ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 53556.10— 2014

Звуковое вещание цифровое

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

ЧАСТЬ III
(MPEG–4 AUDIO)

Передискретизация аудио

ISO/IEC 14496-3:2009
(NEQ)

Издание официальное

Москва
Стандартинформ
2014

gost.e-ecolog.ru

Предисловие

1 РАЗРАБОТАН ТК 480 «Связь»

2 ВНЕСЕН Техническим комитетом по стандартизации № 480 «Связь»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 18 апреля 2014 г. № 386-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

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

© Стандартинформ, 2014

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

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)

Передискретизация аудио

Sound broadcasting digital.
Coding of signals of sound broadcasting with reduction of redundancy for transfer on digital communication channels.
Part III (MPEG–4 audio).
Oversampled audio

Дата введения — 2015—01—01

1 Область применения

Стандарт описывает алгоритм кодирования без потерь MPEG–4 для передискретизированных аудиосигналов.

2 Термины и определения

В этом стандарте используются следующие термины и определения:

Audio Channel *Audio Frame*	Поток битов *DSD*, предназначенный для одного громкоговорителя. Фрейм (кадр), содержащий аудиоданные.
Audio Channel Number	Порядковый номер, присвоенный звуковому каналу. Номера звуковых каналов присваиваются непрерывно, начиная с единицы.
Frame	Блок данных, принадлежащий определенному временному коду. Время воспроизведения фрейма составляет 1/75 с.
Reserved	Все поля, маркированные *Reserved* (Зарезервировано), резервируются для будущей стандартизации. Все поля *Reserved* должны быть обнулены.
Silence Pattern	Сгенерированная в цифровой форме кодограмма *DSD* со следующими свойствами: У всех аудиобайтов одно и то же значение; Каждый аудиобайт должен содержать 4 бита равные нулю и 4 бита равные единице.
Direct Stream Digital	Однобитовое передискретизированное представление аудиосигнала.
Direct Stream Transfer	Метод кодирования без потерь, используемый для сигналов *DSD* в компакт-диске аудио высшего качества.
Half Probability	Половинная вероятность определяет для каждого звукового канала в аудиофрейме, кодируются ли первые биты *DSD* арифметически, используя значения *Ptable*, или используя вероятность равную 1/2.
Mapping	Отображение определяет для каждого сегмента фильтр прогноза и таблицу вероятности.

Издание официальное

Prediction Filter	Фильтр прогноза является трансверсальным фильтром, используемым, чтобы предсказать значение следующего бита *DSD*. Фильтр прогноза характеризуется порядком прогноза и коэффициентами.
Probability Table	Таблица вероятности содержит вероятность того, что для данного вывода фильтра прогноза значение бита *DSD* предсказывается ошибочно.
Sampling Frequency	Частота дискретизации сигнала *DSD* должна быть 64 * 44,1 кГц, 128 * 44,1 кГц или 256 * 44,1 кГц.
Segmentation	Каждый звуковой канал в аудиофрейме может быть разделен на сегменты.

3 Условные обозначения

3.1 Арифметические и битовые операции

a >> *b*	Сдвиг *a* вправо на *b* битов. Новые биты *msb* устанавливаются в '0'.
a << *b*	Сдвиг *a* влево на *b* битов. Новые биты *lsb* устанавливаются в '0'.
a|*b*	Поразрядное ИЛИ для *a* и *b*.
a & *b*	Поразрядное И для *a* и *b*.
min(*a*, *b*)	Наименьшее значение из *a* и *b*.
max(*a*, *b*)	Наибольшее значение из *a* и *b*.
mod *b*	Значение *b* по модулю.
trunc(*a*)	Значение *a*, округленное в меньшую сторону.
|*a*|	Абсолютное значение *a*.
a = = *b*	Оценить равно ли *a* и *b*.
a! =*b*	Оценить не равны ли *a* и *b*.
a=*b*	Переменная *a* устанавливается в значение *b*.
a++	*a* = *a* + 1
a - = *b*	*a* = *a* - *b*
a + = *b*	*a* = *a* + *b*

3.2 Упорядочивание разрядов

Графическое изображение всех многоразрядных величин является таким, что старший значащий бит (*msb*) расположен слева, а младший значащий бит (*lsb*) - справа. Рисунок 1 определяет позицию двоичного разряда в байте.

msb *lsb*							
*b*7	*b*6	*b*5	*b*4	*b*3	b2	*b*1	*b*0

Рисунок 1 – Упорядочивание бита в байте

3.3 Последовательность разрядов

Во всех местах, где используется последовательность битов, применяется нотация со старшим значащим разрядом на первом месте.

3.4 Десятичная запись

Всем десятичным величинам предшествуют пробел или индикатор диапазона (..), когда включено в диапазон. Старшая значащая цифра находится слева, младшая значащая цифра - справа.

3.5 Порядок битов *DSD*

Первый выбранный бит *DSD* сохраняется в старшем значащем бите байта.

3.6 Полярность *DSD*

Бит *DSD*, равный единице, означает "плюс". Бит *DSD*, равный нулю, означает "минус".

3.7 Шестнадцатеричная нотация

Всем шестнадцатеричным значениям предшествует $. Старший значащий полубайт располагается слева, младший значащий полубайт - справа.

3.8 Диапазон

*Constant*_1.. *Constant*_2 обозначают диапазон от и включая *Constant*_1 до и включая *Constant*_2, с инкрементами 1.

3.9 *Until*

Until используется в рисунках, чтобы указать, что для позиции байта структуры используются до, но не включая, данное значение.

В позиции байта *B*1 выражение "*until B*2" определяет байты *B*2-*B*1. В позиции байта *B*1 выражение, "*until esc*" определяет число байтов от *B*1 до и включая последний байт текущего сектора. Позиция байта определяется относительно начала текущего или предыдущего сектора.

4 Основные типы

4.1 *BsMsbf*

Последовательность битов, старший значащий бит сначала, должна интерпретироваться как строка битов.

4.2 *Char*

Закодированный однобайтовый символ. *NUL* (нулевой) символ ($00) не разрешен для *Char*.

4.3 *SiMsbf*

Последовательность битов должна интерпретироваться как целое число со знаком.

4.4 *UiMsbf*

Последовательность битов должна интерпретироваться как целое число без знака.

4.5 *Uintn*

Закодированный двоичный файл *n* битов, численное значение без знака.

4.6 *Uint*8

Двоично закодированное 8-битовое численное значение без знака. Значение *Uint*8 должно записываться в однобайтовом поле.

4.7 *Uint*16

Двоично закодированное 16-битовое численное значение без знака. Значение *Uint*16, представленное шестнадцатеричным представлением $*wxyz*, должно записываться в двухбайтовом поле как $*wx* $*yz* (старший значащий байт сначала).

4.8 *Uint*32

Двоично закодированное 32-битовое численное значение без знака. Значение *Uint*32, представленное шестнадцатеричным представлением $*stuvwxyz*, должно записываться в четырехбайтовом поле как $*st* $*uv* $*wx* $*yz* (старший значащий байт сначала).

5 Полезные нагрузки для аудиообъекта

5.1 Конфигурация декодера (*DSTSpecificConfig*)

Таблица 1 – Синтаксис *Audio_Frame ()*

Синтаксис	Количество битов	Мнемоника
DSTSpecificConfig(channelConfiguration) { *DSDDST_Coded* *N_Channels* *reserved* *}*	 1 14 1	 *UiMsbf UiMsbf* *UiMsbf*

5.2 Полезная нагрузка потока битов

Таблица 2 – Синтаксис *Audio_Frame* ()

Синтаксис	Количество битов	Мнемоника
Audio Frame() { *if (DSDDST Coded == 0)* *{* *DSD()* *}* *else* *{* *DST()* *}* *}*		 *DSD* *DST*

Таблица 3 – Синтаксис *DSD*

Синтаксис	Количество битов	Мнемоника
DSD() { *For (Byte Nr=0; Byte Nr<Frame Length; Byte Nr++)* *{* *For (Channel_Nr=1; Channel_Nr<=N_Channels;* *Channel Nr++)* *{* *DSD Byte[Channel Nr][Byte Nr]* *}* *}* *}*	 1	 *Audio_Byte*

Таблица 4 – Синтаксис *DST*

Синтаксис	Количество битов	Мнемоника
DST() { *Processi ng_Mode* *if (Processing Mode == 0)* *{* *DST_X_Bit* *Reserved* *DSD()* *}* *else* *{* *Segmentation()* *Mapping()* *Half_Probability()* *Filter_Coef_Sets()* *Probability_Tables()* *Arithmetic Coded Data()* *}* *}*	 1 1 6 	 *BsMsbf* *BsMsbf* *BsMsbf* *DSD* *Segmentation* *Mapping* *Half_Probability* *Filter_Coef_Sets* *Probability_Tables* *Arithmetic Coded Data*

Таблица 5 – Синтаксис сегментации

Синтаксис	Количество битов	Мнемоника
Segmentation() { *Same_Segmentation* *if(Same Segmentation == 0)* *{* *Filter_Segmentation()* *Ptable Segmentation()* *}* *else* *{* *Filter And Ptable Segmentation()* *}* *}*	 1 	 *BsMsbf* *Segment_Alloc* *Segment_Alloc* *Segment_Alloc*

Таблица 6 – Синтаксис сегментов

Синтаксис	Количество битов	Мнемоника
{ *Channel Segmentation()[Channel Nr]* *}* *}* *else* *{* *Channel Segmentation()[1]* *}* *}*	 	 *Channel_Segmentation* *Channel_Segmentation*

Таблица 7 – Синтаксис *Channel_Segmentation*

Синтаксис	Количество битов	Мнемоника
Scaled_Length[Nr_Of_Segments]	1..13	*UiMsbf*
*Segment_Length[Nr_Of_Segments] = Resolution * Scaled_Length[Nr_Of_Segments]*		
Start[Nr_Of_Segments+1] = Start[Nr_Of_Segments] + Segment_Length[Nr_Of_Segments]		
Nr_Of_Segments++		
End Of Channel Segm	1	*UiMsbf*
}		
Segment_Length[Nr_Of_Segments] = Frame Length - Start[Nr Of Segments]		
}		

Таблица 8 – Синтаксис отображения

Синтаксис	Количество битов	Мнемоника
Mapping() {		
Same_Mapping	1	*UiMsbf*
if(Same Mapping == 0)		
{		
Filter_Mapping()		*Maps*
Ptable Mapping()		*Maps*
}		
else		
{		
Filter And Ptable Mapping()		*Maps*
}		
}		

Таблица 9 – Синтаксис отображений

Синтаксис	Количество битов	Мнемоника
Maps() {		
Nr_Of_Elements = 0		
Same_Maps_For_All_Channels	1	*UiMsbf*
if(Same Maps For All Channels == 0)		
{		
for (Channel_Nr=1; Channel_Nr<=N_Channels; Channel_Nr++)		
{		
Channel Mapping()[Channel Nr]		
}		
}		
else		*Channel_Mapping*
{		
Channel Mapping()[1]		
}		
		Channel_Mapping

Таблица 10 – Синтаксис *Channel_Mapping*

Синтаксис	Количество битов	Мнемоника
Channel_Mapping() {		
for(Seg Nr=1; Seg Nr<=Nr Of Segments[Channel Nr]; Seg Nr++)		
{		
Element[Channel_Nr][Seg_Nr]	0..15	*UiMsbf*
if (Element[Channel Nr][Seg Nr] == Nr Of Elements)		
{		
Nr Of Elements++		
}		
}		
}		

Таблица 11 – Синтаксис *Half_Probability*

Синтаксис	Количество битов	Мнемоника
Half_Probability() { *for (Channel Nr=1; Channel Nr<=N Channels; Channel Nr++)* *{* *Half Prob[Channel Nr]* *}* *}*	1	*BsMsbf*

Таблица 12 – Синтаксис *Arithmetic_Coded_Data*

Синтаксис	Количество битов	Мнемоника
Arithmetic Coded Data() { *j=0* *do* *{* *A_Data[j]* *j++* *} until end of Audio Frame* *}*	1	*BsMsbf*

Таблица 13 – Синтаксис *Filter_Coef_Sets*

Синтаксис	Количество битов	Мнемоника
Filter_Coef_Sets() { *for (Filter Nr=0; Filter Nr<Nr Of Filters; Filter Nr++)* *{* *Coded_Pred_Order* *Pred_Order[Filter_Nr]=Coded_Pred_Order+1* *Coded_Filter_Coef_Set* *if (Coded Filter Coef Set==0)* *{* *for (Coef Nr=0; Coef Nr<Pred Order[Filter Nr];* *Coef Nr++)* *{* *Coef[Filter Nr][Coef Nr]* *}* *}* *else* *{* *CC_Method* *for (Coef_Nr=0; Coef_Nr<CCPO; Coef_Nr++)* *{* *Coef[Filter_Nr][Coef_Nr]* *}* *CCM* *for (Coef_Nr=CCPO;* *Coef_Nr<Pred_Order[Filter_Nr];* *Coef_Nr++)* *{* *Run_Length=0* *do* *{* *RL_Bit* *if (RL_Bit==0)* *{* *Run_Length++* *}* *} while (RL_Bit==0)* *LSBs* *Delta=(Run_Length<<CCM)+LSBs* *if (Delta!=0)*	7 1 9 2 9 3 1 0…6	*UiMsbf* *BsMsbf* *SiMsbf* *BsMsbf* *SiMsbf* *UiMsbf* *BsMsbf* *UiMsbf*

Окончание таблицы 13

Синтаксис	Количество битов	Мнемоника
{ *Sign* *if (Sign==1)* *{* *Delta = -Delta* *}* *}* *Coef[Filter_Nr][Coef_Nr] = Delta* *Delta8 = 0* *for (Tap_Nr=0; Tap_Nr<CCPO; Tap_Nr++)* *{* *Delta8 += 8*CCPC[Tap_Nr]*Coef[Filter_Nr][Coef_Nr-Tap_Nr-1]* *}* *if (Delta8>=0)* *{* *Coef[Filter_Nr][Coef_Nr] -= trunc((Delta8+4)/8)* *}* *else* *{* *Coef[Filter_Nr][Coef_Nr] += trunc((-Delta8+3)/8)* *}* *}* *}* *}* *}*	1	*BsMsbf*

Таблица 14 – Синтаксис *Probability_Tables*

Синтаксис	Количество битов	Мнемоника
Probability_Tables() {		
for (Ptable Nr=0; Ptable Nr<Nr Of Ptables; Ptable Nr++)		
{		
Coded_Ptable_Len	6	*UiMsbf*
Ptable_Len[Ptable_Nr] = Coded_Ptable_Len+1		
if (Ptable_Len[Ptable_Nr] == 1)		
{		
P_one[Ptable_Nr][0] = 128		
}		
else		
{		
Coded_Ptable	1	*BsMsbf*
If (Coded_Ptable==0)		
{		
for (Entry_Nr=0; Entry_Nr<Ptable_Len[Ptable_Nr];		
Entry_Nr++)		
{		
Coded_P_one	7	*UiMsbf*
P_one[Ptable_Nr][Entry_Nr] = Coded_P_one+1		
}		
}		
else		
{		
PC_Method	2	*BsMsbf*
for (Entry_Nr=0; Entry_Nr<PCPO; Entry_Nr++)		
{		
Coded_P_one	7	*UiMsbf*
P_one[Ptable_Nr][Entry_Nr] = Coded_P_one+1		
}		
PCM	3	*UiMsbf*
for (Entry_Nr=PCPO;Entry_Nr<Ptable_Len[Ptable_Nr];Entry_Nr++)		
{		
Run Length=0		
do		
{		
RL Bit	1	*BsMsbf*
if (RL_Bit==0)		
{		
Run_Length++		
}		
} while (RL_Bit==0)		
LSBs	0…4	*UiMsbf*
Delta = (Run_Length<<PCM)+LSBs		
if (Delta != 0)		
{		
Sign	1	*BsMsbf*
if (Sign==1)		
{		
Delta = - Delta		
}		
}		
P_one[Ptable_Nr][Entry_Nr] = Delta		
for (Tap Nr=0; Tap Nr<PCPO; Tap Nr++)		
{		
P_one[Ptable_Nr][Entry_Nr] -=		
*PCPC[Tap Nr]*P one[Ptable Nr][Entry Nr-Tap Nr-1]*		
}		
}		
}		
}		
}		

6 Семантика

6.1 Аудиопотоки

Аудиопоток содержит аудиосигнал *DSD* (простой *DSD* или *DST*–кодированный *DSD)*. Аудиопоток является конкатенацией всех аудиофреймов в потоке байтов.

6.1.1 Данные потока битов дискретизированного *DSD*

Для сигнала 2-канального аудио разрядная последовательность дискретизированного *DSD* определяется следующим образом: L_0, L_1, L_2, L_3, L_4, L_5, L_6, L_7, R_0, R_1, R_2, R_3, R_4, R_5, R_6, R_7, L_8, L_9, L_{10}, …, где L_0 является первым битом выборки левого канала аудиофрейма.

Для каждого звукового канала восемь последовательных битов выборки группируются в один аудиобайт. Старший значащий бит аудиобайта является первым битом выборки этого байта.

6.1.2 Структура аудиопотока

У кодированных аудиофреймов *DST* существует переменная длина. Эталонная модель декодера *DST* определяется в разделе 7.

6.1.3 *Audio_Frame*

Audio_Frame (аудиофрейм) содержит кодированную *DST* или просто *DSD* аудиоинформацию для одного фрейма. Максимальный размер *Audio_Frame* равен размеру кодированного просто *DSD Audio_Frame* плюс один байт. Синтаксис *Audio_Frame* определяется в таблице 1.

N_Channels является числом используемых звуковых каналов.

6.2 *DSD* содержит аудиоданные для одной *Audio_Frame* простого *DSD*. Синтаксис *DSD* определяется в полезной нагрузке потока битов

Channel_Nr является номером звукового канала.

Frame_Length является длиной аудиофрейма в байтах на звуковой канал. *Frame_Length* может быть вычислено, исходя из частоты дискретизации по формуле:

$$Frame_Length = \frac{Sampling\ Frequence}{75^{*}8}\ \ bytes\ per\ AudioCannel$$

Частота дискретизации может быть: 64*44100 Гц, 128*44100 Гц или 256*44100 Гц. Соотношение между *Frame_Length* и частотой дискретизации дается в таблице 15.

Таблица 15 – Соотношение *Frame_Length* и частоты дискретизации

Частота дискретизации, Гц	*Frame_Length,* байт
64*44100	4704
128*44100	9408
256*44100	18816

DSDDST_Coded сигнализирует, кодирован ли поток битов *DSD* или *DST*. Если *DSDDST_Coded* = % 0 - это *DSD*-кодированный поток, а если *DSDDST_Coded* = % 1 – *DST*-кодированный.

6.2.1. *DSD_Byte*

DSD_Byte [*Channel_Nr*] [*Byte_Nr*] содержит сигнал *DSD* как определено в 6.1.1.

6.2.1.1 *DST*

DST содержит аудиоданные для кодированного *Audio_Frame* одного *DST*. Синтаксис *DST* определяется в таблице 4.

6.2.1.1.1 *Processing_Mode*

Если бит *Processing_Mode* устанавливается в единицу, *Audio_Frame* содержит *DST_X_Bit* и сигнал *DSD* в форме кодирования без потерь. Если бит *Processing_Mode* обнуляется, *Audio_Frame* содержит *DST_X_Bit* и сигнал *DSD* без кодирования без потерь.

6.2.1.1.2 *DST_X_Bit*

Если *Frame_Format DST_Coded*, то каждый *Audio_Frame* содержит один бит *DST_X_Bit*. При кодировании *DST_X_Bit* должен быть обнулен. Декодер должен проигнорировать контент *DST_X_Bit*.

6.2.1.1.3 Зарезервировано

Это значение должно быть установлено в ноль.

6.2.1.1.4 *DSD*

См. 6.1.1.

6.2.1.1.5 Сегментация

Для каждого звукового канала аудиофрейм делится на один или более сегментов для фильтров и один или более сегментов для *Ptables*. Каждый сегмент может использовать различные фильтр прогноза/*Ptable*. Синтаксис сегментации определяется в таблице 5.

Filter_Segmentation

Для каждого звукового канала аудиофрейм делится на один или более сегментов для фильтров прогноза. Каждый сегмент может использовать различные фильтры прогноза. Переменные *Nr_Of_Segments[ ]* и *Segment_Length[ ][ ]* из *Segment_Alloc*, используемые для *Filter_Segmentation,* упоминаются как *Filters.Nr_Of_Segments*[*Channel_Nr*] и *Filters.Segment_Length*[*Channel_Nr*][1..*Filters.Nr_Of_Segments*[*Channel_Nr*]], где *Channel_Nr* = 1.. *N_Channels*.

Ptable_Segmentation

Для каждого звукового канала аудиофрейм делится на один или более сегментов для *Ptables*. Каждый сегмент может использовать различные *Ptable*. Переменные *Nr_Of_Segments[]* и *Segment_Length[][]* из *Segment_Alloc*, используемые для *Ptable_Segmentation*, упоминаются как *Ptables.Nr_Of_Segments*[*Channel_Nr*] и *Ptables.Segment_Length*[*Channel_Nr*][1..*Ptables.Nr_Of_Segments*[*Channel_Nr*]], где *Channel_Nr* = 1.. *N_Channels*.

Filter_And_Ptable_Segmentation

Для каждого звукового канала аудиофрейм делится на один или более сегментов. Каждый сегмент может использовать различные комбинации *Prediction Filter* и *Ptable*. Для каждого звукового канала должны быть истиной следующие уравнения:

Filters.Nr_Of_Segments[Channel_Nr] = Ptables. Nr_Of_Segments[Channel_Nr] = Nr_Of_Segments[Channel_Nr]

Filters.Segment_Length [Channel_Nr] [ ] = Ptables. Segment_Length [Channel_Nr] [ ] = Segment_Length [Channel_Nr] [ ] ,

где *Channel_Nr* = 1.. *N_Channels.*

6.2.1.1.5.1 *Same_Segmentation*

Если *Same_Segmentation* равно единице, *Ptables* и фильтры прогноза используют одну и ту же сегментацию. Если *Same_Segmentation* является нулем, разделение для аудиофрейма для фильтров прогноза независимо от разделения для *Ptables*.

6.2.1.1.5.2 *Segment_Alloc*

Segment_Alloc определяет сегментацию для фильтров прогноза и/или *Ptables*. Синтаксис *Segment_Alloc* определяется в таблице 6.

Для каждого звукового канала переменные *Nr_Of_Segments* и *Segment_Length* [1.. *Nr_Of_Segments*] из *Channel_Segmentation* упоминаются как *Nr_Of_Segments* [*Channel_Nr*] или *Segment_Length* [*Channel_Nr*] [1.. *Nr_Of_Segments* [*Channel_Nr*]].

Resolution_Read указывает, было ли считано разрешение из *Channel_Segmentation*. *Resolution_Read* устанавливается в истину в *Channel_Segmentation* первого звукового канала больше чем с одним сегментом. Если фильтры прогноза и *Ptables* используют независимую сегментацию, они также используют независимое *Resolution_Read*. *Channel_Nr* является локальной индексной переменной. *N_Channels* является числом используемых звуковых каналов.

6.2.1.1.5.2.1 *Same_Segm_For_All_Channels*

Если *Same_Segm_For_All_Channels* равно единице, сохраняется только сегментация для первого звукового канала и *Channel_Segmentation* ()[*Channel_Nr*] = *Channel_Segmentation* () для всех звуковых каналов. Если *Same_Segm_For_All_Channels* является нулем, аудиофрейм делится на сегменты, независимые для каждого звукового канала.

6.2.1.1.5.2.2 *Channel_Segmentation*

Channel_Segmentation определяет сегментацию фильтров прогноза и/или *Ptables*. Синтаксис *Channel_Segmentation* определяется в таблице 7.

В синтаксисе *Channel_Segmentation* используются переменные *Nr_Of_Segments, Start[*1.. *Nr_Of_Segments] , Segment_Length [*1.. *Nr_Of_Segments].*

Nr_Of_Segments является числом сегментов для текущего звукового канала. Максимальное количество сегментов является *MAXNRSEGS. MAXNRSEGS* должно быть 4 для *Filter_Segmentation*, 8 для *Ptable_Segmentation* и 4 для *Filter_And_Ptable_Segmentation*.

Resolution_Read указывает, была ли считана переменная *Resolution* в этом или предыдущем *Channel_Segmentation*. *Resolution_Read* устанавливается в истину в *Channel_Segmentation* первого звукового канала с больше чем одним сегментом. Если *Prediction Filters* и *Ptables* используют независимую сегментацию, они также используют и независимое *Resolution_Read*.

Segment_Length [Seg_Nr] содержит длину сегмента в байтах, где:

1 <= *Seg_Nr* <= *Nr_Of_Segments*.

Start[Seg_Nr] является стартовой позицией в байтах *Segment*[*Seg_Nr*].

6.2.1.1.5.2.2.1 *End_Of_Channel_Segm*

Если *End_Of_Channel_Segm* является нулем, будет следовать одно или более значений для *Scaled_Length*. Если *End_Of_Channel_Segm* равно единице, структура *Channel_Segmentation* заканчивается.

6.2.1.1.5.2.2.2 *Resolution*

Каждое значение *Scaled_Length* умножается на *Resolution* (разрешение), чтобы получить длину сегмента в байтах. Разрешение сохраняется только однажды, в начале первого звукового канала с больше чем одним сегментом. Если у всех звуковых каналов имеется только один сегмент, *Resolution* не кодируется.

Разрешение должно быть в диапазоне от 1 до *Frame_Length* - *MINSEGLEN*. *MINSEGLEN* должно составлять 128 байтов для *Filter_Segmentation*, 4 байта для *Ptable_Segmentation* и 128 байтов для *Filter_And_Ptable_Segmentation*.

6.2.1.1.5.2.2.3 *Scaled_Length*

Для каждого сегмента, кроме последнего, значение *Scaled_Length* кодируется. Длина сегмента в байтах вычисляется по следующей формуле:

$$Segment_Length\ [Seg_Nr] = Resolution * Scaled_Length\ [Seg_Nr],$$

где 1 <= *S_Nr* <*Nr_Of_Segments*.

Минимальная длина сегмента каждого *Segment* является *MINSEGLEN*.

Для *Ptable_Segmentation* длина первого сегмента каждого звукового канала должна быть (*Pred_Order* [*Filter*[*Channel_Nr*] [1]] +7)/8 байтов.

Число битов, необходимых для кодирования *Scaled_Length*[*Seg_Nr*], зависит от значения диапазона. Диапазон (*Range*) должен быть вычислен по формуле:

$$Range = Trunc\left(\frac{Frame_Length\text{-}Start\ [Seg_Nr\ ] - MINSEGLEN}{Resolution}\right).$$

Если $2^{n-1} \le Range < 2^{n}$, чтобы закодировать *Scaled_Length*[*Seg_Nr*] должны использоваться *n* битов, см. таблицу 16. Минимальное значение *Range* равно 1. Длина последнего сегмента не кодируется. Длина последнего сегмента может быть вычислена, исходя из длины фрейма и стартовой позиции последнего сегмента по формуле:

$$Segment_Length\ [Nr_Of_Segments] = Frame_Length - Start[Nr_Of_Segments].$$

Таблица 16 – Диапазон и биты для кодирования *Scaled_Length*

Диапазон	Используемые биты
1	1
2..3	2
4..7	3
8..15	4
16..31	5
32..63	6
64..127	7
128..255	8
256..511	9
512..1023	10
1024..2047	11
2048..4095	12
4096..8191	13

6.2.1.1.6 *Mapping*

Mapping (отображение) определяет фильтры прогноза и *Ptables*, используемого с сегментами, определенными в 6.2.1.1.5. Синтаксис отображения определяется в таблице 8.

Filter_Mapping

Для каждого звукового канала и каждого сегмента кодируется номер фильтра прогноза. Для *Filter_Mapping* переменная *Element [ ][ ]* из *Channel_Mapping* содержит номера фильтра прогноза. Для *Filter_Mapping* переменная *Element[ ][ ]* упоминается как *Filter[Channel_Nr][1.. Filters. Nr_Of_Segments [Channel_Nr]].* Для *Filter_Mapping* переменная *Nr_Of_Elements* упоминается как *Nr_Of_Filters*.

Ptable_Mapping

Для каждого звукового канала и каждого сегмента кодируется номер *Ptable*. Для *Ptable_Mapping* переменная *Element[ ][ ]* из *Channel_Mapping* содержит номера *Ptable*. Для *Ptable_Mapping* переменная *Element[][]* упоминается как *Ptable [Channel_Nr] [1.. Ptables. Nr_Of_Segments [Channel_Nr]].* Для *Ptable_Mapping* переменная *Nr_Of_Elements* упоминается как *Nr_Of_Ptables*.

Filter_And_Ptable_Mapping

Для каждого звукового канала и каждого сегмента кодируется общий номер *Prediction Filter* и *Ptable*. Для *Filter_*and*_Ptable_Mapping* переменная *Element[][]* из *Channel_Mapping* содержит общие номера *Prediction Filter* и *Ptable*. Для *Filter_*and*_Ptable_Mapping* переменная *Element[][]* упоминается как *Filter[Channel_Nr] [1.. Filters.Nr_Of_Segments[Channel_Nr]]* или

Ptable[Channel_Nr][1.. Ptables.Nr_Of_Segments[Channel_Nr]]. Для *Filter_and_Ptable_Mapping* переменная *Nr_Of_Elements* упоминается как *Nr_Of_Filters*, а также как *Nr_Of_Ptables*.

6.2.1.1.6.1 *Same_Mapping*

Если *Same_Mapping* равно единице, *Ptables* и фильтры пргноза используют одно и то же отображение. Если *Same_Mapping* является нулем, для аудиофрейма отображение для фильтров прогноза независимо от отображения для *Ptables*.

6.2.1.1.6.2 *Maps*

Maps (отображения) определяют отображение в фильтры прогноза и в сегменты, определенные в 6.2.1.1.5. Синтаксис отображений определяется в таблице 9.

В синтаксисе отображений используются переменные *Element[Channel_Nr][1.. Nr_Of_Segments[Channel_Nr]]* для каждого звукового канала и *Nr_Of_Elements*

Nr_Of_Elements является общим номером фильтра прогноза и/или *Ptables*, используемого в отображениях. *Nr_Of_Elements* должен быть в диапазоне 1.. 2 * *N_Channels*.

Channel_Nr является локальной индексной переменной, используемой в таблице 9 и таблице 10.

N_Channels является числом используемых звуковых каналов.

6.2.1.1.6.2.1 *Same_Maps_For_All_Channels*

Если *Same_Maps_For_All_Channels* равно единице, сохраняется только *Element*[1][ ], и каждый звуковой канал использует тот же самый *Element[Channel_Nr][ ] = Element[1][ ]* массива. Если *Same_Maps_For_All_Channels* равно нулю, *Element[Channel_Nr][ ]* сохраняется независимо для каждого звукового канала. Если *Nr_Of_Segments* [*Channel_Nr*] не имеет для всех звуковых каналов одинакового значения, то *Same_Maps_For_All_Channels* должно быть нулем.

6.2.1.1.6.2.2 *Channel_Mapping*

Channel_Mapping содержит номера фильтра прогноза и/или *Ptable* на звуковой канал, используемые для каждого сегмента. Синтаксис *Channel_Mapping* определяется в таблице 10.

Nr_Of_Elements является общей численностью фильтров прогноза и/или *Ptables* для всех каналов. *Nr_Of_Elements* инициализируется в отображениях, таблица 9.

Channel_Nr является индексной переменной из отображений, таблица 9.

Seg_Nr является локальной индексной переменной.

Nr_Of_Segments [Channel_Nr] является общим количеством сегментов, используемых в текущем аудиофрейме для звукового канала *Channel_Nr*.

6.2.1.1.6.2.2.1 *Element*

Element является номером фильтра прогноза и/или *Ptable*, используемым в сегменте. Количество используемых для кодирования *Element* битов зависит от значения *Nr_Of_Elements*. В каждой итерации *Element* должен быть <= *Nr_Of_Elements*. Поэтому, *Element[1][1]* всегда является нулем и не сохраняется (*#bits* = 0). Для всех других звуковых каналов и сегментов *Nr_Of_Elements* > 0 и число битов, необходимых для сохранения *Element*, равно *n* с: $2^{n-1} \le Nr_Of_Elements < 2^{n}$, таблица 17.

Таблица 17 – Биты, используемые для кодирования *Element*

Nr_Of_Elements	Номер используемого бита
0	0
1	1
2..3	2
4..7	3
8..15	4
16..31	5
32..63	6
64..127	7
128..255	8
256..511	9
512..1023	10
1024..2047	11
2048..4095	12
4096..8191	13
8192..16383	14
16384..32766	15

6.2.1.1.7 *Half_Probability*

Синтаксис *Half_Probability* определяется в таблице 11.

Channel_Nr является локальной индексной переменной.

N_Channels является числом используемых звуковых каналов.

6.2.1.1.7.1 *Half_Prob*

Half_Prob используется, при кодировании, для каждого звукового канала. Метод обычно используется для того, чтобы применить значение вероятности к арифметическому декодеру. Определение *Half_Prob* дается в таблице 18.

Таблица 18 – Определение *Half_Prob*

Half_Prob[ ]	Вероятность для использования во время первых битов *Pred_Order[ ]* аудиоканала
0	Использовать записи из *Ptable*.
1	Использовать $p=\frac{1}{2}$ (соответствует *P_one* = 128)

Для оптимальной эффективности кодирования требуется, чтобы у следующего остаточного бита в *E* было значение, которое имеет наибольшую вероятность. Если применяется вероятность, которая отражает высокий шанс следующего бита *E* быть 1, в то время как следующий бит *E* является 0, то требуется более 1 бита в арифметическом коде, чтобы отправить этот бит.

Фильтр прогноза первоначально заполнен образцом инициализации. Во время первых выборок *Pred_Order* в звуковом канале *Channel_Nr* фильтр прогноза постепенно заполняется реальными данными *DSD*. Как следствие распределение вероятности может отличаться от остальной части фрейма, и комбинация примененных *E* и *P* для этих битов приведет к большему количеству битов, чем требуется. Применяя вероятность 1/2 во время кодирования, каждый бит будет также стоить только одного бита в арифметическом коде.

Чтобы быть в состоянии отвергнуть плохую комбинацию *E* и *P* в начале фрейма *Half_Prob* доступен для каждого канала отдельно.

6.2.1.1.8 *Filter_Coef_Sets*

Для каждого сегмента в каждом звуковом канале декодер *DST* использует фильтр прогноза. В случае если два или более фильтра прогноза одинаковы, соответствующие коэффициенты фильтра могут кодироваться только однажды. В синтаксисе *Filter_Coef_Sets* используются переменные *Pred_Order[Filter_Nr]* и *Coef[Filter_Nr]*[0..*Pred_Order[Filter_Nr]*-1], где *Filter_Nr* = 0.. *Nr_Of_Filters*-1.

Все коэффициенты фильтра прогноза кодируются. Коэффициенты фильтра прогноза могут быть закодированы, используя простое линейное предсказание и кодирование Райса. Кодирование Райса является методом кодирования переменной длины (особый случай кодирования методом Хаффмана), который используется, чтобы сократить число битов, необходимых для определенного

"сообщения", без потери информации. Синтаксис *Filter_Coef_Sets* определяется в таблице 13.

Младший значащий бит *Coef[0] [0]* называют *DST_Y_Bit*.

Nr_Of_Filters является значением, вычисленным в отображении.

Pred_Order [ ] является массивом, который содержит порядок прогноза для каждого фильтра прогноза, где *Pred_Order* [*Filter_Nr*] = *Coded_Pred_Order* + 1, для *Filter_Nr*∈I }(0.. *Nr_Of_Filters*-1). Допустимый диапазон порядка прогноза равен: 1 ≤ *Pred_Order* [*Filter_Nr*] ≤ 128.

Coef[ ][ ] является двухмерным массивом, который содержит все коэффициенты всех фильтров прогноза. Каждая запись *Coef[ ][ ]* должна быть в диапазоне от -256 до +255. Первый (левый) индекс является *Filter_Nr* и простирается от 0 до *Nr_Of_Filters*-1. Второй (правый) индекс является номером коэффициента и простирается от 0 до *Pred_Order*[*Filter_Nr*]-1.

CCPO является порядком прогноза кодирования коэффициента (*CCPO*). Отношение между *CC_Method* и *CCPO* определяется в таблице 19.

Таблица 19 – Отношение между *CC_Method* и *CCPO*

CC_Method	*CCPO*
'00'	1
'01'	2
'10'	3
'11'	Не используется

Применяется ограничение *CCPO* <*Pred_Order* [*Filter_Nr*].

Run_Length является вспомогательной переменной, предназначенной для подсчета числа нулей в коде длины серии, который является частью кода Райса.

Delta является вспомогательной переменной, чтобы вычислять кодированный по Райсу номер.

*Delta*8 является вспомогательной переменной, чтобы вычислять кодированный по Райсу номер.

CCPC [ ] является массивом, который содержит коэффициенты прогноза кодирования коэффициентов (*CCPC*), которые используются для линейного прогноза коэффициентов фильтра. Отношение между *CC_Method* и *CCPC [ ]* определяется в таблице 20.

Таблица 20 – Отношение между *CC_Method* и *CCPC* [ ]

CC_Method	*CCPC*[0]	*CCPC*[1]	*CCPC*[2]
'00'	-1	-	-
'01'	-2	1	-
'10'	-9/8	-5/8	6/8
'11'	Не используется	Не используется	Не используется

Линейное предсказание требует округления, как определено в синтаксисе таблицы 13.

6.2.1.1.8.1 *DST_Y_Bit*

DST_Y_Bit является младшим значащим битом *Coef*[0][0]. При кодировании *DST_Y_Bit* должен быть установлен в единицу. Декодер должен игнорировать контент *DST_Y_Bit*.

6.2.1.1.8.2 *Coded_Pred_Order*

Coded_Pred_Order является 7-битовым целым числом без знака, которое содержит кодированный порядок прогноза текущего фильтра прогноза.

6.2.1.1.8.3 *Coded_Filter_Coef_Set*

Coded_Filter_Coef_Set указывает, предсказываются ли коэффициенты фильтра прогноза, и кодированы ли они по Райсу. *Coded_Filter_Coef_Set* обнуляется, если коэффициенты фильтра прогноза сохраняются.

Coded_Filter_Coef_Set устанавливается в единицу, если коэффициенты фильтра прогноза предсказываются и кодированы по Райсу.

Максимальное количество битов, разрешенных для единственного фильтра прогноза внутри *Filter_Coef_Sets,* равно 7+1+*Pred_Order* [ ] *9, где 7 - количество битов для *Coded_Pred_Order*, 1 - количество битов *Coded_Filter_Coef_Set* и остальные коэффициенты *Pred_Order [ ]* имеют по 9 битов каждый.

6.2.1.1.8.4 *CC_Method*

CC_Method является 2-битовым кодом, который идентифицирует метод кодирования коэффициентов текущего фильтра прогноза.

6.2.1.1.8.5 *CCM*

CCM является 3-битовым целым числом без знака, которое содержит параметр *M* кодирования коэффициентов, который используется для декодирования по Райсу коэффициентов текущего фильтра прогноза. Минимальное разрешенное значение для *CCM* равно нулю. Максимальное разрешенное значение для *CCM* равно 6.

6.2.1.1.8.6 *RL_Bit*

RL_Bit используется, чтобы получить единственные биты кода длины серии, который состоит из нулей с завершающей единицей. Самым коротким кодом длины серии является '1'.

6.2.1.1.8.7 *LSBs*

Младшие значащие биты *CCM* абсолютного значения предсказанного коэффициента считываются непосредственно из потока и сохраняются в *LSBs*.

6.2.1.1.8.8 *Sign*

Sign (знак) является битом, который указывает, положителен ли предсказанный коэффициент (*Sign* = '0') или отрицателен (*Sign* = '1').

6.2.1.1.9 *Probability_Tables*

Для каждого сегмента в каждом звуковом канале декодер использует таблицу вероятности (*Ptable*). В случае если две или более таблицы вероятности равны, соответствующие записи таблицы вероятности могут быть доступными из потока только однажды. В синтаксисе *Probability_Tables* используются переменные *Ptable_Len [Ptable_Nr]* и *P_One [Ptable_Nr] [*0*...Ptable_Len [Ptable_Nr]-1].* Где *Ptable_Nr* = 0*.. Nr_Of_Ptables-1.*

В *Probability_Tables* кодируются все записи таблицы вероятности. На таблицу вероятности кодируются длина таблицы (= число записей) и записи. Записи *Ptable* могут быть кодированы с использованием простого линейного прогноза и кодирования Райса. Синтаксис *Probability_Tables* определяется в таблице 14.

Nr_Of_Ptables является значением, вычисленным в отображении. *Ptable_Len []* является массивом, который содержит длину таблицы вероятности для каждой *Ptable*, где *Ptable_Len* [*Ptable_Nr*] = *Coded_Ptable_Len* + 1, для *Ptable_Nr*∈{0.. *Nr_Of_Ptables*-1}.

Допустимый диапазон длины *Ptable*: 1 □ *Ptable_Len* [*Ptable_Nr*] □ 64.

P_one[ ][ ] является двухмерным массивом, который содержит все записи всех таблиц вероятности. Первый (левый) индекс является *Ptable_Nr* и находится в диапазоне от 0 до *Nr_Of_Ptables*-1. Второй (правый) индекс является числом записей и лежит в диапазоне от 0 до *Ptable_Len* [*Ptable_Nr*]-1. Каждая запись *P_one[ ][ ]* находится в диапазоне 1 - 128, соответствуя вероятности от 1/256 до 128/256 следующего ошибочного бита.

PCPO является порядком прогноза кодирования *Ptable* (*PCPO*). Отношение между *PC_Method* и *PCPO* определяется в таблице 21.

Таблица 21 – Отношение между *PC_Method* и *PCPO*

PC_Method	*PCPO*
'00'	1
'01'	2
'10'	3
'11'	Не используется

Применяется ограничение *PCPO <Ptable_Len [Ptable_Nr].*

Run_Length является переменной справки, чтобы считать число нулей в коде длины серии, который является частью кода Райса.

Delta является переменной справки, чтобы вычислять декодированное число Райса.

PCPC [ ] является массивом, который содержит коэффициенты прогноза кодирования *Ptable (PCPC),* которые используются для линейного предсказания записей *Ptable*. Отношение между *PC_Method* и *PCPC [ ]* определяется в таблице 22.

Таблица 22 – Отношение между *PC_Method* и *PCPC* [ ]

PC_Method	*PCPC*[0]	*PCPC*[1]	*PCPC*[2
'00'	-1	-	-
'01'	-2	1	-
'10'	-3	3	-1
'11'	Не используется	Не используется	Не используется

6.2.1.1.9.1 *Coded_Ptable_Len*

Coded_Ptable_Len является 6-битовым целым числом без знака, которое содержит кодированную длину таблицы вероятности.

6.2.1.1.9.2 *Coded_Ptable*

Coded_Ptable указывает, прогнозируются ли записи *Ptable* и кодируются ли по Райсу. *Coded_Ptable* обнуляется, если записи *Ptable* сохраняются. *Coded_Ptable* устанавливается в единицу, если записи *Ptable* прогнозируются и кодируются по Райсу.

Максимальное количество битов, разрешенных для единственной *Ptable* внутри *Probability_Tables* равно 6+1+*Ptable_Len[ ]*7, где 6 - количество битов для *Coded_Ptable_Len*, 1 - количество битов *Coded_Ptable* и остальные *Ptable_Len* [ ] кодированные записи *Ptable* по 7 битов каждая.

6.2.1.1.9.3 *Coded_P_one*

Coded_P_one является 7-битовым целым числом без знака, которое содержит кодированное значение следующей записи текущей *Ptable*.

6.2.1.1.9.4 *PC_Method*

PC_Method является 2 битовым полем, которое идентифицируют метод кодирования *Ptable* для текущей *Ptable*.

6.2.1.1.9.5 *PCM*

PCM является 3-битовым целым числом без знака, которое содержит параметр *Ptable Coding M*, использующийся для декодирования по Райсу записей *Ptable* текущей *Ptable*. Минимальное позволенное значение для *PCM* является нулем. Максимальное допустимое значение для *PCM* равно 4.

6.2.1.1.9.6 *RL_Bit*

RL_Bit содержит единственные код длины серии, который состоит из нулей с завершающей единицей. Самый короткий код длины серии равен '1'.

6.2.1.1.9.7 *LSB*

Младшие значащие биты *PCM* абсолютного значения прогнозируемой записи сохраняются в *LSB*.

6.2.1.1.9.8 *Sign*

Sign (знак) представляет собой бит, который указывает, является ли предсказанная запись положительной (*Sign* = '0') или отрицательной (*Sign* = '1').

6.2.1.1.10 *Arithmetic_Coded_Data*

Синтаксис *Arithmetic_Coded_Data* определяется в таблице 12. Длина *Arithmetic_Coded_Data* не кодируется.

6.2.1.1.10.1 *A_Data*

A_Data [ ] содержит арифметический код и биты заполнения.

Биты заполнения добавляются в конце арифметического кода, чтобы выровнять *Audio_Frame* до границы байта. Число битов заполнения составляет 0 ... 7. Значение битов заполнения должно быть нулем.

A_Data [ ] используется функцией "*input next bit D*". Минимальная длина *A_Data* составляет нуль битов. Если длина *A_Data* не равна нулю, у *A_Data* [0] должно быть значение нуль. Максимальная длина арифметического кода является числом битов, обработанных "*input next bit D*". Разрешается, чтобы конечные нули арифметического кода не были закодированы в *A_Data* [ ].

7 Эталонная модель декодера *DST*

7.1 Процессы декодирования *DST*

Параметры, обработанные и извлеченные из потока, используются, чтобы декодировать кодированный фрейм *DST*. Этот подпункт объясняет процессы декодирования, необходимые для кодированных фреймов *DST*.

7.1.1 Введение

Существуют три функции: арифметический декодер, мультиплексор/демультиплексор и ряд моделей источника. Арифметический декодер получает последовательность битов (*D=A_Data*) и последовательность вероятностей (*P*) и генерирует последовательность битов (*E*). Последовательности *E* и *P* присваиваются моделям источника в циклическом порядке, которым управляет мультиплексор/демультиплексор. Каждая модель источника получает необходимые параметры, как коэффициенты фильтра прогноза и записи таблицы вероятности из потока, как определено в синтаксисе и семантике.

Модель источника *S* соответствует каналу *S*. Вывод *X* из модели источника *S* является сигналом *DSD* для канала *S*.

7.1.2 Арифметический декодер

Арифметическое кодирование является методом кодирования переменной длины для сжатия данных близко к их энтропии. Кодированные данные представляются как число. Число использует столько цифр, сколько требуется, чтобы однозначно определять исходные данные.

Таблица 23 определяет переменные, используемые на рисунке 2, рисунке 3, рисунке 4 и рисунке 5.

Таблица 23 – Переменные, используемые в рисунке 2, рисунке 3, рисунке 4 и рисунке 5

Имя	Характеристики	Описание
A	12-битовый регистр	Это целое число без знака представляет текущую величину интервала арифметического декодера
C	12-битовый регистр	Это целое число без знака содержит часть битов арифметического кода
K	4-битовая переменная	Это целое число без знака является 4-битовым приближением *A*
P	8-битовая переменная	Это целое число без знака является величиной вероятности, применяемой к арифметическому декодеру
Q	12-битовая переменная	Это целое число без знака является произведением *K* и *P*

Рисунок 2 показывает полную блок-схему алгоритма декодирования DST.

Процесс инициализации показан на рисунке 3 и требуется в начале декодирования каждого фрейма. Он состоит из загрузки первых 13 битов арифметического кода в регистр *C* и переустановки регистра *A* в 4095. Первый бит, считываемый в *C,* будет перезаписан, это предусмотрено, потому что первый бит всегда 0.

Функция "*Input next bit D*" (ввод следующего бита *D*) на рисунке 2, рисунке 3 и рисунке 5 означает, что бит *D* берется из *A_Data* [ ], начиная с первого бита. После того, как все биты из *A_Data* [ ] были считаны, функция "*Input next bit D*" устанавливает бит *D* в 0.

Чтение информации о вероятности *P* получает из модели источника, за исключением первого бита, где вместо этого читается вероятность *DST_X_Bit*.

Рисунок 4 иллюстрирует декодирование одного бита *E* и обновление регистра *A* и *C*. Сначала вычисляется текущий аппроксимированный размер интервала *K*. Затем произведение аппроксимированного размера интервала *K* и применяемой величины вероятности *P* сохраняется в *Q*. Если *C* больше или равно *A*-*Q*, то арифметический код находится в верхней части интервала, что означает, что исходный закодированный бит *E* был битом '1'; иначе был передан бит '0'. *A* и *C* должны

быть скорректированы таким же образом как в кодере.

Процесс ренормализации показан на рисунке 5. Ренормализация требуется когда значение *A* слишком маленькое. Если *A* является слишком маленьким, то *A* и *C* смещаются на один бит влево, и новый бит арифметического кода *D* читается в младший значащий бит *C*.

Возможно, что последние биты *A_Data* не используются функцией "*Input next bit D*" для того, чтобы декодировать аудиофрейм. Эти неиспользованные биты являются битами заполнения для выравнивания аудиофрейма на границе байта.

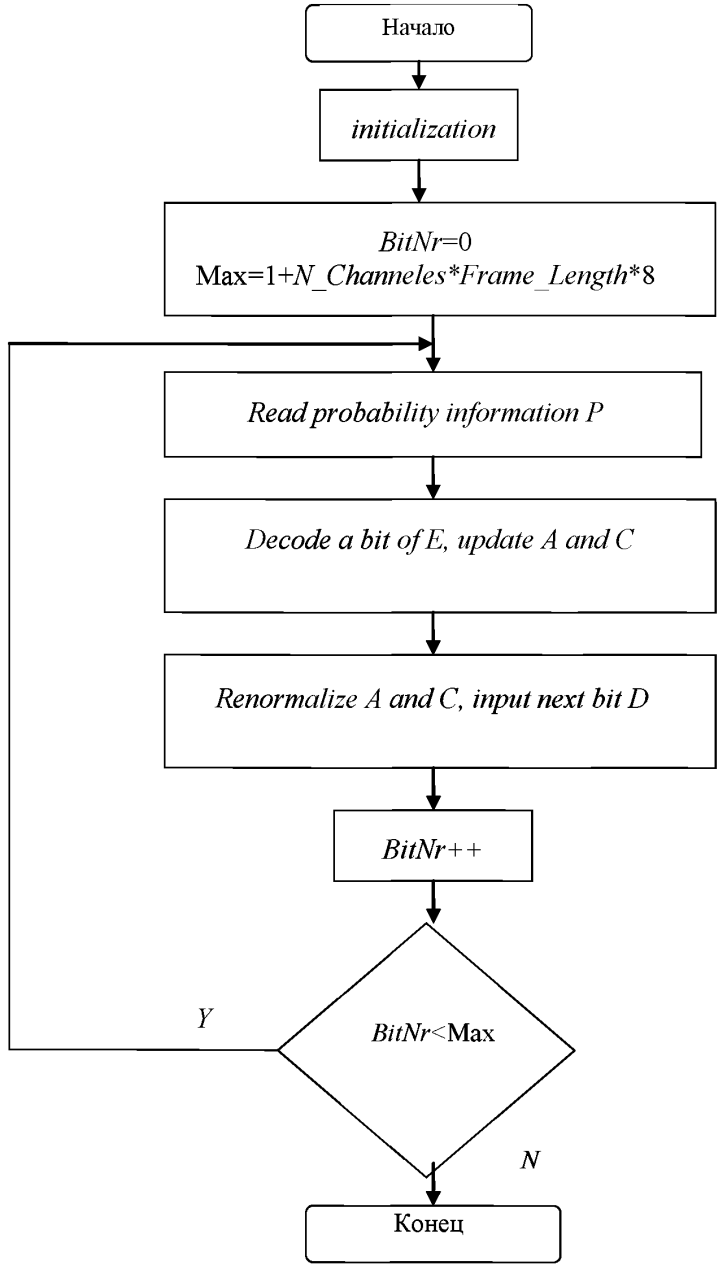

Рисунок 2 – Блок-схема арифметического декодера

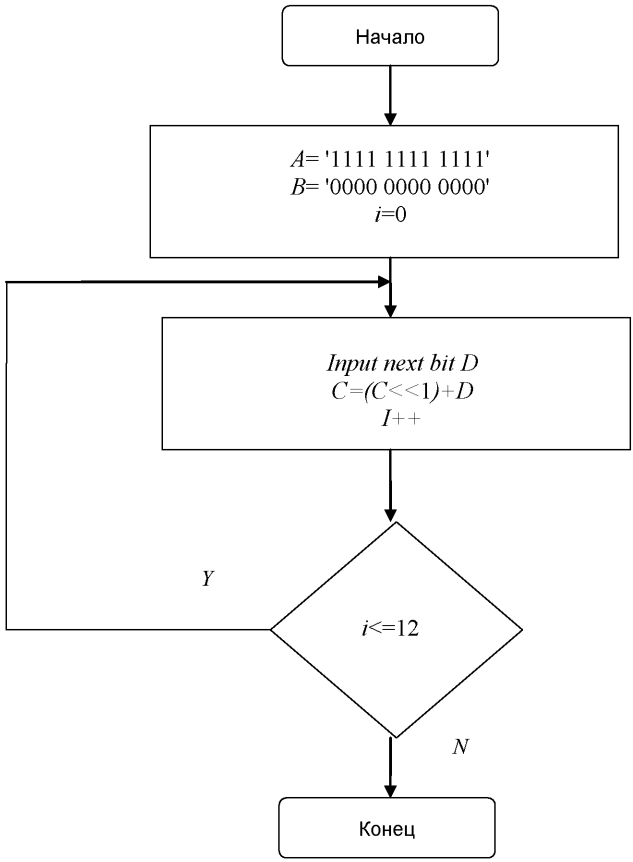

Рисунок 3 – Инициализация

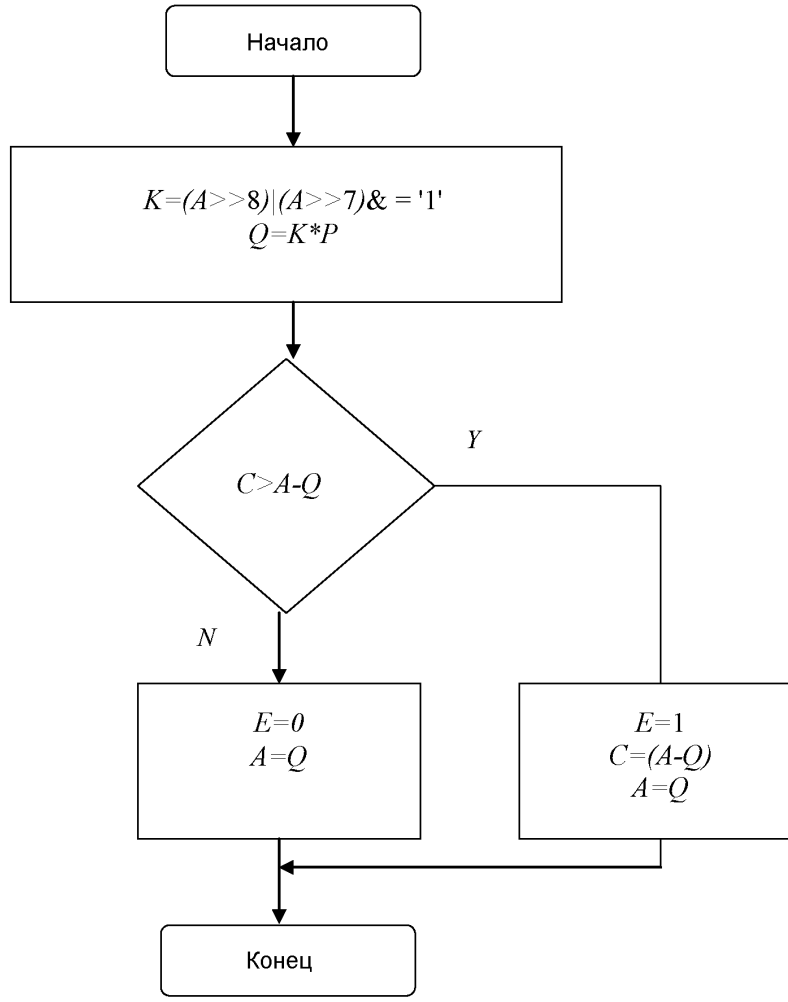

Рисунок 4 – Декодирование бита *E*, обновление *A* и *C*

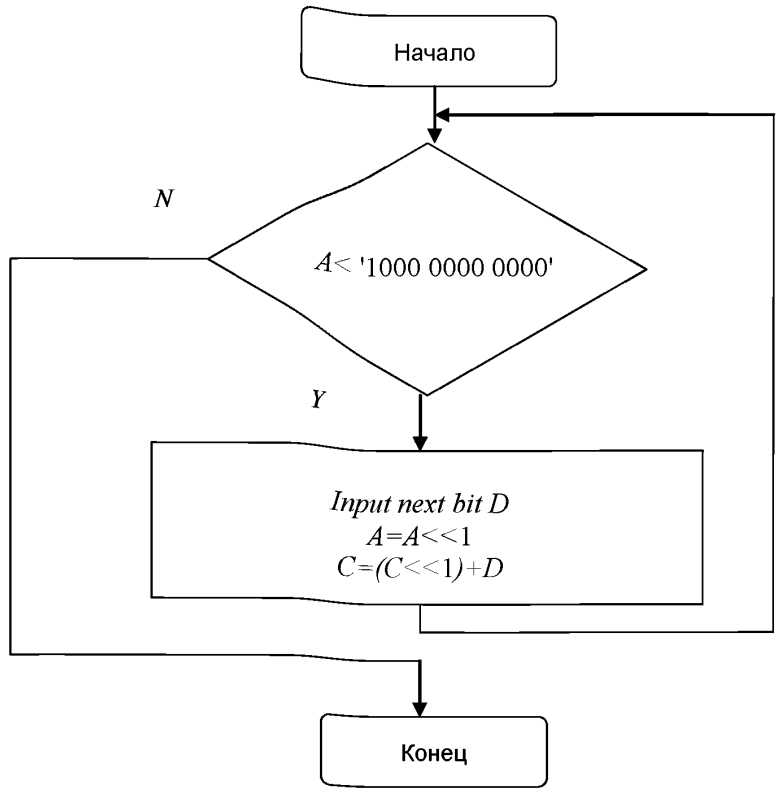

Рисунок 5 – Ренормализация *A* и *C*, ввод следующего бита(ов) *D*

7.1.3 Модель источника

Процесс декодирования в модели источника описывается только для одного канала, поскольку он эквивалентен для всех каналов.

Сегментация и отображение определяют, какой фильтр прогноза и какая *Ptable* должны использоваться, чтобы декодировать следующий бит звукового канала. Две функции *Filter_N*(*n*) и *Ptable_N*(*n*) возвращают номер фильтра прогноза и *Ptable*, используемых для того, чтобы декодировать бит *n* звукового канала. Функции *Filter_N*(*n*) и *Ptable_N*(*n*) используют информацию сегментации и отображения. *Filter_N*(*n*) определяется следующим образом:

Filters.Start[1] = 0

Filters.Start[*Seg*+1] = *Filters.Start*[*Seg*] + *Filters.Segment_Length*[*Channel_Nr*][*Seg*],

где: *Seg* = 1 .. *Filters.Nr_Of_Segments*[*Channel_Nr*] и *Seg* является номером сегмента звукового канала *Channel_Nr*.

Для бита *n* переменный *Seg* может быть определен:

```
if ((n>>3) >= Filter.Start[Filters.Nr_Of_Segments[Channel_Nr]])
{
        Seg = Filters.Nr_Of_Segments[Channel_Nr]
}
else
{
        Filter.Start[Seg] <= (n<<3) < Filters.Start[Seg+1]
}
```

Filter_N(*n*) может быть найдено путем использования формулы *Filter_N*(*n*) = *Filter*[*Channel_Nr*] [*Seg*] .

Функция *Ptable_N* (*n*) определяется, используя тот же самый механизм путем замены *Filters* на *Ptables* и замены *Filter_N* на *Ptable_N.*

Filters.Segment_Length[ ][ ] и *Ptables.Segment_Length[ ][ ]* определяются в 6.2.1.1.5.

Filters.Nr_Of_Segments[ ] и *Ptables.Nr_Of_Segments*[] определяются в 6.2.1.1.5.

Filter[ ][ ] и *Ptable*[ ][ ] определяется в 6.2.1.1.6.

Filter_N(*n*) и *Ptable_N*(*n*) используются в следующих определениях:

N	*Pred_Order[Filter_N(n)]*	Порядок прогноза фильтра прогноза, который использует текущий сегмент.
H [ ]	*Coef [Filter_N(n)][]*	Коэффициенты фильтра прогноза, который использует текущий сегмент.
L	*Ptable_Len[Ptable_N(n)]*	Длина *Ptable*, которую использует текущий сегмент.
T [ ]	*P_one [Ptable_N(n)][]*	Записи *Ptable,* которую использует текущий сегмент.
n	Номер бита в диапозоне 0…8**Frame_Length* – 1	Переменная, которая проходит через все биты текущего аудиофрейма.

7.1.3.1 Инициализация

Инициализация фильтра прогноза в начале каждого фрейма определяется следующим образом:

$$Y[m] = (-1)^m \quad \text{для} \quad -N \le m < 0$$

Выходное значение фильтра прогноза определяется как:

$$Z[n] = \sum_{i=0}^{N-1} Y[n-1-i]H[i]$$

Q-функция преобразовывает *Z* в *F* следующим образом:

$$F[n] = \begin{cases} 1 \text{ если } Z[n] \ge 0 \\ 0 \text{ если } Z[n] < 0 \end{cases}$$

Есть два метода для того, чтобы применить значения вероятности к арифметическому декодеру. В случай *Half_Prob* [*Channel_Nr*] = '0' значение вероятности определяется так:

$$P[n] = T\left[\min\left(\left|Z(n)\right| >> 3, L-1\right)\right]$$

В случае, если *Half_Prob* [*Channel_Nr*] = '1' значение вероятности определяется:

$$P[n] = \begin{cases} 128 & \text{если } 0 \le n < N \\ T\left[\min\left(\left|Z(n)\right| >> 3, L-1\right)\right] & \text{если } n > N \end{cases}$$

Значение *P [n]* применяется к арифметическому декодеру, который возвратит значение *E[n]*. Выходное значение *X*[*n*] (выборка DSD) является исключающим ИЛИ для *E* [*n*] и *F* [*n*]:

$$X[n] = E[n] \oplus F[n]$$

Логическое значение *X*[*n*] преобразовывается в численное значение *Y*[*n*] следующим образом:

$$Y[n] = \begin{cases} +1 & \text{если } X[n] = 1 \\ -1 & \text{если } X[n] = 0 \end{cases}$$

7.1.4 Мультиплексирование/демультиплексирование

Устройство мультиплексирования/демультиплексирования соединяет каждую модель источника с арифметическим декодером в соответствующий момент.

Для удобочитаемости уравнений вводятся некоторые новые переменные:

C *N_Channels*, число используемых звуковых каналов.

N порядковый номер бита, диапазон: 0 .. 8**Frame_Length* - 1.

Для $1 \le i \le C$ имеем:

DST_X_Bit = E[0],
P [0] = P (DST_X_Bit),
E_i *[n] = E[C n + i],*
P [C · n + i] = P_i *[n].*

P (*DST_X_Bit*) должно быть взято из *Coef*[0][0]) следующим образом. Если *Coef*[0][0] = %$c_8c_7c_6c_5c_4c_3c_2c_1c_0$, то *P* (*DST_X_Bits*) должно быть равным %$0c_0c_1c_2c_3c_4c_5c_6$ + 1. c_X представляет значение отдельных битов с *x,* находящимся в диапазоне 0 .. 8. Значение *P* (*DST_X_Bit*) находится в диапазоне 1 .. 128.

7.2 Ограничения на кодированный *DST Audio_Frames*

Чтобы позволить оптимальный проект декодера *DST*, должны быть введены следующие ограничения для кодированных *DST* аудиофреймов.

7.2.1 Ограниченное количество ошибочно предсказанных выборок

Максимальное разрешенное число ошибочно предсказанных выборок в кодированном *Audio_Frame DST* является половиной числа выборок *DSD* во фрейме.

$$N_Errors_max = \frac{N_Channels * Frame_Length * 8}{2}$$

Общее количество ошибочных предсказанных выборок (*N_Errors*) во фрейме является суммой числа ошибочных предсказанных выборок на звуковой канал:

$$N_Errors = \sum_{i=1}^{N_Channels} \left(\sum_{n=0}^{(Frame_Length * 8)-1} E_i[n] \right),$$

где E_i [*n*] равно 0 (хороший прогноз) или 1 (неправильный прогноз).
Для каждого кодированного *DST* аудиофрейма должно применяться следующее правило:
N_Errors ≤ *N_Errors_max*

7.2.2 Требование к разработке таблицы вероятности

Определенные в этом пункте ограничения должны применяться к таблицам *Ptables*.

Для каждой *Ptable*, которая используется во фрейме, содержание определяется следующим алгоритмом:

Обратите внимание на все выборки фрейма, которые используют рассматриваемую *Ptable*, и подсчитайте для этих выборок сколько раз используется запись *Ptable(CA [ ] [ ])* и сколько раз сигнал *E* равен 1 для записи *Ptable (CW [ ] [ ])*

```
for (Ptable_Nr=0; Ptable_Nr<Nr_Of_Ptables; Ptable_Nr++)
{
    for (Entry_Nr=0; Entry_Nr<Ptable_Len[Ptable_Nr]; Entry_Nr++)
    {
        CA[Ptable_Nr][Entry_Nr] = 0; CW[Ptable_Nr][Entry_Nr] = 0;
    }
}
for (Channel_Nr=1; Channel_Nr<=N_Channels; Channel_Nr++)
{
    if (Half_Prob[Channel_Nr]==0)
    {
        Start = 0;
    }
    else
    {
        Start = Pred_Order[Filter[Channel_Nr][1]];
    }
    Stop = 0;
    for (Seg_Nr=1; Seg_Nr<=Ptables.Nr_Of_Segments[Channel_Nr]; Seg_Nr++)
    {
        Stop += 8*Ptables.Segment_Length[Channel_Nr][Seg_Nr];
        for (Bit_Nr=Start; Bit_Nr<Stop; Bit_Nr++)
```

```
        {
            Ptable_Nr = Ptable[Channel_Nr][Seg_Nr];
            Entry_Nr = min(|Z[Channel_Nr][Bit_Nr]|>>3, Ptable_Len[Ptable_Nr]-1);
                CA[Ptable_Nr][Entry_Nr]++;
            CW[Ptable_Nr][Entry_Nr] += EChannel_Nr[Bit_Nr];
        }
        Start = Stop;
    }
}
```

Для каждой записи *Ptable* вероятность для ошибочного прогноза получается из этих чисел следующим образом:

```
for (Ptable_Nr=0; Ptable_Nr<Nr_Of_Ptables; Ptable_Nr++)
{
    for (Entry_Nr=0; Entry_Nr<Ptable_Len[Ptable_Nr]; Entry_Nr++)
    {
        if (CA[Ptable_Nr][Entry_Nr] == 0)
        {
            P_min[Ptable_Nr][Entry_Nr] = 1;
        }
        else
        {
```

$$p = trunc\left(\frac{512 \cdot CW\left[Ptable_Nr\right]\left[Entry_Nr\right] + CA\left[Ptable_Nr\right]\left[Entry_Nr\right]}{2 \cdot CA\left[Ptable_Nr\right]\left[Entry_Nr\right]}\right)$$

```
            P_min[Ptable_Nr][Entry_Nr] = min(max(p, 1), 128);
        }
    }
}
```

P_min [ ] [ ] являются минимальными разрешенными значениями вероятности для записей *Ptables*. Для каждой записи каждой *Ptable* вероятности, фактически используемые для кодирования (*P_one* [ ] [ ]), должны соответствовать следующему условию:

$$P_min\ [Ptable_Nr]\ [Entry_Nr] \le P_one\ [Ptable_Nr]\ [Entry_Nr] \le 1\ .$$

Приложение А
(справочное)

Описание кодера

А.1 Технический обзор

В алгоритме аудиокодирования без потерь имеют место три основных процесса: формирование фреймов (структурирование), прогноз и кодирование энтропии. В кодере входящие одноразрядные данные сначала структурируются, а затем передаются этапу прогноза. После этапа прогноза ошибочный сигнал, вычисленный на основе прогноза и исходного сигнала, передается этапу кодирования энтропии. На этом последнем этапе ошибочный сигнал кодируется, используя арифметическое кодирование, у которого производительность близка к (оптимальному) кодированию энтропии. Фильтры прогноза и таблицы вероятности могут быть сгруппированы для каналов с целью еще лучшей эффективности кодирования. Сгенерированные арифметическим кодером данные объединяются с коэффициентами прогноза и мультиплексируются в потоке битов.

А.1.1 Структурирование

Цель структурирования (формирования фрейма) является двоякой. Во-первых, структурирование необходимо, чтобы обеспечить легкий, "случайный" доступ к аудиоданным во время воспроизведения. По той же самой причине каждый фрейм должен быть независимо закодирован, что позволяет проигрывателю декодировать отдельные фреймы без знания о предыдущих фреймах. Во-вторых, структурирование позволяет аудиоконтенты во фрейме расценивать как стационарные (или по крайней мере, квазистационарные). Это - базовое предположение в процессе прогноза.

Процесс структурирования делит исходный одноразрядный аудиопоток, состоящий из выборок $b \in \{0, 1\}$ на фреймы длиной M=37,632 битов, соответствующие 1/75 с, принимая частоту дискретизации 2,8224 *MS*/с.

А.1.2 Прогноз

Фильтрация прогноза является первым необходимым шагом в процессе сжатия (аудио) данных. Шаг фильтрации прогноза пытается удалить избыточность из аудиопотока битов b, создавая новый поток битов e, который не избыточен. Вместе с коэффициентами фильтра прогноза h поток с ошибками e переносит ту же самую информацию, что и b. Фильтр прогноза обозначается как $z^{-1}H(z)$, чтобы подчеркнуть тот факт, что передача фильтра содержит задержку, которая обязательна, чтобы создать кодер, который может быть инвертирован по времени.

Фильтр прогноза *FIR* может быть разработан согласно стандартным методам, самый известный из них основан на минимальной среднеквадратической ошибке (*MMSE*). Применение критерия *MMSE* приводит к уравнению ошибочного прогноза, которое должно быть минимизировано:

$$\sum_{n=1}^{M} \varepsilon^2[n] = \sum_{n=1}^{M} \left(\sum_{i=1}^{L} h[i] b[n-i] - b[n] \right)^2,$$

где M является числом битов на фрейм и L – числом коэффициентов прогноза, которые кодируются как *'Coded_Pred_Order*+1'. После дальнейшего манипулирования это приводит к коэффициентам h. Фильтр *FIR*, найденный таким образом, будет минимальным фазовым фильтром. Чтобы получить оптимальный баланс между точностью прогноза и числом битов, взятых описанием фильтра прогноза, коэффициенты фильтра прогноза квантуются в 9-разрядные числа с фиксированной точкой первым масштабированием их на 256. Полученные коэффициенты сохраняются в *'Coef[Filter_Nr*] [*Coef_Nr*]', где *Filter_Nr* обозначает индекс фильтра, а *Coef_Nr* - индекс коэффициента фильтра, соответственно. Сигнал прогноза z является мультибитовым. Биты прогноза q получаются из мультибитовых значений z простым усечением, обозначенным блоком, который маркирован как $Q(z)$. Ошибка между потоком битов b и мультибитовым сигналом z минимизируется, тогда как в идеале было бы минимизировано различие между b и q, однобитовая квантованная версия z. Это однако приводит к трудоемким расчетам.

Ошибочный сигнал e вычисляется операцией исключающего или (*XOR*) между b и q. Цель фильтра прогноза состоит в том, чтобы создать столько нулей в e, сколько возможно, поскольку это обеспечит существенное снижение объема данных энтропийным кодированием.

А.1.3 Арифметическое кодирование

Когда используются надлежащие фильтры прогноза, сигнал e будет состоять из большего количества нулей, чем единиц, и может таким образом привести к возможному усилению компрессии. Предположите, что вероятность '1' в e обозначается как p, тогда вероятность '0' равняется $(1-p)$. Минимальное число битов N_{bits}, которым может быть представлен в среднем единственный бит потока e, тогда равняется:

$$Nbits = -\left(p \log_2[p] + (1-p) \log_2[1-p]\right).$$

Предположим, что 90 % всех прогнозов корректны, тогда p = 0,1 и N_{bits} = 0,47. Как правило, возможна компрессия приблизительно с коэффициентом 2. В то время как этот подсчет, основанный на вычислении энтропии, представляет верхний предел достижимого сжатия, алгоритм, который при практических обстоятельствах приближается к этому пределу, является алгоритмом арифметического кодирования.

Методы арифметического кодирования могут успешно использоваться только тогда, когда доступна точная информация о вероятностях символов "0" или "1".

Вероятности символов, необходимые для арифметического кодирования, вычисляются, составляя гистограмму (или таблицу). Обозначая вероятность, что прогноз корректен $P(e=0)$, видим, что, так как $P(e=0) = 1 - P(e=1)$, нет необходимости вычислять две таблицы: для арифметического кодирования используется только таблица вероятности ошибки t для $P(e=1)$, и передается декодеру.

А.1.4 Мультиплексирование канала

В предыдущих разделах "модель источника", состоящая из фильтра прогноза и таблицы вероятности, обсуждалась для одного канала. В полном кодере у каждого канала имеется своя собственная модель источника, тогда как используется только единственный арифметический кодер. Однако чтобы использовать корреляцию между каналами также можно позволить каналам совместно использовать фильтры прогноза и/или таблицы вероятности. Совместное использование фильтров или таблиц вероятности выгодно, когда уменьшение числа битов метаданных, необходимых чтобы передать информацию о фильтре или таблице с кодера на декодер, выше, чем увеличение числа битов арифметического кода. Последнее число обычно будет несколько больше, так как не всегда возможно создать фильтр прогноза (или таблицу вероятности), который приводит к оптимальному арифметическому кодированию для всех каналов, которые его используют.

Арифметический кодер получает для каждого канала потоки e и p, которые поставляются отдельными моделями источника.

Библиография

[1] ИСО/МЭК 14496–3:2009 Информационные технологии. Кодирование аудиовизуальных объектов. Часть 3. Аудио (ИСО/МЭК14496–3:2009 *Information technology - Coding of audio-visual objects - Part* 3: *Audio*)

УДК 621.396:006.354 ОКС 33.170

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

Подписано в печать 05.11.2014. Формат 60x84 1/8.

Усл. печ. л. 3,72. Тираж 32 экз. Зак. 4094.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

ФГУП «СТАНДАРТИНФОРМ»

123995 Москва, Гранатный пер., 4.

gost.e-ecolog.ru gost.e-ecolog.ru

gost.e-ecolog.ru