрации синтеза имеет псевдокод для процесса фильтрации анализа в кодере. Этот процесс также должен быть нормативным с целью совершенной реконструкции. В этом псевдокоде различие между кодером и декодером появляется в последней строке. Ввод и вывод в декодере являются общими, в то время как в кодере они отличаются.

```
INT64 u,
for (smpl=0 ;smpl<end; smpl++)
{
    for (u=1<<6, tap=-2; tap<=2; tap++)
    {
        u += (INT64)LTPgain[tap]*d[smpl-lag+tap];
    }
    dout[smpl] = d[smpl]-(long)(u>>7);
}
```

Здесь *d* является остатком краткосрочного прогноза и *dout* является остатком *LTP*.

6.5 Прогнозирующее устройство *RLS-LMS*

Обратно адаптивный прогноз использует адаптивное прогнозирующее устройство *RLS-LMS*.

В кодере прогнозирующее устройство *RLS-LMS* генерирует оценку текущей входной аудиовыборки при использовании прошлых выборок. Эта оценка вычитается из текущей выборки, чтобы сгенерировать остаток, который затем кодируется кодером энтропии, чтобы сформировать поток битов *ALS*.

В декодере выполняется обратный процесс. Декодер энтропии декодирует поток битов *ALS* в остаток, который затем добавляется к оценке прогнозирующего устройства *RLS-LMS*, чтобы регенерировать исходную аудиовыборку.

Прогнозирующее устройство *RLS-LMS* состоит из каскада прогнозирующих устройств *DPCM*, *RLS* прогнозирующего устройства и серии прогнозирующих устройств *LMS*. Входные выборки последовательно проходят через цепочку прогнозирующих устройств. Остатки из одного прогнозирующего устройства служат вводом в следующее прогнозирующее устройство. Оценки из прогнозирующих устройств в цепочке взвешиваются и складываются линейным объединителем, чтобы сгенерировать заключительную оценку текущей входной выборки.

Прогнозирующее устройство *RLS-LMS* может быть включено/выключено установкой *RLSLMS* в *ALSSpecificConfig* () в 1/0 соответственно.

6.5.1 Прогнозирующее устройство *DPCM*

Ввод: исходная аудиовыборка $x(n)$.

Остаток: $e_1(n)$ как вход в прогнозирующее устройство *RLS*.

Оценка: $y_1(n)$ как вход в линейный объединитель.

Прогнозирующее устройство *DPCM* является первым прогнозирующим устройством в цепочке прогнозирующих устройств *RLS-LMS*. Это простое прогнозирующее устройство первого порядка с набором коэффициентов, то есть предыдущая входная выборка используется в качестве оценки текущей входной выборки. Это иллюстрируется следующим образом:

$$y_1(n) = x(n-1),$$

где $y_1(n)$ является оценкой прогнозирующим устройством *DPCM* и $x(n-1)$ является предыдущей входной выборкой.

6.5.2 Прогнозирующее устройство *RLS*

Ввод: остаток прогнозирующего устройства *DPCM* $e_1(n)$.

Остаток: $e_2(n)$ как вход в прогнозирующее устройство *LMS*.

Оценка: $y_2(n)$ как вход в линейный объединитель.

Прогнозирующее устройство *RLS* является вторым прогнозирующим устройством в цепочке прогнозирующих устройств *RLS-LMS*. Алгоритм *RLS* используется, чтобы адаптировать весовые коэффициенты прогнозирующего устройства. Алгоритм инициализируется, устанавливая матрицы инверсной автокорреляции $M \times M$ в предопределенное значение P следующим образом:

$$P(0) = \delta I,$$

где δ является маленьким положительным числом, I является матрицей $M \times M$ и P является порядком прогнозирующего устройства *RLS*.

Вектор веса прогнозирующего устройства *RLS*, определенный как

$$w_{RLS}(n) = \left[w_{RLS,1}(n), w_{RLS,2}(n), ..., w_{RLS,M}(n)\right]^T,$$

инициализируется как

$$w_{rls}(0) = 0.$$

Для каждого индекса n, $n = 1, 2, \ldots$, выполняются следующие вычисления:

$$v(n) = P(n-1)e_1(n),$$

где $e_1(n)$ является входным вектором прогнозирующего устройства *RLS*, определенным как

$$e_1(n) = \left[e_1(n-1), e_1(n-2), ..., e_1(n-M)\right]^T$$

и

$$m = \begin{cases} \dfrac{1}{e_1^T(n)v(n)} & \text{если } e_1^T(n)v(n) \neq 0 \\ 1 & \text{в других случаях,} \end{cases}$$

$$k(n) = mv(n),$$

$$y_2(n) = w_{RLS}^T(n-1)e_1(n),$$

$$e_2(n) = e_1(n) - y_2(n),$$

$$w_{RLS}(n) = w_{RLS}(n-1) + k(n)e_2(n),$$

$$P(n) = Tri\left\{\lambda^{-1}\left(P(n-1) - k(n)v^T(n)\right)\right\},$$

где $k(n)$ является вектором усиления $M \times 1$, λ является коэффициентом забываемости, который имеет положительное значение, немного меньшее, чем 1, T является символом перемещения и $Tri\{*\}$ обозначает операцию по вычислению более низкой треугольной части $P(n)$ и заполнению верхней треугольной части матрицы теми же значениями, как в более низкой треугольной части. Другими словами, значение $P(n)$ в i-й строке и j-м столбце ($i > j$) копируется в значение $P(n)$ в j-й строке и i-м столбце.

6.5.2.1 Прогнозирующее устройство *RLS* объединеного стерео

Для элемента пары каналов (*CPE*), если *joint_stereo* в *ALSSpecificConfig* () устанавливается в 1, прогнозирующее устройство *RLS* будет работать в режиме объединенного стерео. В режиме объединенного стерео прогнозирующее устройство *RLS* использует как внутриканальный прогноз, так и межканальный прогноз.

Входными сигналами прогнозирующего устройства являются остаток прогнозирующего устройства *DPCM* канала *L*, $e_{L,1}(n)$ и остаток прогнозирующего устройства *DPCM* канала *R*, $e_{R,1}(n)$. Прогнозирующее устройство объединенного стерео состоит из внутриканального прогнозирующего устройства a_L и межканального прогнозирующего устройства b_L. Внутриканальное прогнозирующее устройство a_L генерирует оценку текущей входной $e_{L,1}(n)$ выборки канала *L* из прошлых *L*-выборок, $e_{L,1}(n-1)$, $e_{L,1}(n-2)$, ..., $e_{L,1}(n-M/2)$, где *M* является порядком прогнозирующего устройства *RLS* объединенного стерео. В то же время межканальное прогнозирующее устройство b_L генерирует другую оценку, $e_{L,1}(n)$, из прошлых входных выборок $e_{R,1}(n-1)$, $e_{R,1}(n-2)$, ..., $e_{R,1}(n-M/2)$ канала *R*. Эти две оценки складываются вместе. Результат $y_{L,1}(n)$ представляет оценку прогнозирующего устройства *RLS* объединенного стерео канала *L*. Этот процесс представляется как

$$y_{L,1}(n) = \sum_{m=1}^{M/2} \alpha_{L,m} e_{L,1}(n-m) + \sum_{m=1}^{M/2} b_{L,m} e_{R,1}(n-m),$$

где $a_{L,m}$ — коэффициенты внутриканального прогнозирующего устройства a_L, $b_{L,m}$ — коэффициенты межканального прогнозирующего устройства b_L. Первый элемент суммирования в уравнении является оценкой внутриканального прогнозирующего устройства, а второй элемент суммирования — оценкой межканального прогнозирующего устройства. Этот остаток $e_{L,2}(n)$ генерируется, вычитая оценку $y_{L,1}(n)$ из $e_{L,1}(n)$ таким образом

$$e_{L,2}(n) = e_{L,1}(n) - y_{L,1}(n).$$

Прогнозирующее устройство *RLS* объединенного стерео канала *L* обновляется алгоритмом *RLS*, данным в 6.5.2, где вектор веса, входной вектор и остаток в алгоритме *RLS* пересматриваются следующим образом:

$$w_{RLS}(n) = \left[b_{L,1}(n), a_{L,1}(n), b_{L,2}(n), a_{L,2}(n), ..., b_{L,M/2}(n), a_{L,M/2}(n)\right]^T,$$

$$e_1(n) = \left[e_{R,1}(n-1), e_{L,1}(n-1), e_{R,1}(n-2), e_{L,1}(n-2), ..., e_{R,1}(n-M/2), e_{L,1}(n-M/2)\right]^T,$$

$$e_2(n) = e_{L,2}(n).$$

Входные сигналы в прогнозирующее устройство для правого звукового канала являются остатком прогнозирующего устройства *DPCM* канала *L*, $e_{L,1}(n)$, и остатком прогнозирующего устройства *DPCM* канала *R*, $e_{R,1}(n)$. Прогнозирующее устройство *RLS* объединенного стерео состоит из внутриканального прогнозирующего устройства a_R и межканального прогнозирующего устройства b_R. Внутриканальное прогнозирующее устройство a_R генерирует оценку текущей входной выборки канала *R* $e_{R,1}(n)$ из прошлых выборок $e_{R,1}(n-1)$, $e_{R,1}(n-2)$, ..., $e_{R,1}(n-M/2)$, где *M* является порядком прогнозирующего устройства *RLS* объединенного стерео. В то же время межканальное прогнозирующее устройство b_R генерирует другую оценку $e_{R,1}(n)$ из входных выборок канала *L*, $e_{L,1}(n)$, $e_{R,1}(n-1)$, ..., $e_{R,1}(n-M/2+1)$. Эти две оценки суммируются вместе. Результат $y_{R,1}(n)$ является оценкой прогнозирующего устройства *RLS* объединенного стерео *R*-канала. Этот процесс представляется как

$$y_{R,1}(n) = \sum_{m=1}^{M/2} \alpha_{R,m} e_{R,1}(n-m) + \sum_{m=0}^{M/2-1} b_{R,m} e_{L,1}(n-m),$$

где $a_{R,m}$ — коэффициенты внутриканального прогнозирующего устройства a_R, $b_{R,m}$ — коэффициенты межканального прогнозирующего устройства b_R. В этом уравнении первый элемент суммирования является оценкой внутриканального прогнозирующего устройства, а второй элемент суммирования — оценкой межканального прогнозирующего устройства. Остаток $e_{R,2}(n)$ генерируется, вычитая оценку $y_{R,1}(n)$ из $e_{R,1}(n)$

$$e_{R,2}(n) = e_{R,1}(n) - y_{R,1}(n).$$

Прогнозирующее устройство *RLS* объединенного стерео канала *R* обновляется алгоритмом *RLS*, данным в 6.5.2, где вектор веса, входной вектор и остаток в алгоритме *RLS* переопределяются следующим образом:

$$w_{RLS}(n) = \left[b_{L,0}(n), a_{L,1}(n), b_{L,1}(n), a_{L,2}(n), ..., b_{L,M/2-1}(n), a_{L,M/2}(n)\right]^T,$$

$$e_1(n) = \left[e_{R,1}(n-1), e_{L,1}(n), e_{R,1}(n-2), e_{L,1}(n-1), ..., e_{R,1}(n-M/2), e_{L,1}(n-M/2+1)\right]^T,$$

$$e_2(n) = e_{R,2}(n).$$

Декодер обрабатывает *CPE* в порядке *LRLRLR*... Текущая выборка канала *L* всегда декодируется перед текущей выборкой канала *R*.

6.5.2.2 Прогнозирующее устройство *RLS* моно

Для элемента одиночного канала (*SCE*) прогнозирующее устройство *RLS* работает в режиме моно. В режиме моно прогнозирующее устройство *RLS* обновляется алгоритмом *RLS*.

Для *CPE*, если *joint_stereo* в *ALSSpecificConfig ()* устанавливается в 0, используется моно *RLS* для каждого отдельного канала в *CPE*. Для *CPE*, если *mono_block* в *RLSLMS_extension ()* устанавливается в 1, *CPE* будет кодировано как два отдельных канала *L* и *L-R*. Канал *L* обрабатывается как *SCE*, тогда как канал различия *L-R* проходит непосредственно в кодер энтропии. Для *SCE*, если входной фрейм содержит только постоянные величины, прогнозирующее устройство *RLS-LMS* применяется для этого фрейма. Для *CPE*, если входной фрейм обоих каналов содержит только постоянные величины, прогнозирующее устройство *RLS-LMS* применяется для этого фрейма.

6.5.2.3 Операция фильтрации в прогнозирующем устройстве *RLS*

Следующий псевдокод иллюстрирует, как прогнозирующего устройства *RLS* порядка *M* генерирует оценочный сигнал.

Псевдокод	Комментарии
*INT32 *w, *buf;* *INT64 temp* = 0; *for (i=0; i<M; i++)* *temp += (INT64) w[i] * buf[i];*	*w* является вектором веса прогнозирующего устройства *RLS* (формат .16) *buf* является входным вектором прогнозирующего устройства *RLS* (формат .0) *temp = w * buf*

temp >= 12;
if (*temp*>0x40000000) *temp* = 0x40000000;
if (*temp*<-0x40000000) *temp* =-0x40000000;
	temp является форматом .4
	Ограничение диапазона *temp*
	[-0x40000000, 0x40000000]

INT32 y; *y = (INT32) (temp);*	*y* является оценкой прогнозирующего устройства *RLS* (формат .4)

6.5.2.4 Адаптация веса в прогнозирующем устройстве *RLS*

Следующий псевдокод иллюстрирует, как обновляется вектор веса прогнозирующего устройства *RLS* порядка *M*.

Псевдокод	Комментарии
INT32 x, y, e; *e = x-y;*	*x* является текущей входной выборкой прогнозирующего устройства *RLS* *x* имеет формат .4 *y* является оценкой прогнозирующего устройства (формат .4) *e* является остатком прогнозирующего устройства *RLS* (формат .4)

Продолжение

Псевдокод	Комментарии
*INT64 **P;* *INT32 *buf, *v;* *INT16 vs;* *[v, vs] = MulMtxVec(P, buf);*	*P* является *RLS* алгоритма матрицы *P* (формат .60) *buf* является входным вектором прогнозирующего устройства *RLS* (формат .0) *v* является вектором алгоритма *RLS v* *vs* является масштабным коэффициентом *v* = P**buf* *v* имеет формат .(28-*vs*)
INT64 temp; *INT16 ds;* *[temp, ds] = MulVecVec(buf, v);* *i = 0;* *while(temp>0x20000000 && temp!=0)* *temp >>= 1; i++;* *i += vs + ds;* *if (i<=60)* *temp += (((INT64) 1) << (60-i));* *else reinit_P(P);*	*ds* является масштабным коэффициентом *temp = buf * v* *temp* имеет формат .(60-*vs*-*ds*) *temp* имеет формат .(60-*i*) Если (*i*<=60), то *temp* = *temp* + 1; иначе переинициализация матрицы *P*
INT64 mm; *INT32 m;* *if (temp==0)* *mm = 1L<<30;* *else if (i<=28)* *shift = 28-i;* *mm = (((INT64)1)<<62) / temp;* *if (shift>32)* *mm = 1L<<30;* *else* *mm <<= shift;* *else* *mm = (((INT64)1)<<(90-i)) / temp;* *m = (INT32)mm;*	*m* — это алгоритма *RLS* переменной *m* Если (*temp*==0), то *m* = 1; иначе *m* = 1/*temp*; *m* имеет формат .30
*INT32 *k;* *for (i=0; i<M; i++)* *temp = (INT64) v[i] * m;* *if (vs>=12)* *k[i] = temp«(vs-12);* *else* *k[i] = temp>>(11-vs);* *k[i] = ROUND2(k[i]);* *temp = 0;* *for (i=0; i<M; i++)* *temp |= (k[i]>0 ? k[i]:-k[i]);* *ds = fast_bitcount(temp);* *if (ds>30)* *ds -= 30;* *for (i=0; i<M; i++) k[i] >>= ds;* *else ds = 0;*	*k* — вектор усиления *k* алгоритма *RLS* *к = m * v* Масштаб *k* *k* имеет формат .46 Получение *MSB* в *k* Нахождение позиции бита *MSB* Масштаб *k* *k* имеет формат .(46-ds)
*INT32 *w;* *for (i=0; i<M; i++)* *temp = (((INT64) k[i] * (e>>3))>>(30-ds));* *temp = w[i] + ROUND2(temp);* *w[i] = (long) temp;* *vs += ds;*	*w* — вектор веса прогнозатора *RLS* Обновление вектора веса *RLS* *w = w + k * e* *w* имеет формат .16

Продолжение

Псевдокод	Комментарии
INT16 lambda; *for (i=0; i<M; i++)* *for (j=0; j<=i; j++)* *temp = ((INT64) k[i] * v[j])>>(14-vs);* *P[i][j] -= temp;* *if (P[i][j]>=0x4000000000000000* *II P[i][j]<=-0x4000000000000000)* *reinit_P(P); break;* *temp = P[i][i] / lambda;* *P[i][j] += temp;* *for (i=1; i<M; i++)* *for (j=0; j<i; j++)* *P[j][i] = P[i][j];* *for (i=M-1; i>0; i--)* *buf[i] = buf[i-1];* *buf[0] = x>>4;*	*lambda* определяется в 6.5.6.6 Обновление матрицы *P* (нижний треугольник) *P = P – k * v* *P = P * (1+1/lambda)* *P* имеет формат .60 Обновление матрицы *P* (верхний треугольник) Обновление входного вектора прогнозатора *RLS* *buf* имеет формат .0
*[INT32 *v, INT16 vs] =* *MulMtxVec(INT64 **P, INT32 *buf)*	Умножение матрицы *P* на *buf* Возвращение результата в *v* *P* имеет формат .60, *buf* имеет формат .0
INT64 temp=0; *INT16 ps;* *for(i=0; i<M; i++)* *for(j=0; j<=i; j++)* *temp |= (P[i][j]>0 ? P[i][j] : -P[i][j]);* *ps = 63 – fast_bitcount(temp);*	 *ps* является масштабным коэффициентом Получение *MSB* в матрице *P* Подсчет сдвига, необходимого, чтобы максимизировать *P*
*INT64 *u;* *for (i=0; i<M; i++) u[i]=0;* *for (j=0; j<M; j++) u[i] +=* *(INT64) (((P[i][j]«ps)+(INT64)0x80000000)>>32)* ** buf[j];*	 *u = P * buf* *u* имеет формат .(28+*ps*)
INT16 ns; *temp =0;* *for (i=0; i<M; i++)* *temp |= (u[i]>0 ? u[i] : -u[i]);* *ns = fast_bitcount(temp);* *if (ns>28)* *ns -= 28;* *for(i=0; i<M; i++)* *v[i] = (INT32) (u[i]>>ns);* *vs = ns — ps;* *else for(i=0;i<M;i++)* *v[i] = (INT32) u[i];* *vs = -ps;*	*ns* — это масштабный коэффициент Получение *MSB* Определение позиции бита *MSB* Масштабирование *v* *v* имеет формат .(28-*vs*)
INT64 temp; *temp = (z>0 ? z : -z);* *ds = fast_bitcount(temp);* *if (ds>28)* *ds -= 28;* *z = (z«(32-(ds-1)));* *z = ROUND2(z);* *else* *ds = 0;* *z = (z«32);* *reinit_P(INT64 **P)* *for (i=0; i<M; i++)* *for (j=0; j<M; j++)* *P[i][j]= 0;* *for (i=0; i<M; i++)* *P[i][i] = (INT64) JS_INIT_P;*	 Получение позиции бита *MSB* Масштаб *z* *z* имеет формат .(60-*vs*-*ds*) Переинициализация матрицы *P* Очистка *P* в нуль Инициализация диагональных компонент

Окончание

Псевдокод	Комментарии
*[INT64 z, INT16 ds] = MulVecVec(INT32 *buf, INT32 *v)* *INT 64 z = 0;* *for (i=0; i<M; i++)* *z += (INT64) buf[i] * v[i];*	Вычисление внутреннего продукта *buf* и *v* Возвращение результата в *z* *buf* имеет формат .0, *v* — формат .(28-*vs*) *z = buf * v*
[INT16 count] = fast_bitcount (INT64 temp) *i = 56; j = 0;* *while((temp>>i)==0 && i>0) i -= 8;* *temp>>=i;* *while(temp>0)* *temp>>=1; j++;* *count = i + j;*	Возврат позиции бита *MSB*

Прогнозирующее устройство *RLS* может быть выключено установкой *RLS_order* в нуль. В этом случае оценка прогнозирующего устройства *RLS* $y_2(n)$ обнуляется.

6.5.3 Прогнозирующие устройства *LMS*

Ввод: $e_{k-1}(n)$ остаток предыдущего прогнозирующего устройства (может быть прогнозирующим устройством *RLS* или прогнозирующим устройством *LMS*).

Остаток: $e_k(n)$ как вход в следующее прогнозирующее устройство *LMS*.

Оценка: $y_k(n)$ как вход в линейный объединитель.

Прогнозирующее устройство *RLS-LMS* содержит ряд прогнозирующих устройств *LMS*. Чтобы адаптировать веса прогнозирующего устройства, используется алгоритм нормализованного *LMS* (*NLMS*). Для прогнозирующего устройства *LMS M*-порядка его вектора веса

$$w_{LMS}(n) = \left[w_{LMS,1}(n), w_{LMS,2}(n), ..., w_{LMS,M}(n)\right]^T$$

инициализируется как

$$w_{rls}(0) = 0.$$

Для каждого индекса времени *n*, *n* = 1, 2, ..., оценка вычисляется как

$$y_k(n) = w_{LMS}^T(n) e_{k-1}(n),$$

где $e_{k-1}(n)$ является входным вектором, определенным как

$$e_{k-1}(n) = \left[e_{k-1}(n-1), e_{k-1}(n-2), ..., e_{k-1}(n-M)\right]^T.$$

Вектор веса прогнозирующего устройства *LMS* обновляется согласно

$$e_k(n) = e_{k-1}(n) - y_k(n),$$

$$w_{LMS}(n) = w_{LMS}(n-1) + \frac{e_k(n) e_{k-1}(n)}{2^7 - \mu_k e_{k-1}^T(n) e_{k-1}(n)},$$

где μ_k является длиной шага *NLMS*.

6.5.3.1 Операция фильтрации в прогнозирующем устройстве *LMS*

Следующий псевдокод иллюстрирует, как прогнозирующее устройство *LMS M*-порядка генерирует оценочный сигнал. Порядок прогнозирующего устройства *LMS* определяется в 6.5.6.3.

Псевдокод	Комментарии
*INT32 *w, *buf;* *INT64 temp = 0;* *for (i=0; i<M; i++)*	*w* является вектором веса прогнозирующего устройства *LMS* (формат .24) *buf* является входным вектором прогнозирующего устройства *LMS* (формат .0) *temp = w * buf*

Окончание

Псевдокод	Комментарии
*temp += (INT64) w[i] * buf[i];* *temp >>= 20;* *if (temp>0x7fffffff) temp = 0x7fffffff;* *if (temp<-0x7fffffff) temp = -0x7fffffff;* *INT32 y;* *y = (INT32) (temp);*	*temp* имеет формат .4 Ограничение диапазона *temp* до [-0x7fffffff, 0x7fffffff] *y* является оценкой прогнозирующего устройства *LMS* (формат .4)

6.5.3.2 Адаптация веса в прогнозирующем устройстве *LMS*

Следующий псевдокод иллюстрирует, как обновляется вектор веса прогнозирующего устройства *LMS* порядка *M*.

Псевдокод	Комментарии
INT32 x, y, e; *e = x-y;*	*x* является текущей входной выборкой прогнозирующего устройства *LMS* (формат .4) *y* является оценкой прогнозирующего устройства *LMS* (формат .4) *e* является остатком прогнозирующего устройства *LMS* (формат .4)
*INT32 *buf;* *INT64 pow = 0;* *for (i=0; i<M; i++)* *pow += (INT64) buf[i] * buf[i];* *if (pow>0x4000000000000000)* *pow = 0x4000000000000000;*	*buf* является входным вектором прогнозирующего устройства *LMS* (формат .0) Вычисление полной мощности сигнала в *buf* *pow = buf * buf* *pow* имеет формат .0
INT16 mu; *INT64 temp, temp1;* *temp = (INT64) mu * (pow>>7);* *temp1 = temp;* *i = 0;* *while(temp>0x7fffffff)* *temp>>=1; i++;* *temp = ((INT64) e<<(29-i)) /* *(INT64)((temp1+1)>>i);*	*mu* является длиной шага алгоритма *NLMS* (формат .0) *mu* определяется в *xxx.yyy* *temp = mu * pow* *temp* имеет формат .(-7) Вычислите количества сдвигов, необходимых для смещения *temp* в более низкие 32 бита *temp = e / (128 + mu*buf*buf)* *temp* имеет формат .40
*INT32 *w;* *for (i=0; i<M;i++) w[i] +=* *(INT32) (((INT64) buf[i] * temp +* *0x8000)>>16);* *for (i=M-1; i>0; i--)* *buf[i] = buf[i-1];* *buf[0] = x>>4;*	*w* является вектором веса прогнозирующего устройства *LMS* (формат .24) Обновление вектора веса *w = w + (buf*e) / (128 + mu*buf*buf)* Обновление входного вектора *buf* имеет формат .0

6.5.4 Линейный объединитель

Ввод: $y_1(n)$, $y_2(n)$, ..., $y_K(n)$ оценивает прогнозирующие устройства из *DPCM*, *RLS* и *LMS*.

Вывод: $x(n)$ заключительная оценка прогнозирующего устройства *RLS-LMS*.

Линейный объединитель умножает оценки из прогнозирующих устройств *DPCM*, *RLS* и *LMS* на ряд весов. Результаты суммируются, чтобы обеспечить заключительную оценку прогнозирующего устройства *RLS-LMS*. Знак алгоритм *Sign-Sign LMS* используется, чтобы обновить веса линейного объединителя. Если в каскаде прогнозирующего устройства *RLS-LMS* есть всего *K* прогнозирующих устройств, вектор веса линейного объединителя дается выражением

$$c(n) = \left[c_1(n), c_2(n), ..., c_k(n)\right]^T.$$

Входной вектор линейного объединителя дается так

$$y(n) = \left[y_1(n), y_2(n), ..., y_k(n)\right]^T.$$

Заключительная оценка прогнозирующего устройства *RLS-LMS* дается следующим образом

$$x(n) = c^T(n)y(n).$$

Вектор веса линейного объединителя обновляется алгоритмом *Sign-Sign LMS*

$$c(n) = c(n-1) + \alpha \operatorname{sgn}\{y(n)\} \operatorname{sgn}\{x(n) - x(n)\},$$

где $x(n)$ является текущей входной выборкой прогнозирующего устройства *RLS-LMS*, a является длиной шага *Sign-Sign LMS*. Функция $sgn\{*\}$ определяется как

$$\operatorname{sgn}\{r\} = \begin{cases} 1, & \text{если } r > 0 \\ 0, & \text{если } r = 0 \\ -1, & \text{если } r < 0. \end{cases}$$

Следующий псевдокод иллюстрирует, как линейный объединитель *K*-порядка генерирует заключительную оценку прогнозирующего устройства *RLS-LMS*. Порядок линейного объединителя дается из *LMS stage*+2. Код также показывает, как обновляется вектор веса линейного объединителя. Первые два веса линейного объединителя не обновляются.

Псевдокод	Комментарии
*INT32 *c, *y;* *INT64 xhat = 0;* *for (i=0; i<K; i++)* *xhat += (INT64) c[i] * y[i];* *xhat >>= 24;*	*c* является вектором веса линейного объединителя (формат .24) *y* является входным вектором линейного объединителя (формат .4) *xhat* является выводом линейного объединителя, то есть заключительной оценкой прогнозирующего устройства *RLS-LMS* *xhat* имеет формат .4
INT32 x; *INT64 r;* *r = (x«4) - xhat;*	*x* является текущей входной выборкой прогнозирующего устройства *RLS-LMS* *x* имеет формат .0 Вычисление различия между *x* и *xhat* — *r* имеет формат .4
INT32 LMS_stepsize; *INT64 temp;* *for (i=2; i<K; i++)* *temp = (INT64) r * y[i];* *if (temp>0 && c[i]<0x40000000)* *c[i] += LMS_stepsize;* *if (temp<0 && c[i]>-0x40000000)* *c[i] -= LMS_stepsize;*	Длина шага алгоритма *Sign-Sign LMS* (формат .24) *LMS stepsize* определяется в 6.5.6.5 Алгоритм *Sign-Siign LMS* Обновление только весов линейного объединителя для прогнозирующих устройств *LMS*
INT32 e; *e = x - R0UND1(xhat)*	*e* является остатком прогнозирующего устройства *RLS-LMS* (формат .0)

6.5.5 Инициализация прогнозирующего устройства *RLS-LMS*

Прогнозирующее устройство *RLS-LMS* инициализируется в следующие моменты: запуск кодирования, запуск декодирования, старт каждого фрейма произвольного доступа (*PA*) и всякий раз, когда изменяется порядок фильтра. Прогнозирующее устройство *RLS-LMS* инициализируется заполнением нулями следующих буферов: предыдущая входная выборка прогнозирующего устройства *DPCM*, входной вектор и вектор веса прогнозирующего устройства *RLS*, все входные векторы и векторы веса прогнозирующего устройства *LMS* и входной вектор линейного объединителя. Матрица *P* прогнозирующего устройства *RLS* инициализируется, вызывая функцию *reinit_P*(*P*). Вектор веса линейного объединителя устанавливается в константу *FRACTION*, которая представляет 1,0 в формате 8.24.

В таблице 27 перечисляются константы и макросы, используемые прогнозирующим устройством *RLS-LMS*.

Таблица 27 — Константы и макросы

Константы и макросы	Значение	Комментарии
JS_INIT_P	115292150460684	0,0001 в формате 4.60
FRACTION	(1*L*«24)	1,0 в формате 8.24
ROUND1(x)	((*INT*32) ((*x*+8)>>4))	Функция округления
ROUND2(x)	((*INT*64) ((*INT*64) *x* + (*INT*64) 1)>>1)	Функция округления

6.5.6 Параметры прогнозирующего устройства *RLS-LMS*

Параметры прогнозирующего устройства *RLS-LMS* могут быть изменены каждый фрейм. Об этом сообщается в *RLSLMS extension*(), когда *ext mode* = 1.

6.5.6.1 *RLS_order*

Параметр *RLS order* определяет порядок прогнозирующего устройства *RLS*. Допустимые значения и соответствующие 4-битовые индексы перечисляются в таблице 28.

Таблица 28 — *RLS_order*

Индекс	*RLS_order*	Индекс	*RLS_order*
0	0	8	16
1	2	9	18
2	4	10	20
3	6	11	22
4	8	12	24
5	10	13	26
6	12	14	28
7	14	15	30

6.5.6.2 *LMS_stage*

Параметр *LMS stage* определяет число прогнозирующих устройств *LMS* в цепочке прогнозирующих устройств *RLS-LMS*. Допустимые значения и соответствующие 3-битовые индексы перечисляются в таблице 29.

Таблица 29 — *LMS stage*

Индекс	*LMS_stage*
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8

6.5.6.3 *LMS_order*

Параметр *LMS_order* определяет порядок прогнозирующего устройства *LMS*. Допустимые значения и соответствующие 5-битовые индексы перечисляются в таблице 30.

Таблица 30 — *LMS order*

Индекс	*LMS_order*	Индекс	*LMS_order*
0	2	16	32
1	3	17	36
2	4	18	48
3	5	19	64
4	6	20	80
5	7	21	96
6	8	22	128
7	9	23	256
8	10	24	384
9	12	25	448
10	14	26	512
11	16	27	640
12	18	28	768
13	20	29	896
14	24	30	1024
15	28	31	Зарезервировано

6.5.6.4 *LMS_mu*

Параметр *LMS_mu* определяет длину шага (*stepsize*) алгоритма *NLMS*, который используется, чтобы обновить прогнозирующее устройство *LMS*. Допустимые значения и соответствующие 5-битовые индексы перечисляются в таблице 31.

Таблица 31 — *LMS mu*

Индекс	*LMS_mu*	Индекс	*LMS_mu*
0	1	16	18
1	2	17	20
2	3	18	22
3	4	19	24
4	5	20	26
5	6	21	28
6	7	22	30
7	8	23	35
8	9	24	40
9	10	25	45
10	11	26	50
11	12	27	55
12	13	28	60
13	14	29	70
14	15	30	80
15	16	31	100

6.5.6.5 *LMS stepsize*

Параметр *LMS stepsize* определяет длину шага (*stepsize*) алгоритма *Sign-Sign LMS*, который используется, чтобы обновить линейный объединитель. Допустимые значения (формат 7.24) и соответствующие 3-битовые индексы перечисляются в таблице 32.

Таблица 32 — *LMS_stepsize*

Индекс	*LMS_stepsize*	Комментарии
0	1677	0,0001 (формат .24)
1	3355	0,0002
2	8388	0,0005
3	16777	0,001
4	33554	0,002
5	83886	0,005
6	167772	0,01
7	335544	0,02

6.5.6.6 *RLS_lambda* и *RLS_lambda_ra*

RLS_lambda и *RLS_lambda_ra* являются параметрами, которые управляют фактором игнорирования прогнозирующего устройства *RLS*. *RLS_lambda_ra* используется только для фрейма *PA*, тогда как *RLS_lambda* используется и для обоих фреймов *PA* и *non-PA*. *RLS_lambda* и *RLS_lambda_ra* принимают целочисленные значения в диапазоне [1, 1023]. Фактор игнорирования *RLS* λ вычисляется как

$$\lambda = \frac{RLS_lambda}{RLS_lambda + 1}$$

и

$$\lambda = \frac{RLS_lambda_ra}{RLS_lambda_ra + 1}.$$

6.5.7 Произвольный доступ

Прогнозирующее устройство *RLS-LMS* переинициализируется в начале каждого фрейма произвольного доступа. Во фрейме *PA* прогнозирующее устройство *RLS-LMS* использует *RLS_lambda_ra*, чтобы определить фактор игнорирования *RLS* для первых 300 выборок. Для остальной части выборок во фрейме используется *RLS_lambda*. Во фрейме *PA* прогнозирующие устройства *LMS* в цепочке прогнозирующих устройств *RLS-LMS* начинают обновлять свои состояния только после первых *N*/32 выборок во фрейме, где *N* является длиной фрейма.

6.6 Кодированный остаток

Есть два возможных режима для передачи остатка прогноза: быстрая схема кодирования, использующая простые коды Райса, и более сложная и эффективная схема, которая использует блок кодов Гильберта — Мура (*BGMC*).

6.6.1 Коды Райса (*Rice Codes*)

Когда флаг *bgmc_mode* в *ALSSpecificConfig* устанавливается в 0, остаточные значения являются кодированной энтропией, использующей коды Райса. Выбранный синтаксис для генерации кодовой комбинации определяется в следующем.

Код Райса определяется параметром $s \geq 0$. Для данного значения s каждая кодовая комбинация состоит из ρ-битного префикса и s-битного субкода. Префикс сообщается, используя $\rho - 1$ «1» биты и один «0» бит, где ρ зависит от кодированного значения. Для значения сигнала x и $s > 0$ $\rho - 1$ вычисляется следующим образом («÷» означает целочисленное деление без остатка):

$$\rho - 1 = \begin{cases} x \div 2^{s-1} & \text{для } x \geq 0 \\ (-x-1) \div 2^{s-1} & \text{для } x < 0. \end{cases}$$

Для $s = 0$ используем модифицированное вычисление

$$\rho - 1 = \begin{cases} 2x & \text{для } x \geq 0 \\ -2x - 1 & \text{для } x < 0. \end{cases}$$

Подкод для $s > 0$ вычисляется следующим образом:

$$sub = \begin{cases} x - 2^{s-1}(\rho - 1) + 2^{s-1} & \text{для } x \geq 0 \\ (-x - 1) - 2^{s-1}(\rho - 1) & \text{для } x < 0. \end{cases}$$

Для $s = 0$ нет никакого подкода, а только префикс, таким образом, префикс и кодовая комбинация идентичны. Разрешенными значениями являются $s = 0…15$ для разрешения выборки ≤ 16 битов и $s = 0…31$ для разрешения выборки > 16 битов.

Таблицы 33 и 34 показывают примеры для кода Райса с $s = 4$. Таблица 35 показывает специальный код Райса с $s = 0$.

Таблица 33 — Код Райса с $s = 4$. Биты xxxx содержат 4-битовый подкод *sub*

Значения	ρ	Префикс	Кодовая комбинация
– 8...+ 7	1	0	0xxxx
– 16...– 9; + 8... + 15	2	1	10xxxx
– 24...– 17; + 16...+ 23	3	110	110xxxx
– 32...– 25; + 24...+ 31	4	1110	1110xxxx
– 40...– 33; + 32...+ 39	5	11110	11110xxxx

Таблица 34 — Подкоды кода Райса с $s = 4$ для первых трех префиксов

Значения (ρ = 1)	Значения (ρ = 2)	Значения (ρ = 3)	Подкод (xxxx)
– 8	– 16	– 24	0111
– 7	– 15	– 23	0110
– 6	– 14	– 22	0101
– 5	– 13	– 21	0100
– 4	– 12	– 20	0011
– 3	– 11	– 19	0010
– 2	– 10	– 18	0001
– 1	– 9	– 17	0000
0	8	16	1000
1	9	17	1001
2	10	18	1010
3	11	19	1011
4	12	20	1100
5	13	21	1101
6	14	22	1110
7	15	23	1111

Таблица 35 — «Специальный» код Райса с $s = 0$ (префикс и кодовая комбинация идентичны)

Значения	ρ	Префикс	Кодовая комбинация
0	1	0	0
– 1	2	10	10
+ 1	3	110	110
– 2	4	1110	1110
+ 2	5	11110	11110

Для каждого блока остаточных значений все значения могут быть закодированы, используя тот же самый код Райса, или, если установлен флаг *sb_part* в заголовке файла, блок может быть разделен на четыре подблока, каждый из которых кодируется различными кодами Райса. В последнем случае флаг *ec_sub* в заголовке блока указывает, используется один или четыре блока.

В то время как параметр $s[i = 0]$ первого подблока непосредственно передается 4 битами (разрешение ≤ 16 битов) либо 5 битами (разрешение > 16 битов), передаются только различия следующих параметров $s[i > 0]$. Эти различия дополнительно кодируются, снова используя соответственно выбранные коды Райса (таблица 36).

Таблица 36 — Кодирование параметров кода Райса $s[i]$

Параметр кода (i = индекс подблока)	Различие	Параметр кода Райса, используемый для различий
$s[i]$ (i>0)	$s[i] - s[i-1]$	0

6.6.2 Режим кодирования *BGMC*

Когда флаг *bgmc_mode* в заголовке файла устанавливается в 1, остаточные значения разделяются на *MSB*, *LSB* и хвостовые компоненты, которые затем кодируются, используя блок Гильберта — Мура, фиксированную длину и коды Райса соответственно.

Кроме того, используется различная схема разделения на подблоки. Если флаг *sb_part* в заголовке файла устанавливается, каждый блок может быть разделен на 1, 2, 4 или 8 подблоков, где фактическое число обозначается 2-битовым полем *ec_sub* в заголовке блока. Если *sb_part* не устанавливается, каждый блок может быть разделен только на 1 или 4 подблока, и фактическое число указывается 1-битовым полем ec_*sub*.

6.6.2.1 Дополнительные параметры

В дополнение к параметру кода s (используется для создания кодов Райса), кодер/декодер *BGMC* опирается на следующие величины.

Число самых младших значащих битов (*LSBs*) k остатков, которые будут переданы непосредственно:

$$k = \begin{cases} 0, & \text{если } s \le B \\ s - B, & \text{если } s > B, \end{cases}$$

где s является параметром Райса и B является параметром, зависящим от размера подблока N:

$$B = \left(\left\lceil \log_2 N \right\rceil - 3\right) \gg 1,$$

где $0 \le B \le 5$ (значения за пределами границ отсекаются до границ). Число отсутствующих (в таблицах частот доступа) битов *delta*

$$delta = 5 - s + k.$$

Индекс таблицы частот sx используется для того, чтобы кодировать/декодировать *MSBs*.

Параметр sx передается в дополнение к s для каждого подблока, где 'полный' параметр *BGMC* может быть представлен как $S = 16 \cdot s + sx$. Подобно режиму кодирования Райса, первый параметр передается непосредственно, в то время как для последующих параметров передаются только закодированные различия (таблица 37).

Таблица 37 — Кодирование *BGMC* кодирует параметры *S[i]* = 16 · *s[i]* + *sx[i]*

Параметр кода (*i* = индекс подблока)	Различие	Параметр кода Райса, используемый для различий
S[i] (i>0)	*S[i] – S[i–1]*	2

6.6.2.2 Разделение остаточных значений на *MSB*, *LSB* и хвостовые части

Процесс получения отсеченных и с удаленным знаком значений *MSB*, *LSB* или хвостовых частей, соответствующих остаточным выборкам (*res [i]*), может быть описан следующим образом:

```
for (i = 1; i <= N; i++)
{
long msbi = res[i] >> k;                    // remove lsbs
if (msbi >= max_msb[sx][delta]) {           // positive tail
      msb[I] = tail_code[sx][delta];
      tail[i] = res[i] - (max_msb[sx][delta] << k);
} else
if (msbi <= -max_msb[sx][delta]) {          // negative tail
      msb[I] = tail_code[sx][delta];
      tail[i] = res[i] + ((max_msb[sx][delta] – 1) << k);
} else {                        // normal msb range
      if (msbi >= 0) msbi = msbi * 2;
      else msbi = -msbi * 2 –1;             // remove sign
      if (msbi >= tail_code[sx][delta])
         msbi ++;                           // skip tail code
msb[i] = msbi;                              // msb and lsb values
lsb[i] = res[i] & ((1<<k)-1);               // to encode
}
}
```

Максимальные абсолютные значения *MSB* и коды хвостовых частей, используемых в этом алгоритме (массивы *max_msb [ ]* и *tail_code [ ]* соответственно), определяются в таблицах 38 и 39.

Таблица 38 — Максимальные/минимальные значения остаточных *MSB*

sx \ *delta*	0	1	2	3	4	5
0	± 64	± 32	± 16	± 8	± 4	± 2
1	± 64	± 32	± 16	± 8	± 4	± 2
2	± 64	± 32	± 16	± 8	± 4	± 2
3	± 96	± 48	± 24	± 12	± 6	± 3
4	± 96	± 48	± 24	± 12	± 6	± 3
5	± 96	± 48	± 24	± 12	± 6	± 3
6	± 96	± 48	± 24	± 12	± 6	± 3
7	± 96	± 48	± 24	± 12	± 6	± 3
8	± 96	± 48	± 24	± 12	± 6	± 3
9	± 96	± 48	± 24	± 12	± 6	± 3
10	± 96	± 48	± 24	± 12	± 6	± 3
11	± 128	± 64	± 32	± 16	± 8	± 4
12	± 128	± 64	± 32	± 16	± 8	± 4
13	± 128	± 64	± 32	± 16	± 8	± 4
14	± 128	± 64	± 32	± 16	± 8	± 4
15	± 128	± 64	± 32	± 16	± 8	± 4

Таблица 39 — Коды хвостовой части

sx \ delta	0	1	2	3	4	5
0	74	44	25	13	7	3
1	68	42	24	13	7	3
2	58	39	23	13	7	3
3	126	70	37	19	10	5
4	132	70	37	20	10	5
5	124	70	38	20	10	5
6	120	69	37	20	11	5
7	116	67	37	20	11	5
8	108	66	36	20	10	5
9	102	62	36	20	10	5
10	88	58	34	19	10	5
11	162	89	49	25	13	7
12	156	87	49	26	14	7
13	150	86	47	26	14	7
14	142	84	47	26	14	7
15	131	79	46	26	14	7

Инверсный процесс (декодирование), восстанавливающий исходные остаточные выборки (*res [i]*) на основе их *MSB*, *LSB* или хвостовых частей, может быть описан следующим образом:

```
for (i = 1; i <= N; i++)
{
    if (msb[i] == tail_code[sx][delta]) {
        if (tail[i] >= 0)          // positive tail
            res[i] = tail[i] + (abs_max_x) << k;
        else                       // negative tail
            res[i] = tail[i] -(abs_max_x - 1) << k;
} else {
    int msbi = msb[i];
if (msbi > tail_code[sx][delta])
    msbi --;                       // skip tail code
if (msbi & 1)
    msbi = (-msbi –1)/2;           // remove sign
else
    msbi = msbi/2;
res[i] = (msbi << k) | lsb[i];     // add lsbs
}
}
```

6.6.2.3 Кодирование и декодирование *MSB*

Отсеченные *MSB* остаточных выборок блочно кодируются, используя коды Гильберта — Мура, созданные для распределения (кумулятивная таблица частот) индексированного параметром *sx*.

Процесс кодирования состоит из инициализации состояния (арифметического) кодера блока Гильберта — Мура, последовательного кодирования всех значений *MSB* во всех подблоках и сбрасывания состояния кодера.

Спецификации на языке *C* соответствующих функций кодера даются ниже.

```
#define FREQ_BITS 14 // # bits used by freq. counters
#define VALUE_BITS 18 // # bits used to describe code range
```

```
#define TOP_VALUE 0x3FFFF // largest code value
#define FIRST_QTR 0x10000 // first quarter
#define HALF 0x20000 // first half
#define THIRD_QTR 0x30000 // third quarter

// encoder state variables:
static unsigned long high, low, bits_to_follow;

// start encoding:
void bgmc_start_encoding (void)
{
        high = TOP_VALUE;
        low = 0;
        bits_to_follow = 0;
        }

// sends a bit followed by a sequence of opposite bits:
void put_bit_plus_follow (unsigned long bit)
{
put_bit (bit);
while (bits_to_follow) {
        put_bit (bit ^ 1);
        bits_to_follow --;
}
}

// encodes a symbol using Gilbert-Moore code for
// a distribution s_freq[] subsampled by delta bits:
void bgmc_encode (unsigned long symbol, long delta, unsigned long *s_freq)
{
 unsigned long range = high –low +1;
 high=low+((range*s_freq[symbol<<delta]-(1<<FREQ_BITS))>>FREQ_BITS);
 low =low+((range*s_freq[(symbol+1)<< delta])>>FREQ_BITS);

 for ( ; ; ) {
   if (high < HALF) {

  put_bit_plus_follow (0, p);
} else if (low >= HALF) {
    put_bit_plus_follow (1, p);
    low -= HALF;
    high -= HALF;
} else if (low >= FIRST_QTR && high < THIRD_QTR) {
    bits_to_follow += 1;
    low -= FIRST_QTR;
    high -= FIRST_QTR;
} else
    break;
    low = 2 * low;
    high = 2 * high + 1;
    }
}

// Finish the encoding:
static void bgmc_finish_encoding ()
{
```

```
        bits_to_follow += 1;
        if (low < FIRST_QTR) put_bit_plus_follow (0,p);
        else put_bit_plus_follow (1,p);
    }
```

Спецификации соответствующих функций декодера блока Гильберта — Мура на языке *C* даются ниже.

```
    // decoder state variables:
    static unsigned long high, low, value;
    // start decoding:
    void bgmc_start_decoding (void)
    {
        high = TOP_VALUE;
        low = 0;
        value = get_bits(VALUE_BITS);
        }
    // decodes a symbol using Gilbert-Moore code for
    // a distribution s_freq[] subsampled by delta bits:
    unsigned long bgmc_decode (long delta, unsigned long *s_freq)
    {
        unsigned long range, target, symbol;
        range = high - low + 1;
        target = (((value - low + 1) << FREQ_BITS) - 1) / range;
        symbol = 0,
        while (s_freq [(symbol+1) << delta] > target)
            symbol ++;
    high=low+((range*s_freq[symbol<<delta]-(1<<FREQ_BITS))>>FREQ_BITS);
    low =low+((range*s_freq[(symbol+1)<<delta])>>FREQ_BITS);
    for ( ; ; ) {
        if (high < HALF) ;
        else if (low >= HALF) {
            value -= HALF;
            low -= HALF;
            high -= HALF;
    } else if (low >= FIRST_QTR && high < THIRD_QTR) {
            value -= FIRST_QTR;
            low -= FIRST_QTR;
            high -= FIRST_QTR;
    } else
        break;
            low = 2 * low;
            high = 2 * high + 1,
            value = 2 * value + get_bit ();
    }
    return symbol; }

    // Finish decoding:
    void bgmc_finish_decoding ()
    {
        scroll_bitstream_position_back(VALUE_BITS-2);
    }
```

Совокупные таблицы частот (массивы *s_freq [ ]*), используемые вышеприведенными алгоритмами для кодирования/декодирования остаточных *MSB*, перечисляются ниже. Соответствующая (в пределах каждого подблока) таблица выбирается, используя параметр *sx*.

Таблица 40 — Совокупные таблицы частот, используемые кодером/декодером *BGMC*

Номер	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384	16384
'	16066	16080	16092	16104	16116	16128	16139	16149	16159	16169	16177	16187	16195	16203	16210	16218
2	15748	15776	15801	15825	15849	15872	15894	15915	15934	15954	15970	15990	16006	16022	16036	16052
3	15431	15473	15510	15546	15582	15617	15649	1568'	15709	15739	15764	15793	15817	15842	15863	15886
4	15114	15170	15219	15268	15316	15362	15405	15447	15485	15524	15558	15597	15629	15662	15690	15720
5	14799	14868	14930	14991	15050	15107	15162	15214	15261	15310	15353	1540'	15441	15482	15517	15554
6	14485	14567	14641	14714	14785	14853	14919	1498'	15038	15096	15148	15205	15253	15302	15344	15389
7	14173	14268	14355	14439	1452'	14600	14677	14749	14816	14883	14944	15009	15065	15122	15172	15224
8	13861	13970	14069	14164	14257	14347	14435	14517	14594	14670	14740	14813	14878	14942	15000	15059
9	13552	13674	13785	13891	13995	14096	14195	14286	14373	14458	14537	14618	14692	14763	14828	14895
10	13243	13378	13501	13620	13734	13846	13955	14055	14152	14246	14334	14423	14506	14584	14656	14731
11	12939	13086	13219	13350	13476	13597	13717	13827	13933	14035	14132	14230	14321	14406	14485	14567
12	12635	12794	12938	13081	13218	13350	13479	13599	13714	13824	13930	14037	14136	14228	14314	14403
13	12336	12505	12661	12815	12963	13105	13243	13373	13497	13614	13729	13845	13952	14051	14145	14240
14	12038	12218	12384	12549	12708	12860	13008	13147	13280	13405	13529	13653	13768	13874	13976	14077
15	11745	11936	12112	12287	12457	12618	12775	12923	13065	13198	13330	13463	13585	13698	13808	13915
16	11452	11654	1184'	12025	12206	12376	12542	12699	12850	12991	13131	13273	13402	13522	13640	13753
17	1116'	11373	1157'	11765	11956	12135	12310	12476	12636	12785	12933	13083	13219	13347	13472	13591
18	10870	11092	1130'	11505	11706	11894	12079	12253	12422	12579	12735	12894	13037	13172	13304	13429
19	10586	10818	11037	11250	11460	11657	11851	12034	12211	12376	12539	12706	12857	12998	13137	13269
20	10303	10544	10773	10996	11215	1142'	11623	11815	12000	12173	12343	12518	12677	12824	12970	13109
21	10027	10276	10514	10746	10975	11189	11399	11599	11791	11972	12150	12332	12499	12652	12804	12950
22	975'	10008	10256	10497	10735	10957	11176	11383	11583	11772	11957	12146	12321	12480	12639	12791
23	9483	9749	10005	10254	10500	10730	10956	1117'	11378	11574	11766	11962	12144	12310	12475	12633
24	9215	9490	9754	10011	10265	10503	10737	10959	11173	11377	11576	11778	11967	12140	12312	12476
25	8953	9236	9508	9772	10034	10279	10521	10750	10971	11182	11388	11597	11792	11971	12149	12320

Продолжение таблицы 40

Номер	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
26	8692	8982	9263	9534	9803	10056	10305	1054′	10769	10987	11200	11416	11617	11803	11987	12164
27	8440	8737	9025	9303	9579	9838	10094	10337	10571	10795	11015	11237	11444	11637	11827	12009
28	8189	8492	8787	9072	9355	9620	9883	10133	10373	10603	10830	11059	1127′	11471	11667	11854
29	7946	8256	8557	8848	9136	9407	9677	9933	10179	10414	10647	10882	11100	11307	11508	1170′
30	7704	8020	8327	8624	8917	9195	9471	9733	9985	10226	10465	10706	10930	11143	11349	11548
31	7472	7792	8103	8406	8703	8987	9268	9536	9793	10040	10285	10532	10762	10980	11192	11396
32	7240	7564	7879	8188	8489	8779	9065	9339	9601	9854	10105	10358	10594	10817	11035	11244
33	7008	7336	7655	7970	8275	8571	8862	9142	9409	9668	9925	10184	10426	10654	10878	11092
34	6776	7108	7431	7752	8061	8363	8659	8945	9217	9482	9745	10010	10258	10491	1072′	10940
35	6554	6888	7215	7539	7853	8159	8459	875′	9029	9299	9568	9838	10091	10330	10565	10790
36	6333	6669	7000	7327	7645	7955	8260	8557	8842	9116	939′	9666	9925	10169	10410	10640
37	6122	6459	6792	7123	7444	7758	8067	8369	8658	8937	9218	9497	9761	10011	10257	10492
38	5912	6249	6585	6919	7244	7561	7874	818′	8475	8759	9045	9328	9598	9853	10104	10344
39	5711	6050	6387	6724	7051	7371	7688	7998	8297	8585	8876	9163	9438	9697	9953	10198
40	5512	5852	6190	6529	6858	7182	7502	7816	8120	8411	8707	8999	9278	9542	9802	10052
41	5320	5660	5998	6339	6671	6997	7321	7638	7946	8241	854′	8837	9120	9389	9654	9908
42	5128	5468	5807	6150	6484	6812	7140	7460	7773	8071	8375	8675	8963	9236	9506	9764
43	4947	5286	5625	5970	6305	6635	6965	7288	7604	7906	8213	8517	8809	9086	9359	9622
44	4766	5104	5445	5790	6127	6459	6790	7116	7435	7741	805′	8359	8655	8936	9213	9481
45	4595	4931	5272	5618	5956	6289	6621	6950	7271	7580	7894	8205	8504	8789	9070	9342
46	4425	4760	5100	5446	5785	6120	6452	6785	7108	7419	7737	8051	8354	8642	8927	9203
47	4264	4598	4937	5282	5622	5957	6290	6625	6950	7263	7583	7901	8207	8498	8787	9066
48	4104	4436	4774	5119	5459	5795	6128	6465	6792	7107	7429	7751	8060	8355	8647	8929
49	3946	4275	4613	4957	5298	5634	5968	6306	6634	6952	7277	7602	7914	8212	8508	8793
50	3788	4115	4452	4795	5137	5473	5808	6147	6477	6797	7125	7453	7769	8070	8369	8657
51	3640	3965	4301	4642	4983	5319	5655	5995	6326	6647	6977	7308	7627	793′	8233	8524

Продолжение таблицы 40

Номер	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
52	3493	3816	4150	4490	4830	5165	5503	5843	6175	6497	6830	7163	7485	7792	8097	8391
53	3355	3674	4007	4345	4684	5018	5356	5697	6029	6353	6687	7022	7347	7656	7964	8261
54	3218	3534	3865	4201	4539	4871	5209	555′	5883	6209	6544	6882	7209	7520	7831	8131
55	3090	3403	3731	4065	440′	4732	5069	5411	5742	6070	6406	6745	7074	7388	7700	8003
56	2963	3272	3597	3929	4263	4593	4929	527′	5602	5931	6268	6609	6939	7256	7570	7875
57	2842	3147	3469	3798	413′	4458	4794	5135	5466	5796	6133	6476	6807	7126	7442	7749
58	2721	3023	3341	3669	3999	4324	4660	5000	5330	5661	5998	6343	6676	6996	7315	7624
59	2609	2907	3218	3547	3874	4197	4532	487′	5199	5531	5868	6214	6548	6870	7190	7502
60	2498	2792	3099	3425	3750	4071	4404	4742	5068	5401	5738	6085	6420	6744	7065	7380
61	2395	2684	2981	3310	3632	3951	4282	4618	4943	5275	5612	5960	6296	662′	6943	7260
62	2292	2577	2869	3196	3515	3831	4160	4495	4818	5150	5487	5835	6172	6498	6821	7140
63	2196	2476	2758	3086	340′	3714	4041	4374	4696	5027	5364	5712	6050	6377	6701	7022
64	2100	2375	2652	2976	3287	3597	3922	4253	4574	4904	524′	5589	5928	6256	6581	6904
65	2004	2274	2546	2866	3173	3480	3803	4132	4452	4781	5118	5466	5806	6135	6461	6786
66	1908	2173	2440	2756	3059	3363	3684	4011	4330	4658	4995	5343	5684	6014	6341	6668
67	1820	2079	2334	2650	2949	3250	3568	3893	421′	4538	4875	5223	5564	5895	6223	6551
68	1732	1986	2234	2545	2840	3138	3452	3775	4093	4419	4755	5103	5444	5776	6105	6435
69	1651	1897	2134	2447	2737	3032	3343	3663	3979	4304	4640	4987	5328	5660	5990	6322
70	1570	1810	2041	2350	2635	2927	3234	3552	3866	4190	4525	4872	5212	5545	5876	6209
71	1497	1724	1949	2260	2539	2828	3131	3446	3759	4081	4414	4761	5100	5433	5764	6099
72	1424	1645	1864	2170	2444	2729	3029	3340	3652	3972	4304	4650	4988	532′	5653	5989
73	1355	1567	1779	2085	2354	2635	2931	3239	3549	3867	4198	4542	4879	5212	5545	5881
74	1287	1493	1699	2000	2264	2541	2833	3138	3446	3762	4092	4435	4771	5104	5437	5773
75	1223	1419	1620	1921	218′	2453	2741	3043	3348	3662	3990	4332	4667	4999	5331	5668
76	1161	135′	1547	1843	2098	2366	2649	2948	3250	3562	3888	4229	4563	4895	5226	5563
77	1100	1284	1474	1770	2020	2284	2563	2858	3157	3467	3790	4130	4462	4793	5124	5461

Продолжение таблицы 40

Номер	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
78	1044	1222	1407	1698	1943	2202	2477	2768	3065	3372	3693	4031	4362	4692	5022	5359
79	988	1161	1340	1632	1872	2126	2396	2684	2977	3281	3600	3936	4265	4594	4924	5260
80	938	1105	1278	1566	1801	2050	2316	2600	2889	3191	3507	3841	4169	4496	4826	5161
81	888	1050	1217	1501	1731	1975	2236	2516	2802	3101	3415	3747	4073	4400	4729	5063
82	839	995	1157	1436	1661	1900	2157	2433	2716	3012	3323	3653	3978	4304	4632	4965
83	790	941	1097	1376	1596	1830	2083	2355	2634	2928	3235	3563	3886	4211	4538	4871
84	746	891	1043	1316	1532	1761	2009	2278	2553	2844	3147	3473	3795	4118	4444	4777
85	702	842	989	1261	1472	1697	1940	2205	2476	2764	3064	3387	3707	4028	4353	4686
86	662	797	940	1207	1412	1633	1871	2133	2399	2684	298′	3302	3619	3939	4262	4595
87	623	753	891	1157	1357	1574	1807	2065	2326	2608	2902	3220	3535	3853	4174	4506
88	588	713	846	1108	1303	1515	1743	1997	2254	2533	2823	3138	3451	3767	4087	4417
89	553	673	801	1061	1251	1459	1683	1932	2185	2460	2746	3059	3369	3684	4002	4331
90	520	636	759	1015	1200	1403	1623	1867	2117	2387	2670	2980	3288	360′	3917	4245
91	488	599	718	973	1153	1351	1567	1807	2052	2318	2594	2905	3210	352′	3835	4162
92	459	566	680	93′	1106	1300	1511	1747	1987	2250	2522	2830	3133	344′	3753	4079
93	43′	533	643	893	1063	1252	1459	1690	1926	2185	2450	2759	3059	3364	3674	3999
94	405	503	609	855	1020	1205	1407	1634	1866	2121	2382	2688	2985	3287	3595	3919
95	380	473	575	819	979	1160	1357	1580	1808	2059	2314	2619	2913	3212	3518	3841
96	357	446	543	783	938	1115	1307	1526	1750	1997	2248	2550	2841	3137	3441	3763
97	334	419	511	747	897	1070	1257	1472	1692	1935	2182	2481	2769	3062	3364	3685
98	311	392	479	711	856	1025	1207	1418	1634	1873	2116	2412	2697	2987	3287	3607
99	288	365	447	677	818	982	1159	1366	1578	1813	2050	2345	2627	2915	3212	3530
100	268	340	418	644	780	939	1111	1314	1522	1754	1987	2278	2557	2843	3138	3454
101	248	316	389	614	746	899	1067	1266	1470	1698	1924	2215	2490	2773	3066	3380
102	230	294	363	584	712	860	1023	1218	1418	1642	1864	2152	2424	2704	2995	3307
103	213	272	337	557	681	824	983	1174	1369	1588	1804	2092	2360	2638	2926	3236

Продолжение таблицы 40

Номер	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
104	197	253	314	530	650	789	943	1130	1321	1535	1748	2032	2297	2572	2858	3166
105	182	234	291	505	621	756	905	1088	1275	1483	1692	1974	2237	2508	2792	3097
106	168	216	270	480	592	723	868	1047	1229	1433	1638	1917	2177	2445	2726	3029
107	154	199	249	458	566	693	834	1009	1187	1384	1585	1863	2119	2384	2662	2963
108	142	184	230	436	540	663	800	971	1145	1338	1534	1809	2062	2324	2599	2897
109	130	169	212	416	517	636	769	936	1105	1292	1484	1758	2007	2266	2538	2834
110	119	155	195	396	494	609	738	901	1066	1249	1437	1707	1953	2208	2478	2771
111	108	142	179	378	473	584	709	868	1027	1206	1390	1659	1901	2153	2420	2710
112	99	130	164	360	452	559	68′	836	991	1165	1346	161′	1849	2098	2362	2650
113	90	118	149	343	431	535	653	804	955	1125	1302	1564	1798	2044	2305	2591
114	81	106	135	326	410	511	625	772	919	1085	1258	1517	1748	1990	2249	2532
115	72	95	121	310	391	489	600	743	883	1045	1215	1473	1700	1939	2195	2475
116	64	85	108	295	373	467	575	714	850	1008	1174	1429	1652	1888	2141	2418
117	56	75	96	28′	356	447	552	685	817	971	1133	1387	1607	1839	2089	2363
118	49	66	85	267	340	427	529	658	786	937	1095	1346	1562	179′	2037	2309
119	42	57	74	255	325	409	508	631	756	903	1057	1307	1519	1745	1988	2257
120	36	49	64	243	310	391	487	606	728	871	102′	1268	1476	1699	1939	2205
121	30	41	54	232	296	374	466	582	700	840	986	1230	1435	1655	1891	2155
122	25	34	45	22′	282	358	447	559	674	810	952	1193	1394	1611	1844	2105
123	20	27	36	211	270	343	428	536	648	780	918	1158	1355	1569	1799	2057
124	15	21	28	20′	258	328	410	515	624	752	887	1123	1317	1527	1754	2009
125	11	15	20	192	247	313	392	494	600	724	856	1090	1281	1487	1711	1963
126	7	10	13	183	236	300	376	475	578	698	827	1058	1245	1448	1668	1918
127	3	5	6	174	225	287	360	456	556	672	798	1026	1210	1409	1626	1873
128	0	0	0	166	214	274	344	437	534	647	770	994	1175	1370	1584	1828
129	—	—	—	158	203	261	328	418	512	622	742	962	1140	133′	1542	1783

Продолжение таблицы 40

Номер	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
130	—	—	—	150	192	248	313	399	490	597	714	930	1105	1292	1500	1738
131	—	—	—	142	182	235	298	380	468	572	686	899	1071	1255	1459	1694
132	—	—	—	134	172	223	283	362	447	548	659	869	1037	1218	1418	1650
133	—	—	—	126	162	211	268	344	426	524	632	841	1005	1183	1380	1607
134	—	—	—	119	153	200	255	328	407	502	607	813	973	1148	1342	1565
135	—	—	—	112	144	189	242	312	388	480	582	786	943	1115	1305	1524
136	—	—	—	106	136	179	230	297	371	460	559	760	913	1082	1269	1484
137	—	—	—	100	128	169	218	283	354	440	536	735	885	105′	1234	1445
138	—	—	—	95	121	160	207	270	338	42′	514	710	857	1020	1199	1407
139	—	—	—	90	114	151	196	257	322	403	492	687	830	990	1166	1369
140	—	—	—	85	108	143	186	245	307	386	472	664	804	960	1133	1333
114	—	—	—	80	102	135	176	233	293	369	452	643	779	932	1102	1297
142	—	—	—	76	97	128	167	222	280	353	433	622	754	904	1071	1263
143	—	—	—	72	92	121	158	21′	267	337	415	602	73′	878	1041	1229
144	—	—	—	69	87	115	150	201	255	323	398	582	708	852	1012	1197
145	—	—	—	66	82	109	142	191	243	309	381	562	685	826	983	1165
146	—	—	—	63	77	103	135	181	231	295	364	543	663	801	954	1134
147	—	—	—	60	73	97	128	172	219	28′	348	525	642	777	926	1103
148	—	—	—	57	69	92	12′	163	209	268	333	507	62′	753	899	1073
149	—	—	—	54	65	87	114	155	199	255	318	490	60′	731	872	1043
150	—	—	—	51	62	82	108	147	189	243	304	473	58′	709	847	1015
151	—	—	—	48	59	77	102	139	179	23′	290	457	563	687	822	987
152	—	—	—	46	56	73	97	132	170	220	277	442	545	666	798	960
153	—	—	—	44	53	69	92	125	161	209	264	427	528	645	774	933
154	—	—	—	42	50	65	87	119	153	199	252	412	511	625	751	907
155	—	—	—	40	47	6′	82	113	145	189	240	398	495	605	728	882

Продолжение таблицы 40

Номер	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
156	—	—	—	38	45	58	78	107	138	180	229	385	479	586	707	858
157	—	—	—	36	43	55	74	101	131	171	218	373	463	567	686	834
158	—	—	—	34	41	52	70	96	124	163	208	361	448	550	666	811
159	—	—	—	33	39	49	66	91	117	155	198	349	433	533	646	788
160	—	—	—	32	37	46	62	86	111	147	188	337	419	516	627	766
161	—	—	—	31	35	43	58	81	105	139	178	325	405	499	608	744
162	—	—	—	30	33	40	54	76	99	131	168	313	391	482	589	722
163	—	—	—	29	31	37	50	71	93	123	158	301	377	465	570	700
164	—	—	—	28	29	35	47	66	87	116	149	290	364	449	552	679
165	—	—	—	27	27	33	44	62	81	109	140	279	351	433	534	658
166	—	—	—	26	26	31	41	58	76	102	132	269	338	418	517	638
167	—	—	—	25	25	29	38	54	71	95	124	259	326	403	500	618
168	—	—	—	24	24	27	35	50	66	89	116	249	314	389	484	599
169	—	—	—	23	23	25	32	46	61	83	108	240	302	375	468	581
170	—	—	—	22	22	23	30	43	57	77	101	231	291	362	453	563
171	—	—	—	21	21	21	28	40	53	72	94	222	280	349	438	545
172	—	—	—	20	20	20	26	37	49	67	87	214	270	337	424	528
173	—	—	—	19	19	19	24	34	45	62	81	206	260	325	410	511
174	—	—	—	18	18	18	22	31	42	57	75	199	251	314	397	495
175	—	—	—	17	17	17	20	28	39	52	69	192	242	303	384	480
176	—	—	—	16	16	16	18	26	36	48	64	185	234	293	372	465
177	—	—	—	15	15	15	16	24	33	44	59	178	226	283	360	451
178	—	—	—	14	14	14	14	22	30	40	54	171	218	273	348	437
179	—	—	—	13	13	13	13	20	27	36	49	165	210	263	336	423
180	—	—	—	12	12	12	12	18	24	32	44	159	202	254	325	410
181	—	—	—	11	11	11	11	16	21	28	39	153	195	245	314	397

Продолжение таблицы 40

Номер	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
182	—	—	—	10	10	10	10	14	19	25	35	148	188	236	303	384
183	—	—	—	9	9	9	9	12	17	22	31	143	18′	227	293	372
184	—	—	—	8	8	8	8	10	15	19	27	138	174	219	283	360
185	—	—	—	7	7	7	7	8	13	16	23	133	168	211	273	348
186	—	—	—	6	6	6	6	6	11	13	19	128	162	204	264	337
187	—	—	—	5	5	5	5	5	9	10	15	123	156	197	255	326
188	—	—	—	4	4	4	4	4	7	8	12	119	150	190	246	315
189	—	—	—	3	3	3	3	3	5	6	9	115	144	183	237	305
190	—	—	—	2	2	2	2	2	3	4	6	111	139	177	229	295
191	—	—	—	1	1	′	1	1	′	2	3	107	134	171	221	285
192	—	—	—	0	0	0	0	0	0	0	0	103	129	165	213	275
193	—	—	—	—	—	—	—	—	—	—	—	99	124	159	205	265
194	—	—	—	—	—	—	—	—	—	—	—	95	119	153	197	255
195	—	—	—	—	—	—	—	—	—	—	—	91	114	147	189	245
196	—	—	—	—	—	—	—	—	—	—	—	87	109	141	181	236
197	—	—	—	—	—	—	—	—	—	—	—	83	104	135	174	227
198	—	—	—	—	—	—	—	—	—	—	—	80	100	130	167	219
199	—	—	—	—	—	—	—	—	—	—	—	77	96	125	160	211
200	—	—	—	—	—	—	—	—	—	—	—	74	92	120	154	203
201	—	—	—	—	—	—	—	—	—	—	—	71	88	115	148	195
202	—	—	—	—	—	—	—	—	—	—	—	68	84	110	142	188
203	—	—	—	—	—	—	—	—	—	—	—	65	80	105	136	181
204	—	—	—	—	—	—	—	—	—	—	—	63	77	101	131	174
205	—	—	—	—	—	—	—	—	—	—	—	61	74	97	126	167
206	—	—	—	—	—	—	—	—	—	—	—	59	7′	93	121	161
207	—	—	—	—	—	—	—	—	—	—	—	57	68	89	116	155

Продолжение таблицы 40

Номер	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
208	—	—	—	—	—	—	—	—	—	—	—	55	65	85	111	149
209	—	—	—	—	—	—	—	—	—	—	—	53	62	81	106	143
210	—	—	—	—	—	—	—	—	—	—	—	51	59	77	101	137
211	—	—	—	—	—	—	—	—	—	—	—	49	56	74	97	131
212	—	—	—	—	—	—	—	—	—	—	—	47	54	71	93	126
213	—	—	—	—	—	—	—	—	—	—	—	45	52	68	89	121
214	—	—	—	—	—	—	—	—	—	—	—	43	50	65	85	116
215	—	—	—	—	—	—	—	—	—	—	—	41	48	62	81	111
216	—	—	—	—	—	—	—	—	—	—	—	40	46	59	77	106
217	—	—	—	—	—	—	—	—	—	—	—	39	44	56	73	101
218	—	—	—	—	—	—	—	—	—	—	—	38	42	53	70	97
219	—	—	—	—	—	—	—	—	—	—	—	37	40	51	67	93
220	—	—	—	—	—	—	—	—	—	—	—	36	38	49	64	89
221	—	—	—	—	—	—	—	—	—	—	—	35	36	47	61	85
222	—	—	—	—	—	—	—	—	—	—	—	34	34	45	58	81
223	—	—	—	—	—	—	—	—	—	—	—	33	33	43	55	77
224	—	—	—	—	—	—	—	—	—	—	—	32	32	41	52	73
225	—	—	—	—	—	—	—	—	—	—	—	31	31	39	49	69
226	—	—	—	—	—	—	—	—	—	—	—	30	30	37	46	65
227	—	—	—	—	—	—	—	—	—	—	—	29	29	35	43	61
228	—	—	—	—	—	—	—	—	—	—	—	28	28	33	40	58
229	—	—	—	—	—	—	—	—	—	—	—	27	27	31	37	55
230	—	—	—	—	—	—	—	—	—	—	—	26	26	29	35	52
231	—	—	—	—	—	—	—	—	—	—	—	25	25	27	33	49
232	—	—	—	—	—	—	—	—	—	—	—	24	24	25	31	46
233	—	—	—	—	—	—	—	—	—	—	—	23	23	23	29	43

Окончание таблицы 40

Номер	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
234	—	—	—	—	—	—	—	—	—	—	—	22	22	22	27	40
235	—	—	—	—	—	—	—	—	—	—	—	21	2'	21	25	37
236	—	—	—	—	—	—	—	—	—	—	—	20	20	20	23	34
237	—	—	—	—	—	—	—	—	—	—	—	19	19	19	21	32
238	—	—	—	—	—	—	—	—	—	—	—	18	18	18	19	30
239	—	—	—	—	—	—	—	—	—	—	—	17	17	17	17	28
240	—	—	—	—	—	—	—	—	—	—	—	16	16	16	16	26
241	—	—	—	—	—	—	—	—	—	—	—	15	15	15	15	24
242	—	—	—	—	—	—	—	—	—	—	—	14	14	14	14	22
243	—	—	—	—	—	—	—	—	—	—	—	13	13	13	13	20
244	—	—	—	—	—	—	—	—	—	—	—	12	12	12	12	18
245	—	—	—	—	—	—	—	—	—	—	—	11	11	1'	11	16
246	—	—	—	—	—	—	—	—	—	—	—	10	10	10	10	14
247	—	—	—	—	—	—	—	—	—	—	—	9	9	9	9	12
248	—	—	—	—	—	—	—	—	—	—	—	8	8	8	8	10
249	—	—	—	—	—	—	—	—	—	—	—	7	7	7	7	8
250	—	—	—	—	—	—	—	—	—	—	—	6	6	6	6	6
251	—	—	—	—	—	—	—	—	—	—	—	5	5	5	5	5
252	—	—	—	—	—	—	—	—	—	—	—	4	4	4	4	4
253	—	—	—	—	—	—	—	—	—	—	—	3	3	3	3	3
254	—	—	—	—	—	—	—	—	—	—	—	2	2	2	2	2
255	—	—	—	—	—	—	—	—	—	—	—	'	1	1	'	1
256	—	—	—	—	—	—	—	—	—	—	—	0	0	0	0	0

6.6.2.4 Кодирование остаточных *LSB* и концевых частей

LSB и концевые части передаются способом чередования (то есть если *msb [i]! = tail_code*, передаем *lsb [i]*, иначе *tail [i]*) для всех остатков в подблоке.

LSB передаются непосредственно, используя *k* битов, в то время как концевые части кодируются, используя коды Райса с параметром *s*, как описано в 6.6.1.

6.7 Объединенное кодирование пар каналов

Объединенное кодирование пар каналов идентично объединенному кодированию стереопары. Таким образом, используем термины «левый канал» и «правый канал», чтобы сослаться на два канала любой пары каналов, независимо от того, переносит ли она подлинный сигнал стерео или два других канала многоканального сигнала.

Когда декодируемый блок, который обычно представлял левый канал (*L*) или правый канал (*R*), содержит разностный сигнал объединенного стерео (*D = R – L*, обозначенный *js_block*), исходные данные канала должны быть явно восстановлены. Если первый канал содержит разностный сигнал, левый канал был заменен и может быть восстановлен, используя *L = R – D* для всех выборок этого блока. Если второй канал содержит разностный сигнал, правый канал был заменен и восстанавливается, используя *R = D + L* для всех выборок. Если *block_switching* включено, это относится к любой паре синхронизируемых блоков пары каналов, то есть любой блок может содержать разностный сигнал или исходный сигнал канала. С точки зрения прогноза разностные сигналы обрабатываются как нормальные сигналы. Если фильтр прогноза использует выборки из предыдущего блока, то они должны быть разностными значениями тех же самых двух каналов в этом предыдущем блоке. То же самое применяется к инверсному фильтру прогноза декодера.

6.8 Многоканальное кодирование (*MCC*)

Многоканальное кодирование использует адаптивно взвешенное межканальное вычитание, чтобы далее уменьшить амплитуды остатков прогноза.

6.8.1 Декодирование и восстановление одиночных каналов

Наборы информации о межканальной зависимости, включая *master_channel_index*, *weighting_factor* и *time difference flag*, декодируются для каждого канала. Числами итераций управляет *stop flag*. Кроме того, имеется бит *js_switch*, который активирует простое кодирование входных сигналов объединенного стерео вместо многоканального кодирования остаточных сигналов прогноза *LPC*, даже когда используется режим межканальной корреляции.

Если флаг *time_difference_flag* является нулем, выполняется синфазное взвешенное дополнение с тремя ответвлениями, как показано в следующем псевдокоде.

```
INT64 u;
for(smpl=1 ;smpl<N-1; smpl++)
{
        u = (INT64)MCCgain[0]*d_masterchannel[smpl-1]
        +(INT64)MCCgain[1]*d_masterchannel[smpl]
        +(INT64)MCCgain[2]*d_masterchannel[smpl+1];

        d[smpl] += (long) ((u+64)>>7);
}
```

d: целое число для остаточного сигнала

MCCgain: величина усиления $\gamma(i)$*128

Если *time difference flag* равен единице, выполняется взвешенное дополнение с шестью ответвлениями с декодированным параметром временного различия *TDL*, как показано в следующем псевдокоде.

```
INT64 u;
if(TDL>0) {ss=1; se=N-TDL-1;}
else {ss=-TDL+1; se=N-1;}
for(smpl=ss ;smpkse; smpl++)
{
        u = (INT64)MCCgain[0]*d_masterchannel[smpl-1]
        + (INT64)MCCgain[1]*d_masterchannel[smpl]
        + (INT64)MCCgain[2]*d_masterchannel[smpl+1]
```

```
        + (INT64)MCCgain[3]*d_masterchannel[smpl+TDL-1]
        + (INT64)MCCgain[4]*d_masterchannel[smpl+TDL]
        + (INT64)MCCgain[5]*d_masterchannel[smpl+TDL+1];

        d[smpl] += (long) ((u+64)>>7);
    }
```

d: целое число для остаточного сигнала

MCCgain: величина усиления $\gamma(i)$*128

TDL: величина задержки временного различия: τ

Диапазон *TDL* зависит от частоты дискретизации, как показано в таблице 41. Имеется бит знака для *TDL*, чтобы поддерживать и положительные, и отрицательные различия относительно сигнала кодирования. Чтобы избежать конфликтов с синфазным взвешиванием, *TDL* должно быть больше 3 или меньше – 3.

Таблица 41 — Диапазон задержки τ

Диапазон поиска $\tau(i)$	Начало	Конец
Частота < 96 кГц	3	31+3
Частота ≥ 96 кГц	3	63+3
Частота ≤ 192 кГц	3	127+3

Значение усиления $\gamma(i)$*128 квантуется согласно таблице 42. Фактические индексы значений усиления кодируются с использованием кодов Райса с параметром 1 и смещением 16, за исключением *weighting_factor* [1]. Индекс *weighting_factor* [1] декодируется, используя коды Райса с параметром 2 со смещением 14.

Таблица 42 — Код Райса факторов межканального взвешивания $\gamma(i)$*128

Индекс фактора взвешивания	Для фактора взвешивания [1]	Для других факторов	$\gamma(i)$*128	Индекс фактора взвешивания	Для фактора взвешивания [1]	Для других факторов	$\gamma(i)$*128
0	111111001	1111111111111100	204	16	1010	01	0
1	111111000	111111111111100	192	17	1011	101	– 12
2	11111001	11111111111100	179	18	11010	1101	– 25
3	11111000	1111111111100	166	19	11011	11101	– 38
4	1111001	111111111100	153	20	111010	111101	– 51
5	1111000	111111111100	140	21	111011	1111101	– 64
6	111001	11111111100	128	22	1111010	11111101	– 76
7	111000	1111111100	115	23	1111011	111111101	– 89
8	11001	111111100	102	24	11111010	1111111101	– 102
9	11000	11111100	89	25	11111011	11111111101	– 115
10	1001	1111100	76	26	111111010	111111111101	– 128
11	1000	111100	64	27	111111011	1111111111101	– 140
12	001	11100	51	28	1111111010	11111111111101	– 153
13	000	1100	38	29	1111111011	111111111111101	– 166
14	010	100	25	30	11111111010	1111111111111101	– 179
15	011	00	12	31	11111111011	11111111111111101	– 192

6.8.2 Последовательное декодирование для многоканальных сигналов

Последовательность остаточных выборок прогноза в одном канале и одном блоке обрабатывается как вектор, $e = (e_0, e_1, \ldots, e_{n-1})^T$, где n представляет число выборок во фрейме. Пусть e_i будет вектором остатка прогноза основного канала, где j определяется *master_channel_index*, и ei является вектором остатка прогноза кодированного (ведомого) канала. Остаточный сигнал ведомого канала восстанавливается путем добавления сигнала основного канала с коэффициентами умножения γ (фактор взвешивания) в пределах от – 192/128 до 204/128.

$$e^i = \tilde{e}^i + \sum_{m=1}^{1} \gamma_{m,k}^{i,j} e^j.$$

Если *stop_flag* включено, никакая межканальная информация для этого канала не следует. Если *stop_flag* выключено, межканальная информация следует, и это позволено неоднократно. Таким образом, синтаксису разрешено иметь многократные этапы межканальной информации. Число этапов должно быть общим для всех каналов. Если *master_channel_index* идентично номеру канала на всех дальнейших этапах, что означает, что взвешенное вычитание более не активируется, элементы синтаксиса для дальнейших этапов пропускаются для этого канала. В результате *stop_flag* устанавливается в конце эффективного набора информации.

Для того чтобы восстановить остаточные сигналы из многоступенчатой межканальной информации, процесс должен запуститься с правого набора информации. Промежуточный вектор остатка в *i*-м канале и *k*-м этапе, e_{k-1}^i, восстанавливается следующим образом, используя вектор основного канала e_{k-1}^j и $\gamma_{k-1}^{i,j}$ для *k*-го этапа.

$$e_{k-1}^i = e_k^i + \sum_m \gamma_{m,k-1}^{i,j} e_{k-1}^j,$$

где $\sum_m \gamma_{m,k-1}^{i,j}$ либо 3-отводная, либо 6-отводная фильтрация.

Пример потока битов для пяти каналов и трех этапов итерации показан в таблице 43. У канала 0 есть три элемента синтаксиса, соответствующие этапам. Реконструкцию следует выполнять с последнего этапа, то есть третьего этапа. У третьего этапа имеется только *stop_flag*, поэтому работа начинается со второго этапа обращением к e_2^i ($i = 0, 1, \ldots, 4$), которые берутся из декодирования энтропии.

На втором этапе у канала 0 как основной канал есть канал 4 с коэффициентами взвешивания γ[14], γ[12] и γ[15]. Таким образом, $e_1^0 = e_2^0 + \Sigma\gamma \cdot e_1^4$, где $e_1^1 = e_2^1$, $e_1^2 = e_2^2$, $e_1^3 = e_2^3$, $e_1^4 = e_2^4$, с этого момента другие каналы не нуждаются ни в каких операциях на втором этапе итерации.

На первом этапе у канала 0 как основной канал служит канал 2 с коэффициентами взвешивания γ[13], γ[10] и γ[15]. Это означает $e_0^0 = e_1^0 + \Sigma\gamma \cdot e_0^2$. Однако e_0^2 не готово. Этот процесс должен быть приостановлен до восстановления e_0^2. У канала 1 основным каналом служит канал 4 с коэффициентами умножения γ[13], γ[11] и γ[12]. Таким образом, $e_0^1 = e_1^1 + \Sigma\gamma \cdot e_0^4$. У канала 2 основной канал — канал 1 со значением временного различия «5» и коэффициентами умножения γ[16], γ[15], γ[16], γ[13], γ[12] и γ[14]. Таким образом, $e_0^2 = e_1^2 + \Sigma\gamma \cdot e_0^1$. Теперь, когда канал 2 был восстановлен, может быть восстановлен канал 0. Другие каналы не имеют никаких операций на первом этапе. Таким образом, $e_0^3 = e_1^3$, $e_0^4 = e_1^4$.

Таблица 43 — Пример потока битов для межканальной информации для пяти каналов и трех этапов

Этап 1					
№ канала	*stop_flag (S)*	*master_channel_index (M)*	*time_difference_flag (T)*	Коэффициент взвешивания (*W*)	*time_difference_value(L)*
0	0	2	0	13-10-15	—
1	0	4	0	13-11-12	—
2	0	1	1	16-15-16-13-12-14	5
3	1	—	—	—	—
4	1	—	—	—	—

Окончание таблицы 43

Этап 2					
№ канала	*stop_flag (S)*	*master_channel_ index (M)*	*time_difference_flag (T)*	Коэффициент взвешивания (*W*)	*time_difference_ value(L)*
0	0	4	0	14-12-15	—
1	1	—	—	—	—
2	1	—	—	—	—
3	—	—	—	—	—
4	—	—	—	—	—
Этап 3					
№ канала	*stop_flag (S)*	*master_channel_ index (M)*	*time_difference_flag (T)*	Коэффициент взвешивания (*W*)	*time_difference_ value(L)*
0	1	—	—	—	—
1	—	—	—	—	—
2	—	—	—	—	—
3	—	—	—	—	—
4					

Мы можем восстановить вектор только тогда, когда вектор основного канала уже восстановлен на этом этапе.

6.9 Расширение для данных с плавающей точкой

В дополнение к целочисленным аудиосигналам *MPEG*-4 *ALS* также поддерживает сжатие аудиосигналов без потерь в 32-разрядном формате с плавающей точкой *IEEE*.

6.9.1 Кодер для данных с плавающей точкой

Если входной сигнал является 32-битовым с плавающей точкой, входные значения разлагаются на три части: предполагаемый общий множитель *A*, усеченная целочисленная последовательность сомножителя *Y* и разностный сигнал *Z*. Та же самая схема сжатия как для нормального целочисленного ввода применяется для усеченной и нормализованной целочисленной последовательности сомножителя. Когда предполагаемый общий множитель *A* равняется 1,0, извлекается и упаковывается только необходимый код длины мантиссы для последовательности различия. Необходимая длина слова уникально определяется значением соответствующего целого числа. 23 или меньше битов сигнала мантиссы-различия *Z* кодируются, используя модуль сжатия *Masked-LZ*, кроме тех случаев, когда *Y* равняется 0. Если целое число *Y* равняется 0, все 32 бита данных *x* кодируются с модулем отдельно. Во всех случаях для усеченной целочисленной последовательности *Y* используется модуль сжатия *ALS*.

6.9.2 Декодер для данных с плавающей точкой

В случае данных с плавающей точкой восстанавливается целочисленная последовательность сомножителя *Y*, и множитель *A* умножается, чтобы получить последовательность с плавающей точкой (*Y***A*). Для операции округления умножения используется округляющийся режим «*round to nearest, to even when tie*». Последовательность различия декодируется модулем распаковки *Masked-LZ* и преобразовывается в последовательность формата с плавающей точкой *Z*. Часть *A* и часть *B* декодируются отдельно и выравниваются для реконструкции. Если множитель *A* равняется 1,0, последовательность различия декодируется, используя информацию о длине слова, которая определяется из значения соответствующей целочисленной величины. Дополнительные биты более, чем необходимая длина в битах, отключаются (выбрасываются), так как они — фиктивные биты, добавленные кодером. Обе последовательности, (*Y***A*) и *Z*, суммируются, чтобы генерировать выходную последовательность с плавающей точкой.

6.9.3 Декодирование потока битов для данных с плавающей точкой

В случае данных с плавающей точкой необходимо декодировать последовательность различий так же, как целочисленный поток битов. Можно воспользоваться тем, что граница слова каждой выборки различия уникально определяется абсолютным значением связанного усеченного целого числа. На заключительном этапе процесса декодирования обе последовательности от целого числа и мантиссы различия суммируются, чтобы восстановить последовательность в формате с плавающей точкой.

6.9.3.1 Декодирование размера данных элемента различия

В процессе, названном *frame_data,* декодируется *num_bytes_diff_float*. Следующие байты *num_bytes_diff_float* являются сжатыми данными различия.

6.9.3.2 Декодирование элемента различия

6.9.3.2.1 Инициализация переменных

В первом фрейме или фреймах произвольного доступа некоторые переменные должны быть инициализированы следующим образом.

last_acf_mantissa [c] устанавливается в 0. *last_shift_value [c]* устанавливается в 0.

В *FlushDict* () флаг *freeze_flag* устанавливается в 0, *code_bits* устанавливается в 9, *bump_code* устанавливается в 511, *next_code* устанавливается в 258.

Таблица 44 — Начальные значения

Переменная	Начальное значение	Описание
last_acf_mantissa[c]	0	Это означает общий множитель 1,0
last_shift_value[c]	0	
freeze_flag	0	Инициализировано в *FlushDict*().
code_bits	9	Инициализировано в *FlushDict*().
bump_code	511	Инициализировано в *FlushDict*().
next_code	258	Инициализировано в *FlushDict*().

6.9.3.2.2 Параметры нормализации

Перед декодированием разностного сигнала параметры нормализации должны декодироваться следующим способом.

Во-первых, декодируется *use_acd*. Если *use_acf* не 0, для каждого канала декодируется *acf_flag [c]*. Если *acf_flag [c]* не 0, декодируется *acf_mantissa [c]*. Если *acf_flag [c]* равняется 0, то же самое значение *last_acf_mantissa [c]* устанавливается в *acf_mantissa [c]*. Когда *use_acf* равняется 0, *last_acf_mantissa [c]* и *acf_mantissa [c]* устанавливаются в 0. Это означает, что общий множитель инициализируется в 1,0.

Кроме того, для каждого канала декодируются *highest_byte* [*c*], *partA_flag* [*c*] и *shift_amp* [*c*]. Когда *partA_flag* [*c*] равно 0, значения всех выборок в части *A* являются всеми нулями или нет никакой выборки, для которой соответствующее усеченное целое число равняется 0. Если *shift_amp* [*c*] равняется 0, значение *last_shift_value* [*c*] копируется в *shift_value* [*c*]. Иначе *shift_value* [*c*] декодируется из потока битов. Заключительная выходная последовательность с плавающей точкой, восстановленная из усеченного целого числа, денормализовывается добавлением *shift_value* [*c*] к экспоненте выходных данных. В конце концов *last_acf_mantissa* [*c*] и *last_shift_value* [*c*] обновляются текущими величинами *acf_mantissa* [*c*] и *shift_value* [*c*].

6.9.3.2.3 Различия мантиссы (когда *acf_mantissa [c]* равняется 0)

Если *acf_mantissa* [*c*] равняется 0, мантисса различия восстанавливается следующим способом.

Последовательность различий во фрейме упаковывается. У каждой выборки имеется уникальная длина слова фактической информации, где длина определяется значением соответствующего целого числа. Каждое значение различия мантиссы *D*[*c*][*n*] восстанавливается следующим образом.

Поток битов для восстановления значения различия разделяется на две части (часть *A* и часть *B*). Часть *A* содержит выборки, необходимые для кодирования всех 32 битов. Это выборки, для которых соответствующее усеченное целое число *y* равняется 0. Если усеченное целочисленное значение является нулем, получаются исходные данные с плавающей точкой.

Если *partA_flag* [*c*] равняется 0, все значения в части *A* являются нулями или в части *A* нет никаких выборок. Когда *partA_flag* [*c*] не является нулем, в части *A* есть выборки и должен быть считан *compressed_flag*. Если *compressed_flag* [*c*] равняется 0, все потоки битов упаковываются несжатыми. Если *compressed_flag* [*c*] равняется 1, выборки части *A* сжимаются, используя схему сжатия *Masked-LZ*. В этом случае распаковка *Masked-LZ* применяется для потока битов в части *A*.

Часть *B* содержит выборки, для которых соответствующее усеченное целое число *y* не равно 0. Перед декодированием потока битов в части *B* должно быть считано *compressed_flag* [*c*] для части *B*. Если *compressed_flag* [*c*] равняется 0, поток битов упаковывается несжатым. Если *compressed_flag* [*c*] равняется 1, выборки части *B* сжимаются, используя схему сжатия *Masked-LZ*. В этом случае для потока битов в части *B* применяется распаковка *Masked-LZ*.

Необходимые биты этих выборок различаются от 0 до 23 битов в зависимости от *highest_byte* [*c*], *acf_mantissa* [*c*] и соответствующего усеченного целого числа *y*.

Самая высокая длина различия в битах *nbits*[*c*][*n*], которая будет кодироваться, определяется следующим образом:

*nbits[c][n]= min (word_length[c][n], highest_byte*8).*

где *word_length* [*c*] [*n*] показано в таблице 45.

«*highest_byte* [*c*]» обозначает наибольшую длину байта всех значений различия мантиссы во фрейме. «*highest_byte* [*c*]» равно 0, если у исходных данных с плавающей точкой имеется точность 16-битового целого числа.

Таблица 45 — Необходимая длина слова для различия мантиссы (когда *acf_mantissa[c]* равняется 0)

Состояние *acf_mantissa[c]*	Диапазон абсолютного целого значения |*x*|	*word_length*
acf_mantissa [*c*]==0	|*x*| = 0	32
(общий множитель равен 1,0)	$2^n \le \|x\| < 2^{(n+1)}$ $(0 \le n < 23)$	$23 - n$

6.9.3.2.4 Различия мантиссы (когда *acf_mantissa* [*c*] не равно 0)

Если *acf_mantiisa* [*c*] не 0, мантисса различия восстанавливается следующим способом.

Последовательность различий во фрейме упаковывается. У каждой выборки есть уникальная длина слова фактической информации, где длина определяется значением соответствующего целого числа. Каждое значение различия мантиссы *D*[*c*][*n*] восстанавливается следующим образом.

Поток битов для реконструкции значений различия разделяется на две части (часть *A* и часть *B*). Часть *A* содержит выборки, необходимые для кодирования всех 32 битов. Это выборки, для которых соответствующее усеченное целое число *y* равняется 0. Если усеченное целочисленное значение является нулем, получаются исходные данные с плавающей точкой.

Если *partA_flag* [*c*] равняется 0, все значения в части *A* являются нулями или в части *A* нет никаких выборок. Когда *partA_flag* [*c*] не является нулем, в части *A* есть выборки, и *compressed_flag* должен быть считан. Если *compressed_flag* [*c*] равняется 0, все потоки битов упаковываются несжатые. Если *compressed_flag* [*c*] равняется 1, выборки части *A* сжимаются, используя схему сжатия *Masked-LZ*. В этом случае для потока битов в части *A* применяется распаковка *Masked-LZ*.

Часть *B* содержит выборки, для которых соответствующее усеченное целое число *y* не равно 0. *compressed_flag* [*c*] для части *B* должно быть считано до декодирования потока битов в части *B*. Если *compressed_flag* [*c*] равняется 0, поток битов упаковывается несжатым. Если *compressed_flag* [*c*] равняется 1, выборки части *B* сжимаются, используя схему сжатия *Masked-LZ*. В этом случае для потока битов в части *B* применяется распаковка *Masked-LZ*. Необходимые биты этих выборок различаются от 0 до 23 битов в зависимости от *highest_byte* [*c*], *acf_mantissa* [*c*] и соответствующего усеченного целого числа *y*.

Наибольшая длина в битах различия *nbits* [*c*] [*n*], которое будет кодировано, определяется следующим образом:

*nbits[c][n]= min (word_length[c][n], highest_byte[c]*8),*

где *word_length* [*c*] [*n*] показано в таблице 46.

«*highest_byte* [*c*]» обозначает наибольшую длину байта всех значений мантиссы различия во фрейме. «*highest_byte* [*c*]» равно 0, если у исходных данных с плавающей точкой имеется точность 16-битого целого числа.

Таблица 46 — Необходимая длина слова для различия мантиссы (когда *acf_mantissa[c]* не равно 0)

Состояние *acf_mantissa*[c]	Диапазон абсолютного целого значения |y|	*word_length*
acf_mantissa[c]==0	|y| = 0	32
(общий множитель не равен 1,0)	|y| != 0	23

6.9.3.2.5 Распаковка *Masked-LZ*

Сжатие *Masked-LZ* является видом, основанном на словаре схемы сжатия.

Это весьма подобно другим разновидностям сжатия *Lempel-Ziv*, таким как схема сжатия *LZW*, то есть существует словарь строк, которые встретились ранее. Ищется самая длинная строка соответствия входных символов, используя строку, сохраненную в словаре.

Диапазон *code_bits* меняется от 9 до 14 битов, так как индекс словаря кодируется как 9—15-битовый в зависимости от числа записей, сохраненных в словаре. Для синхронизации словаря в кодере и декодере используются коды *Bump* и код *Flush* (таблица 47). *bump code* сначала устанавливается в 511 и увеличивается в зависимости от записей, хранящихся в словаре.

Декодер читает (*code_bits*) биты из потока битов, и получают *string_code*. Когда *string_code* является *FLUSH_CODE* или *MAX_CODE*, словарь и переменные, связанные со словарем, должны быть повторно инициализированы начальными значениями (таблица 44). Когда *string_code* является *FREEZE_CODE*, декодер прекращает добавлять новые записи в словарь, пока он не встречается с фреймом произвольного доступа или не получает *FLUSH_CODE*.

Таблица 47 — Специальные индексные коды замаскированных *LZ*

Диапазон *string_code*	Коды специального индекса	Величина
9<= *code bits* <= 15	*FLUSH_CODE*	256
(0<= *stringCode* < 32768)	*FREEZE_CODE*	257
	FIRST_CODE	258
	BUMP_CODE	$(2^{code_bits}) - 1$
	MAX_CODE	$2^{15} - 1$

Алгоритм для распаковки *Masked-LZ* дается ниже.

```
// Masked-LZ decompression.

long n, i, readBits, string_code, last_string_code, charCode;
unsigned long dec_chars

last_string_code = -1;
for ( dec_chars = 0; dec_chars < nchars; ) {
        readBits = inputCode( &string_code, code_bits );
        if ( string_code == FLUSH_CODE ) || ( string_code == MAX_CODE )) {
                FlushDict();
                last_string_code = -1;
}
else if ( string_code == FREEZE_CODE ) {
        freeze_flag = 1;
}
else if ( string_code == bump_code ) {
        code_bits++;
        bump_code = bump_code * 2 + 1;
}
else {
        if ( string_code >= next_code ) {
            dec_chars += decodeString( &dec_buf[dec_chars], last_string_code, &charCode );
```

```
                dec_chars += decodeString( &dec_buf[dec_chars], charCode, &charCode );
                setNewEntryToDict( next_code, last_string_code, charCode );
                next_code ++;
            }
            else {
                dec_chars += decodeString(&dec_buf[dec_chars], string_code, &charCode);
                if ( ( dec_chars <= nchars ) && ( last_string_code != -1 ) && ( freeze_flag == 0 ) ) {
                setNewEntryToDict( next_code, last_string_code, charCode );
                next_code ++;
            }
        }
        last_string_code = string_code;
    }
}
```

Примечание — Функция *inputCode*() читает число битов «*code_bits*» из закодированного потока битов и возвращает *string_code*. Функция *decodeString* () берет *string_code* в качестве входного значения и возвращает декодированную строку символов, связанную с *string_code*, путем поиска в словаре, число символов в декодируемой строке и код первого символа строки *charCode*. Функция *setNewEntryToDict*() берет *last_string_code* и *charCode* и устанавливает их в свободную запись словаря, представленную как *next_code*. Функция *FlushDict*() очищает все записи словаря и инициализирует связанные значения словаря. В вышеупомянутом псевдокоде «*dec_buf*» является буфером для сохранения декодируемых символов, а «*nchars*» является числом символов, которые должны декодироваться. В *FlushDict*() «*code_bits*» устанавливается в 9, «bump_code» устанавливается в 511 и «*freeze_flag*» устанавливается в 0.

После того как входные символы декодируются из *string_code*, эти символы преобразовываются в значения различия мантиссы, *D*[*c*][*n*].

Если *nbits* [*c*] [*n*], которое является размером слова символов, используемых в модуле *Masked-LZ*, не кратно 8, это означает, что на стороне кодера были добавлены фиктивные биты.

Дополнительные биты более, чем *nbits*[*c*][*n*], вырезаются (выбрасываются), используя следующий алгоритм:

```
// reconstruction of difference values from decoded characters.
long n, i, nbits_aligned;
unsigned long acc, j;
j = 0.
for (n = 0; n < frame_length; n++) {
    if ( !int_zero[c][n] ) {
        if ( nbits[c][n] % 8) > 0)
            nbits_aligned = 8 * ((unsigned int )(nbits[c][n] / 8) + 1);
        else
            nbits_aligned = nbits[c][n];
        acc = 0;
        for ( i = 0; i < nbits_aligned / 8; i++ )
            acc = ( acc << 8 ) + dec_buf[j++];
        acc >>= ( nbits_aligned — nbits[c][n] ); // throw away dummy bits added by the encoder.
        D[c][n] = acc;
    }
}
```

Примечание — «*int_zero*» является истиной, если соответствующее усеченное целое число равно 0. «*nbits* [*c*] [*n*]» является необходимой длиной слова для различия мантиссы. «*dec_buf* [ ]» является буфером для сохранения декодируемых символов.

6.9.3.3 Реконструкция данных с плавающей точкой

6.9.3.3.1 Реконструкция данных с плавающей точкой (когда *acf_mantissa* [*c*] равно 0)

Усеченные целочисленные значения преобразовываются в 32-разрядные данные с плавающей запятой *F[c][n]* согласно определению 32-разрядного формата с плавающей точкой *IEEE* с нормализацией с точки зрения максимального целочисленного значения (2^{23}).

F[c][n] = (float) (truncated_integer_value)$^*2^{-23}$

Поле экспоненты *F[c][n]* является *shift_value[c]*, если *shift_amp[c]* равно 1.

Если целое число «0», кодированные данные с плавающей точкой используются как для окончательного результата. Иначе восстановленное значение различия мантиссы *D*[*c*][*n*] добавляется к мантиссе данных с плавающей точкой *F*[*c*][*n*], преобразованных из целочисленного значения. Нет никакой необходимости изменять ни знак, ни поле экспоненты *F*[*c*][*n*].

6.9.3.3.2 Реконструкция данных с плавающей точкой (когда *acf mantissa*[*c*] не равно 0)

Усеченные целочисленные значения преобразовываются в 32-разрядные данные с плавающей точкой *F*[*c*][*n*] согласно определению 32-битового формата с плавающей точкой *IEEE* с нормализацией с точки зрения максимального целочисленного значения (2^{23}).

F[c][n] = (float) (truncated_integer_value)$*2^{-23}$

Полем экспоненты *F*[*c*][*n*] является *shift value* [*c*], если *shift amp* [*c*] равно 1.

Если целое число является «0», кодированные данные с плавающей точкой используются как для окончательного выхода. Иначе восстановленное значение различия мантиссы *D*[*c*][*n*] добавляется к мантиссе данных с плавающей точкой *F*[*c*][*n*], которая преобразовывается из целочисленного значения и умножается на общий множитель.

6.9.3.3.3 Умножение на общий сомножитель

После преобразования общий множитель *A* восстанавливается из *acf mantissa* [*c*] и умножается на *F*[*c*][*n*], а результат устанавливается в *F*[*c*][*n*]. Вычислительная процедура умножения имеет следующий вид.

Шаг 1: Установка бита знака

Знак результата является таким же, как знак *F*[*c*][*n*].

Шаг 2: Умножение мантиссы

(*acf mantissa* [*c*] | 0×0800000) умножается на (биты мантиссы *F*[*c*][*n*] | 0×0800000) в 64-битовом целочисленном регистре.

Шаг 3: Нормализация

Результат 64-битового целочисленного умножения нормализуется до точности 23 бита.

Поскольку 1,0 ≤ (*acf_mantissa* [*c*] | 0×0800000) *2^{-23} (часть мантиссы *F*[*c*][*n*] | 0×0800000) *2^{-23} < 2,0, результат умножения находится в диапазоне [1, 4).

Следовательно, может понадобиться нормализация путем смещения на один бит вправо и приращения экспоненты.

Шаг 4: Округление

Режим округления «округление до ближайшего, до четного, когда привязка» применяется к округлению нормализованной мантиссы результата.

6.9.3.3.4 Добавление значения различия мантиссы

После умножения восстановленное значение различия мантиссы *D*[*c*][*n*] добавляется к данным с плавающей точкой *F*[*c*][*n*], и результат устанавливается в *F*[*c*][*n*]. Вычислительная процедура дополнения следующая.

Шаг 1: Добавление мантиссы

(*D*[*c*][*n*]) добавляется к (биты мантиссы *F*[*c*][*n*] | 0×0800000) в 32-разрядном целочисленном регистре.

Шаг 2: Нормализация

Результат 32-битового целочисленного дополнения нормализуется до 23-битовой точности.

Поскольку (*D*[*c*][*n*]) *2^{-23} < 1,0 и 1,0 ≤ (часть мантиссы *F*[*c*][*n*] | 0×0800000) *2^{-23} < 2,0, результат умножения находится в диапазоне [1, 3).

Следовательно, может понадобиться нормализация смещением на один бит вправо и наращиванием экспоненты.

Шаг 4: Усечение

Округление для этого дополнения не требуется, потому что всякий раз, когда происходит смещение, *LSB* получающейся мантиссы равняется 0.

Библиография

[1] ИСО/МЭК 14496-3:2009 Информационные технологии. Кодирование аудиовизуальных объектов. Часть 3. Аудио (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 переиздания *Е.В. Яковлева*
Технические редакторы *В.Н. Прусакова, И.Е. Черепкова*
Корректор *Е.М. Поляченко*
Компьютерная верстка *Д.В. Кардановской*

Сдано в набор 13.07.2020. Подписано в печать 11.11.2020. Формат 60 × 84 1/8. Гарнитура Ариал.
Усл. печ. л. 7,91. Уч.-изд. л. 7,40.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

ИД «Юриспруденция», 115419, Москва, ул. Орджоникидзе, 11.
gost.e-ecolog.ru gost.e-ecolog.ru

Создано в единичном исполнении во ФГУП «СТАНДАРТИНФОРМ»
для комплектования Федерального информационного фонда стандартов,
117418 Москва, Нахимовский пр-т, д. 31, к. 2.
gost.e-ecolog.ru stinfo.ru

gost.e-ecolog.ru