INTERNATIONAL STANDARD

ISO/IEC 14496-3

Second edition
2001-12-15

AMENDMENT 1
2003-11-01

Information technology — Coding of audio-visual objects —

Part 3:
Audio

AMENDMENT 1: Bandwidth extension

Technologies de l'information — Codage des objets audiovisuels —

Partie 3: Codage audio

AMENDEMENT 1: Extension de largeur de bande

Reference number
ISO/IEC 14496-3:2001/Amd.1:2003(E)

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

Amendment 1 to ISO/IEC 14496-3:2001 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

Introduction

This document specifies the first Amendment to the ISO/IEC 14496-3:2001 standard. The document specifies the normative syntax of the SBR tool and the decoding process. An informative encoder description is given as well. Furthermore, this document specifies two new profiles, one based on the AAC LC Audio Object Type and one based on AAC in combination with SBR.

Information technology — Coding of audio-visual objects —

Part 3:
Audio

AMENDMENT 1: Bandwidth extension

In ISO/IEC 14496-3:2001, Introduction, MPEG-4 general audio coding tools, add:

“

MPEG-4 SBR, (Spectral Band Replication) is a bandwidth extension tool used in combination with the AAC general audio codec. When integrated into the MPEG AAC codec, a significant improvement of the performance is available, which can be used to lower the bitrate or improve the audio quality. This is achieved by replicating the highband, i.e. the high frequency part of the spectrum. A small amount of data representing a parametric description of the highband is encoded and used in the decoding process. The data rate is by far below the data rate required when using conventional AAC coding of the highband.

“

Amendment Subpart 1

In Part 3: Audio, Subpart 1, in subclause 1.3 Terms and Definitions, add:
“

206. **SBR**: Spectral Band Replication.

”

and increase the index-number of subsequent entries.

In Part 3: Audio, Subpart 1, in subclause 1.5.1.1 Audio object type definition, replace table 1.1 with the following table:

“

Table 1.1 – Audio object definition

Tools/ Modules / Audio Object Type	gain control	block switching	window shapes - standard	window shapes – AAC LD	filterbank - standard	filterbank – SSR	TNS	LTP	intensity	coupling	MPEG-2 prediction	PNS	MS	SIAQ	FSS	upsampling filter tool	quantisation&coding - AAC	quantisation&coding - TwinVQ	quantisation&coding - BSAC	AAC ER Tools	ER payload syntax	EP Tool 1)	CELP	Silence Compression	HVXC	HVXC 4kbs VR	SA tools	SASBF	MIDI	HILN	TTSI	SBR	Remark	Object Type ID
Null																																	0
AAC main		X	X		X		X		X	X	X	X	X				X															2)	1
AAC LC		X	X		X		X		X	X		X	X				X																2
AAC SSR	X	X	X			X	X		X	X		X	X				X																3
AAC LTP		X	X		X		X	X	X	X		X	X				X															2)	4
SBR																															X		5
AAC Scalable		X	X		X		X	X	X			X	X	X	X	X	X															6)	6
TwinVQ		X	X		X		X	X					X					X															7
CELP																							X										8
HVXC																									X								9
(Reserved)																																	10
(Reserved)																																	11
TTSI																															X		12
Main synthetic																											X	X	X			3)	13
Wavetable synthesis																												X	X			4)	14
General MIDI																													X				15
Algorithmic Synthesis and Audio FX																											X						16
ER AAC LC		X	X		X		X		X			X	X				X			X	X	X										17
(Reserved)																																	18
ER AAC LTP		X	X		X		X	X	X			X	X				X			X	X	X										5)	19
ER AAC scalable		X	X		X		X		X			X	X	X	X	X	X			X	X	X										6)	20
ER TwinVQ		X	X		X		X						X					X			X	X											21
ER BSAC		X	X		X		X		X			X	X						X		X	X											22
ER AAC LD				X	X		X	X	X			X	X				X			X	X	X											23
ER CELP																					X	X	X	X									24
ER HVXC																					X	X			X	X							25
ER HILN																					X	X								X			26
ER Parametric																					X	X			X	X				X			27
(Reserved)																																	28
(Reserved)																																	29
(Reserved)																																	30
(Reserved)																																	31

”

In Part 3: Audio, Subpart 1, in subclause 1.5.1.2 Description, add after 1.5.1.2.5, add:

“

1.5.1.2.6 SBR-Object

The SBR Object contains the SBR-Tool and can be combined with the audio object types indicated in Table 1.2A

Table 1.2A – Audio object types that can be combined with the SBR Tool

Audio Object Type	Combination with SBR Tool permitted	Object Type ID
Null		0
AAC main	X	1
AAC LC	X	2
AAC SSR	X	3
AAC LTP	X	4
SBR		5
AAC Scalable	X	6
TwinVQ		7
CELP		8
HVXC		9
(Reserved)		10
(Reserved)		11
TTSI		12
Main synthetic		13
Wavetable synthesis		14
General MIDI		15
Algorithmic Synthesis and Audio FX		16
ER AAC LC	X	17
(Reserved)		18
ER AAC LTP	X	19
ER AAC scalable	X	20
ER TwinVQ		21
ER BSAC		22
ER AAC LD		23
ER CELP		24
ER HVXC		25
ER HILN		26
ER Parametric		27
(Reserved)		28
(Reserved)		29
(Reserved)		30
(Reserved)		31

”

In Part 3: Audio, Subpart 1, subclause 1.5.2.1 (Profiles), replace:

“

Eight Audio Profiles have been defined:

”

with

“

Ten Audio Profiles have been defined:

”

and add after item 8:

“

9. The **AAC Profile** contains the audio object type 2 (AAC-LC).

10. The **High Efficiency AAC Profile** contains the audio object types 5 (SBR) and 2 (AAC LC) The High Efficiency AAC Profile is a superset of the AAC Profile

”

In Part 3: Audio, Subpart 1, replace Table 1.2 (Audio Profiles definition) with the following table:

“

Table 1.2 – Audio Profiles definition

Audio Object Type	Main Audio Profile	Scalable Audio Profile	Speech Audio Profile	Synthetic Audio Profile	High Quality Audio Profile	Low Delay Audio Profile	Natural Audio Profile	Mobile Audio Internet-working Profile	AAC Profile	High Efficiency AAC Profile	Object Type ID
Null											0
AAC main	X						X				1
AAC LC	X	X			X		X		X	X	2
AAC SSR	X						X				3
AAC LTP	X	X			X		X				4
SBR										X	5
AAC Scalable	X	X			X		X				6
TwinVQ	X	X					X				7
CELP	X	X	X		X	X	X				8
HVXC	X	X	X			X	X				9
(reserved)											10
(reserved)											11
TTSI	X	X	X	X		X	X				12
Main synthetic	X			X							13
Wavetable synthesis											14
General MIDI											15
Algorithmic Synthesis and Audio FX											16
ER AAC LC					X		X	X			17
(reserved)											18
ER AAC LTP					X		X				19
ER AAC Scalable					X		X	X			20
ER TwinVQ							X	X			21
ER BSAC							X	X			22
ER AAC LD						X	X	X			23
ER CELP					X	X	X				24
ER HVXC						X	X				25
ER HILN							X				26
ER Parametric							X				27
(reserved)											28
(reserved)											29
(reserved)											30
(reserved)											31

”

In Part 3: Audio, Subpart 1, subclause 1.5.2.2 (Complexity units), replace table 1.3 by the table below:

"

Table 1.3 – Complexity of Audio Object Types and SR conversion

Object Type	Parameters	PCU (MOPS)	RCU	Remarks
AAC Main	fs = 48 kHz	5	5	1)
AAC LC	fs = 48 kHz	3	3	1)
AAC SSR	fs = 48 kHz	4	3	1)
AAC LTP	fs = 48 kHz	4	4	1)
SBR	fs = 24/48 kHz (in/out) (SBR tool)	3	2.5	1)
	fs = 24/48 kHz (in/out) (Low Power SBR tool)	2	1.5	1)
	fs = 48/48 kHz (in/out) (Down Sampled SBR tool)	4.5	2.5	1)
	fs = 48/48 kHz (in/out) (Low Power Down Sampled SBR tool)	3	1.5	1)
AAC Scalable	fs = 48 kHz	5	4	1), 2)
TwinVQ	fs = 24 kHz	2	3	1)
CELP	fs = 8 kHz	1	1	
CELP	fs = 16 kHz	2	1	
CELP	fs = 8/16 kHz (bandwidth scalable)	3	1	
HVXC	fs = 8 kHz	2	1	
TTSI		-	-	4)
General MIDI		4	1	
Wavetable Synthesis	fs = 22.05 kHz	depends on bitstreams (3)	depends on bitstreams (3)	
Main Synthetic		depends on bitstreams (3)	depends on bitstreams (3)	
Algorithmic Synthesis and AudioFX		depends on bitstreams (3)	depends on bitstreams (3)	
Sampling Rate Conversion	rf = 2, 3, 4, 6	2	0.5	7)
ER AAC LC	fs = 48 kHz	3	3	1)
ER AAC LTP	fs = 48 kHz	4	4	1)
ER AAC Scalable	fs = 48 kHz	5	4	1), 2)
ER TwinVQ	fs = 24 kHz	2	3	1)
ER BSAC	fs = 48 kHz (input buffer size=26000bits)	4	4	1)
	fs = 48 kHz (input buffer size=106000bits)	4	8	
ER AAC LD	fs = 48 kHz	3	2	1)
ER CELP	fs = 8 kHz	2	1	
	fs = 16 kHz	3	1	
ER HVXC	fs = 8 kHz	2	1	
ER HILN	fs = 16 kHz, ns=93	15	2	6)
	fs = 16 kHz, ns=47	8	2	
ER Parametric	fs = 8 kHz, ns=47	4	2	5),6)

"

In Part 3: Audio, Subpart 1, subclause 1.5.2.3 (Levels within the profiles), add at the end:

“

- **Levels for the AAC Profile**

Table 1.7A - Levels for the AAC Profile

Level	Max. channels/ object	Max. sampling rate [kHz]	Max. PCU	Max. RCU
1	2	24	3	5
2	2	48	6	5
3	NA	NA	NA	NA
4	5	48	19	15
5	5	96	38	15

For the audio object type 2 (AAC LC), mono or stereo mixdown elements are not permitted.

The NA (Not Applicable) levels are introduced to emphasize the hierarchical structure of the AAC Profile and the High Efficiency AAC Profile. Hence, a decoder supporting the High Efficiency AAC Profile at a given level can decode an AAC Profile stream of the same or a lower level. The NA levels are not indicated in the audioProfileLevelIndication table (Table 1.7z).

- **Levels for the High Efficiency AAC Profile**

Table 1.8A - Levels for the High Efficiency AAC Profile

Level	Max. channels/ object	Max. AAC sampling rate, SBR not present [kHz]	Max. AAC sampling rate, SBR present [kHz]	Max. SBR sampling rate [kHz] (in/out)	Max. PCU	Max. RCU	Max. PCU Low power SBR	Max. RCU Low power SBR
1	NA	NA	NA	NA	NA	NA	NA	NA
2	2	48	24	24/48	9	10	7	8
3	2	48	48	48/48 (Note 1)	15	10	12	8
4	5	48	24/48 (Note 2)	48/48 (Note 1)	25	28	20	23
5	5	96	48	48/96	49	28	39	23

Note 1: For level 3 and level 4 decoders, it is mandatory to operate the SBR tool in downsampled mode if the sampling rate of the AAC core is higher than 24kHz. Hence, if the SBR tool operates on a 48kHz AAC signal, the internal sampling rate of the SBR tool will be 96kHz, however, the output signal will be downsampled by the SBR tool to 48kHz.
Note 2: For one or two channels the maximum AAC sampling rate, with SBR present, is 48kHz. For more than two channels the maximum AAC sampling rate, with SBR present, is 24kHz.

For the audio object type 2 (AAC LC), mono or stereo mixdown elements are not permitted.

”

In Part 3: Audio, Subpart 1, subclause 1.5.2.4 (Table 1.7z - audioProfileLevelIndication Values), replace the row:

“

0x28-0x7F	reserved for ISO use	-

”

with

“

0x28	AAC Profile	L1
0x29	AAC Profile	L2
0x2A	AAC Profile	L4
0x2B	AAC Profile	L5
0x2C	High Efficiency AAC Profile	L2
0x2D	High Efficiency AAC Profile	L3
0x2E	High Efficiency AAC Profile	L4
0x2F	High Efficiency AAC Profile	L5
0x30-0x7F	reserved for ISO use	-

”

In Part 3: Audio, Subpart 1, in subclause 1.6.2.1 AudioSpecificConfig, replace table 1.8 with the following table:

“

Table 1.8 – Syntax of AudioSpecificConfig()

Syntax	No. of bits	Mnemonic
AudioSpecificConfig ()		
{		
audioObjectType;	**5**	**uimsbf**
samplingFrequencyIndex;	**4**	**uimsbf**
if (samplingFrequencyIndex==0xf)		
samplingFrequency;	**24**	**uimsbf**
channelConfiguration;	**4**	**uimsbf**
sbrPresentFlag = -1;		
if (audioObjectType == 5) {		
extensionAudioObjectType = audioObjectType;		
sbrPresentFlag = 1;		
extensionSamplingFrequencyIndex;	**4**	**uimsbf**
if (extensionSamplingFrequencyIndex==0xf)		
extensionSamplingFrequency;	**24**	**uimsbf**
audioObjectType;	**5**	**uimsbf**
}		
else {		
extensionAudioObjectType = 0;		
}		
if (audioObjectType == 1 || audioObjectType == 2 ||		
audioObjectType == 3 || audioObjectType == 4 ||		
audioObjectType == 6 || audioObjectType == 7)		
GASpecificConfig();		
if (audioObjectType == 8)		
CelpSpecificConfig();		
if (audioObjectType == 9)		
HvxcSpecificConfig();		
if (audioObjectType == 12)		
TTSSpecificConfig();		
if (audioObjectType == 13 || audioObjectType == 14 ||		
audioObjectType == 15 || audioObjectType==16)		
StructuredAudioSpecificConfig();		
/* the following Objects are Amendment 1 Objects */		
if (audioObjectType == 17 || audioObjectType == 19 ||		
audioObjectType == 20 || audioObjectType == 21 ||		
audioObjectType == 22 || audioObjectType == 23)		
GASpecificConfig();		
if (audioObjectType == 24)		
ErrorResilientCelpSpecificConfig();		
if (audioObjectType == 25)		
ErrorResilientHvxcSpecificConfig();		
if (audioObjectType == 26 || audioObjectType == 27)		
ParametricSpecificConfig();		
if (audioObjectType == 17 || audioObjectType == 19 ||		
audioObjectType == 20 || audioObjectType == 21 ||		
audioObjectType == 22 || audioObjectType == 23 ||		
audioObjectType == 24 || audioObjectType == 25 ||		
audioObjectType == 26 || audioObjectType == 27) {		
epConfig;	**2**	**uimsbf**
if (epConfig == 2 || epConfig == 3) {		
ErrorProtectionSpecificConfig();		
}		
if (epConfig == 3) {		

```
            directMapping;                                          1           uimsbf
            if ( ! directMapping ) {
                /* tbd */
            }
        }
    }

    if (  extensionAudioObjectType != 5 &&
        bits_to_decode() >= 16 ) {
        syncExtensionType;                                          11          bslbf
        if (syncExtensionType == 0x2b7) {
            extensionAudioObjectType;                               5           uimsbf
            if ( extensionAudioObjectType == 5 ) {
                sbrPresentFlag;                                     1           uimsbf
                if (sbrPresentFlag == 1) {
                    extensionSamplingFrequencyIndex;                4           uimsbf
                    if ( extensionSamplingFrequencyIndex == 0xf )
                        extensionSamplingFrequency;                 24          uimsbf
                }
            }
        }
    }
}
```

”

In Part 3: Audio, Subpart 1, in subclause 1.6.2.2.1 Overview, replace table 1.9 by the following table:

“

Table 1.9 – Audio Object Types

Audio Object Type	Object Type ID	definition of elementary stream payloads and detailed syntax	Mapping of audio payloads to access units and elementary streams
AAC MAIN	1	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC LC	2	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC SSR	3	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
AAC LTP	4	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.2
SBR	5	ISO/IEC 14496-3 subpart 4	
AAC scalable	6	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.3
TwinVQ	7	ISO/IEC 14496-3 subpart 4	
CELP	8	ISO/IEC 14496-3 subpart 3	
HVXC	9	ISO/IEC 14496-3 subpart 2	
TTSI	12	ISO/IEC 14496-3 subpart 6	
Main synthetic	13	ISO/IEC 14496-3 subpart 5	
Wavetable synthesis	14	ISO/IEC 14496-3 subpart 5	
General MIDI	15	ISO/IEC 14496-3 subpart 5	
Algorithmic Synthesis and Audio FX	16	ISO/IEC 14496-3 subpart 5	
ER AAC LC	17	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER AAC LTP	19	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER AAC scalable	20	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER Twin VQ	21	ISO/IEC 14496-3 subpart 4	
ER BSAC	22	ISO/IEC 14496-3 subpart 4	
ER AAC LD	23	ISO/IEC 14496-3 subpart 4	see subclause 1.6.2.2.2.1.4
ER CELP	24	ISO/IEC 14496-3 subpart 3	
ER HVXC	25	ISO/IEC 14496-3 subpart 2	
ER HILN	26	ISO/IEC 14496-3 subpart 7	
ER Parametric	27	ISO/IEC 14496-3 subpart 2 and 7	

”

In Part 3: Audio, Subpart 1, under 1.6.3 Semantics, after 1.6.3.6 Direct Mapping add:

“

1.6.3.7 extensionSamplingFrequencyIndex

A four bit field indicating the output sampling frequency of the extension tool corresponding to the extensionAudioObjectType, according to Table 1.10.

1.6.3.8 extensionSamplingFrequency

The output sampling frequency of the extension tool corresponding to the extensionAudioObjectType. Either transmitted directly, or coded in the form of extensionSamplingFrequencyIndex.

1.6.3.9 bits_to_decode

A helper function; returns the number of bits not yet decoded in the current AudioSpecificConfig(), if the length of this element has been signaled by a system/transport layer. If the length of this element is unknown, bits_to_decode() returns 0.

1.6.3.10 syncExtensionType

Syncword which marks the beginning of appended extension configuration data. This configuration data corresponds to an extension tool of which the coded data is embedded (in a backward compatible manner) in that of the underlying audioObjectType. If syncExtensionType is present, the configuration data of the extension tool is separated from that of the underlying audioObjectType, which allows for backward compatible signaling (see subclause 1.6.5). Decoders that do not support the extension tool can ignore the extension tool configuration data. Note that this backward compatible signaling can only be used in MPEG-4 based systems that convey the length of the AudioSpecificConfig().

1.6.3.11 sbrPresentFlag

A flag indicating the presence or absence of SBR data in case of extensionAudioObjectType==5 (i.e. explicit SBR signaling, see subclause 1.6.5). The value –1 indicates that the sbrPresentFlag was not conveyed in the AudioSpecificConfig(). In this case, a High Efficiency AAC Profile decoder shall be able to detect the presence of SBR data in the Elementary Stream (i.e. implicit SBR signaling, see subclause 1.6.5).

1.6.3.12 extensionAudioObjectType

A five bit field indicating the extension audio object type. This object type corresponds to an extension tool, which is used to enhance the underlying audioObjectType.

”

In Part 3: Audio, Subpart 1, after 1.6.4 Upstream, add the following subclause:

“

1.6.5 Signaling of SBR

1.6.5.1 Generating and Signaling AAC+SBR Content

The SBR tool in combination with the AAC coder provides a significant increase of audio compression efficiency. At the same time it allows for compatibility with existing AAC-only decoders. However, the audio quality for decoders without the SBR tool will of course be significantly lower than for those supporting the SBR tool. Therefore, depending on the application, a content provider or content creator will want to choose between the two alternatives given below. In general, the SBR data is always embedded in the AAC stream in a AAC compatible way (in the extension_payload), and SBR is a pure post processing step in the decoder. Therefore, compatibility can be achieved. However, by means of different signaling the content creator can select between the full-quality mode and the backward compatibility mode as follows.

1.6.5.1.1 Ensuring Full Audio Quality of AAC+SBR for the Listener

To ensure that all listeners get the full audio quality of AAC+SBR, the stream generated shall only play on SBR capable decoders (decoders that support the HE AAC Profile, hereinafter referred to as HE AAC Profile decoders). This is achieved by indicating the HE AAC profile and using the explicit, hierarchical signalling (signaling 2.A. as described below). As a result, decoders without SBR support will not play such streams. With regard to AAC-only streams, an HE AAC Profile decoders will decode all AAC Profile streams of the appropriate level, as the HE AAC Profile is a superset of the AAC Profile.

1.6.5.1.2 Achieving Backward Compatibility with Existing AAC-only Decoders

The aim of this mode is to get all AAC-based decoders to play the stream, even if they don't support the SBR tool. Compatible streams can be created using the following two signaling methods:

a) indicating a profile containing AAC (e.g. the AAC Profile), except the HE AAC Profile, and using the explicit backward compatible signalling (2.B. as described below). This method is recommended for all MPEG-4 based systems in which the length of the AudioSpecificConfig() is known in the decoder. As this is not the case for LATM with audioMuxVersion==0 (see clause 1.7), this method cannot be used for LATM with audioMuxVersion==0. In explicit backward compatible signaling, SBR-specific configuration data is added at the end of the AudioSpecificConfig(). Decoders that do not know about SBR will ignore these parts, while HE AAC Profile decoders will detect its presence and configure the decoder accordingly.

b) indicating a profile containing AAC (e.g. the AAC Profile, or an MPEG-2 AAC profile), except the HE AAC Profile, and using implicit signalling. In this mode, there is no explicit indication of the presence of SBR data. Instead, decoders check the presence while decoding the stream and use the SBR tool if SBR data is found. This is possible because SBR can be decoded without SBR-specific configuration data if a certain way of handling decoder output sample rate is obeyed, as described below for HE AAC Profile decoders.

Both methods lead to the result that the AAC part of an AAC+SBR streams will be decoded by AAC-only decoders. AAC+SBR decoders will detect the presence of SBR and decode the full quality AAC+SBR stream.

1.6.5.2 Implicit and Explicit Signaling of SBR

This subclause outlines the different signaling methods of SBR, and the decoder behavior for different types of signaling.

There are several ways to signal the presence of SBR data:

1. **implicit signaling:** If EXT_SBR_DATA or EXT_SBR_DATA_CRC extension_payload() elements are detected in the bitstream, this implicitly signals the presence of SBR data. The ability to detect and decode implicitly signaled SBR is mandatory for all High Efficiency AAC Profile (HE AAC Profile) decoders.

2. **explicit signaling:** The presence of SBR data is signaled explicitly by means of the SBR Audio Object Type in the AudioSpecificConfig(). When explicit signaling is used, implicit signaling shall not occur. Two different types of explicit signaling are available:

   2.A. **hierarchical signaling:** If the first audioObjectType (AOT) signaled is the SBR AOT, a second audio object type is signaled which indicates the underlying audio object type. This signaling method is not backward compatible.

   2.B. **backward compatible signaling:** The extensionAudioObjectType is signaled at the end of the AudioSpecificConfig(). This method shall only be used in systems that convey the length of the AudioSpecificConfig(). Hence, it shall not be used for LATM with audioMuxVersion==0.

Table 1.15A shows the decoder behavior depending on profile and audio object type indication when implicit or explicit signaling is used.

Table 1.15A – SBR Signaling and Corresponding Decoder Behavior

Bitstream characteristics				Decoder behavior (Note 4)	
Profile indication	**extension AudioObjectType**	**sbrPresent Flag**	**raw_data_block**	**AAC decoders not supporting HE AAC Profile**	**AAC decoders supporting HE AAC Profile**
Profiles with AAC support other than High Efficiency AAC Profile	!= SBR (signaling 1)	-1 (Note 1)	AAC	Play AAC	Play AAC
			AAC+SBR	Play AAC	Play at least AAC, should play AAC+SBR
	== SBR (signaling 2.B)	0 (Note 2)	AAC	Play AAC	Play AAC
		1 (Note 3)	AAC+SBR	Play AAC	Play at least AAC, should play AAC+SBR
High Efficiency AAC Profile	== SBR (signaling 2.A or 2.B)	1 (Note 3)	AAC+SBR	Unsupported Profile - Don't play	Play AAC+SBR

Note 1: Implicit signaling, check payload in order to determine output sampling frequency, or assume the presence of SBR data in the payload, giving an output sampling frequency of twice the sampling frequency indicated by samplingFrequency in the AudioSpecificConfig() (unless the down sampled SBR Tool is operated, or twice the sampling frequency indicated by samplingFrequency exceeds the maximum allowed output sampling frequency of the current level, in which case the output sampling frequency is the same as indicated by samplingFrequency).
Note 2: Explicitly signals that there is no SBR data, hence no implicit signaling is present, and the output sampling frequency is given by samplingFrequency in the AudioSpecificConfig().
Note 3: Output sampling frequency is the extensionSamplingFrequency in AudioSpecificConfig().
Note 4: In all cases a decoder has to support the Profile and Level indicated in the bitstream in order to be able to decode and play the content of the bitstream.

The upper part of Table 1.15A displays bitstream characteristics and decoder behavior if the profile indication is any profile with AAC, apart from the High Efficiency AAC Profile. The lower part displays bitstream characteristics and decoder behavior if the profile indication is the High Efficiency AAC Profile.

1.6.5.3 HE AAC Profile Decoder Behavior in Case of Implicit Signaling

If the presence of SBR data is backward compatible implicitly signaled (signaling 1, in the list above) the extensionAudioObjectType is not the SBR AOT, and the sbrPresentFlag is set to –1, indicating that implicit signaling may occur.

Since the HE AAC Profile decoder is a dual rate system, with the SBR Tool operating at twice the sample rate of the underlying AAC decoder, the output sample rate cannot be assumed to be that of the AAC decoder just because SBR is not explicitly signaled. The decoder shall determine the output sample rate by either of the following two methods:

- Check for the presence of SBR data in the bitstream prior to decoding. If no SBR data is found, the output sample rate is equal to that signaled as samplingFrequency in the AudioSpecificConfig(). If SBR data is found the output sample rate is twice that signaled as samplingFrequency in the AudioSpecificConfig

- Assume that the SBR data is available and decide the output sample rate to be twice that signaled in the AudioSpecificConfig(). If no SBR data is found once the decoding process has started, the SBR Tool can be used for upsampling only, as described in subclause 4.6.18.5.

The above only applies if twice the sample rate signaled in the AudioSpecificConfig() does not exceed the maximum output sample rate allowed for the current level. Hence, for a HE AAC Profile decoder of levels 2, 3, or 4, the output sample rate is equal to the sample rate signaled in the AudioSpecificConfig() if the latter exceeds 24kHz.

The down sampled SBR Tool shall be used when needed to ensure that the output sample rate does not exceed the maximum allowed sample rate of the present level of the High Efficiency AAC Profile decoder.

1.6.5.4 HE AAC Profile Decoder Behavior in Case of Explicit Signaling

If the presence of SBR data is explicitly signaled (signaling 2, in the list above) the presence of SBR data is backward compatible explicitly signaled (signaling 2.B) or non-backward explicitly signaled (signaling 2.A).

For the backward compatible explicit signaled (signaling 2.B) the extensionAudioObjectType signaled is the SBR AOT. For this backward compatible explicit signaling the sbrPresentFlag is transmitted and can be either zero or one. If the sbrPresentFlag is zero, this indicates that SBR data is not present, and hence the HE AAC Profile decoder does not have to check the Fill-element for the presence of SBR data or make assumptions on the output sample rate in anticipation of SBR data. If the sbrPresentFlag is one, SBR data is present and the HE AAC Profile decoder shall operate the SBR Tool.

For the non-backward compatible explicit signaling of SBR (signaling 2.A) the extensionAudioObjectType signaled is the SBR AOT. For this hierarchical explicit signaling, the sbrPresentFlag is set to one if the extensionAudioObjectType is SBR. The sbrPresentFlag is not transmitted and hence it is not possible to explicitly signal the absence of implicit signaling. Hence, for the hierarchical explicit signaling, SBR data is always present and the HE AAC Profile decoder shall operate the SBR Tool.

The down sampled SBR Tool shall be operated if the output sample rate would otherwise exceed the maximum allowed output sample rate for the present level, or if the extensionSamplingFrequency is the same as the samplingFrequency.

”

In clause Annex 1.C (informative) Patent statements, replace the table by the following table:

“

Legend:

1. The presence of a name of a company in the list below indicates that a patent statement has been received from that company
2. The presence of a cross indicates that the statement identifies the part of the MPEG-4 version 2 standard to which the statement applies
3. No cross in a line indicates that the statement does not identify which part of the standard the statement applies

	Company	Part 1	Part 2	Part 3	Part 5	Part 6
1.	Alcatel	x	x	x	x	x
2.	Apple	x			x	
3.	AT&T	x	x	x		
4.	BBC	x	x	x	x	x
5.	Bosch	x	x	x	x	x
6.	British Telecommunications	x	x	x	x	x
7.	Canon	x	x	x	x	x
8.	CCETT	x	x	x	x	x
9.	Coding Technologies			x		
10.	Columbia Innovation Enterprise	x	x	x	x	x
11.	Columbia University	x	x	x	x	x
12.	Creative	x		x	x	
13.	CSELT			x		
14.	DemoGraFX	x	x	x	x	x
15.	DirecTV	x	x	x		
16.	Dolby	x	x	x	x	x
17.	EPFL	x	x	x	x	
18.	ETRI	x	x	x	x	x
19.	France Telecom	x	x	x	x	x
20.	Fraunhofer	x	x	x	x	x
21.	Fujitsu	x	x	x	x	x
22.	GC Technology Corporation	x	x	x		
23.	General Instrument		x		x	
24.	Hitachi	x	x	x	x	x
25.	Hyundai	x	x	x	x	x
26.	IBM	x	x	x	x	x
27.	Institut fuer Rundfunktechnik	x	x	x		x
28.	Intertrust					
29.	JVC	x	x	x	x	x
30.	KDD Corporation	x	x			
31.	KPN	x	x	x	x	x
32.	LG Semicon					
33.	Lucent					
34.	Matsushita Electric Industrial Co., Ltd.	x	x	x	x	x
35.	Microsoft	x	x	x	x	x
36.	MIT					
37.	Mitsubishi	x	x	x	x	x

38.	Motorola		x		x	
39.	NEC	x	x	x	x	x
40.	NHK	x	x	x	x	x
41.	Nokia	x	x	x	x	
42.	NTT	x	x	x	x	x
43.	NTT Mobile Communication Networks			x		
44.	OKI	x	x	x	x	x
45.	Optibase	x			x	
46.	Philips	x	x	x	x	x
47.	PictureTel Corporation		x		x	
48.	Rockwell	x	x	x	x	x
49.	Samsung	x	x	x	x	
50.	Sarnoff	x	x	x	x	x
51.	Scientific Atlanta	x	x	x	x	x
52.	Sharp	x	x	x	x	x
53.	Siemens	x	x	x	x	x
54.	Sony	x	x	x	x	x
55.	Sun	x				
56.	Telenor	x	x	x	x	x
57.	Teltec DCU		x		x	
58.	Texas Instruments					
59.	Thomson	x	x	x	x	x
60.	Toshiba		x			
61.	Unisearch Ltd.		x		x	
62.	Vector Vision		x			

”

Amendment Subpart 4

In Part 3: Audio, Subpart 4, subclause 4.1.1 Technical Overview, 4.1.1.1 Encoder Decoder Block Diagrams, replace Figure 4.1 by the following figure:

“

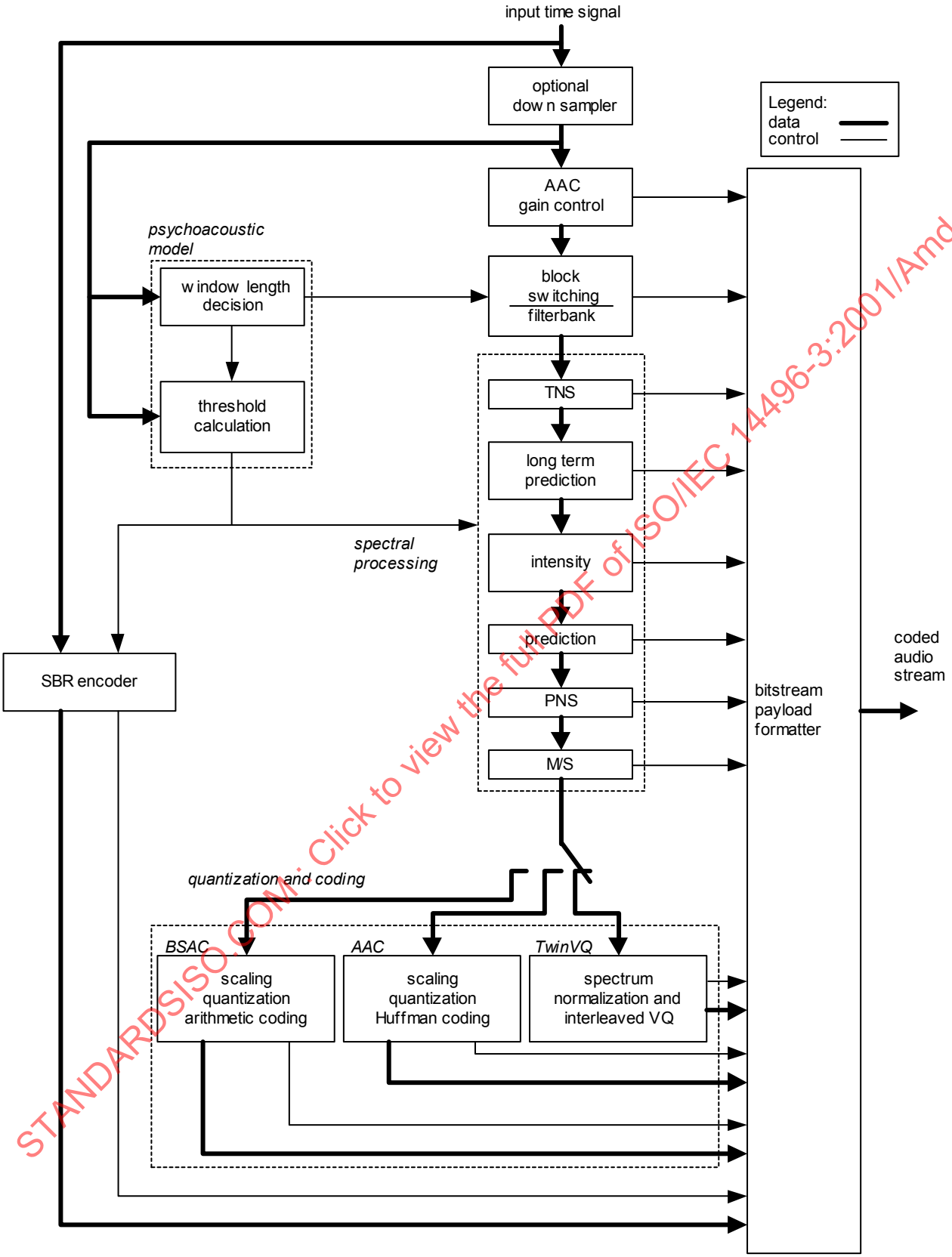


Figure 4.1 – Block diagram GA non scalable encoder

”

In Part 3: Audio, Subpart 4, subclause 4.1.1 Technical Overview, 4.1.1.1 Encoder Decoder Block Diagrams, add the following to Figure 4.2:

"

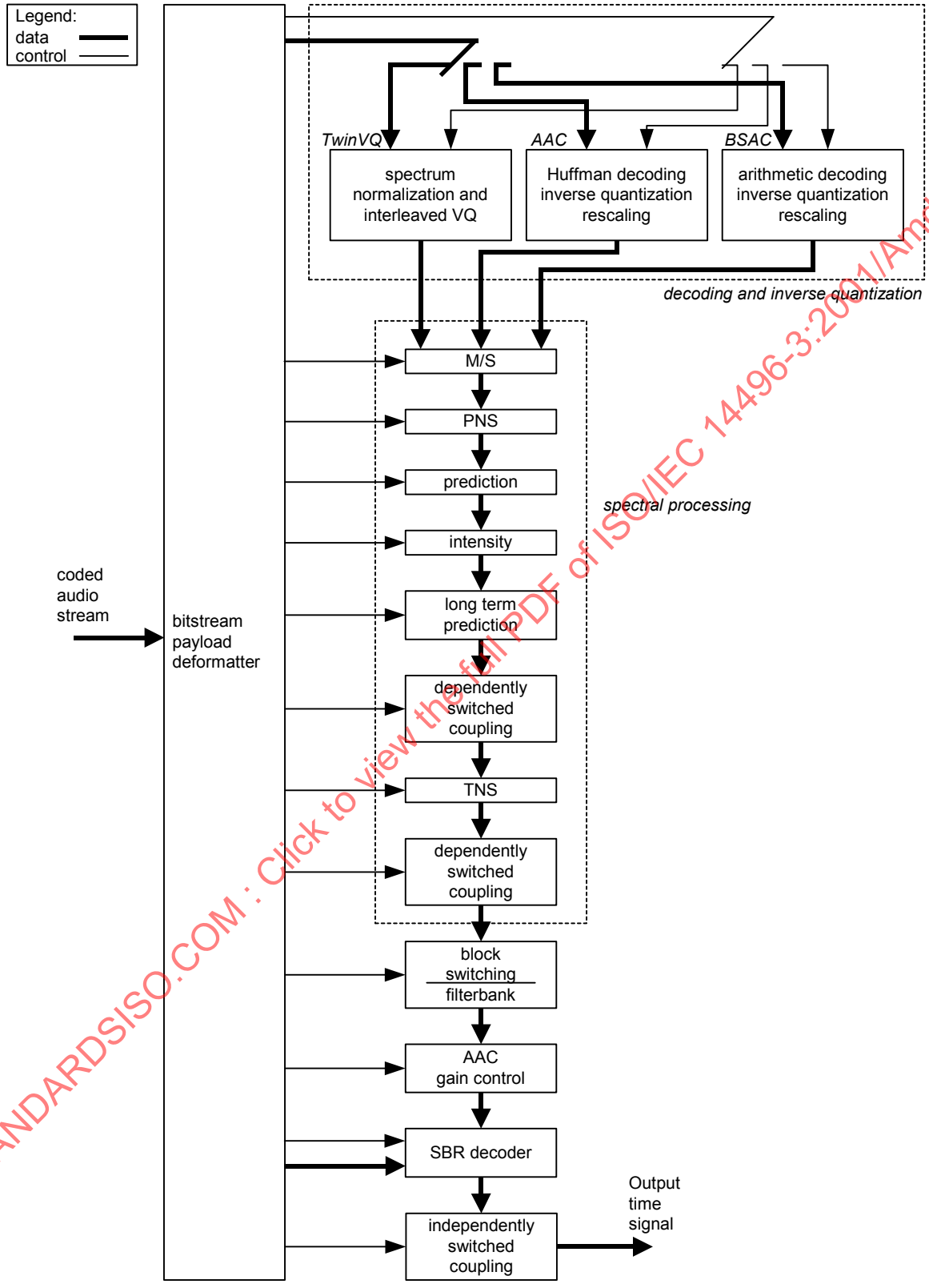


Figure 4.2 – Block diagram of the GA non scalable decoder

"

In Part 3: Audio, Subpart 4, subclause 4.1.1 Technical Overview, 4.1.1.2 Overview of the encoder and Decoder Tools, add the following:

“

The SBR tool regenerates the highband of the audio signal. It is based on replication of the sequences of harmonics, truncated during encoding. It adjusts the spectral envelope of the generated high-band and applies inverse filtering, and adds noise and sinusoidal components in order to recreate the spectral characteristics of the original signal.

The input to the SBR tool is:

- The quantized envelope data;
- Misc. control data;
- A time domain signal from the AAC core decoder.

The output of the SBR tool is:

- A time domain signal.

”

In Part 3: Audio, Subpart 4, Subclause 4.4.2.7 Subsidiary payloads, replace the definition of extension_payload(), Table 4.51:

“

Table 4.51 – Syntax of extension_payload()

Syntax	No. of bits	Mnemonic

extension_payload(cnt)
{
  extension_type;
  align = 4;
  switch(extension_type) {
  case EXT_DYNAMIC_RANGE:
  return dynamic_range_info();
  case EXT_SBR_DATA:
  return sbr_extension_data(id_aac, 0);
  case EXT_SBR_DATA_CRC:
  return sbr_extension_data(id_aac, 1);
  case EXT_FILL_DATA:
  fill_nibble; /* must be ‘0000’ */
  for (i=0; i<cnt-1; i++) {
  fill_byte[i]; /* must be ‘10100101’ */
  }
  return cnt;
  case EXT_DATA_ELEMENT:
  data_element_version;
  switch(data_element_version) {
  case ANC_DATA:
  loopCounter = 0;
  dataElementLength = 0;
  do {
  dataElementLengthPart;
  dataElementLength += dataElementLengthPart;
  loopCounter++;
  } while (dataElementLengthPart == 255);
  for (i=0; i<dataElementLength; i++) {
  data_element_byte[i];
  }
  return (dataElementLength+loopCounter+1);
  case default:
  align = 0;
  }
  case EXT_FIL:
  case default:
  for (i=0; i<8*(cnt-1)+align; i++) {
  other_bits[i];
  }
  return cnt;
  }
}
	4
	
	4
	8
	
	4
	
	8
	
	8
	
	1
		uimsbf
		
		Note 1
		
		Note 1
		
		uimsbf
		uimsbf
		
		uimsbf
		
		uimsbf
		
		uimsbf

Note 1: id_aac is the id_syn_ele of the corresponding AAC element (ID_SCE or ID_CPE) or ID_SCE in case of CCE.		

”

In Part 3: Audio, Subpart 4, after Subclause 4.4.2.7 Subsidiary payloads, add the following subclause:

“

4.4.2.8 Payloads for the audio object type SBR

Table 4.54A – Syntax of sbr_extension_data()

Syntax	No. of bits	Mnemonic

sbr_extension_data(id_aac, crc_flag)
{
num_sbr_bits = 0;
 	
if (crc_flag) {
bs_sbr_crc_bits;
num_sbr_bits += 10;
}
 	
if (sbr_layer != SBR_STEREO_ENHANCE) {
num_sbr_bits += 1;
if (**bs_header_flag**)
num_sbr_bits += sbr_header();
}
 	
num_sbr_bits += sbr_data(id_aac, bs_amp_res);
 	
num_align_bits = 8*cnt - 4 - num_sbr_bits;
bs_fill_bits;
 	
return ((num_sbr_bits + num_align_bits + 4) / 8)
}
	
	10
	
	1
	
	num_align_bits
		
		uimsbf
		
		Note 1
		
		Note 2
		
		Note 2
		
		uimsbf

Note 1: When the SBR tool is used with a non-scalable AAC core coder, the value of the helper variable sbr_layer is SBR_NOT_SCALABLE. When the SBR tool is used with a scalable AAC core coder, the value of the helper variable sbr_layer depends on the current layer and the scalability configuration of the AAC core coder as defined in Table 4.86 in Subclause 4.5.2.8.2.4. Note 2: sbr_header() and sbr_data() return the number of bits read (cnt is a parameter in extension_payload()).		

Table 4.55A – Syntax of sbr_header()

Syntax	No. of bits	Mnemonic
sbr_header()		
{		
bs_amp_res;	**1**	
bs_start_freq;	**4**	**uimsbf**, Note 1
bs_stop_freq;	**4**	**uimsbf**, Note 1
bs_xover_band;	**3**	**uimsbf**, Note 2
bs_reserved;	**2**	**uimsbf**
bs_header_extra_1;	**1**	
bs_header_extra_2;	**1**	
if (bs_header_extra_1) {		Note 3
bs_freq_scale;	**2**	**uimsbf**
bs_alter_scale;	**1**	
bs_noise_bands;	**2**	**uimsbf**
}		
if (bs_header_extra_2) {		Note 3
bs_limiter_bands;	**2**	**uimsbf**
bs_limiter_gains;	**2**	**uimsbf**
bs_interpol_freq;	**1**	
bs_smoothing_mode;	**1**	
}		
}		

Note 1: bs_start_freq and bs_stop_freq shall define a frequency band that does not exceed the limits defined in subclause 4.6.18.3.6.
Note 2: Index to the master frequency band table, indicating where the current SBR range begins
Note 3: If this bit is not set the default values for the underlying bitstream elements should be used.

Table 4.56A – Syntax of sbr_data()

Syntax	No. of bits	Mnemonic

sbr_data(id_aac, bs_amp_res)
{
switch (sbr_layer) {
case SBR_NOT_SCALABLE
switch (id_aac) {
case ID_SCE
sbr_single_channel_element(bs_amp_res)
break;
case ID_CPE
sbr_channel_pair_element(bs_amp_res)
break;
}
break;
case SBR_MONO_BASE
sbr_channel_pair_base_element(bs_amp_res)
break;
case SBR_STEREO_ENHANCE
sbr_channel_pair_enhance_element(bs_amp_res)
break;
case SBR_STEREO_BASE
sbr_channel_pair_element(bs_amp_res)
break;
}
}
		Note 1

Note 1: When the SBR tool is used with a non-scalable AAC core coder, the value of the helper variable sbr_layer is SBR_NOT_SCALABLE. When the SBR tool is used with a scalable AAC core coder, the value of the helper variable sbr_layer depends on the current layer and the scalability configuration of the AAC core coder as defined in Table 4.86 in Subclause 4.5.2.8.2.4.		

Table 4.57A – Syntax of sbr_single_channel_element()

Syntax	No. of bits	Mnemonic
sbr_single_channel_element(bs_amp_res)		
{		
if (**bs_data_extra**)	**1**	
bs_reserved;	**4**	**uimsbf**
sbr_grid(0);		
sbr_dtdf(0);		
sbr_invf(0);		
sbr_envelope(0, 0, bs_amp_res);		
sbr_noise(0, 0);		
if (**bs_add_harmonic_flag**[0])	**1**	
sbr_sinusoidal_coding(0);		
if (**bs_extended_data**) {	**1**	
cnt = **bs_extension_size**;	**4**	**uimsbf**
if (cnt == 15)		
cnt += **bs_esc_count**;	**8**	**uimsbf**
num_bits_left = 8 * cnt;		
while (num_bits_left > 7) {		
bs_extension_id;	**2**	**uimsbf**
num_bits_left -= 2;		
sbr_extension(bs_extension_id, num_bits_left);		Note 1
}		
}		
}		
Note 1: sbr_extension() shall decrease the variable num_bits_left by the number of bits read from the bitstream within sbr_extension(). The sbr_extension() element is reserved for future use.		

Table 4.58A – Syntax of sbr_channel_pair_element()

Syntax	No. of bits	Mnemonic
sbr_channel_pair_element(bs_amp_res)		
{		
if (**bs_data_extra**) {	**1**	
bs_reserved;	**4**	**uimsbf**
bs_reserved;	**4**	**uimsbf**
}		
if (**bs_coupling**) {	**1**	
sbr_grid(0);		
sbr_dtdf(0);		
sbr_dtdf(1);		
sbr_invf(0);		
sbr_envelope(0,1, bs_amp_res);		
sbr_noise(0,1);		
sbr_envelope(1,1, bs_amp_res);		
sbr_noise(1,1);		
} else {		
sbr_grid(0);		
sbr_grid(1);		
sbr_dtdf(0);		
sbr_dtdf(1);		
sbr_invf(0);		
sbr_invf(1);		
sbr_envelope(0,0, bs_amp_res);		
sbr_envelope(1,0, bs_amp_res);		
sbr_noise(0,0);		
sbr_noise(1,0);		
}		
if (**bs_add_harmonic_flag**[0])	**1**	
sbr_sinusoidal_coding(0);		
if (**bs_add_harmonic_flag**[1])	**1**	
sbr_sinusoidal_coding(1);		
if (**bs_extended_data**) {	**1**	
cnt = **bs_extension_size**;	**4**	**uimsbf**
if (cnt == 15)		
cnt += **bs_esc_count**;	**8**	**uimsbf**
num_bits_left = 8 * cnt;		
while (num_bits_left > 7) {		
bs_extension_id;	**2**	**uimsbf**
num_bits_left -= 2;		
sbr_extension(bs_extension_id, num_bits_left);		Note 1
}		
}		
}		

Note 1: sbr_extension() shall decrease the variable num_bits_left by the number of bits read from the bitstream within sbr_extension(). The sbr_extension() element is reserved for future use.

Table 4.59A – Syntax of sbr_channel_pair_base_element()

Syntax	No. of bits	Mnemonic
sbr_channel_pair_base_element(bs_amp_res)		
{		
if (**bs_data_extra**) {	**1**	
bs_reserved;	**4**	**uimsbf**
bs_reserved;	**4**	**uimsbf**
}		
bs_coupling	**1**	Note 1
sbr_grid(0);		
sbr_dtdf(0);		
sbr_invf(0);		
sbr_envelope(0,1, bs_amp_res);		
sbr_noise(0,1);		
if (**bs_add_harmonic_flag**[0])	**1**	
sbr_sinusoidal_coding(0);		
if (**bs_extended_data**) {	**1**	
cnt = **bs_extension_size**;	**4**	**uimsbf**
if (cnt == 15)		
cnt += **bs_esc_count**;	**8**	**uimsbf**
num_bits_left = 8 * cnt;		
while (num_bits_left > 7) {		
bs_extension_id;	**2**	**uimsbf**
num_bits_left -= 2;		
sbr_extension(bs_extension_id, num_bits_left);		Note 2
}		
}		
}		

Note 1: bs_coupling shall have the value 1.
Note 2: sbr_extension() shall decrease the variable num_bits_left by the number of bits read from the bitstream within sbr_extension(). The sbr_extension() element is reserved for future use.

Table 4.60A – Syntax of sbr_channel_pair_enhance_element()

Syntax	No. of bits	Mnemonic
sbr_channel_pair_enhance_element(bs_amp_res)		
{		
sbr_dtdf(1);		
sbr_envelope(1,1, bs_amp_res);		
sbr_noise(1,1);		
if (**bs_add_harmonic_flag**[1])	**1**	
sbr_sinusoidal_coding(1);		
}		

Table 4.61A – Syntax of sbr_grid()

Syntax	No. of bits	Mnemonic
sbr_grid(ch)		
{		
switch (**bs_frame_class**) {	**2**	**uimsbf**
case FIXFIX		
bs_num_env[ch] = 2^ **tmp**;	**2**	**uimsbf**, Note 1
if (bs_num_env[ch] == 1)		
bs_amp_res = 0;		
bs_freq_res[ch][0];	**1**	
for (env = 1; env < bs_num_env[ch]; env++)		
bs_freq_res[ch][env] = bs_freq_res[ch][0];		
break;		
case FIXVAR		
bs_var_bord_1[ch];	**2**	**uimsbf**
bs_num_env[ch] = **bs_num_rel_1**[ch] + 1;	**2**	**uimsbf**
for (rel = 0; rel < bs_num_env[ch]-1; rel++)		
bs_rel_bord_1[ch][rel] = 2* **tmp** + 2;	**2**	**uimsbf**
ptr_bits = ceil (log (bs_num_env[ch] + 1) / log (2));		Note 2
bs_pointer[ch];	ptr_bits	**uimsbf**
for (env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res[ch][bs_num_env[ch] – 1 – env];	**1**	
break;		
case VARFIX		
bs_var_bord_0[ch];	**2**	**uimsbf**
bs_num_env[ch] = **bs_num_rel_0**[ch] + 1;	**2**	**uimsbf**
for (rel = 0; rel < bs_num_env[ch]-1; rel++)		
bs_rel_bord_0[ch][rel] = 2* **tmp** + 2;	**2**	**uimsbf**
ptr_bits = ceil (log (bs_num_env[ch] + 1) / log (2));		Note 2
bs_pointer[ch];	ptr_bits	**uimsbf**
for (env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res[ch] [env];	**1**	
break;		
case VARVAR		
bs_var_bord_0[ch];	**2**	**uimsbf**
bs_var_bord_1[ch];	**2**	**uimsbf**
bs_num_rel_0[ch];	**2**	**uimsbf**
bs_num_rel_1[ch];	**2**	**uimsbf**
bs_num_env[ch] = bs_num_rel_0[ch] + bs_num_rel_1[ch] + 1;		Note 1
for (rel = 0; rel < bs_num_rel_0[ch]; rel++)		
bs_rel_bord_0[ch][rel] = 2* **tmp** + 2;	**2**	**uimsbf**
for (rel = 0; rel < bs_num_rel_1[ch]; rel++)		
bs_rel_bord_1[ch][rel] = 2* **tmp** + 2;	**2**	**uimsbf**
ptr_bits = ceil (log(bs_num_env[ch] + 1) / log (2));		Note 2
bs_pointer[ch];	ptr_bits	**uimsbf**
for (env = 0; env < bs_num_env[ch]; env++)		
bs_freq_res[ch][env];	**1**	
break;		
}		
if (bs_num_env[ch] > 1)		
bs_num_noise[ch] = 2;		
else		
bs_num_noise[ch] = 1;		
}		

Note 1: bs_num_env is restricted according to subclause 4.6.18.3.6.
Note 2: the division (/) is a float division without rounding or truncation.

Table 4.62A – Syntax of sbr_dtdf()

Syntax	No. of bits	Mnemonic

sbr_dtdf(ch)
{
for (env = 0; env < bs_num_env[ch]; env++)
bs_df_env[ch][env];
for (noise = 0; noise < bs_num_noise[ch]; noise++)
bs_df_noise[ch][noise];
}
	
	1
	
	1	

Table 4.63A – Syntax of sbr_invf()

Syntax	No. of bits	Mnemonic

sbr_invf(ch)
{
for (n = 0; n<num_noise_bands[ch]; n++)
bs_invf_mode[ch][n];
}
	
	2
		
		Note 1
		uimsbf

Note 1: num_noise_bands[ch] is derived from the header, according to subclause 4.6.18.3 and is named N_Q.		

Table 4.64A – Syntax of sbr_envelope()

Syntax	No. of bits	Mnemonic
sbr_envelope(ch, bs_coupling, bs_amp_res)		
{		
if (bs_coupling) {		
if (ch) {		
if (bs_amp_res) {		
t_huff = t_huffman_env_bal_3_0dB;		
f_huff = f_huffman_env_bal_3_0dB;		
} else {		
t_huff = t_huffman_env_bal_1_5dB;		
f_huff = f_huffman_env_bal_1_5dB;		
}		
} else {		
if (bs_amp_res) {		
t_huff = t_huffman_env_3_0dB;		
f_huff = f_huffman_env_3_0dB;		
} else {		
t_huff = t_huffman_env_1_5dB;		
f_huff = f_huffman_env_1_5dB;		
}		
}		
} else {		
if (bs_amp_res) {		
t_huff = t_huffman_env_3_0dB;		
f_huff = f_huffman_env_3_0dB;		
} else {		
t_huff = t_huffman_env_1_5dB;		
f_huff = f_huffman_env_1_5dB;		
}		
}		
for (env = 0; env < bs_num_env[ch]; env++) {		
if (bs_df_env[ch][env] == 0) {		
if (bs_coupling && ch) {		
if (bs_amp_res)		
bs_data_env[ch][env][0] = **bs_env_start_value_balance**;	**5**	**uimsbf**
else		
bs_data_env[ch][env][0] = **bs_env_start_value_balance**;	**6**	**uimsbf**
} else {		
if (bs_amp_res)		
bs_data_env[ch][env][0] = **bs_env_start_value_level**;	**6**	**uimsbf**
else		
bs_data_env[ch][env][0] = **bs_env_start_value_level**;	**7**	**uimsbf**
}		
for (band = 1; band < num_env_bands[bs_freq_res[ch][env]]; band++)		Note 1
bs_data_env[ch][env][band] = sbr_huff_dec(f_huff, **bs_codeword**);	**1..18**	Note 2
} else {		
for (band = 0; band < num_env_bands[bs_freq_res[ch][env]]; band++)		Note 1
bs_data_env[ch][env][band] = sbr_huff_dec(t_huff, **bs_codeword**);	**1..18**	Note 2
}		
}		
}		

Note 1: num_env_bands[bs_freq_res[ch][env]] is derived from the header according to subclause 4.6.18.3 and is named **n**.
Note 2: sbr_huff_dec() is defined in Annex 4.A.6.1.

Table 4.65A – Syntax of sbr_noise()

Syntax	No. of bits	Mnemonic
sbr_noise(ch,bs_coupling) {		
if (bs_coupling) {		
if (ch) {		
t_huff = t_huffman_noise_bal_3_0dB;		
f_huff = f_huffman_noise_bal_3_0dB;		
} else {		
t_huff = t_huffman_noise_3_0dB;		
f_huff = f_huffman_noise_3_0dB;		
}		
} else {		
t_huff = t_huffman_noise_3_0dB;		
f_huff = f_huffman_noise_3_0dB;		
}		
for (noise = 0; noise < bs_num_noise[ch]; noise++) {		
if (bs_df_noise[ch][noise] == 0) {		
if (bs_coupling && ch)		
bs_data_noise[ch][noise][0] = **bs_noise_start_value_balance**;	**5**	**uimsbf**
else		
bs_data_noise[ch][noise][0] = **bs_noise_start_value_level**;	**5**	**uimsbf**
for (band = 1; band < num_noise_bands[ch]; band++)		Note 1
bs_data_noise[ch][noise][band] = sbr_huff_dec(f_huff,**bs_codeword**);	**1..18**	Note 2
} else {		
for (band = 0; band < num_noise_bands[ch]; band++)		Note 1
bs_data_noise[ch][noise][band] = sbr_huff_dec(t_huff,**bs_codeword**);	**1..18**	Note 2
}		
}		
}		
Note 1: num_noise_bands[ch] is derived from the header according to subclause 4.6.18.3 and is named N_Q.		
Note 2: sbr_huff_dec() is defined in Annex 4.A.6.1.		

Table 4.66A – Syntax of sbr_sinusoidal_coding()

Syntax	No. of bits	Mnemonic
sbr_sinusoidal_coding(ch)		
{		
for (n = 0; n<num_high_res[ch]; n++)		Note 1
bs_add_harmonic[ch][n]	**1**	
}		
Note 1: num_high_res[ch] is derived from the header according to subclause 4.6.18.3 and is named N_{High}.		

”

In Part 3: Audio, Subpart 4, Subclause 4.5.2.1.6 Fill element (FIL), replace defined values of the extension_type field in Table 4.59:

“

Table 4.59 – Values of the extension_type field

Symbol	Value of extension_type	Purpose
EXT_FILL	‘0000’	bitstream filler
EXT_FILL_DATA	‘0001’	bitstream data as filler
EXT_DATA_ELEMENT	’0010‘	data element
EXT_DYNAMIC_RANGE	‘1011’	dynamic range control
EXT_SBR_DATA	‘1101’	SBR enhancement
EXT_SBR_DATA_CRC	‘1110’	SBR enhancement with CRC
-	all other values	reserved

”

And add the following restriction:

“

Fill elements containing an extension_payload with an extension_type of EXT_SBR_DATA or EXT_SBR_DATA_CRC shall not contain any other extension_payload of any other extension_type. For fill elements containing an extension_payload with an extension_type of EXT_SBR_DATA or EXT_SBR_DATA_CRC, the fill_element count field shall be set equal to the total length in bytes, including the SBR enhancement data plus the extension_type field.

”

In Part 3: Audio, Subpart 4, subclause 4.5.2.4.1 (Error sensitivity category assignment), replace table 4.64 with:

“

Table 4.64 – Error sensitivity category assignment

category	payload	mandatory	leads / may lead to one instance per	Description
0	main	yes	CPE	commonly used side information
1	main	yes	ICS	channel dependent side information
2	main	no	ICS	error resilient scale factor data
3	main	no	ICS	TNS data
4	main	yes	ICS	spectral data
5	extended	no	EPL	extension type / data_element_version
6	extended	no	EPL	DRC data
7	extended	no	EPL	bit stuffing
8	extended	no	EPL	ANC data
9	extended	no	EPL	SBR data

”

In Part 3: Audio, Subpart 4, subclause 4.5.2 Decoding of the GA bitstream payloads, add the following subclause 4.5.2.8 Payloads for the audio object type SBR:

“

4.5.2.8 Payloads for the audio object type SBR

4.5.2.8.1 Definitions

bs_reserved Bits reserved for future use, default value is zero.

sbr_extension_data() Syntactic element that contains the SBR extension data.

bs_sbr_crc_bits Cyclic redundancy checksum for the SBR extension data. The CRC code is defined by the generator polynomial $G_{10}(x) = x^{10} + x^9 + x^5 + x^4 + x + 1$ and the initial value for the CRC calculation is zero.

bs_header_flag Indicates if an SBR header is present.

sbr_header() Syntactic element that contains the SBR header.

sbr_data() Syntactic element that contains the SBR data.

bs_fill_bits Byte alignment bits.

bs_amp_res Defines the resolution of the envelope estimates according to:

Table 4.70A – bs_amp_res

bs_amp_res	Meaning
0	1.5 dB
1	3.0 dB

bs_start_freq Input parameter to function that calculates start of master frequency table.

bs_stop_freq Input parameter to function that calculates stop of master frequency table.

bs_xover_band Index to master frequency table.

bs_header_extra_1 Indicates if the optional header part 1 is present.

bs_header_extra_2 Indicates if the optional header part 2 is present.

bs_freq_scale Input parameter to function that calculates master frequency table, defined by:

Table 4.71A – bs_freq_scale

bs_freq_scale	Meaning
0	linear
1	12 bands/octave
2 (default)	10 bands/octave
3	8 bands/octave

bs_alter_scale Input parameter to function that calculates master frequency table, defined by:

Table 4.72A – bs_alter_scale

bs_alter_scale	**Meaning** for bs_freq_scale = 0	**Meaning** for bs_freq_scale > 0
0	no grouping of channels	no alteration
1 (default)	groups of 2 channels	extra wide bands in highest range

bs_noise_bands Input parameter to function that calculates noise band table, defined by:

Table 4.73A – bs_noise_bands

bs_noise_bands	Meaning
0	1 band
1	1 band/octave
2 (default)	2 bands/octave
3	3 bands/octave

bs_limiter_bands Input parameter to function that calculates limiter band table, defined by:

Table 4.74 – bs_limiter_bands

bs_limiter_bands	Meaning	Note
0	1 band	single band
1	1.2 bands/octave	multi-band
2 (default)	2.0 bands/octave	multi-band
3	3.0 bands/octave	multi-band

bs_limiter_gains Defines the maximum gain of the limiters according to:

Table 4.75A – bs_limiter_gains

bs_limiter_gains	Meaning
0	-3 dB Max Gain
1	0 dB Max Gain
2 (default)	3 dB Max Gain
3	inf. dB Max Gain(i.e. limiter off)

bs_interpol_freq Defines if the frequency interpolation shall be applied according to:

Table 4.76A – bs_interpol_freq

bs_interpol_freq	Meaning
0	off
1 (default)	on

bs_smoothing_mode Defines if smoothing shall be applied according to:

Table 4.77A – bs_smoothing_mode

bs_smoothing_mode	Meaning
0	on
1 (default)	off

sbr_single_channel_element() Syntactic element that contains data for an SBR single channel element.

sbr_channel_pair_element() Syntactic element that contains data for an SBR channel pair element.

sbr_channel_pair_base_element() Syntactic element that contains base-layer data for an SBR channel pair element, in a scalable system.

sbr_channel_pair_enhance_element() Syntactic element that contains enhancement-layer data for an SBR channel pair element, in a scalable system.

bs_data_extra Indicates if reserved bits in the SBR data part are present.

sbr_grid() Syntactic element that contains the time frequency grid.

sbr_dtdf() Syntactic element that contains the information on how the envelope and noise data is delta coded.

sbr_invf() Syntactic element that contains the inverse filtering data.

sbr_envelope() Syntactic element that contains the huffman coded envelope data.

sbr_noise() Syntactic element that contains the huffman coded noise floor data.

bs_add_harmonic_flag Indicates if sinusoidal coding information is present.

sbr_sinusoidal_coding() Syntactic element that contains sinusoidal coding data.

bs_extended_data Indicates if an SBR extended data element is present.

bs_extension_size Defines the size of the SBR extended date element in bytes.

bs_esc_count Further defines the size of the SBR extended data element in cases where the size is larger than 14 bytes.

bs_extension_id Defines the ID of the SBR extended data element, for future use, according to:

Table 4.78A – bs_extension_id

bs_extension_id	Meaning
0	reserved
1	reserved
2	reserved
3	reserved

sbr_extension() Syntactic element for future extensions.

bs_coupling Indicates if the stereo information between the two channels is coupled or not according to:

Table 4.79A – bs_coupling

bs_coupling	Meaning
0	the channels are not coupled
1	the channels are coupled

bs_frame_class Indicates the frame class of the current SBR frame according to:

Table 4.80A – bs_frame_class

bs_frame_class	Meaning
0	FIXFIX
1	FIXVAR
2	VARFIX
3	VARVAR

tmp Helper variable used in sbr_grid().

bs_num_env Indicates the number of SBR envelopes in the current SBR frame.

bs_freq_res Indicates the frequency resolution for each channel and SBR envelope according to:

Table 4.81A – bs_freq_res vector element

bs_freq_res[]	Meaning
0	low frequency resolution
1	high frequency resolution

bs_pointer Pointer to a specific border.

bs_var_bord_0 Indicates the position of the leading variable border for class VARVAR and VARFIX.

bs_var_bord_1 Indicates the position of the trailing variable border for class VARVAR and FIXVAR.

bs_num_rel_0 Indicates number of relative borders starting from bs_var_bord_0.

bs_num_rel_1 Indicates number of relative borders starting from bs_var_bord_1.

bs_rel_bord_0 Indicates the lengths of the relative borders starting from bs_var_bord_0.

bs_rel_bord_1 Indicates the lengths of the relative borders starting from bs_var_bord_1.

bs_df_env Indicates delta coding direction for each SBR envelope according to:

Table 4.82A – bs_df_env vector element

bs_df_env[]	Meaning
0	apply delta decoding in frequency direction for the indicated frequency band
1	apply delta decoding in time direction for the indicated frequency band

bs_df_noise Indicates delta coding direction for each noise floor according to:

Table 4.83A – bs_df_noise vector element

bs_df_noise[]	Meaning
0	apply delta decoding in frequency direction for the indicated frequency band
1	apply delta decoding in time direction for the indicated frequency band

bs_invf_mode Indicates the level of inverse filtering for each frequency band according to:

Table 4.84A – bs_invf_mode vector element:

bs_invf_mode[]	Meaning
0	no inverse filtering
1	low level inverse filtering
2	intermediate inverse filtering
3	strong inverse filtering

bs_env_start_value_balance Holds the first envelope scalefactor in case of a coupled stereo bitstream.
bs_data_env Holds the raw envelope scalefactors for each channel, SBR envelope and band.
bs_env_start_value_level Holds the first envelope scalefactor in case of a non-coupled stereo or a mono bitstream.
sbr_huff_dec() Huffman decoder defined in Annex 4.A.6
bs_codeword Huffman code word.
bs_noise_start_value_balance Holds the first noise floor value in case of a coupled stereo bitstream.
bs_data_noise Holds the raw noise floor data for each envelope and band.
bs_noise_start_value_level Holds the first noise floor value in case of a non-coupled stereo or a mono bitstream.
bs_add_harmonic Indicates if a sinusoidal should be added to a specific frequency band according to:

Table 4.85A – bs_add_harmonic vector element

bs_add_harmonic[]	Meaning
0	do not add a sinusoidal to the indicated frequency band
1	add a sinusoidal to the indicated frequency band

4.5.2.8.2 Decoding process

4.5.2.8.2.1 SBR Frame Overview

An overview of the contents of the two possible SBR extension data elements is given in Figure 4.19A below.

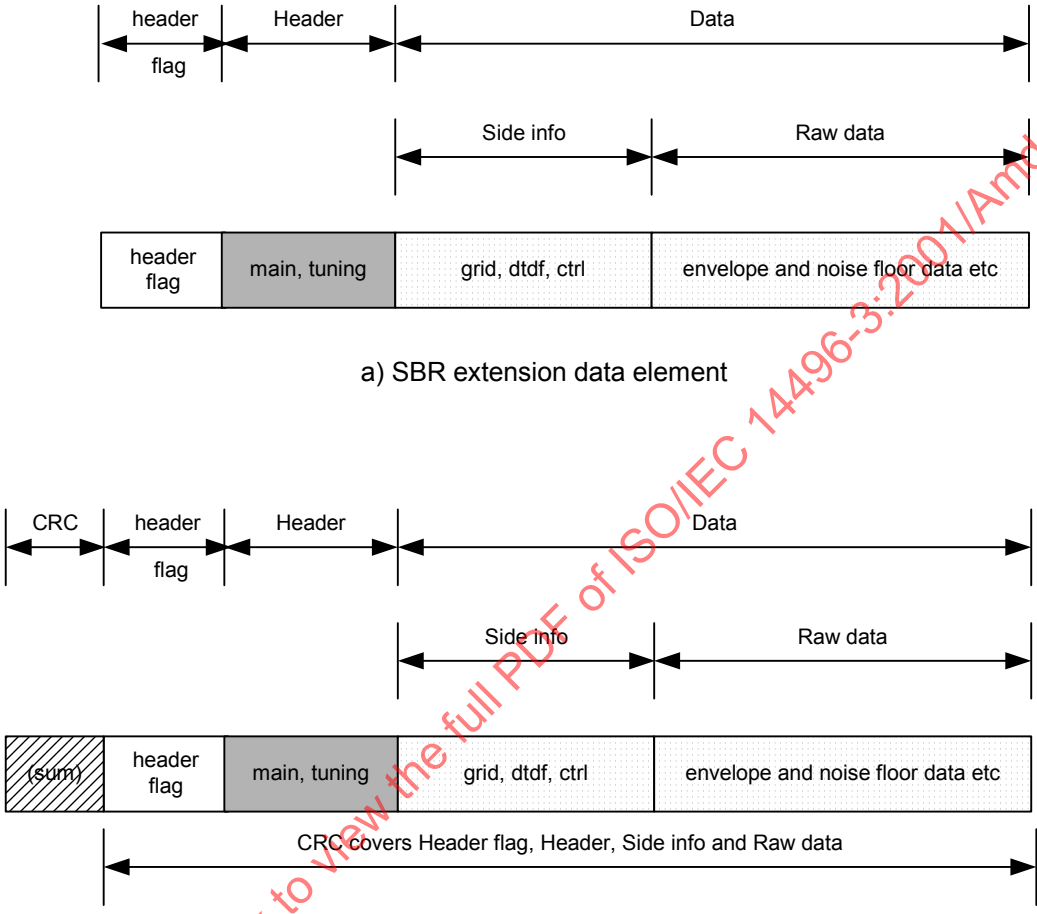


Figure 4.19A – The basic sections of the SBR extension data elements

The CRC field (if applicable) holds a Cyclic Redundancy Code checksum of 10 bit length. The checksum shall be calculated covering the whole SBR data range including possible bs_fill_bits.

The bs_header_flag, if set, indicates that an SBR header part is present. The SBR header part contains fundamental information such as SBR frequency range (denoted as main in Figure 4.19), as well as control signals that do not require frequent changes (denoted as tuning). Prior to SBR decoding, a SBR header part must be present. As long as no SBR header part is present, the SBR decoder performs upsampling and delay adjustment only. In continuous broadcast applications, SBR extension data elements with an SBR header part are typically sent twice per second. In addition, a SBR header part can any time be inserted, if an instantaneous, possibly program dependent, change of header parameters is required.

The SBR data part can be subdivided into side info and raw data, where side info is defined as signals needed to decode the raw data and some decoder tuning signals. Raw data consists of Huffman coded envelope scalefactors and noise floor estimates. The grid part describes how the current SBR frame is subdivided in time into time segments, and the frequency resolution of those time segments. The dtdf part signals how the data is encoded (delta coding in time or frequency direction). Channel configuration issues and decoding procedures are discussed in detail in subclause 4.6.18.3

4.5.2.8.2.2 SBR Extension Payload for the Audio Object Types AAC main, AAC SSR, AAC LC and AAC LTP

One SBR fill element is used per AAC syntactic element that is to be enhanced by SBR. SBR elements are inserted into the raw_data_block() after the corresponding AAC elements. Each AAC SCE, CPE or independently switched CCE must be succeeded by a corresponding SBR element. LFE elements are decoded according to standard AAC procedures but must be delay adjusted and re-sampled to match the output sample rate. Given below is an example of the structure of syntactic elements within a raw data block of a 5.1 (multi-channel) configuration, where SBR is used without a CRC check.

```
<SCE> <FIL <EXT_SBR_DATA(SCE)>>     // center
<CPE> <FIL <EXT_SBR_DATA(CPE)>>     // front L/R
<CPE> <FIL <EXT_SBR_DATA(CPE)>>     // back L/R
<LFE>                               // sub
<END>                               // (end of raw data block)
```

The time domain mix of an independently switched CCE is done after SBR decoding. A dependently switched CCE is first added to the target SCE or CPE channels and SBR is applied after this addition.

4.5.2.8.2.3 SBR Extension Payload for the Audio Object Types ER AAC LC and ER AAC LTP

The number and the order of the SBR extension data elements (if present) is given by the channelConfiguration. To each SCE or CPE in one er_raw_data_block(), there is a corresponding SBR extension_payload() containing either sbr_extension_data(ID_SCE) or sbr_extension_data(ID_CPE). There is no SBR extension_payload() for LFE. LFE elements are decoded according to standard AAC procedures but must be delay adjusted and re-sampled to match the output sample rate. Only SBR extension data elements without CRC check are allowed for the audio object types ER AAC LC and ER AAC LTP. The extension payload shall not include both DRC extension elements and SBR extension elements simultaneously. If SBR extension elements are used, DRC extension elements are prohibited. Given below is an example of the structure of syntactic elements for channelConfiguration 6.

```
<SCE> <CPE> <CPE> <LFE> <EXT <SBR(SCE)> <SBR(CPE)> <SBR(CPE)>>
```

4.5.2.8.2.4 SBR Extension Payload for the Audio Object Types AAC Scalable and ER AAC Scalable

Bandwidth scalability and mono/stereo scalability of the AAC core coder is supported by the SBR bandwidth extension tool. For core coder bandwidth scalability, the SBR data is transmitted in the lowest AAC core coder layer and the SBR data covers the largest SBR frequency range used in the scalable system. The start frequency transmitted for the SBR frequency range is set to the lowest upper frequency of the AAC core coder frequency range. Hence, such a core coder bandwidth scalable system with SBR provides a constant audio bandwidth for all layers. If only the lowest layer is available for decoding, the high frequency range is covered by SBR. If higher layers are available, the high frequency range is partly (or even completely) covered by AAC, and the SBR signal is only partly needed for the remaining upper part of the high frequency range that is not covered by AAC.

The scalable SBR data is embedded into the MPEG-4 stream in the same way as for non-scalable SBR data elements, by means of using the extension_payload(). However, only the lowest layer, or, in case mono/stereo scalability is chosen, in addition also the lowest stereo layer, carries an SBR data element. The different types of scalable SBR layers for all possible configurations of bandwidth scalability and mono/stereo scalability of the AAC core coder are described in Table 4.86.

Table 4.86A – SBR data in scalable SBR layers

Description	**sbr_layer**	**sbr_extension_data() in extension_playload()**
Non-scalable AAC core coder	SBR_NOT_SCALABLE	yes
First layer of scalable AAC core coder (mono)	SBR_MONO_BASE	yes
First stereo AAC layer in a mono/stereo scalable AAC core coder configuration	SBR_STEREO_ENHANCE	yes
First layer of scalable AAC core coder (stereo)	SBR_STEREO_BASE	yes
All other layers of a scalable AAC core coder	SBR_NO_DATA	no

”

In Part 3: Audio, Subpart 4, subclause 4.5.4 (Tables), add the following entries to table 4.93:

“

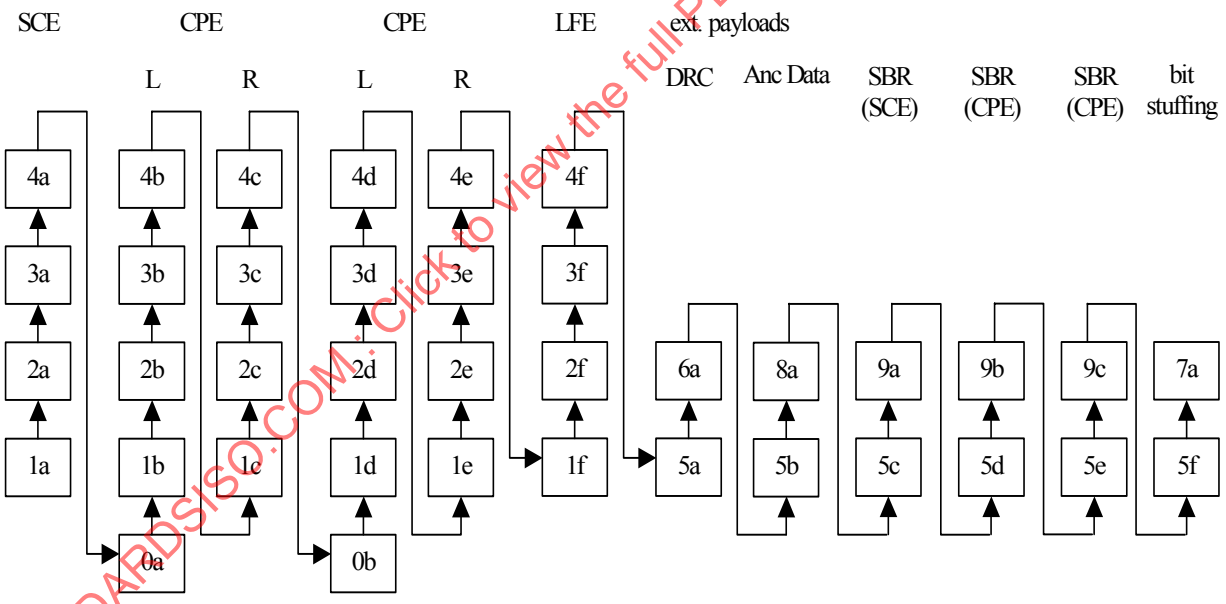
9	bs_sbr_crc_bits	sbr_extension_data()
9	bs_header_flag	sbr_extension_data()
9	bs_fill_bits	sbr_extension_data()
9	bs_amp_res	sbr_header()
9	bs_start_freq	sbr_header()
9	bs_stop_freq	sbr_header()
9	bs_xover_band	sbr_header()
9	bs_reserved	sbr_header()
9	bs_header_extra_1	sbr_header()
9	bs_header_extra_2	sbr_header()
9	bs_freq_scale	sbr_header()
9	bs_alter_scale	sbr_header()
9	bs_noise_bands	sbr_header()
9	bs_limiter_bands	sbr_header()
9	bs_limiter_gains	sbr_header()
9	bs_interpol_freq	sbr_header()
9	bs_smoothing_mode	sbr_header()
9	bs_data_extra	sbr_single_channel_element()
9	bs_reserved	sbr_single_channel_element()
9	bs_add_harmonic_flag	sbr_single_channel_element()
9	bs_extended_data	sbr_single_channel_element()
9	bs_extension_size	sbr_single_channel_element()
9	bs_esc_count	sbr_single_channel_element()
9	bs_extension_id	sbr_single_channel_element()
9	bs_data_extra	sbr_channel_pair_element()
9	bs_reserved	sbr_channel_pair_element()
9	bs_coupling	sbr_channel_pair_element()
9	bs_add_harmonic_flag	sbr_channel_pair_element()
9	bs_extended_data	sbr_channel_pair_element()
9	bs_extension_size	sbr_channel_pair_element()
9	bs_esc_count	sbr_channel_pair_element()

9	bs_extension_id	sbr_channel_pair_element()
9	bs_frame_class	sbr_grid()
9	tmp	sbr_grid()
9	bs_freq_res	sbr_grid()
9	bs_pointer	sbr_grid()
9	bs_var_bord_0	sbr_grid()
9	bs_var_bord_1	sbr_grid()
9	bs_num_rel_0	sbr_grid()
9	bs_num_rel_1	sbr_grid()
9	bs_df_env	sbr_dtdf()
9	bs_df_noise	sbr_dtdf()
9	bs_invf_mode	sbr_invf()
9	bs_env_start_value_balance	sbr_envelope()
9	bs_env_start_value_level	sbr_envelope()
9	bs_codeword	sbr_envelope()
9	bs_noise_start_value_balance	sbr_noise()
9	bs_noise_start_value_level	sbr_noise()
9	bs_codeword	sbr_noise()
9	bs_add_harmonic	sbr_sinusoidal_coding()

”

In Part 3: Audio, Subpart 4, subclause 4.5.5 (Figures), replace figure 4.23 with:

“



”

In Part 3: Audio, Subpart 4, subclause 4.6 GA-Tool Descriptions, after 4.6.17 Low delay codec, add the following clause:

“

4.6.18 SBR Tool

4.6.18.1 Tool Description

The human voice and musical instruments generate either quasi-stationary excitation signals that emerge from oscillating systems or signals originated from different noise sources. A wide-band excitation spectrum could be initialized by one or by a set of several sources, e.g. vocal cords, strings, and reeds etc. They all have different frequency components depending on the source. The excitation signals are subsequently filtered by resonators such as the vocal tract, violin body etc, giving the voice or musical instrument its characteristic tone color or timbre. A bandwidth limitation of such a signal is equivalent to a truncation of the sequence of harmonics. Such a truncation alters the perceived timbre and the audio signal sounds “muffled” or “dull”, and particularly for speech the intelligibility may be reduced.

The SBR tool (Spectral Band Replication) extends the audio bandwidth of the decoded bandwidth-limited audio signal. The process is based on replication of the sequences of harmonics, previously truncated in order to reduce data rate, from the available bandwidth limited signal and control data obtained from the encoder. The ratio between tonal and noise-like components is maintained by adaptive inverse filtering as well as optional addition of noise and sinusoidals.

4.6.18.2 Definitions

4.6.18.2.1 SBR specific definitions

For the purpose of this document, the following terms and definitions apply.

4.6.18.2.1.1
band
(as in limiter band, noise floor band, etc.) a group of consecutive QMF subbands

4.6.18.2.1.2
chirp factor
the bandwidth expansion factor of the formants described by a LPC polynomial

4.6.18.2.1.3
Down Sampled SBR
the SBR Tool with a modified synthesis filterbank resulting in a down sampled output signal with the same sample rate as the input signal to the SBR Tool. May be used whenever a lower sample rate output is desired.

4.6.18.2.1.4
envelope scalefactor
an element representing the averaged energy of a signal over a region described by a frequency band and a time segment

4.6.18.2.1.5
frequency band
interval in frequency, group of consecutive QMF subbands

4.6.18.2.1.6
frequency border
frequency band delimiter, expressed as a specific QMF subband

4.6.18.2.1.7
NA
Not Applicable

4.6.18.2.1.8
noise floor
a vector of noise floor scalefactors

4.6.18.2.1.9
noise floor scalefactor
an element associated with a region described by a frequency band and a time segment, representing the ratio between the energy of the noise to be added to the envelope adjusted HF generated signal and the energy of the same

4.6.18.2.1.10
patch
a number of adjoining QMF subbands moved to a different frequency location

4.6.18.2.1.11
QMF
Quadrature Mirror Filter

4.6.18.2.1.12
SBR
Spectral Band Replication

4.6.18.2.1.13
SBR envelope
a vector of envelope scalefactors

4.6.18.2.1.14
SBR frame
time segment associated with one SBR extension data element

4.6.18.2.1.15
SBR range
the frequency range of the signal generated by the SBR algorithm

4.6.18.2.1.16
subband
a frequency range represented by one row in a QMF matrix, carrying a subsampled signal

4.6.18.2.1.17
time border
time segment delimiter, expressed as a specific time slot

4.6.18.2.1.18
time segment
interval in time, group of consecutive time slots

4.6.18.2.1.19
time / frequency grid
a description of SBR envelope time segments and associated frequency resolution tables as well as description of noise floor time segments

4.6.18.2.1.20
time slot
finest resolution in time for SBR envelopes and noise floors. One time slot equals two subsamples in the QMF domain

4.6.18.2.2 SBR specific notation

The description of the SBR tool uses the following notation:

- Vectors are indicated by bold lower-case names, e.g. **vector**.
- Matrices (and vectors of vectors) are indicated by bold upper-case single letter names, e.g. **M**.
- Variables are indicated by italic, e.g. *variable*.
- Functions are indicated as *func(x)*.
- Bitstream elements are indicated as multiple-word names with prefix "bs_", e.g. bs_bitstream_element.

For equations in the text, normal mathematical interpretation is assumed (no rounding or truncation unless explicitly stated). Hence the following example from the text,

$$\mathbf{Q}_{Mapped}(m-lsb,l)=\mathbf{Q}_{Orig}\left(i,k(l)\right),\mathbf{f}_{TableNoise}(i)\le m<\mathbf{f}_{TableNoise}(i+1),0\le i<N_Q,0\le l<L_E$$

where $k(l)$ is defined by $\mathbf{t}(l)\ge\mathbf{t}_Q\left(k(l)\right),\mathbf{t}(l+1)\le\mathbf{t}_Q\left(k(l)+1\right)$

should be interpreted as follows. $\mathbf{Q}_{Mapped}(m-lsb,l)$ equals $\mathbf{Q}_{Orig}\left(i,k(l)\right)$ for $\mathbf{f}_{TableNoise}(i)\le m<\mathbf{f}_{TableNoise}(i+1)$ and $0\le i<N_Q$ and $0\le l<L_E$. The function $k(l)$ returns, for a given l, a value for which $\mathbf{t}(l)\ge\mathbf{t}_Q\left(k(l)\right)$ and $\mathbf{t}(l+1)\le\mathbf{t}_Q\left(k(l)+1\right)$ is true. The result is a $\mathbf{Q}_{Mapped}$ matrix that is piecewise constant.

The expression $\sum_{k=a}^{b} f(k)$ evaluates to zero if $b<a$.

For flowcharts, normal pseudo-code interpretation is assumed, with no rounding or truncation unless explicitly stated.

4.6.18.2.3 Scalar operations

X^* is the complex conjugate of X

4.6.18.2.4 Vector operations

$\mathbf{y}=sort(\mathbf{x})$. $\mathbf{y}$ is equal to the sorted vector $\mathbf{x}$, where the elements of $\mathbf{x}$ are sorted in ascending order.

$y=length(\mathbf{x})$. y is the number of elements of the vector $\mathbf{x}$.

$y=ceil(x)$ represents rounding to the nearest integer towards infinity.

4.6.18.2.5 Constants

ε	A constant to avoid division by zero, e.g. 96 dB below maximum signal input.
$HI = 1$	Index used for SBR envelope high frequency resolution.
$LO = 0$	Index used for SBR envelope low frequency resolution.
$NOISE\_FLOOR\_OFFSET = 6$	Offset of noise floor.
$RATE = 2$	A constant indicating the number of QMF subband samples per timeslot.

4.6.18.2.6 Variables

Description of variables defined in one subclause and used in other subclauses.

ch	is the current channel.
$bsco$	bandwidth scalable codec offset, indicates for a bandwidth scalable system the number of QMF subbands above k_x in the SBR range that should be replaced by AAC data from a bandwidth scalable enhancement layer. If a bandwidth scalable core coder is not used, the variable is zero.
$\mathbf{E}_{Orig}$	has L_E columns where each column is of length N_{Low} or N_{High} depending on the frequency resolution for each SBR envelope. The elements in $\mathbf{E}_{Orig}$ contains the envelope scalefactors of the original signal.
$\mathbf{F} = \left[\mathbf{f}_{TableLow}, \mathbf{f}_{TableHigh}\right]$	has two column vectors containing the frequency border tables for low and high frequency resolution.
Fs_{SBR}	internal sampling frequency of the SBR Tool, twice the sampling frequency of the core coder (after sampling frequency mapping, Table 4.55). The sampling frequency of the SBR enhanced output signal is equal to the internal sampling frequency of the SBR Tool, unless the SBR Tool is operated in downsampled mode. If the SBR Tool is operated in downsampled mode, the output sampling frequency is equal to the sampling frequency of the core coder.
$\mathbf{f}_{Master}$	is of length $N_{Master}+1$ and contains QMF master frequency grouping information.
$\mathbf{f}_{TableHigh}$	is of length $N_{High}+1$ and contains frequency borders for high frequency resolution SBR envelopes.
$\mathbf{f}_{TableLim}$	is of length N_L+1 and contains frequency borders used by the limiter.
$\mathbf{f}_{TableLow}$	is of length $N_{Low}+1$ and contains frequency borders for low frequency resolution SBR envelopes.
$\mathbf{f}_{TableNoise}$	is of length N_Q+1 and contains frequency borders used by noise floors.
k_x	the first QMF subband in the SBR range.
k_0	the first QMF subband in the $\mathbf{f}_{Master}$ table.
L_E	number of SBR envelopes.
L_Q	number of noise floors.
M	number of QMF subbands in the SBR range.
$middleBorder$	points to a specific time border.
N_L	number of limiter bands.
N_{Master}	number of frequency bands in the master frequency resolution table.

N_Q	number of noise floor bands.
$\mathbf{n} = [N_{Low}, N_{High}]$	number of frequency bands for low and high frequency resolution.
numPatches	a variable indicating the number of patches in the SBR range.
numTimeSlots	number of SBR envelope time slots that exist within an AAC frame, 16 for a 1024 AAC frame and 15 for a 960 AAC frame.
$\mathbf{panOffset} = [24, 12]$	offset-values for the SBR envelope and noise floor data, when using coupled channels.
patchBorders	a vector containing the frequency borders of the patches.
patchNumSubbands	a vector holding the number of subbands in every patch.
$\mathbf{Q}_{Orig}$	has L_Q columns where each column is of length N_Q and contains the noise floor scalefactors.
$\mathbf{r} = [r_0, ..., r_{L-1}]$	frequency resolution for all SBR envelopes in the current SBR frame, zero for low resolution, one for high resolution.
reset	a variable in the encoder and the decoder set to one if certain bitstream elements have changed from the previous SBR frame, otherwise set to zero.
$\mathbf{t}_E$	is of length L_E+1 and contains start and stop time borders for all SBR envelopes in the current SBR frame.
t_{HFAdj}	offset for the envelope adjuster module.
t_{HFGen}	offset for the HF-generation module.
$\mathbf{t}_Q$	is of length L_Q+1 and contains start and stop time borders for all noise floors in the current SBR frame.
$\mathbf{X}_{High}$	is the complex input QMF bank subband matrix to the HF adjuster.
$\mathbf{X}_{Low}$	is the complex input QMF bank subband matrix to the HF generator.
$\mathbf{Y}$	is the complex output QMF bank subband matrix from the HF adjuster.

4.6.18.3 Decoding Process

4.6.18.3.1 Introduction

SBR incorporates adaptive time and frequency resolution for the envelope coding and adjustment. The adaptation is obtained by flexible grouping of QMF subband samples in time and frequency. For each such group, a corresponding scalefactor is calculated and transmitted. The subclause 4.6.18.3 describes how to recreate the time and frequency grouping chosen by the encoder. Furthermore, it shows how delta coded SBR envelopes and noise floors are decoded. The decoding process is outlined for one single channel element as well as for one channel pair element. For a single channel element, the channel number is denoted zero. For channel pair elements, the two channels are indexed zero and one, where zero represents the firstly decoded channel data in the channel pair element and one represents the secondly decoded channel in the channel pair element. The system is reset ($reset = 1$) if the value of any of the following bitstream elements in the SBR header differs from that of the previous SBR frame:

- *bs_start_freq*
- *bs_stop_freq*
- *bs_freq_scale*
- *bs_alter_scale*
- *bs_xover_band*
- *bs_noise_bands*

4.6.18.3.2 Frequency Band Tables

The grouping of QMF subband samples in frequency is described by frequency band tables. The tables are defined by functions, most arguments of which are transmitted in the SBR header. For each SBR envelope, two frequency band tables are available, a high frequency resolution table, $\mathbf{f}_{TableHigh}$, and a low frequency resolution table, $\mathbf{f}_{TableLow}$. The noise floor and the limiter also have corresponding frequency band tables, $\mathbf{f}_{TableNoise}$ and $\mathbf{f}_{TableLim}$. All aforementioned tables are derived from one master frequency band table, $\mathbf{f}_{Master}$. The frequency band tables contain the frequency borders for each frequency band, represented as QMF subbands. Each frequency band is defined by a start frequency border and a stop frequency border. The QMF subband indicated by the start frequency border is included in the frequency band, the QMF subband indicated by the stop frequency border is excluded from the frequency band. For all tables derived from $\mathbf{f}_{Master}$ the stop frequency border of band n equals the start frequency border of band $n+1$, where n is an arbitrary frequency band in the table. The following subclauses describe how to calculate $\mathbf{f}_{Master}$, $\mathbf{f}_{TableHigh}$, $\mathbf{f}_{TableLow}$, $\mathbf{f}_{TableNoise}$ and $\mathbf{f}_{TableLim}$.

4.6.18.3.2.1 Master Frequency Band Table

In order to build the master frequency band table, the QMF subbands representing the boundaries of the table must first be calculated. The subband representing the lower frequency boundary of the table is denoted k_0, and is defined by:

$$k_0 = startMin + \mathbf{offset}\left(bs\_start\_freq\right),$$

where

$$\mathbf{offset} = \begin{cases} [-8,-7,-6,-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7] & , Fs_{SBR} = 16000 \\ [-5,-4,-3,-2,-1,0,1,2,3,4,5,6,7,9,11,13] & , Fs_{SBR} = 22050 \\ [-5,-3,-2,-1,0,1,2,3,4,5,6,7,9,11,13,16] & , Fs_{SBR} = 24000 \\ [-6,-4,-2,-1,0,1,2,3,4,5,6,7,9,11,13,16] & , Fs_{SBR} = 32000 \\ [-4,-2,-1,0,1,2,3,4,5,6,7,9,11,13,16,20] & , 44100 \le Fs_{SBR} \le 64000 \\ [-2,-1,0,1,2,3,4,5,6,7,9,11,13,16,20,24] & , Fs_{SBR} > 64000 \end{cases}$$

$$startMin = \begin{cases} NINT\left(3000 \cdot \frac{128}{Fs_{SBR}}\right) & , Fs_{SBR} < 32000 \\ NINT\left(4000 \cdot \frac{128}{Fs_{SBR}}\right) & , 32000 \le Fs_{SBR} < 64000 \\ NINT\left(5000 \cdot \frac{128}{Fs_{SBR}}\right) & , 64000 \le Fs_{SBR} \end{cases}$$

The upper frequency boundary, denoted k_2, is defined by:

$$k_2 = \begin{cases} \min\left(64, stopMin + \sum_{i=0}^{bs\_stop\_freq-1} \mathbf{stopDkSort}(i)\right) & , 0 \le bs\_stop\_freq < 14 \\ \min(64, 2 \cdot k_0) & , bs\_stop\_freq = 14 \\ \min(64, 3 \cdot k_0) & , bs\_stop\_freq = 15 \end{cases}$$

where

$$stopMin = \begin{cases} NINT\left(6000 \cdot \frac{128}{Fs_{SBR}}\right) & , Fs_{SBR} < 32000 \\ NINT\left(8000 \cdot \frac{128}{Fs_{SBR}}\right) & , 32000 \le Fs_{SBR} < 64000 \\ NINT\left(10000 \cdot \frac{128}{Fs_{SBR}}\right) & , 64000 \le Fs_{SBR} \end{cases}$$

$$\mathbf{stopDkSort} = sort(\mathbf{stopDk})$$

$$\mathbf{stopDk}(p) = NINT\left(stopMin \cdot \left(\frac{64}{stopMin}\right)^{\frac{p+1}{13}}\right) - NINT\left(stopMin \cdot \left(\frac{64}{stopMin}\right)^{\frac{p}{13}}\right), 0 \le p \le 12$$

The master frequency band table, $\mathbf{f}_{Master}$, is calculated according to the flowcharts in Figure 4.38 and Figure 4.39. The input variables to the flowcharts are k_0 *and* k_2, as calculated above, and the bitstream elements *bs_freq_scale* and *bs_alter_scale*. The table $\mathbf{f}_{Master}$ is only defined for $k_2 > k_0$. Furthermore, the restrictions in subclause 4.6.18.3.6 apply.

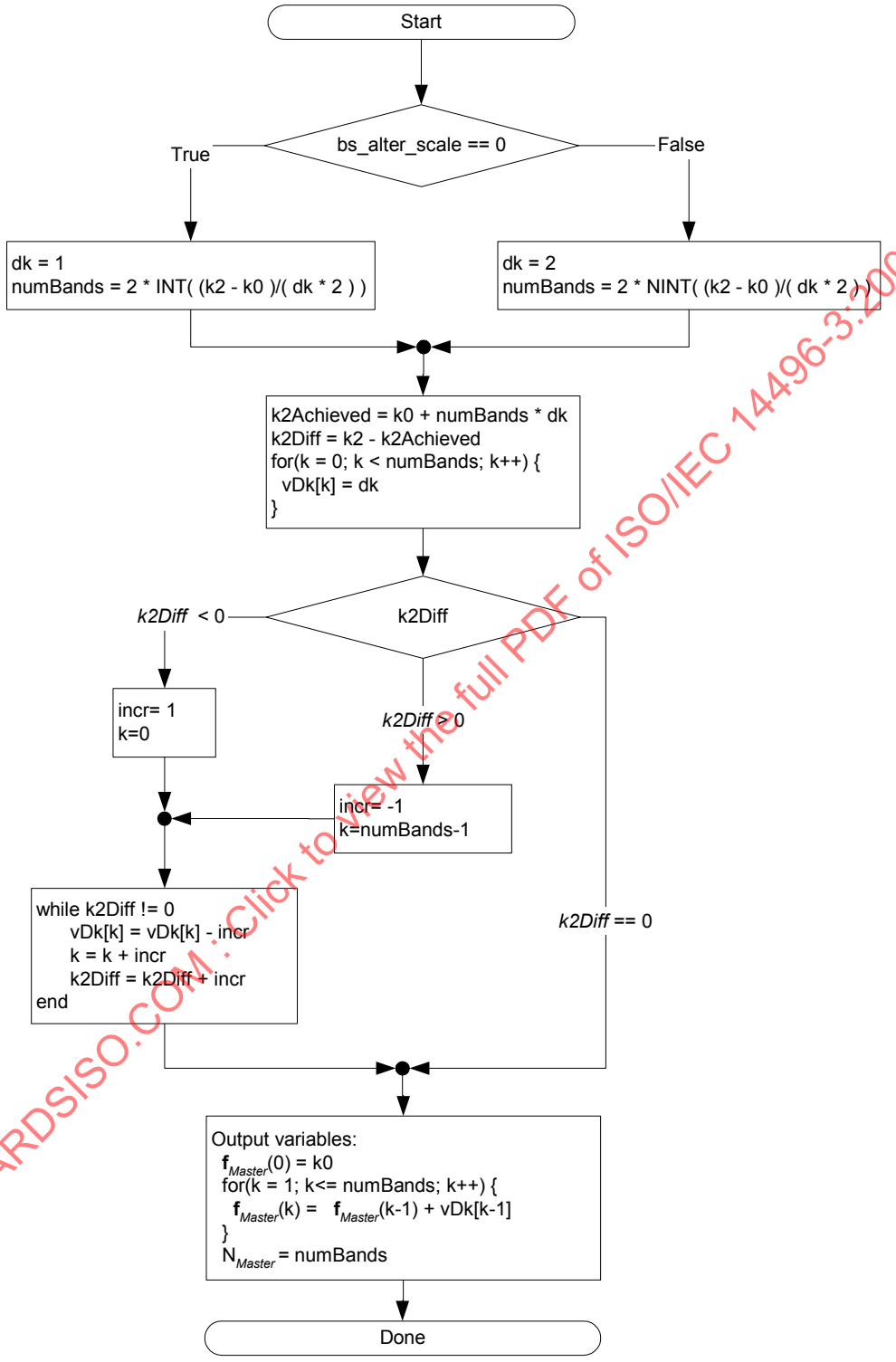


Figure 4.38 – Flowchart calculation of $\mathbf{f}_{Master}$ when *bs_freq_scale = 0*

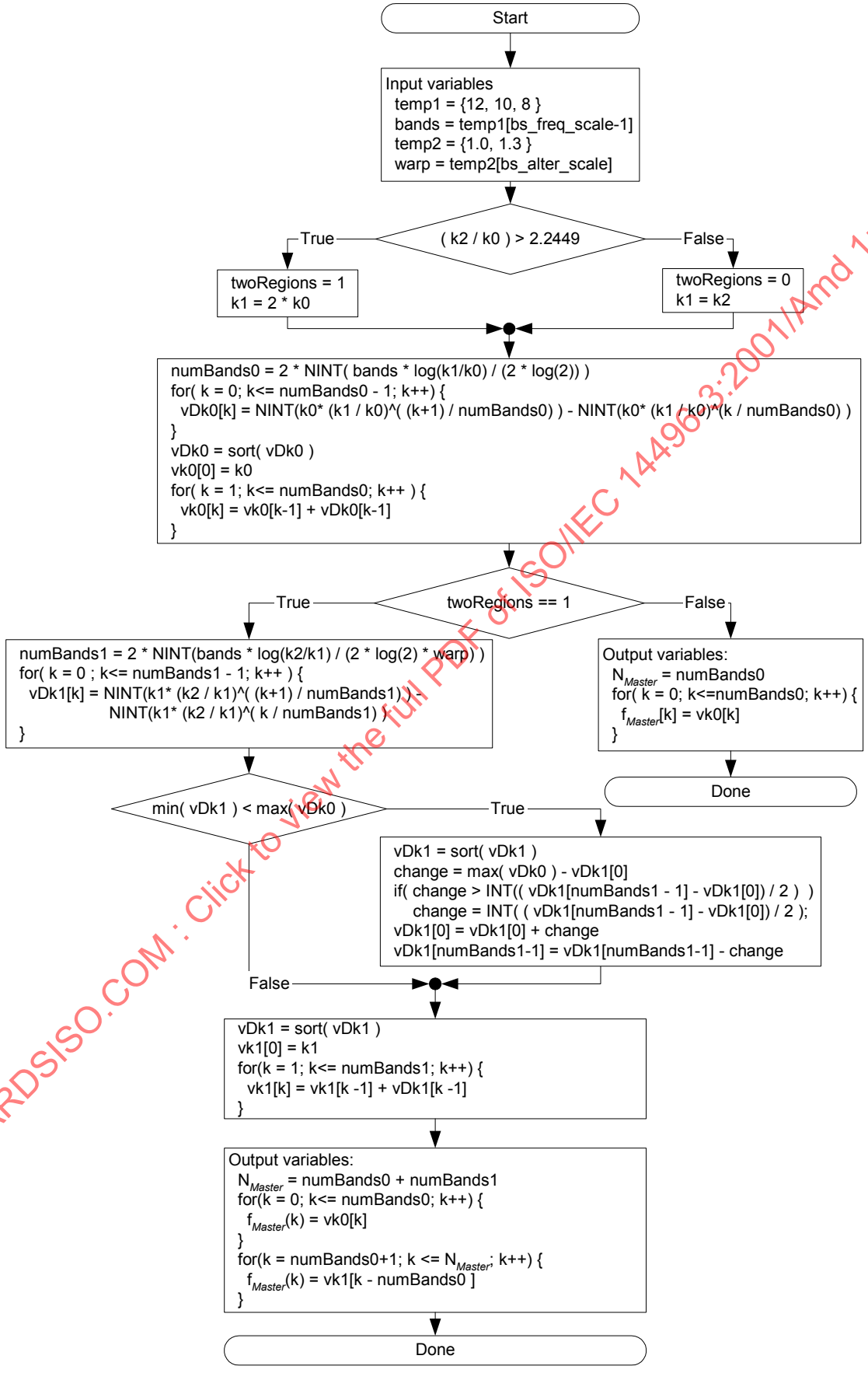


Figure 4.39 – Flowchart calculation of f_{Master} when *bs_freq_scale > 0*

4.6.18.3.2.2 Derived Frequency Band Tables

The frequency band table $\mathbf{f}_{TableHigh}$, used for high frequency resolution SBR envelopes, is obtained by extracting a subset of the borders from $\mathbf{f}_{Master}$ according to:

$$N_{High} = N_{Master} - bs\_xover\_band$$

$$N_{Low} = INT\left(\frac{N_{High}}{2}\right) + \left(N_{High} - 2 \cdot INT\left(\frac{N_{High}}{2}\right)\right)$$

$$\mathbf{n} = [N_{Low}, N_{High}]$$

$$\mathbf{f}_{TableHigh}(k) = \mathbf{f}_{Master}(k + bs\_xover\_band) \quad ,0 \le k \le N_{High}$$

$$M = \mathbf{f}_{TableHigh}(N_{high}) - \mathbf{f}_{TableHigh}(0)$$

$$k_x = \mathbf{f}_{TableHigh}(0)$$

The frequency band table $\mathbf{f}_{TableLow}$, used for low frequency resolution SBR envelopes, is obtained by extracting a subset of the borders from $\mathbf{f}_{TableHigh}$ according to:

$$\mathbf{f}_{TableLow}(k) = \mathbf{f}_{TableHigh}(i(k)) \quad ,0 \le k \le N_{Low}$$

where $i(k)$ is defined by

$$i(k) = \begin{cases} 0 & if\ k = 0 \\ 2 \cdot k - \dfrac{1-(-1)^{N_{High}}}{2} & if\ k \neq 0 \end{cases}.$$

The noise floor frequency band table $\mathbf{f}_{TableNoise}$, is extracted from $\mathbf{f}_{TableLow}$ according to

$$\mathbf{f}_{TableNoise}(k) = \mathbf{f}_{TableLow}(i(k)) \quad ,0 \le k \le N_Q$$

where $i(k)$ and N_Q are defined by

$$i(k) = \begin{cases} 0 & if\ k = 0 \\ i(k-1) + INT\left(\dfrac{N_{Low} - i(k-1)}{N_Q + 1 - k}\right) & if\ k \neq 0 \end{cases},$$

$$N_Q = \max\left(1, NINT\left(bs\_noise\_bands \cdot \frac{\log\left(\frac{k_2}{k_x}\right)}{\log(2)}\right)\right), 0 \le bs\_noise\_bands \le 3 \cdot$$

4.6.18.3.2.3 Limiter Frequency Band Table

The limiter frequency band table $\mathbf{f}_{TableLim}$, is constructed to have either exactly one limiter band over the entire SBR range, or approximately 1.2, 2 or 3 bands per octave, signaled by *bs_limiter_bands* from the bitstream. The table holds indices of the synthesis filterbank subbands, where the number of elements equals the number of bands plus one. The first element is always k_x, $\mathbf{f}_{TableLim}$ is a subset of the union of $\mathbf{f}_{TableLow}$ and the patch borders defined in subclause 4.6.18.6.

If *bs_limiter_bands* is zero only one limiter band is used and $\mathbf{f}_{TableLim}$ is created as

$$\mathbf{f}_{TableLim} = \left[\mathbf{f}_{TableLow}(0), \mathbf{f}_{TableLow}(N_{Low})\right]$$
$$N_L = 1$$

If *bs_limiter_bands* > *0* the limiter frequency resolution table is created according to the flowchart of Figure 4.40. The variables *numPatches*, **patchBorders** and **patchNumSubbands** are calculated in subclause 4.6.18.6.

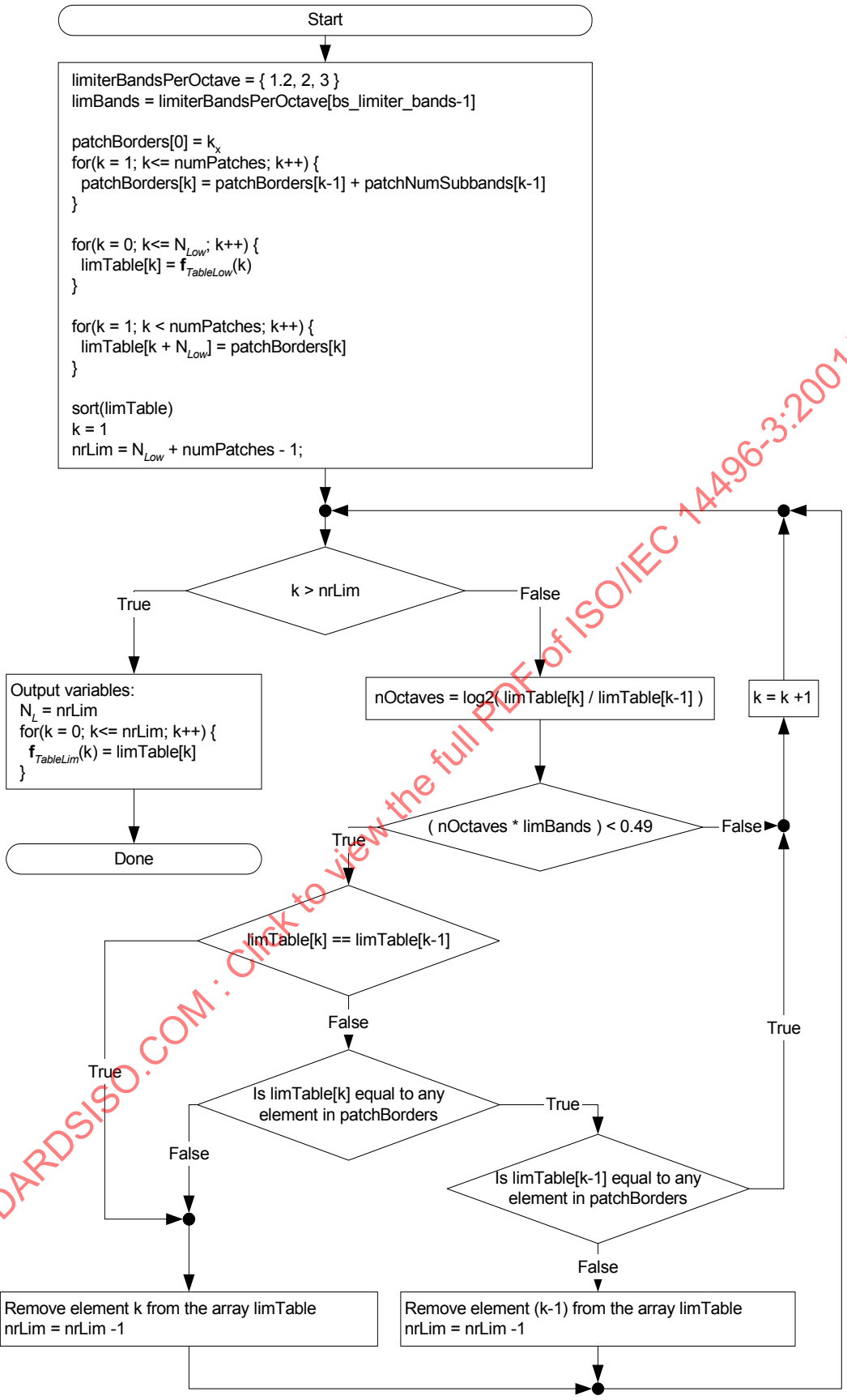


Figure 4.40 – Calculation of $f_{TableLim}$ if *bs_limiter_bands > 0*

4.6.18.3.3 Time / Frequency Grid

The time/frequency grid part of the bitstream describes the number of SBR envelopes and noise floors as well as the time segment associated with each SBR envelope and noise floor. Furthermore, it describes what frequency band table to use for each SBR envelope. Four different SBR frame classes, FIXFIX, FIXVAR, VARFIX and VARVAR, are used, each of which has different capabilities with respect to time/frequency grid selection. The names refer to whether the locations of the leading and trailing SBR frame borders (i.e. the SBR frame boundaries) are variable or not from a syntactical point of view. The SBR envelope and noise floor time segments are described by the vectors $\mathbf{t}_E(l)$ and $\mathbf{t}_Q(l)$ respectively, which contain the borders for each time segment expressed in time slots. Each time segment is defined by a start time border and a stop time border. The time slot indicated by the start time border is included in the time segment, the time slot indicated by the stop time border is excluded from the time segment. For both vectors, the stop time border of time segment l equals the start time border of time segment $l+1$, where l is an arbitrary time segment in the vector. The calculation of $\mathbf{t}_E(l)$ is described below.

First the leading SBR frame border, *absBordLead*, and the trailing SBR frame border, *absBordTrail*, are obtained from the bitstream data according to:

$$absBordLead = \begin{cases} 0 & ,bs\_frame\_class = FIXFIX \text{ or } FIXVAR \\ bs\_var\_bord\_0 & ,bs\_frame\_class = VARVAR \text{ or } VARFIX \end{cases}$$

$$absBordTrail = \begin{cases} numTimeSlots & ,bs\_frame\_class = FIXFIX \text{ or } VARFIX \\ bs\_var\_bord\_1 + numTimeSlots & ,bs\_frame\_class = VARVAR \text{ or } FIXVAR \end{cases}.$$

In order to decode the time borders of all SBR envelopes within the SBR frame, the number of relative borders associated with the leading and trailing time borders respectively are calculated according to:

$$n_{RelLead} = \begin{cases} L_E - 1 & ,bs\_frame\_class = FIXFIX \\ 0 & ,bs\_frame\_class = FIXVAR \\ bs\_num\_rel\_0 & ,bs\_frame\_class = VARVAR \text{ or } VARFIX \end{cases}$$

$$n_{RelTrail} = \begin{cases} 0 & ,bs\_frame\_class = FIXFIX \text{ or } VARFIX \\ bs\_num\_rel\_1 & ,bs\_frame\_class = VARVAR \text{ or } FIXVAR \end{cases}$$

where

$$L_E = bs\_num\_env$$

The SBR envelope time border vector of the current SBR frame, $\mathbf{t}_E(l)$, is calculated according to:

$$\mathbf{t}_E(l) = \begin{cases} absBordLead & if\ l = 0 \\ absBordTrail & if\ l = L_E \\ absBordLead + \sum_{i=0}^{l-1} \mathbf{relBordLead}(i) & if\ 1 \le l \le n_{RelLead} \\ absBordTrail - \sum_{i=0}^{L_E - l - 1} \mathbf{relBordTrail}(i) & if\ n_{RelLead} < l < L_E \end{cases}$$

where $0 \le l \le L_E$ and $\mathbf{relBordLead}(l)$ and $\mathbf{relBordTrail}(l)$ are vectors containing the relative borders associated with the leading and trailing borders respectively. Both vectors are (if applicable) defined below.

$$\mathbf{relBordLead}(l) = \begin{cases} NINT\left(\frac{numTimeSlots}{L_E}\right) & ,bs\_frame\_class = FIXFIX \\ NA & ,bs\_frame\_class = FIXVAR \\ \mathbf{bs\_rel\_bord\_0}(l) & ,bs\_frame\_class = VARVAR\ or\ VARFIX \end{cases}$$

where $0 \le l < n_{RelLead}$

$$\mathbf{relBordTrail}(l) = \begin{cases} NA & ,bs\_frame\_class = FIXFIX\ or\ VARFIX \\ \mathbf{bs\_rel\_bord\_1}(l) & ,bs\_frame\_class = VARVAR\ or\ FIXVAR \end{cases}$$

where $0 \le l < n_{RelTrail}$

Within one SBR frame there can either be one or two noise floors. The noise floor time borders are derived from the SBR envelope time border vector according to:

$$L_Q = bs\_num\_noise$$

$$\mathbf{t}_Q = \begin{cases} \left[\mathbf{t}_E(0), \mathbf{t}_E(1)\right] & ,L_E = 1 \\ \left[\mathbf{t}_E(0), \mathbf{t}_E(middleBorder), \mathbf{t}_E(L_E)\right] & ,L_E > 1 \end{cases}$$

where $middleBorder = func(bs\_frame\_class, bs\_pointer, L_E)$ is calculated according to Table 4.117 below.

Table 4.117 – *middleBorder* function

bs_pointer	*bs_frame_class*		
	FIXFIX	*VARFIX*	*FIXVAR,VARVAR*
= 0	$\frac{L_E}{2}$	1	$L_E - 1$
=1	$\frac{L_E}{2}$	$L_E - 1$	$L_E - 1$
>1	$\frac{L_E}{2}$	$bs\_pointer - 1$	$L_E + 1 - bs\_pointer$

As previously stated, each SBR envelope can be of either high or low frequency resolution. This is described by an SBR envelope frequency resolution vector, $\mathbf{r}(l)$, which is calculated according to:

$$\mathbf{r}(l) = \mathbf{bs\_freq\_res}(l) \quad , 0 \le l < L_E$$

where

$\mathbf{r}(l) = 0$ signals the usage of $\mathbf{f}_{TableLow}$ for SBR envelope l

$\mathbf{r}(l) = 1$ signals the usage of $\mathbf{f}_{TableHigh}$ for SBR envelope l

4.6.18.3.4 SBR Envelope and Noise Floor Decoding

Delta coding of envelope scalefactors and noise floor scalefactors is done in either time or frequency direction for each SBR envelope, and noise floor. When delta coding in the time direction across SBR frame boundaries is applied, the first SBR envelope in the current SBR frame is delta coded with respect to the last SBR envelope of the previous SBR frame. The same is true for the noise floors.

If the SBR Tool is used with a scalable AAC codec, the envelope scalefactors of a stereo enhancement layer can only be decoded if the enhancement layer is available and the variable *enhanceLayDecE* is one. The *enhanceLayDecE* variable is defined as:

$$enhanceLayDecE = \begin{cases} 1 & , if\ enhanceLayDecE' = 1 \quad or \quad bs\_df\_env(0) = 0 \\ 0 & , otherwise \end{cases}$$

where *enhanceLayDecE'* represents the value of the previous SBR frame. For the first SBR frame *enhanceLayDecE'* is set to one. If a scalable system is not used, the variable *enhanceLayDecE* shall be a constant set to one.

The deduction of the envelope scalefactors $\mathbf{E}$ from the delta coded envelope scalefactors $\mathbf{E}_{Delta}$ is defined by:

$$\mathbf{E}(k,l) = \begin{cases} \sum_{i=0}^{k} \delta \cdot \mathbf{E}_{Delta}(i,l) & , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{r}(l)) \\ \mathbf{bs\_df\_env}(l) = 0 \end{cases} \\ g_E(k,l) + \delta \cdot \mathbf{E}_{Delta}(k,l) & , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{r}(l)) \\ \mathbf{bs\_df\_env}(l) = 1 \\ \mathbf{r}(l) = g(l) \end{cases} \\ g_E(i(k),l) + \delta \cdot \mathbf{E}_{Delta}(k,l) & , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{r}(l)) \\ \mathbf{bs\_df\_env}(l) = 1 \\ \mathbf{r}(l) = 0 \\ g(l) = 1 \\ i(k) \text{ is defined by} \\ \mathbf{f}_{TableHigh}(i(k)) = \mathbf{f}_{TableLow}(k) \end{cases} \\ g_E(i(k),l) + \delta \cdot \mathbf{E}_{Delta}(k,l) & , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{r}(l)) \\ \mathbf{bs\_df\_env}(l) = 1 \\ \mathbf{r}(l) = 1 \\ g(l) = 0 \\ i(k) \text{ is defined by} \\ \mathbf{f}_{TableLow}(i(k)) \le \mathbf{f}_{TableHigh}(k) < \mathbf{f}_{TableLow}(i(k)+1) \end{cases} \end{cases}$$

where

$$\delta = \begin{cases} 2 & if\ ch = 1\ AND\ bs\_coupling = 1 \\ 1 & otherwise \end{cases}$$

and where $g_E(k,l)$ and $g(l)$ are defined below and $\mathbf{E}_{Delta}(k,l)$ is read from the bitstream element *bs_data_env* as shown below. The envelope scalefactors from the previous SBR frame, $\mathbf{E}'$, are needed when delta coding in the time direction over SBR frame boundaries. The number of SBR envelopes of the previous SBR frame is denoted L'_E, and is also needed in that case, as well as the frequency resolution vector of the previous SBR frame, denoted $\mathbf{r}'$.

$$g_E(k,l) = \begin{cases} \mathbf{E}(k,l-1) & , \begin{cases} 1 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{r}(l)) \end{cases} \\ \mathbf{E}'(k,L'_E-1) & , \begin{cases} l = 0 \\ 0 \le k < \mathbf{n}(\mathbf{r}(l)) \end{cases} \end{cases} \text{ and } g(l) = \begin{cases} \mathbf{r}(l-1) & , 1 \le l < L_E \\ \mathbf{r}'(L'_E-1) & , l = 0 \end{cases}.$$

$$\mathbf{E}_{Delta}(k,l) = bs\_data\_env[ch][l][k] \quad , \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{r}(l)) \end{cases}$$

If the SBR Tool is used with a scalable AAC codec, the noise floor data of a stereo enhancement layer can only be decoded if the enhancement layer is available and the variable *enhanceLayDecQ* is one. The *enhanceLayDecQ* variable is defined as:

$$enhanceLayDecQ = \begin{cases} 1 & , if\ enhanceLayDecQ' = 1 \quad or \quad bs\_df\_noise(0) = 0 \\ 0 & , otherwise \end{cases}$$

where *enhanceLayDecQ′* represents the value of the previous SBR frame. For the first SBR frame *enhanceLayDecQ′* is set to one. If a scalable system is not used, the variable *enhanceLayDecQ* shall be a constant set to one.

The deduction of the noise floor data $\mathbf{Q}$ from the delta coded noise floor data $\mathbf{Q}_{Delta}$ is defined by.

$$\mathbf{Q}(k,l) = \begin{cases} \sum_{i=0}^{k} \delta \cdot \mathbf{Q}_{Delta}(i,l) & , \begin{cases} 0 \le l < L_Q \\ 0 \le k < N_Q \\ \mathbf{bs\_df\_noise}(l) = 0 \end{cases} \\ \mathbf{Q}(k,l-1) + \delta \cdot \mathbf{Q}_{Delta}(k,l) & , \begin{cases} 1 \le l < L_Q \\ 0 \le k < N_Q \\ \mathbf{bs\_df\_noise}(l) = 1 \end{cases} \\ \mathbf{Q}'(k, L'_Q - 1) + \delta \cdot \mathbf{Q}_{Delta}(k,0) & , \begin{cases} l = 0 \\ 0 \le k < N_Q \\ \mathbf{bs\_df\_noise}(0) = 1 \end{cases} \end{cases}$$

where

$$\delta = \begin{cases} 2 & if\ ch = 1\ AND\ bs\_coupling = 1 \\ 1 & otherwise \end{cases}$$

and where $\mathbf{Q}'$ is the noise floor scalefactors from the previous SBR frame and L'_Q is the number of noise floors from the previous SBR frame. $\mathbf{Q}_{Delta}(k,l)$ is read from the bitstream element *bs_data_noise* as shown below.

$$\mathbf{Q}_{Delta}(k,l) = bs\_data\_noise[ch][l][k] \quad , \begin{cases} 0 \le l < L_Q \\ 0 \le k < N_Q \end{cases}$$

4.6.18.3.5 Dequantization and Stereo Decoding

For the quantization of the envelope scalefactors, there are two quantization steps available. *bs_amp_res* = 0 corresponds to a quantization step of 1.5 dB and *bs_amp_res* = 1 corresponds to a quantization step of 3.0 dB.

For a single channel element, the envelope scalefactors are dequantized according to:

$$\mathbf{E}_{Orig}(k,l) = 64 \cdot 2^{\frac{\mathbf{E}(k,l)}{a}}, \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{r}(l)) \end{cases}$$

where $a = \begin{cases} 2, & bs\_amp\_res = 0 \\ 1, & bs\_amp\_res = 1 \end{cases}$.

The noise floor scalefactors are dequantized according to:

$$\mathbf{Q}_{Orig}(k,l) = 2^{NOISE\_FLOOR\_OFFSET - \mathbf{Q}(k,l)}, \begin{cases} 0 \le l < L_Q \\ 0 \le k < N_Q \end{cases}$$

For a channel pair element where coupling mode is not used, i.e. *bs_coupling* = 0, the individual channels are treated as the single channel element case above.

If coupling mode is used, i.e. *bs_coupling* = 1, the time-grids $\mathbf{t}_E$ and $\mathbf{t}_Q$ are the same for both channels. Let $\mathbf{E}_0$, $\mathbf{E}_1$, $\mathbf{Q}_0$, and $\mathbf{Q}_1$ represent the decoded envelope scalefactors and noise floor scalefactors, in accordance with the decoding process outlined above. Subscript zero represents the firstly decoded channel (the energy average and the noise-floor average of the original left and right channel) and subscript one represents the secondly decoded channel (the energy ratio and the noise-floor ratio of the original left and right channel).

Below it is shown how to dequantize the envelope scalefactors and noise floor scalefactors in coupling mode (*bs_coupling* = 1).

$$\mathbf{E}_{LeftOrig}(k,l) = 64 \cdot \frac{2^{\frac{\mathbf{E}_0(k,l)}{a}+1}}{1+2^{\frac{\mathbf{panOffset}(bs\_amp\_res)-\mathbf{E}_1(k,l)}{a}}}, \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{r}(l)) \end{cases}$$

$$\mathbf{E}_{RightOrig}(k,l) = 64 \cdot \frac{2^{\frac{\mathbf{E}_0(k,l)}{a}+1}}{1+2^{\frac{\mathbf{E}_1(k,l)-\mathbf{panOffset}(bs\_amp\_res)}{a}}}, \begin{cases} 0 \le l < L_E \\ 0 \le k < \mathbf{n}(\mathbf{r}(l)) \end{cases}$$

$$\mathbf{Q}_{OrigLeft}(k,l) = \frac{2^{NOISE\_FLOOR\_OFFSET - \mathbf{Q}_0(k,l)+1}}{1+2^{\mathbf{panOffset}(1)-\mathbf{Q}_1(k,l)}}, \begin{cases} 0 \le l < L_Q \\ 0 \le k < N_Q \end{cases}$$

$$\mathbf{Q}_{OrigRight}(k,l) = \frac{2^{NOISE\_FLOOR\_OFFSET - \mathbf{Q}_0(k,l)+1}}{1+2^{\mathbf{Q}_1(k,l)-\mathbf{panOffset}(1)}}, \begin{cases} 0 \le l < L_Q \\ 0 \le k < N_Q \end{cases}.$$

Above, the SBR envelopes and noise floors are denoted $\mathbf{E}_{OrigLeft}$, $\mathbf{E}_{OrigRight}$, $\mathbf{Q}_{OrigLeft}$ and $\mathbf{Q}_{OrigRight}$, in order to differentiate between the two channels in the channel pair element. Since no channel dependency exists after the above decoding and dequantization, the envelopes and noise floors will from this point on be referred to as $\mathbf{E}_{Orig}$ and $\mathbf{Q}_{Orig}$.

If the SBR Tool is used with a scalable AAC codec and a stereo enhancement layer is not present, the data in the channel pair element available in the mono base layer shall be dequantized as a single channel element, i.e. $\mathbf{E}_{LeftOrig}(k,l)$ is calculated as $\mathbf{E}_{Orig}(k,l)$.

If the stereo enhancement layer is available, the envelope scalefactors in the channel pair element shall be dequantized and stereo decoded as a channel pair element, provided that the variable *enhanceLayDecE* is one. If the variable *enhanceLayDecE* is zero, the envelope scalefactors of the first channel are dequantized as a single channel element, i.e. $\mathbf{E}_{LeftOrig}(k,l)$ is calculated as $\mathbf{E}_{Orig}(k,l)$, and the envelope scalefactors of the second channel, shall be set to the same values as the envelope scalefactors of the first channel, i.e. $\mathbf{E}_{RightOrig}(k,l) = \mathbf{E}_{LeftOrig}(k,l)$.

The same applies for the noise floor scalefactors, i.e. if the a stereo enhancement layer is not present, the data in the channel pair element available in the mono base layer shall be dequantized as a single channel element, i.e. $\mathbf{Q}_{LeftOrig}(k,l)$ is calculated as $\mathbf{Q}_{Orig}(k,l)$.

If the stereo enhancement layer is available, the noise floor scalefactors in the channel pair element shall be dequantized and stereo decoded as a channel pair element, provided that the variable *enhanceLayDecQ* is one. If the variable *enhanceLayDecQ* is zero, the noise floor scalefactors of the first channel are dequantized as a single channel element, i.e. $\mathbf{Q}_{LeftOrig}(k,l)$ is calculated as $\mathbf{Q}_{Orig}(k,l)$, and the noise floor scalefactors of the second channel, shall be set to the same values as the noise floor scalefactors of the first channel, i.e. $\mathbf{Q}_{RightOrig}(k,l) = \mathbf{Q}_{LeftOrig}(k,l)$.

4.6.18.3.6 Requirements

The following requirements apply to the SBR data.

- The number of QMF subbands covered by SBR, i.e. k_2-k_0, shall satisfy:

$$k_2 - k_0 \le \begin{cases} 48 & , Fs_{SBR} \le 32kHz \\ 35 & , Fs_{SBR} = 44.1kHz \\ 32 & , Fs_{SBR} \ge 48kHz \end{cases}.$$

- The stop frequency border of the SBR range shall be within $\frac{Fs_{SBR}}{2}$, i.e. $k_x + M \le 64$.

- The start frequency border of the SBR range shall be within $\frac{Fs_{SBR}}{4}$, i.e. $k_x \le 32$.

- The number of SBR envelopes in an SBR frame, L_E, shall satisfy:

$$L_E \le \begin{cases} 4 & , bs\_frame\_class = FIXFIX \\ 5 & , bs\_frame\_class = VARVAR \end{cases}$$

- The number of noise floor scale factors, N_Q, shall satisfy $N_Q \le 5$.

- The number of patches *numPatches*, shall satisfy $numPatches \le 5$.

- For single channel elements and for channel pair elements where coupling is not used (i.e. *bs_coupling* = 0), the quantized noise floor scalefactors $\mathbf{Q}$ shall satisfy: $\mathbf{Q} \in [0,30]$.

- For channel pair elements where coupling is used (i.e. *bs_coupling* = 1), the quantized noise floor scalefactors shall satisfy:

- $\mathbf{Q}_0 \in [0,30]$
- $\mathbf{Q}_1 \in [0,2 \cdot \mathbf{panOffset}(1)]$

- For channel pair elements where coupling is used (i.e. *bs_coupling* = 1), the quantized noise floor scalefactors for the second decoded channel (i.e. $\mathbf{Q}_1$), and the quantized envelope scalefactors for the second decoded channel (i.e. $\mathbf{E}_1$) shall be an even integer.

- In the flowchart in Figure 4.39, when the flowchart has been executed, the following relations shall be satisfied:
  - $numBands0 > 0$
  - $vDk0(i) > 0 \; \forall \; i$
  - $vDk1(i) > 0 \; \forall \; i$

- In the flowchart in Figure 4.38, when the flowchart has been executed, the variable *numBands* shall satisfy: $numBands > 0$.

- The following relation shall be satisfied: $bs\_xover\_band < N_{Master}$.

- Delta coded envelope scalefactors and noise floor scalefactors, shall be within the range of the Huffman tables in 4.A.6.1

- If the SBR tool is used in a system that operates with mono/stereo scalability, the *bs_coupling* bit shall be set to one.

- If the SBR tool is used in a system that operates with bandwidth or mono/stereo scalability, all SBR bitstream data, except the stereo enhancement part of the SBR data, shall be placed in the lowest layer of the data stream. The stereo enhancement part of the SBR data that shall be placed in the lowest layer of the data stream carrying stereo data. All SBR data shall cover the largest SBR range that can occur for the different layers in the stream.

4.6.18.4 SBR Filterbanks

4.6.18.4.1 Analysis Filterbank

A QMF bank is used to split the time domain signal output from the core decoder into 32 subband signals. The output from the filterbank, i.e. the subband samples, are complex-valued and thus oversampled by a factor two compared to a regular QMF bank. The flowchart of this operation is given in Figure 4.41. The filtering involves the following steps, where an array $\mathbf{x}$ consisting of 320 time domain input samples is assumed. A higher index into the array corresponds to older samples.

- Shift the samples in the array $\mathbf{x}$ by 32 positions. The oldest 32 samples are discarded and 32 new samples are stored in positions 0 to 31.
- Multiply the samples of array $\mathbf{x}$ by every other coefficient of window $\mathbf{c}$. The window coefficients can be found in Table 4.A.87.
- Sum the samples according to the formula in the flowchart to create the 64-element array $\mathbf{u}$.
- Calculate 32 new subband samples by the matrix operation $\mathbf{Mu}$, where $\mathbf{M}(k,n) = 2 \cdot \exp\left(\frac{i \cdot \pi \cdot (k+0.5) \cdot (2 \cdot n - 0.5)}{64}\right)$, $\begin{cases} 0 \le k < 32 \\ 0 \le n < 64 \end{cases}$
  In the equation, $\exp()$ denotes the complex exponential function and i is the imaginary unit.

Every loop in the flowchart produces 32 complex-valued subband samples, each representing the output from one filterbank subband. For every SBR frame the filterbank will produce $numTimeSlots \cdot RATE$ subband samples for every subband, corresponding to a time domain signal of length $numTimeSlots \cdot RATE \cdot 32$ samples. In the flowchart $\mathbf{X}_{Low}$[k][l] corresponds to subband sample l in QMF subband k.

4.6.18.4.2 Synthesis Filterbank

Synthesis filtering of the SBR-processed subband signals is achieved using a 64-subband QMF bank. The output from the filterbank is real-valued time domain samples. The process is given by the flowchart in Figure 4.42. The synthesis filtering comprises the following steps, where an array $\mathbf{v}$ consisting of 1280 samples is assumed:

- Shift the samples in the array $\mathbf{v}$ by 128 positions. The oldest 128 samples are discarded.
- The 64 new complex-valued subband samples are multiplied by the matrix $\mathbf{N}$, where $\mathbf{N}(k,n) = \frac{1}{64} \cdot \exp\left(\frac{i \cdot \pi \cdot (k+0.5) \cdot (2 \cdot n - 255)}{128}\right)$, $\begin{cases} 0 \le k < 64 \\ 0 \le n < 128 \end{cases}$
  In the equation, $\exp()$ denotes the complex exponential function and i is the imaginary unit. The real part of the output from this operation is stored in the positions 0 to 127 of array $\mathbf{v}$.
- Extract samples from $\mathbf{v}$ according to the flowchart in Figure 4.42 to create the 640-element array $\mathbf{g}$.
- Multiply the samples of array $\mathbf{g}$ by window $\mathbf{c}$ to produce array $\mathbf{w}$. The window coefficients of $\mathbf{c}$ can be found in Table 4.A.87, and are the same as for the analysis filterbank.
- Calculate 64 new output samples by summation of samples from array $\mathbf{w}$ according to the last step in the flowchart of Figure 4.42.

Every SBR frame produces an output of $numTimeSlots \cdot RATE \cdot 64$ time domain samples. In the flowchart of Figure 4.42 $\mathbf{X}$[k][l] corresponds to subband sample l in the QMF subband k, and every new loop produces 64 time domain samples as output.

4.6.18.4.3 Downsampled Synthesis Filterbank

The downsampled synthesis filtering of the SBR-processed subband signals is achieved using a 32-channel QMF bank. The output from the filterbank is real-valued time domain samples. The process is given of the flowchart in Figure 4.43. The synthesis filtering comprises the following steps, where an array $\mathbf{v}$ consisting of 640 samples is assumed:

- Shift the samples in the array $\mathbf{v}$ by 64 positions. The oldest 64 samples are discarded.
- The 32 new complex-valued subband samples are multiplied by the matrix $\mathbf{N}$, where
  $$\mathbf{N}(k,n) = \frac{1}{64} \cdot \exp\left(\frac{i \cdot \pi \cdot (k+0.5) \cdot (2 \cdot n - 127)}{64} \right) \quad , \begin{cases} 0 \le k < 32 \\ 0 \le n < 64 \end{cases}$$
  In the equation, exp() denotes the complex exponential function and i is the imaginary unit. The real part of the output from this operation is stored in the positions 0 to 63 of array $\mathbf{v}$.
- Extract samples from $\mathbf{v}$ according to the flowchart in Figure 4.43 to create the 320-element array $\mathbf{g}$.
- Multiply the samples of array $\mathbf{g}$ by every other coefficient of window $\mathbf{w}$. The window coefficients of $\mathbf{c}$ can be found in Table 4.A.87, and are the same as for the analysis filterbank.
- Calculate 32 new output samples by summation of samples from array $\mathbf{w}$ according to the last step in the flowchart of Figure 4.43

Every SBR frame produces an output of $numTimeSlots \cdot RATE \cdot 32$ time domain samples. In the flowchart of Figure 4.43 $\mathbf{X}$[k][l] corresponds to subband sample l in the QMF subband k, and every new loop produces 32 time domain samples as output.

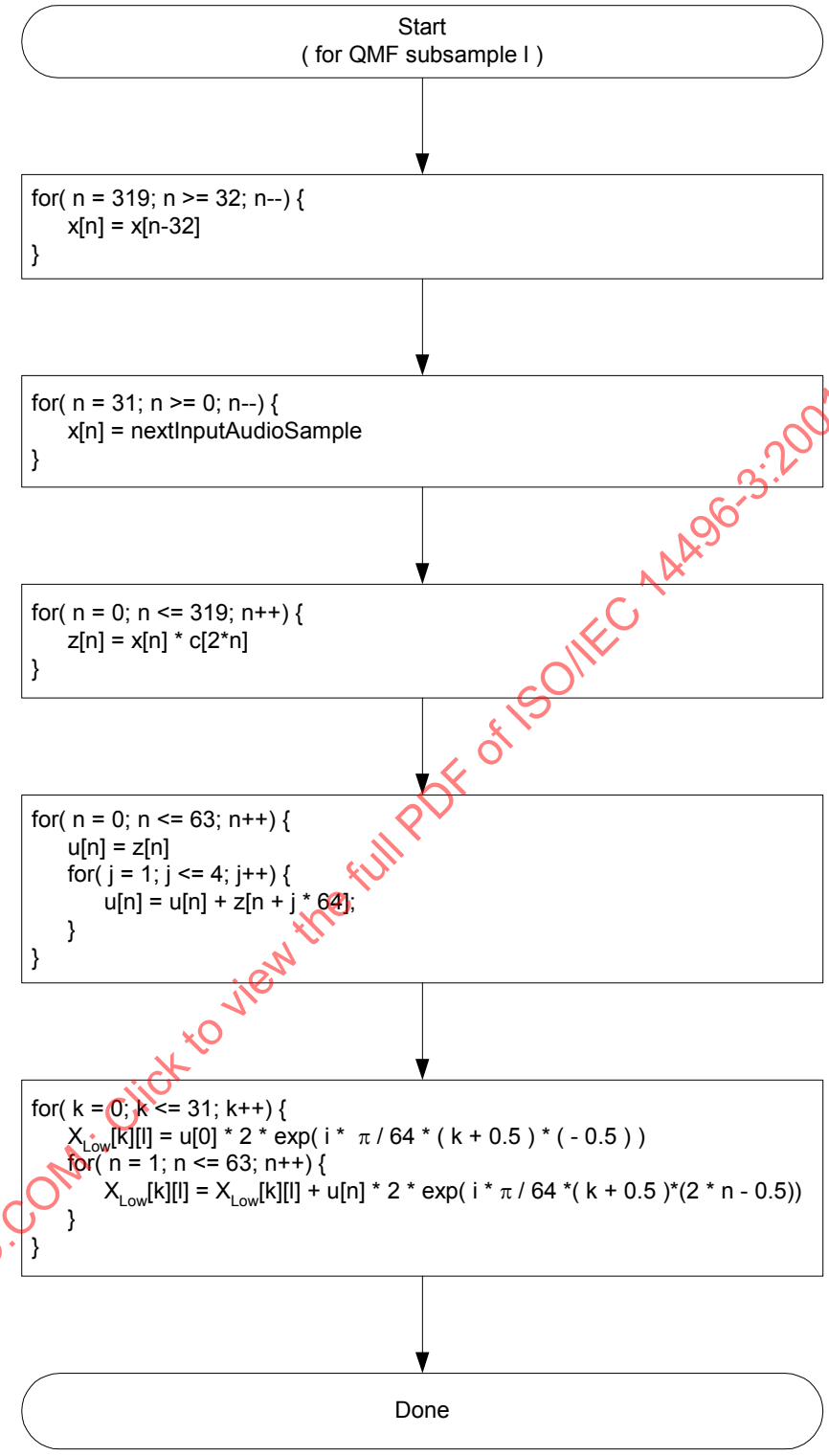


Figure 4.41 – Flowchart of decoder analysis QMF bank

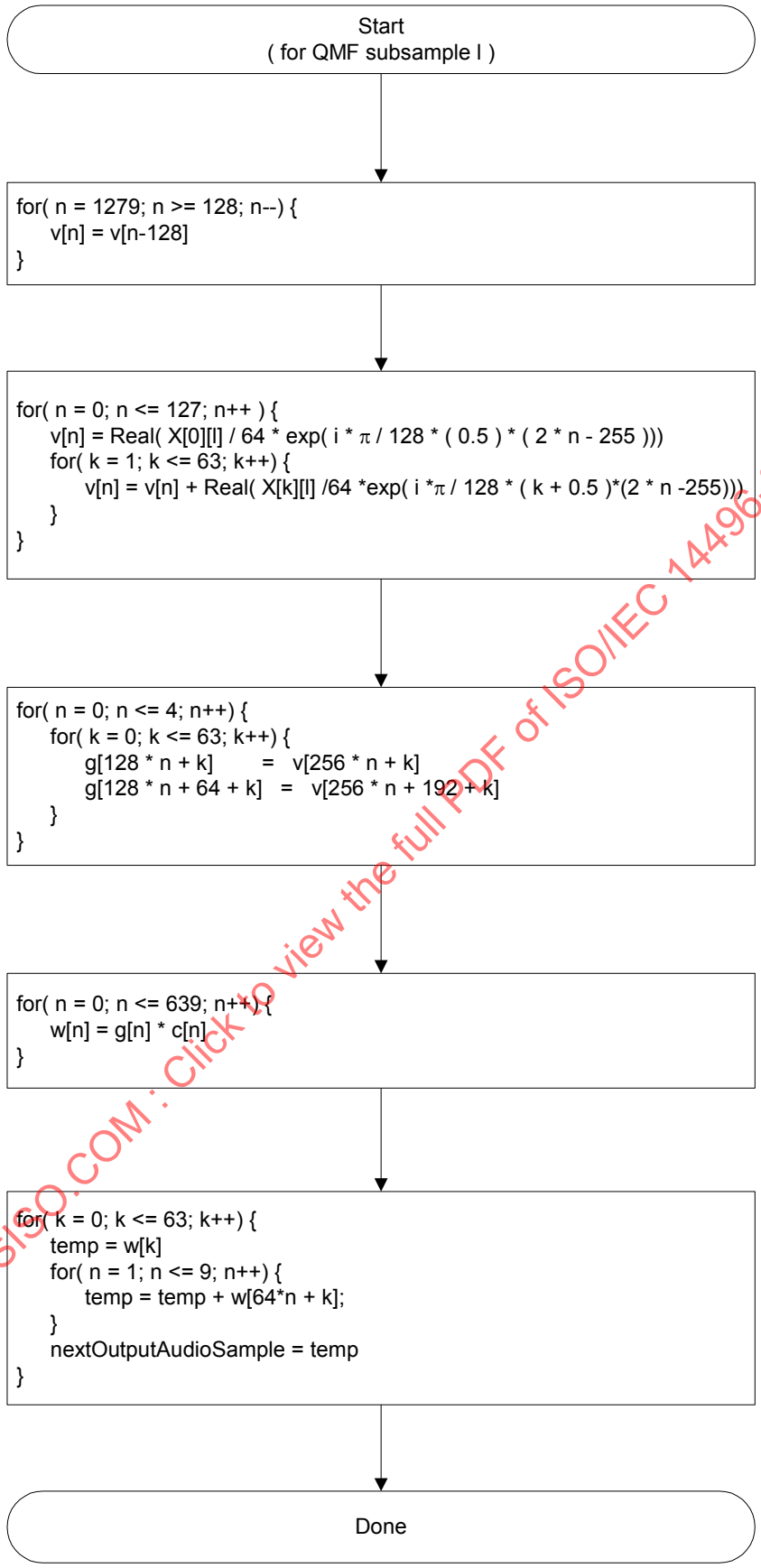


Figure 4.42 – Flowchart of decoder synthesis QMF bank

Figure 4.43 – Flowchart of decoder downsampled synthesis QMF bank

4.6.18.5 SBR Tool Overview

The decoder block diagram of Figure 4.44 shows how the SBR parts and the AAC core decoder are interconnected. In order to synchronize the SBR envelope data and the AAC core decoder output, the SBR bitstream data has to be time delayed with respect to the AAC core bitstream data, i.e. the SBR parts in the encoder are operating on time delayed audio samples with respect to the AAC core encoder. To achieve a synchronized output signal, the following steps have to be acknowledged in the decoder:

- The bitstream payload deformatter divides the bitstream payload into two parts; the AAC core coder part and the SBR part.
- The SBR bitstream part is fed to the bitstream parser followed by dequantization The raw data is Huffman decoded.
- The AAC bitstream part is fed to the AAC core decoder, where the bitstream data of the current SBR frame is decoded, yielding a time domain audio signal block of 1024 samples or 960 samples depending on frame size.
- The core coder audio block is fed to the analysis QMF bank. This is illustrated in Figure 4.45 (a) by the dashed block. If a scalable core coder is used the audio block representing the highest available layer shall be used.
- The analysis QMF bank performs the filtering of the core coder audio signal. Section 4.6.18.4.1 describes the analysis filter bank and Figure 4.45 (a) shows the timing of the analysis windowing. The output from the filtering is stored in the matrix

  $$\mathbf{X}_{Low}(k,\ l\ +\ t_{HFGen}),\ 0\ \le\ k\ <\ 32,\ 0\ \le\ l\ <\ numTimeSlots\cdot RATE\ .$$

  The output from the analysis QMF bank is hence delayed t_{HFGen} subband samples, before being fed to the synthesis QMF bank. To achieve synchronization $t_{HFGen} = 8$. The resulting subband samples are shown in Figure 4.45 (b) as the upper dashed block.
- The HF generator calculates $\mathbf{X}_{High}$ given the matrix $\mathbf{X}_{Low}$ according to the scheme outlined in section 4.6.18.6.1. The process is guided by the SBR data contained in the current SBR frame. The result is illustrated by the dashed block in Figure 4.45 (b).
- The envelope adjuster outlined in subclause 4.6.18.7 calculates the matrix $\mathbf{Y}$ given the matrix $\mathbf{X}_{High}$ and the SBR envelope data, extracted from the SBR bitstream. To achieve synchronization, t_{HFAdj} has to be set to $t_{HFAdj} = 2$, i.e. the envelope adjuster operates on data delayed t_{HFGen} - t_{HFAdj} subband samples.
- The synthesis QMF bank operates on the output from the analysis QMF bank and the output from the envelope adjuster. It first creates the matrix $\mathbf{X}$ from these outputs according to:

$$\mathbf{X}(k,l)=\begin{cases}\mathbf{X}_{Low}(k,l+t_{HFAdj}) & ,\ 0\ \le\ k\ <\ k_x'+bsco',\ 0\ \le\ l\ <\ l_{Temp}\\ \mathbf{X}_{Low}(k,l+t_{HFAdj}) & ,\ 0\ \le\ k\ <\ k_x+bsco,\ l_{Temp}\ \le\ l\ <\ numTimeSlots\cdot RATE\\ \mathbf{Y}(k,l+t_{HFAdj}) & ,\ k_x'+bsco'\ \le\ k\ <\ k_x\ +\ M,\ 0\ \le\ l\ <\ l_{Temp}\\ \mathbf{Y}(k,l+t_{HFAdj}) & ,\ k_x+bsco\le\ k\ <\ k_x\ +\ M,\ l_{Temp}\ \le\ l\ <\ numTimeSlots\cdot RATE\\ 0 & ,\ \max(k_x+bsco,k_x\ +\ M)+\ M\ \le\ k\ <\ 64,\ 0\ \le\ l\ <\ numTimeSlots\cdot RATE\end{cases},$$

where $l_{Temp} = RATE \cdot \mathbf{t}_E'(L_E') - numTimeSlots \cdot RATE$, and ' indicates the value of the previous SBR frame. At start-up l_{Temp}, lsb' and $bsco'$ are set to zero. And where

$bsco = \max\left(INT\left(maxAACLine \cdot 32 / frameLength\right) - lsb, 0\right)$, and where

$$maxAACLine = \begin{cases} swb\_offset\_long\_window\left[\max\left(max\_sfb - 1, 0\right)\right], for\ long\ blocks \\ 8 \cdot swb\_offset\_short\_window\left[\max\left(max\_sfb - 1, 0\right)\right], for\ short\ blocks \end{cases}$$

If the SBR tool is used for pure upsampling without SBR processing, the matrix $\mathbf{X}$ is created according to :

$$\mathbf{X}(k,l) = \begin{cases} \mathbf{X}_{Low}\left(k, l + t_{HFAdj}\right) & , 0 \le k < 32,\ 0 \le l < numTimeSlots \cdot RATE \\ 0 & , 32 \le k < 64,\ 0 \le l < numTimeSlots \cdot RATE \end{cases}$$

Subsequently, the subband sample matrix

$$\mathbf{X}(k, l),\ 0 \le k < 64,\ 0 \le l < numTimeSlots \cdot RATE,$$

is synthesized in the synthesis QMF bank in accordance to section 4.6.18.4.2. The resulting output samples are shown as the dashed block of Figure 4.45 (c) (Figure 4.45 (d) if downsampled SBR is used), where also the timing of the synthesis windows is shown.

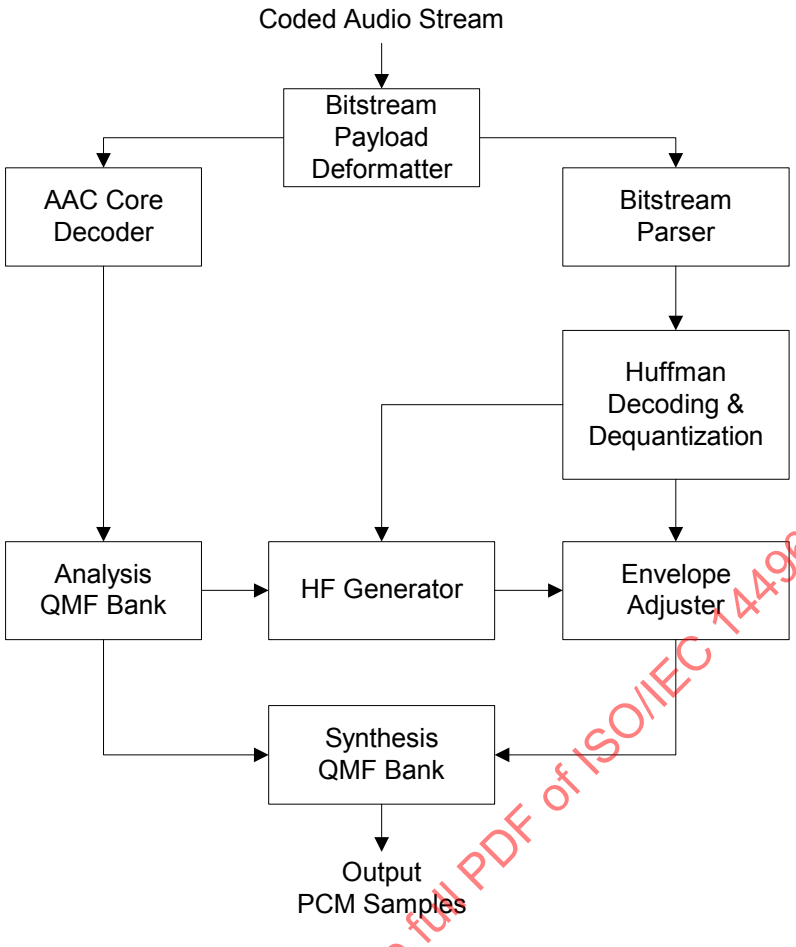
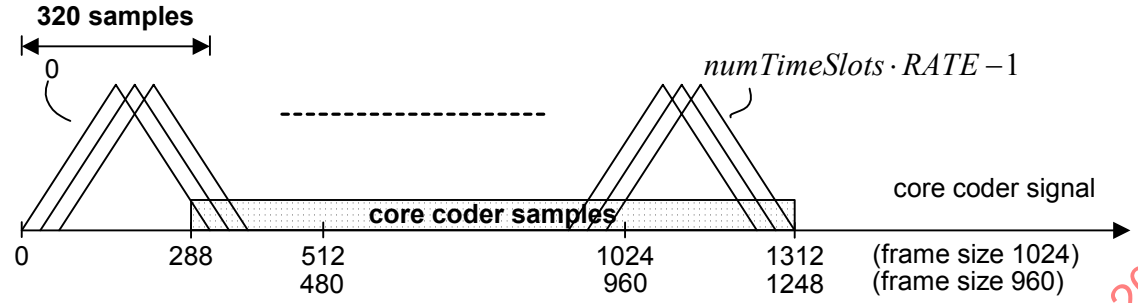
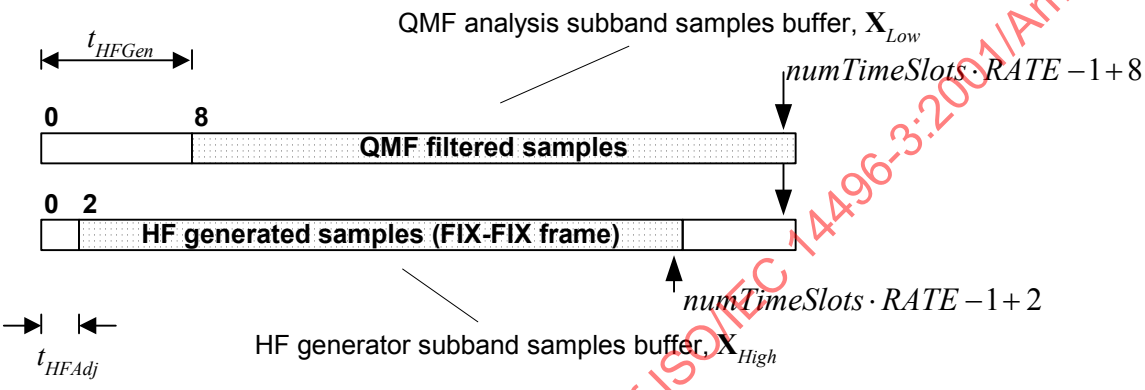


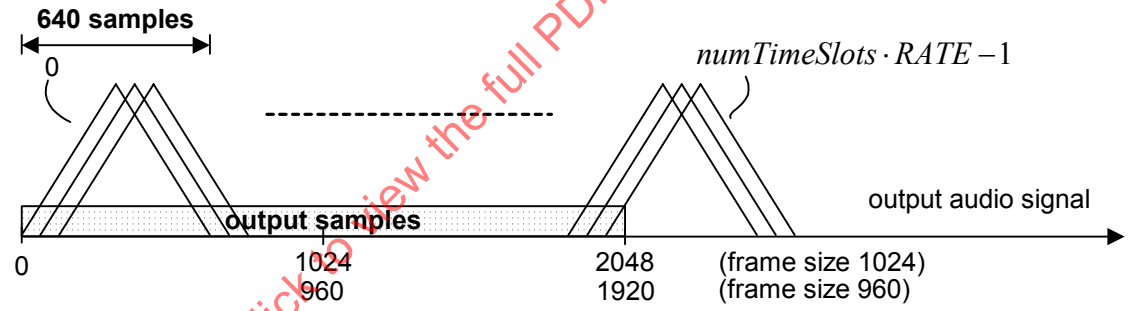
Figure 4.44 – Decoder block diagram



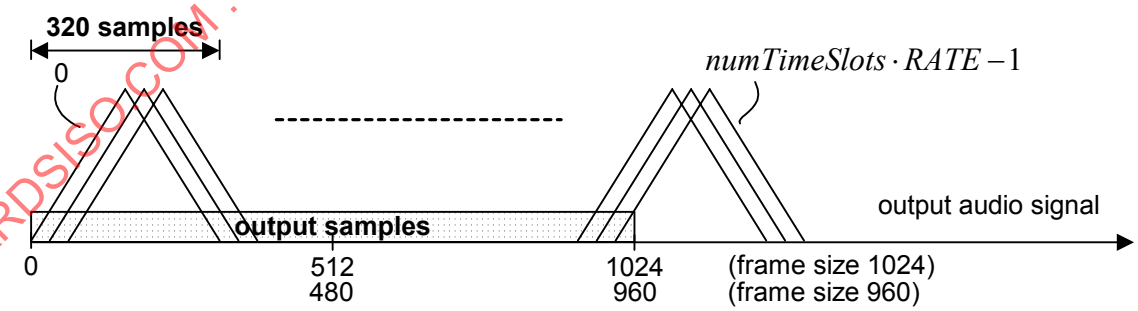
(a) core coder signal QMF analysis windowing



(b) subband samples buffers, $\mathbf{X}_{Low}$ and $\mathbf{X}_{High}$



(c) output signal QMF synthesis windowing



(d) output signal downsampled QMF synthesis windowing

Figure 4.45 – Synchronization and timing

4.6.18.6 HF Generation

4.6.18.6.1 Introduction

The objective of the HF generator is to patch, or copy, a number of subband signals obtained from the analysis filterbank from consecutive subbands of matrix $\mathbf{X}_{Low}$ to consecutive subbands of matrix $\mathbf{X}_{High}$. The definition of the patching, i.e. the number of patches and the source ranges for the individual patches, is described by the vectors **patchNumSubbands** and **patchStartSubband**, and the variable *numPatches*. The subband signals $\mathbf{X}_{High}$ are inverse filtered according to the inverse filtering levels signaled from the encoder.

4.6.18.6.2 Inverse Filtering

The inverse filtering is done in two steps. Linear prediction is first performed on the subband signals of $\mathbf{X}_{Low}$ Then the actual inverse filtering is done independently for each of the subband signals patched to $\mathbf{X}_{High}$ by the HF generator. The subband signals are complex valued, which results in complex filter coefficients for the linear prediction as well as for the inverse filtering. The prediction filter coefficients are obtained from the covariance method. The covariance matrix elements calculated are:

$$\phi_k(i,j) = \sum_{n=0}^{numTimeSlots \cdot RATE + 6 - 1} \mathbf{X}_{Low}\left(k, n - i + t_{HFAdj}\right) \cdot \mathbf{X}_{Low}^{*}\left(k, n - j + t_{HFAdj}\right), \begin{cases} 0 \le i < 3 \\ 1 \le j < 3 \\ 0 \le k < k_0 \end{cases}$$

The coefficients $\alpha_0(k)$ and $\alpha_1(k)$ used to filter the subband signal are calculated as:

$$\mathrm{d}(k) = \phi_k(2,2) \cdot \phi_k(1,1) - \frac{1}{1+\varepsilon_{Inv}} \left|\phi_k(1,2)\right|^2,$$

$$\alpha_1(k) = \begin{cases} \dfrac{\phi_k(0,1) \cdot \phi_k(1,2) - \phi_k(0,2) \cdot \phi_k(1,1)}{d(k)} & , d(k) \neq 0 \\ 0 & , d(k) = 0 \end{cases},$$

$$\alpha_0(k) = \begin{cases} -\dfrac{\phi_k(0,1) + \alpha_1(k) \cdot \phi_k^{*}(1,2)}{\phi_k(1,1)} & , \phi_k(1,1) \neq 0 \\ 0 & , \phi_k(1,1) = 0 \end{cases}.$$

In the first formula above ε_{Inv} is the relaxation parameter (ε_{Inv} = 1E-6). Moreover, if either of the magnitudes of $\alpha_0(k)$ and $\alpha_1(k)$ is greater than or equal to 4, both coefficients are set to zero.

The calculation of the chirp factors, **bwArray,** is shown below. Each chirp factor is used within a specific frequency range defined by the noise floor frequency band table, $\mathbf{f}_{TableNoise}$.

$$\mathbf{bwArray}(i) = \begin{cases} 0 & if\ \mathbf{tempBw}(i) < 0.015625 \\ \mathbf{tempBw}(i) & if\ \mathbf{tempBw}(i) \ge 0.015625 \end{cases}, \quad 0 \le i < N_Q$$

where $\mathbf{tempBw}(i)$ is calculated as

$$\mathbf{tempBw}(i) = \begin{cases} 0.75000 \cdot newBw + 0.25000 \cdot \mathbf{bwArray}'(i) & if\ newBw < \mathbf{bwArray}'(i) \\ 0.90625 \cdot newBw + 0.09375 \cdot \mathbf{bwArray}'(i) & if\ newBw \geq \mathbf{bwArray}'(i) \end{cases}, 0 \leq i < N_Q$$

bwArray´ are the **bwArray** values calculated in the previous SBR frame, and are assumed to be zero for the first SBR frame. *newBw* is a function of **bs_invf_mode**(*i*) and **bs_invf_mode´**(*i*), given by Table 4.118, where **bs_invf_mode´** are the **bs_invf_mode** values from the previous SBR frame.

Table 4.118 – *newBw* function

bs_invf_mode(*i*)´	**bs_invf_mode(*i*)**			
	Off	Low	Intermediate	Strong
Off	0.0	0.6	0.9	0.98
Low	0.6	0.75	0.9	0.98
Intermediate	0.0	0.75	0.9	0.98
Strong	0.0	0.75	0.9	0.98

4.6.18.6.3 HF Generator

The patch is built in accordance to the flowchart of Figure 4.46, where the output variable *numPatches* is an integer value specifying the number of patches. **patchStartSubband** and **patchNumSubbands** are vectors holding the data output from the patch decision algorithm.

The HF generation is obtained according to:

$$\mathbf{X}_{High}(k, l + t_{HFAdj}) = \mathbf{X}_{Low}(p, l + t_{HFAdj}) + \mathbf{bwArray}(g(k)) \cdot \alpha_0(p) \cdot \mathbf{X}_{Low}(p, l - 1 + t_{HFAdj}) + \left[\mathbf{bwArray}(g(k))\right]^2 \cdot \alpha_1(p) \cdot \mathbf{X}_{Low}(p, l - 2 + t_{HFAdj}),$$

where $g(k)$ is defined by $\mathbf{f}_{TableNoise}(g(k)) \leq k < \mathbf{f}_{TableNoise}(g(k)+1)$ and

$$\begin{cases} k = k_x + x + \sum_{q=0}^{i-1} \mathbf{patchNumSubbands}(q) \\ p = \mathbf{patchStartSubband}(i) + x \end{cases}$$

for $0 \leq x < \mathbf{patchNumSubbands}(i)$, $0 \leq i < numPatches$, $RATE \cdot \mathbf{t}_E(0) \leq l < RATE \cdot \mathbf{t}_E(L_E)$.

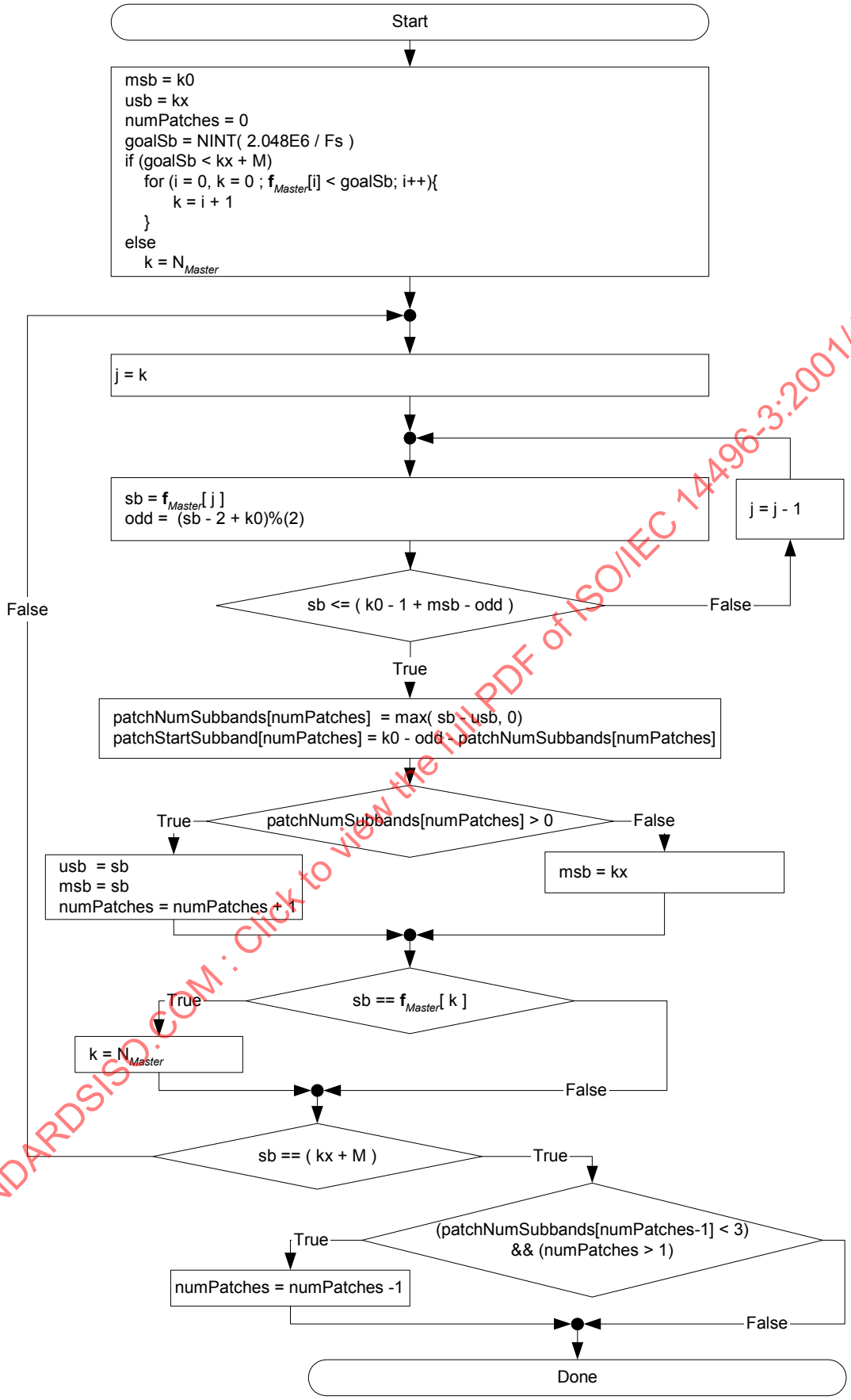


Figure 4.46 – Flowchart of patch construction

4.6.18.7 HF Adjustment

4.6.18.7.1 Introduction

The envelope adjuster takes the input QMF-matrix $\mathbf{X}_{High}$ and produces the output QMF matrix $\mathbf{Y}$. The envelope adjustment is done upon the entire SBR range covering M QMF subbands, starting on subband k_x, for the time-frame spanned by the current SBR frame (indicated by the vector $\mathbf{t}_E$). Throughout the description below several temporary vectors and matrices are introduced in order to make the explanation stringent. All temporary matrices and vectors are indexed from zero, removing the k_x offset. The below description of the envelope adjustment is channel independent, and outlined for one channel only, and for one SBR frame only. Variables used below that originate from the processing of the previous SBR frame, are assumed to be zero for the first SBR frame.

4.6.18.7.2 Mapping

Some of the data extracted from the bitstream are vectors (or matrices) containing data elements representing a frequency range of several QMF subbands. In order to simplify the explanation below, and sometimes out of necessity, this grouped data is mapped to the highest available frequency resolution for the envelope adjustment, i.e. to the individual QMF subbands within the SBR range. This means that several adjacent subbands in the mapped vectors (or matrices) will have the same value.

The mapping of the envelope scalefactors and the noise floor scalefactors is outlined below. The SBR envelope is mapped to the resolution of the QMF bank, albeit with preserved time resolution. The noise floor scalefactors are also mapped to the frequency resolution of the filterbank, but with the time resolution of the envelope scalefactors.

$$\mathbf{E}_{OrigMapped}(m-k_x,l)=\mathbf{E}_{Orig}(i,l) \quad ,\mathbf{F}(i,\mathbf{r}(l))\le m<\mathbf{F}(i+1,\mathbf{r}(l)),0\le i<\mathbf{n}(\mathbf{r}(l)),0\le l<L_E$$

$$\mathbf{Q}_{Mapped}(m-k_x,l)=\mathbf{Q}_{Orig}(i,k(l)) \quad ,\mathbf{f}_{TableNoise}(i)\le m<\mathbf{f}_{TableNoise}(i+1),0\le i<N_Q,0\le l<L_E$$

where $k(l)$ is defined by $RATE\cdot\mathbf{t}_E(l)\ge RATE\cdot\mathbf{t}_Q(k(l)),RATE\cdot\mathbf{t}_E(l+1)\le RATE\cdot\mathbf{t}_Q(k(l)+1)$, and $\mathbf{F}(i,\mathbf{r}(l))$ is indexed as row, column i.e. $\mathbf{F}(i,\mathbf{r}(l))$ gives $\mathbf{f}_{TableLow}(i)$ for $\mathbf{r}(l)=LO$ and $\mathbf{f}_{TableHigh}(i)$ for $\mathbf{r}(l)=HI$.

The mapping of the additional sinusoids is done below. In order to simplify two matrices are introduced, $\mathbf{S}_{IndexMapped}$ and $\mathbf{S}_{Mapped}$. The former is a binary matrix indicating in which QMF subbands sinusoids should be added, the latter is a matrix used to compensate the energy-values for the frequency bands where a sinusoid is added. If the bitstream indicates a sinusoid in a QMF subband where there was none present in the previous SBR frame, the generated sine should start at the position indicated by l_A (Table 4.119) in the present SBR frame. The generated sinusoid is placed in the middle of the high frequency resolution band, according to the below:

Let,

$$\mathbf{S}_{Index}(i)=\begin{cases}\mathbf{bs\_add\_harmonic}(i) & ,bs\_add\_harmonic\_flag=1\\ 0 & ,bs\_add\_harmonic\_flag=0\end{cases},0\le i<N_{High}$$

$$\mathbf{S}_{IndexMapped}(m-k_x,l)=\begin{cases}0 & if\ m\neq INT\left(\dfrac{\mathbf{f}_{TableHigh}(i+1)+\mathbf{f}_{TableHigh}(i)}{2}\right)\\ \mathbf{S}_{Index}(i)\cdot\delta_{Step}(i,l) & if\ m=INT\left(\dfrac{\mathbf{f}_{TableHigh}(i+1)+\mathbf{f}_{TableHigh}(i)}{2}\right)\end{cases}$$

for $\mathbf{f}_{TableHigh}(i)\le m<\mathbf{f}_{TableHigh}(i+1)$, $0\le i<N_{High}$, $0\le l<L_E$

where

$$\delta_{Step}(i,l)=\begin{cases}1 & if\ (l\ge l_A)\,OR\,(\mathbf{S}'_{Index}(i)=1)\\ 0 & otherwise\end{cases},$$

and where l_A is defined according to the table below,

Table 4.119 – Table for calculation of l_A

bs_pointer	*bs_frame_class*		
	FIXFIX	*FIXVAR,VARVAR*	*VARFIX*
= 0	-1	-1	-1
=1	-1	L_E+1-*bs_pointer*	-1
>1	-1	L_E+1-*bs_pointer*	*bs_pointer*-1

and $\mathbf{S}'_{Index}(i)$ is $\mathbf{S}_{Index}(i)$ of the previous SBR frame.

The frequency resolution of the transmitted information on additional sinusoids is constant, therefore the varying frequency resolution of the envelope scalefactors needs to be considered. Since the frequency resolution of the envelope scalefactors is always coarser or as fine as that of the additional sinusoid data, the varying frequency resolution is handled according to the below:

$$\mathbf{S}_{Mapped}(m-k_x,l)=\delta_S(i,l),\ l_i\le m<u_i,\begin{cases}u_i=\mathbf{F}\left(k(i+1,l),\mathbf{r}(l)\right)\\ l_i=\mathbf{F}\left(k(i,l),\mathbf{r}(l)\right)\end{cases}$$

for $0\le i<N_{High}$, $0\le l<L_E$ where $k(i,l)$ is defined by

$$\begin{cases}\mathbf{F}(i,HI)\ge\mathbf{F}\left(k(i,l),LO\right),\mathbf{F}(i+1,HI)\le\mathbf{F}\left(k(i,l)+1,LO\right) & ,\mathbf{r}(l)=LO\\ k(i,l)=i & ,\mathbf{r}(l)=HI\end{cases}$$

and where

$$\delta_S(i,l)=\begin{cases}1 & ,1\in\left\{\mathbf{S}_{IndexMapped}(k,l):\ \ \mathbf{F}\left(k(i,l),\mathbf{r}(l)\right)\le k<\mathbf{F}\left(k(i+1,l),\mathbf{r}(l)\right)\right\}\\ 0 & ,otherwise\end{cases}.$$

In order to handle the varying frequency resolution of the envelope scalefactors, $k(i,l)$ is introduced. For a given high frequency resolution band, $k(i,l)$ gives the proper indices to the corresponding low frequency resolution band of which the former is a subset, if the current SBR envelope is of low frequency resolution. Finally, the $\delta_S(i,l)$ function returns one if any entry in the $\mathbf{S}_{IndexMapped}$ matrix is one within the given boundaries, i.e. if an additional sinusoid is present within the present frequency band.

4.6.18.7.3 Estimation of Current Envelope

In order to envelope adjust the present SBR frame, the envelope of the current SBR signal needs to be estimated. This is done according to below, dependent on the bitstream element *bs_interpol_freq*. The SBR envelope is estimated by averaging the squared complex subband samples over different time and frequency regions, given by the time/frequency grid represented by $\mathbf{t}_E$ and $\mathbf{r}$.

If interpolation (*bs_interpol_freq* = 1) is used:

$$\mathbf{E}_{Curr}(m,l) = \frac{\sum_{i=RATE\cdot\mathbf{t}_E(l)+t_{HFAdj}}^{RATE\cdot\mathbf{t}_E(l+1)-1+t_{HFAdj}} \left|\mathbf{X}_{High}(m+k_x,i)\right|^2}{\left(RATE\cdot\mathbf{t}_E(l+1)-RATE\cdot\mathbf{t}_E(l)\right)}, \quad 0\le m<M, 0\le l<L_E$$

else, no interpolation (*bs_interpol_freq* = 0):

$$\mathbf{E}_{Curr}(k-k_x,l) = \frac{\sum_{i=RATE\cdot\mathbf{t}_E(l)+t_{HFAdj}}^{RATE\cdot\mathbf{t}_E(l+1)-1+t_{HFAdj}} \sum_{j=k_l}^{k_h} \left|\mathbf{X}_{High}(j,i)\right|^2}{\left(RATE\cdot\mathbf{t}_E(l+1)-RATE\cdot\mathbf{t}_E(l)\right)\cdot(k_h-k_l+1)},$$

$$k_l \le k \le k_h, \begin{cases} k_l = \mathbf{F}(p,\mathbf{r}(l)) \\ k_h = \mathbf{F}(p+1,\mathbf{r}(l))-1 \end{cases}, 0\le p<\mathbf{n}(\mathbf{r}(l)), 0\le l<L_E$$

If interpolation is used, the energies are averaged over every QMF filterbank subband, else the energies are averaged over every frequency band. In either case, the energies are stored with the frequency resolution of the QMF filterbank. Hence the $\mathbf{E}_{Curr}$ matrix has L_E columns (one for every SBR envelope) and M rows (the number of QMF subbands covered by the SBR range).

4.6.18.7.4 Calculation of Levels of Additional HF Signal Components

The noise floor scalefactor is the ratio between the energy of the noise to be added to the envelope adjusted HF generated signal $\mathbf{X}_{High}$ and the energy of the same. Hence, in order to add the correct amount of noise, the noise floor scalefactor needs to be converted to a proper amplitude value, according to the following.

$$\mathbf{Q}_M(m,l) = \sqrt{\mathbf{E}_{OrigMapped}(m,l)\cdot\frac{\mathbf{Q}_{Mapped}(m,l)}{1+\mathbf{Q}_{Mapped}(m,l)}}, \quad 0\le m<M, 0\le l<L_E$$

The level of the sinusoids are derived from the SBR envelope scalefactors according to below.

$$\mathbf{S}_M(m,l) = \sqrt{\mathbf{E}_{OrigMapped}(m,l)\cdot\frac{\mathbf{S}_{Mapped}(m,l)}{1+\mathbf{Q}_{Mapped}(m,l)}}, \quad 0\le m<M, 0\le l<L_E$$

4.6.18.7.5 Calculation of Gain

The gain to be applied for the subband samples in order to retain the correct envelope is calculated according to below. The level of additional sinusoids, as well as the level of the additional added noise, are taken into account.

$$\mathbf{G}(m,l)=\begin{cases}\sqrt{\dfrac{\mathbf{E}_{OrigMapped}(m,l)}{\left(\varepsilon+\mathbf{E}_{Curr}(m,l)\right)\cdot\left(1+\delta(l)\cdot\mathbf{Q}_{Mapped}(m,l)\right)}} & if\quad \mathbf{S}_M(m,l)=0\\ \sqrt{\dfrac{\mathbf{E}_{OrigMapped}(m,l)}{\left(\varepsilon+\mathbf{E}_{Curr}(m,l)\right)}\cdot\dfrac{\mathbf{Q}_{Mapped}(m,l)}{\left(1+\mathbf{Q}_{Mapped}(m,l)\right)}} & if\quad \mathbf{S}_M(m,l)\neq 0\end{cases}, 0\le m<M, 0\le l<L_E$$

where

$$\delta(l)=\begin{cases}0 & if\ l=l_A\ OR\ l=l_{APrev}\\ 1 & otherwise\end{cases},$$

and where

$$l_{APrev}=\begin{cases}0 & if\ l'_A=L'_E\\ -1 & otherwise\end{cases}$$

is introduced, derived from l'_A and L'_E, which are the l_A and L_E values of the previous SBR frame.

In order to avoid unwanted noise substitution, the gain values are limited according to the following. Furthermore, the total level of a particular limiter band is adjusted in order to compensate for the energy-loss imposed by the limiter.

$$\mathbf{G}_{Max_{Temp}}(k,l)=\sqrt{\frac{\varepsilon_0+\sum_{i=\mathbf{f}_{TableLim}(k)}^{\mathbf{f}_{TableLim}(k+1)-1}\mathbf{E}_{OrigMapped}(i,l)}{\varepsilon_0+\sum_{i=\mathbf{f}_{TableLim}(k)}^{\mathbf{f}_{TableLim}(k+1)-1}\mathbf{E}_{Curr}(i,l)}}\cdot\mathbf{limGain}(bs\_limiter\_gains),\quad 0\le k<N_L, 0\le l<L_E$$

$$\mathbf{G}_{Max}(m,l)=\min\left(\mathbf{G}_{Max_{Temp}}(k(m),l),10^5\right),\quad 0\le m<M, 0\le l<L_E$$

where $k(m)$ is defined by $\mathbf{f}_{TableLim}(k(m))\le m<\mathbf{f}_{TableLim}(k(m)+1)$,

and where $\mathbf{limGain}=\left[0.70795,\ 1.0,\ 1.41254,\ 10^{10}\right]$, and where $\varepsilon_0 = 10^{-12}$.

The additional noise added to the HF generated signal is limited in proportion to the energy lost due to the limitation of the gain values, according to the following:

$$\mathbf{Q}_{M_{Lim}}(m,l)=\min\left(\mathbf{Q}_M(m,l),\mathbf{Q}_M(m,l)\cdot\frac{\mathbf{G}_{Max}(m,l)}{\mathbf{G}(m,l)}\right),\quad 0\le m<M, 0\le l<L_E$$

The gain values are limited according to the following:

$$\mathbf{G}_{Lim}(m,l) = \min\left(\mathbf{G}(m,l), \mathbf{G}_{Max}(m,l)\right), \quad 0 \le m < M, 0 \le l < L_E$$

As mentioned above, the limiter is compensated for by adjusting the total gain for a limiter band, in proportion to the lost energy due to limitation. This is calculated according to the following:

$$\mathbf{G}_{Boost_{Temp}}(k,l) = \sqrt{\frac{\varepsilon_0 + \sum_{i=\mathbf{f}_{TableLim}(k)}^{\mathbf{f}_{TableLim}(k+1)-1} \mathbf{E}_{OrigMapped}(i,l)}{\varepsilon_0 + \sum_{i=\mathbf{f}_{TableLim}(k)}^{\mathbf{f}_{TableLim}(k+1)-1} \left(\mathbf{E}_{Curr}(i,l)\cdot\mathbf{G}_{Lim}^2(i,l) + \mathbf{S}_M^2(i,l) + \delta\left(\mathbf{S}_M(i,l),l\right)\cdot\mathbf{Q}_M^2(i,l)\right)}}$$

for $0 \le k < N_L, 0 \le l < L_E$ where, $\delta\left(\mathbf{S}_M(i,l),l\right) = \begin{cases} 0 & , \mathbf{S}_M(i,l) \ne 0 \; OR \; l = l_A \\ 1 & , otherwise \end{cases}$.

The compensation, or boost factor, is limited in order not to get too high energy values, according to:

$$\mathbf{G}_{Boost}(m,l) = \min\left(\mathbf{G}_{Boost_{Temp}}(k(m),l), 1.584893192\right), \quad 0 \le m < M, 0 \le l < L_E$$

where $k(m)$ is defined by $\mathbf{f}_{TableLim}(k(m)) \le m < \mathbf{f}_{TableLim}(k(m)+1)$, and where $\varepsilon_0 = 10^{-12}$.

This compensation is applied to the gain, the noise floor scalefactors and the sinusoid levels, according to below.

$$\mathbf{G}_{LimBoost}(m,l) = \mathbf{G}_{Lim}(m,l)\cdot\mathbf{G}_{Boost}(m,l), \quad 0 \le m < M, 0 \le l < L_E$$

$$\mathbf{Q}_{M_{LimBoost}}(m,l) = \mathbf{Q}_{M_{Lim}}(m,l)\cdot\mathbf{G}_{Boost}(m,l), \quad 0 \le m < M, 0 \le l < L_E$$

$$\mathbf{S}_{MBoost}(m,l) = \mathbf{S}_M(m,l)\cdot\mathbf{G}_{Boost}(m,l), \quad 0 \le m < M, 0 \le l < L_E$$

4.6.18.7.6 Assembling HF Signals

Analogous to the mapping of SBR envelope data and noise floor data to a higher time and frequency resolution, the gain values, representing a time-span of several QMF subsamples, are mapped to the highest time-resolution available for the envelope adjustment, i.e. to the individual QMF subsamples within the current SBR frame.

The gain values to be applied to the subband samples are smoothed using the filter $\mathbf{h}_{Smooth}$. The variable h_{SL} is used to control whether smoothing is applied or not, according to:

$h_{SL} = \begin{cases} 4 & , bs\_smoothing\_mode = 0 \\ 0 & , bs\_smoothing\_mode = 1 \end{cases}$ and the filter used is defined as following:

$$\mathbf{h}_{Smooth} = \begin{bmatrix} 0.33333333333333 \\ 0.30150283239582 \\ 0.21816949906249 \\ 0.11516383427084 \\ 0.03183050093751 \end{bmatrix}.$$

The smoothed gain values $\mathbf{G}_{Filt}$ are calculated according to the following equation:

$$\mathbf{G}_{Temp}(m, i + h_{SL}) = \mathbf{G}_{LimBoost}(m, l),\ RATE \cdot \mathbf{t}_E(l) \le i < RATE \cdot \mathbf{t}_E(l+1), 0 \le l < L_E, 0 \le m < M$$

$$\mathbf{G}_{Filt}(m,i) = \begin{cases} \sum_{j=0}^{h_{SL}} \mathbf{G}_{Temp}(m, i - j + h_{SL}) \cdot \mathbf{h}_{Smooth}(j) & if\ l \neq l_A\ AND\ h_{SL} \neq 0 \\ \mathbf{G}_{Temp}(m, i + h_{SL}) & otherwise \end{cases},$$

for $rate \cdot \mathbf{t}_E(l) \le i < rate \cdot \mathbf{t}_E(l+1), 0 \le m < M, 0 \le l < L_E$.

The first h_{SL} columns of the $\mathbf{G}_{Temp}$ matrix are the last h_{SL} columns of the $\mathbf{G}_{Temp}$ matrix of the previous SBR frame, unless the reset-flag is set (*reset=1*) for which case the first h_{SL} columns of the $\mathbf{G}_{Temp}$ matrix are equal to $\mathbf{G}_{LimBoost}(m, 0)$ for all QMF subbands within the SBR range.

The smoothed gain values are applied to the input subband matrix $\mathbf{X}_{High}$, for all SBR envelopes of the current SBR frame, according to:

$$\mathbf{W}_1(m,i) = \mathbf{G}_{Filt}(m,i) \cdot \mathbf{X}_{High}(m + lsb, i + t_{HfAdj}), \quad RATE \cdot \mathbf{t}_E(0) \le i < RATE \cdot \mathbf{t}_E(L_E), 0 \le m < M$$

The level of the noise is smoothed similar to the smoothing of the gain values using the filter $\mathbf{h}_{Smooth}$ of length h_{SL}

$$\mathbf{Q}_{Temp}(m, i + h_{SL}) = \mathbf{Q}_{M_{LimBoost}}(m, l),\ RATE \cdot \mathbf{t}_E(l) \le i < RATE \cdot \mathbf{t}_E(l+1), 0 \le l < L_E, 0 \le m < M$$

$$\mathbf{Q}_{Filt}(m,i) = \begin{cases} \sum_{j=0}^{h_{SL}} \mathbf{Q}_{Temp}(m, i - j + h_{SL}) \cdot \mathbf{h}_{Smooth}(j) & if\ l \neq l_A\ AND\ \ l \neq l_{APrev}\ AND\ \mathbf{S}_{MBoost}(m,l) = 0 \\ 0 & otherwise \end{cases}$$

for $RATE \cdot \mathbf{t}_E(l) \le i < RATE \cdot \mathbf{t}_E(l+1), 0 \le m < M, 0 \le l < L_E$.

The first h_{SL} columns of the $\mathbf{Q}_{Temp}$ matrix are the last h_{SL} columns of the $\mathbf{Q}_{Temp}$ matrix of the previous SBR frame, unless the reset-flag is set (*reset=1*) for which case the first h_{SL} columns of the $\mathbf{Q}_{Temp}$ matrix are equal to $\mathbf{Q}_{M_{LimBoost}}(m, 0)$ for all QMF subbands within the SBR range.

The noise, based on the noise table $\mathbf{V}$ (Table 4.A.88), is added to the output according to:

$$\begin{cases} Re\{\mathbf{W}_2(m,i)\} = Re\{\mathbf{W}_1(m,i)\} + \mathbf{Q}_{Filt}(m,i) \cdot \mathbf{V}(0, f_{IndexNoise}(i)) \\ Im\{\mathbf{W}_2(m,i)\} = Im\{\mathbf{W}_1(m,i)\} + \mathbf{Q}_{Filt}(m,i) \cdot \mathbf{V}(1, f_{IndexNoise}(i)) \end{cases}, \begin{cases} RATE \cdot \mathbf{t}_E(l) \le i < RATE \cdot \mathbf{t}_E(l+1), 0 \le l < L_E \\ 0 \le m < M \end{cases}$$

where

$$f_{IndexNoise}(i) = \left(index_{Noise} + \left(i - RATE \cdot \mathbf{t}_E(0)\right) \cdot M + m + 1\right) \bmod (512),$$

and $index_{Noise}$ is the last $f_{IndexNoise}$ from the previous SBR frame, unless the reset-flag is set ($reset=1$) for which case $f_{IndexNoise} = 0$.

In the equation above, $\mathbf{V}\left(0, f_{IndexNoise(i)}\right) = \varphi_{\mathrm{Re},noise}\left(f_{IndexNoise}(i)\right)$ and $\mathbf{V}\left(1, f_{IndexNoise(i)}\right) = \varphi_{\mathrm{Im},noise}\left(f_{IndexNoise}(i)\right)$, where $\varphi_{\mathrm{Re},noise}(i)$ and $\varphi_{\mathrm{Im},noise}(i)$ are defined in Table 4.A.88.

The sinusoids are added at the level given by $\mathbf{S}_{MBoost}(m,l)$ for the QMF subbands indicated by $\mathbf{S}_{IndexMapped}(m,l)$. This gives the final output QMF matrix $\mathbf{Y}$, according to:

$$\begin{cases} Re\left\{\mathbf{Y}\left(m + k_x, i + t_{HFAdj}\right)\right\} = Re\left\{\mathbf{W}_2(m,i)\right\} + \boldsymbol{\psi}_{Re}(m,l,i) \\ Im\left\{\mathbf{Y}\left(m + k_x, i + t_{HFAdj}\right)\right\} = Im\left\{\mathbf{W}_2(m,i)\right\} + \boldsymbol{\psi}_{Im}(m,l,i) \end{cases}, \begin{cases} RATE \cdot \mathbf{t}_E(l) \le i < RATE \cdot \mathbf{t}_E(l+1), 0 \le l < L_E \\ 0 \le m < M \end{cases}$$

where

$$\boldsymbol{\psi}_{Re}(m,l,i) = \mathbf{S}_{IndexMapped}(m,l) \cdot \mathbf{S}_{MBoost}(m,l) \cdot \boldsymbol{\varphi}_{Re,sin}\left(f_{IndexSine}(i)\right)$$

$$\boldsymbol{\psi}_{Im}(m,l,i) = \mathbf{S}_{IndexMapped}(m,l) \cdot \mathbf{S}_{MBoost}(m,l) \cdot (-1)^{m+lsb} \cdot \boldsymbol{\varphi}_{Im,sin}\left(f_{IndexSine}(i)\right)$$

where $\boldsymbol{\varphi}$ and $f_{IndexSine}$ are defined below as:

$$\begin{cases} \boldsymbol{\varphi}_{Re,sin} = [1,0,-1,0] \\ \boldsymbol{\varphi}_{Im,sin} = [0,1,0,-1] \end{cases} \text{ and } f_{IndexSine}(i) = (index_{Sine} + i) \bmod (4),$$

$index_{Sine}$ is the last $f_{IndexSine}$ from the previous SBR frame.

4.6.18.8 Low power SBR Tool

4.6.18.8.1 Introduction

This subclause outlines the differences for the implementation of the low power version of the SBR tool compared to the high quality version of the SBR tool outlined in subclauses 4.6.18.1 to 4.6.18.7. The low power SBR tool operates on real-valued signals, and hence a real-valued filterbank is used and all references to the imaginary part of variables in subclauses 4.6.18.1 to 4.6.18.7 should be ignored. Furthermore, the low power SBR tool incorporates additional modules in order to reduce aliasing introduced due to the real-valued processing.

In Figure 4.47 a block diagram of the low power SBR decoder displays how the additional modules are interconnected into the SBR decoder.

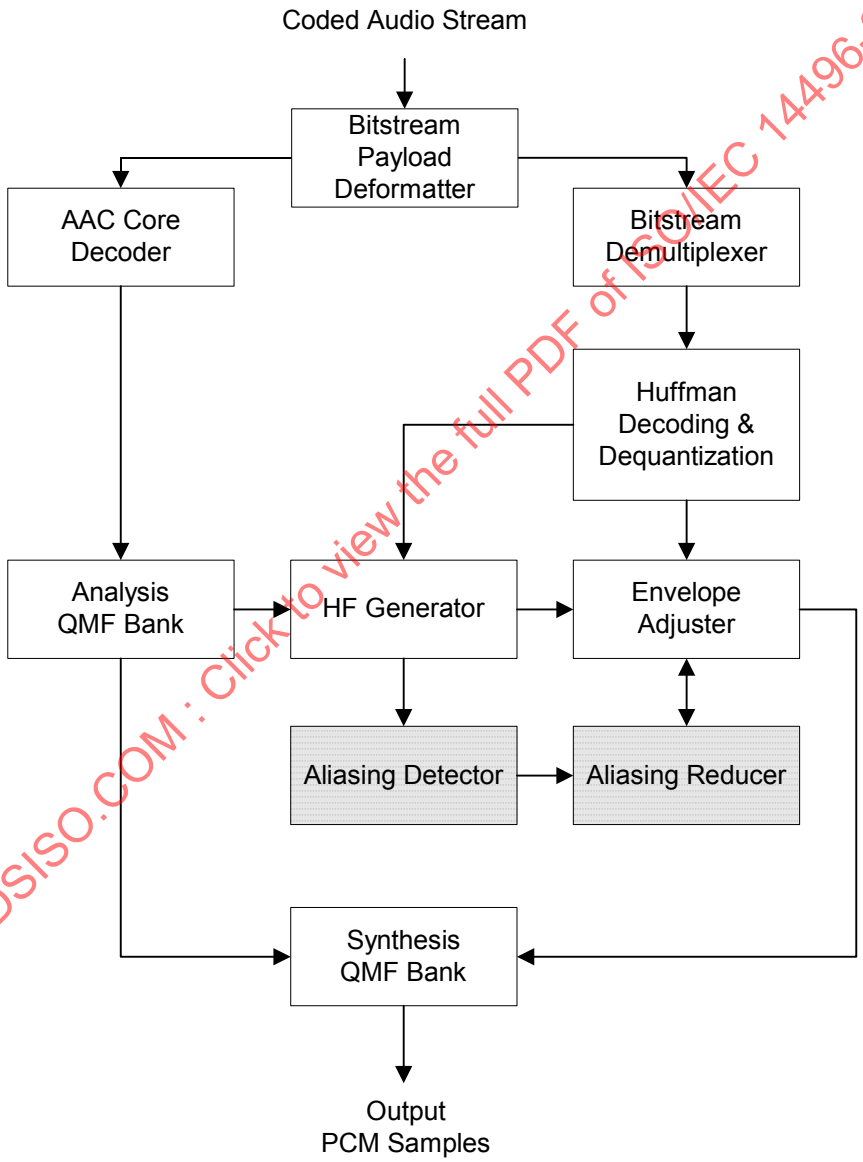


Figure 4.47 – Block diagram of the SBR low power decoder

4.6.18.8.2 Low power SBR Tool Filterbanks

For the low power SBR tool, real-valued filterbanks are used. Hence, the filterbanks outlined in subclause 4.6.18.4 should be replaced by the following analysis and synthesis filterbanks.

4.6.18.8.2.1 Real-valued Analysis Filterbank

The real-valued QMF bank is used to split the time domain signal output from the core decoder into 32 subband signals. The output from the filterbank, i.e. the subband samples, are real-valued and critically sampled. The flowchart of the operation is given in Figure 4.48. The filtering involves the following steps, where an array $\mathbf{x}$ consisting of 320 time domain input samples is assumed. A higher index into the array corresponds to older samples.

- Shift the samples in the array $\mathbf{x}$ by 32 positions. The oldest 32 samples are discarded and 32 new samples are stored in positions 0 to 31.
- Multiply the samples of array $\mathbf{x}$ by every other coefficient of window $\mathbf{c}$. The window coefficients can be found in Table 4.A.87.
- Sum the samples according to the formula in the flowchart to create the 64-element array $\mathbf{u}$.
- Calculate new 32 subband samples by the matrix operation $\mathbf{M}_r\mathbf{u}$, where

$$\mathbf{M}_r(k,n) = 2 \cdot \cos\left(\frac{\pi \cdot (k+0.5) \cdot (2 \cdot n - 96)}{64}\right), \begin{cases} 0 \le k < 32 \\ 0 \le n < 64 \end{cases}$$

Every loop in the flowchart produces 32 subband samples, each representing the output from one filterbank subband. For every SBR frame the filterbank will produce $numTimeSlots \cdot RATE$ subband samples for every subband, corresponding to a time domain signal of length $numTimeSlots \cdot RATE \cdot 32$ samples. In the flowchart $\mathbf{X}_{Low}[k][l]$ corresponds to subband sample l of QMF subband k.

4.6.18.8.2.2 Real-valued Synthesis Filterbank

Synthesis filtering of the SBR-processed subband signals is achieved using a 64-subband QMF bank. The output from the filterbank are real-valued time domain samples. The process is given by the flowchart in Figure 4.49. The synthesis filtering comprises the following steps, where an array $\mathbf{v}$ consisting of 1280 samples is assumed:

- Shift the samples in the array $\mathbf{v}$ by 128 positions. The oldest 128 samples are discarded.
- The 64 new subband samples are multiplied by the matrix $\mathbf{N}_r$, where

$$\mathbf{N}_r(k,n) = \frac{1}{32} \cdot \cos\left(\frac{\pi \cdot (k+0.5) \cdot (2 \cdot n - 64)}{128}\right), \begin{cases} 0 \le k < 64 \\ 0 \le n < 128 \end{cases}$$

  The output from this operation is stored in the positions 0 to 127 of array $\mathbf{v}$.
- Extract samples from $\mathbf{v}$ according to the flowchart in Figure 4.49 to create the 640-element array $\mathbf{g}$.
- Multiply the samples of array $\mathbf{g}$ by window $\mathbf{c}$ to produce array $\mathbf{w}$. The window coefficients of $\mathbf{c}$ can be found in Table 4.A.87, and are the same as for the analysis filterbank.
- Calculate 64 new output samples by summation of samples from array $\mathbf{w}$ according to the formula in the flowchart of Figure 4.49.

Every SBR frame produces an output of $numTimeSlots \cdot RATE \cdot 64$ time domain samples. In the flowchart below $\mathbf{X}$[k][l] corresponds to subband sample l of QMF subband k, and every new loop produces 64 time domain samples as output.

4.6.18.8.2.3 Downsampled Real-valued Synthesis Filterbank

Synthesis filtering of the SBR-processed subband signals is achieved using a 32-channel QMF bank. The output from the filterbank is real-valued time domain samples. The process is given by the flowchart in Figure 4.50. The synthesis filtering comprises the following steps, where an array $\mathbf{v}$ consisting of 640 samples is assumed:

- Shift the samples in the array $\mathbf{v}$ by 64 positions. The oldest 64 samples are discarded.
- The 32 new subband samples are multiplied by the matrix $\mathbf{N}_r$, where

$$\mathbf{N}_r(k,n) = \frac{1}{32} \cdot \cos\left(\frac{\pi \cdot (k+0.5) \cdot (2 \cdot n - 32)}{64} \right), \begin{cases} 0 \le k < 32 \\ 0 \le n < 64 \end{cases}$$

  The output from this operation is stored in the positions 0 to 61 of array $\mathbf{v}$.
- Extract samples from $\mathbf{v}$ according to the flowchart in Figure 4.50 to create the 320-element array $\mathbf{g}$.
- Multiply the samples of array $\mathbf{g}$ by every other coefficient of window $\mathbf{c}$ to produce array $\mathbf{w}$. The window coefficients of $\mathbf{c}$ can be found in Table 4.A.87, and are the same as for the analysis filterbank.
- Calculate 32 new output samples by summation of samples from array $\mathbf{w}$ according to the formula in the flowchart of Figure 4.50.

Every SBR frame produces an output of $numTimeSlots \cdot RATE \cdot 32$ time domain samples. In the flowchart of Figure 4.50 $\mathbf{X}$[k][l] corresponds to subband sample l of QMF subband k, and every new loop produces 32 time domain samples as output.

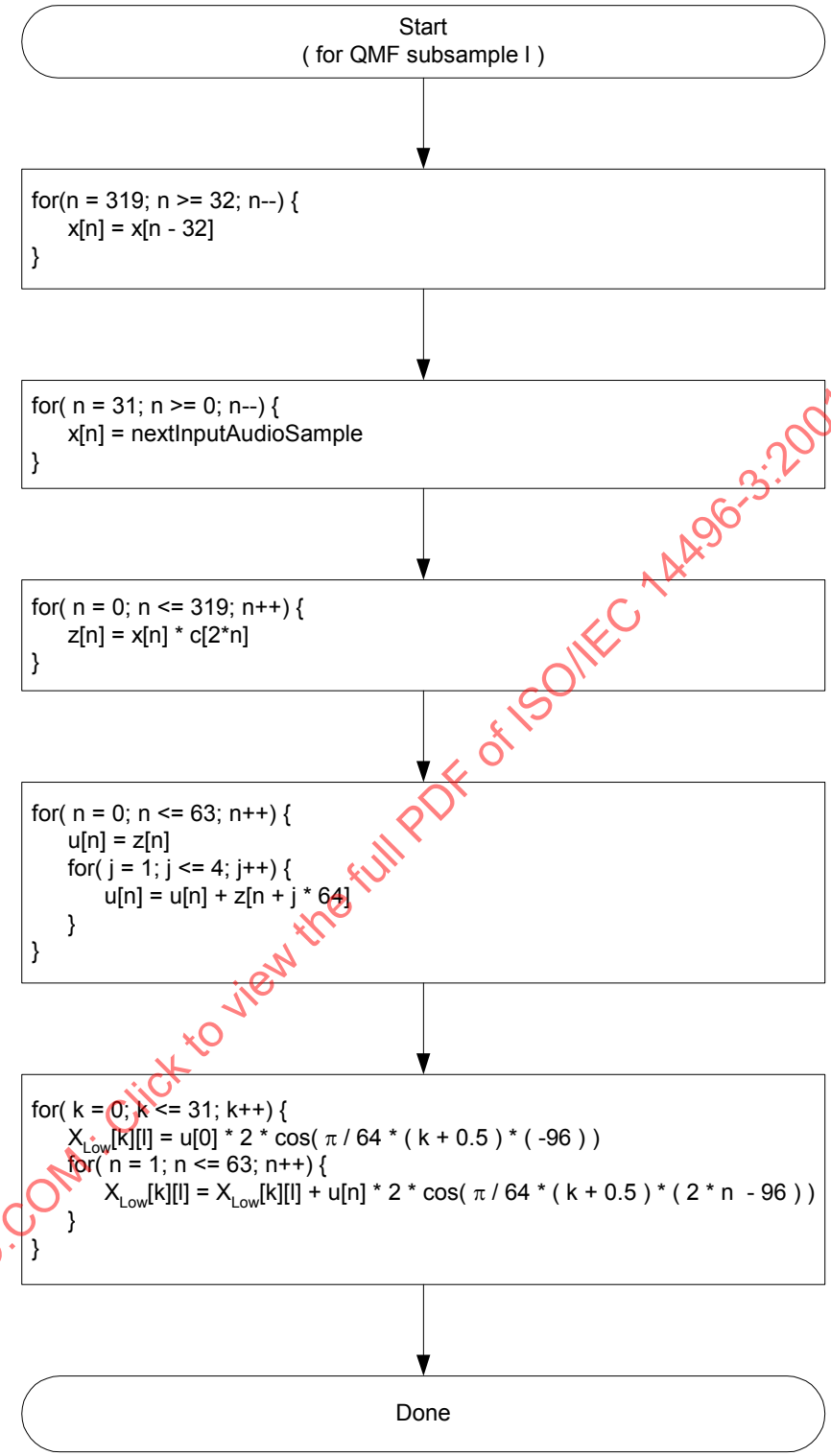


Figure 4.48 – Flowchart of decoder real-valued analysis QMF bank

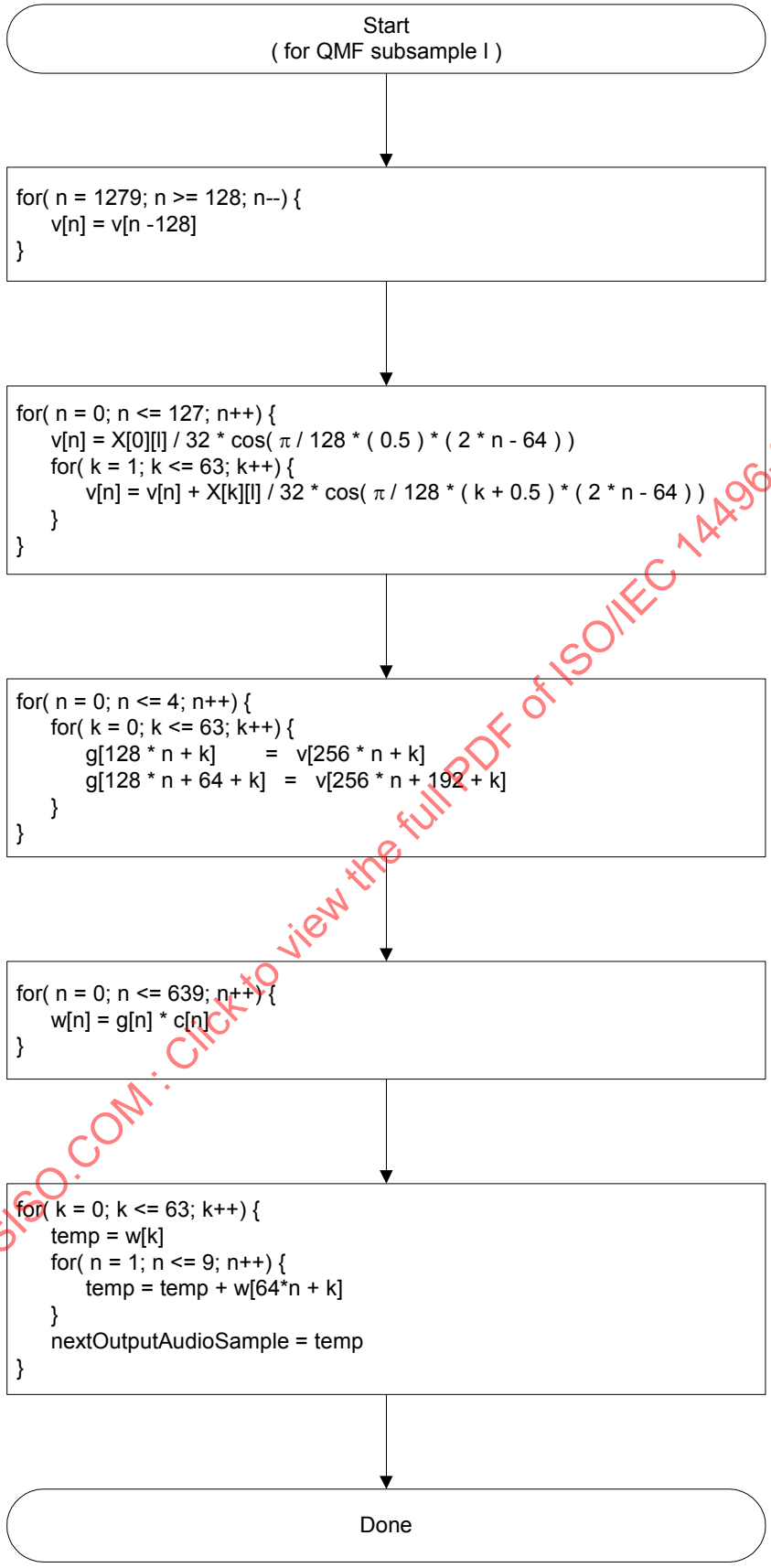


Figure 4.49 – Flowchart of decoder real-valued synthesis QMF bank

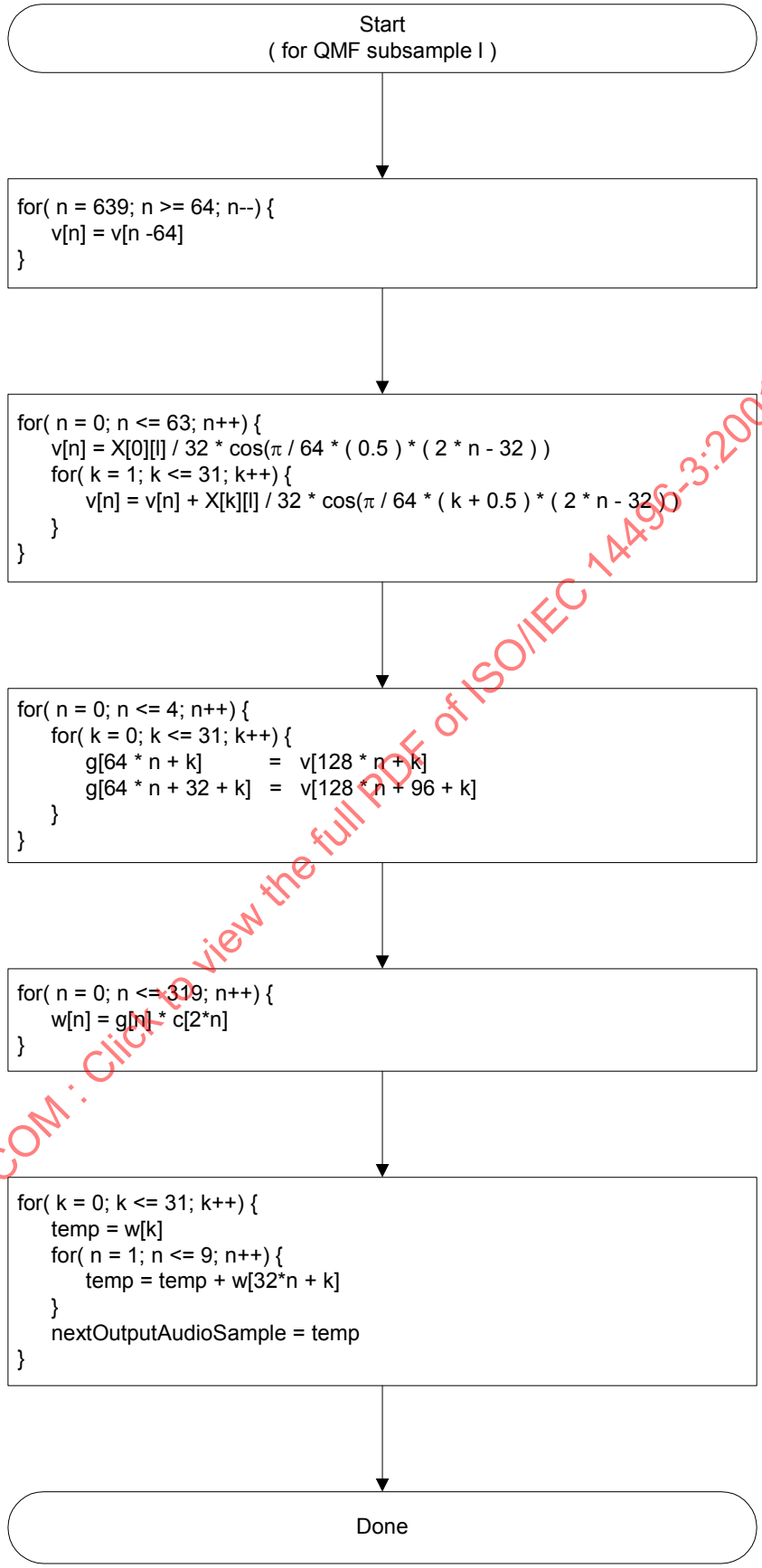


Figure 4.50 – Flowchart of decoder downsampled real-valued synthesis QMF bank

4.6.18.8.3 Aliasing Detection

In order to minimize the introduction of aliasing by the envelope adjuster, the QMF subbands where strong aliasing will potentially be introduced are identified. The detection module uses data from the HF Generation module outlined in subclause 4.6.18.6, and from the HF Adjustment module outlined in subclause 4.6.18.7.

The aliasing detection algorithm calculates the reflection coefficient for every subband in the low-band.

$$\mathbf{ref}(k) = \begin{cases} \min\left(\max\left(-\frac{\phi_k(0,1)}{\phi_k(1,1)}, -1\right), 1\right) & \text{if } \phi_k(1,1) \neq 0 \\ 0 & \text{otherwise} \end{cases}, \quad 0 \leq k < k_0$$

Given the reflection coefficients **ref**, the degree of aliasing **deg** is calculated for the low-band according to the flowchart given in Figure 4.51.

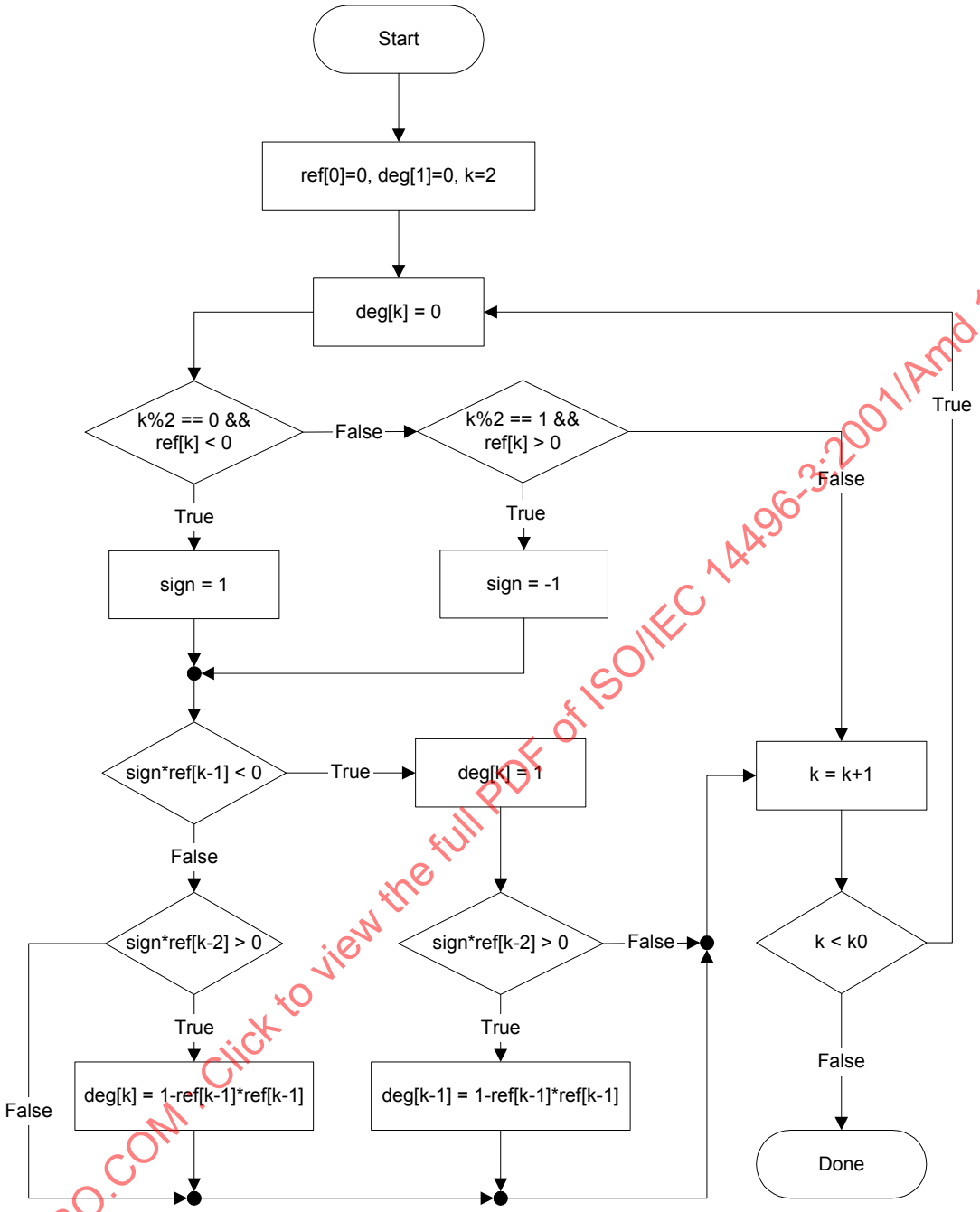


Figure 4.51 – Flowchart of the calculation of the aliasing degree

The degree of aliasing in the highband is obtained by using the patch information available in subclause 4.6.18.6, according to:

$$\mathbf{degPatched}(k) = \begin{cases} 0 & if\ x = 0 \\ \mathbf{deg}(p) & otherwise \end{cases}$$

where

$$\begin{cases} k = k_x + x + \sum_{q=0}^{i-1} \mathbf{patchNumSubbands}(q) \\ p = \mathbf{patchStartSubband}(i) + x \end{cases}, \quad 0 \le x < \mathbf{patchNumSubbands}(i), 0 \le i < numPatches\ .$$

Furthermore, the aliasing reduction algorithm needs a table to indicate the grouping of the gain-values. This table $\mathbf{F}_{Group}$ has L_E vectors of length $2 \cdot \mathbf{n}_G(l)$ representing the desired gain grouping for every SBR envelope of the SBR frame. It is calculated by the flowchart given in Figure 4.52. The table differs from previous tables in the text since it has individual start and stop indices for every group in frequency, whereas for the previous tables, the stop index of the previous group is taken as the start index of the current group. Hence, a vector representing N_G groups is $2N_G$ entries long, whereas a table of the style previously used would have been N_G +1 entries long. The stop index of a group is exclusive, i.e. the stop index subband is not included in the group.

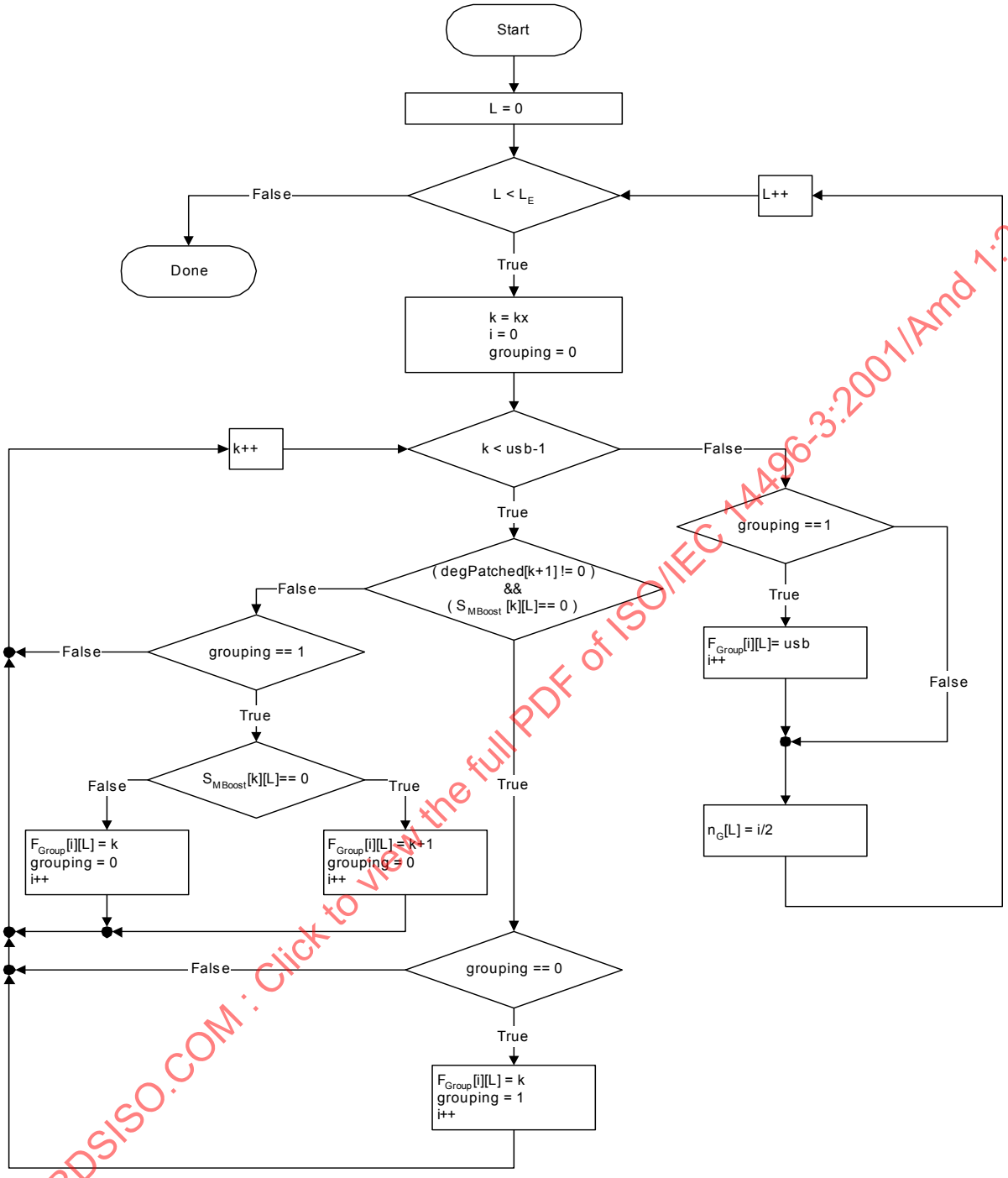


Figure 4.52 – Flowchart of the calculation of the gain groups

4.6.18.8.4 Modification of the Energy Calculation

Since the low power version of the SBR tool does not use complex-valued representation of signals, a modification of the energy calculation in subclause 4.6.18.7.3 is required. The equations given:

$$\mathbf{E}_{Curr}(m,l) = \frac{\sum_{i=RATE\cdot\mathbf{t}_E(l)+t_{HFAdj}}^{RATE\cdot\mathbf{t}_E(l+1)-1+t_{HFAdj}} \left|\mathbf{X}_{High}(m+k_x,i)\right|^2}{\left(RATE\cdot\mathbf{t}_E(l+1)-RATE\cdot\mathbf{t}_E(l)\right)} \quad , \quad 0\le m<M, 0\le l<L_E$$

and

$$\mathbf{E}_{Curr}(k-k_x,l) = \frac{\sum_{i=RATE\cdot\mathbf{t}_E(l)+t_{HFAdj}}^{RATE\cdot\mathbf{t}_E(l+1)-1+t_{HFAdj}} \sum_{j=k_l}^{k_h} \left|\mathbf{X}_{High}(j,i)\right|^2}{\left(RATE\cdot\mathbf{t}_E(l+1)-RATE\cdot\mathbf{t}_E(l)\right)\cdot(k_h-k_l+1)}$$

is replaced by

$$\mathbf{E}_{Curr}(m,l) = \frac{2\cdot\sum_{i=RATE\cdot\mathbf{t}_E(l)+t_{HFAdj}}^{RATE\cdot\mathbf{t}_E(l+1)-1+t_{HFAdj}} \left|\mathbf{X}_{High}(m+k_x,i)\right|^2}{\left(RATE\cdot\mathbf{t}_E(l+1)-RATE\cdot\mathbf{t}_E(l)\right)} \quad , \quad 0\le m<M, 0\le l<L_E$$

and

$$\mathbf{E}_{Curr}(k-k_x,l) = \frac{2\cdot\sum_{i=RATE\cdot\mathbf{t}_E(l)+t_{HFAdj}}^{RATE\cdot\mathbf{t}_E(l+1)-1+t_{HFAdj}} \sum_{j=k_l}^{k_h} \left|\mathbf{X}_{High}(j,i)\right|^2}{\left(RATE\cdot\mathbf{t}_E(l+1)-RATE\cdot\mathbf{t}_E(l)\right)\cdot(k_h-k_l+1)} .$$

4.6.18.8.5 Aliasing Reduction

The aliasing reduction module re-calculates gain values calculated by the (real-valued) HF Adjustment module outlined in subclause 4.6.18.7. The variables available in the HF Adjustment subclause 4.6.18.7.5 are used by the aliasing reduction module outlined below. For the low power implementation, the output variable from the aliasing reduction module $\mathbf{G}_A$, as calculated below, should be used instead of $\mathbf{G}_{LimBoost}$ in the subsequent parts of the HF Adjustment module, i.e. subclause 4.6.18.7.6.

The energy of the subband signals in the affected subbands, if the calculated gain values $\mathbf{G}_{LimBoost}$ were used, would be:

$$\mathbf{E}_{Total}(k,l) = \sum_{i=\mathbf{F}_{Group}(2\cdot k,l)-k_x}^{\mathbf{F}_{Group}(2\cdot k+1,l)-1-k_x} \mathbf{G}^2_{LimBoost}(i,l)\cdot\mathbf{E}_{Curr}(i,l) \quad , \quad 0\le k<\mathbf{n}_G(l), 0\le l<L_E$$

Given this target energy $\mathbf{E}_{Total}$, a target gain value is calculated as follows:

$$\mathbf{G}^2_{Target}(k,l) = \frac{\mathbf{E}_{Total}(k,l)}{\varepsilon_0 + \sum_{i=\mathbf{F}_{Group}(2\cdot k,l)-k_x}^{\mathbf{F}_{Group}(2\cdot k+1,l)-1-k_x} \mathbf{E}_{Curr}(i,l)} \quad , \quad 0\le k<\mathbf{n}_G(l), 0\le l<L_E .$$

Given the above calculated target gain, a new gain value is calculated as a weighted sum of the original gain value and the newly calculated target gain:

$$\mathbf{G}^2_{ARtemp}(m-k_x,l)=\alpha(m)\cdot\mathbf{G}^2_{Target}(k,l)+(1-\alpha(m))\cdot\mathbf{G}^2_{LimBoost}(m-k_x,l),\quad \mathbf{F}_{Group}(2\cdot k,l)\le m<\mathbf{F}_{Group}(2\cdot k+1,l)$$

$$0\le k<\mathbf{n}_G(l),0\le l<L_E$$

where

$$\alpha(m)=\begin{cases}\max(\mathbf{degPatched}(m),\mathbf{degPatched}(m+1)) & ,if\ m<M+k_x-1\\ \mathbf{degPatched}(m) & ,if\ m=M+k_x-1\end{cases}$$

where $\mathbf{degPatched}(m)$, calculated in the aliasing detection part, is used as the degree of gain equalization between subband m-1 and subband m.

A new energy value is calculated based on the new gain values, according to:

$$\mathbf{E}_{TotalNew}(k,l)=\sum_{i=\mathbf{F}_{Group}(2\cdot k,l)-k_x}^{\mathbf{F}_{Group}(2\cdot k+1,l)-1-k_x}\mathbf{G}^2_{ARtemp}(i,l)\cdot\mathbf{E}_{Curr}(i,l)\quad,\quad 0\le k<\mathbf{n}_G(l),0\le l<L_E$$

In order to retain the correct output energy, while limiting the gain-adjustment in order to avoid introduction of aliasing, the gain value $\mathbf{G}_A$ is calculated according to:

$$\mathbf{G}_A(m-k_x,l)=\begin{cases}\mathbf{G}_{ARtemp}(m-k_x,l)\cdot\sqrt{\dfrac{\mathbf{E}_{Total}(\kappa(m),l)}{\varepsilon_0+\mathbf{E}_{TotalNew}(\kappa(m),l)}} & ,m\in A_G(l)\\ \mathbf{G}_{LimBoost}(m-k_x,l) & ,m\notin A_G(l)\end{cases},\quad k_x\le m<k_x+M,0\le l<L_E$$

where

$$A_G(l)=\bigcup_{k=0}^{\mathbf{n}_G(l)-1}\{m:\mathbf{F}_{Group}(2\cdot k,l)\le m<\mathbf{F}_{Group}(2\cdot k+1,l)\},$$

and for $m\in A_G(l)$ define $\kappa(m)$ by $\mathbf{F}_{Group}(2\cdot\kappa(m),l)\le m<\mathbf{F}_{Group}(2\cdot\kappa(m)+1,l)$.

The $\mathbf{G}_A$ values are the new gain values that should be used instead of the $\mathbf{G}_{LimBoost}$ values in subclause 4.6.18.7.6.

For the low power version of the SBR tool, the gain smoothing process described in 4.6.18.7.6 is not applied regardless of the value of *bs_smoothing_mode*.

For the sinusoids added in subclause 4.6.18.7.6, modifications are required for the low-power version of the SBR tool. The following equations:

$$\begin{cases}Re\{\mathbf{Y}(m+k_x,i+t_{HFAdj})\}=Re\{\mathbf{W}_2(m,i)\}+\boldsymbol{\psi}_{Re}(m,l,i)\\ Im\{\mathbf{Y}(m+k_x,i+t_{HFAdj})\}=Im\{\mathbf{W}_2(m,i)\}+\boldsymbol{\psi}_{Im}(m,l,i)\end{cases},\quad\begin{cases}RATE\cdot\mathbf{t}_E(l)\le i<RATE\cdot\mathbf{t}_E(l+1),0\le l<L_E\\ 0\le m<M\end{cases}$$

where

$$\boldsymbol{\psi}_{Re}(m,l,i) = \mathbf{S}_{IndexMapped}(m,l) \cdot \mathbf{S}_{MBoost}(m,l) \cdot \boldsymbol{\varphi}_{Re,sin}\left(f_{IndexSine}(i)\right)$$

$$\boldsymbol{\psi}_{Im}(m,l,i) = \mathbf{S}_{IndexMapped}(m,l) \cdot \mathbf{S}_{MBoost}(m,l) \cdot (-1)^{m+lsb} \cdot \boldsymbol{\varphi}_{Im,sin}\left(f_{IndexSine}(i)\right)$$

is replaced by:

$$Re\left\{\mathbf{Y}\left(m+k_x, i+t_{HFAdj}\right)\right\} = \begin{cases} \boldsymbol{\psi}_m(m,l,i) & ,m=-1 \\ Re\left\{\mathbf{W}_2(m,i)\right\} + \boldsymbol{\psi}_m(m,l,i) & ,0 \le m < M \\ \boldsymbol{\psi}_m(m,l,i) & ,m = M \ \ AND \ m+k_x < 64 \end{cases}$$

where

$$-1 \le m \le M$$

$$rate \cdot \mathbf{t}_E(l) \le i < rate \cdot \mathbf{t}_E(l+1), 0 \le l < L_E$$

$$\boldsymbol{\psi}_m(m,l,i) = \boldsymbol{\psi}_{Re}(m,l,i) - 0.00815 \cdot (-1)^{m+lsb} \cdot \left(\boldsymbol{\psi}_{Re}(m-1,l,i-1) + \boldsymbol{\psi}_{Re}(m+1,l,i+1)\right)$$

$$\boldsymbol{\psi}_{Re}(m,l,i) = \mathbf{S}_{IndexMapped}(m,l) \cdot \mathbf{S}_{MBoost}(m,l) \cdot \boldsymbol{\varphi}_{Re,sin}\left(f_{IndexSine}(i)\right)$$

and where

$\mathbf{S}_{IndexMapped}(m,l) = 0$, for $m < 0$ or $m \ge M$.

The above modifications are only done for the first 16 (calculated in increasing frequency order) sinusoids, for every time segment.

Furthermore, since a signal, according to the above, may be added to $\mathbf{Y}(k_x - 1, i)$, i.e. the lowband, or $\mathbf{Y}(k_x + M, i)$, i.e. one QMF subband above the SBR range, the following equation in subclause 4.6.18.5 needs to be modified:

$$\mathbf{X}(k,l) = \begin{cases} \mathbf{X}_{Low}\left(k, l+t_{HFAdj}\right) & ,0 \le k < k_x' + bsco', 0 \le l < l_{Temp} \\ \mathbf{X}_{Low}\left(k, l+t_{HFAdj}\right) & ,0 \le k < k_x + bsco, l_{Temp} \le l < numTimeSlots \cdot RATE \\ \mathbf{Y}\left(k, l+t_{HFAdj}\right) & ,k_x' + bsco' \le k < k_x + M, 0 \le l < l_{Temp} \\ \mathbf{Y}\left(k, l+t_{HFAdj}\right) & ,k_x + bsco \le k < k_x + M, l_{Temp} \le l < numTimeSlots \cdot RATE \\ 0 & ,\max\left(k_x + bsco, k_x + M\right) + M \le k < 64, 0 \le l < numTimeSlots \cdot RATE \end{cases}$$

The above is replaced by

$$\mathbf{X}(k,l) = \begin{cases} \mathbf{X}_{Low}\left(k, l + t_{HFAdj}\right) & , 0 \le k < k_x\text{'} - 1 + bsco\text{'}, 0 \le l < l_{Temp} \\ \mathbf{X}_{Low}\left(k, l + t_{HFAdj}\right) + \mathbf{Y}\left(k, l + t_{HFAdj}\right) & , k = k_x\text{'} - 1 + bsco\text{'}, 0 \le l < l_{Temp} \\ \mathbf{X}_{Low}\left(k, l + t_{HFAdj}\right) & , 0 \le k < k_x - 1 + bsco,\ l_{Temp} \le l < numTimeSlots \cdot RATE \\ \mathbf{X}_{Low}\left(k, l + t_{HFAdj}\right) + \mathbf{Y}\left(k, l + t_{HFAdj}\right) & , k = k_x - 1 + bsco,\ l_{Temp} \le l < numTimeSlots \cdot RATE \\ \mathbf{Y}\left(k, l + t_{HFAdj}\right) & , k_x\text{'} + bsco\text{'} \le k \le \min\left(k_x + M, 63\right), 0 \le l < l_{Temp} \\ \mathbf{Y}\left(k, l + t_{HFAdj}\right) & , \begin{cases} k_x + bsco \le k \le \min\left(k_x + M, 63\right) \\ l_{Temp} \le l < numTimeSlots \cdot RATE \end{cases} \\ 0 & , \begin{cases} \max\left(k_x + bsco, k_x + M\right) < k < 64 \\ 0 \le l < numTimeSlots \cdot RATE \end{cases} \end{cases}$$

”

In Part 3: Audio, Subpart 4, subclause Annex A (Normative) Normative Tables, add the following tables:

“

4.A.6 Tables for SBR

4.A.6.1 SBR Huffman Tables

The function *sbr_huff_dec()* is used as:

data = sbr_huff_dec(t_huff, codeword),

where *t_huff* is the selected Huffman table and *codeword* is the word read from the bitstream. The returned value *data,* is a Huffman table index corresponding to a specific code word, with the largest absolute value (LAV) of the table subtracted.

Huffman table overview:

Table 4.A.76

table name	df_env_flag	df_noise_flag	amp_res	LAV	Notes
t_huffman_env_1_5dB	0	dc	0	60	Note 1
f_huffman_env_1_5dB	1	dc	0	60	
t_huffman_env_bal_1_5dB	0	dc	0	24	
f_huffman_env_bal_1_5dB	1	dc	0	24	
t_huffman_env_3_0dB	0	dc	1	31	
f_huffman_env_3_0dB	1	dc	1	31	
t_huffman_env_bal_3_0dB	0	dc	1	12	
f_huffman_env_bal_3_0dB	1	dc	1	12	
t_huffman_noise_3_0dB	dc	0	dc	31	
f_huffman_noise_3_0dB	dc	1	dc	31	Note 2
t_huffman_noise_bal_3_0dB	dc	0	dc	12	
f_huffman_noise_bal_3_0dB	dc	1	dc	12	Note 2
Note 1: dc (don't care), indicates that the variable is not relevant. Note 2: The Huffman tables of f_huffman_noise_3_0dB and f_huffman_noise_bal_3_0dB are the same as for f_huffman_env_3_0dB and f_huffman_env_bal_3_0dB, respectively.					

Table 4.A.77 – t_huffman_env_1_5dB

index	Length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)
0	0x00000012	0x0003FFD6	61	0x00000003	0x00000004
1	0x00000012	0x0003FFD7	62	0x00000004	0x0000000C
2	0x00000012	0x0003FFD8	63	0x00000005	0x0000001C
3	0x00000012	0x0003FFD9	64	0x00000006	0x0000003C
4	0x00000012	0x0003FFDA	65	0x00000007	0x0000007C
5	0x00000012	0x0003FFDB	66	0x00000008	0x000000FC
6	0x00000013	0x0007FFB8	67	0x00000009	0x000001FC
7	0x00000013	0x0007FFB9	68	0x0000000A	0x000003FD
8	0x00000013	0x0007FFBA	69	0x0000000C	0x00000FFA
9	0x00000013	0x0007FFBB	70	0x0000000D	0x00001FF8

10	0x00000013	0x0007FFBC	71	0x0000000E	0x00003FF6
11	0x00000013	0x0007FFBD	72	0x0000000E	0x00003FF8
12	0x00000013	0x0007FFBE	73	0x0000000F	0x00007FF5
13	0x00000013	0x0007FFBF	74	0x00000010	0x0000FFEF
14	0x00000013	0x0007FFC0	75	0x00000011	0x0001FFE8
15	0x00000013	0x0007FFC1	76	0x00000010	0x0000FFF2
16	0x00000013	0x0007FFC2	77	0x00000013	0x0007FFD4
17	0x00000013	0x0007FFC3	78	0x00000013	0x0007FFD5
18	0x00000013	0x0007FFC4	79	0x00000013	0x0007FFD6
19	0x00000013	0x0007FFC5	80	0x00000013	0x0007FFD7
20	0x00000013	0x0007FFC6	81	0x00000013	0x0007FFD8
21	0x00000013	0x0007FFC7	82	0x00000013	0x0007FFD9
22	0x00000013	0x0007FFC8	83	0x00000013	0x0007FFDA
23	0x00000013	0x0007FFC9	84	0x00000013	0x0007FFDB
24	0x00000013	0x0007FFCA	85	0x00000013	0x0007FFDC
25	0x00000013	0x0007FFCB	86	0x00000013	0x0007FFDD
26	0x00000013	0x0007FFCC	87	0x00000013	0x0007FFDE
27	0x00000013	0x0007FFCD	88	0x00000013	0x0007FFDF
28	0x00000013	0x0007FFCE	89	0x00000013	0x0007FFE0
29	0x00000013	0x0007FFCF	90	0x00000013	0x0007FFE1
30	0x00000013	0x0007FFD0	91	0x00000013	0x0007FFE2
31	0x00000013	0x0007FFD1	92	0x00000013	0x0007FFE3
32	0x00000013	0x0007FFD2	93	0x00000013	0x0007FFE4
33	0x00000013	0x0007FFD3	94	0x00000013	0x0007FFE5
34	0x00000011	0x0001FFE6	95	0x00000013	0x0007FFE6
35	0x00000012	0x0003FFD4	96	0x00000013	0x0007FFE7
36	0x00000010	0x0000FFF0	97	0x00000013	0x0007FFE8
37	0x00000011	0x0001FFE9	98	0x00000013	0x0007FFE9
38	0x00000012	0x0003FFD5	99	0x00000013	0x0007FFEA
39	0x00000011	0x0001FFE7	100	0x00000013	0x0007FFEB
40	0x00000010	0x0000FFF1	101	0x00000013	0x0007FFEC
41	0x00000010	0x0000FFEC	102	0x00000013	0x0007FFED
42	0x00000010	0x0000FFED	103	0x00000013	0x0007FFEE
43	0x00000010	0x0000FFEE	104	0x00000013	0x0007FFEF
44	0x0000000F	0x00007FF4	105	0x00000013	0x0007FFF0
45	0x0000000E	0x00003FF9	106	0x00000013	0x0007FFF1
46	0x0000000E	0x00003FF7	107	0x00000013	0x0007FFF2
47	0x0000000D	0x00001FFA	108	0x00000013	0x0007FFF3
48	0x0000000D	0x00001FF9	109	0x00000013	0x0007FFF4
49	0x0000000C	0x00000FFB	110	0x00000013	0x0007FFF5
50	0x0000000B	0x000007FC	111	0x00000013	0x0007FFF6
51	0x0000000A	0x000003FC	112	0x00000013	0x0007FFF7
52	0x00000009	0x000001FD	113	0x00000013	0x0007FFF8
53	0x00000008	0x000000FD	114	0x00000013	0x0007FFF9
54	0x00000007	0x0000007D	115	0x00000013	0x0007FFFA
55	0x00000006	0x0000003D	116	0x00000013	0x0007FFFB
56	0x00000005	0x0000001D	117	0x00000013	0x0007FFFC
57	0x00000004	0x0000000D	118	0x00000013	0x0007FFFD
58	0x00000003	0x00000005	119	0x00000013	0x0007FFFE
59	0x00000002	0x00000001	120	0x00000013	0x0007FFFF
60	0x00000002	0x00000000			

Table 4.A.78 – f_huffman_env_1_5dB

index	length (hexadecimal)	codeword (hexadecimal)	index	length (hexadecimal)	codeword (hexadecimal)
0	0x00000013	0x0007FFE7	61	0x00000003	0x00000004
1	0x00000013	0x0007FFE8	62	0x00000004	0x0000000D
2	0x00000014	0x000FFFD2	63	0x00000005	0x0000001D
3	0x00000014	0x000FFFD3	64	0x00000006	0x0000003D
4	0x00000014	0x000FFFD4	65	0x00000008	0x000000FA
5	0x00000014	0x000FFFD5	66	0x00000008	0x000000FC
6	0x00000014	0x000FFFD6	67	0x00000009	0x000001FB
7	0x00000014	0x000FFFD7	68	0x0000000A	0x000003FA
8	0x00000014	0x000FFFD8	69	0x0000000B	0x000007F8
9	0x00000013	0x0007FFDA	70	0x0000000B	0x000007FA
10	0x00000014	0x000FFFD9	71	0x0000000B	0x000007FB
11	0x00000014	0x000FFFDA	72	0x0000000C	0x00000FF9
12	0x00000014	0x000FFFDB	73	0x0000000C	0x00000FFB
13	0x00000014	0x000FFFDC	74	0x0000000D	0x00001FF8
14	0x00000013	0x0007FFDB	75	0x0000000D	0x00001FFB
15	0x00000014	0x000FFFDD	76	0x0000000E	0x00003FF8
16	0x00000013	0x0007FFDC	77	0x0000000E	0x00003FF9
17	0x00000013	0x0007FFDD	78	0x00000010	0x0000FFF1
18	0x00000014	0x000FFFDE	79	0x00000010	0x0000FFF2
19	0x00000012	0x0003FFE4	80	0x00000011	0x0001FFEA
20	0x00000014	0x000FFFDF	81	0x00000011	0x0001FFEB
21	0x00000014	0x000FFFE0	82	0x00000012	0x0003FFE1
22	0x00000014	0x000FFFE1	83	0x00000012	0x0003FFE2
23	0x00000013	0x0007FFDE	84	0x00000012	0x0003FFEA
24	0x00000014	0x000FFFE2	85	0x00000012	0x0003FFE3
25	0x00000014	0x000FFFE3	86	0x00000012	0x0003FFE6
26	0x00000014	0x000FFFE4	87	0x00000012	0x0003FFE7
27	0x00000013	0x0007FFDF	88	0x00000012	0x0003FFEB
28	0x00000014	0x000FFFE5	89	0x00000014	0x000FFFE6
29	0x00000013	0x0007FFE0	90	0x00000013	0x0007FFE2
30	0x00000012	0x0003FFE8	91	0x00000014	0x000FFFE7
31	0x00000013	0x0007FFE1	92	0x00000014	0x000FFFE8
32	0x00000012	0x0003FFE0	93	0x00000014	0x000FFFE9
33	0x00000012	0x0003FFE9	94	0x00000014	0x000FFFEA
34	0x00000011	0x0001FFEF	95	0x00000014	0x000FFFEB
35	0x00000012	0x0003FFE5	96	0x00000014	0x000FFFEC
36	0x00000011	0x0001FFEC	97	0x00000013	0x0007FFE3
37	0x00000011	0x0001FFED	98	0x00000014	0x000FFFED
38	0x00000011	0x0001FFEE	99	0x00000014	0x000FFFEE
39	0x00000010	0x0000FFF4	100	0x00000014	0x000FFFEF
40	0x00000010	0x0000FFF3	101	0x00000014	0x000FFFF0
41	0x00000010	0x0000FFF0	102	0x00000013	0x0007FFE4
42	0x0000000F	0x00007FF7	103	0x00000014	0x000FFFF1
43	0x0000000F	0x00007FF6	104	0x00000012	0x0003FFEC
44	0x0000000E	0x00003FFA	105	0x00000014	0x000FFFF2
45	0x0000000D	0x00001FFA	106	0x00000014	0x000FFFF3
46	0x0000000D	0x00001FF9	107	0x00000013	0x0007FFE5
47	0x0000000C	0x00000FFA	108	0x00000013	0x0007FFE6
48	0x0000000C	0x00000FF8	109	0x00000014	0x000FFFF4
49	0x0000000B	0x000007F9	110	0x00000014	0x000FFFF5