ФЕДЕРАЛЬНОЕ АГЕНТСТВО

ПО ТЕХНИЧЕСКОМУ РЕГУЛИРОВАНИЮ И МЕТРОЛОГИИ

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

ГОСТ Р 53556.2— 2012

Звуковое вещание цифровое

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

Часть 3

(MPEG-4 audio)

Кодирование речевых сигналов с использованием гармонических векторов — HVXC

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

Издание официальное

Москва
Стандартинформ
2020

Предисловие

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

2 ВНЕСЕН Техническим комитетом по стандартизации ТК 480 «Связь»

3 УТВЕРЖДЕН И ВВЕДЕН В ДЕЙСТВИЕ Приказом Федерального агентства по техническому регулированию и метрологии от 20 ноября 2012 г. № 941-ст

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

5 ВВЕДЕН ВПЕРВЫЕ

6 ПЕРЕИЗДАНИЕ. Июнь 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 (www.gost.ru)

Содержание

НАЦИОНАЛЬНЫЙ СТАНДАРТ РОССИЙСКОЙ ФЕДЕРАЦИИ

Звуковое вещание цифровое

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

Часть 3

(MPEG-4 AUDIO)

Кодирование речевых сигналов с использованием гармонических векторов — HVXC

Sound broadcasting digital.
Coding of signals of sound broadcasting with reduction of redundancy for transfer on digital communication channels.
Part 3 (MPEG-4 audio). Harmonic Vector Excitation Coding

Дата введения — 2013—09—01

1 Область применения

Параметрическое речевое кодирование MPEG-4 использует алгоритм гармонического кодирования с векторным возбуждением (*HVXC*), где используется гармоническое кодирование остаточных сигналов *LPC* для речевых сегментов и кодирование с векторным возбуждением (*VXC*) для неречевых сегментов. *HVXC* позволяет кодировать речевые сигналы на 2,0 Кбит/с и 4,0 Кбит/с масштабируемой схемой, где возможно декодирование потока 2,0 Кбит/с, используя поток битов 2,0 Кбит/с и поток битов 4,0 Кбит/с. *HVXC* также обеспечивает кодирование потока битов с варьируемой битовой скоростью, где типичная средняя скорость передачи информации в битах составляет приблизительно 1,2—1,7 Кбит/с. Возможно независимое изменение скорости и шага во время декодирования, которое является мощной функциональной возможностью для быстрого поиска в базе данных. Длина фрейма равна 20 мс, и может быть выбрана одна из четырех различных алгоритмических задержек 33,5 мс, 36 мс, 53,5 мс, 56 мс.

Кроме того, как расширение *HVXC*, тип объекта *ER_HVXC* предлагает эластичный синтаксис ошибок и режим с переменной скоростью передачи данных на 4,0 Кбит/с.

2 Термины и определения

Термины и определения — в соответствии с ГОСТ Р 53556.0—2009.

3 Синтаксис потока битов

Естественный Звуковой Объектный *HVXC/ER_HVXC* передается в одном или двух элементарных потоках: потоке базового уровня и опционном потоке уровня расширения.

Когда инструмент *HVXC* используется с инструментом защиты от ошибок, таким как инструмент MPEG-4 *EP*, должен использоваться порядок битов, упорядоченных в соответствии с чувствительностью к ошибкам. *HVXC* с эластичным синтаксисом ошибок и режимом переменной скорости передачи данных на 4,0 Кбит/с, описанным в 3.3 и 3.4, называют *ER_HVXC*.

Синтаксис потока битов описан в коде pseudo-C.

3.1 Конфигурация декодера (*HvxcSpecificConfig*)

Информация о конфигурации декодера для типа объекта *HVXC* передается в *DecoderConfigDescriptor* () базового уровня и *Elementary Stream* опционного уровня расширения.

Издание официальное

Требуется следующий *HvxcSpecificConfig* ():

Синтаксис	Количество битов	Мнемоника
HvxcSpecificConfig(){ *isBaseLayer* *if (isBaseLayer) {* *HVXCconfig()* *}* *}*	1	*uimcbf*

Тип объекта *HVXC* обеспечивает немасштабируемые режимы и масштабируемый режим базового уровня 2,0 Кбит/с плюс уровня расширения на 2,0 Кбит/с. В этом масштабируемом режиме конфигурация базового уровня должна быть следующей:

HVXCvarMode = 0 *HVXC fixed bit rate*
HVXCrateMode = 0 *HVXC 2kbps*
isBaseLayer = 1 *base layer*

Таблица 1 — Синтаксис *HVXCconfig* ()

Синтаксис	Количество битов	Мнемоника
HVXCconfig() *{* *HVXCvarMode;* *HVXCrateMode;* *extensionFlag;* *if (extensionFlag) {* *< to be defined in MPEG-4 Version 2* *>*	 1 2 1 	 *uimcbf* *uimcbf* *uimcbf*

Таблица 2 — Режим *HVXCvarMode*

HVXCvarMode	Описание
0	*HVXC fixed bit rate*
1	*HVXC variable bit rate*

Таблица 3 — Режим *HVXCrateMode*

HVXCrateMode	*HVXCrate*	Описание
0	2000	*HVXC* 2,0 Кбит/с
1	4000	*HVXC* 4,0 Кбит/с
2	3700	*HVXC* 3,7 Кбит/с

Таблица 4 — Константы *HVXC*

NUM_SUBF1	*NUM_SUBF2*
2	4

3.2 Фрейм потока битов (*alPduPayload*)

Динамические данные для типа объекта *HVXC* передаются как полезная нагрузка *AL-PDU* в базовом уровне и опционном уровне расширения *Elementary Stream*.

Базовый уровень *HVXC* — полезная нагрузка Модуля Доступа.

alPduPayload {
HVXCframe();
}

Уровень расширения *HVXC* — полезная нагрузка Модуля Доступа.

Чтобы анализировать и декодировать уровень расширения *HVXC*, требуется информация, декодированная из базового уровня *HVXC*.

```
alPduPayload {
    HVXCenhaFrame()
}
```

Таблица 5 — Синтаксис фрейма *HVXCframe* ()

Синтаксис	Количество битов	Мнемоника
HVXCframe() *{* *if (HVXCvarMode == 0) {* *HVXCfixframe(HVXCrate);* *}* *else {* *HVXCvarframe();* *}* *}*		

3.2.1 Фрейм потока битов *HVXC*

Таблица 6 — Синтаксис *HVXCfixframe (rate)*

Синтаксис	Количество битов	Мнемоника
HVXCfixframe(rate) *{* *if (rate >= 2000) {* *idLsp1();* *idVUV();* *idExc1();* *}* *if (rate >= 3700) {* *2();* *idExc2(rate);* *}* *}*		

Таблица 7 — Синтаксис *HVXCenhaFrame* ()

Синтаксис	Количество битов	Мнемоника
HVXCenhaFrame() *{* *idLsp2();* *idExc2(4000);* *}*		

Таблица 8 — Синтаксис *idLsp1* ()

Синтаксис	Количество битов	Мнемоника
idLsp1() *{* *LSP1;* *LSP2;* *LSP3;* *LSP4;* *}*	 5 7 5 1	 *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 9 — Синтаксис *idLsp2* ()

Синтаксис	Количество битов	Мнемоника
idLsp2() *{* *LSP5;* *}*	8	*uimcbf*

Таблица 10 — Синтаксис *idVUV* ()

Синтаксис	Количество битов	Мнемоника
idVUV() *{* *VUV;* *}*	8	*uimcbf*

Таблица 11 — *VUV* (для режима фиксированной скорости передачи битов)

VUV	Описание
0	*Unvoiced Speech*
1	*Mixed Voiced Speech*-1
2	*Mixed Voiced Speech*-2

Таблица 12 — Синтаксис *idExc1* ()

Синтаксис	Количество битов	Мнемоника
idExc1()		
{		
if (VUV != 0) {		
Pitch;	7	*uimcbf*
SE_shape1;	4	*uimcbf*
SE_shape2;	4	*uimcbf*
SE_gain;	5	
}		
else{		
for (sf_num=0; sf_num < NUM_SUBF1; sf_num++){		
VX_shape1 [sf_num];	6	*uimcbf*
VX_gain1 [sf_num];	4	*uimcbf*
}		
}		
}		

Таблица 13 — Синтаксис *idExc2 (rate)*

Синтаксис	Количество битов	Мнемоника
idExc2(rate)		
{		
if (VUV != 0) {		
SE_shape3;	7	*uimcbf*
SE_shape4;	10	*uimcbf*
SE_shape5;	9	*uimcbf*
if (rate >= 4000) {		
SE_shape6;	6	*uimcbf*
}		
}		

Окончание таблицы 13

Синтаксис	Количество битов	Мнемоника
else { *for(sf_num=0; sf_num < NUM_SUBF2-1;sf_num++){* *VX_shape2[sf_num];* *VX_gain2[sf_num];* *}* *if (rate >= 4000) {* *VX_shape2[3];* *VX_gain2[3];* *}* *}* *}*	 5 3 5 3 	 *uimcbf* *uimcbf* *uimcbf* *uimcbf*

*idLsp*1 (), *idExc*1 (), *idVUV* () обработаны как базовый уровень в случае масштабируемого режима. *idLsp*2 (), *idExc*2 () обработаны как уровень расширения в случае масштабируемого режима.

Таблица 14 — Синтаксис *HVXCvarframe* ()

Синтаксис	Количество битов	Мнемоника
HVXCvarframe() *{* *idvarVUV();* *idvarLsp1();* *idvarExc1()* *}*		

Таблица 15 — Синтаксис *idvarVUV* ()

Синтаксис	Количество битов	Мнемоника
idvarVUV() *{* *VUV;* *}*	2	*uimcbf*

Таблица 16 — *VUV* (для режима варьируемой битовой скорости)

VUV	Описание
0	*Unvoiced Speech*
1	*Background Noise*
2	*Mixed Voiced Speech*
3	*Voiced Speech*

Таблица 17 — Синтаксис *idvarLsp1* ()

Синтаксис	Количество битов	Мнемоника
idvarLsp1() *{* *if (VUV != 1) {* *LSP1;* *LSP2;* *LSP3;* *LSP4;* *}* *}*	 5 7 5 1 	 *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 18 — Синтаксис *idvarExc1* ()

Синтаксис	Количество битов	Мнемоника
idvarExc1() *{* *if (VUV != 1) {* *if (VUV != 0) {* *Pitch;* *SE_Shape1;* *SE_Shape2;* *SE_Gain;* *}* *else {* *for(sf_num=0;sf_num<NUM_SUBF1;sf_num++)* *{* *VX_gain1[sf_num];* *}* *}* *}* *}*	 7 4 4 5 4	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

3.3 Конфигурация декодера (*ErrorResilientHvxcSpecificConfig*)

Информация конфигурации декодера для типа объекта *ER_HVXC* передается в *DecoderConfigDescriptor ()* базового уровня и в *Elementary Stream* опционного уровня расширения.

Требуется следующий *ErrorResilientHvxcSpecificConfig ()*:

```
ErrorResilientHvxcSpecificConfig () {
  isBaseLayer 1 uimcbf
  if (isBaseLayer) {
    ErHVXCconfig();
  }
}
```

Тип объекта *ERH_VXC* обеспечивает немасштабируемые режимы и масштабируемые режимы. В масштабируемых режимах конфигурация базового уровня должна быть следующей:

HVXCrateMode = 0 ER_HVXC 2.0 kbit/s
isBaseLayer = 1 base layer

Таблица 19 — Синтаксис *ErHvxcConfig* ()

Синтаксис	Количество битов	Мнемоника
ErHVXCconfig() *{* *HVXCvarMode;* *HVXCrateMode;* *extensionFlag;* *if (extensionFlag) {* *var_ScalableFlag;* *}* *}*	 1 2 1 1	 *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 20 — Режим *HVXCvarMode*

HVXCvarMode	Описание
0	*ERH_VXC fixed bitrate*
1	*ERH_VXC variable*

Таблица 21 — *HVXCrateMode*

HVXCrateMode	*HVXCrate*	Описание
0	2000	*ER_HVXC 2,0 Кбит/с*
1	4000	*ER_HVXC 4,0 Кбит/с*
2	3700	*ER_HVXC 3,7 Кбит/с*
3 (*reserved*)		

Таблица 22 — *var_ScalableFlag*

var_ScalableFlag	Описание
0	*ER_HVXC variable rate non-scalable*
1	*ER_HVXC variable rate scalable mode*

3.4 Фрейм потока битов (*alPduPayload*)

Динамические данные для типа объекта *ER_HVXC* передаются как полезная нагрузка *AL-PDU* в базовом уровне и *Elementary Stream* опционного уровня расширения.

Базовый уровень *ER_HVXC* — полезная нагрузка Модуля Доступа.

```
alPduPayload {
    ErHVXCframe ();
}
```

Уровень расширения *ER_HVXC* — полезная нагрузка Модуля Доступа.

Чтобы анализировать и декодировать уровень расширения *ER_HVXC*, требуется информация, декодированная из базового уровня *ER_HVXC*.

```
alPduPayload {
    ErHVXCenhaFrame ();
}
```

Таблица 23 — Синтаксис *ErHVXCframe* ()

Синтаксис	Количество битов	Мнемоника
ErHVXCframe() *{* *if (HVXCvarMode == 0) {* *ErHVXCfixframe(HVXCrate);* *}* *else {* *ErHVXCvarframe(HVXCrate);* *}* *}*		

Таблица 24 — Синтаксис *ErHVXCenhaframe* ()

Синтаксис	Количество битов	Мнемоника
ErHVXCenhaframe() *{* *if (HVXCvarMode == 0) {* *ErHVXCenh_fixframe();* *}* *else {* *ErHVXCenh_varframe();* *}* *}*		

3.4.1 Синтаксис потока битов режима фиксированной скорости

Таблица 25 — Синтаксис *ErHVXCfixframe* ()

Синтаксис	Количество битов	Мнемоника
ErHVXCfixframe(rate) *{* *if (rate == 2000){* *2k_ESC0();* *2k_ESC1();* *2k_ESC2();* *2k_ESC3();* *}* *else if (rate >= 3700) {* *4k_ESC0(rate);* *4k_ESC1(rate);* *4k_ESC2();* *4k_ESC3();* *4k_ESC4(rate);* *}* *}*		

Таблица 26 — Синтаксис *2k_ESC0* ()

Синтаксис	Количество битов	Мнемоника
2k_ESC0() *{* *VUV , 1-0;* *if (VUV != 0) {* *LSP4, 0;* *SE_gain, 4-0;* *LSP1, 4-0;* *Pitch, 6-1;* *LSP2, 6;* *LSP3, 4;* *LSP2, 5;* *}* *else {* *LSP4, 0;* *VX_gain1[0], 3-0;* *VX_gain1[1], 3-0;* *LSP1, 4-0;* *LSP2, 6-3;* *LSP3, 4-3;* *}* *}*	 2 1 5 5 6 1 1 1 1 4 4 5 4 2	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 27 — Синтаксис *2k_ESC1* ()

Синтаксис	Количество битов	Мнемоника
2k_ESC1() *{* *if (VUV != 0) {* *SE_shape1, 3-0;* *}* *else {* *LSP2, 2-0;* *LSP3, 2;* *}* *}*	 4 3 1	 *uimcbf* *uimcbf* *uimcbf*

Таблица 28 — Синтаксис *2k_ESC2* ()

Синтаксис	Количество битов	Мнемоника
2k_ESC2() *{* *if (VUV != 0) {* *SE_shape2, 3-0;* *}* *else {* *LSP3, 1-0;* *VX_shape1[0], 5-4;* *}* *}*	 4 2 2	 *uimcbf* *uimcbf* *uimcbf*

Таблица 29 — Синтаксис *2k_ESC3* ()

Синтаксис	Количество битов	Мнемоника
2k_ESC3() *{* *if (VUV != 0) {* *LSP2, 4-0;* *LSP3, 3-0;* *Pitch, 0;* *}* *else {* *VX_shape1[0], 3-0;* *VX_shape1[1], 5-0;* *}* *}*	 5 4 1 4 6	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 30 — Синтаксис *4k_ESC0* ()

Синтаксис	Количество битов	Мнемоника
4k_ESC0() *{* *VUV , 1-0;* *if (VUV != 0) {* *LSP4, 0;* *SE_gain, 4-0;* *LSP1, 4-0;* *Pitch, 6-1;* *LSP2, 6-3;* *SE_shape3, 6-2;* *LSP3, 4;* *LSP5, 7;* *SE_shape4, 9;* *SE_shape5, 8;* *if (rate >= 4000) {* *SE_shape6, 5;* *}* *}* *else {* *LSP4, 0;* *VX_gain1[0], 3-0;* *VX_gain1[1], 3-0;* *LSP1, 4-0;* *LSP2, 6-3;* *LSP3, 4;* *LSP5, 7;*	 2 1 5 5 6 4 5 1 1 1 1 1 1 4 4 5 4 1 1	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Окончание таблицы 30

Синтаксис	Количество битов	Мнемоника
VX_gain2[0], 2-0;	3	*uimcbf*
VX_gain2[1], 2-0;	3	*uimcbf*
VX_gain2[2], 2-0;	3	*uimcbf*
if (rate >= 4000) {		
VX_gain2[3], 2-1;	2	*uimcbf*
}		
}		
}		

Таблица 31 — Синтаксис *4k_ESC1* ()

Синтаксис	Количество битов	Мнемоника
4k_ESC1(rate)		
{		
if (VUV != 0) {		
SE_shape4, 8-0;	9	*uimcbf*
SE_shape5, 7-0;	8	*uimcbf*
if (rate >= 4000) {		
SE_shape6, 4-0;	5	*uimcbf*
}		
}		
else {		
if (rate >= 4000) {		
VX_gain2[3], 0;	1	*uimcbf*
}		
LSP2, 2-0;	3	*uimcbf*
LSP3, 3-0;	4	*uimcbf*
LSP5, 6-0;	7	*uimcbf*
VX_shape1[0], 5-0;	6	*uimcbf*
VX_shape1[1], 5;	1	*uimcbf*
}		
}		

Таблица 32 — Синтаксис *4k_ESC2* ()

Синтаксис	Количество битов	Мнемоника
4k_ESC2()		
{		
if (VUV != 0) {		
SE_shape1, 3-0;	4	*uimcbf*
}		
else {		
VX_shape1[1], 4-1;	4	*uimcbf*
}		
}		

Таблица 33 — Синтаксис *4k_ESC3* ()

Синтаксис	Количество битов	Мнемоника
4k_ESC3()		
{		
if (VUV != 0) {		
SE_shape2, 3-0;	4	*uimcbf*
}		
else {		
VX_shape1[1], 0;	1	*uimcbf*
VX_shape2[0], 4-2;	3	*uimcbf*
}		
}		

Таблица 34 — Синтаксис *4k_ESC4* ()

Синтаксис	Количество битов	Мнемоника
4k ESC4(rate)		
{		
if (VUV != 0) {		
LSP2, 2-0;	3	*uimcbf*
LSP3, 3-0;	4	*uimcbf*
LSP5, 6-0;	7	*uimcbf*
Pitch, 0;	1	*uimcbf*
SE_shape3, 1-0;	2	*uimcbf*
}		
else {		
VX_shape2[0], 1-0;	2	*uimcbf*
VX_shape2[1], 4-0;	5	*uimcbf*
VX_shape2[2], 4-0;	5	*uimcbf*
if (rate >= 4000) {		
VX_shape2[3], 4-0;	5	*uimcbf*
}		
}		
}		

3.4.2 Синтаксис потока битов для масштабируемого режима

Синтаксис потока битов базового уровня для масштабируемого режима такой же самый, как синтаксис *ErHVXCfixframe* (2000). Синтаксис потока битов уровня расширения *ErHVXCenhaFrame* () для масштабируемого режима показан ниже.

Таблица 35 — Синтаксис *ErHVXCenh_fixframe* ()

Синтаксис	Количество битов	Мнемоника
ErHVXCenh fixframe()		
{		
Enh_ESC0();		
Enh_ESC1();		
Enh_ESC2();		
}		

Таблица 36 — Синтаксис *Enh_ESC0* ()

Синтаксис	Количество битов	Мнемоника
Enh_ESC0()		
{		
if (VUV != 0) {		
SE_shape3, 6-2;	5	*uimcbf*
LSP5, 7;	1	*uimcbf*
SE_shape4, 9;	1	*uimcbf*
SE_shape5, 8;	1	*uimcbf*
SE_shape6, 5;	1	*uimcbf*
SE_shape4, 8-6;	3	*uimcbf*
}		
else {		
LSP5, 7;	1	*uimcbf*
VX gain2[0], 2-0	3	*uimcbf*
VX gain2[1], 2-0	3	*uimcbf*
VX gain2[2], 2-0	3	*uimcbf*
VX_gain2[3], 2-1	2	*uimcbf*
}		
}		

Таблица 37 — Синтаксис *Enh_ESC1* ()

Синтаксис	Количество битов	Мнемоника
Enh_ESC1()		
{		
if (VUV != 0) {		
SE_shape4, 5-0;	6	*uimcbf*
SE_shape5, 7-0;	8	*uimcbf*
SE_shape6, 4-0;	5	*uimcbf*
}		
else {		
VX_gain2[3], 0;	1	*uimcbf*
LSP5, 6-0;	7	*uimcbf*
VX shape2[0], 4-0;	5	*uimcbf*
VX shape2[1], 4-0;	5	*uimcbf*
VX_shape2[2], 4;	1	*uimcbf*
}		
}		

Таблица 38 — Синтаксис *Enh_ESC2* ()

Синтаксис	Количество битов	Мнемоника
Enh_ESC2()		
{		
if (VUV != 0) {		
LSP5, 6-0;	7	*uimcbf*
SE_shape3, 1-0;	2	*uimcbf*
}		
else {		
VX_shape2[2], 3-0;	4	*uimcbf*
VX_shape2[3], 4-0;	5	*uimcbf*
}		
}		

3.4.3 Синтаксис потока битов режима с варьируемой скоростью передачи данных

Таблица 39 — Синтаксис *ErHVXCvarframe* ()

Синтаксис	Количество битов	Мнемоника
ErHVXCvarframe(rate)		
{		
if (rate == 2000) {		
if (var_ScalableFlag == 1) {		
BaseVar_ESC0();		
BaseVar_ESC1();		
BaseVar_ESC2();		
BaseVar_ESC3();		
} else {		
Var2k_ESC0();		
Var2k_ESC1();		
Var2k_ESC2();		
Var2k_ESC3();		
}		
} else {		
Var4k_ESC0();		
Var4k_ESC1();		
Var4k_ESC2();		
Var4k_ESC3();		
Var4k_ESC4();		
}		
}		

Таблица 40 — Синтаксис *Var2k_ESC0* ()

Синтаксис	Количество битов	Мнемоника
Var2k_ESC0()		
{		
VUV, 1-0;	2	*uimcbf*
if (VUV == 2 || VUV == 3) {		
LSP4, 0;	1	*uimcbf*
SE_gain, 4-0;	5	*uimcbf*
LSP1, 4-0;	5	*uimcbf*
Pitch, 6-1;	6	*uimcbf*
LSP2, 6;	1	*uimcbf*
LSP3, 4;	1	*uimcbf*
LSP2, 5;	1	*uimcbf*
}		
else if (VUV == 0) {		
LSP4, 0;	1	*uimcbf*
VX_gain1[0], 3-0;	4	*uimcbf*
VX_gain1[1], 3-0;	4	*uimcbf*
LSP1, 4-0;	5	*uimcbf*
LSP2, 6-3;	4	*uimcbf*
LSP3, 4-3;	2	*uimcbf*
}		
}		

Таблица 41 — Синтаксис *Var2k_ESC1* ()

Синтаксис	Количество битов	Мнемоника
Var2k_ESC1()		
{		
if (VUV == 2 || VUV == 3) {		
SE_shape1, 3-0;	4	*uimcbf*
}		
else if (VUV == 0) {		
LSP2, 2-0;	3	*uimcbf*
LSP3, 2;	1	*uimcbf*
}		
}		

Таблица 42 — Синтаксис *Var2k_ESC2* ()

Синтаксис	Количество битов	Мнемоника
Var2k_ESC2()		
{		
if (VUV == 2 || VUV == 3) {		
SE_shape2, 3-0;	4	*uimcbf*
}		
else if (VUV == 0) {		
LSP3, 1-0;	2	*uimcbf*
}		
}		

Таблица 43 — Синтаксис *Var2k_ESC3* ()

Синтаксис	Количество битов	Мнемоника
Var2k_ESC3()		
{		
if (VUV == 2 || VUV == 3) {		
LSP2, 4-0;	5	*uimcbf*
LSP3, 3-0;	4	*uimcbf*
Pitch, 0;	1	*uimcbf*
}		
}		

Таблица 44 — Синтаксис *Var4k_ESC0* ()

Синтаксис	Количество битов	Мнемоника
Var4k_ESC0()		
{		
VUV,1-0;	2	*uimcbf*
if (VUV == 2 || VUV == 3) {		
LSP4, 0;	1	*uimcbf*
SE gain,4-0;	5	*uimcbf*
LSP1, 4-0;	5	*uimcbf*
Pitch, 6-1;	6	*uimcbf*
LSP2, 6-3;	4	*uimcbf*
SE_shape3, 6-2;	5	*uimcbf*
LSP3, 4;	1	*uimcbf*
LSP5, 7;	1	*uimcbf*
SE_shape4, 9;	1	*uimcbf*
SE_shape5, 8;	1	*uimcbf*
SE_shape6, 5;	1	*uimcbf*
}		
else if (VUV == 0) {		
LSP4, 0;	1	*uimcbf*
VX_gain1[0], 3-0;	4	*uimcbf*
VX_gain1[1], 3-0;	4	*uimcbf*
LSP1, 4-0;	5	*uimcbf*
LSP2, 6-3;	4	*uimcbf*
LSP3, 4-3;	2	*uimcbf*
}		
else {		
UpdateFlag, 0;	1	*uimcbf*
if (UpdateFlag == 1) {		
LSP4, 0;	1	*uimcbf*
VX_gain1[0], 3-0;	4	*uimcbf*
LSP1, 4-0;	5	*uimcbf*
LSP2, 6-3;	4	*uimcbf*
LSP3, 4-3;	2	*uimcbf*
}		
}		
}		

Таблица 45 — Синтаксис *Var4k_ESC1* ()

Синтаксис	Количество битов	Мнемоника
Var4k_ESC1()		
{		
if (VUV == 2 || VUV == 3) {		
SE_shape4, 8-0;	9	*uimcbf*
SE_shape5, 7-0;	8	*uimcbf*
SE_shape6, 4-0;	5	*uimcbf*
}		
else if (VUV == 0) {		
LSP2, 2-0;	3	*uimcbf*
LSP3, 2-0;	3	*uimcbf*
VX_shape1[0], 5-0;	6	*uimcbf*
VX_shape1[1], 5-0;	6	*uimcbf*
}		
else {		
if (UpdateFlag == 1) {		
LSP2, 2-0;	3	*uimcbf*
LSP3, 2-0;	3	*uimcbf*
}		
}		
}		

Таблица 46 — Синтаксис *Var4k_ESC2* ()

Синтаксис	Количество битов	Мнемоника
Var4k_ESC2() *{* *if (VUV == 2 || VUV == 3) {* *SE_shape1, 3-0;* *}* *}*	4	*uimcbf*

Таблица 47 — Синтаксис *Var4k_ESC3* ()

Синтаксис	Количество битов	Мнемоника
Var4k_ESC3() *{* *if (VUV == 2 || VUV == 3) {* *SE_shape2, 3-0;* *}* *}*	4	*uimcbf*

Таблица 48 — Синтаксис *Var4k_ESC4* ()

Синтаксис	Количество битов	Мнемоника
Var4k_ESC4() *{* *if (VUV == 2 || VUV == 3) {* *LSP2, 2-0;* *LSP3, 3-0;* *LSP5, 6-0;* *Pitch, 0;* *SE_shape3, 1-0;* *}* *}*	 3 4 7 1 2	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 49 — Синтаксис *BaseVar_ESC0* ()

Синтаксис	Количество битов	Мнемоника
BaseVar_ESC0() *{* *VUV, 1-0;* *if (VUV == 2 || VUV == 3) {* *LSP4, 0;* *SE_gain, 4-0;* *LSP1, 4-0;* *Pitch, 6-1;* *LSP2, 6;* *LSP3, 4;* *LSP2, 5;* *}* *else if (VUV == 0) {* *LSP4, 0;* *VX_gain1[0], 3-0;* *VX_gain1[1], 3-0;* *LSP1, 4-0;* *LSP2, 6-3;* *LSP3, 4-3;* *}*	 2 1 5 5 6 1 1 1 1 4 4 5 4 2	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Окончание таблицы 49

Синтаксис	Количество битов	Мнемоника
else {		
UpdateFlag, 0;	1	*uimcbf*
if (UpdateFlag ==1) {		
LSP4, 0;	1	*uimcbf*
VX_gain1[0], 3-0;	4	*uimcbf*
LSP1, 4-0;	5	*uimcbf*
LSP2, 6-3;	4	*uimcbf*
LSP3, 4-3;	2	*uimcbf*
}		
}		
}		

Таблица 50 — Синтаксис *BaseVar_ESC1* ()

Синтаксис	Количество битов	Мнемоника
BaseVar_ESC1()		
{		
if (VUV == 2 || VUV == 3) {		
SE_shape1, 3-0;	4	*uimcbf*
}		
else if (VUV ==0) {		
LSP2, 2-0;	3	*uimcbf*
LSP3, 2;	1	*uimcbf*
}		
else {		
if (UpdateFlag == 1) {		
LSP2, 2-0;	3	*uimcbf*
LSP3, 2-0;	3	*uimcbf*
}		
}		
}		

Таблица 51 — Синтаксис *BaseVar_ESC2* ()

Синтаксис	Количество битов	Мнемоника
BaseVar_ESC2()		
{		
if (VUV == 2 || VUV == 3) {		
SE_shape2, 3-0;	4	*uimcbf*
}		
else if (VUV == 0) {		
LSP3, 1-0;	2	*uimcbf*
VX_shape1[0], 5-4;	2	*uimcbf*
}		
}		

Таблица 52 — Синтаксис *BaseVar_ESC3* ()

Синтаксис	Количество битов	Мнемоника
BaseVar_ESC3()		
{		
if (VUV == 2 || VUV == 3) {		
LSP2, 4-0;	5	*uimcbf*
LSP3, 3-0;	4	*uimcbf*
Pitch, 0;	1	*uimcbf*
}		

Окончание таблицы 52

Синтаксис	Количество битов	Мнемоника
else if (VUV == 0) { *VX_shape1[0], 3-0;* *VX_shape1[1], 5-0;* *}* *}*	 4 6	 *uimcbf* *uimcbf*

3.4.4 Уровень расширения масштабируемого режима с переменной скоростью передачи данных

Таблица 53 — Синтаксис *ErHVXCenh_varframe* ()

Синтаксис	Количество битов	Мнемоника
ErHVXCenh varframe() *{* *EnhVar_ESC0();* *EnhVar_ESC1();* *EnhVar_ESC2();* *}*		

Таблица 54 — Синтаксис *EnhVar_ESC0* ()

Синтаксис	Количество битов	Мнемоника
EnhVar_ESC0() *{* *if (VUV == 2 || VUV == 3) {* *SE_shape3, 6-2;* *LSP5, 7;* *SE_shape4, 9;* *SE_shape5, 8;* *SE_shape6, 5;* *SE_shape4, 8-6;* *}* *}*	 5 1 1 1 1 3	 *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf* *uimcbf*

Таблица 55 — Синтаксис *EnhVar_ESC1* ()

Синтаксис	Количество битов	Мнемоника
EnhVar_ESC1() *{* *if (VUV == 2 || VUV == 3) {* *SE_shape4, 5-0;* *SE_shape5, 7-0;* *SE_shape6, 4-0;* *}* *}*	 6 8 5	 *uimcbf* *uimcbf* *uimcbf*

Таблица 56 — Синтаксис *EnhVar_ESC2* ()

Синтаксис	Количество битов	Мнемоника
EnhVar_ESC2() *{* *if (VUV == 2 || VUV == 3) {* *LSP5, 6-0;* *SE_shape3, 1-0;* *}* *}*	 7 2	 *uimcbf* *uimcbf*

4 Семантика потока битов

4.1 Конфигурация декодера (*HvxcSpecificConfig, ErrorResilientHvxcSpecificConfig*)

HVXCvarMode: Флажок, указывающий режим варьируемой скорости *HVXC* (таблица 1).
HVXCrateMode: 2-битовое поле, указывающие режим битовой скорости *HVXC* (таблица 1).
extensionFlag: Флажок, указывающий присутствие данных *MPEG-4* версии 2 (таблица 1).
varScalableFlag: Флажок, указывающий варьируемый режим масштабирования *ER_HVXC* (таблица 22).
isBaseLayer: Однобитовый идентификатор, представляющий является ли соответствующий уровень базовым уровнем (1) или уровнем расширения (0).

4.2 Фрейм потока битов (*alPduPayload*)

LSP1: Это поле из 5 битов представляет индекс первой стадии квантования *LSP* (базовый уровень, таблица 8 и таблица 17).
LSP2: Это поле из 7 битов представляет индекс второй стадии квантования *LSP* (базовый уровень, таблица 8 и таблица 17).
LSP3: Это поле из 5 битов представляет индекс второй стадии квантования *LSP* (базовый уровень, таблица 8 и таблица 17).
LSP4: Это однобитовое поле представляет флажок указания, используется ли межкадровое предсказание или не используется во второй стадии квантования *LSP* (базовый уровень, таблица 8 и таблица 1 7).
LSP5: Это поле из 8 битов представляет индекс третьей стадии квантования *LSP* (уровень расширения, таблица 9).
VUV: Это поле из 2 битов представляет решающий режим *V/UV.* У этого поля есть различные значения согласно режиму варьируемой скорости *HVXC* (таблица 10 и таблица 15).
Pitch: Это поле из 7 битов представляет индекс линейно квантованной задержки шага в пределах от 20—147 выборок (отсчетов) (таблица 12 и таблица 18).
SE_shape1: Это 4-битовое поле представляет индекс формы огибающей спектра (базовый уровень, таблица 12 и таблица 18).
SE_shape2: Это 4-битовое поле представляет индекс формы огибающей спектра (базовый уровень, таблица 12 и таблица 18).
SE_gain. Это 5-битовое поле представляет индекс усиления огибающей спектра (базовый уровень, таблица 12 и таблица 18).
VX_shape1 [*sf_num*]: Это 6-битовое поле представляет индекс формы *VXC* подфрейма *sf_num-th* (базовый уровень, таблица 12 и таблица 18).
VX_gain 1 [*sf_num*]: Это 4-битовое поле представляет индекс усиления *VXC* подфрейма *sf_num-th* (базовый уровень, таблица 12 и таблица 18).
SE_shape3: Это поле из 7 битов представляет индекс формы огибающей спектра (уровень расширения, таблица 13).
SE_shape4: Это поле из 10 битов представляет индекс формы огибающей спектра (уровень расширения, таблица 13).
SE_shape5: Это поле из 9 битов представляет индекс формы огибающей спектра (уровень расширения, таблица 13).
SE_shape6: Это поле из 6 битов представляет индекс формы огибающей спектра (уровень расширения, таблица 13).
VX_shape2 [*sf_num*]: Это поле из 5 битов представляет индекс формы *VXC* подфрейма *sf_num-th* (уровень расширения, таблица 13).
VX_gain2 [*sf_num*]: Это поле из 3 битов представляет индекс усиления *VXC* подфрейма *sf_num-th* (уровень расширения, таблица 13).
UpdateFlag: Это 1-битовое поле представляет флажок для указания обновления фрейма шума (только для режима варьируемой скорости *ER_HVXC* на 4 Кбит/с).

5 Инструменты декодера *HVXC*

5.1 Обзор

HVXC обеспечивает эффективную схему для разности Кодирования с линейным предсказанием (*LPC*) на базе гармонического и стохастического векторного представления. Векторное квантование (*VQ*) огибающей спектра остатков *LPC* со взвешенной мерой искажения используется, когда сигнал

является речевым. Кодирование с векторным возбуждением (*VXC*) используется, когда сигнал не является речевым. Главные алгоритмические особенности следующие:

- взвешенный *VQ* спектрального вектора варьируемой размерности;
- алгоритм быстрого гармонического синтеза *IFFT*;
- параметры кодера интерполяции для управления скоростью/шагом.

Кроме того, функциональные особенности включают:

- низкую, до 33,5 мс, полную алгоритмическую задержку;
- масштабируемый режим 2,0—4,0 Кбит/с;
- кодирование с варьируемой битовой скоростью для скоростей меньше 2,0 Кбит/с.

5.1.1 Структура кадрирования и блок-схема декодера

Инструменты декодера *HVXC* позволяют декодировать речевые сигналы на скорости 2,0 Кбит/с и выше, до 4,0 Кбит/с. Инструменты декодера *HVXC* также позволяют декодировать с режимом варьируемой битовой скорости при битовой скорости приблизительно 1,2—1,7 Кбит/с. Основной процесс декодирования состоит из четырех шагов: деквантование параметров, генерация сигналов возбуждения для разговорных фреймов синусоидальным синтезом (гармонический синтез) и добавление шумовой составляющей, генерация сигналов возбуждения для неразговорных фреймов путем просмотра книги шифров и синтез *LPC*. Чтобы повысить качество синтезируемой речи, используют спектральную постфильтрацию.

5.1.2 Режим задержки

Кодер/декодер *HVXC* поддерживает режим низкой/нормальной задержки кодирования/декодирования, позволяя любые комбинации режима задержки при 2,0—4,0 Кбит/с с масштабируемой схемой. Рисунок ниже показывает структуру кадрирования каждого режима задержки. Длина фрейма равна 20 мс для всех режимов задержки. Например, использование режима с низкой задержкой кодирования и низкой задержкой декодирования приводит к полной задержке кодера/декодера 33,5 мс.

В кодере алгоритмическая задержка может быть выбрана равной 26 мс либо 46 мс. Когда выбрана задержка 46 мс, для обнаружения шага используется просмотр одного фрейма вперед. Когда выбрана задержка 26 мс, для обнаружения шага используется только текущий фрейм. Синтаксис для обоих случаев общий, все квантователи общие, и потоки битов совместимы. В декодере алгоритмическая задержка может быть выбрана равной или 10 мс (режим нормальной задержки), или 7,5 мс (режим низкой задержки). Когда выбрана задержка 7,5 мс, интервал фрейма декодера сдвинут на 2,5 мс (20 отсчетов) по сравнению с 10 мс режимами задержки. В этом случае генерация возбуждения и фаза синтеза *LPC* сдвинуты на 2,5 мс. Для обоих случаев синтаксис общий, все квантователи общие и потоки битов совместимы.

Возможен любой независимый выбор задержки кодера/декодера из следующей комбинации:

- задержка кодера: 26 мс или 46 мс;
- задержка декодера: 10 мс или 7,5 мс.

В зависимости от приложения должны поддерживаться одна или несколько комбинаций режима задержки.

5.2 Декодер *LSP*

5.2.1 Описание инструмента

Для квантования параметров *LSP* используется многоступенчатая структура квантователя. Выходные векторы каждой ступени нужно просуммировать, чтобы получить параметры *LSP*.

Когда битовая скорость равна 2,0 Кбит/с, *LSPs* текущего фрейма, которые закодированы разбиением и двухступенчатым векторным квантованием, декодируются, используя двухступенчатый процесс декодирования. При 4,0 Кбит/с к основанию схемы квантователя *LSP* кодера 2,0 Кбит/с добавлен 10-мерный векторный квантователь, у которого имеется книга шифров на 8 битов. Биты, необходимые для *LSPs*, увеличены с 18 битов/20 мс до 26 битов/20 мс.

Таблица 57 — Конфигурация многоступенчатого *LSP VQ*

1-й этап	10 *LSP VQ*	5 битов
2-й этап	(5 + 5) *LSP VQ*	(7 + 5 + 1) битов
3-й этап	10 *LSP VQ*	8 битов

5.2.2 Определения

Определения констант

LPCORDER: Порядок анализа *LPC* (= 10).

dim [][]: Размерности для квантования вектора разбиения.

min_gap: Минимальное расстояние между соседними коэффициентами *LSP* (базовый уровень = = 4,0/256,0).

ratio_ predict: Коэффициент межкадрового предсказания *LSP* (= 0,7).

THRSLD_L: Минимальное расстояние между соседними коэффициентами *LSP* (низкочастотная часть уровня расширения = 0,020).

THRSLD_M: Минимальное расстояние между смежными коэффициентами *LSP* (среднечастотная часть уровня расширения = 0,020).

THRSLD_H: Минимальное расстояние между смежными коэффициентами *LSP* (высокочастотная часть уровня расширения = 0,020).

Определения переменных

qLsp []: Параметры квантованного *LSP*.

*LSP*1: Индекс первой стадии квантования *LSP* (базовый уровень).

*LSP*2, *LSP*3: Индексы второго квантования *LSP* (базовый уровень).

*LSP*4: Флажок, показывающий используется ли межкадровое предсказание (базовый уровень).

*LSP*5: Индекс третьего квантования *LSP* (уровень расширения).

lsp_tbl[][][]: Таблицы поиска для первой стадии процесса декодирования.

d_tbl[][][]: Таблицы поиска для второй стадии процесса декодирования *VQ* без межкадрового предсказания.

pd_tbl[][][]: Таблицы поиска для второй стадии процесса декодирования *VQ* с межкадровым предсказанием.

vqLsp [][]: Таблица поиска для уровня расширения.

sign: Знак вектора кода для второй стадии процесса декодирования.

idx: Индекс распаковки для второй стадии процесса декодирования.

lsp_predict []: *LSPs*, предсказанные из *lsp_previous* и *lsp_first* [].

lsp_previous []: *LSPs*, декодированные в предыдущем фрейме.

lsp_current []: *LSPs*, декодированные в текущем фрейме.

lsp_first []: *LSPs*, декодированные на первой стадии процесса декодирования.

5.2.3 Процесс декодирования

Процесс декодирования параметров *LSP* для базового уровня (2,0 Кбит/с) является тем же самым, что и процесс узкополосного *CELP*. Процесс декодирования описан ниже.

Индексы преобразования к *LSPs*

LSPs текущего фрейма (*lsp_current* []), которые закодированы разбиением и двухступенчатым векторным квантованием, декодируются двухступенчатым процессом декодирования. Размерность каждого вектора приведена в таблицах ниже. *LSP*1 и *LSP*2, *LSP*3 представляют индексы для первой и второй стадии соответственно.

Таблица 58 — Размерность вектора *LSP* первой стадии

Индекс вектора разбиения. *i*	Размерность вектора: *dim* [0] [*i*]
0	10

Таблица 59 — Размерность вектора *LSP* второй стадии

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

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

```
for (i = 0; i < dim[0] [0]; i++) {
  lsp_first[i] = lsp_tbl[0] [LSP1][i];
}
```

Во второй стадии есть два типа процессов декодирования, а именно, процесс декодирования *VQ* без межкадрового предсказания и *VQ* с межкадровым предсказанием. Флажок *LSP*4 указывает, какой процесс должен быть выбран.

Таблица 60 — Процесс декодирования для второй стадии

Индекс *LSP*: *LSP*4	Процесс декодирования
0	Без межкадрового предсказания *VQ*
1	С межкадровым предсказанием *VQ*

Процесс декодирования *VQ* без межкадрового предсказания

Чтобы получить *LSPs* текущего фрейма *lsp_current* [], декодированные векторы во второй стадии добавляются к декодированному в первой стадии вектору *LSP lsp_first* []. MCB для *LSP*2 и *LSP*3 представляет знак декодированного вектора, а остающиеся биты представляют индекс для таблицы *d_tbl*[][][].

```
sign = LSP2>>6;
idx = LSP2&0x3f;
if (sign == 0) {
  for (i = 0; i < dim[ 1 ][0]; i++) {
    lsp_current[i] = lsp_first[i] + d_tbl[0][idx][i];
  }
}
else {
  for (i = 0; i < dim[ 1 ][0]; i++) {
    lsp_current[i] = lsp_first[i] - d_tbl[0][idx][i];
  }
}
sign = LSP3>>4;
idx = LSP3&0x0f;
if (sign == 0) {
  for (i = 0; i < dim[1][1]; i++) {
    lsp_current[dim[1][0]+i] = lsp_first[dim[1][0]+i] + d_tbl[1][idx][i];
  }
}
else {
  for (i = 0; i < dim[1][1]; i++) {
    lsp_current[dim[ 1 ][0]+i] = lsp_first[dim[1][0]+i] - d_tbl[1][idx][i];
  }
}
```

Процесс декодирования *VQ* с межкадровым предсказанием

Чтобы получить *LSPs* текущего фрейма *lsp_current* [], декодированные векторы второй стадии добавляются к вектору *LSP lsp_predict* [], которые предсказаны из декодированного *LSPs* предыдущего фрейма *lsp_previous* [] и декодированного в первой стадии вектора *LSP lsp_first* []. Как в процессе декодирования *VQ* без межкадрового предсказания, *MCB* для *LSP*2 и *LSP*3 представляет знак декодированного вектора, а остающиеся биты представляют индекс для таблицы *pd_tbl*[][][].

```
for (i = 0; i < LPCORDER; i++) {
  lsp_predict[i] = (1-ratio_predict)*lsp_first[i]
                 + ratio_predict*lsp_previous[i].
}
sign = LSP2>>6;
idx = LSP2&0x3f;
if (sign == 0) {
  for (i = 0; i < dim[ 1 ][0]; i++) {
    lsp_current[i] = lsp_predict[i] + pd_tbl[0][idx][i].
  }
}
```

```
else {
  for (i = 0; i < dim[ 1 ][0]; i++) {
    lsp_current[i] = lsp_predict[i] - pd_tbl[0][idx][i];
  }
}
sign = LSP3>>4;
idx = LSP3&0x0f;
if (sign == 0) {
 for (i = 0; i < dim[1][1]; i++) {
   lsp_current[dim[1][0]+i] = lsp_predict[dim[1][0]+i] + pd_tbl[1][idx][i];
}
}
else {
  for (i = 0; i < dim[1][1]; i++) {
    lsp_current[dim[1][0]+i] = lsp_predict[dim[1][0]+i] - pd_tbl[1][idx][i];
 }
}
```

Стабилизация *LSPs*

Декодированные *LSPs lsp_current* [] стабилизированы, чтобы гарантировать стабильность фильтра синтеза *LPC*, который извлекается из декодированных *LSPs*. Декодированные *LSPs* упорядочены в порядке возрастания, имея минимум расстояния между соседними коэффициентами.

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

Сохранение коэффициентов

После процесса декодирования *LSP* декодированные *LSPs* должны быть сохранены в памяти, так как они используются для предсказания в следующем фрейме.

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

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

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

Процесс декодирования для уровня расширения

Для уровня расширения (4,0 и 3,7 Кбит/с), дополнительные векторы кода и *LSPs* базового уровня суммируются следующим образом.

```
for (i = 0; i < LPCORDER; i++) {
qLsp[i] += vqLsp[LSP5][i];
}
```

После вычисления *LSPs* снова стабилизируются.

```
for (i = 0; i < 2; i++) {
  if (qLsp[i+1] - qLsp[i] < 0) {
    tmp = qLsp[i+1];
    qLsp[i+1] = qLsp[i],
    qLsp[i] = tmp;
  }
  if (qLsp[i+1] - qLsp[i] < THRSLD L) {
    qLsp[i + 1] = qLsp[i] + THRSLDL;
  }
}
for (i = 2; i < 6; i++) {
  if (qLsp[i+1] - qLsp[i] < THRSLD_M) {
    tmp = (qLsp[i+1] + qLsp[i])/2.0;
    qLsp[i+1] = tmp + THRSLDM/2.0;
    qLsp[i] = tmp - THRSLD_M/2.0;
    }
  }
  for (i = 6; i < LPCORDER-1; i++) {
    if (qLsp[i+1] - qLsp[i] < 0) {
    tmp = qLsp[i+1];
    qLsp[i+1] = qLsp[i];
    qLsp[i] = tmp;
  }
  if (qLsp[i+1] - qLsp[i] < THRSLD_H){
    qLsp[i] = qLsp[i+1] - THRSLD_H;
   }
}
```

5.3 Декодер гармонического *VQ*

5.3.1 Описание инструмента

Процесс декодирования состоит из двух главных шагов для базового уровня, а именно, инверсного векторного квантования векторов огибающей спектра и преобразования размерности. Для уровня расширения используются дополнительные инверсные квантователи.

5.3.2 Определения Определения констант

SAMPLE: Число выборок в частотном спектре в пределах от 0 до 2π (= 256).

R: Превышение частоты выборок в преобразовании размерности (= 8).

*vqdim*0: Размерность вектора квантования огибающей спектра (= 44).

JISU: Порядок фильтра передискретизации в домене с квантованием (= 9).

f coef []: Коэффициенты фильтра передискретизации.

Определения переменных

pitch: Индекс величины отставания линейно квантованного шага.

pch: Значение отставания шага в текущем фрейме.

pch_mode: Фактор модификации шага.

w0f: Исходная фундаментальная частота, где *SAMPLE* представляет 2π.

send: Число гармоник в текущем фрейме (между 0 и 3800 Гц).

w0: Целевая фундаментальная частота после преобразования размерности, где *SAMPLE* × *R* представляет 2π.

HVXCrate: Операционная битовая скорость декодера.

qedvec []: Вектор квантованной огибающей спектра в фиксированной размерности.

SE_gain: Индекс усиления огибающей спектра (базовый уровень).

SE_shape1, *SE_shape2*: *SE_shape3*, *SE_shape4*, *SE_shape5*, Индексы формы огибающей спектра (базовый уровень).

SE_shape6: Индексы формы огибающей спектра (уровень расширения).
g0 []: Кодовое слово *SE_gain*.
cb0 [] []: Кодовое слово *SE_shape1*.
cb1 [] []: Кодовое слово *SE_shape2*.
cb4k[0][][]: Кодовое слово *SE_shape3*.
cb4k[1][][]: Кодовое слово *SE_shape4*.
cb4k[2][][]: Кодовое слово *SE_shape5*.
cb4k[3][][]: Кодовое слово *SE_shape6*.
re []: Ввод преобразования размерности.
rel0, *rel1*: Значения 8-кратного превышения дискретности в преобразовании размерности.
ip_ratio: Коэффициент линейной интерполяции преобразования размерности.
target []: Восстановленный вектор, обусловленный векторными квантователями для уровня расширения.
arm []: Восстановленный вектор гармонических величин.
feneq: Среднеквадратичное значение вектора квантованной огибающей спектра в текущем фрейме.
feneqold: Среднеквадратичное значение вектора квантованной огибающей спектра в предыдущем фрейме.

5.3.3 Процесс декодирования

Декодирование индекса шага

Значение задержки шага в текущем фрейме, *pch*, декодировано из индекса шага *Pitch* следующим образом

$$pch = Pitch + 20{,}0.$$

Модификация шага может быть выполнена делением *pch* на фактор модификации шага *pch_mod*

$$pch = pch / pch\_mod.$$

Если модификацией шага управляет поле *pitch* в узле *AudioSource BIFS*, коэффициент модификации равен

$$pch\_mod = pitch.$$

Значение модулированной задержки шага должно быть в пределах диапазона от 8,0 до 147,0.

Тогда число гармоник в частотном диапазоне между 0 и 3800 Гц, *send,* и фундаментальная частота, *w*0 (где *SAMPLE* × *R* представляет 2π), вычисляются следующим образом:

$$send = (\text{int})(0{,}95 \times pch \times 0{,}5),$$

$$w0 = \frac{SAMPLE \times R}{pch}.$$

Декодирование гармонических величин

Декодирование гармонических величин состоит из следующих шагов:

(*S*1) Инверсия квантования вектора базового уровня;

(*S*2) Подавление малых сигналов;

(*S*3) Преобразование размерности выхода базового уровня;

(*S*4) Инверсия квантования вектора уровня расширения.

Для режима 2,0 Кбит/с выполняются вышеприведенные *S*1, *S*2 и *S*3 для того, чтобы получить гармонические величины. Для режимов 4,0 и 3,7 Кбит/с в дополнение к *S*1, *S*2 и *S*3 выполняется *S*4.

В режиме 2,0 Кбит/с используется комбинация двухступенчатого квантования вектора формы и *скалярного квантования усиления, чьими индексами являются SE_shape1*, *SE_shape2* и *SE_gain* соответственно. Размерность двух книг шифров формы фиксирована (= 44). В *S*1 добавляются два вектора формы, представленные *SE_shape1* и *SE_shape2*, и затем умножаются на усиление, представленное *SE_gain*. Вектор огибающей спектра, полученный в *S*1, охватывает частотный диапазон от 0 до 3800 Гц. Вектор огибающей спектра очень малой энергии затем подавляется в *S*2. Чтобы получить

вектор гармонических величин исходной размерности, *send*, затем в *S*3 к вектору огибающей спектра применяется преобразование размерности. В режиме 4,0 Кбит/с используется дополнительная стадия с разбиением схемы *VQ*, составленной из четырех векторных квантователей для уровня расширения. *SE_shape3*, *SE_shape4*, *SE_shape5* и *SE_shape6* представляют индексы квантователей для уровня расширения. В *S*4 вывод этих квантователей добавляется к выводу *S*3 для гармонических величин в самых нижних 14 частотных гармониках. Когда выбран режим 3,7 Кбит/с, *SE_shape6* недоступно и *S*4 выполняется для гармонических величин только самых низких 10 частотных гармоник.

Таблица 61 — Конфигурация многоступенчатого гармонического *VQ*

Размерность двухэтапного *VQ* на 2,0 Кбит/с	4-битовая форма + 4-битовая форма + 5-битовое усиление 44			
Размерность расщепленного *VQ* на 4,0 Кбит/с	7 битов 2	10 битов 4	9 битов 4	6 битов 4

Конверсионный алгоритм размерности

Теоретический фон конверсионного алгоритма размерности, используемый в этом инструменте, объяснен ниже.

Число точек, которые составляют огибающую спектра, изменяется в зависимости от значения шага, поскольку огибающая спектра является рядом оценок величин в каждой гармонике. Число гармоник колеблется от 9 до 70. Чтобы получить величины варьируемого количества гармоник, декодер должен преобразовать кодовый вектор фиксированной размерности (= 44) в вектор варьируемой размерности. Число точек, которые представляют форму огибающей спектра, должно быть изменено без изменения формы. С этой целью используется конвертер размерности, состоящий из комбинации фильтра нижних частот и линейного интерполятора 1-го порядка. Фильтр нижних частот *FIR* с 7 наборами коэффициентов. Каждый набор, состоящий из 8 коэффициентов, используется для 8-разовой передискретизации первой стадии. 7 наборов коэффициентов фильтра получены, группируя каждые 8 коэффициентов от реализуемого методом окна *sinc f_coef* [i], со смещениями 1—7, где

$$f\_coef[i] = \frac{\sin\pi(i-32)/8}{\pi(i-32)/8}(0{,}5-0{,}5\cos 2\pi i/64) \quad 0 \le i \le 64.$$

Фильтрация *FIR* позволяет вдесятеро уменьшить вычисления, в которых вычисляются только точки, используемые в следующей стадии. Они представляют собой левые и правые смежные точки конечного вывода конвертера размерности.

Во второй стадии передискретизации применяется линейная интерполяция первого порядка, чтобы получить необходимые точки вывода. Таким образом, гармонические векторы величин переменной размерности получаются из векторов огибающей спектра фиксированной размерности (= 44).

(*S*1) Квантование вектора инверсии базового уровня:

```
qedvec[0] = 0.0f;
for (i = 0; i < vqdim0; i++)
  qedvec[i+1] =
g0[SE_gain]*(cb0[SE_shape1 ][i]+cb1[SE_shape2][i]);
```

(*S*2) Подавление малых сигналов:

```
feneq = 0.0f;
for (i = 0; i < vqdim0; i++)
  feneq += qedvec[i+1]*qedvec[i+1],
feneq = sqrt(feneq/(float)vqdim0);
if (feneq < 1,0f ||
0,5f*(feneqold+feneq) < 1.4f) {
  for (i = 0; i < vqdim0; i++)
qedvec[i+1] = 0.0f;
}
feneqold = feneq;
```

(*S*3) Преобразование размерности вывода базового уровня:

```
for (i = 0; i < (JISU-1)/2; i++)
```

```
    re[i] = 0.0f;
  for (i = 0; i <= vqdim0; i++)
    re[i+(JISU-1)/2] = qedvec[i];
  for (i = 0; i < (JISU-1)/2; i++)
    re[i+vqdim0+1+(JISU-1)/2] = qedvec[vqdim0],
  w0f = (float)(SAMPLE*0,5*0,95)/(float)vqdim0;
  ii = 0;
  for (i = 0; i <= vqdim0 && ii <= send; i++) {
    for (p = 0; p < R && ii <= send; p++) {
      ipratio = (i*R+p+1)*w0f-w0*ii;
      if (ip_ratio > 0) {
      ip_ratio/= w0f;
      rel0 = rell = 0.0f;
      for (j = 1; j < JISU; j++) {
        rel0 += f_coef[j*R-p]*re[i+j];
        rel1 += f_coef[j*R-(p+1)]*re[i+j];
      }
    am[ii] = rel0*ip_ratio + rel1*(1,0f-ip_ratio);
    ii++;
  }
}
```

(*S*4) Квантование вектора инверсии уровня расширения:

```
  target[0] = 0;
  k = 1;
  for (i = 0; i < 2; i++, k++)
    target[k] = cb4k[0][SE_shape3][i];
  for (i = 0; i < 4; i++, k++)
    target[k] = cb4k[1][SE_shape4][i];
  for (i = 0; i < 4; i++, k++)
    target[k] = cb4k[2][SE_shape5][i];
  if (HVXCrate >= 4000){
    for (i = 0; i < 4; i++, k++)
      target[k] = cb4k[3][SE_shape6][i];
  }
  else{
    for (i = 0; i < 4; i++, k++)
      target[k] = 0.0f;
  }
  if (send > 14) {
    for (i = 15; i <= send; i++)
      target[i] = 0.0f;
  }
  for (i = 0; i <= send; i++)
    am[i] += target[i]
```

5.4 Декодер временного домена

5.4.1 Описание инструмента

Для неречевых сегментов в составе речи используется схема, которая подобна *VXC* (Кодирование с векторным возбуждением). Декодер временного домена генерирует форму волны возбуждения для неречевой части, просматривая таблицы и используя переданные индексы. Вектор формы и усиление базового уровня обновляются каждые 10 мс. Форма масштабируется путем умножения каждого отсчета на величину усиления. В режиме 2,0 Кбит/с используется только вывод первой стадии (базовый уровень). В режимах 4,0 Кбит/с и 3,7 Кбит/с умножаются вектор формы и усиление второй стадии (уровень расширения) и добавляются к выводу первой стадии. Форма и усиление уровня расширения обновляются каждые 5 мс.

Таблица 62 — Конфигурация временного домена *VQ*

1-й этап	(80 мерная 6-битовая форма + 4-битовое усиление) × 2
2-й этап	(40 мерная 5-битовая форма + 3-битовое усиление) × 4

5.4.2 Определения

Определения констант

DiMchape: Длина фрейма *VXC* первой стадии (= 80).

DiMchape2: Длина фрейма *VXC* второй стадии (= 40).

Определения переменных

HVXCrate: Операционная битовая скорость декодера.

res[*i*]: Вывод декодера *VXC* ($0 \le i \le FRM$).

cbL0_g[*i*]: *i*-й вход книги шифров усиления *VXC* первой стадии.

cbL0_s[*i*][*j*]: *j*-й компонент *i*-го входа книги шифров формы *VXC* первой стадии.

cbL1_g[*i*]: *i*-й вход книги шифров усиления *VXC* второй стадии.

cbL1_s[*i*][*j*]: *j*-й компонент *i*-го входа книги шифров формы *VXC* второй стадии.

VX_gain1[*i*]: Индекс усиления *VXC i*-го подфрейма (базовый уровень, $i = 0, 1$).

VX_shape1[*i*]: Индекс формы *VXC i*-го подфрейма (базовый уровень, $i = 0, 1$).

VX_gain2[*i*]: Индекс усиления *VXC i*-го подфрейма (уровень расширения, $i = 0, 1, 2, 3$).

VX_shape2[*i*]: Индекс формы *VXC i*-го подфрейма (уровень расширения, $i = 0, 1, 2, 3$).

5.4.3 Процесс декодирования

Для базового уровня:

```
for (i = 0; i < DimShape; i++)
  res[i] = cbL0_g[VX_gain[0]] * cbL0_s[VX_shape[0]][i];
for (i = 0; i < DimShape; i++)
  res[i + DimShape] = cbL0_g[VX_gain[1]] *cbL0_s[VX_shape[1]][i];
```

Добавить уровень расширения:

```
 if (HVXCrate >= 4000){ /* 4.0 kbps mode */
   for (i = 0; i < 4; i++){
     for (j = 0; j < DimShape2; j++){
       res[j+DimShape2*i] +=
cbL1_g[VX_gain2[i]]*cbL1_s[VX_shape2[i]][j];
     }
   }
 }
   else{                  /* 3.7kbps mode */
     for (i = 0; i < 3; i++){
       for (j = 0; j < DimShape2; j++){
         res[j+DimShape2*i] +=
cbL1_g[VX_gain2[i]]*cbL1_s[VX_shape2[i]][j];
       }
     }
   }
```

5.5 Интерполяция параметров для управления скоростью

5.5.1 Описание инструмента

У декодера имеется схема интерполяции параметров, чтобы генерировать ввод для «Синтезатора Речевой Компоненты» и «Синтезатора Неречевой Компоненты» в любой произвольный момент времени. С помощью этой схемы вычисляется последовательность параметров в измененных интервалах и применяется в обоих синтезаторах. Таким образом получается вывод декодера в измененной временной шкале.

5.5.2 Определения

Определения констант:

FRM: Интервал фрейма (= 160).

p: Порядок *LPC* (= 10).

Определения переменных

Блок "*Parameter Interpolation*" (Интерполяция параметров) вычисляет параметры в измененной временной шкале, интерполируя полученные параметры.

Работа этого блока состоит в основном из линейной интерполяции и замены параметров.

Обозначим массивы оригинальных параметров как:

pch [*n*]: Величина отставания шага в момент времени *n*.

vuv [*n*]: Индекс *V/UV* в момент времени с индексом *n*.

lsp [*n*] [*i*]: Декодированные *LSPs* при временном индексе *n* ($0 \le i < P$).

send [*n*]: Число гармоник при временном индексе *n*.

am [*n*] [*i*]: Величины гармоник при временном индексе *n* ($0 \le i <$ send [*n*]).

vex [*n*] [*i*]: Декодированный сигнал возбуждения *VXC* при временном индексе *n* ($0 \le i <$ *FRM)*.

param [*n*]: Параметр при временном индексе *n*.

и интерполированные параметры как:

mdf_pch [*m*]: Величина задержки шага при временном индексе *m*.

mdf_vuv [*m*]: Индекс *V/UV* при временном индексе *m*.

mdf_lsp [*m*] [*i*]: Декодированные *LSPs* при временном индексе *m* ($0 \le i < 10$).

mdf_send [*m*]: Число гармоник при временном индексе *m*.

mdf_am [*m*] [*i*]: Величины гармоник при временном индексе *m* ($0 \le i <$ *send* [*n*]).

mdf_vex [*m*] [*i*]: Декодированный сигнал возбуждения *VXC* при временном индексе *m* ($0 \le i <$ *FRM)*.

mdf_param [*m*]: Параметр при временном индексе *m*,

где *n* и *m* являются временными индексами (номер фрейма) до и после модификации временной шкалы. Интервалы фрейма в обоих случаях равны 20 мс.

spd: Отношение изменения скорости ($0{,}5 \le spd \le 2{,}0$).

N_1: Продолжительность речевого оригинала (полное число фреймов).

N_2: Продолжительность речи с управлением скоростью (полное число фреймов).

fr_0, fr_1: Индексы фрейма, смежного с точкой интерполяции.

left, right: Коэффициент интерполяции.

tmp_vuv: Временной индекс *V/UV*.

5.5.3 Процесс управления скоростью

Определим коэффициент изменения скорости как *spd*

$$spd = N_1 / N_2, \quad (1)$$

где N_1 является продолжительностью речевого оригинала и N_2 является продолжительностью речи с управляемой скоростью. Поэтому

$$0 \le n < N_1 \text{ и } 0 \le m < N_2.$$

Если скоростью управляют соответственно фактору временного масштабирования в поле *speed* узла *AudioSource BIFS*, отношение изменения скорости равно

$$spd = 1 / speed.$$

В основном измененные параметры временной шкалы выражены как

$$mdf\_param\ [m] = param\ [m \times spd], \quad (2)$$

где *param* являются: *pch, vuv, lsp* и *am*. Однако, $m \times spd$ не является целым числом.

Поэтому определяем:

$$fr_0 = m \times spd, \quad (3)$$

$$fr_1 = fr_0 + 1,$$

чтобы генерировать параметры при временном индексе $m \times spd$ линейной интерполяцией параметров при временных индексах fr_0 и fr_1.

Чтобы выполнить линейную интерполяцию, определим:

$$left = m \times spd - fr_0, \tag{4}$$

$$right = fr_1 - m \times spd.$$

Тогда уравнение (2) может быть аппроксимировано как

$$mdf\_param\ [m] = param\ [fr_0]\ fr_0 + param\ [fr_1] \times left, \tag{5}$$

где *param* являются: *pch*, *vuv*, *lsp* и *am*.

Для *lsp* [*n*] [*i*] и *am*[*n*] [*i*] эта линейная интерполяция применяется с фиксируемым индексом *i*.

Параметр *vex*

vex [*n*] [*i*] имеет сигналы возбуждения для фреймов *UV* в результате просмотра книги шифров.

Берутся отсчеты *FRM* из *vex* [*n*] [*i*], центрированные вокруг времени *m* × *spd*, и вычисляется энергия по отсчетам *FRM*. Затем генерируется состоящий из отсчетов *FRM* гауссовский шум, и его норма корректируется так, чтобы его энергия была равна энергии из отсчетов *FRM*, взятых из *vex* [*n*] [*i*]. Эта последовательность гауссовского шума с регулируемым усилением используется для *mdf_vex* [*m*] [*i*].

Главная операция изменения временной шкалы может быть выражена уравнением (5), однако до интерполяции нужно рассмотреть решения *V/UV* при fr_0 и fr_1.

Стратегии интерполяции и замены, адаптированные к решениям *V/UV*, описаны ниже. В объяснении полностью речевой и смешанный речевой (*vuv* [*n*] ≠ 0) сгруппированы как "*Voiced*" (речевой), и только случай *vuv*[*n*] = 0 расценивается как "*Unvoiced*" (неречевой). В случае варьируемой скорости кодирования режим "*Background Noise*" (фоновый шум) (*vuv* [*n*] = 1) также рассматривается как "*Unvoiced*".

Когда решения *V/UV* при fr_0 и fr_1 являются *Voiced — Voiced*, новый индекс *V/UV mdf_vuv* [*m*] получают следующим образом:

```
tmp_vuv = vuv [fr0] × right + vuv [fr1] × left
if (tmp_vuv>2)
    mdf_vuv [m] = 3
else if (tmp_vuv>1)
    mdf_vuv [m] = 2
else if (tmp_vuv>0)
    mdf_vuv [m] = 1
```

Новая величина задержки шага *mdf_pch* [*m*] получается следующим образом:

```
if (0,57 < pch [fr0] / pch [fr1] && pch [fr0] / pch [fr1] < 1,75)
   mdf_pch [m] = pch [fr0] × right + pch [fr1] × left
else
   if (left < right)
      mdf_pch [m] = pch [fr0]
   else
      mdf_pch [m] = pch [fr1]
```

Все другие параметры интерполированы по уравнению (5).

Когда решения *V/UV* при fr_0 и fr_1 являются *Unvoiced — Unvoiced*, все параметры интерполируются по уравнению (5), исключая *mdf_vex*. *mdf_vex* [*m*] [*i*] генерируется по гауссовскому шуму, имеющему ту же энергию, что и энергия выборок *FRM*, взятых из *vex* [*n*] [*i*], центрированных вокруг времени *m* × *spd*.

Когда решения *V/UV* при fr_0 и fr_1 являются *Voiced — Unvoiced*,

If left < right.

Вместо того чтобы вычислять параметры при *m* × *spd*, используются все параметры в момент времени fr_0.

If left ≥ right

Вместо того чтобы вычислять параметры при *m* × *spd*, используются все параметры в момент времени fr_1.

mdf_vex [*m*] [*i*] также сгенерирован гауссовским шумом, имеющим ту же самую энергию, что и выборки *FRM* из *vex* [fr_1] [*i*]. (0 ≤ *i* < *FRM*)

Когда решения *V/UV* при fr_0 и fr_1 являются *Unvoiced — Voiced*,

If left < right.

Все параметры во время fr_0 используются вместо того, чтобы вычислять параметры при $m \times spd$.

mdf_vex [m] [*i*] также сгенерирован гауссовским шумом, имеющим ту же самую энергию, как энергия выборок *FRM* из *vex* [fr_0] [*i*]. ($0 \le i < FRM$)

If left < right.

Все параметры во время fr_1 используются вместо того, чтобы вычислять параметры при $m \times spd$.

Этим способом получают все необходимые параметры для декодера *HVXC*. Только применяя эти измененные параметры, *mdf_param* [*m*], к "Синтезатору Речевой Компоненты" и "Синтезатору Неречевой Компоненты" тем же способом, как обычный (нормальный) процесс декодирования, получают вывод с измененной временной шкалой.

Очевидно, когда $N_2 < N_1$, выполняется ускоренное декодирование, а когда $N_2 > N_1$, выполняется декодирование со сниженной скоростью. На спектр мощности и шаг это управление скоростью не влияет, таким образом, мы можем получить хорошее качество речи для коэффициента управления скоростью — приблизительно $0,5 < spd < 2,0$.

5.6 Синтезатор речевой компоненты

5.6.1 Описание инструмента

Синтезатор речевой компоненты состоит из следующих этапов:

- модификация величин гармоник,
- синтез возбуждения гармоник,
- добавление шумовой составляющей,
- синтез *LPC*,
- постфильтр.

Эффективный метод синтеза возбуждения гармоник прежде всего используется для того, чтобы получить периодическую волну возбуждения из величин гармоник. Добавляя шумовую компоненту к периодической волне, получают сигнал речевого возбуждения, который затем подается в фильтр синтеза *LPC* и постфильтр, чтобы генерировать речевой сигнал. Конфигурация постфильтра не нормативна и описана в приложении Б.

5.6.2 Определения

Определения констант

PI: $\pi = 3,14159265358979...$

FRM: Интервал фрейма (= 160).

SAMPLE: Длина фрейма анализа (= 256).

WAVE_LEN: Длина волны одного периода шага (= 128).

WDEVI: Порог отношения фундаментального изменения частоты (= 0,1).

LD LEN: Сдвиг интервала декодирования для режима низкой задержки (= 20).

RND_MAX: Максимальное число, сгенерированное генератором случайных чисел (= 0 × 7fffffff).

SCALEFAC: Масштабный коэффициент для измененных величин гармоник для генерации шумовой составляющей (= 10).

c_dis[*i*]: Окно трапецоида для гармонического синтеза шага прерывистая волна показано на рисунке 1 ($0 \le i < FRM + LD\_LEN$).

ham [i]: Окно Хэмминга ($0 \le i < SAMPLE$).

ham_z [*i*]: Дополненное нулями окно Хэмминга ($0 \le i < 2 \times$ FRM).

HAMLD: Длина окна Хэмминга для режима низкой задержки (= 240).

P: Порядок *LPC* (= 10).

Определения переменных

am2 [*i*]: Гармонические величины в границе окончания интервала декодирования. Они получены как *am* [*i*] в 5.3.3.

am_h [*i*]: Измененные величины гармоник на конечной границе интервала декодирования.

am_noise [*i*]: Измененные величины гармоник для генерации компоненты шума на конечной границе интервала декодирования.

vuv2: Индекс *V/UV* на конечной границе интервала декодирования. Получен как *VUV* из потока битов.

vuv1: Индекс *V/UV* на границе начала интервала декодирования.

vuv0: Индекс *V/UV* на границе начала предыдущего интервала декодирования.

*pch*2: Значение задержки шага [отсчет] на конечной границе интервала декодирования. Получено как *pch* (значение задержки шага в текущем фрейме) в 5.3.3.

*pch*1: Значение задержки шага [отсчет] на начальной границе интервала декодирования.

*send*2: Число гармоник на конечной границе интервала декодирования.

*send*1: Число гармоник на начальной границе интервала декодирования.

*w*02: Фундаментальная частота на конечной границе интервала декодирования [*rad/sample*].

*w*01: Фундаментальная частота на начальной границе интервала декодирования [*rad/sample*].

*pha*2 [*l*]: Значения фазы гармоник на конечной границе интервала декодирования.

*pha*1 [*l*]: Значения фазы гармоник на начальной границе интервала декодирования.

*ovsr*2: Коэффициент передискретизации на конечной границе интервала декодирования.

*ovsr*1: Коэффициент передискретизации на начальной границе интервала декодирования.

ovsrc: Линейно интерполированный коэффициент передискретизации.

*wave*2 [*l*]: Форма волны периода в один шаг, сгенерированная из шага и величин гармоник на конечной границе интервала декодирования ($0 \le i <$ *WAVE_LEN*).

*wave*1 [*l*]: Форма волны периода в один шаг, сгенерированная из шага и величин гармоник на начальной границе интервала декодирования ($0 \le i <$ *WAVE_LEN*).

*lp*12, *lp*12*r*: Длина передискретизированной формы волны, необходимая чтобы восстановить форму волны интервала декодирования (в случае непрерывного перехода шага).

*lp*1, *lp2, lp2r*: Длина передискретизированной формы волны, необходимая чтобы восстановить форму волны интервала декодирования (в случае прерывающегося перехода шага).

st: Смещение для циклического расширения *wave*2 [*l*] на начальной границе интервала декодирования.

*iflat*1, *iflat*2: Длина периода, где интерполяция параметров не производится в домене перекодирования (для режима малой задержки).

*out*2 []: Циклически расширенная форма волны, генерированная из *wave*2 [*l*].

*out*1 []: Циклически расширенная форма волны, генерированная из *wave*1 [*l*].

*out*3 []: Взвешенная перекрытая и добавленная циклически расширенная форма волны в интервале декодирования, сгенерированная из *out*1 [] и *out*2 [] (в случае непрерывного перехода шага).

*sv*2 [*l*]: Повторно квантованная форма волны из *out*2 [] (в случае прерывающегося перехода шага, $0 \le i <$ *FRM*).

*sv*1 [*l*]: Повторно квантованная форма волны из *out*1 [] (в случае прерывающегося перехода шага, $0 \le i <$ *FRM*).

sv [*l*]: Сгенерированный сигнал речевого возбуждения ($0 \le i <$ *FRM*).

ns [*l*]: Гауссовский шум с нулевым средним и единичной дисперсией ($0 \le i <$ *SAMPLE*).

wns [*l*]: Взвешенный функцией Хемминга Гауссовский шум ($0 \le i <$ *SAMPLE*).

rmc [*l*]: Массив амплитуд спектра ($0 \le i <$ *SAMPLE* / 2).

ang [*l*]: Массив фаз спектра ($0 \le i <$ *SAMPLE* / 2).

re [*l*]: Вещественная часть коэффициентов FFT ($0 \le i <$ *SAMPLE*).

im [*l*]: Мнимая часть коэффициентов FFT ($0 \le i <$ *SAMPLE*).

*w*0: Фундаментальная частота текущего фрейма, где 2π выражен как *SAMPLE* (= 256).

cns [*l*]: Результат *IFFT* текущего фрейма для генерации шумовой компоненты ($0 \le i <$ *SAMPLE*).

cns_z [*l*]: Заполненный нулями массив для *cns*[] ($0 \le i < 2 \times$ *FRM*).

cns_z_p [*l*]: *cns_z* [] предыдущего фрейма для перекрытия и добавления ($0 \le i < 2 \times$ *FRM*).

add_uv [*i*]: Сгенерированная шумовая компонента в интервале декодирования ($0 \le i <$ *FRM*).

*lsp*2 []: Деквантованные *LSPs* текущего фрейма, полученные как *qLsp*[] в 5.2.3.

*lsp*1 []: Деквантованные *LSPs* предыдущего фрейма.

lspip [] []: Интерполированные *LSPs*.

alpaip [] []: Коэффициенты линейного прогнозирования, преобразованные из интерполированных *LSPs lspip* [] [].

Определения функций

random(): Генератор случайных чисел, который возвращает случайные числа в пределах от 0 до *RND_MAX*,

ceil(*x*): Функция, которая возвращает наименьшее целое число, большее чем или равное *x*.

floor(*x*): Функция, которая возвращает наибольшое целое число, меньшее чем или равное *x*.

5.6.3 Процесс синтеза

Алгоритм синтеза может применяться как для режима нормальной задержки, так и для режима малой задержки. Для режима малой задержки используется смещение интервала декодирования *LD_LEN*. Синтезированная форма волны охватывает от *N* = – 160 + *LD_LEN* [выборка] до *N* = 0 + + *LD_LEN* [выборка]. *N* = 0 представляет собой центр текущего фрейма. Если сдвиг фрейма равен 0 (*LD_LEN* = 0), синтезируемая форма волны идентична форме режима нормальной задержки. Это показано на рисунке ниже.

Во время периода от *N* = 0 до *N* = *LD_LEN* задержка шага, величины гармоник и параметры *LSP* не интерполированы и сохраняются. Если *LD_LEN* = 0, задержка декодера равна 10 мс, а если *LD_LEN* = 20, задержка декодера равна 7,5 мс.

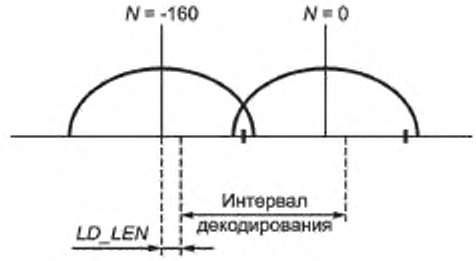


Рисунок 1 — Интервал декодирования

5.6.3.1 Модификация величин гармоник

Гармонические величины *am2*[*i*] изменены для гармонического синтеза возбуждения и шумового составляющего поколения независимо, согласно индексу *V/UV* текущего фрейма *vuv2*.

Измененные величины гармоник *am_h*[*i*] для синтеза возбуждения гармоник и *am_noise*[*i*] для генерации шумой компоненты получают, как описано ниже.

Когда индекс *V/UV*, *vuv2*, 0, ничего не делается (работает только декодер *VXC*).

Когда индекс *V/UV, vuv2*, равен 1:

$$am\_h[i] = \begin{cases} am2[i] & (1 \le i < send2 \times B\_TH1) \\ am2[i] \times AH1 & (send2 \times B\_TH1 \le i \le send2) \end{cases}$$

$$am\_noise[i] = \begin{cases} 0 & (1 \le i < send2 \times B\_TH1) \\ SCALEFAC \times am2[i] \times AN1 & (send2 \times B\_TH1 \le i \le send2) \end{cases}$$

Когда индекс *V/UV, vuv2*, равен 2:

$$am\_h[i] = \begin{cases} am2[i] & (1 \le i < send2 \times B\_TH2) \\ am2[i] \times AH2 & (send2 \times B\_TH2 \le i < send2 \times B\_TH2\_2) \\ am2[i] \times AH2\_2 & (send2 \times B\_TH2\_2 \le i \le send2) \end{cases}$$

$$am\_noise[i] = \begin{cases} 0 & (1 \le i < send2 \times B\_TH2) \\ SCALEFAC \times am2[i] \times AN2 & (send2 \times B\_TH2 \le i < send2 \times B\_TH2\_2) \\ SCALEFAC \times am2[i] \times AN2\_2 & (send2 \times B\_TH2\_2 \le i \le send2) \end{cases}$$

Когда индекс *V/UV, vuv2*, равен 3:

$$am\_h[i] = \begin{cases} am2[i] & (1 \le i < send2 \times B\_TH3) \\ am2[i] \times AH3 & (send2 \times B\_TH3 \le i \le send2) \end{cases}$$

$$am\_noise[i] = \begin{cases} 0 & (1 \le i < send2 \times B\_TH3) \\ SCALEFAC \times am2[i] \times AN3 & (send2 \times B\_TH3 \le i \le send2) \end{cases}$$

Таблица 63 — Значения констант для модификации величины гармоник:

2,0 Кбит/с

*B TH*1	*AN*1	*AH*1	*B TH*2	*AN*2	*AH*2	*B TH*2 2	*AN*2 2	*AH*2 2	*B TH*3	*AN*3	*AH*3
0,5	0,4	0,8	0,5	0,3	0,9	0,85	0,5	0,5	0,7	0,2	1,0

4 Кбит/с

*B TH*1	*AN*1	*AH*1	*B TH*2	*AN*2	*AH*2	*B TH*2 2	*AN*2 2	*AH*2 2	*B TH*3	*AN*3	*AH*3
0,5	0,3	0,9	0,5	0,3	0,9	0,85	0,5	0,5	0,8	0,2	1,0

5.6.3.2 Синтез возбуждения гармоник

Сигнал возбуждения гармоник *sv*[*i*] ($0 \le i < FRM$) может быть получен методом быстрого синтеза, состоящим из *IFFT* и преобразования частоты дискретизации.

Во-первых, используя величины гармоник и значения фазы, форма волны за период одного шага генерируется *IFFT*. Согласно непрерывности шага, выбирается одна из двух различных операций циклического расширения и перевыборки формы волны за один период шага и выполняется, чтобы получить сигнал возбуждения гармоник.

Генерация формы волны за период одного шага.

Фундаментальная частота в начальной границе интервала декодирования, *w*01, вычислена как

$$w01 = 2{,}0 * PI / pch1.$$

Фундаментальная частота в конечной границе интервала декодирования, *w*02, вычислена как

$$w02 = 2{,}0 * PI / pch2.$$

Значения фазы гармоник в конечной границе интервала декодирования, *pha*2[], вычисляются из значений фазы гармоник в начальной границе, *pha*1[]. Когда и индекс *V/UV* в начале текущего интервала декодирования, *vuv*1, и индекс *V/UV* в начальной границе предыдущего интервала декодирования, *vuv*0, 0 *(Unvoiced)*, значения фазы гармоник *pha*2[] инициализируются, используя случайные значения фазы, равномерно распределенные между 0 и 0,5π.

Для режима нормальной задержки:

```
if (vuvl == 0 && vuv0 == 0) {
  for (i = 0; i < WAVE_LEN/2; i++)
   pha2[i] = 0,5*PI*(float)random()/(float)RND_MAX;
}
else {
  for (i = 0; i < WAVE_LEN/2; i++)
    pha2[i] = pha1[i] + 0,5*(w01+w02)*i*FRM;
}
```

Для режима малой задержки:

```
if (vuvl == 0 && vuv0 == 0) {
  for (i = 0; i < WAVE_LEN/2; i++)
    pha2[i] = 0,5*PI*(float)random()/(float)RND_MAX;
}
else {
  for (i = 0; i < WAVE_LEN/2; i++) {
    pha2[i] = pha1[i] + 0,5*(w01+w02)*i*(FRM-LD_LEN) + w01*i*LD_LEN;
}
```

На конечной границе интервала декодирования имеется спектр с *send*2 гармониками, чьи величины равны *am_h*[*i*] ($1 \le i \le send2$), а значения фазы равны *pha*2 [i] ($1 \le i \le send2$). Добавление нулей к этим массивам приводит к новым массивам с компонентами *WAVE_LEN/2* (= 64) в пределах от 0 до π. Если *send*2 больше, чем 63, используются первые 64 значения *of am_h*[*i*] и *pha*2[*i*]. 128 точек *IFFT* применены к этим массивам значений величин и фазы с ограничением, которое дает в результате

вещественные числа. Теперь есть форма волны передискретизации за период одного шага, *wave*2 [*i*] ($0 \le i < WAVE\_LEN$).

WAVE_LEN (= 128) отсчеты используются, чтобы выразить период одного шага формы волны. Так как фактическое значение задержки шага равно *pch*2, скорость передискретизации *ovsr*2 равна

$$ovsr2 = WAVE\_LEN / pch2.$$

Аналогично скорость передискретизации *ovsr*1 для формы волны за период одного шага на начальной границе интервала декодирования, *wave*1[], равна

$$ovsr1 = WAVE\_LEN / pch1.$$

Проверка непрерывности шага

Когда отношение фундаментальных частот |(*w*02 – *w*01) / *w*02| меньше, чем *WDEVI* (= 0,1), то переход шага непрерывен. В таком случае фундаментальные частоты и величины гармоник линейно интерполируются между началом и окончанием интервала декодирования. Иначе это расценивается как прерывистый переход шага, а фундаментальные частоты и величины гармоник не интерполированы линейно. В этом случае добавляются независимо синтезируемые периодические формы волны, используя соответствующие весовые функции.

Циклическое расширение и операция передискретизации (непрерывный переход шага)

Формы волны за период одного шага, *wave*1[] и *wave*2[], циклически расширены соответственно, чтобы иметь достаточную длину в домене с чрезвычайной передискретизацией.

Длина сверхдискретизированной формы волны, необходимая чтобы восстановить форму волны длины *FRM* (= 160) при оригинальной частоте дискретизации (8 кГц), составляет не менее *lp*12.

Для режима нормальной задержки:

```
lpl2 =
ceil(FRM*0,5*(over1+ovsr2));
lpl2r =
floor(FRM*0,5*(ovsr1 +ovsr2)+
0,5);
st = WAVE_LEN – (lp12r%WAVE_LEN);
for (i = 0; i < lp12; i++) {
  out1[i] = wave1[i%WAVE_LEN];
  out2[i] =
 wave2[(st+i)%WAVE_LEN];
 }
```

Для режима малой задержки:

```
lpl2 = ceil((FRM-LD_LEN)*0,5*(over1+ovsr2)+LD_LEN*ovsr1);
lpl2r = floor((FRM-LD_LEN) *0,5*(ovsr1 +ovsr2) +LD_LEN*ovsr1+0,5);
st = WAVE_LEN – (lp12r%WAVE_LEN)
iflat1 = floor(ovsr1*LD_LEN+0,5)
iflat2 = floor(ovsr2*LD_LEN+0,5)
for (i = 0; i < lp12+iflat2+10; i++) {
  out1[i] = wave1[i%WAVE_LEN];
  out2[i] = wave2[(st+i)%WAVE_LEN];
}
```

У этих двух форм волны, *out*1[] и *out*2[], есть тот же самый «псевдо» период шага (= *WAVE_LEN* [*sample*]), и они выравниваются. Так простое добавление этих двух форм волны, используя треугольные окна производит форму волны *out*3[].

Для режима нормальной задержки:

```
for (i = 0; i < lp12; i++)
  out3[i] = outl [i]*(float)(lp12-i)/(float)lp12 +
out2[i]*(float)i/(float)lp12;
```

Для режима малой задержки:

```
for (i = 0; i < iflatl; i++)
  out3[i] = out1[i];
```

```
for (i = iflatl; i < lp12; i++)
  out3[i] = outl [i]*(float)(lp12-i)/(float)(lp12-iflat1)
        +out2[i]*(float)(i-flat1)/(float)(lp12-iflat1);
for (i = lp12; i < lp12+iflat2; i++)
  out3[i] = out2[i].
```

Наконец, *out*3[] должен быть передискретизирован так, чтобы получающаяся форма волны могла быть выражена в оригинальной частоте дискретизации (8 кГц). Эта операция возвращает форму волны из домена «псевдо» шага в домен фактического шага. Операция передискретизации

$$sv\,[i] = out3\,[f(i)]\ (0 < i < FRM),$$

где

$$f(i) = \int_0 \left(ovsr1 \cdot \frac{FRM - t}{FRM} + ovsr2 \cdot \frac{t}{FRM} \right) dt.$$

Функция *f*(*i*) отображает время индекса *i* из оригинальной частоты дискретизации (8 кГц) к индексу времени в *over-sampledrate* при условии, что фундаментальные частоты *w*01 и *w*02 линейно интерполированы. С тех пор *f*(*i*) doesnot возвращают целочисленное значение, *sv*[*i*] получен, линейно интерполируя *out*3 [*f*(*i*)] и *out*3 [*f*(*i*)].

Для нормального режима задержки:

```
sv[0] = out3[0];
ffi = 0;
for (i = 1; i < FRM; i++) {
  ovsrc = ovsr1*(float)(FRM-i)/(float)FRM +
  over2*(float)i/(float)FRM;
  ffi += ovsrc;
  ffim = floor(ffi);
  ffip = ceil(ffi);
  if (ffim == ffip)
    sv[i] = out3[(int)ffip];
  else
    sv[i] = (ffi-ffim)*out3[(int)ffip] + (ffip–
    ffi)*out3[(int)ffim];
  }
```

Для режима малой задержки:

```
sv[0] = out3[iflat1];
ffi = 0;
for (i = 1; i < FRM; i++) {
  ovsrc = ovsrl *(float)(FRM-LD_LEN-i)/(float)(FRM-LD_LEN)
        + over2*(float)i/(float) (FRM-LD_LEN);
  ffi += ovsrc;
  ffim = floor(ffi);
  ffip = ceil(ffi);
  if (ffim == ffip)
    sv[i] = out3[(int)ffip+iflat1];
  else
    sv[i] = (ffi-ffim)*out3[(int)ffip+iflat1]
        + (ffip-ffi)*out3[(int)ffim+iflat1];
}
```

Циклическое расширение и операция передискретизации (прерывистый переход шага)

Формы волны за один период шага *wave*1[] и *wave*2[] циклически расширены, чтобы иметь достаточную длину в домене передискретизации. И на начальной и на конечной границах интервала декодирования получены циклически расширенные формы волны *out*1[] и *out*2[].

```
lp1 = ceil(FRM*ovsr1);
lp2 = ceil(FRM*ovsr2);
lp2r = floor(FRM*ovsr2+0,5);
st = WAVE_LEN – (lp2r%WAVE_LEN)
```

Для режима нормальной задержки:

```
for (i = 0; i < lp1; i++)
  out1[i] = wave1[i%WAVE_LEN];
for (i = 0; i < lp2; i++)
  out2[i] = wave2[(st+i)%WAVE_LEN]
```

Для режима малой задержки:

```
iflat1 = floor(ovsr1*LD_LEN+0,5);
iflat2 = floor(ovsr2*LD_LEN+0,5);
for (i = 0; i < lp1+iflat1+10; i++)
  out1[i] = wave1[i%WAVE_LEN];
for (i = 0; i < lp2+iflat2+10; i++)
  out2[i] = wave2[(st+i)%WAVE_LEN],
```

где *lp*1 и *lp*2 — длина передискретизированных форм волны, необходимая чтобы восстановить форму волны длиной *FRM* (= 160) при первоначальной частоте дискретизации (8 кГц).

Эти две формы волны, *out*1[] и *out*2[], передискретизируются независимо, используя тот же самый метод линейной интерполяции, как в случае «непрерывного перехода шага».

Для режима нормальной задержки:

```
sv1[0] = out1[0];
ffi = 0;
for (i = 1; i < FRM; i++) {
  ffi += ovsr1;
  ffim = floor(ffi);
  ffip = ceil(ffi);
  if (ffim == ffip)
    sv1[i] = outl [(int)ffip];
  else
    sv1[i] = (ffi-ffim)*out1 [(int)ffip] + (ffip-ffi)*out1 [(int)ffim];
  }
sv2[0] = out2[0];
ffi = 0;
for (i = 1; i < FRM; i++) {
  ffi += ovsr2;
  ffim = floor(ffi);
  ffip = ceil(ffi);
  if (ffim == ffip)
    sv2[i] = out2[(int)ffip];
  else
    sv2[i] = (ffi-ffim)*out2[(int)ffip] + (ffip-ffi)*out2[(int)ffim];
  }
```

Для режима малой задержки:

```
sv1[0] = out1[iflat1];
ffi = 0;
for (i = 1; i < FRM; i++) {
  ffi += ovsr1;
  ffim = floor(ffi);
  ffip = ceil(ffi);
  if (ffim == ffip)
    sv1[i] = out1[(int)ffip+iflat1];
  else
    sv1[i] = (ffi-ffim)*out1[(int)ffip+iflat1]
          + (ffip-ffi)*out1 [(int)ffim+iflat1];
  }
sv2[0] = out2[iflat2];
ffi = 0;
for (i = 1; i < FRM; i++) {
  ffi += ovsr2;
  ffim = floor(ffi);
```

```
ffip = ceil(ffi);
if (ffim == ffip)
  sv2[i] = out2[(int)ffip+iflat2];
else
  sv2[i] = (ffi-ffim)*out2[(int)ffip+iflat2]
       + (ffip-ffi)*out2[(int)ffim+iflat2];
}
```

Используя передискретизированные формы волны при первоначальной частоте дискретизации (8 кГц) *sv*1[] и *sv*2[], тогда перекрываются и добавлены окном трапецоида *c_dis*[], показанным на рисунке 2, где *HP_UP* = *HM_DOWN* = 60, *HM_FLAT* = 50.

```
for (i = 0; i < HM_FLAT; i++)
  c_dis[i] = 1,0f;
for (i = HM_FLAT; i < HM_FLAT+HM_DOWN; i++)
  c_dis[i] = (-i+(HM_FLAT+HM_DOWN))/(float) HM_DOWN;
for (i = HM_FLAT+HM_DOWN; i < FRM; i++)
  c_dis[i] = 0.0f;
```

Для режима нормальной задержки:

```
for (i = 0; i < FRM; i++)
  sv[i] = sv1[i]*c_dis[i] + sv2[i]*(1,0 – c_dis[i]);
```

Для режима малой задержки:

```
for (i = 0; i < FRM; i++)
  sv[i] = sv1[i]*c_dis[i+LD_LEN] + sv2[i]*(1,0 – c_dis[i + LD_LEN])
```

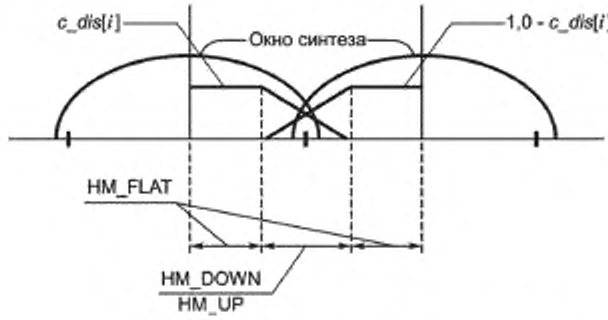


Рисунок 2 — Окно синтеза для прерывающегося шага 38

5.6.3.3 Генерация шумовой компоненты

Для генерации шумовой компоненты для речевого возбуждения сначала генерируется белый гауссовский шум. Затем он настраивается по цвету и усилению модифицированными величинами гармоник, *am_noise*[], а чтобы генерировать непрерывную шумовую компоненту во временном домене, используются взвешенное перекрытие и добавление.

Сначала определяется окно Хемминга длины *SAMPLE* (= 256), *ham*[*i*].

Для режима нормальной задержки:

```
for (i = 0; i < SAMPLE; i++)
  ham[i] = 0,54–0,46*cos(2.0*PI*i/(SAMPLE-1))
  Для режима малой задержки:
for (i = 0; i < (SAMPLE–HAMLD)/2; i++)
  ham[i] = 0.0;
for (i = (SAMPLE-HAMLD)/2; i < (SAMPLE+HAMLD)/2; i++)
  ham[i] = 0,54-0,46*cos(2.0*PI*i/(SAMPLE-1));
for (i = (SAMPLE+HAMLD)/2; i < SAMPLE; i++)
ham[i] = 0.0;
```

ham[] окна нормализуется, чтобы получить единичную энергию.

Пусть *ns*[*I*] (0 ≤ *i* < *SAMPLE*) будет отсчетами белого гауссовского шума с нулевым средним значением и единичной дисперсией. Затем окно Хэмминга *ham*[*i*] умножается на *ns*[*I*] и получается *wns*[*I*].

```
for (i = 0; i < SAMPLE; i++)
  wns[i] = ns[i]*ham[i].
```

Обсчитывают 256 точек *FFTwns*[*i*] и вычисляют массив спектральных амплитуд *rmc*[*i*] ($0 \le I \le SAMPLE/2$) и массив спектральных фаз *ang*[*i*] ($0 \le i \le SAMPLE/2$) как

```
for (i = 0; i <= SAMPLE/2; i++) {
  rms[i] = sqrt(re[i]*re[i]+im[i]*im[i]);
  ang[i] = atan2(im[i], re[i]);
}
```

где *re*[*i*] ($0 \le I < SAMPLE$) и *im*[*i*] ($0 \le i < SAMPLE$) являются вещественной частью и мнимой частью коэффициентов *FFT* соответственно. Затем спектральное среднеквадратичное значение амплитуды *rmc*[] настраивается по цвету и усилению модифицированными гармоническими величинами *am_noise*[]. (*w*0 является фундаментальной частотой текущего фрейма, где w0 значение *SAMPLE* представляет собой 2π.)

```
for (i = 0; i <= send2; i++) {
  if (i == 0)
    lb = 0;
  else
    lb = ub+1;
  if (i == send2)
    ub = SAMPLE/2
  else
    ub = floor((float)i*w0+w0/2.0+0,5);
  if (ub >= SAMPLE/2)
    ub = SAMPLE/2;
  bw = ub-lb+1;
  s = 0.0;
  for (j = lb; j <= ub; j++)
    s += rms[j]*rms[j];
  s = sqrt(s/(float)bw);
  for (j = lb; j <= ub; j++)
    rms[j] *= am_noise[i]/s;
}
```

256-точечный *IFFT* вычисляется с настроенным по цвету и усилению массивом спектральных амплитуд *rmc*[] и исходным массивом спектральных фаз *ang*[] с ограничением, что результат будет вещественными числами. Пусть результат *IFFT* будет *cns*[*i*] ($0 \le I \le SAMPLE$).

Когда текущий фрейм является "*Unvoiced*",

frame (vuv2 = 0), cns[i] = 0.0 (0 ≤ i < SAMPLE).

Чтобы генерировать сигнал шумовой компоненты по интервалу декодирования, выполняются взвешенное перекрытие и добавление результата *IFFT* предыдущего фрейма. Массив *cns_z*[*i*] ($0 \le i < 2Ч FRM$) получен путем дополнения (*FRM-SAMPLE/2*) нулей к обеим сторонам (начало и окончание) *cns*[].

```
for (i = 0; i < FRM-SAMPLE/2; i++)
cns_z[i] = 0.0;
for (i = FRM-SAMPLE/2; i < FRM+SAMPLE/2; i++)
  cns_z[i] = cns[i-FRM+SAMPLE/2];
for (i = FRM+SAMPLE/2; i < 2*FRM; i++)
  cns_z[i] = 0.0.
```

Тем же самым способом дополненный нулями массив окна Хэмминга *ham_z*[*i*] ($0 \le i < 2Ч FRM$) определен как:

```
for (i = 0; i < FRM-SAMPLE/2; i++)
  ham_z[i] = 0.0;
for (i = FRM-SAMPLE/2; i < FRM+SAMPLE/2; i++)
  ham_z[i] = ham[i-FRM+SAMPLE/2];
for (i = FRM+SAMPLE/2; i < 2*FRM; i++)
  ham_z[i] = 0.0.
```

Обозначим *cns_z*[] предыдущего фрейма как *cns_z_p*[]. Теперь шумовая компонента по интервалу декодирования *add_uv*[*i*] ($0 \le I < FRM$) получается объединением *cns_z*[] и *cns_z_p*[].

Для режима нормальной задержки:

```
for (i = 0; i < FRM; i++)
  add_uv[i] = (cns_z_p[FRM+i]*ham_z[FRM+i]+cns_z[i]*ham_z[i])
              /(ham_z[FRM+i]*ham_z[FRM+i]+ham_z[i]*ham_z[i]).
```

Для режима малой задержки шумовая компонента в сдвинутом интервале декодирования получается следующим образом:

```
for (i = 0; i < FRM-LD_LEN; i++)
  add_uv[i] = (cns_z_p[FRM+i+LD_LEN]*ham_z[FRM+i+LD_LEN]+
               cns_z[i+LD_LEN]*ham_z[i+LD_LEN])
              /(ham_z[FRM+i+LD_LEN]*ham_z[FRM+i+LD_LEN]
                +ham_z[i+LD_LEN]*ham_z[i+LD_LEN]);
for (i = FRM-LDLEN; i < FRM; i++)
  add_uv[i] = cns_z[i+LD_LEN]/ham_z[i+LD_LEN]).
```

Шумовая компонента *add_uv*[*i*] добавляется к сигналу возбуждения гармоник *sv*[], чтобы создать сигнал речевого возбуждения:

```
for (i = 0; i < FRM; i++)
  sv[i] += add_uv[i];
```

5.6.3.4 Синтез *LPC*

Сигнал возбуждения, полученный ранее, *sv*[*i*] ($0 \le i < FRM$), подается на фильтр синтеза *LPC*, коэффициенты которого обновляются каждые 2,5 мс (= 20 отсчетов).

Линейным интерполированием деквантованных *LSPs* предыдущего и текущего фрейма, *lsp*1[] [] и *lsp*2[] [],получают 8 наборов интерполированного *LSPs*, *lspip*[] [].

```
for (i = 0; i < 8; i++)
  for (j = 0; j < P; j++)
    lspip[i] [j] = (2.0*i+ 1,0)/16.0*lsp2[j] + (16.0-2.0*i-1,0)/16.0*lsp1[j].
```

Для режима малой задержки, так как интервал декодирования сдвинут на *LD_LEN*(= 20) отсчетов (2,5 мс), интерполяция *LSPs* выполнена для первых 17,5 мс интервала декодирования, показанного на рисунке 9, а *LSPs* текущего фрейма, *lsp*2[], используется для последних 2,5 мс без интерполяции.

```
for(i = 0; i < 7; i++)
  for (j = 0; j < P; j++)
    lspip[i]jj] = (2.0*i+ 1,0)/14.0*lsp2[j] + (14.0–2,0*
    i–1,0)/14.0*lsp1[j];
for (j = 0; j < P; j++)
  lspip[7][j] = lsp2[j];
```

8 наборов интерполированных *LSPs*, *lspip*[] [], преобразованы в коэффициенты линейного прогнозирования *alpaip*[] [] соответственно.

Функция преобразования *i*-го интервала 2,5 мс (20 отсчетов) фильтра синтеза *LPC* равна

$$H_i(z) = \frac{1}{\sum_{n=0}^{P} alphair[i][n]z^{-n}} (0 \le i < 8).$$

Выход фильтра синтеза *LPC* подан в постфильтр, описанный в Б.1.3.1. Выходной сигнал находится в диапазоне от – 32768 до 32767.

5.7 Синтезатор неречевой компоненты

5.7.1 Описание инструмента

Синтезатор неречевой компоненты составлен из трех шагов, которые являются выделением окон неголосового сигнала возбуждения, фильтром синтеза *LPC* и работой постфильтра. Для неречевых сегментов используется схема *VXC (CELP)*.

5.7.2 Определения

Определения констант

FRM: Интервал фрейма (= 160).

LD_LEN: Сдвиг интервала декодирования для режима малой задержки (= 20).

w_*celp_up*[*i*]: Окно из речевого фрейма в неречевой фрейм ($0 \le i < FRM + LD\_LEN$).

w_celp_down[*i*]: Окно из неречевого фрейма в речевой фрейм ($0 \le i < FRM + LD\_LEN$).

P: Порядок *LPC* (= 10).

Определения переменных

qRes [*i*]: Декодированный неречевой сигнал возбуждения, полученный как *res* [*i*] в 5.4.3 ($0 \le i < FRM$).

old_qRes [*i*]: Последняя половина декодированного неречевого сигнала возбуждения предыдущего фрейма ($0 \le i < FRM/2$).

suv[*i*]: Неречевой сигнал возбуждения по интервалу декодирования ($0 \le i < FRM$).

*vuv*2: Индекс *V/UV* текущего фрейма, полученный как *VUV* в потоке битов.

*vuv*1: Индекс *V/UV* предыдущего фрейма.

*lsp*2[]: Деквантованные *LSPs* текущего фрейма, полученного как *qLsp*[] в 5.2.3.

*lsp*1[]: Деквантованные *LSPs* предыдущего фрейма.

*alpha*2[]: Коэффициенты *LPC*, преобразованные из *LSPs lsp*2[].

*alpha*1[]: Коэффициенты *LPC*, конвертированные из *LSPs lsp*1[].

5.7.3 Процесс синтеза

Сигнал неречевого возбуждения в интервале декодирования *suv*[] сгенерирован из декодированного сигнала неречевого возбуждения текущего фрейма *qRes*[] и последней половины декодированного сигнала неречевого возбуждения предыдущего фрейма *old_qRes*[].

```
for (i = 0; i < FRM/2; i++) {
  suv[i] = old_qRes[i];
  suv[i+FRM/2] = qRes[i];
  old_qRes[i] = qRes[i+FRM/2];
}
```

Для режима малой задержки используется сигнал возбуждения сдвинутой версии выборок *LD_LEN* (= 20):

```
for (i = 0; i < FRM/2-LDLEN; i++) {
  suv[i] = old_qRes[i+LD_LEN];
  suv[i+FRM/2] = qRes[i+LD_LEN];
  old_qRes[i] = qRes[i+FRM/2];
}
for (i = FRM/2-LD_LEN; i < FRM/2; i++) {
  suv[i] = qRes[i-FRM/2+LD_LEN];
  suv[i+FRM/2] = qRes[i+LD_LEN];
  old_qRes[i] = qRes[i+FRM/2];
}
```

Сгенерированный сигнал неречевого возбуждения является реализуемым методом окна, чтобы подключаться к речевому фрейму. Рисунки 3 и 4 показывают форму окна для формы волны возбуждения, где *V/UV* переходит от неречевого к речевому и от речевого к неречевому соответственно. Параметры на рисунке установлены как: *TD_UP* = 30, *TD_FLAT* = 50, *TD_DOWN* = 30, *HM_DOWN* = 60, *HM_FLAT* = 50, *HM_UP* = 60. Эти окна для неречевого фрейма используются, только когда неречевой фрейм смежный с речевым или смешанным речевым фреймом.

```
for (i = 0; i < FRM–TD_UP–TD_FLAT; i++)
  w_celp_up[i] = 0.0;
for (i = FRM–TD_UP–TD_FLAT; i < FRM–TD_FLAT; i++)
  w_celp_up[i] = (float) (i–FRM+TD_UP+TD_FLAT)/(float) TD_UP;
for (FRM–TD_FLAT; i < FRM+LD_LEN; i++)
  w_celp_up[i] = 1,0; for (i = 0; i < TD_FLAT; i++)
  w_celp_down[i] = 1,0;
for (i = TD_FLAT; i < TD_FLAT+ TD_DOWN; i++)
  w_celp_down[i] = (float) (TD_FLAT+ TD_DOWN–i)/(float) TD_DOWN;
for (i = TD_FLAT+TD_DOWN; i < FRM+LD_LEN; i++)
  w_celp_down[i] = 0.0;
```

Для режима нормальной задержки:

```
if (vuvl != 0 && vuv2 != 0) {
  for (i = 0; i < FRM; i++)
    s_uv[i] = 0.0f;
}
```

```
else if (vuv1 != 0 && vuv2 == 0) {
  for (i = 0; i < FRM; i++)
    s_uv[i] *= w_celp_up[i];
}
else if (vuv1 == 0 && vuv2 != 0) {
  for (i = 0; i < FRM; i++)
    s_uv[i] *= w_celp_down[i]
}
```

Для режима малой задержки позиция работы с окнами сдвинута выборками *LD_LEN*:

```
if (vuv1 != 0 && vuv2 != 0) {
  for (i = 0; i < FRM; i++)
    s_uv[i] = 0.0f;
}
else if (vuv1 != 0 && vuv2 == 0) {
  for (i = 0; i < FRM; i++)
    s_uv[i] *= w_celp_up[i+LD_LEN];
}
else if (vuv1 == 0 && vuv2 != 0) {
  for (i = 0; i < FRM; i++)
    s_uv[i] *= w_celp_down[i+LD_LEN]
}
```

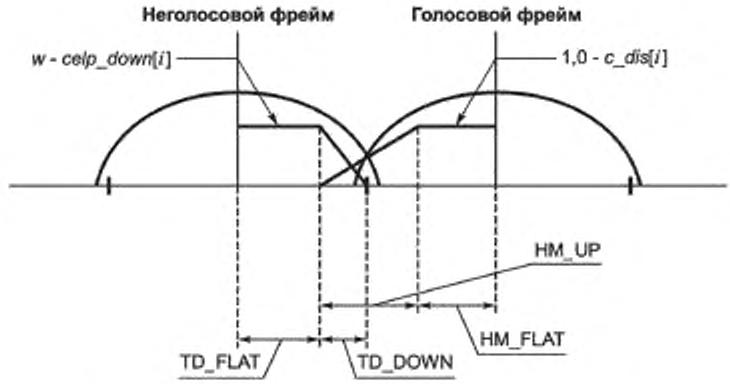


Рисунок 3 — Окно синтеза для случая неречевой/речевой

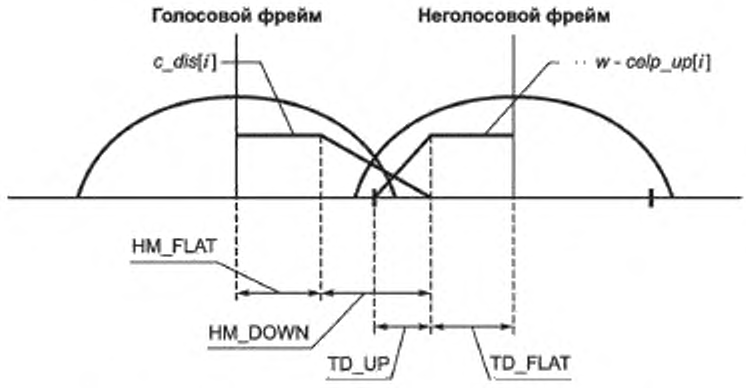


Рисунок 4 — Окно синтеза для случая речевой/неречевой

Первая половина неречевого сигнала возбуждения, *suv*[*i*] ($0 \le i < FRM/2$), подается в фильтр синтеза *LPC*

$$H_1(z) = \frac{1}{\sum_{n=0}^{P} alphair1[n]z^{-n}},$$

где *alpha*1[] являются линейными прогнозирующими коэффициентами, преобразованными из деквантованных *LSPs* предыдущего фрейма, *lsp*1[]. Когда вторая половина неречевого сигнала возбуждения, *suv*[*i*] ($FRM/2 \le i < FRM$), подается в фильтр синтеза *LPC*, функция преобразования переключается в

$$H_2(z) = \frac{1}{\sum_{n=0}^{P} alphair2[n]z^{-n}},$$

где *alpha*2[] являются линейными прогнозирующими коэффициентами, преобразованными из деквантованных *LSPs* текущего фрейма, *lsp*2[].

Для режима малой задержки, так как интервал декодирования сдвинут на *LD_LEN* выборок, неречевой сигнал возбуждения *suv*[*i*] ($0 \le i < FRM/2\text{-}LD\_LEN$) подается в фильтр синтеза *LPC* $H_1(z)$, а неречевой сигнал возбуждения ($FRM/2 - LD\_LEN \le i < FRM$) подается в фильтр синтеза *LPC* $H_2(z)$.

Выход фильтра синтеза *LPC* подается в постфильтр, описанный в Б.1.3.2. Выходной сигнал находится в диапазоне от – 32768 до 32767.

5.8 Декодер варьируемой скорости

5.8.1 Описание инструмента

Инструмент для декодирования переменной скорости с ядром *HVXC*.

Этот инструмент позволяет *HVXC* работать в потоках данных с переменной скоростью, где средняя битовая скорость уменьшена до 1,2—1,7 Кбит/с для типичного речевого материала. Главная часть алгоритма составлена из «декодирования фонового шума», где только биты режима получают во время «режима фонового шума», и неречевой фрейм получают через определенный промежуток времени для генерации фонового шума.

5.8.2 Определения

Определения констант

BGN_INTVL: Максимальный интервал фонового шума (= 8).

Определения переменных

idVUV: Решение *V/UV* текущего фрейма.

*prevLSP*1: Вектор ранее переданного *LSP*.

*prevLSP*2: Вектор ранее переданного *LSP* прежде *prevLSP*1.

bgnCnt: Счет последовательных фреймов «фонового шума».

5.8.3 Процесс декодирования

idVUV — параметр, который имеет в результате решение *V/UV* и определенный как:

$$idVUV = \begin{cases} 0\ \text{Непроизнесенная речь} \\ 1\ \text{Интервал фонового шума} \\ 2\ \text{Смешанная вокализованная речь} \\ 3\ \text{Вокализованная речь} \end{cases}$$

Используя метод обнаружения фонового шума, кодирование с варьируемой скоростью выполняется на базе фиксированной битовой скорости 2,0 Кбит/с *HVXC*.

Таблица 64 — Распределение разрядов закодированных параметров для режима варьируемой битовой скорости

Режим (*idVUV*)	Фоновый шум (1)	*UV* (0)	*MV* (2), *V* (3)
V/UV	2 бит/20 мс	2 бит/20 мс	2 бит/20 мс
LSP	0 бит/20 мс	18 бит/20 мс	18 бит/20 мс

Окончание таблицы 64

Режим (*idVUV*)	Фоновый шум (1)	*UV* (0)	*MV* (2), *V* (3)
Возбуждение	0 бит/20 мс	8 бит/20 мс (только усиление)	20 бит/20 мс (Основной тон и гармонические параметры спектра)
Совокупно	2 бит/20 мс 0,1 Кбит/с	28 бит/20 мс 1,4 Кбит/с	40 бит/20 мс 2,0 Кбит/с

Для Смешанной вокализованной речи и Вокализованной речи (*idVUV* = 2,3) используется тот же самый метод декодирования, как для режима фиксированной битовой скорости.

В декодере содержатся два набора параметров *LSP*, *prevLSP*1 и *prevLSP*2, где *prevLSP*1 представляет ранее переданные параметры *LSP*, а *prevLSP*2 представляет ранее переданные параметры *LSP* прежде *prevLSP*1. Для фрейма фонового шума (*idVUV* = 1) декодер *VXC* используется тем же самым способом, как для фрейма *UV*, но никакие параметры *LSP* не передаются. Сгенерированные линейной интерполяцией параметры *LSP prevLSP*1 и *prevLSP*2 используются для синтеза *LPC*, и тот же самый индекс усиления предыдущего фрейма используется для генерации возбуждения декодирования *VXC*. Во время фрейма фонового шума, через каждые (*BGN_INTVL* + 1) (= 9) фреймов вставляется фрейм Невокализованный речевой (*UV*), чтобы передать параметры фонового шума. Этот фрейм *UV* может быть или не быть реальным *UV*-фреймом начала речевых пакетов. Является ли фрейм реальным *UV*, оценивает переданный индекс усиления. Если индекс усиления меньше или равен индексу из предыдущего + 2, то этот *UV*-фрейм расценивается как фрейм фонового шума, и поэтому ранее переданный вектор *LSP* (= *prevLSP*1) используется, чтобы сохранить гладкое изменение параметров *LSP*, иначе передаваемые в данное время *LSPs* используются как реальный *UV*-фрейм. Индексы усиления сортируются согласно величинам. Если снова выбран режим «фоновый шум», то используют интерполированные *LSPs*, использующие *prevLSP*1 и *prevLSP*2.

И для фрейма Невокализованной речи и для фонового шума (*idVUV* = 0,1) используется гауссовский шум с единичной энергией для возбуждения декодирования *VXC* (вместо стохастического кодового вектора формы для декодирования *VXC*).

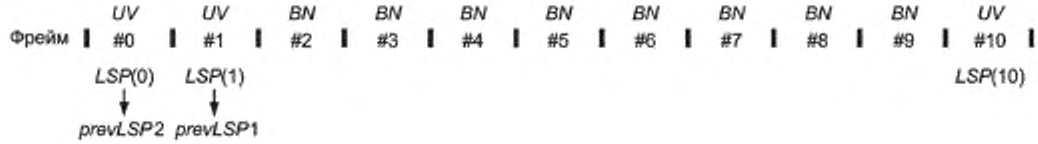
Рисунок 5 показывает пример. Предположим, что фрейм #0 и фрейм # 1 являются Невокализованным речевым фреймом, а фрейм #2 ... фрейм #9 являются фреймом фонового шума. Во время декодирования фрейма #2 ... фрейма #9, *prevLSP*1 и *prevLSP*2 установлены как: *prevLSP*1 = *LSP* (1) и *prevLSP*2 = *LSP* (0) и вектор *LSP* фрейма #*i*, *LSP* (*i*) ($2 \le i \le 9$), генерируется как

$$LSP(i) = \frac{prevLSP2 \times (2 \times BGH\_INTVL - 2 \times bgnCnt - 1) + prevLSP1 \times (2 \times bgnCnt + 1)}{2 \times BGH\_INTVL},$$

где *BGN_INTVL* — интервал максимального фонового шума (= 8),

bgnCnt — является счетчиком последовательных фреймов фонового шума.

В этом примере *bgnCnt* = 0 для фрейма #2, *bgnCnt* = 1 для фрейма #3, ..., *bgnCnt* = 7 для фрейма #9. Для индекса усиления декодирования *VXC* во время фрейма #2 ... фрейма #9 используется индекс усиления фрейма #1. Когда получены параметры фрейма #10, *prevLSP*1 и *prevLSP*2 обновляются как: *prevLSP*1 = *LSP*(10) и *prevLSP*2 = *LS*(1). Для декодирования фрейма #10 сначала проверяется, больше ли индекс усиления, чем значение индекса "фрейм #1 + 2". Если больше, фрейм #10 декодируется как обычный *UV*-фрейм; иначе он декодируется как фрейм фонового шума и вместо *LSP*(10) используется *LSP*(1), в то время как полученный индекс усиления используется для обоих случаев.



UV – неголосовой фрейм;
BN – фрейм фонового шума

Рисунок 5 — Генерация параметров декодера для интервала фонового шума

5.9 Расширение режима варьируемой скорости *HVXC*

5.9.1 Описание инструмента

В 5.8 описан режим с варьируемой битовой скоростью, основанный на режиме 2,0 Кбит/с. Здесь описан режим с варьируемой битовой скоростью максимум 4,0 Кбит/с.

В режиме с фиксированной битовой скоростью есть решение 2-битового *V/UV*, а именно:

VUV = 3: полностью речевой, *VUV* = 2: смешанный речевой, *VUV* = 1: смешанный речевой, *VUV* = 0: неречевой.

Когда режимом работы является режим с переменной битовой скоростью, *VUV* = 1 указывает состояние фонового шума вместо смешанного речевого. Текущий операционный режим определяется "*HVXCconfig* ()", и декодер знает, является ли это режимом переменной или фиксированной скорости, и может понять значение *VUV* = 1. В кодировании варьируемой скорости распределение разрядов меняется в зависимости от речевого/неречевого решения, и сохранение битовой скорости получают, главным образом уменьшая распределение разрядов для сегмента невокализованной речи (*VUV* = 0). Когда выбран *VUV* = 0, тогда проверяется, является ли сегмент реальным сегменом невокализованной речи или сегментом фонового шума. Если он заявлен как фоновый шум, то *VUV* изменяется на 1 и распределение разрядов для фрейма дополнительно уменьшается. Во время режима фонового шума передаются только биты режима или фрейм обновления шума согласно изменению характеристик фонового шума. Используя этот режим варьируемой скорости, средняя битовая скорость уменьшается до 56 % — 85 % режима с фиксированной битовой скоростью в зависимости от исходных элементов.

5.9.2 Определения

Определения констант

*NUM_SUBF*1: Число подфреймов в одном фрейме (= 2).
*NUM_SHAPE_L*0: Число индекса книги шифров (= 64).
BGN_INTVL: Интервал обновления фонового шума (= 12).

Определения переменных

*prevLSP*1: Переданные параметры *LSP*.
*prevLSP*2: Переданные параметры *LSP* прежде *prevLSP*1.
qLsp: *LSP*, который будет использоваться для операции декодирования текущего фрейма.
bgnIntval: Счетчик, который считает число последовательных фреймов фонового шума.
rnd: Случайно сгенерированное целочисленное значение между – 3 и 3.

5.9.3 Полезная нагрузка передачи

В зависимости от решения *V/UV* и результата обнаружения фонового шума используются полезные нагрузки передачи с четырьмя различными битовыми скоростями. Флажки *VUV* и *UpdateFlag* указывают тип передачи полезной нагрузки.

VUV является параметром, который содержит результат решения *V/UV* и определяется как:

$$VUV = \begin{cases} 0\ \text{Невокализованная речь} \\ 1\ \text{Интервал фонового шума} \\ 2\ \text{Вокализованная речь 1} \\ 3\ \text{Вокализованная речь 2} \end{cases}$$

Чтобы определить, является ли отмеченный "*VUV* = 1" фрейм фреймом обновления шума, вводится параметр "*UpdateFlag*". *UpdateFlag* используется, только когда *VUV* = 1.

$$UpdateFlag = \begin{cases} 0\ \text{нефрейм обновления шума} \\ 1\ \text{фрейм обновления шума} \end{cases}$$

Если *UpdateFlag* равно 0, фрейм не является фреймом обновления шума, и если *UpdateFlag* равно 1, фрейм является фреймом обновления шума. Первый фрейм режима «фоновый шум» всегда классифицируется как фрейм обновления шума. Кроме того, если усиление или огибающая спектра фрейма фонового шума изменяются, вводится фрейм обновления шума.

В фрейме обновления шума подсчитывается среднее параметров *LSP* по последним 3 фреймам и кодируется как индексы *LSP* в кодере. Тем же самым способом подсчитывается среднее усиление *Celp* по последним 4 фреймам (8 подфреймов) и кодируется как индекс усиления *Celp*.

Во время интервала фонового шума (*VUV* = 1) параметры *LSP* и параметры возбуждения посылают, только когда выбран шумовой фрейм обновления (*UpdateFlag* = 1). Выходные сигналы декодера для интервала фонового шума генерируются с использованием *LSP* и параметров возбуждения, переданные в шумовых фреймах обновления.

Если текущий фрейм или предыдущий фрейм является режимом «фоновый шум», другой режим в квантовании *LSP* запрещен в кодере, потому что параметры *LSP* не посылаются во время режима «фоновый шум», а межфреймовое кодирование невозможно.

Используя описанный выше метод обнаружения фонового шума, кодирование с варьируемой скоростью выполнено на базе *HVXC* фиксированной битовой скорости 4 Кбит/с. Битовая скорость в каждом режиме показана ниже.

Режим (*VUV*)	Фоновый шум (1)		*UV*(0)	*V*(2,3)
	Флагобновления = 0	Флагобновления = 1		
Флагобновления *V/UV* Возбуждение *LSP*	2 бит/20 мс 1 бит/20 мс 0 бит/20 мс	2 бит/20 мс 1 бит/20 мс 18 бит/20 мс 4 бит/20мс (только усиление)	2 бит/20 мс 0 бит/20 мс 18 бит/20 мс 20 бит/20 мс	2 бит/20 мс 0 бит/20 мс 26 бит/20 мс 52 бит/20 мс
Совокупно	3 бит/20 мс 0,15 Кбит/с	25 бит/20 мс 1,25 Кбит/с	40 бит/20 мс 2,0 Кбит/с	80 бит/20 мс 4,0 Кбит/с

5.9.4 Процесс декодирования

В декодере речевой фрейм (*VUV* = 2,3) обрабатывается тем же самым способом, как в режиме с фиксированной битовой скоростью 4 Кбит/с, а неречевой фрейм (*VUV* = 0) обрабатывается тем же способом, как в режиме с фиксированной битовой скоростью 2,0 Кбит/с. Когда выбран режим фонового шума (*VUV* = 1), выходной сигнал декодера генерируется так же, как невокализованный речевой в режиме с фиксированной битовой скоростью 2,0 Кбит/с. Параметры декодера для интервала обратного фонового шума генерируются при использовании параметров, переданных в шумовых фреймах обновления (*VUV* = 1, *UpdateFlag* = 1) и иногда в предшествующих неречевых фреймах (*VUV* = 0). Ниже показано, как генерировать параметры декодера для интервала обратного фонового шума.

5.9.4.1 Декодирование *LSP*

В декодере удерживаются два набора ранее переданных параметров *LSP*, *prevLSP*1 и *prevLSP*2.

*prevLSP*1: переданные параметры *LSP*

*prevLSP*2: переданные прежде *prevLSP*1 параметры *LSP*

Режим фонового шума имеет место только после режима «неречевой» или «фоновый шум». Если выбран режим фонового шума, параметры *LSP* передаются, только когда фрейм является шумовым фреймом обновления (*UpdateFlag* = 1). Если переданы новые параметры *LSP*, *prevLSP*1 копируется в *prevLSP*2, и недавно переданные *LSPs* копируются в *prevLSP*1 независимо от решения *VUV*.

Параметры *LSP* для каждого фрейма во время режима фонового шума генерируются интерполяцией между *prevLSP*1 и *prevLSP*2, используя уравнение

$$qLsp(i) = ratio \cdot prevLsp1(i) + (1 - ratio) \cdot prevLsp2(i) ... i = 1...10, \quad (6)$$

где

$$ratio = \frac{2 \cdot (bgnIntval + md) + 1}{2 \cdot BGN\_INTVL}, \quad (7)$$

qLsp (i) является *i*-м *LSP*, который должен использоваться для операции декодирования текущего фрейма, *prevLsp*1 (*i*) является *i*-м *LSP prevLSP*1, *prevLsp*2 (*i*) является *i*-м *LSP prevLSP*2 ($1 \le i \le 10$). В этом уравнении *bgnIntval* представляет собой счетчик, который считает число последовательных фреймов фонового шума и сбрасывается в 0 при приеме фрейма обновления фонового шума. *BGN_INTVL* (= 12) является константой, а *rnd* — случайно сгенерированное целочисленное значение между – 3 и 3. Если счетчик *bgnIntval* достигает *BGN_INTVL*, *bgnIntvl* устанавливается в *BGN_INTVL-1*, и если отношение,

полученное из уравнения (7), меньше 0 или больше 1, значение *rnd* установливается в 0 и отношение пересчитывается.

5.9.5 Генерация возбуждения

Во время режима фонового шума индекс усиления (*VX_gain* [0]), переданный в шумовом фрейме обновления, используется для всех подфреймов, значения индекса *Shape* (*VX_Shape*1 [0,1]) случайным образом сгенерированы между 0 и *NUM_SHAPE_L*0-1. Эти параметры возбуждения используются с интерполированными параметрами *LSP*, как описано ранее, чтобы генерировать сигналы режима фонового шума.

Приложение А
(справочное)

Инструменты кодера *HVXC*

А.1 Краткий обзор инструментов кодера

Речевой ввод при частоте дискретизации 8 кГц сформирован во фреймы с длиной и интервалом 256 и 160 отсчетов соответственно. Анализ *LPC* выполнен, используя windowed (обработанные методом окна) входные данные по одному фрейму. Остаточные сигналы *LPC* вычисляются обратной фильтрацией входных данных, используя параметры квантованного и интерполированного *LSP*. Остаточные сигналы затем подаются в блок оценки величины шага и спектра, где огибающие спектра для остаточного *LPC* оцениваются тем же способом, что и в кодере *MBE*, за исключением того, что используется только двухбитовое решение *V/UV* на фрейм. Огибающая спектра для речевого сегмента, затем — вектор, квантованный со взвешенной мерой искажения. Для неречевого сегмента выполняется поиск по замкнутому контуру для кодирования вектора возбуждения.

А.2 Нормализация

А.2.1 Описание инструмента

Процесс нормализации составлен из трех операций, а именно, анализа *LPC*, квантования параметров *LSP* и обратной фильтрации.

А.2.2 Процесс нормализации

А.2.2.1 Анализ *LPC*

Для каждого фрейма вычисляются коэффициенты *LPC* 10-го порядка, используя входные сигналы Хемминга, обработанные методом окна, методом автокорреляции.

А.2.2.2 Квантование *LSP*

Используется тот же самый квантователь *LSP*, как в узкополосных *CELP*.

Коэффициенты *LPC* сначала преобразованы в параметры пары *Line Spectral Pair* (*LSP*). Параметры *LSP* затем квантуются *Vector Quantization* (*VQ*). В случае базового уровня есть два метода квантования *LSPs*, как описано в разделе декодирования; двухступенчатый *VQ* без межфреймового предсказания и комбинация *VQ* и *VQ* с межфреймовым прогнозированием. В процессе кодирования оба метода используются для квантования *LSPs*, и один из них выбирается путем сравнения ошибок квантования. Ошибка квантования вычисляется как взвешенное евклидово расстояние.

В случае уровня расширения 10-мерный векторный квантователь, у которого есть книга шифров на 8 битов, добавляется к основанию текущей схемы квантователя *LSP* кода на 2,0 Кбит/с. Битовая скорость *LSPs* увеличена с 18 бит/20 мс до 26 бит/20 мс.

Процесс кодирования базового уровня следующий.

Коэффициенты взвешивания (*w* []) равны:

$$\frac{1}{lsp[0]} + \frac{1}{lsp[0] - lsp[0]} (i = 0)$$

$$w[i] = \frac{1}{lsp[i] - lsp[i-1]} + \frac{1}{lsp[i+1] - lsp[i]} \left(0 < i < N_p - 1\right)$$

$$\frac{1}{lsp\left[N_p - 1\right] - lsp\left[N_p - 2\right]} + \frac{1}{1{,}0 - lsp\left[N_p - 1\right]} \left(i = N_p - 1\right),$$

где *Np* — порядок анализа *LP*,

lsp [] *s* — конвертированные *LSPs*.

```
w_fact = 1.;
for (i = 0; i < 4; i++) w[i] *= w_fact;
for (i = 4; i < 8; i++) {
  w_fact *= .694;
  w[i] *= w_fact;
}
for (i = 8; i < 10; i++) {
  w_fact *= .510;
  w[i] *= w_fact;
}
```

Квантизатор первой стадии — тот же для каждого метода квантования. *LSPs* квантованы при использовании векторного квантователя, и соответствующий индекс сохранен в *LSP*1. Чтобы выполнить отсроченное решение,

множественные индексы сохранены как кандидаты на вторую стадию. Ошибка квантования в первой стадии *err*1 [] дается формулой

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

где *n* — число вектора разбиения,
m — является индексом вектора разбиения кандидата,
sp — порядок стартового *LSP n*-го вектора разбиения,
dim — является размерностью *n*-го вектора разбиения.

Таблица А.1 — Порядок запуска и размерность вектора *LSP* первой стадии

Вектор разбиения: *n*	Порядок стартового *LSP*: *sp*	Размерность *n*-го вектора разбиения. *dim*
0	0	10

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

VQ без межфреймового предсказания:

$$err2\_total = err2[0] + err2[1],$$

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

$$lsp\_res[sp+i] = lsp[sp+i] - lsp\_first[sp+i],$$

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

VQ с межфреймовым предсказанием:

$$err2\_total = err2[0] + err2[1],$$

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

$$lsp\_pres[sp+i] = lsp[sp+i] -$$
$$\{(1 - ratio\_predict) \cdot lsp\_first[sp+i] + ratio\_predict \cdot lsp\_previous[sp+i]\},$$

где *lsp_first* [] — является квантованным вектором *LSP* первой стадии,
n — число вектора разбиения,
m — является индексом вектора разбиения кандидата,
sp — порядок стартового *LSP n*-го вектора разбиения,
dim — является размерностью *n*-го вектора разбиения,
ratio_predict = 0,7.

Таблица А.2 — Порядок запуска и размерность вектора *LSP* второй стадии

Вектор разбиения: *n*	Порядок стартового *LSP*: *sp*	Размерность *n*-го вектора разбиения: *dim*
0	0	5
1	5	5

Квантованные *LSPs lsp_current* [] стабилизируются, чтобы гарантировать стабильность фильтра синтеза *LPC*, который получен из квантованных *LSPs*. Квантованные *LSPs* упорядочены в порядке возрастания, имея минимальное расстояние между смежными коэффициентами.

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

где *min_gap* = 4,0/256,0

После процесса кодирования *LSP* текущие *LSPs* должны быть сохранены в памяти, так как они используются для предсказания в следующем фрейме.

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

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

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

Третья стадия для уровня расширения (4,0 и 3,7 Кбит/с) имеет 10-мерную структуру *VQ*. Ошибка между версией квантования базового уровня и оригинальной версией квантуется, и соответствующий индекс сохраняется в *LSP*5.

После квантования *LSP*5 уровня расширения *qLsp* [] снова стабилизируются.

```
for (i = 0; i <
2; i++)
{
  if (qLsp[i + 1] - qLsp[i] <
0)
  {
    tmp = qLsp[i +
1];
    qLsp[i + 1] =
qLsp[i];
    qLsp[i] = tmp;
  }
  if (qLsp[i + 1] - qLsp[i] < THRSLD_L)
  {
    qLsp[i + 1] = qLsp[i] + THRSLD_L;
  }
}
for (i = 2; i < 6; i++)
{
  if (qLsp[i + 1] - qLsp[i] < THRSLD_M)
  {
    tmp = (qLsp[i + 1] + qLsp[i]) / 2,0;
    qLsp[i + 1] = tmp + THRSLD_M / 2,0;
    qLsp[i] = tmp – THRSLD_M/ 2,0;
  }
```

```
    }
    for (i = 6; i < LPCORDER - 1; i++)
    {
      if (qLsp[i + 1] - qLsp[i] < 0)
      {
        tmp = qLsp[i + 1];
        qLsp[i + 1] = qLsp[i];
        qLsp[i] = tmp;
      }
      if (qLsp[i + 1] = qLsp[i] < THRSLD_H)
      {
        qLsp[i] = qLsp[i + 1] - THRSLD_H;
      }
    }
```

где *THRSLD_L* = 0,020, *THRSLD_M* = 0,020 и *THRSLD_H* = 0,020.

Таблица А.3 — Конфигурация многоступенчатого *LSP VQ*

1-й этап	10 *LSP VQ*	5 битов
2-й этап	(5 + 5) *LSP VQ*	(7 + 5 + 1) битов
3-й этап	10 *LSP VQ*	8 битов

А.2.2.3 Фильтр инверсии *LPC*

LSPs преобразованы к альфа-параметрам, чтобы сформировать фильтр инверсии *LPC* в прямой форме. Остаточные сигналы *LPC* затем вычисляются путем инверсной фильтрации входного сигнала. Только квантованный *LPC* текущего фрейма используется без какой-либо интерполяции для инверсной фильтрации, чтобы вычислить остаточный сигнал *LPC*. Остаточный сигнал тогда является 256-точечной оконной обработкой Хемминга, чтобы вычислить спектр мощности. Функция преобразования инверсного фильтра *LPC* такова

$$A(z) = \sum_{n=0}^{P} \alpha_n z^{-n},$$

где $P = 10$ и $\alpha_0 = 1$. Коэффициенты *LPC* α_n остаются неизменными во время вычисления остаточных отсчетов длины одного фрейма (256 отсчетов).

А.3 Оценка шага

А.3.1 Описание инструмента

Чтобы получить первую оценку значения задержки шага, значений автокорреляции *LPC*, вычисляются остаточные сигналы. Основываясь на значениях задержки, которые дают пики автокорреляции, оценивается шаг разомкнутого цикла. Отслеживание шага выполняется в процессе оценки шага так, чтобы оцениваемый шаг стал более достоверным.

А.3.2 Процесс оценки шага

В кодере режима малой задержки трекинг (отслеживание) шага проводится, используя только текущие и прошлые фреймы, чтобы сохранить задержку кодера 26 мс. Когда используется режим нормальной задержки, отслеживание шага использует один предстоящий фрейм, и задержка кодера становится 46 мс.

А.3.3 Трекинг шага

Для режима малой задержки *HVXC* алгоритм прослеживания шага не использует значение шага будущего (предвидение) фрейма. Алгоритм прослеживания шага работает, основываясь на надежном шаге "*rblPch*", решении *V/UV* предыдущего фрейма "*prevVUV*", и параметрах прошлых/текущих шагов.

Основная операция состоит в следующем.

Когда *prevVUV* = 0 и *rblPch*! = 0, шаг прослеживается на основе *rblPch*. Однако у значения шага предыдущего фрейма более высокий приоритет.

Когда *prevVUV* = 0 и *rblPch*! = 0, шаг прослеживается на основе *rblPch*.

Когда *prevVUV* = 0 и *rblPch* = 0, трекинг проводится на основе шага предыдущего фрейма.

Когда *prevVUV* = 0 и *rblPch* = 0, просто используется параметр текущего шага.

prevVUV = 0 представляет речевой, и *prevVUV* = 0 представляет неречевой статус соответственно.

В соответствии с этой стратегией трекинг шага выполняется без какого-либо заглядывания вперед. Исходный текст программы трекинга шага показан ниже.

```
typedef struct
{
```

```
  Float pitch;      /* pitch */
  Float prob;       /* 1st peak devided by 2nd peak of autocorrelation */
  Float r0r;        /* 1st peak of autocorrelation – modified */
  Float rawROr;     /* 1st peak of autocorrelation – raw */
  Float rawPitch;   /* pitch with no tracking */
}
NgbPrm;
static int NearPitch(
float p0,
float p1,
float ratio)
{
  return((p0 * (1,0 – ratio) < p1) && (p1 < p0 * (1,0 + ratio)));
}
static float TrackingPitch(
NgbPrm *crntPrm, /* current parameter */
NgbPrm *prevPrm, /* previous parameter */
int *scanlimit, /* Number of autocorrelation peaks */
float *ac, /* Autocorrelation */
int peakPos[PEAKMAX], /* Position of autocorrelation peaks */
int prevVUV) /* V/UV of ptrevious frame */
  {
  float kimete;
  float pitch;
  int i;
  static float prevRawp= 0.0;
  int st0, st1, st2;
  float stdPch;
  static float rblPch = 0.0;
  static float prevRblPch = 0.0;
  rblPch = global_pitch;
  if (prevVUV != 0 && rblPch != 0.0)
  {
st0 = Ambiguous(prevPrm->pitch, rblPch,
0, 11);
st1 = Ambiguous(crntPrm->pitch, rblPch,
0,11);
  if(!(st0 || st1))
{
  if (NearPitch(crntPrm->pitch, prevPrm-
>pitch, 0,2))
  pitch = crntPrm->pitch;
    else if (NearPitch(crntPrm->pitch, rblPch, 0,2))
  pitch = crntPrm->pitch;
    else if (NearPitch(prevPrm->pitch, rblPch, 0,2))
    {
  if (crntPrm->r0r > prevPrm->r0r && crntPrm->prob >
prevPrm->prob)
    pitch = crntPrm->pitch;
  else
    pitch = prevPrm->pitch;
    }
    else
  pitch = GetStdPch2Elms(crntPrm, prevPrm, scanlimit,
peakPos, ac);
}
else if (!st0)
{
  if (NearPitch(prevPrm->pitch, crntPrm->pitch,
0,2))
```

```
    pitch = crntPrm->pitch;
      else if ((gpMax * 1,2 > crntPrm->pitch) &&
                    NearPitch(prevPrm->rawPitch, crntPrm->pitch,
0,2))
    pitch = crntPrm->pitch;
      else
    pitch = prevPrm->pitch;
}
else if (!st1)
{
      if ((crntPrm->rawPitch != crntPrm->pitch) &&
    NearPitch(crntPrm->rawPitch, prevPrm->rawPitch,
0,2))
    pitch = crntPrm->rawPitch;
      else
    pitch = crntPrm->pitch;
}
else
{
    if (NearPitch(prevPrm->pitch, crntPrm->pitch,
0,2))
    pitch = crntPrm->pitch;
      else
    pitch = rblPch;
}
    }
    else if (prevVUV == 0 && rblPch != 0.0)
    {
st1 = Ambiguous(crntPrm->pitch, rblPch,
0,11);
if (!st1)
    pitch = crntPrm->pitch;
else
    pitch = rblPch;
    }
    else if (prevVUV != 0 && rblPch == 0.0)
    {
st1 = Ambiguous(crntPrm->pitch, prevPrm->pitch,
0,11);
if (!st1)
    pitch = GetStdPch2Elms(crntPrm, prevPrm, scanlimit, peakPos,
ac);
    else {
      if (prevPrm->r0r < crntPrm->r0r)
    pitch = crntPrm->pitch;
      else
    pitch = prevPrm->pitch;
    }
      }
      else
    pitch = crntPrm->pitch;
      crntPrm->pitch = pitch;
      prevRblPch = rblPch;
      prevRawp = pitch;
      return(pitch);
    }
```

А.4 Извлечение величин гармоник

А.4.1 Описание инструмента

Извлечение величин гармоник состоит из двух шагов: поиск шага и оценка огибающей спектра. Операция для каждого шага описана ниже.

А.4.2 Процесс извлечения величин гармоник

А.4.2.1 Поиск шага

Используя целочисленную задержку шага разомкнутого цикла, оценивается дробное значение задержки шага. Размер шага фракции равен 0,25. Это выполненяется путем минимизации ошибки между синтезируемым спектром и исходным спектром. Здесь одновременно оцениваются значение задержки шага и величины гармоник спектра. Оцениваемое значение задержки шага, *pch*, передается в декодер как *Pitch*

$$Pitch = (int)\,(pch - 20{,}0).$$

А.4.2.2 Оценка огибающей спектра

Чтобы получить исходный спектр к остаточным сигналам *LPC* применяется *DFT* 256 точек. Используя исходный спектр, производится определение огибающей спектра. Огибающая спектра — это ряд величин спектра, оцениваемых для каждой гармоники. Эта оценка величин выполняется вычислением оптимальной амплитуды A_m, используя предопределенную базисную функцию $E(j)$ и исходный спектр $X(j)$. Пусть a_m и b_m будут индексами коэффициента *DTS* нижней и верхней границы *m*-й полосы соответственно, охватывая полосу *m*-й гармоники. Ошибка оценки амплитуды ε_m определяется как

$$\varepsilon_m = \sum_{j=a_m}^{b_m} \left(|X(j)| - |A_m||E(j)|\right)^2.$$

Решение

$$\frac{\partial \varepsilon_m}{\partial |A_m|} = 0$$

дает

$$|A_m| = \frac{\sum_{j=a_m}^{b_m} |X(j)||E(j)|}{\sum_{j=a_m}^{b_m} |E(j)|^2}.$$

Это значение определено как спектральная величина. Базисная функция $E(j)$ может быть получена *DFT* окна Хемминга 256 точек.

А.5 Перцепционное взвешивание

А.5.1 Описание инструмента

Вычисляется частотная характеристика фильтра перцепционного взвешивания, предназначенного для использования взвешенного векторного квантования огибающей спектра гармоник. Здесь фильтр перцепционного взвешивания получен из линейных прогнозирующих коэффициентов α_n. Функция преобразования фильтра перцепционного взвешивания такова

$$w(z) = \frac{\sum_{n=0}^{p} \alpha_n A^n z^{-n}}{\sum_{n=0}^{p} \alpha_n B^n z^{-n}},$$

где могли использоваться $A = 0{,}9$; $B = 0{,}4$. В этом инструменте также вычисляется частотная характеристика фильтра синтеза *LPC* – $h(z)$ так, чтобы ее можно было включить

$$h(z) = \frac{1}{\sum_{n=0}^{p} \alpha_n z^{-n}}.$$

В этом инструменте вычислена частотная характеристика $w(z)h(z)$ и выведена как массив **per_weight*, который может быть использован как диагональные компоненты матриц взвешивания *WH*.

А.6 Кодер гармоник *VQ*

А.6.1 Описание инструмента

Процесс кодирования гармоник *VQ* состоит из двух шагов: преобразование размерности и квантование векторов огибающей спектра. Операции каждого шага приведены ниже.

А.6.2 Процесс кодирования

А.6.2.1 Конвертер размерности

Количество точек, которые составляют огибающую спектра, изменяется в зависимости от величин шага, так как огибающая спектра является набором оценок величин каждой гармоники. Число гармоник равно приблизительно от 9 до 70. Чтобы векторно квантовать огибающую спектра, кодер должен преобразовать их в постоянное число для *VQ* фиксированной размерности. Для преобразования частоты взятия выборки используется интерполяция с ограниченной полосой, чтобы получить спектральные векторы фиксированной размерности. Число точек, которые представляют форму огибающей спектра, должно быть изменено, не изменяя ее форму. С этой целью

используется конвертер размерности для огибающей спектра, состоящей из комбинации низкочастотного фильтра и линейного интерполятора 1-го порядка. Для 8-кратной передискретизации первой стадии используется низкополосный фильтр *FIR* с 7 наборами коэффициентов; каждый набор состоит из 8 коэффициентов. 7 наборов коэффициентов фильтра получены группировкой каждых 8 коэффициентов из *windowed sinc, coef* [*i*], со смещениями от 1 до 7, где

$$coef[i] = \frac{\sin \pi (i - 32)/8}{\pi (i - 32)/8}(0{,}5 - 0{,}5\cos 2\pi i / 64) \quad 0 \le i \le 64.$$

Эта фильтрация *FIR* позволяет вдесятеро уменьшить вычисление, в котором вычислены только точки, используемые в следующей стадии. Они являются левыми и правыми соседями конечного выхода конвертера размерности.

Во второй стадии передискретизации применяется линейная интерполяция 1-го порядка, чтобы получить необходимые точки выхода. Таким образом, мы получаем спектральные векторы фиксированной размерности (= 44).

А.6.2.2 Квантование вектора

Спектральный вектор фиксированной размерности (= 44) затем квантуется. В базовом уровне используется двухступенчатая схема векторного квантования для спектральной формы совместно со скалярным квантователем для усиления. Мера взвешенного искажения *D* используется для поиска в книге шифров как формы, так и усиления.

$$D = ||WH(x - g(s_1 + s_2))||^2,$$

где x — исходный вектор,

s_1 — выход книги шифров *Spectral Envelope* (*SE*) *shape*1,

s_2 — выход *SE shape*2 книги шифров,

g — выход книги шифров усиления *SE*.

Соответствующие индексы обозначены как *SE_shape*1, *SE_shape*2 и *SE_gain* соответственно. Размерность книг шифров формы фиксируется (= 44). Диагональные компоненты матриц *H* и *W* являются величинами частотной характеристики фильтра синтеза *LPC* и перцепционного фильтра взвешивания соответственно. Для уровня расширения к основанию квантизатора базового уровня добавлены дополнительные векторные квантизаторы. Режим 2,0 Кбит/с использует только квантизаторы базового уровня.

Для режима 4,0 Кбит/с величины квантованных гармоник с фиксированной размерностью (= 44) в базовом уровне сначала преобразуются к размерности исходного вектора гармоник, который изменяется в зависимости от величины шага. Вычисляется различие между квантованным (с восстановленной размерностью вектором гармоник) и исходным вектором гармоник. Это различие затем квантуется схемой *VQ* с разбиением, составленной из четырех векторных квантователей в уровне расширения. Соответствующие индексы — *SE_shape*3, *SE_shape*4, *SE_shape*5 и *SE_shape*6.

*Индекс SE_shape*6 не используется, когда выбран режим 3,7 Кбит/с.

А.7 Решение *V/UV*

А.7.1 Описание инструмента

Решение *V/UV* принимается каждым фреймом 20 мс. Решение принимается на основе: подобия формы синтезируемого спектра и исходного спектра, мощности сигнала, максимальной автокорреляции остаточных сигналов *LPC*, нормализованных остаточной мощностью сигнала и числом переходов через нулевого.

А.7.2 Процесс кодирования

Решение *V/UV* составлено из трех различных режимов, т. е. неречевого, смешанного речевого и полностью речевого.

Чтобы отослать информацию этих трех режимов, используются два бита для *V/UV*. Во-первых, нужно решить, неречевой ли текущий фрейм. Когда это решение «речевой», тогда оценивается сила звучания, исходя из значения нормализованного максимального пика автокорреляции остаточного сигнала *LPC*. Обозначим значение *r*0*r*.

Ниже показано правило принятия решения:

Когда первым решением является речевой	*V/UV* = 0	Неречевой
Когда первым решением является неречевой	*V/UV* = 1 для *r*0*r* < *TH*2	Смешанный речевой
	V/UV = 2 для *TH*2 <= *r*0*r* < *TH*1	1 Смешанный речевой
	V/UV = 3 для *TH*1 <= *r*0*r*	2 Полностью речевой

А.8 Кодер временного домена

А.8.1 Описание инструмента

Когда речевой сегмент является неголосовым, используется алгоритм *Vector Excitation Coding (VXC)*. Работа *VXC* описана ниже.

А.8.2 Процесс кодирования

Сначала выполняется анализ *LPC*, а затем коэффициенты *LPC* (α) преобразуются в параметры *LSP* тем же самым способом, как в речевом случае. Параметры *LSP* квантуются, и квантованные *LSPs* преобразуются в коэффициенты *LPC* ($\tilde{\alpha}$).

Перцепционно взвешенный фильтр синтеза *LPC* - - *H*(*z*) выражен как

$$H(z) = A(z)\ W(z),$$

где *A*(*z*) — функция преобразования фильтра синтеза *LPC*,

W(*z*) — является перцепционно взвешенным фильтром, полученным из коэффициентов *LPC*.

Пусть x_w (*n*) будет перцепционно взвешенным входным сигналом. Вычитая отклик на нулевой ввод *z*(*n*) из x_w(*n*), получаем опорный сигнал *r*(*n*) для процедуры анализа через синтез *VXC*. Векторы оптимальной формы и усиления разыскиваются с использованием меры искажения *E*

$$E = \sum_{n=0}^{N-1}\left(r(n) - g \times syn(n)\right)^2,$$

где *syn*(*n*) является реакцией нулевого состояния *H*(*z*), управляемого вводом возбуждения только вектором формы *s*(*n*), который является выходом книги шифров *VX_shape*. Здесь *g* — это усиление, которое является выводом книги шифров *VX_gain*. *N* является векторной размерностью книги шифров *VX_shape*.

Процесс поиска книги шифров для *VXC* состоит из двух шагов.

1. Поиск *s*(*n*), которое максимизирует

$$E_s = \frac{\sum_{n=0}^{N-1} r(n) \times syn(n)}{\sqrt{\sum_{m=0}^{N-1} syn(m)^2}}.$$

2. Поиск *g*, которое минимизирует

$$E_g = (g_{ref} - g)^2,$$

где

$$g_{ref} = \frac{\sum_{n=0}^{N-1} r(n) \times syn(n)}{\sum_{m=0}^{N-1} syn(m)^2}.$$

Ошибка квантователя *e*(*n*) подсчитывается как

$$e(n) = r(n) - g \times syn(n).$$

Когда битовая скорость равна 4 Кбит/с, используется еще одна стадия для квантования неголосовых сегментов, и *e*(*n*) используется как опорный ввод для *VQ* второй стадии.

Работа *VQ* второй стадии такая же, как работа *VQ* первой стадии.

Кодер 2,0 Кбит/с использует книги шифров формы на 6 битов и усиления на 4 бита двоичного для неречевого возбуждения каждые 10 мс. Соответствующие индексы — *VX_shape* [*i*], *VX_gain* [*i*] (*i* = 0,1). Схема 4,0 Кбит/с добавляет книги шифров для формы на 5 битов и усиления на 3 бита для каждых 5 мс внизу текущего квантователя. Соответствующими индексами являются — *VX_shape*2 [*i*], *VX_gain*2 [*i*] (*i* = 0,1,2,3). Режим 3,7 Кбит/с может использовать ту же самую процедуру кодирования, как режим 4,0 Кбит/с, хотя *VX_shape*2 [3] и *VX_gain*2 [3] не используются в декодере режима 3,7 Кбит/с.

Таблица А.4 — Конфигурация книг шифров *VXC*

1-й этап	(80-мерная 6-битовая форма + 4-битовое усиление) × 2
2-й этап	(40-мерная 5-битовая форма + 3-битовое усиление) × 4

А.9 Кодер варьируемой скорости

Здесь описан инструмент для кодирования варьируемой скорости с ядром *HVXC*. Данный инструмент позволяет *HVXC* работать при изменяющихся битовых скоростях. Главная часть алгоритма составлена из «обнаружения интервала фонового шума», где передаются только биты режима во время «режима фонового шума», и неречевой фрейм вставляется с определенным промежутком времени, чтобы послать параметры для генерации фонового шума.

В алгоритме кодирования трекер (следящая система) минимального уровня имеет временный минимальный уровень, чтобы корректировать пороговое значение, по которому принимается решение о том, является ли входной сегмент речевым.

Трекинг минимального уровня

Параметры определены следующим образом:

lev: *r.m.s.* речевого фрейма;

vCont: число непрерывных голосовых фреймов;

cdLev: значение кандидата минимального уровня;

prevLev: *r.m.s.* предыдущего фрейма;

gmlSetState: число фреймов, в которых установлено значение кандидата;

gmlResetState: число фреймов, в которых значение кандидата не установлено после установки минимального уровня;

gml: минимальный уровень.

Минимальный уровень отслеживается согласно алгоритму, который показан ниже.

```
if (vCont > 4)
{
  cdLev = 0.0;
  gmlSetState = 0;
  gmlResetState
++;
}
else if (lev <
MIN_GML)
{
  *gml = MIN_GML;
  gmlResetState = 0;
  gmlSetState =
  0;
}
else if (*gml >
lev)
{
  *gml = lev;
  gmlResetState = 0;
  gmlSetState =
0;
}
else if ((lev<500.0 && cdLev * 0,70<lev && lev<cdLev * 1,30)
|| lev<100,0)
{
  if (gmlSetState > 6)
  {
    *gml = lev;
    gmlResetState
  = 0;
    gmlSetState =
  0;
     }
     else
     {
     cdLev = lev;
     gmlResetState
  = 0;
     gmlSetState++;
  }
}
else if ((lev<500.0 && prevLev * 0,70<lev && lev<prevLev *
1,30) || lev<100,0)
{
  cdLev = lev;
  c = 0;
  gmlSetStat
e++;
}
```

```
else if (gmlResetState
> 40)
{
  *gml = MIN_GML;
  gmlResetState = 0;
  gmlSetState =
0
}
else
{
  cdLev = 0.0;
  gmlSetState = 0;
  gmlResetState++
}
```

Как показано выше, минимальный уровень удерживается в течение соответствующего промежутка времени и обновляется. Минимальный уровень всегда устанавливается выше, чем предопределенное значение *MIN_GML*.

Обнаружение фонового шума, основанное на минимальном уровне

Справочный уровень *refLev* вычисляется как

$$refLev = A \times max(lev, refLev) + (1{,}0 - A) \times min(lev, refLev). \quad (A.1)$$

Обычно A устанавливается в 0,75. Обнаружение фонового шума выполняется, используя *refLevel*, полученное из уравнения (A.1).

Для фреймов, где принимается решение «речевой»:

```
if (refLev < B *gml && contV < 2) {
idVUV = 1;
}
```

Для фреймов, где принимается решение «неречевой»:

```
if (refLev < B *gml) {/* condition-1 */
  if (bgnCnt < 3) {
   bgnCnt++;
}
else {
  if (bgnIntvl < 8) {
   idVUV=1;
   bgnIntvl++;
}
else {
  bgnIntvl=0;
 }
}
}
else {
   bgnCnt=0;
}
```

где *B* — константа. В этом случае устанавливаем *B* = 2,0.

Каждый параметр определен ниже.

countV: число последовательных речевых фреймов;

bgnCnt: число фреймов, которое удовлетворяет условию 1;

bgnIntvl: число фреймов, где объявлен режим «фоновый шум»;

idVUV: параметр, результат которого решение *V/UV*, определенный как

$$idVUV = \begin{cases} 0 & \text{Невокализованная речь} \\ 1 & \text{Интервал фонового шума} \\ 2 & \text{Смешанная вокализованная речь} \\ 3 & \text{Вокализованная речь} \end{cases}$$

Если текущий фрейм объявлен как речевой, производится проверка, был ли предыдущий фрейм тоже речевой. Если режим предыдущего фрейма речевой, режим фонового шума не выбирается; иначе выбирается режим фонового шума.

Если текущий фрейм, заявлен как неречевой, режим фонового шума выбирается только после того, как условие 1 удовлетворено для четырех последовательных фреймов. Когда режим фонового шума выбран для *N* последовательных фреймов, режим последнего фрейма заменяется режимом «неречевой», чтобы передать речевые параметры, которые представляют характеристики времени изменения фонового шума.

В этот момент *N* устанавливается в 9.

Кодирование с переменной скоростью

Используя описанный выше метод обнаружения фонового шума, выполненяется кодирование с варьируемой скоростью на базе *HVXC* с фиксированной битовой скоростью 2,0 Кбит/с.

Таблица А.5 — Распределение разрядов закодированных параметров для режима варьируемой битовой скорости

Режим (*dVUV*)	Фоновый шум (1)	*UV*(0)	*MV*(2), *V*(3)
V/UV *LSP* Возбуждение	2 бит/20 мс 0 бит/20 мс 0 бит/20 мс	2 бит/20 мс 18 бит/20 мс 8 бит/20 мс (только усиление)	2 бит/20 мс 18 бит/20 мс 20 бит/20 мс
Совокупно	2 бит/20 мс 0,1 Кбит/с	28 бит/20 мс 1,4 Кбит/с	40 бит/20 мс 2,0 Кбит/с

Если режим текущего фрейма или предыдущего фрейма — «фоновый шум», другой режим квантования *LSP* запрещается в кодере, потому что параметры *LSP* не передаются во время режима «фоновый шум» и межфреймовое кодирование невозможно.

А.10 Расширение кодера с варьируемой скоростью *HVXC*

В А.9 описана операция кодера режима варьируемой скорости максимом на 2,0 Кбит/с. Здесь описан один пример реализации кодера режима варьируемой скорости *HVXC* максимом на 4,0 Кбит/с. По существу, может использоваться любой вид алгоритма выбора решения «фоновый шум/неречевой». Изменение уровня сигнала и огибающей спектра используются, чтобы обнаружить интервал фонового шума из неречевых фреймов. Во время шумового интервала предполагается, что уровень сигнала и форма огибающей спектра устойчивы. Характеристика зарегистрированной величины в квадрате в низкочастотном диапазоне огибающей спектра подсчитывается из коэффициентов кепстра *LPC*. Характеристика зарегистрированной величины в квадрате текущего фрейма сравнивается со средним зарегистрированной величины в квадрате нескольких предыдущих фреймов. Если различие мало, предполагается, что сигнал устойчив. Когда эти параметры показывают что состояние сигнала достаточно стабильно, фрейм классифицируется как «интервал фонового шума». Если характеристика сигнала изменяется в определенном диапазоне, предполагается, что характеристика фонового шума изменилась, и передается фрейм обновления шума. Если характеристика сигнала изменилась больше, чем заданный диапазон, то предполагается, что сигнал является невокализованной речью.

А.10.1 Определения

stFlag: Флажок стабильности сигнала
bgnCnt: Счетчик фонового шума
BgnIntvl: Счетчик интервала фонового шума
UpdateFlag: Флажок обновления фонового шума

А.10.2 Вычисление *RMC*

Сначала вычисляется *RMC* входного сигнала *s* [], чтобы получить минимальный уровень сигнала (*min_rmc*). Если величина *RMC* меньше, чем предопределенное значение, которое является наименьшим возможным речевым уровнем, и отклонения значения *RMC* нескольких фреймов находятся в определенном диапазоне, то минимальный уровень обновляется, используя обнаруженное значение *RMC* и текущее min_*rmc*. Текущее значение *RMC* "*rmc*" затем делится на текущее min_*rmc*, чтобы получить величину "*ratio*".

$$ratio = \frac{rmc}{\min\_rmc}.$$

Это значение используется для обнаружения стабильности сигнала, как описано в А.10.4.

А.10.3 Сравнение спектра

Огибающая спектра рассчитывается, используя коэффициенты линейного предсказания (*LP*). Коэффициенты *LP* преобразуются в параметры кепстра C_L []. Исходя из параметров кепстра характеристика зарегистрированной величины в квадрате $ln|H_L(e^{j\Omega})|^2$ вычисляется как

$$\ln\left|H_L\left(e^{j\Omega}\right)\right|^2 = 2\cdot\sum_{m=0}^{M} C_L(m)\cos(\Omega m).$$

Средний уровень каждого диапазона частот в *n*-м фрейме вычисляется как

$$\log Amp(n,i) = \frac{1}{w}\int_{i}^{\Omega_{i+1}} In\left|H_L\left(e^{j\Omega}\right)\right|^2 d\Omega = \frac{1}{w}\left[2\sum_{m=1}^{M}\frac{1}{m}C_L(m)\sin(\Omega m)\right]_{\Omega_i}^{\Omega_{i+1}} \quad i = 0...3$$

Здесь ω ширина полосы (= 500 Гц), и *i* — число полос, где вычислены 4 полосы между 0 и 2 кГц. Исходя из полученных значений зарегистрированной величины в квадрате для последних 4 фреймов рассчитываются усредненные значения для каждой полосы как

$$aveAmp(n,i) = \frac{1}{4}\sum_{j=1}^{4}\log Amp(n-j,i).$$

Используя эти уравнения, различие "*wdif*" между текущим уровнем и усредненным уровнем для каждого диапазона частот *i* вычисляются, как показано ниже

$$wdif = \sqrt{=\frac{1}{4}\sum_{i=0}^{3}\left(\log Amp(n,i) - aveAmp(n,i)\right)^2}.$$

А.10.4 Обнаружение стабильности сигнала

Исходя из комбинации *ratio* и *wdif* рассчитывается сигнал флажка стабильности "*stFlag*", у которого есть следующие три состояния. Естественно, чем меньше эти переменные, тем более стабильно состояние сигнала.

$$stFlag = \begin{cases} 0\ \text{Почти стабильный} \\ 1\ \text{Не стабильный} \\ 2\ \text{Более стабильный} \end{cases}$$

А.10.5 Обнаружение фонового шума

В соответствии с переменной *stFlag* и решением *V/UV* обнаруживается фоновый шум. Блок-схема показана на рисунке А.1. Когда решением *V/UV* является речевой (*VUV* = 2,3), фрейм классифицируется как речевой независимо от значения *stFlag*. Когда в неречевом интервале обнаруживается фоновый шум (*VUV* = 0), *VUV* устанавливается в 1. Когда параметры фонового шума должны быть обновлены, "*UpdateFlag*" устанавливается в 1. Чтобы обеспечить устойчивую работу алгоритма обнаружения фонового шума, введены переменные, счетчик фонового шума (*bgnCnt*) и счетчик интервала фонового шума (*bgnIntvl*). *BGN_CNT* и *BGN_INTVL* — предопределенные константы.

А.10.6 Решение о принятых параметрах

VUV — параметр содержит результат решения *V/UV* и определен как

$$VUV = \begin{cases} 0\ \text{Невокализованная речь} \\ 1\ \text{Интервал фонового шума} \\ 2\ \text{Вокализованная речь 1} \\ 3\ \text{Вокализованная речь 2} \end{cases}$$

Чтобы узнать, является ли фрейм, маркированный как "*VUV* = 1", фреймом обновления шума, введен параметр "*UpdateFlag*". *UpdateFlag* ниже используется, только когда *VUV* = 1.

$$UpdateFlag = \begin{cases} 0\ \text{нефрейм обновления шума} \\ 1\ \text{фрейм обновления шума} \end{cases}$$

В фрейме обновления шума вычисляются параметры усредненного *LSP* и *Celp Gain*.

Необработанные *LSPs* усреднены по последним 3 фреймам

$$aveLsp[n][i] = \frac{1}{3}\sum_{j=0}^{2} Lsp[n-j][i], i = 1, ..., NP,$$

где *aveLsp*[*n*] [] обозначает усредненное *LSP* в *n*-м фрейме и *Lsp*[*n*] [] обозначает необработанный *LSP* в *n*-м фрейме. *NP* представляет собой порядок *LSP*. *aveLsp*[*n*] [] является квантованным и закодированным, где кодирование *LSP* другим режимом запрещено.

Усреднены усиления необработанных *Celp* по последним 4 фреймам (8 подфреймам).

$$aveGain[n] = \frac{1}{8}\sum_{j=1}^{3}\sum_{k=0}^{1} Gain[n-j][k],$$

где *aveGain* [*n*] является усредненным усилением *Celp* в *n*-м фрейме, а *Gain*[*n*] [*K*] — необработанное усиление *Celp* в *n*-м фрейме и *k*-м подфрейме. Усредненное усиление *Celp* квантовано и закодировано тем же самым способом, как обычное усиление *Celp*.

Согласно *VUV*, передаются следующие параметры.

Таблица А.6 — Закодированные параметры для режима варьируемой скорости на 4 Кбит/с

Режим (*VUV*)	Ф (1)		*UV*(0)	*V*(2,3)
	Флагобновления = 0	Флагобновления = 1		
V/UV Флагобновления *LSP* Возбуждение	2 бит/20 мс 1 бит/20 мс 0 бит/20 мс	2 бит/20 мс 1 бит/20 мс 18 бит/20 мс 4 бит/20мс (только усиление)	2 бит/20 мс 0 бит/20 мс 18 бит/20 мс 20 бит/20 мс	2 бит/20 мс 0 бит/20 мс 26 бит/20 мс 52 бит/20 мс
Совокупно	3 бит/20 мс 0,15 Кбит/с	25 бит/20 мс 1,25 Кбит/с	40 бит/20 мс 2,0 Кбит/с	80 бит/20 мс 4,0 Кбит/с

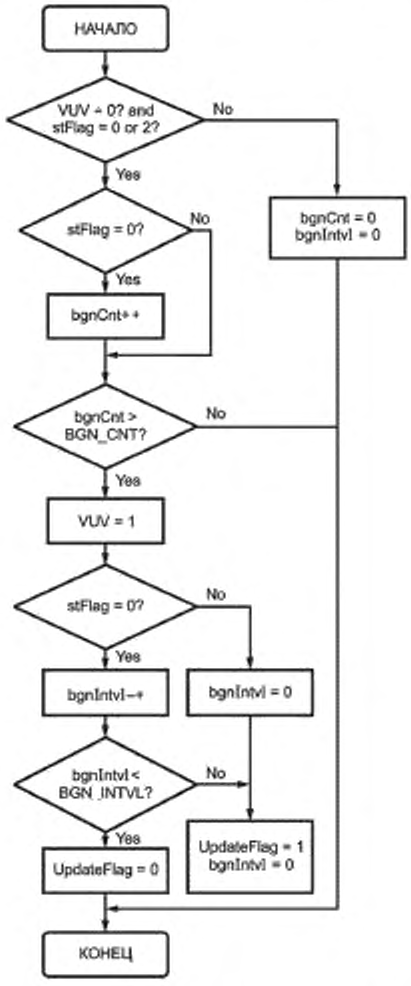


Рисунок А.1 — Блок-схема обнаружения шума

Приложение Б
(справочное)

Инструменты декодера *HVXC*

Б.1 Постфильтр

Б.1.1 Описание инструмента

Основной работой постфильтра является расширение спектральных формант и подавление спектральных точек минимума. Можно использовать единый постфильтр после добавления синтеза вокализованной и невокализованной речи. Альтернативно можно использовать независимые постфильтры для вокализованной и невокализованной речи соответственно. Рекомендуется использование независимых постфильтров для речевых и неречевых сигналов.

В описании ниже каждая из полученных вокализованной и невокализованной речи подается в независимые постфильтры.

Использование постфильтра требуется, однако конфигурация и константы постфильтров, описанные здесь, являются одним из примеров, а не нормой, и они могут быть изменены.

Б.1.2 Определения

$Pf_v(z)$: Передаточная функция фильтра формирования спектра для вокализованной речи.

$Pf_{UV}(z)$: Передаточная функция фильтра формирования спектра для невокализованной речи.

r_{adj}: Коэффициент регулировки усиления для формирования спектра.

$S(n)$: Выход фильтра синтеза *PC H*(*z*).

$S_{pf}(n)$: Выход фильтра формирования спектра *Pfv*(*z*).

$S'_{pf}(n)$: Сформированный по спектру и отрегулированный по усилению выход.

$S'_{pf\_prev}(n)$: $S_{pf}S'_{pf}(n)$, **рассчитанный с использованием** α_n и r_{adj} предыдущего фрейма.

vspeech(*n*): Постфильтрованная вокализованная речь.

uvspeech(*n*): Постфильтрованная невокализованная речь.

Б.1.3 Обработка

Каждая операция постфильтра состоит из трех шагов, т. е. спектрального формирования, регулировки усиления и процесса сглаживания.

Б.1.3.1 Вокализованная речь

Формирование спектра

$$Pf_v(z) = \frac{\sum_{n=0}^{p} \alpha_n \gamma^n z^{-n}}{\sum_{n=0}^{p} \alpha_n \beta^n z^{-n}} \left(1 - \delta z^{-1}\right),$$

где α_n — коэффициенты линейного предсказания, преобразованные из деквантованных и линейно интерполированных *LSPs*, которые обновляются каждые 2,5 мс. $p = 10$, $\gamma = 0{,}5$, $\beta = 0{,}8$, $\delta = -0{,}15 \cdot \alpha_1$, где значение δ ограничено диапазоном $0 \le \delta \le 0{,}5$. Когда выбран режим декодирования с низкой задержкой, интервал фрейма декодера сдвигается на 2,5 мс, и интерполяция *LSPs* выполняется для первого 17,5 мс интервала фрейма декодирования, а последние *LSP*s используются для последних 2,5 мс без интерполяции. Вывод с фильта синтеза *LPC* — *s*(*n*) сначала подается на фильтр спектрального формирования $Pf_v(z)$.

Регулировка усиления

Вывод фильтра спектрального формирования $S_{pf}(n)$ затем регулируется по усилению так, чтобы усиление фрейма ввода и вывода спектрального формирования было неизменно. Регулировка усиления выполняется однажды за каждые 160 фреймов выборки, в то время как *LSPs* обновляются каждые 2,5 мс. Фактор регулировки усиления r_{adj} вычисляется следующим образом

$$r_{adj} = \sqrt{\frac{\sum_{n=0}^{159} \{s(n)\}^2}{\sum_{n=0}^{159} \{s_{Pf}(n)\}^2}}.$$

Сформированный по спектру и отрегулированный по усилению выход $S'_{pf}(n)$ получается как

$$S'_{Pf} = r_{adj} \cdot s_{Pf}(n) \quad (0 \le n \le 159)$$

Процесс сглаживания

Отрегулированный по усилению выход $S'_{Pf}(n)$ затем сглаживается, чтобы избежать нарушения непрерывности из-за изменения параметра в начале каждого фрейма. Постфильтрованный речевой выход *vspeech* (*n*), получают следующим образом

$$vspeech(n) = 1 - \frac{n}{20} S'_{Pf\_prev}(n) + \frac{n}{20} S'_{Pf}(n) \quad (0 \le n \le 19)$$

$$S'_{Pf}(n) \quad (20 \le n \le 159),$$

где $S'_{Pf\_prev}(n)$ — это $s'_{Pf}(n)$, вычисленный с использованием α_n и r_{adj} предыдущего фрейма.

Б.1.3.2 Невокализованная речь

Спектральное формирование

Аналогично здесь дается $Pf_{uv}(z)$ как функция преобразования фильтра спектрального формирования для невокализованной речи

$$Pf_{uv}(z) = \frac{\sum_{n=0}^{p} \alpha_n \gamma^n z^{-n}}{\sum_{n=0}^{p} \alpha_n \beta^n z^{-n}} \left(1 - \delta z^{-1}\right),$$

где α_n — линейные предиктивные коэффициенты, преобразованные из деквантованных *LSPs,* которые обновляются каждые 20 мс.

Когда выбран режим нормального декодирования, коэффициенты *LSP* обновляются в середине интервала фрейма декодирования. Если выбран режим декодирования с малой задержкой, интервал фрейма декодирования сдвигается на 2,5 мс и обновление *LSP* происходит в точке 7,5 мс с начала интервала декодирования на рисунке 1. $p = 10$, $\gamma = 0{,}5$, $\beta = 0{,}8$, $\delta = 0{,}1$. Регулировка усиления и процесс сглаживания — те же самые, как в речевой части, описанной выше, и производят постфильтрованную невокализованную речь *uvspeech (n).*

Выход каждого из постфильтров, *vspeech (n)* и *uvspeech (n),* добавляется, чтобы генерировать выход постфильтрованной речи.

Б.2 Постобработка

Б.2.1 Описание инструмента

Выход постфильтра подается в подсистему постобработки. Постобработка составлена из трех фильтров: это фильтр нижних частот, высокочастотный фильтр акцента и фильтр верхних частот. Фильтр верхних частот используется, чтобы удалить ненужные низкочастотные компоненты, высокочастотный фильтр акцента используется, чтобы увеличить яркость речи, и фильтр нижних частот используется, чтобы удалить ненужные высокочастотные компоненты. Конфигурации фильтров и описанные здесь константы являются одним из примеров и не нормативны, они могут быть изменены.

Б.2.2 Определения

HPF(*z*): Функция преобразования фильтра нижних частот.

Emp(*z*): Функция преобразования высокочастотного фильтра акцента.

LPF(*z*): Функция преобразования фильтра верхних частот.

Б.2.3 Обработка

Три фильтра применены к выходу постфильтра.

Фильтр нижних частот:

$$HPF(z) = G_{inv} \frac{1 + A_1 z^{-1} + B_1 z^{-2}}{1 + C_1 z^{-1} + D_1 z^{-2}} \cdot \frac{1 + A_2 z^{-1} + B_2 z^{-2}}{1 + C_2 z^{-1} + D_2 z^{-2}}.$$

Высокочастотный фильтр акцента:

$$Emp(z) = GG \frac{1 + AAz^{-1} + BBz^{-2}}{1 + CCz^{-1} + DDz^{-2}}.$$

Фильтр верхних частот:

$$Emp(z) = GL \frac{1 + ALz^{-1} + BLz^{-2}}{1 + CLz^{-1} + DLz^{-2}}.$$

Б.2.4 Таблицы

Таблица Б.1 — Коэффициенты фильтра нижних частот

G_{inv}	1,100000000000000
A_1	– 1,998066423746901
B_1	1,000000000000000
C_1	– 1,962822436245804

Окончание таблицы Б.1

D_1	0,9684991816600951
A_2	– 1,999633313803449
B_2	0,9999999999999999
C_2	– 1,858097918647416
D_2	0,8654599838007603

Таблица Б.2 — Коэффициенты высокочастотного фильтра акцента

AA	0,551543
BB	0,152100
CC	0,89
DD	0,198025
GG	1,226

Таблица Б.3 — Коэффициенты фильтра верхних частот

AL	– 2. * 1. * cos((4,0/4,0)*π)
BL	1.
CL	– 2. * 0,78 * cos((3,55/4,0)*π)
DL	0,78*0,78
GL	0,768

Приложение В
(справочное)

Определения системного уровня

В.1 Точка случайного доступа

Точка случайного доступа в потоке битов *HVXC* может быть установлена в любой граничной точке фрейма.

Приложение Г
(справочное)

Пример установки инструмента *EP* и маскировки ошибок для *HVXC*

Г.1 Краткий обзор

Этот раздел описывает пример реализации инструмента *EP* (защита от ошибок) и метод маскировки ошибок для *HVXC*. Некоторые из перцепционно важных битов защищены схемой *FEC* (прямая коррекция ошибок), а некоторые проверяются циклическим контролем избыточности (*CRC*), чтобы решать, включены ли в состав ошибочные биты. Когда происходит ошибка *CRC*, выполняется маскировка ошибок, чтобы уменьшить заметную деградацию.

Метод коррекции ошибок и установка инструмента *EP*, алгоритм маскировки ошибок, описанный ниже, являются одним из примеров, и они должны быть изменены в зависимости от фактических условий канала.

Г.2 Установка инструмента *EP*

Г.2.1 Пример внеполосной информации для *HVXC*

Фиксированная скорость 2,0 Кбит/с

ESC (Категория чувствительности ошибок) примеры двух смежных фреймов приложены непосредственно для классов *EP*:

Класс 1: 22 бита (фиксированных), 2 фрейма связанные, скорость кода *SRCPC* 8/16, *CRC* на 6 битов;
Класс 2: 4 бита (фиксированных), скорость кода *SRCPC* 8/8, 1 битовый *CRC*;
Класс 3: 4 бита (фиксированных), скорость кода *SRCPC* 8/8, 1 разрядный *CRC*;
Класс 4: 20 битов (фиксированных), 2 фрейма связанные, скорость кода *SRCPC* 8/8, *CRC* отсутствует.

```
1           /* number of predefined sets */
2           /* bit interleaving */ 0
3           /* bitstuffing */
2           /* 2 frame concatenate */
4           /* number of classes */
0 0 0 1 0 0 2 0  /* length_esc, srcpc_esc, crc_esc, concatenate, FEC type, No termination, interleave SW, class option */
22          /* bits used for class length (0 = until the end) */
8           /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
6           /* crc length */
0 0 0 0 0 0 3 0  /* length esc, srcpc esc, crc esc, concatenate, FEC type, No termination, interleave SW, class option */
4           /* bits used for class length (0 = until the end) */
0           /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
1           /* crc length */
0 0 0 0 0 0 3 0  /* length_esc, srcpc_esc, crc_esc, concatenate, FEC type, No termination, interleave SW, class option */
4           /* bits used for class length (0 = until the end) */
0           /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
1           /* crc length */
0 0 0 1 0 0 3 0  /* length_esc, srcpc_esc, crc_esc, concatenate, FEC type, No termination, interleave SW, class option */
10          /* 6its used for class length (0 = until the end) */
0           /* puncture rate for srcpc 0 = 8/8 ... 24 = 32/8 */
0           /* crc length */
```

Фиксированная скорость 4 Кбит/с.

ESC (Категория чувствительности ошибок) примеры двух смежных фреймов приложены непосредственно для классов *EP*:

Класс 1: 33 бита (фиксированных), 2 фрейма связанные, скорость кода *SRCPC* 8/16, 6 битов *CRC*;
Класс 2: 22 бита (фиксированных), 2 фрейма связанные, скорость кода *SRCPC* 8/8, 6 битов *CRC*;
Класс 3: 4 бита (фиксированных), скорость кода *SRCPC* 8/8, 1 бит *CRC*;
Класс 4: 4 бита (фиксированных), скорость кода *SRCPC* 8/8, 1 бит *CRC*;
Класс 5: 17 битов (фиксированных), 2 фрейма связанные, скорость кода SRCPC 8/8, *CRC* отсутствует.

```
1           /* number of predefined sets */
2           /* 1 bit interleaving */ 0 /* bitstuffing */
2           /* 2 frame concatenate */
5           /* number of classes */
0 0 0 1 0 0 2 0  / length_esc, srcpc_esc, crc_esc, concatenate, FEC type, No termination, interleave SW, class option */
33          /* bits used for class length (0 =until the end) */
```

```
8           /* puncture rate for srcpc 0 = 8/8... 24 = 32/8 */
6           /* crc length */
0 0 0 1 0 0 3 0   /* length esc, srcpc esc, crc esc, concatenate, FEC type, No termination, interleave SW, class option */
22          /* bits used for class length (0 =until the end) */
0           /* puncture rate for srcpc 0 = 8/8... 24 = 32/8 */
6           /* crc length */
0 0 0 0 0 0 3 0   /* length esc, srcpc esc, crc esc, concatenate, FEC type, No termination, interleave SW, class option */
4           /* bits used for class length (0 =until the end) */
0           /* puncture rate for srcpc 0 = 8/8... 24 = 32/8 */
1           /* crc length */
0 0 0 0 0 0 3 0   /* length esc, srcpc esc, crc esc, concatenate, FEC type, No termination, interleave SW, class option */
4           /* bits used for class length (0 =until the end) */
0           /* puncture rate for srcpc 0 = 8/8... 24 = 32/8 */
0           /* crc length */
0 0 0 1 0 0 3 0   /* length esc, srcpc esc, crc esc concatenate, FEC type, No termination, interleave SW, class option */
17          /* bits used for class length (0 =until the end) */
0           /* puncture rate for srcpc 0 = 8/8... 24 = 32/8 */
0           /* crc length */
```

Таблица Г.1 показывает назначение битов закодированного канала для использования параметров настройки вышеупомянутого инструмента *EP*.

Таблица Г.1 — Закодированное каналом назначение битов для использования инструмента *EP*

	Фиксированная скорость 2,0 Кбит/с	Фиксированная скорость 4 Кбит/с
Класс I		
Биты кодера источника	44(*1)	66(*1)
Контроль *CRC*	6	6
Кодовая скорость	8/16	8/16
Класс I совокупно	100	144
Класс II		
Биты кодера источника	4	44(*1)
Контроль *CRC*	1	6
Кодовая скорость	8/8	8/8
Класс II совокупно	5	50
Класс III		
Биты кодера источника	4	4
Контроль *CRC*	1	1
Кодовая скорость	8/8	8/8
Класс III совокупно	5	5
Класс IV		
Биты кодера источника	4	4
Контроль *CRC*	1	1
Кодовая скорость	8/8	8/8
Класс IV совокупно	5	5

Окончание таблицы Г.1

	Фиксированная скорость 2,0 Кбит/с	Фиксированная скорость 4 Кбит/с
Класс V		
Биты кодера источника	4	4
Контроль *CRC*	1	1
Кодовая скорость	8/8	8/8
Класс V совокупно	5	5
Класс VI		
Биты кодера источника	20(*1)	4
Контроль *CRC*	0	0
Кодовая скорость	8/8	8/8
Класс VI совокупно	20	5
Класс VII		
Биты кодера источника		34(*1)
Контроль *CRC*		0
Кодовая скорость		8/8
Класс VII совокупно		34
Совокупность битов всех классов	140	248
Битовая скорость	3,5 Кбит/с	6,20 Кбит/с

(*1) 2 фрейма связаны.

Класс I:

CRC охватывает все биты Класса I. Биты Класса I, включая циклический контроль избыточности *CRC*, защищены сверхточным кодированием.

Классы II—V (2,0 Кбит/с), II—VI (4 Кбит/с):

биты по крайней мере одного *CRC* охватывают исходные биты кодера этих классов.

Класс VI (2,0 Кбит/с), VII (4 Кбит/с):

исходные биты кодера не проверяются *CRC* и не защищены какой-либо схемой исправления ошибок.

Г.3 Маскирование ошибок

Когда обнаружена ошибка *CRC*, выполняется процесс маскирования ошибок (маскировка плохого фрейма). Пример метода маскировки описан ниже.

Состояние маскирования текущего фрейма обновляется на базе результата декодирования *CRC* Класса I. Диаграмма переходов состояния показана на рисунке Г.1. Начальное состояние — состояние = 0. Стрелка с символом «1» обозначает переход для плохого фрейма, и стрелка с символом «0» — хороший фрейм.

Г.3.1 Замена параметра

Согласно значению состояния выполняется следующая замена параметров. При отсутствии ошибок значение состояния становится 0, и полученные исходные биты кодера используются без какой-либо обработки маскированием.

Г.3.1.1 Параметры *LSP*

При состоянии = 1...6 параметры *LSP* заменяются таковыми из предыдущих.

Когда состояние = 1...7, если *LSP*4 = 0 (режим квантования *LSP* без межфреймового предсказания), то параметры *LSP* вычисляются исходя из всех индексов *LSP*, полученных в текущем фрейме. Если *LSP*4 = 1 (режим квантования *LSP* с межфреймовым кодированием), то параметры *LSP* вычисляются следующим методом.

В этом режиме параметры *LSP* от индекса *LSP*1 интерполируются с предыдущими *LSPs*.

$LSP_{base}(n) = p \cdot LSP_{prev}(n) + (1 - p)LSP_{1st}(n)$ для n = 1..10

$LSP_{base}(n)$ является параметрами *LSP* базового уровня, $LSP_{prev}(n)$ является предыдущими *LSPs*, $LSP_{1st}(n)$ — декодированные *LSPs* от текущего индекса *LSP*1 и p — коэффициент интерполяции. P меняется согласно числу

битов предыдущих ошибочных фреймов *CRC* Класса I, как показано в таблице Г.2. Индексы *LSP* - - *LSP*2, *LSP*3 и *LSP*5 не используются, и $LSP_{base}(n)$ используются как параметры текущего *LSP*.

Таблица Г.2 — Фактор *p*

Фрейм	P
0	0,7
1	0,6
2	0,5
3	0,4
4	0,3
5	0,2
6	0,1
≥ 7	0,0

Г.3.1.2 Переменная отключения звука

Согласно значению "*state*" (состояние), значение переменной "*mute*" (молчание) установлено для управления уровнем вывода речи.

Значение "*mute*" используется ниже.

В состоянии = 7 используется среднее число 1,0 и значения "*mute*" предыдущего фрейма (= 0,5 (1,0 + предыдущее "значение *mute*")), но когда это значение больше 0,8, значение "*mute*" заменяется на 0,8.

Таблица Г.3 — Переменная *mute* (молчание)

Состояние	Молчание
0	1,000
1	0,800
2	0,700
3	0,500
4	0,25000
5	0,125
6	0,000
7	Среднее/0,800

Г.3.1.3 Замена и управление усилением «речевых» параметров

В состоянии = 1..6, параметр спектра *SE_shape*1, *SE_shape*2, параметр усиления *SE_gain*, параметр спектра для кодека на 4 Кбит/с *SE_shape*3. *SE_shape*6 заменены соответствующими параметрами предыдущего фрейма. Кроме того, чтобы управлять громкостью вывода речи, параметры величины гармоник остаточного сигнала *LPC*, "*Am* [0 ... 127]" управляются по усилению, как показано в (Г.1). В этом уравнении $Am_{(org)}$ [*i*] вычисляется исходя из принятых параметров спектра последнего фрейма, свободного от ошибок

$$Am[i] = mute * Am_{(org)}[i] \text{ для } i = 0..127. \quad (Г.1)$$

Если предыдущий фрейм является неречевым и текущее состояние = 7, (Г.1) заменяется (Г.2)

$$Am[i] = 0,6 * mute * Am_{(org)}[i] \text{ для } i = 0..127. \quad (Г.2)$$

Как описано ранее, *SE_shape*1 и *SE_shape*2 индивидуально защищены *CRC* на 1 бит. В состоянии = 0 или 7, когда ошибки *CRC* этих классов обнаружены в то же самое время, квантованные величины гармоник с фиксированной размерностью Am_{qnt} [1..44] подавляются по усилению, как показано в (Г.3).

$$Am_{qnt}[i] = s[i] * Am_{qnt(org)}[i] \text{ для } i = 1..44, \quad (Г.3)$$

где $s[i]$ — коэффициент для подавления усиления.

Таблица Г.4 — Коэффициент для подавления усиления 's [0..44]'

i	1	2	3	4	5	6	7...44
s[i]	0,10	0,25	0,40	0,55	0,70	0,85	1,00

При 4 Кбит/с *SE_shape*4, *SE_shape*5 и *SE_shape*6 проверяются по *CRC* как биты Класса II. Когда обнаружена ошибка *CRC*, параметр спектра уровня расширения не используется.

Г.3.1.4 Замена и управление усилением «неречевых» параметров

В состоянии = 1..6, стохастические параметры усиления книги шифров *VX_gain*1[0]. *VX_gain*1[1] заменяются на *VX_gain*1[1] из последнего фрейма, свободного от ошибок. Также стохастические параметры усиления книги шифров для кодека на 4 Кбит/с *VX_gain*2[0]..*VX_gain*2[3] заменяются на *VX_gain*2[3] из последнего фрейма, свободного от ошибок.

Стохастические параметры формы книги шифров *VX_shape*1[0], *VX_shape*1[1] и стохастический параметр формы книги шифров кодека на 4 Кбит/с генерируются, исходя из случайно сгенерированных значений индекса.

Кроме того, чтобы управлять громкостью речевого выхода, остаточный сигнал *LPC res*[0 ... 159] управляется по усилению, как показано в (Г.4). В этом уравнении $res_{(org)}[i]$ вычисляется, исходя из стохастических параметров книги шифров.

$$res[i] = mute \cdot res_{org}[i] \ (0 \le i \le 159). \quad (Г.4)$$

Г.3.1.5 Переходы состояния маскирования фрейма

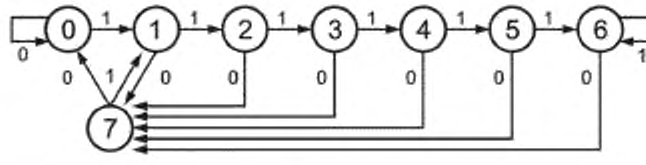


Рисунок Г. 1 — Переходы состояния маскирования фрейма

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 *Н.Е. Рагузина*
Технические редакторы *В.Н. Прусакова, И.Е. Черепкова*
Корректор *Е.И. Рычкова*
Компьютерная верстка *Г.В. Струковой*

Сдано в набор 09.06.2020. Подписано в печать 30.09.2020. Формат 60 × 84 1/8. Гарнитура Ариал.
Усл. печ. л. 8,37. Уч.-изд. л. 7,80.

Подготовлено на основе электронной версии, предоставленной разработчиком стандарта

ИД «Юриспруденция», 115419, Москва, ул. Орджоникидзе, 11.
www.jurisizdat.ru y-book@mail.ru

Создано в единичном исполнении во ФГУП «СТАНДАРТИНФОРМ»
для комплектования Федерального информационного фонда стандартов,
117418 Москва, Нахимовский пр-т, д. 31, к. 2.
www.gostinfo.ru info@gostinfo.ru