|  |  |
| --- | --- |
| 3GPP TR 26.858 V0.0.2 (2024-11) | |
| Technical Report | |
| 3rd Generation Partnership Project;  Technical Specification Group SA4;  Study on APIs for 3GPP Speech and Audio Codecs  (Release 19) | |
|  | |
|  |  |
| The present document has been developed within the 3rd Generation Partnership Project (3GPP TM) and may be further elaborated for the purposes of 3GPP. The present document has not been subject to any approval process by the 3GPPOrganizational Partners and shall not be implemented. This Specification is provided for future development work within 3GPPonly. The Organizational Partners accept no liability for any use of this Specification. Specifications and Reports for implementation of the 3GPP TM system should be obtained via the 3GPP Organizational Partners' Publications Offices. | |

|  |
| --- |
|  |
| ***3GPP***  Postal address  3GPP support office address  650 Route des Lucioles - Sophia Antipolis  Valbonne - FRANCE  Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16  Internet  http://www.3gpp.org |
| ***Copyright Notification***  No part may be reproduced except as authorized by written permission. The copyright and the foregoing restriction extend to reproduction in all media.  © 2023, 3GPP Organizational Partners (ARIB, ATIS, CCSA, ETSI, TSDSI, TTA, TTC).  All rights reserved.  UMTS™ is a Trade Mark of ETSI registered for the benefit of its members  3GPP™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners LTE™ is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners  GSM® and the GSM logo are registered and owned by the GSM Association |

Contents

Foreword 5

Introduction 6

1 Scope 6

2 References 6

3 Definitions of terms, symbols and abbreviations 6

3.1 Terms 7

3.2 Symbols 7

3.3 Abbreviations 7

4 Interfaces of codecs and other processing blocks 7

5 Web Interfaces for codecs and other processing blocks 7

5.1 Introduction 7

5.2 WebCodecs API 7

Codec Registration Procedure 9

6 Common APIs 9

7 Recommendations for Potential Interfaces and Adapters 9

8 Recommendations for Potential Normative Work 10

Annex <X> (informative): Change history 11

Annex <X1> (informative): Relevant C APIs of 3GPP Speech and Audio Codecs 11

X1.1 Introduction 11

X1.2 AMR 11

X1.2.1 General 11

X1.2.2 AMR Fixed-Point Code (TS 26.073) 11

X1.2.2.1 General 11

X1.2.2.2 Encoder API (cod\_amr.h) 11

X1.2.2 3 Decoder (dec\_amr.h) 14

X1.2.3 AMR Floating-Point Code (TS 26.104) 17

X1.2.3.1 General 17

X1.2.3.2 Encoder (interf\_enc.h) 17

X1.2.3.3 Decoder (interf\_dec.h) 17

X1.3 AMR-WB 18

X1.3.2 AMR-WB Fixed-Point (TS 26.173) 18

X1.3.2.1 General 18

X1.3.2.2 Encoder (main.h) 18

X1.3.2.3 Decoder (main.h) 18

X1.3.3 AMR-WB Floating-Point (TS 26.204): 19

X1.3.3.1 General 19

X1.3.3.2 Encoder (enc.h) 19

X1.3.3.3 Decoder (dec.h) 19

X1.4 EVS 19

X1.4.1 General 19

X1.4.2 Example API in S4-211541 19

X1.4.1 Encoder 19

X1.4.2 Decoder 20

X1.5 eAAC+ 22

X1.5.1 eAAC+ Floating-Point (TS 26.410) 22

X1.5.1.1 AAC Encoder (aacenc.h) 22

X1.5.1.2 SBR Encoder (sbr\_main.h) 24

X1.5.1.3 Resampler (iir32resample.h) 26

X1.5.1.4 AAC Decoder (aacdecoder.h) 26

X1.5.1.5 SBR Decoder (sbrdecoder.h) 27

X1.5.2 eAAC+ Fixed-Point (TS 26.411) 29

X1.5.2.1 AAC Encoder (aacenc.h) 29

X1.5.2.2 SBR Encoder (sbr\_main.h) 30

X1.5.2.3 Resample (downsample\_FIR.h) 33

X1.5.2.4 AAC Decoder (aacdecoder.h) 35

X1.5.2.5 SBR Decoder (sbrdecoder.h) 36

X1.6 AMR-WB+ 37

X1.6.1 AMR-WB+ Fixed-Point (TS 26.273) 37

X1.6.1.1 Encoder (enc\_if\_fx.h) 37

X1.6.1.2 Decoder (dec\_if\_fx.h) 37

X1.6.2 AMR-WB+ Floating-Point (TS 26.304) 38

X1.6.2.1 Encoder (proto\_func.h) 38

X1.6.2.2 Decoder (proto\_func.h) 38

X1.7 IVAS 39

For definitive guidance on drafting 3GPP TSs and TRs, see [3GPP TS 21.801](http://www.3gpp.org/DynaReport/21801.htm) supplemented by the 3GPP web page <http://www.3gpp.org/specifications-groups/delegates-corner/writing-a-new-spec>.

Ensure all blue guidance text is removed before submitting the TS/TR to the TSG for approval.

# Foreword

This clause is mandatory; do not alter the text in any way other than to choose between "Specification" and "Report".

This Technical Report has been produced by the 3rd Generation Partnership Project (3GPP).

The contents of the present document are subject to continuing work within the TSG and may change following formal TSG approval. Should the TSG modify the contents of the present document, it will be re-released by the TSG with an identifying change of release date and an increase in version number as follows:

Version x.y.z

where:

x the first digit:

1 presented to TSG for information;

2 presented to TSG for approval;

3 or greater indicates TSG approved document under change control.

y the second digit is incremented for all changes of substance, i.e. technical enhancements, corrections, updates, etc.

z the third digit is incremented when editorial only changes have been incorporated in the document.

In drafting the TS/TR, pay particular attention to the use of modal auxiliary verbs! TRs shall not contain any normative provisions.

In the present document, modal verbs have the following meanings:

**shall** indicates a mandatory requirement to do something

**shall not** indicates an interdiction (prohibition) to do something

The constructions "shall" and "shall not" are confined to the context of normative provisions, and do not appear in Technical Reports.

The constructions "must" and "must not" are not used as substitutes for "shall" and "shall not". Their use is avoided insofar as possible, and they are not used in a normative context except in a direct citation from an external, referenced, non-3GPP document, or so as to maintain continuity of style when extending or modifying the provisions of such a referenced document.

**should** indicates a recommendation to do something

**should not** indicates a recommendation not to do something

**may** indicates permission to do something

**need not** indicates permission not to do something

The construction "may not" is ambiguous and is not used in normative elements. The unambiguous constructions "might not" or "shall not" are used instead, depending upon the meaning intended.

**can** indicates that something is possible

**cannot** indicates that something is impossible

The constructions "can" and "cannot" are not substitutes for "may" and "need not".

**will** indicates that something is certain or expected to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

**will not** indicates that something is certain or expected not to happen as a result of action taken by an agency the behaviour of which is outside the scope of the present document

**might** indicates a likelihood that something will happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

**might not** indicates a likelihood that something will not happen as a result of action taken by some agency the behaviour of which is outside the scope of the present document

In addition:

**is** (or any other verb in the indicative mood) indicates a statement of fact

**is not** (or any other negative verb in the indicative mood) indicates a statement of fact

The constructions "is" and "is not" do not indicate requirements.

# Introduction

This clause is optional. If it exists, it shall be the second unnumbered clause.

# 1 Scope

This clause shall start on a new page.

The present document …

# 2 References

The following documents contain provisions which, through reference in this text, constitute provisions of the present document.

- References are either specific (identified by date of publication, edition number, version number, etc.) or non‑specific.

- For a specific reference, subsequent revisions do not apply.

- For a non-specific reference, the latest version applies. In the case of a reference to a 3GPP document (including a GSM document), a non-specific reference implicitly refers to the latest version of that document *in the same Release as the present document*.

[1] 3GPP TR 21.905: "Vocabulary for 3GPP Specifications".

…

[x] <doctype> <#>[ ([up to and including]{yyyy[-mm]|V<a[.b[.c]]>}[onwards])]: "<Title>".

It is preferred that the reference to 21.905 be the first in the list.

# 3 Definitions of terms, symbols and abbreviations

This clause and its three subclauses are mandatory. The contents shall be shown as "void" if the TS/TR does not define any terms, symbols, or abbreviations.

## 3.1 Terms

For the purposes of the present document, the terms given in 3GPP TR 21.905 [1] and the following apply. A term defined in the present document takes precedence over the definition of the same term, if any, in 3GPP TR 21.905 [1].

Definition format (Normal)

**<defined term>:** <definition>.

**example:** text used to clarify abstract rules by applying them literally.

## 3.2 Symbols

For the purposes of the present document, the following symbols apply:

Symbol format (EW)

<symbol> <Explanation>

## 3.3 Abbreviations

For the purposes of the present document, the abbreviations given in 3GPP TR 21.905 [1] and the following apply. An abbreviation defined in the present document takes precedence over the definition of the same abbreviation, if any, in 3GPP TR 21.905 [1].

Abbreviation format (EW)

<ABBREVIATION> <Expansion>

# 4 Interfaces of codecs and other processing blocks

# 5 Web Interfaces for codecs and other processing blocks

[

## 5.1 Introduction

## 5.2 WebCodecs API

The WebCodecs API is a powerful web API that provides developers with low-level access to the individual frames of a video stream and chunks of audio. It is particularly useful for web applications that require full control over the way media is processed, such as video or audio editors, and video conferencing applciations.

The WebCodecs API provides access to codecs that are already in the browser, eliminating the need for additional software codecs and leveraging the existing hardware acceleration on the device. It gives access to raw video frames, chunks of audio data, image decoders, audio and video encoders, and decoders.

The WebCodecs API uses an asynchronous processing model. Each instance of an encoder or decoder maintains an internal, independent processing queue. Methods named configure(), encode(), decode(), and flush() operate asynchronously by appending control messages to the end of the queue, while methods named reset() and close() synchronously abort all pending work and purge the processing queue.

The WebCodecs API provides several interfaces:

- AudioDecoder: Decodes EncodedAudioChunk objects.

- VideoDecoder: Decodes EncodedVideoChunk objects.

- AudioEncoder: Encodes AudioData objects.

- VideoEncoder: Encodes VideoFrame objects.

- EncodedAudioChunk: Represents codec-specific encoded audio bytes.

- EncodedVideoChunk: Represents codec-specific encoded video bytes.

- AudioData: Represents unencoded audio data.

- VideoFrame: Represents a frame of unencoded video data.

- VideoColorSpace: Represents the color space of a video frame.

- ImageDecoder: Unpacks and decodes image data, giving access to the sequence of frames in an animated image.

- ImageTrackList: Represents the list of tracks available in the image.

- ImageTrack: Represents an individual image track.

The following table provides a simple example code for the usage of WebCodecs to demonstrate the functionality of the WebCodecs API:

|  |
| --- |
| // Create a new VideoDecoder and configure it  const init = {  output: handleFrame,  error: (e) => { console.log(e.message); },  };  const config = {  codec: "hevc",  codedWidth: 1280,  codedHeight: 720  };  let decoder = new VideoDecoder(init);  decoder.configure(config);  // Create a new VideoEncoder and configure it  let encoder = new VideoEncoder({  output: (chunk) => {  const buffer = new ArrayBuffer(chunk.byteLength);  chunk.copyTo(buffer);  chunks.push(buffer);  },  error: (e) => console.error(e.message)  });  encoder.configure({  codec: 'hevc',  width: 1280,  height: 720,  bitrate: 2000000,  framerate: 25  });  // Encode every image as a frame  track.requestFrame().then((frame) => {  encoder.encode(frame, {keyFrame: true});  frame.close();  });  // Create a video from it  encoder.flush().then(() => {  const blob = new Blob(chunks, {type: 'video/webm; codecs=vp8'});  const url = URL.createObjectURL(blob);  decoder.decode(new EncodedVideoChunk({  type: 'key',  timestamp: 0,  data: blob  }));  });  })  .catch((error) => {  console.error("Error: ", error);  }); |

A full example can be found under <https://bouazizi.dev/webcodecs/>

## 5.3 Codec Registration Procedure

The codec registration procedure for new codecs is defined by W3C in [2]. The registration request should define the EncodedAudioChunk or EncodedVideoChunk format as well as the configuration data format in AudioDecoderConfig or VideoDecoderConfig. These structures may be extended to carry codec-specific information.

The request must then be sent to the GitHub issue trucker of WebCodecs for evaluation.

## 5.4 Configuration Properties for 3GPP Speech and Audio Codecs

For 3GPP speech and audio codecs the configuration properties were extracted, based on the APIs in Annex <X1>, to allow full configuration for applicati0on as WebCodecs. The identified encoder and decoder configuration properties are listed in Tables 1 and 2, respectively.

Table 1: Encoder Configuration Properties

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Codec | Property | Description | Type | Default |
| AMR | bitRate | Bitrate in bits per second (e.g., 4750, 5150, 12200) | integer | 12200 |
| AMR | allowDtx | Enable Discontinuous Transmission (DTX) | boolean | false |
| AMR-WB | bitRate | Bitrate in bits per second (e.g., 6600, 8850, 23850) | integer | 23850 |
| AMR-WB | allowDtx | Enable Discontinuous Transmission (DTX) | boolean | false |
| AMR-WB+ | bitRate | Bitrate in bits per second (e.g., 8000, 16000, 24000) | integer | 24000 |
| AMR-WB+ | stereo | Enable stereo encoding | boolean | false |
| EVS | bitRate | Bitrate in bits per second (e.g., 9600, 13200, 24400) | integer | 13200 |
| EVS | bandwidth | Bandwidth mode (e.g., "NB", "WB", "SWB", "FB") | string | WB |
| EVS | dtx | DTX interval for adaptive SID (0 for adaptive intervals) | integer | 0 |
| EVS | partialCopyOffset | Offset for partial copies in channel-aware mode | integer | 3 |
| EVS | fecIndicator | FEC indicator for channel-aware mode (either "HI" or "LO") | string | HI |

Table 2: Decoder Configuration Elements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Codec | Property | Description | Type | Default |
| AMR | None | AMR decoder does not require additional config options | N/A | N/A |
| AMR-WB | None | AMR-WB decoder does not require additional config options | N/A | N/A |
| AMR-WB+ | stereo | Enable stereo decoding | boolean | false |
| EVS | outputSampleRate | Output sample rate (8000, 16000, 32000, 48000) | integer | 16000 |

]

# 6 Common APIs

# 7 Recommendations for Potential Interfaces and Adapters

# 8 Recommendations for Potential Normative Work

Annex <X> (informative):  
Change history

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Change history** | | | | | | | |
| **Date** | **Meeting** | **TDoc** | **CR** | **Rev** | **Cat** | **Subject/Comment** | **New version** |
|  |  |  |  |  |  |  |  |

Annex <X1> (informative):  
Relevant C APIs of 3GPP Speech and Audio Codecs

## X1.1 Introduction

The following clauses collect the interfaces to the 3GPP coding schemes.

## X1.2 AMR

### X1.2.1 General

### X1.2.2 AMR Fixed-Point Code (TS 26.073)

#### X1.2.2.1 General

#### X1.2.2.2 Encoder API (cod\_amr.h)

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DEFINITION OF DATA TYPES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\*-----------------------------------------------------------\*

\* Coder constant parameters (defined in "cnst.h") \*

\*-----------------------------------------------------------\*

\* L\_WINDOW : LPC analysis window size. \*

\* L\_NEXT : Samples of next frame needed for autocor. \*

\* L\_FRAME : Frame size. \*

\* L\_FRAME\_BY2 : Half the frame size. \*

\* L\_SUBFR : Sub-frame size. \*

\* M : LPC order. \*

\* MP1 : LPC order+1 \*

\* L\_TOTAL7k4 : Total size of speech buffer. \*

\* PIT\_MIN7k4 : Minimum pitch lag. \*

\* PIT\_MAX : Maximum pitch lag. \*

\* L\_INTERPOL : Length of filter for interpolation \*

\*-----------------------------------------------------------\*/

typedef struct {

/\* Speech vector \*/

Word16 old\_speech[L\_TOTAL];

Word16 \*speech, \*p\_window, \*p\_window\_12k2;

Word16 \*new\_speech; /\* Global variable \*/

/\* Weight speech vector \*/

Word16 old\_wsp[L\_FRAME + PIT\_MAX];

Word16 \*wsp;

/\* OL LTP states \*/

Word16 old\_lags[5];

Word16 ol\_gain\_flg[2];

/\* Excitation vector \*/

Word16 old\_exc[L\_FRAME + PIT\_MAX + L\_INTERPOL];

Word16 \*exc;

/\* Zero vector \*/

Word16 ai\_zero[L\_SUBFR + MP1];

Word16 \*zero;

/\* Impulse response vector \*/

Word16 \*h1;

Word16 hvec[L\_SUBFR \* 2];

/\* Substates \*/

lpcState \*lpcSt;

lspState \*lspSt;

clLtpState \*clLtpSt;

gainQuantState \*gainQuantSt;

pitchOLWghtState \*pitchOLWghtSt;

tonStabState \*tonStabSt;

vadState \*vadSt;

Flag dtx;

dtx\_encState \*dtx\_encSt;

/\* Filter's memory \*/

Word16 mem\_syn[M], mem\_w0[M], mem\_w[M];

Word16 mem\_err[M + L\_SUBFR], \*error;

Word16 sharp;

} cod\_amrState;

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DECLARATION OF PROTOTYPES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Function : cod\_amr\_init

\* Purpose : Allocates memory and initializes state variables

\* Description : Stores pointer to filter status struct in \*st. This

\* pointer has to be passed to cod\_amr in each call.

\* - initilize pointers to speech buffer

\* - initialize static pointers

\* - set static vectors to zero

\* Returns : 0 on success

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

int cod\_amr\_init (cod\_amrState \*\*st, Flag dtx);

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Function : cod\_amr\_reset

\* Purpose : Resets state memory

\* Returns : 0 on success

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

int cod\_amr\_reset (cod\_amrState \*st);

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Function : cod\_amr\_exit

\* Purpose : The memory used for state memory is freed

\* Description : Stores NULL in \*st

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void cod\_amr\_exit (cod\_amrState \*\*st);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION: cod\_amr\_first

\*

\* PURPOSE: Copes with look-ahead.

\*

\* INPUTS:

\* No input argument are passed to this function. However, before

\* calling this function, 40 new speech data should be copied to the

\* vector new\_speech[]. This is a global pointer which is declared in

\* this file (it points to the end of speech buffer minus 200).

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int cod\_amr\_first(cod\_amrState \*st, /\* i/o : State struct \*/

Word16 new\_speech[] /\* i : speech input (L\_FRAME) \*/

);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* FUNCTION: cod\_amr

\*

\* PURPOSE: Main encoder routine.

\*

\* DESCRIPTION: This function is called every 20 ms speech frame,

\* operating on the newly read 160 speech samples. It performs the

\* principle encoding functions to produce the set of encoded parameters

\* which include the LSP, adaptive codebook, and fixed codebook

\* quantization indices (addresses and gains).

\*

\* INPUTS:

\* No input argument are passed to this function. However, before

\* calling this function, 160 new speech data should be copied to the

\* vector new\_speech[]. This is a global pointer which is declared in

\* this file (it points to the end of speech buffer minus 160).

\*

\* OUTPUTS:

\*

\* ana[]: vector of analysis parameters.

\* synth[]: Local synthesis speech (for debugging purposes)

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int cod\_amr(cod\_amrState \*st, /\* i/o : State struct \*/

enum Mode mode, /\* i : AMR mode \*/

Word16 new\_speech[], /\* i : speech input (L\_FRAME) \*/

Word16 ana[], /\* o : Analysis parameters \*/

enum Mode \*usedMode, /\* o : used mode \*/

Word16 synth[] /\* o : Local synthesis \*/

);

#### X1.2.2 3 Decoder (dec\_amr.h)

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* LOCAL VARIABLES AND TABLES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\*---------------------------------------------------------------\*

\* Postfilter constant parameters (defined in "cnst.h") \*

\*---------------------------------------------------------------\*

\* L\_FRAME : Frame size. \*

\* PIT\_MAX : Maximum pitch lag. \*

\* L\_INTERPOL : Length of filter for interpolation. \*

\* M : LPC order. \*

\*---------------------------------------------------------------\*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DEFINITION OF DATA TYPES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

typedef struct{

/\* Excitation vector \*/

Word16 old\_exc[L\_SUBFR + PIT\_MAX + L\_INTERPOL];

Word16 \*exc;

/\* Lsp (Line spectral pairs) \*/

/\* Word16 lsp[M]; \*/ /\* Used by CN codec \*/

Word16 lsp\_old[M];

/\* Filter's memory \*/

Word16 mem\_syn[M];

/\* pitch sharpening \*/

Word16 sharp;

Word16 old\_T0;

/\* Memories for bad frame handling \*/

Word16 prev\_bf;

Word16 prev\_pdf;

Word16 state;

Word16 excEnergyHist[9];

/\* Variable holding received ltpLag, used in background noise and BFI \*/

Word16 T0\_lagBuff;

/\* Variables for the source characteristic detector (SCD) \*/

Word16 inBackgroundNoise;

Word16 voicedHangover;

Word16 ltpGainHistory[9];

Bgn\_scdState\* background\_state;

Word16 nodataSeed;

Cb\_gain\_averageState \*Cb\_gain\_averState;

lsp\_avgState \*lsp\_avg\_st;

D\_plsfState\* lsfState;

ec\_gain\_pitchState\* ec\_gain\_p\_st;

ec\_gain\_codeState\* ec\_gain\_c\_st;

gc\_predState\* pred\_state;

ph\_dispState\* ph\_disp\_st;

dtx\_decState\* dtxDecoderState;

} Decoder\_amrState;

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* DECLARATION OF PROTOTYPES

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Function : Decoder\_amr\_init

\* Purpose : Allocates initializes state memory

\* Description : Stores pointer to filter status struct in \*st. This

\* pointer has to be passed to Decoder\_amr in each call.

\* Returns : 0 on success

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

int Decoder\_amr\_init (Decoder\_amrState \*\*st);

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Function : Decoder\_amr\_reset

\* Purpose : Resets state memory

\* Returns : 0 on success

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

int Decoder\_amr\_reset (Decoder\_amrState \*st,enum Mode mode);

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Function : Decoder\_amr\_exit

\* Purpose : The memory used for state memory is freed

\* Description : Stores NULL in \*s

\* Returns : void

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

void Decoder\_amr\_exit (Decoder\_amrState \*\*st);

/\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*

\* Function : Decoder\_amr

\* Purpose : Speech decoder routine.

\* Returns : 0

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

int Decoder\_amr (

Decoder\_amrState \*st, /\* i/o : State variables \*/

enum Mode mode, /\* i : AMR mode \*/

Word16 parm[], /\* i : vector of synthesis parameters

(PRM\_SIZE) \*/

enum RXFrameType frame\_type, /\* i : received frame type \*/

Word16 synth[], /\* o : synthesis speech (L\_FRAME) \*/

Word16 A\_t[] /\* o : decoded LP filter in 4 subframes

(AZ\_SIZE) \*/

);

### X1.2.3 AMR Floating-Point Code (TS 26.104)

#### X1.2.3.1 General

#### X1.2.3.2 Encoder (interf\_enc.h)

/\*

\* Encodes one frame of speech

\* Returns packed octets

\*/

int Encoder\_Interface\_Encode( void \*st, enum Mode mode, short \*speech,

#ifndef ETSI

unsigned char \*serial, /\* max size 31 bytes \*/

#else

short \*serial, /\* size 500 bytes \*/

#endif

int forceSpeech ); /\* use speech mode \*/

/\*

\* Reserve and init. memory

\*/

void \*Encoder\_Interface\_init( int dtx );

/\*

\* Exit and free memory

\*/

void Encoder\_Interface\_exit( void \*state );

#endif

#### X1.2.3.3 Decoder (interf\_dec.h)

/\*

\* Conversion from packed bitstream to endoded parameters

\* Decoding parameters to speech

\*/

void Decoder\_Interface\_Decode( void \*st,

#ifndef ETSI

unsigned char \*bits,

#else

short \*bits,

#endif

short \*synth, int bfi );

/\*

\* Reserve and init. memory

\*/

void \*Decoder\_Interface\_init( void );

/\*

\* Exit and free memory

\*/

void Decoder\_Interface\_exit( void \*state );

## X1.3 AMR-WB

### X1.3.2 AMR-WB Fixed-Point (TS 26.173)

#### X1.3.2.1 General

#### X1.3.2.2 Encoder (main.h)

void Init\_coder(void \*\*spe\_state);

void Close\_coder(void \*spe\_state);

void coder(

Word16 \* mode, /\* input : used mode \*/

Word16 speech16k[], /\* input : 320 new speech samples (at 16 kHz) \*/

Word16 prms[], /\* output: output parameters \*/

Word16 \* ser\_size, /\* output: bit rate of the used mode \*/

void \*spe\_state, /\* i/o : State structure \*/

Word16 allow\_dtx /\* input : DTX ON/OFF \*/

);

void Reset\_encoder(void \*st, Word16 reset\_all);

Word16 encoder\_homing\_frame\_test(Word16 input\_frame[]);

#### X1.3.2.3 Decoder (main.h)

void Init\_decoder(void \*\*spd\_state);

void Close\_decoder(void \*spd\_state);

void decoder(

Word16 mode, /\* input : used mode \*/

Word16 prms[], /\* input : parameter vector \*/

Word16 synth16k[], /\* output: synthesis speech \*/

Word16 \* frame\_length, /\* output: lenght of the frame \*/

void \*spd\_state, /\* i/o : State structure \*/

Word16 frame\_type /\* input : received frame type \*/

);

void Reset\_decoder(void \*st, Word16 reset\_all);

Word16 decoder\_homing\_frame\_test(Word16 input\_frame[], Word16 mode);

Word16 decoder\_homing\_frame\_test\_first(Word16 input\_frame[], Word16 mode);

### X1.3.3 AMR-WB Floating-Point (TS 26.204):

#### X1.3.3.1 General

#### X1.3.3.2 Encoder (enc.h)

Word16 E\_MAIN\_init(void \*\*spe\_state);

void E\_MAIN\_reset(void \*st, Word16 reset\_all);

Word16 E\_MAIN\_encode(Word16 \* mode, Word16 input\_sp[], Word16 prms[],

void \*spe\_state, Word16 allow\_dtx);

void E\_MAIN\_close(void \*\*spe\_state);

#### X1.3.3.3 Decoder (dec.h)

void D\_MAIN\_reset(void \*st, Word16 reset\_all);

Word32 D\_MAIN\_init(void \*\*spd\_state);

void D\_MAIN\_close(void \*\*spd\_state);

Word32 D\_MAIN\_decode(Word16 mode, Word16 prms[], Word16 synth16k[],

void \*spd\_state, UWord8 frame\_type);

## X1.4 EVS

### X1.4.1 General

### X1.4.2 Example API in S4-211541

#### X1.4.1 Encoder

//! Creates the EVS encoder state.

DLL\_PUBLIC

Encoder\_State\* EVS\_cod\_open();

//! Configures the EVS encoder - needs to be called after EVS\_cod\_open().

/\*! @param[in] st encoder state

\* @param[in] inSampleRate sample rate of the audio samples to encode: 8000, 16000, 32000, 48000

\* @param[in] bandwidth audio bandwith to encode: NB=8000, WB=16000, SWB=32000, FB=48000

\* @param[in] bitrate codec bitrate in bits per second, e.g. 9600, 13200, 24400

\* @param[in] dtx flag to enable DTX: 0=DTX off, 1=DTX on

\* @param[in] partialCopyOffset offset of partial copies in case of channel aware mode (bitrate=13200): 0 (no CA), 2, 3 (default), 5, 7

\* @return 0 if successful

\*/

DLL\_PUBLIC

int EVS\_cod\_configure(Encoder\_State \*st, uint32\_t inSampleRate, uint32\_t bandwidth, uint32\_t bitrate, int dtx, uint32\_t partialCopyOffset);

//! Switches the codec mode - can be called between encoding two frames.

/\*! @param[in] st encoder state

\* @param[in] bandwidth audio bandwith to encode: NB=8000, WB=16000, SWB=32000, FB=48000

\* @param[in] bitrate codec bitrate in bits per second, e.g. 9600, 13200, 24400

\* @param[in] dtx flag to enable DTX: 0=DTX off, 1=DTX on

\* @param[in] partialCopyOffset offset of partial copies in case of channel aware mode (bitrate=13200): 0 (no CA), 2, 3 (default), 5, 7

\* @return 0 if successful

\*/

DLL\_PUBLIC

int EVS\_cod\_switchMode(Encoder\_State \*st, uint32\_t bandwidth, uint32\_t bitrate, int dtx, uint32\_t partialCopyOffset);

//! Encodes one frame of audio samples.

/\*! @param[in] st encoder state

\* @param[in] samples input signal to encode

\* @param[in] nSamples number of input samples - must equal 20ms

\* @param[out] bitstream buffer to store the bitstream

\* @param[out] nBitstreamBytes number of bytes written to the bitstream buffer

\* @param[out] isSid flag if the current frame was encoded as SID

\* @return 0 if successful

\*/

DLL\_PUBLIC

int EVS\_cod\_encode(Encoder\_State \*st, const int16\_t \*samples, uint32\_t nSamples, uint8\_t \*bitstream, uint32\_t \*nBitstreamBytes, int \*isSid);

//! Returns the encoder delay and its time scale.

/\*! @param[in] st encoder state

\* @param[out] nSamples delay in samples

\* @param[out] timeScale time scale of nSamples

\* @return 0 if successful \*/

DLL\_PUBLIC

int EVS\_cod\_delay(Encoder\_State \*st, uint32\_t \*nSamples, uint32\_t \*timeScale);

//! Destructs the EVS encoder state and frees the memory.

DLL\_PUBLIC

void EVS\_cod\_close(Encoder\_State \*st);

//! Returns a string containing the version number of the EVS codec (e.g. 12.1.0).

DLL\_PUBLIC

const char \* EVS\_cod\_version();

#### X1.4.2 Decoder

struct Decoder\_State;

typedef struct Decoder\_State Decoder\_State;

typedef enum

{

EVS\_BITSTREAM\_FORMAT\_NON\_VOIP,

EVS\_BITSTREAM\_FORMAT\_VOIP\_G192\_RTP,

EVS\_BITSTREAM\_FORMAT\_VOIP\_RTPDUMP

} BitstreamFormat;

//! Creates the EVS decoder state.

DLL\_PUBLIC

Decoder\_State\* EVS\_dec\_open();

//! Configures the EVS decoder - needs to be called after EVS\_dec\_open().

/\*! @param[in] st decoder state

\* @param[in] bitstreamFormat bitstream format (G.192/MIME/VOIP\_G192\_RTP/VOIP\_RTPDUMP)

\* @param[in] outSampleRate sample rate of the audio samples to create in Hz: 8000, 16000, 32000, 48000

\* @param[out] frameSize the number of samples created in one call to EVS\_dec\_decode() or EVS\_dec\_conceal()

\* @return 0 if successful

\*/

DLL\_PUBLIC

int EVS\_dec\_configure(Decoder\_State \*st, BitstreamFormat bitstreamFormat, uint32\_t outSampleRate, uint32\_t \*frameSize);

//! Decodes one bitstream frame.

/\*! @param[in] st decoder state

\* @param[in] bitstream buffer containing the bitstream

\* @param[in] partialCopy flag if the partial copy contained in the bitstream should be decoded

\* @param[out] nBits number of bits in the bitstream buffer

\* @param[out] samples one frame of decoded signal

\* @param[out] bandwidth for information: outputs the current bandwidth of the bitstream

\* @return 0 if successful

\*/

DLL\_PUBLIC

int EVS\_dec\_decode(Decoder\_State \*st, const uint8\_t \*bitstream, uint32\_t nBits, int partialCopy,

int16\_t \*samples, uint32\_t \*bandwidth);

//! Creates audio samples for one missing frame (PLC or DTX).

/\*! @param[in] st decoder state

\* @param[out] samples one frame of decoded signal

\* @return 0 if successful

\*/

DLL\_PUBLIC

int EVS\_dec\_conceal(Decoder\_State \*st, int16\_t \*samples);

//! Returns the decoder delay and its time scale.

/\*! @param[in] st decoder state

\* @param[out] nSamples delay in samples

\* @param[out] timeScale time scale of nSamples

\* @return 0 if successful \*/

DLL\_PUBLIC

int EVS\_dec\_delay(Decoder\_State \*st, uint32\_t \*nSamples, uint32\_t \*timeScale);

//! Destructs the EVS decoder state and frees the memory.

DLL\_PUBLIC

void EVS\_dec\_close(Decoder\_State \*st);

//! Checks if a frame contains a partial copy and gets its offset.

DLL\_PUBLIC

int EVS\_dec\_previewFrame(const uint8\_t \*bitstream, uint32\_t nBitstreamBytes, int \*rfNoData, uint32\_t \*partialCopyOffset);

//! Returns a string containing the version number of the EVS codec (e.g. 12.1.0).

DLL\_PUBLIC

const char \* EVS\_dec\_version();

## X1.5 eAAC+

### X1.5.1 eAAC+ Floating-Point (TS 26.410)

#### X1.5.1.1 AAC Encoder (aacenc.h)

/\* here we distinguish between stereo and the mono only encoder \*/

#ifdef MONO\_ONLY

#define MAX\_CHANNELS 1

#else

#define MAX\_CHANNELS 2

#endif

#define AACENC\_BLOCKSIZE 1024 /\*! encoder only takes BLOCKSIZE samples at a time \*/

#define AACENC\_TRANS\_FAC 8 /\*! encoder short long ratio \*/

#define AACENC\_PCM\_LEVEL 1.0 /\*! encoder pcm 0db refernence \*/

/\*-------------------------- defines --------------------------------------\*/

#define BUFFERSIZE 1024 /\* anc data \*/

/\*-------------------- structure definitions ------------------------------\*/

typedef struct {

int sampleRate; /\* audio file sample rate \*/

int bitRate; /\* encoder bit rate in bits/sec \*/

int nChannelsIn; /\* number of channels on input (1,2) \*/

int nChannelsOut; /\* number of channels on output (1,2) \*/

int bandWidth; /\* core coder audio bandwidth in Hz \*/

} AACENC\_CONFIG;

struct AAC\_ENCODER;

/\*

\* p u b l i c a n c i l l a r y

\*

\*/

/\*-----------------------------------------------------------------------------

functionname: AacInitDefaultConfig

description: gives reasonable default configuration

returns: ---

------------------------------------------------------------------------------\*/

void AacInitDefaultConfig(AACENC\_CONFIG \*config);

/\*---------------------------------------------------------------------------

functionname:AacEncOpen

description: allocate and initialize a new encoder instance

returns: AACENC\_OK if success

---------------------------------------------------------------------------\*/

int AacEncOpen

( struct AAC\_ENCODER\*\* phAacEnc, /\* pointer to an encoder handle, initialized on return \*/

const AACENC\_CONFIG config /\* pre-initialized config struct \*/

);

int AacEncEncode(struct AAC\_ENCODER \*hAacEnc,

float \*timeSignal,

unsigned int timeInStride,

const unsigned char \*ancBytes, /\*!< pointer to ancillary data bytes \*/

unsigned int \*numAncBytes, /\*!< number of ancillary Data Bytes, send as fill element \*/

unsigned int \*outBytes, /\*!< pointer to output buffer \*/

int \*numOutBytes /\*!< number of bytes in output buffer \*/

);

/\*---------------------------------------------------------------------------

functionname:AacEncClose

description: deallocate an encoder instance

---------------------------------------------------------------------------\*/

void AacEncClose (struct AAC\_ENCODER\* hAacEnc); /\* an encoder handle \*/

#### X1.5.1.2 SBR Encoder (sbr\_main.h)

#define MAX\_TRANS\_FAC 8

#define MAX\_CODEC\_FRAME\_RATIO 2

#define MAX\_PAYLOAD\_SIZE 256

typedef struct

{

int bitRate;

int nChannels;

int sampleFreq;

int transFac;

int standardBitrate;

} CODEC\_PARAM;

typedef enum

{

SBR\_MONO,

SBR\_LEFT\_RIGHT,

SBR\_COUPLING,

SBR\_SWITCH\_LRC

}

SBR\_STEREO\_MODE;

typedef struct sbrConfiguration

{

CODEC\_PARAM codecSettings;

int SendHeaderDataTime;

int crcSbr;

int detectMissingHarmonics;

int parametricCoding;

int tran\_thr;

int noiseFloorOffset;

unsigned int useSpeechConfig;

int sbr\_data\_extra;

int amp\_res;

int ana\_max\_level;

int tran\_fc;

int tran\_det\_mode;

int spread;

int stat;

int e;

SBR\_STEREO\_MODE stereoMode;

int deltaTAcrossFrames;

float dF\_edge\_1stEnv;

float dF\_edge\_incr;

int sbr\_invf\_mode;

int sbr\_xpos\_mode;

int sbr\_xpos\_ctrl;

int sbr\_xpos\_level;

int startFreq;

int stopFreq;

int usePs;

int psMode;

int freqScale;

int alterScale;

int sbr\_noise\_bands;

int sbr\_limiter\_bands;

int sbr\_limiter\_gains;

int sbr\_interpol\_freq;

int sbr\_smoothing\_length;

} sbrConfiguration, \*sbrConfigurationPtr ;

unsigned int

IsSbrSettingAvail (unsigned int bitrate,

unsigned int numOutputChannels,

unsigned int sampleRateInput,

unsigned int \*sampleRateCore);

unsigned int

AdjustSbrSettings (const sbrConfigurationPtr config,

unsigned int bitRate,

unsigned int numChannels,

unsigned int fsCore,

unsigned int transFac,

unsigned int standardBitrate);

unsigned int

InitializeSbrDefaults (sbrConfigurationPtr config

);

typedef struct SBR\_ENCODER \*HANDLE\_SBR\_ENCODER;

int

EnvOpen (HANDLE\_SBR\_ENCODER\* hEnvEncoder,

float \*pCoreBuffer,

sbrConfigurationPtr params,

int \*coreBandWith

);

void

EnvClose (HANDLE\_SBR\_ENCODER hEnvEnc);

int

SbrGetXOverFreq(HANDLE\_SBR\_ENCODER hEnv,

int xoverFreq );

int

SbrGetStopFreqRaw(HANDLE\_SBR\_ENCODER hEnv);

int

EnvEncodeFrame (HANDLE\_SBR\_ENCODER hEnvEncoder,

float \*samples,

float \*pCoreBuffer,

unsigned int timeInStride,

unsigned int \*numAncBytes,

unsigned char \*ancData);

#### X1.5.1.3 Resampler (iir32resample.h)

int

IIR32Resample( float \*inbuf,

float \*outbuf,

int inSamples,

int outSamples,

int stride);

int

IIR32GetResamplerFeed( int blockSizeOut);

void

IIR32Init( void);

#### X1.5.1.4 AAC Decoder (aacdecoder.h)

enum {

AAC\_DEC\_OK = 0x0,

AAC\_DEC\_UNSUPPORTED\_FORMAT,

AAC\_DEC\_DECODE\_FRAME\_ERROR,

AAC\_DEC\_INVALID\_CODE\_BOOK,

AAC\_DEC\_UNSUPPORTED\_WINDOW\_SHAPE,

AAC\_DEC\_PREDICTION\_NOT\_SUPPORTED\_IN\_LC\_AAC,

AAC\_DEC\_UNIMPLEMENTED\_PCE,

AAC\_DEC\_UNIMPLEMENTED\_DSE,

AAC\_DEC\_UNIMPLEMENTED\_LFE,

AAC\_DEC\_UNIMPLEMENTED\_CCE,

AAC\_DEC\_UNIMPLEMENTED\_GAIN\_CONTROL\_DATA,

AAC\_DEC\_UNIMPLEMENTED\_EP\_SPECIFIC\_CONFIG\_PARSE,

AAC\_DEC\_UNIMPLEMENTED\_CELP\_SPECIFIC\_CONFIG\_PARSE,

AAC\_DEC\_UNIMPLEMENTED\_HVXC\_SPECIFIC\_CONFIG\_PARSE,

AAC\_DEC\_OVERWRITE\_BITS\_IN\_INPUT\_BUFFER,

AAC\_DEC\_CANNOT\_REACH\_BUFFER\_FULLNESS

};

typedef struct AAC\_DECODER\_INSTANCE \*AACDECODER;

#define FRAME\_SIZE 1024

/\* initialization of aac decoder \*/

AACDECODER CAacDecoderOpen(HANDLE\_BIT\_BUF pBs,

SBRBITSTREAM \*streamSBR,

float \*pTimeData);

int CAacDecoderInit(AACDECODER self,

int samplingRate,

int bitrate);

/\* aac decoder \*/

int CAacDecoder\_DecodeFrame(AACDECODER aacDecoderInstance,

int \*frameSize,

int \*sampleRate,

int \*numChannels,

char \*channelMode,

char errorStatus

);

#### X1.5.1.5 SBR Decoder (sbrdecoder.h)

#define SBR\_EXTENSION 13 /\* 1101 \*/

#define SBR\_EXTENSION\_CRC 14 /\* 1110 \*/

#define MAXNRELEMENTS 2

#define MAXNRSBRCHANNELS MAXNRELEMENTS

#ifdef MONO\_ONLY

#define MAXNRQMFCHANNELS 1

#else

#define MAXNRQMFCHANNELS MAXNRSBRCHANNELS

#endif

#define MAXSBRBYTES 269

typedef enum

{

SBRDEC\_OK = 0,

SBRDEC\_CONCEAL,

SBRDEC\_NOSYNCH,

SBRDEC\_ILLEGAL\_PROGRAM,

SBRDEC\_ILLEGAL\_TAG,

SBRDEC\_ILLEGAL\_CHN\_CONFIG,

SBRDEC\_ILLEGAL\_SECTION,

SBRDEC\_ILLEGAL\_SCFACTORS,

SBRDEC\_ILLEGAL\_PULSE\_DATA,

SBRDEC\_MAIN\_PROFILE\_NOT\_IMPLEMENTED,

SBRDEC\_GC\_NOT\_IMPLEMENTED,

SBRDEC\_ILLEGAL\_PLUS\_ELE\_ID,

SBRDEC\_CREATE\_ERROR,

SBRDEC\_NOT\_INITIALIZED

}

SBR\_ERROR;

typedef enum

{

SBR\_ID\_SCE = 0,

SBR\_ID\_CPE,

SBR\_ID\_CCE,

SBR\_ID\_LFE,

SBR\_ID\_DSE,

SBR\_ID\_PCE,

SBR\_ID\_FIL,

SBR\_ID\_END

}

SBR\_ELEMENT\_ID;

typedef struct

{

int ElementID;

int ExtensionType;

int Payload;

unsigned char Data[MAXSBRBYTES];

}

SBR\_ELEMENT\_STREAM;

typedef struct

{

int NrElements;

int NrElementsCore;

SBR\_ELEMENT\_STREAM sbrElement[MAXNRELEMENTS]; /\* for the delayed frame \*/

}

SBRBITSTREAM;

typedef struct SBR\_DECODER\_INSTANCE \*SBRDECODER;

SBRDECODER openSBR (int sampleRate, int samplesPerFrame, int bDownSample, int bApplyQmfLp) ;

SBR\_ERROR applySBR (SBRDECODER self,

SBRBITSTREAM \* Bitstr,

float \*TimeData,

int \*numChannels,

int frameOK,

int bDownSample,

int bBitstreamDownMix);

### X1.5.2 eAAC+ Fixed-Point (TS 26.411)

#### X1.5.2.1 AAC Encoder (aacenc.h)

/\* here we distinguish between stereo and the mono only encoder \*/

#ifdef MONO\_ONLY

#define MAX\_CHANNELS 1

#else

#define MAX\_CHANNELS 2

#endif

#define AACENC\_BLOCKSIZE 1024 /\*! encoder only takes BLOCKSIZE samples at a time \*/

#define AACENC\_TRANS\_FAC 8 /\*! encoder short long ratio \*/

#define AACENC\_PCM\_LEVEL 1.0 /\*! encoder pcm 0db refernence \*/

/\*-------------------------- defines --------------------------------------\*/

/\*-------------------- structure definitions ------------------------------\*/

struct AAC\_ENCODER;

typedef struct {

Word32 sampleRate; /\* audio file sample rate \*/

Word32 bitRate; /\* encoder bit rate in bits/sec \*/

Word16 nChannelsIn; /\* number of channels on input (1,2) \*/

Word16 nChannelsOut; /\* number of channels on output (1,2) \*/

Word16 bandWidth; /\* targeted audio bandwidth in Hz \*/

} AACENC\_CONFIG;

/\*-----------------------------------------------------------------------------

functionname: AacInitDefaultConfig

description: gives reasonable default configuration

returns: ---

------------------------------------------------------------------------------\*/

void AacInitDefaultConfig(AACENC\_CONFIG \*config);

/\*---------------------------------------------------------------------------

functionname:AacEncOpen

description: allocate and initialize a new encoder instance

returns: AACENC\_OK if success

---------------------------------------------------------------------------\*/

Word16 AacEncOpen (struct AAC\_ENCODER\*\* phAacEnc, /\* pointer to an encoder handle, initialized on return \*/

const AACENC\_CONFIG config); /\* pre-initialized config struct \*/

Word16 AacEncEncode(struct AAC\_ENCODER \*hAacEnc,

Word16 \*timeSignal,

const UWord8 \*ancBytes, /\*!< pointer to ancillary data bytes \*/

Word16 \*numAncBytes, /\*!< number of ancillary Data Bytes, send as fill element \*/

UWord8 \*outBytes, /\*!< pointer to output buffer \*/

Word32 \*numOutBytes /\*!< number of bytes in output buffer \*/

);

/\*---------------------------------------------------------------------------

functionname:AacEncClose

description: deallocate an encoder instance

---------------------------------------------------------------------------\*/

void AacEncClose (struct AAC\_ENCODER\* hAacEnc); /\* an encoder handle \*/

#### X1.5.2.2 SBR Encoder (sbr\_main.h)

/\* core coder helpers \*/

#define MAX\_TRANS\_FAC 8

#define MAX\_CODEC\_FRAME\_RATIO 2

#define MAX\_PAYLOAD\_SIZE 128

typedef struct

{

Word32 bitRate;

Word16 nChannels;

Word32 sampleFreq;

Word16 transFac;

Word32 standardBitrate;

} CODEC\_PARAM;

typedef enum

{

SBR\_MONO,

SBR\_LEFT\_RIGHT,

SBR\_COUPLING,

SBR\_SWITCH\_LRC

}

SBR\_STEREO\_MODE;

typedef struct sbrConfiguration

{

/\*

core coder dependent configurations

\*/

CODEC\_PARAM codecSettings; /\*!< Core coder settings, to be set from core coder \*/

Word16 SendHeaderDataTime; /\*!< SBR-Header send update frequency in msec \*/

Word16 crcSbr; /\*!< Flag: usage of SBR-CRC \*/

Word16 detectMissingHarmonics; /\*!< Flag: usage of missing harmonics detection \*/

Word16 parametricCoding; /\*!< Flag: usage of parametric coding tool \*/

/\*

core coder dependent tuning parameters

\*/

Word32 tran\_thr; /\*!< SBR transient detector threshold (\* 100) \*/

Word32 noiseFloorOffset; /\*! Noise floor offset \*/

UWord16 useSpeechConfig; /\*!< Flag: adapt tuning parameters according to speech \*/

/\*

core coder independent configurations

\*/

Word16 sbrFrameSize; /\*!< SBR frame size in samples, will be calculated from core coder settings \*/

Word16 sbr\_data\_extra; /\*!< Flag usage of data extra \*/

Word16 amp\_res; /\*!< Amplitude resolution \*/

Word32 ana\_max\_level; /\*!< Noise insertion maximum level \*/

Word16 tran\_fc; /\*!< Transient detector start frequency \*/

Word16 tran\_det\_mode; /\*!< Transient detector mode \*/

Word16 spread; /\*!< Flag: usage of SBR spread \*/

Word16 stat; /\*!< Flag: usage of static framing \*/

Word16 e; /\*!< Number of envelopes when static framing is chosen \*/

SBR\_STEREO\_MODE stereoMode; /\*!< SBR stereo mode \*/

Word16 deltaTAcrossFrames; /\*!< Flag: allow time-delta coding \*/

Word16 sbr\_invf\_mode; /\*!< Inverse Filtering mode \*/

Word16 sbr\_xpos\_mode; /\*!< Transposer mode \*/

Word16 sbr\_xpos\_ctrl; /\*!< Transposer control \*/

Word16 sbr\_xpos\_level; /\*!< Transposer 3rd order level \*/

Word16 startFreq; /\*!< The start frequency table index \*/

Word16 stopFreq; /\*!< The stop frequency table index \*/

Word16 usePs; /\*!< Flag: usage of parametric stereo \*/

Word32 psMode;

/\*

header\_extra1 configuration

\*/

Word16 freqScale; /\*!< Frequency grouping \*/

Word16 alterScale; /\*!< Scale resolution \*/

Word16 sbr\_noise\_bands; /\*!< Number of noise bands \*/

/\*

header\_extra2 configuration

\*/

Word16 sbr\_limiter\_bands; /\*!< Number of limiter bands \*/

Word16 sbr\_limiter\_gains; /\*!< Gain of limiter \*/

Word16 sbr\_interpol\_freq; /\*!< Flag: use interpolation in freq. direction \*/

Word16 sbr\_smoothing\_length; /\*!< Flag: choose length 4 or 0 (=on, off) \*/

} sbrConfiguration, \*sbrConfigurationPtr ;

UWord32

IsSbrSettingAvail (UWord32 bitrate,

UWord16 numOutputChannels,

UWord32 sampleRateInput,

UWord32 \*sampleRateCore);

UWord32

AdjustSbrSettings (const sbrConfigurationPtr config,

UWord32 bitRate,

UWord16 numChannels,

UWord32 fsCore,

UWord16 transFac,

UWord32 standardBitrate);

UWord32

InitializeSbrDefaults (sbrConfigurationPtr config);

typedef struct SBR\_ENCODER \*HANDLE\_SBR\_ENCODER;

Word32

EnvOpen (HANDLE\_SBR\_ENCODER\* hEnvEncoder,

sbrConfigurationPtr params,

Word16 \*coreBandWith /\*\*< encoder (lowband) bandwith in Hz \*/

);

void

EnvClose (HANDLE\_SBR\_ENCODER hEnvEnc);

Word32

SbrGetXOverFreq(HANDLE\_SBR\_ENCODER hEnv,

Word32 xoverFreq );

Word32

SbrGetStopFreqRaw(HANDLE\_SBR\_ENCODER hEnv);

Word32

EnvEncodeFrame (HANDLE\_SBR\_ENCODER hEnvEncoder,

Word16 \*samples, /\*!< time samples, always interleaved \*/

Word16 \*pCoreBuffer,

UWord16 timeInStride,

Word16 \*numAncBytes,

UWord8 \*ancData);

#### X1.5.2.3 Resample (downsample\_FIR.h)

#define BUFFER\_SIZE\_2\_1 64

#define BUFFER\_SIZE\_3\_2 128

typedef struct

{

const Word16 \*coeffFIR; /\*! pointer to filter coeffs \*/

Word16 noOffCoeffs; /\*! number of filter coeffs \*/

Word16 delayLine[BUFFER\_SIZE\_2\_1]; /\*! ringbuffer 1 input delay line \*/

} FIR\_FILTER\_2\_1;

typedef struct

{

const Word16 \*coeffFIR; /\*! pointer to filter coeffs \*/

Word16 noOffCoeffs; /\*! number of filter coeffs \*/

Word16 delayLine[BUFFER\_SIZE\_3\_2]; /\*! ringbuffer 1 input delay line \*/

} FIR\_FILTER\_3\_2;

typedef struct

{

FIR\_FILTER\_2\_1 firFilter; /\*! fir filter instance \*/

Word32 fIn; /\*! input fs \*/

Word32 fOut ; /\*! output fs \*/

Word32 delay; /\*! delay input vs. output in samples \*/

} RESAMPLER\_FIR\_2\_1;

typedef struct

{

FIR\_FILTER\_3\_2 firFilter; /\*! fir filter instance \*/

Word32 fIn; /\*! input fs \*/

Word32 fOut ; /\*! output fs \*/

Word32 delay; /\*! delay input vs. output in samples \*/

} RESAMPLER\_FIR\_3\_2;

Word32 InitResampler\_firDown2(RESAMPLER\_FIR\_2\_1 \*ReSampler\_fir, /\*!< pointer to downsampler instance \*/

Word32 fIn, /\*!< Input Sampling frequency \*/

Word32 fOut); /\*!< Output Sampling frequency \*/

Word32 InitResampler\_firDown3(RESAMPLER\_FIR\_3\_2 \*ReSampler\_fir, /\*!< pointer to downsampler instance \*/

Word32 fIn, /\*!< Input Sampling frequency \*/

Word32 fOut); /\*!< Output Sampling frequency \*/

Word32 InitResampler\_firUp2(RESAMPLER\_FIR\_2\_1 \*ReSampler\_fir, /\*!< pointer to downsampler instance \*/

Word32 fIn, /\*!< Input Sampling frequency \*/

Word32 fOut); /\*!< Output Sampling frequency \*/

Word32 InitResampler\_firDown32(RESAMPLER\_FIR\_2\_1 \*ReSampler\_firUp2, /\*!< pointer to downsampler instance \*/

RESAMPLER\_FIR\_3\_2 \*ReSampler\_firDown3, /\*!< pointer to downsampler instance \*/

Word32 fIn, /\*!< Input Sampling frequency \*/

Word32 fOut); /\*!< Output Sampling frequency \*/

Word32 Resample\_firDown2(RESAMPLER\_FIR\_2\_1 \*ReSampler\_fir, /\*!< pointer to downsampler instance \*/

Word16 \*inSamples, /\*!< pointer to input samples \*/

Word16 numInSamples, /\*!< number of input samples \*/

Word16 inStride, /\*!< increment of input samples \*/

Word16 \*outSamples, /\*!< pointer to output samples \*/

Word16 \*numOutSamples, /\*!< pointer to number of output samples \*/

Word16 outStride /\*!< increment of output samples \*/

);

Word32 Resample\_firUp2(RESAMPLER\_FIR\_2\_1 \*ReSampler\_fir, /\*!< pointer to downsampler instance \*/

Word16 \*inSamples, /\*!< pointer to input samples \*/

Word16 numInSamples, /\*!< number of input samples \*/

Word16 inStride, /\*!< increment of input samples \*/

Word16 \*outSamples, /\*!< pointer to output samples \*/

Word16 \*numOutSamples, /\*!< pointer tp number of output samples \*/

Word16 outStride /\*!< increment of output samples \*/

);

Word32 Resample\_firDown32(RESAMPLER\_FIR\_2\_1 \*ReSampler\_firUp2, /\*!< pointer to downsampler instance \*/

RESAMPLER\_FIR\_3\_2 \*ReSampler\_firDown3, /\*!< pointer to downsampler instance \*/

Word16 \*inSamples, /\*!< pointer to input samples \*/

Word16 numInSamples, /\*!< number of input samples \*/

Word16 inStride, /\*!< increment of input samples \*/

Word16 \*outSamples, /\*!< pointer to output samples \*/

Word16 \*numOutSamples, /\*!< pointer tp number of output samples \*/

Word16 outStride /\*!< increment of output samples \*/

);

#### X1.5.2.4 AAC Decoder (aacdecoder.h)

typedef enum {

AAC\_DEC\_OK = 0x0,

AAC\_DEC\_UNSUPPORTED\_FORMAT,

AAC\_DEC\_DECODE\_FRAME\_ERROR,

AAC\_DEC\_INVALID\_CODE\_BOOK,

AAC\_DEC\_UNSUPPORTED\_WINOW\_SHAPE,

AAC\_DEC\_PREDICTION\_NOT\_SUPPORTED\_IN\_LC\_AAC,

AAC\_DEC\_UNIMPLEMENTED\_CCE,

AAC\_DEC\_UNIMPLEMENTED\_PCE,

AAC\_DEC\_UNIMPLEMENTED\_LFE,

AAC\_DEC\_UNIMPLEMENTED\_GAIN\_CONTROL\_DATA,

AAC\_DEC\_OVERWRITE\_BITS\_IN\_INPUT\_BUFFER,

AAC\_DEC\_CANNOT\_REACH\_BUFFER\_FULLNESS,

AAC\_DEC\_TNS\_RANGE\_ERROR,

AAC\_DEC\_TNS\_ORDER\_ERROR

}

AAC\_DEC\_STATUS;

typedef struct AAC\_DECODER\_INSTANCE \*AACDECODER;

#define FRAME\_SIZE 1024

/\* initialization of aac decoder \*/

AACDECODER CAacDecoderOpen(HANDLE\_BIT\_BUF hBitBufCore,

SBRBITSTREAM \*streamSbr,

Word32 samplingRate);

/\* aac decoder \*/

Word16 CAacDecoder\_DecodeFrame(AACDECODER aacDecoderInstance,

Word16 \*frameSize,

Word32 \*sampleRate,

Word8 \*channelMode,

Word16 \*numChannels,

Word16 \*timeData,

Flag frameOK

);

#### X1.5.2.5 SBR Decoder (sbrdecoder.h)

#define SBR\_EXTENSION 13 /\* 1101 \*/

#define SBR\_EXTENSION\_CRC 14 /\* 1110 \*/

#define MAXNRELEMENTS 2

#define MAXNRSBRCHANNELS 2

#define MAXSBRBYTES 269

#define SBRDEC\_OK 0

typedef enum

{

SBR\_ID\_SCE = 0,

SBR\_ID\_CPE,

SBR\_ID\_CCE,

SBR\_ID\_LCS,

SBR\_ID\_LFE,

SBR\_ID\_DSE,

SBR\_ID\_PCE,

SBR\_ID\_FIL,

SBR\_ID\_END

}

SBR\_ELEMENT\_ID;

typedef struct

{

Word16 elementID; /\*!< ID\_SCE (mono) or ID\_CPE (stereo) \*/

Word16 extensionType; /\*!< e.g. SBR\_EXTENSION or SBR\_EXTENSION\_MPEG \*/

Word16 sizePayload; /\*!< length of data \*/

Word8 \*pData; /\*!< Pointer to actual data \*/

}

SBR\_ELEMENT\_STREAM;

typedef struct

{

Word16 nrElements; /\*!< Number of valid SBR streams \*/

SBR\_ELEMENT\_STREAM sbrElement[MAXNRELEMENTS];

}

SBRBITSTREAM;

typedef struct SBR\_DECODER\_INSTANCE \*SBRDECODER;

SBRDECODER openSBR (Word32 sampleRate, Word16 samplesPerFrame, Flag bDownSample, Flag bApplyQmfLp) ;

Word16 applySBR (SBRDECODER self,

SBRBITSTREAM \*Bitstr,

Word16 \*TimeData,

Word16 \*numChannels,

Flag frameOK,

Flag bDownSample,

Flag bBitstreamDownMix);

## X1.6 AMR-WB+

### X1.6.1 AMR-WB+ Fixed-Point (TS 26.273)

#### X1.6.1.1 Encoder (enc\_if\_fx.h)

#define L\_FRAME16k 320 /\* Frame size at 16kHz \*/

#define NB\_SERIAL\_MAX 61 /\* max serial size \*/

int E\_IF\_encode\_fx(void \*st, Word16 mode, Word16 \*speech,

UWord8 \*serial, Word16 dtx);

void \*E\_IF\_init\_fx(void);

void E\_IF\_exit\_fx(void \*state);

void E\_IF\_encode\_first\_fx(void \*st, Word16 \*speech);

#### X1.6.1.2 Decoder (dec\_if\_fx.h)

#define NB\_SERIAL\_MAX 61 /\* max serial size \*/

#define L\_FRAME16k 320 /\* Frame size at 16kHz \*/

#define \_good\_frame 0

#define \_bad\_frame 1

#define \_lost\_frame 2

#define \_no\_frame 3

void D\_IF\_decode\_fx(void \*st, UWord8 \*bits, Word16 \*synth, Word16 bfi);

void \* D\_IF\_init\_fx(void);

void D\_IF\_exit\_fx(void \*state);

### X1.6.2 AMR-WB+ Floating-Point (TS 26.304)

#### X1.6.2.1 Encoder (proto\_func.h)

void init\_coder\_amrwb\_plus(Coder\_State\_Plus \* st, int num\_chan, int fscale, short use\_case\_mode, short full\_reset);

int coder\_amrwb\_plus\_stereo(float channel\_right[], /\* input: used on mono and stereo \*/

float channel\_left[], /\* input: used on stereo only \*/

int codec\_mode, /\* input: AMR-WB+ mode (see cnst.h) \*/

int L\_frame, /\* input: 80ms frame size \*/

short serial[], /\* output: serial parameters \*/

Coder\_State\_Plus \* st, /\* i/o : coder memory state \*/

short useCaseB, int bwe\_flag, /\* 32kHz NBWE \*/

int StbrMode);

int coder\_amrwb\_plus\_mono(float channel\_right[], /\* input: used on mono and stereo \*/

int codec\_mode, /\* input: AMR-WB+ mode (see cnst.h) \*/

int L\_frame, /\* input: 80ms frame size \*/

short serial[], /\* output: serial parameters \*/

Coder\_State\_Plus \* st, /\* i/o : coder memory state \*/

short useCaseB, int bwe\_flag /\* 32kHz NBWE \*/

);

void coder\_amrwb\_plus\_mono\_first(float channel\_right[], /\* input: used on mono and stereo \*/

int n\_channel, /\* input: 1 or 2 (mono/stereo) \*/

int L\_frame, /\* input: frame size \*/

int L\_next, /\* input: lookahead \*/

int bwe\_flag, /\* for 32kHz NBWE \*/

Coder\_State\_Plus \* st /\* i/o : coder memory state \*/

);

int coder\_amrwb\_plus\_first(float channel\_right[], /\* input: used on mono and stereo \*/

float channel\_left[], /\* input: used on stereo only \*/

int n\_channel, /\* input: 1 or 2 (mono/stereo) \*/

int L\_frame, /\* input: frame size \*/

int L\_next, /\* input: lookahead \*/

int bwe\_flag, /\* AriL: for 32kHz NBWE \*/

Coder\_State\_Plus \* st /\* i/o : coder memory state \*/

);

#### X1.6.2.2 Decoder (proto\_func.h)

void init\_decoder\_amrwb\_plus(Decoder\_State\_Plus \* st, int num\_chan, int fscale, short full\_reset);

int decoder\_amrwb\_plus(int codec\_mode, /\* input: AMR-WB+ mode (see cnst.h) \*/

short serial[], /\* input: serial parameters (4x20ms) \*/

int bad\_frame[], /\* input: bfi (bad\_frame[4]) \*/

int L\_frame, /\* input: frame size of synthesis \*/

int n\_channel, /\* input: 1 or 2 (mono/stereo) \*/

float channel\_right[], /\* (o): used on mono and stereo \*/

float channel\_left[], /\* (o): used on stereo only \*/

Decoder\_State\_Plus \* st, /\* i/o : decoder memory state \*/

int fscale,

int StbrMode,

int mono\_dec\_stereo,

short upsamp\_fscale);

void decoder\_amrwb\_plus\_1(float \*chan\_right,

float \*chan\_left,

int \*mod,

int \*param,

int \*prm\_hf\_right,

int \*prm\_hf\_left,

int \*nbits\_AVQ,

int codec\_mode,

int \*bad\_frame,

int \*bad\_frame\_hf,

float \*AqLF,

float \*synth,

int \*pitch,

float \*pit\_gain, Decoder\_State\_Plus \* st, int n\_channel, int L\_frame, int bwe\_flag,

int mono\_dec\_stereo);

## X1.7 IVAS

tbd