Client AMR Encoding and Decoding Library

API Reference

Issue	02
Date	2008-11-30
Part Number	N/A

HiSilicon Technologies CO., LIMITED. provides customers with comprehensive technical support and service. Please feel free to contact our local office or company headquarters.

HiSilicon Technologies CO., LIMITED.

Address:	Manufacture Center of Huawei Electric, Huawei Base, Bantian, Longgang District, Shenzhen, 518129, People’s Republic of China
Website:	http://www.hisilicon.com
Tel:	+86-755-28788858
Fax:	+86-755-28357515
Email:	support@hisilicon.com

Contents

Tables

About This Document

Purpose

This document describes the document contents, related product versions, intended audience, conventions and update history.

Related Versions

The following table lists the product versions related to this document.

Product Name	Version
Hi3510 Communications Media Processor	V100
Hi3511 H.264 Encoding and Decoding Processor	V100
Hi3512 H.264 Encoding and Decoding Processor	V100

Intended Audience

This document is intended for:

- Software development engineers
- Technical support engineers

Organization

This document is organized as follows:

Chapter	Description
1 Overview	Describes the classification of the reference information for the AMR APIs. It also provides the information on the layout of the API and structure descriptions.
2 API Reference	Describes each API in detail.

Chapter	Description
3 Other Information	Provides additional information to the API reference, including data structures and error codes.
A Acronyms and Abbreviations	Lists the abbreviations and acronyms and gives their full spellings.

Conventions

Symbol Conventions

The following symbols may be found in this document. They are defined as follows.

Symbol	Description
⚠ DANGER	Indicates a hazard with a high level of risk which, if not avoided, will result in death or serious injury.
⚠ WARNING	Indicates a hazard with a medium or low level of risk that, if not avoided, could result in minor or moderate injury.
⚠ CAUTION	Indicates a potentially hazardous situation that, if not avoided, could cause equipment or component damage, data loss, and performance degradation, or unexpected results.
TIP	Indicates a tip that may help you solve a problem or save time.
NOTE	Provides additional information to emphasize or supplement important points of the main text.

General Conventions

Convention	Description
Times New Roman	Normal paragraphs are in Times New Roman.
Boldface	Names of files, directories, folders, and users are in **boldface**. For example, log in as user **root**.
Italic	Book titles are in *italics*.
`Courier New`	Terminal display is in `Courier New`.

Update History

Updates between document versions are cumulative. Therefore, the latest document version contains all updates made to previous versions.

Updates in Issue 02 (2008-11-20)

The initial commercial release has the following updates:

Information about the Hi3511/Hi3512 is added.

Updates in Issue 01 (2007-04-30)

Initial release.

1 Overview

1.1 Introduction to AMR-NB

Adaptive Multi-Rate Narrow-Band (AMR-NB), a standard for audio encoding and decoding, is presented by the 3rd Generation Partnership Project (3GPP). It is used for the third generation mobile audio devices.

The features of AMR-NB are as follows:

- AMR-NB supports the following bit rates:
  - 12.2 kbit/s
  - 10.2 kbit/s
  - 7.95 kbit/s
  - 7.40 kbit/s
  - 6.70 kbit/s
  - 5.90 kbit/s
  - 5.15 kbit/s
  - 4.75 kbit/s

  All these rates are based on the algebraic code excited linear prediction (ACELP).
- To lower the bit rate, AMR-NB supports the discontinuous transmission (DTX), voice activity detection (VAD), and comfort noise generation (CNG). The VAD1 algorithm is applied.
- AMR-NB supports 8 kHz sampling and 20 ms (160 sampling points) frames input/output.
- AMR-NB supports three file storage formats:
  - MIME

    It is an RTP payload format. For details, see the *rfc4239 AMR and AMR-WB Storage Format*.
  - IF1

    For details, see **3GPP 26101-600.doc**.
  - IF2

    For details, see **3GPP 26101-600.doc**.
- AMR-NB is compliant with the following standards for 3GPP GSM-AMR.
  - 3GPP TS 26.071 V4.0.0 AMR Speech Codec; General Description

- 3GPP TS 26.090 V4.0.0 AMR Speech Codec; Transcoding functions
- 3GPP TS 26.091 V4.0.0 AMR Speech Codec; Error concealment of lost frames
- 3GPP TS 26.092 V4.0.0 AMR Speech Codec; Comfort noise aspects
- 3GPP TS 26.093 V4.0.0 AMR Speech Codec; Source controlled rate operation
- 3GPP TS 26.094 V4.0.0 AMR Speech Codec; Voice activity detector

- AMR-NB is the same as the following standard reference code of 3GPP GSM-AMR:
  - 3GPP TS 26.071 V4.0.0 ANSI-C code for AMR speech codec (Code Version 7.5.0)
- AMR-NB is tested by using the following test stream provided by 3GPP GSM-AMR:
  - 3GPP TS 26.074 V4.0.0 AMR Speech Codec; Test sequences
- The bit rate can be selected on a frame basis. In addition, the DTX can be enabled or disabled on a frame basis.

1.2 Data Types

Table 1-1 lists all the data types used in the software development kit (SDK).

Table 1-1 Data type description

Data Type	Description
HI_U8	8-bit unsigned characters
HI_S8	8-bit signed characters
HI_U16	16-bit unsigned integers
HI_S16	16-bit signed integers
HI_U32	32-bit unsigned integers
HI_S32	32-bit signed integers
HI_VOID	void

1.3 API List

Table 1-2 lists the APIs defined by the encode SDK.

Table 1-2 APIs defined by the encode SDK

API	Description
AMR_Encode_Init	Initializes an encode device.
AMR_Encode_Frame	Encodes frames.
AMR_Encode_Exit	Releases an encode device.

Table 1-3 lists the APIs defined by the decode SDK.

Table 1-3 APIs defined by the decode SDK

API	Description
AMR_Decode_Init	Initializes a decode device.
AMR_Get_Length	Gets the frame length. The unit is HI_U8.
AMR_Decode_Frame	Decodes streams.
AMR_Decode_Exit	Releases a decode device.

1.4 Classification of Reference Information

Table 1-4 lists the types of reference information.

Table 1-4 Types of reference information

Type	Description
API reference	Describes each API in detail.
Other information	Provides additional information to the API reference, such as data types and error codes.

1.5 Information Associated with APIs

The data types and error code definitions associated with the APIs are described in 3 "Other Information."

1.6 Layout of API Descriptions

This document describes the APIs from the aspects listed in Table 1-5.

Table 1-5 API description layout

Aspect	Description
Purpose	Describes the function of an API.
Syntax	Presents the syntax expression of an API.
Description	Describes the working procedure of an API.
Parameter	Lists the parameters of an API and provides the parameter descriptions and properties.

Aspect	Description
Return Value	Lists the return values of an API and provides the descriptions of return values.
Error Code	Lists the error codes of an API and provides the meaning of the error codes.
Request	Lists the header files contained in an API and provides library files to be linked for the API compilation.
Note	Describes the notes that are worth your special attention when you call an API.
Example	Provides an example of calling an API.
See Also	Describes information related to an API.

1.7 Layout of Structure Descriptions

This document describes the structures from the aspects listed in Table 1-6

Table 1-6 Structure description layout

Aspect	Description
Description	Describes the main function implemented by a structure.
Definition	Presents the structure definition.
Note	Provides the notes that are worth your special attention when you use a structure.

2 API Reference

2.1 APIs for Initialization

AMR_Encode_Init

[Purpose]

This API is used to initialize an encode device in an encode task.

[Syntax]

```
#include "amr_enc.h"
HI_S32 AMR_Encode_Init (HI_VOID **pEncState, HI_S16 dtx);
```

[Description]

The encode device is initialized according to the encode device pointer that is transmitted from the upper layer. The DTX enable is defined by users. If the DTX is enabled, the encoder starts the voice activity detection. If the encoder/decoder finds a frame that contains no voice activities, it reduces the bit rate to lower the user power consumption and to improve the network capacity.

[Parameter]

Parameter	Description	Input/Output	Global/Local
pEncState	An encode device that is specified by a user.	Input/Output	Global
dtx	The DTX enable. 0: Disable 1: Enable	Input	Local

[Return Value]

Return Value	Description
0	Initializing the encode device succeeded.

Return Value	Description
Non-zero	Initializing the encode device failed. The return value is an error code.

[Error Code]

Error Code	Description
HI_ERR_AMRNB_INIT_FAIL	Initializing the encode device failed.

[Request]

- Header file: /include/amr_enc.h
- Library file: /lib/amrnb.lib

[Note]

None.

[Example]

```
#include "amr_enc.h"
HI_VOID *pEncState = NULL;
HI_S16   dtx = 0;
if (AMR_Encode_Init(&pEncState, dtx))
{
    fprintf(stderr, "\nerror AMR_Encode_Init fail: %s\n",strerror(errno));
    exit(-1);
}
```

[See Also]

- HI_S32   AMR_Encode_Frame
- HI_VOID   AMR_Encode_Exit

AMR_Encode_Exit

[Purpose]

This API is used to release an encode device after the encode task is complete.

[Syntax]

```
#include "amr_enc.h"
HI_VOID AMR_Encode_Exit (HI_VOID **pEncState);
```

[Description]

None.

[Parameter]

Parameter	Description	Input/Output	Global/Local
pEncState	An encode device that is specified by a user.	Input	Global

[Return Value]

None.

[Request]

- Header file: /include/amr_enc.h
- Library file: /lib/amrnb.lib

[Note]

None.

[Example]

```
#include "amr_enc.h"
HI_VOID *pEncState = NULL;
AMR_Encode_Exit(&pEncState);
```

[See Also]

- HI_S32 AMR_Encode_Init
- HI_S32 AMR_Encode_Frame

AMR_Decode_Init

[Purpose]

This API is used to initialize a decode device in a decode task.

[Syntax]

```
#include "amr_dec.h"
HI_S32 AMR_Decode_Init (HI_VOID **pDecState);
```

[Description]

None.

[Parameter]

Parameter	Description	Input/Output	Global/Local
pDecState	A decode device that is specified by a user.	Input/Output	Global

[Return Value]

Return Value	Description
0	Initializing the decode device succeeded.
Non-zero	Initializing the decode device failed. The return value is an error code.

[Error Code]

Error Code	Description
HI_ERR_AMRNB_INIT_FAIL	Initializing the decode device failed.

[Request]

- Header file: /include/amr_dec.h
- Library file:/lib/amrnb.lib

[Note]

None.

[Example]

```
#include "amr_enc.h"
HI_VOID *pDecState = NULL;
if (AMR_Decode_Init(&pDecState))
{
    fprintf(stderr, "\nerror AMR_Decode_Init fail: %s\n",strerror(errno));
    exit(-1);
}
```

[See Also]

- HI_S32 AMR_Decode_Frame
- HI_VOID AMR_Decode_Exit

AMR_Decode_Exit

[Purpose]

This API is used to release a decode device after the decode task is complete.

[Syntax]

```
#include "amr_dec.h"
HI_VOID AMR_Decode_Exit (HI_VOID ** pDecState);
```

[Description]

None.

[Parameter]

Parameter	Description	Input/Output	Global/Local
pDecState	A decode device that is specified by a user.	Input	Global

[Return Value]

None.

[Request]

- Header file: /include/amr_dec.h
- Library file: /lib/amrnb.lib

[Note]

None.

[Example]

```
#include "amr_dec.h"
HI_VOID *pDecState = NULL;
AMR_Decode_Exit(&pDecState);
```

[See Also]

- HI_S32 AMR_Decode_Init
- HI_S32 AMR_Decode_Frame

2.2 APIs Used by Applications

AMR_Encode_Frame

[Purpose]

This API is used to encode the raw voice data.

[Syntax]

```
#include "amr_enc.h"
HI_S32 AMR_Encode_Frame (HI_VOID *pEncState,
HI_S16 *pInBuf,
HI_U8 *pOutBuf,
enum Mode mode,
enum Format frame_type);
```

[Description]

A user inputs the voice data to be encoded and ensures that its length is L_FRAME (160; the unit is HI_S16). The encoder starts encoding data based on the bit rate specified by the **enum Mode** variable. After the encoding, based on the format type specified by the **enum Format** variable, the encoder packs the encoded data and saves them in the buffer area specified by the user. Finally, the encoder returns the length of the encoded data (the unit is HI_U8).

[Parameter]

Parameter	Description	Input/Output	Global/Local
pEncState	An encode device that is specified by a user.	Input/Output	Global
pInBuf	The input buffer for saving the data to be encoded.	Input	Local
pOutBuf	The output buffer for saving the encoded data.	Output	Local
mode	The bit rate for encoding data.	Input	Local
frame_type	The frame format.	Input	Local

[Return Value]

Return Value	Description
Positive values	Encoding the raw voice data succeeded. The return value is also the length of the encoded data. Its unit is HI_U8.
Negative values	Encoding the raw voice data failed. The return value is an error code.

[Error Code]

Error Code	Description
HI_ERR_AMRNB_INVALID_DEVICE	Invalid encode device.
HI_ERR_AMRNB_INVALID_INBUF	Invalid input buffer.
HI_ERR_AMRNB_INVALID_OUTBUF	Invalid output buffer.
HI_ERR_AMRNB_MODE_TYPE	Invalid encode bit rate.
HI_ERR_AMRNB_FORMAT_TYPE	Invalid frame format.
HI_ERR_AMRNB_ENCODE_FAIL	Encoding the data failed.

[Request]

- Header file: /include/amr_enc.h
- Library file: /lib/amrnb.lib

[Note]

- Ensure that the length of the input voice data is L_FRAME (160; the unit is HI_S16).

- Ensure that the size of the output buffer is equal to or larger than MAX_PACKED_SIZE (35; the unit is HI_U8).

[Example]

```
#include "amr_enc.h"
HI_S16 pInBuf[L_FRAME];        /*L_FRAME is 160 (the length of the input
data)*/
/*MAX_PACKED_SIZE is the maximum length of the encoded data; its unit is HI_U8*/
HI_U8  pOutBuf[MAX_PACKED_SIZE];
HI_VOID *pEncState = NULL;
HI_S32 packed_size;
HI_S16 dtx = 1;
enum Mode mode = MR122;
enum Format frame_type = MIME;

if (AMR_Encode_Init(&pEncState, dtx))
{
    fprintf(stderr, "\nerror AMR_Encode_Init fail: %s\n",strerror(errno));
    exit(-1);
}
If (frame_type == MIME)  /* In the case of local storage, if the frame format
is MIME, the file header needs to be added.*/
{
    fwrite(AMR_MAGIC_NUMBER, sizeof(HI_S8), strlen(AMR_MAGIC_NUMBER),
           file_out);
}
while(fread(pInBuf, sizeof(HI_S16), L_FRAME, file_in) == L_FRAME)
{
    packed_size=
          AMR_Encode_Frame (pEncState, pInBuf, pOutBuf, mode, frame_type);
    if (packed_size < 0)
    {
        fprintf(stderr, "\nerror AMR_Encode_Frame fail:
                        %s\n",strerror(errno));
        exit(-1);
    }
    fwrite (pOutBuf, sizeof(HI_U8), packed_size, file_out);
}
AMR_Encode_Exit(&pEncState);
```

[See Also]

- HI_S32 AMR_Encode_Init
- HI_VOID AMR_Encode_Exit

AMR_Get_Length

[Purpose]

This API is used to get the length of a frame according to the frame header. The unit is HI_U8.

[Syntax]

```
HI_S32 AMR_Get_Length(enum Format frame_type, HI_U8 toc)
```

[Description]

AMR-NB supports eight of bit rates for encoding and decoding data. In addition, it also supports three frame formats. Therefore, the length of the encoded data may vary. Before the decoding, to obtain the stream properly, you need to get the length of the frame according to the frame header.

[Parameter]

Parameter	Description	Input/Output	Global/Local
frame_type	The frame format.	Input	Local
toc	The frame header.	Input	Local

[Return Value]

Return Value	Description
Non-negative values	Computing frame length succeeded. The return value is the frame stream length minus 1. The unit is HI_U8.
Negative values	Computing frame length failed. The return value is an error code.

[Error Code]

Error Code	Description
HI_ERR_AMRNB_FORMAT_TYPE	Invalid frame format.

[Request]

- Header file: /include/amr_dec.h
- Library file: /lib/amrnb.lib

[Note]

- Ensure that the frame format remains the same before the encoding and after the decoding. In other words, remain the **enum Format** variable the same.
- AMR-NB reads the frame header (1 byte) before the decoding, and then it gets the frame length accordingly. Since AMR-NB reads 1 byte already, the return value of AMR_Get_Length is the frame stream length minus 1.

[Example]

```
#include "amr_dec.h"
HI_U8 pInBuf[MAX_PACKED_SIZE];
HI_S32 packed_size;
/* To obtain information from the frame header and save the information in
the input buffer*/
fread(&pInBuf[0], sizeof(HI_U8), 1, file_in);
packed_size = AMR_Get_Length(MIME, pInBuf[0]);
```

[See Also]

HI_S32 AMR_Decode_Frame

AMR_Decode_Frame

[Purpose]

This API is used by the decoder to decode an input frame. After the decoding, the decoder outputs the decoded voice data.

[Syntax]

```
#include "amr_dec.h"
HI_S32 AMR_Decode_Frame(HI_VOID *pDecState,
HI_U8  *pInBuf,
HI_S16 *pOutBuf,
enum Format frame_type);
```

[Description]

A user inputs the encoded stream. The decoder decodes the stream and save the decoded voice data in the output buffer specified by the user. The length of the decoded voice data is 160 (the unit is HI_S16).

[Parameter]

Parameter	Description	Input/Output	Global/Local
pDecState	A decode device that is specified by a user.	Input/Output	Global
pInBuf	The input buffer for saving the data to be decoded.	Input	Local
pOutBuf	The output buffer for saving the decoded data.	Output	Local
frame_type	The frame format.	Input	Local

[Return Value]

Return Value	Description
0	Decoding the input frame succeeded.
Non-zero	Decoding the input frame failed. The return value is an error code.

[Error Code]

Error Code	Description
HI_ERR_AMRNB_INVALID_DEVICE	Invalid encoder/decoder.
HI_ERR_AMRNB_INVALID_INBUF	Invalid input buffer.
HI_ERR_AMRNB_INVALID_OUTBUF	Invalid output buffer.
HI_ERR_AMRNB_FORMAT_TYPE	Invalid frame format.
HI_ERR_AMRNB_DECODE_FAIL	Decoding the stream failed.

[Request]

- Header file: /include/amr_dec.h
- Library file: /lib/amrnb.lib

[Note]

- Ensure that the frame format remains the same. In other words, remain the **enum Format** variable the same.
- Ensure that the size of the input buffer is equal to or larger than MAX_PACKED_SIZE (35; the unit is HI_U8).
- Ensure that the size of the output buffer is equal to or larger than L_FRAME (160; the unit is HI_S16).

[Example]

```
#include "amr_dec.h"
HI_VOID *pDecState = NULL;
HI_U8 pInBuf[MAX_PACKED_SIZE];
HI_S16 pOutBuf[L_FRAME];
enum Format frame_type = MIME;
HI_S32 packed_size;

If (frame_type == MIME)  /* In the case of local decoding, if the frame format
is MIME, the file header needs to be read*/
{
fread(magic, sizeof(HI_S8), strlen(AMR_MAGIC_NUMBER), file_in);
if (strncmp((const HI_S8 *)magic, AMR_MAGIC_NUMBER,
strlen(AMR_MAGIC_NUMBER)))
    {
        fprintf(stderr, "%s%s\n", "Invalid magic number: ", magic);
        exit(-1);
    }
}
if (AMR_Decode_Init(&pDecState))
{
    fprintf(stderr, "\nerror AMR_Decode_Init fail: %s\n", strerror(errno));
    exit(-1);
}

while(fread (&pInBuf[0], sizeof(HI_U8), 1, file_in) == 1)
{
     packed_size = AMR_Get_Length(frame_type, pInBuf[0]);
     fread(&pInBuf[1], sizeof(HI_U8), packed_size, file_in);
     AMR_Decode_Frame(pDecState, pInBuf, pOutBuf, frame_type);
     fwrite (pOutBuf, sizeof (HI_S16), L_FRAME, file_out);
}

AMR_Decode_Exit(&pDecState);
```

[See Also]

- HI_S32 AMR_Decode_Init
- HI_S32 AMR_Get_Length
- HI_VOID AMR_Decode_Exit

3 Other Information

3.1 Data Types

Constant Definition

```
/*Frame length. It specifies the size (HI_S16) of the data that is input in
an encoder or of the data that is output by a decoder.*/
#define L_FRAME             160
/*The maximum size (HI_U8) of the data packed by an encoder or the maximum
size (HI_U8) of the data input into the decoder for decoding.*/
#define MAX_PACKED_SIZE     35
/*The file header when the MIME frame format is used for local storage*/
#define AMR_MAGIC_NUMBER   "#!AMR\n"
```

enum Mode

[Purpose]

This structure is used to specify the bit rates supported by the AMR-NB encoder/decoder.

[Definition]

```
enum Mode {
            MR475 = 0,   /*Available bit rate: 4.75 kbit/s          */
            MR515,       /*Available bit rate: 5.15 kbit/s          */
            MR59,        /*Available bit rate: 5.9 kbit/s           */
            MR67,        /*Available bit rate: 6.7 kbit/s           */
            MR74,        /*Available bit rate: 7.4 kbit/s           */
            MR795,       /*Available bit rate: 7.95 kbit/s          */
            MR102,       /*Available bit rate: 10.2 kbit/s          */
            MR122,       /*Available bit rate: 12.2 kbit/s          */
            MRDTX,       /*Mute mode, internal mode, not available. */
            N_MODES      /*Number of bit rates, not available.      */
            }
```

[Note]

Only the bit rates of MR475–MR122 are available for users to choose.

enum Format

[Purpose]

The frame format. The frame formats are as follows:

- MIME

  It is the real-time transport protocol (RTP) payload format. For details, see the *rfc4239 AMR and AMR-WB Storage Format.*
- IF1

  For details, see **3GPP 26101-600.doc**.
- IF2

  For details, see **3GPP 26101-600.doc**.

[Definition]

```
enum Format { MIME = 0, IF1, IF2 };
```

[Note]

If the MIME frame format is used, the file is saved as the **.amr**. The Storm player supports this format. At this time, there are no players that support the IF1 and IF2 formats.

3.2 Error Codes

Error Code	Description
HI_ERR_AMRNB_INVALID_DEVICE	Invalid encoder/decoder.
HI_ERR_AMRNB_INVALID_INBUF	Invalid input buffer.
HI_ERR_AMRNB_INVALID_OUTBUF	Invalid output buffer.
HI_ERR_AMRNB_MODE_TYPE	Invalid encode bit rate.
HI_ERR_AMRNB_FORMAT_TYPE	Invalid frame format.
HI_ERR_AMRNB_INIT_FAIL	Initializing the encoder/decoder failed.
HI_ERR_AMRNB_ENCODE_FAIL	Encoding the data failed.
HI_ERR_AMRNB_DECODE_FAIL	Decoding the stream failed.

A Acronyms and Abbreviations

Numerics	
3GPP	3rd Generation Partnership Project
A	
ACELP	Algebraic Code Excited Linear Prediction
AMR	Adaptive Multi-Rate
AMR-NB	Adaptive Multi-Rate Narrow-Band
C	
CNG	Comfort Noise Generation
CRC	Cyclic Redundancy Check
D	
DTX	Discontinuous Transmission
F	
FQI	Frame Quality Indication
I	
IF1	AMR Interface Format 1
IF2	AMR Interface Format 2
M	
MMS	MIME file storage format

P	
PCM	Pulse Code Modulation
S	
SID	Silence Descriptor
T	
TS	Transport Stream
V	
VAD	Voice Activity Detection