08-7222-API-ZCH66
27-OCT- 2006
4.0

Application Programmers Interface for WB-AMR Decoder and Encoder

ABSTRACT:

Application Programmers Interface for WB AMR Decoder and Encoder

KEYWORDS:

Multimedia codecs, speech, WB AMR, Wide Band AMR

APPROVED:

Shang Shidong

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	27-July-2004	Aniruddha Sinha	Initial draft
1.0	29-July-2004	Aniruddha Sinha	Review comments incorporated
1.1	20-Aug-2004	Aniruddha Sinha	Corrected mistake on sampling rate and the defines are prefixed with WBAMR
1.2	10-Nov-2004	Naganna	Added an additional variable to the encoder config structure
1.3	25-Jan-2005	Shashi	Added an additional variable to the decoder config structure (for supporting IF2/MIME/ITU formats)
1.4	03-Feb-2005	Ashok Kumar	Modified coding mode of encoder config structure
2.0	28-Apr-2005	Ashok Kumar	Updated performance table
3.0	06-Feb-2006	Lauren Post	Using new format
4.0	27-Oct-2008	Jackiea Pan	Update document

Table of Contents

Introduction

1.1 Purpose

This document gives the details of the application programmer's interface (API) of Wide Band Adaptive Multi-Rate (WB-AMR) codec. The WB-AMR encoder compresses linear PCM speech input data at 16 kHz sampling rate to one of nine data rate modes- 6.60, 8.85, 12.65, 14.25, 15.85, 18.25, 19.85, 23.05 and 23.85 kbps. WB-AMR coding scheme is based on the principle of Algebraic Code Excited Linear Prediction algorithm.

The WB-AMR algorithms also implements silence compression techniques to reduce the transmitted bit rate during the silent intervals of speech. Voice activity detector (VAD) and Comfort noise generation (CNG) algorithms are used to enable the transmission of silence descriptor (SID) frames during the periods of silence.

The WB-AMR codec is OS independent and do not assume any underlying drivers.

1.2 Scope

This document describes only the functional interface of the WB-AMR codec. It does not describe the internal design of the codec. Specifically, it describes only those functions which are required for this codec to be integrated in a system.

1.3 Audience Description

The reader is expected to have basic understanding of Speech Signal processing and WB-AMR vocoder. The intended audience for this document is the development community who wish to use the WB-AMR codec in their systems.

1.4 References

1.4.1 Standards

- **3GPP TS 26.190 V5.1.0 (2001-12)**: AMR Wideband Speech Codec; Transcoding functions (Release 5)
- **3GPP TS 26.191 V5.1.0 (2002-03)**: AMR Wideband Speech Codec; Error Concealment of Lost Frames (Release 5)
- **3GPP TS 26.192 V5.0.0 (2001-03)**: AMR Wideband Speech Codec; Comfort Noise Aspects (Release 5)
- **3GPP TS 26.193 V5.0.0 (2001-03)**: AMR Wideband Speech Codec; Source Controlled Rate operation (Release 5)
- **3GPP TS 26.194 V5.0.0 (2001-03)**: AMR Wideband Speech Codec; Voice Activity Detection (VAD) (Release 5)

- **3GPP TS 26.174 V5.4.0 (2002-12)**: AMR Wideband Speech Codec; Test sequences (Release 5)
- **3GPP TS 26.173 V5.8.0 (2003-09)**: AMR Wideband Speech Codec; ANSI-C code (Release 5)
- **3GPP TS 26.201 V5.0.0** (2001-03): AMR Wideband Speech Codec; Frame Structure.
- **ITU-T Recommendation G.711 (1988)** – Coding of analogue signals by pulse code modulation Pulse code modulation (PCM) of voice frequencies.

1.4.2 General references

- E. Paksoy, J.C.D Martin, Alan McCree, C.G.Gerlach, Anand Anandakumar, Wai-Ming Lai and Vishu Viswanathan, ICASS Proceeding 1999 "An Adaptive Multi-Rate Speech Coder for Digital Cellular Telephony"
- Real-Time Transport Protocol (RTP) Payload Format and File Storage Format for the Adaptive Multi-Rate (AMR) and Adaptive Multi-Rate Wideband (AMR-WB) Audio Codecs "RFC3267"

1.4.3 Freescale Multimedia References

- WB AMR Codec Application Programming Interface – wbamr_codec_api.doc
- WB AMR Codec Requirements Book – wbamr_codec_reqb.doc
- WB AMR Codec Test Plan - wbamr_codec_test_plan.doc
- WB AMR Codec Release notes - wbamr_codec_release_notes.doc
- WB AMR Codec Test Results – wbamr_codec_test_results.doc
- WB AMR Codec Performance Results – wbamr_codec_perf_results.doc
- WB ARM Interface Common Header – wbamr_common_interface.h
- WB ARM Interface Decoder Header – wbamr_dec_interface.h
- WB ARM Interface Encoder Header – wbamr_enc_interface.h
- WB ARM Decoder Application Code – wbamr_dectest.c
- WB ARM Encoder Application Code – wbamr_enctest.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
3GPP	3rd Generation Partnership Project
ACELP	Algebraic Code Excited Linear Prediction
API	Application Programming Interface
ARM	Advanced RISC Machine
CNG	Comfort Noise Generation
DTX	Discontinuous Transmission
ETSI	European Standard Telecommunications Series
FSL	Freescale

IF-1	Interface format 1
IF-2	Interface format 2
ITU	International Telecommunication Union
LSP	Line Spectral Pair
LP	Linear Prediction
MIME	Multimedia Storage Format
MIPS	Million Instructions per Second
OS	Operating System
PCM	Pulse Code Modulation
RVDS	ARM RealView Development Suite
SCR	Source Controlled Rate
SID	Silence Insertion Descriptor
RVDS	ARM RealView Development Suite
TBD	To Be Determined
UNIX	Linux PC x/86 C-reference binaries
WB-AMR	Wide Band Adaptive Multi-Rate Codec
VAD	Voice Activity Detection

1.6 Document Location

docs/wb_amr

2 API Description

This section describes the steps followed by the application to call the WB-AMR encoder and decoder. During each step the data structures and the functions used will be explained. Pseudo code is given at the end of each step.

2.1 Encoder API Description

The member variables inside the structure are prefixed as wbamre_ or wbappe_ to indicate if that member variable needs to be initialized by the encoder or application calling the encoder.

Step 1: Print version information

The application can get version information such as version number and build time by calling the below function.

C prototype:

```
const char *WBAMR_get_version_info(void);
```

Arguments:

- None.

Return Value:

- Returns a sting which includes version information

Step 2: Allocate memory for Encoder config parameter structure

The application allocates memory for below mentioned structure.

```
/* encoder parameter structure */
typedef struct
{
      WBAMRE_Mem_Alloc_Info    wbamre_mem_info;
      Void                    *wbamre_encode_info_struct_ptr;
      WBAMR_U8                *wbappe_initialized_data_start;
      WBAMR_S32               wbappe_dtx_flag;
      WBAMR_S16               *wbappe_mode;
      WBAMR_U16               *wbamre_output_size;
      WBAMR_U8                wbamre_output_format;
} WBAMRE_Encoder_Config;
```

Description of the encoder parameter structure *`WBAMRE_Encoder_Config`*

<u>*wbamre_mem_info*</u>

This is memory an information structure. The application needs to call the function wbamre_query_enc_mem to get the memory requirements from encoder. The encoder will fill this structure with its memory requirements. This will be discussed in **step 2**.

wbamre_encode_info_struct_ptr:

This is a void pointer. This will be initialized by the encoder during the initialization routine. This will then be a pointer to a structure which contains the pointers to tables, buffers and symbols used by the encoder.

wbappe_initialized_data_start

The application has to assign this pointer with the symbol supplied in the header file. This symbol is the start address of the initialized data that the encoder uses. The encoder needs to know this as the OS can relocate the data tables of the WB-AMR encoder every time the application is invoked.

wbappe_dtx_flag

The application shall set the value of this variable to 0 (disable DTX) or 1 (enable DTX). Encoder reads this value during its initialization.

wbappe_mode

This is pointer to requested mode value. Application needs to set this to valid coding mode value before calling encode frame function. Please refere Appendix **Error! Reference source not found.** for details of coding mode.

wbamre_output_size

This is pointer to the actual size of the encoded data in bytes. This information will be filled by the encoder.

Example pseudo code for this step:

```
/* allocate memory for the encoder parameter */
WBAMRE_Encoder_Config *enc_config;

/* allocate fast memory for encoder config */
enc_config = (WBAMRE_Encoder_Config *)
                         alloc_fast (sizeof(WBAMRE_Encoder_Config);

/* Fill up the relocated data position */
enc_config->wbappe_initialized_data_start = BEGIN_WBAMRE_DATA;

/* Enable DTX */
enc_config-> wbappe_dtx_flag = 1;

/* Set the requested mode */
config->wbappe_mode = 0;  /* (corresponds to 6.60 kbps) */
```

Step 3: Get the encoder memory requirements

The WB-AMR encoder does not do any dynamic memory allocation. The application calls the function *wbamre_query_enc_mem* to get the encoder memory requirements. This function must be called before any other encoder functions are invoked.

The function prototype of *wbamre_query_enc_mem* is:

C prototype:
```
WBAMRE_RET_TYPE wbamre_query_enc_mem (WBAMRE_Encoder_config *
enc_config);
```

Arguments:
- enc_config - Encoder config pointer.

Return value:

- WBAMRE_OK - Memory query successful.
- Other codes - Error (For other error codes refer to appendix).

This function populates the memory information structure, which is described below:

/* Memory information structure array */
```
typedef struct
{
      /* Number of valid memory requests */
      WBAMR_S32                         wbamre_num_reqs;
      WBAMRE_Mem_Alloc_Info_Sub mem_info_sub[WBAMR_MAX_NUM_MEM_REQS];
} WBAMRE_Mem_Alloc_Info;
```

Description of the structure *WBAMRE_Mem_Alloc_Info*

wbamre_num_reqs

The number of memory chunks requested by the encoder.

mem_info_sub

This structure contains each chunk's memory config parameters.

```
typedef struct
 {
      WBAMR_S32         wbamre_size;            /* Size in bytes */
      WBAMRMemType      wbamre_mem_type;
            /* Memory is static or scratch */
      WBAMR_S32         wbamre_type_fs;   /* Memory type Fast or Slow */
      WBAMR_U8          wbamre_priority;  /* Priority level */
      void              *wbappe_base_ptr;
            /* Pointer to the base memory, which will be
              allocated and  filled by the application*/
} WBAMRE_Mem_Alloc_Info_sub

enum {
      WBAMR_STATIC,
      WBAMR_SCRATCH,
} WBAMRMemType;
```

Description of the structure *WBAMRE_Mem_Alloc_Info_sub*

wbamre_size

The size of each chunk in bytes.

wbamre_mem_type

The type of the memory indicates if the requested chunk of memory is static or scratch. This is indicated by the enum *WBAMRMemType.*

wbamre_type_fs

The type of the memory indicates if the requested chunk of memory needs to be allocated in external or internal memory. The type of memory can be SLOW_MEMORY or external memory, FAST_MEMORY or internal memory. In targets where there is no internal memory, the application can allocate memory in external memory.
(Note: If the encoder requests for a FAST_MEMORY for which the application allocates a SLOW_MEMORY, the encoder will still encode, but the performance (MHz) will suffer.)

wbamre_priority

This indicates the priority level of the memory type, *wbamre_type_fs*. The type of memory can be SLOW_MEMORY or external memory, FAST_MEMORY or internal memory. In case the type of memory is FAST_MEMORY then the field *wbamre_priority* indicates the importance or the priority of the request. A priority value of zero indicates highest priority and 255 indicates lowest priority.

wbappe_base_ptr

This will be initialized by the application. The application will allocate the memory for each chunk depending on the requested size and the type, and then assign the base address of this chunk of memory to *wbappe_base_ptr.* The application should allocate the memory that is aligned to a 4 byte boundary in any case.

Example pseudo code for the memory information request

```
/* Query for memory */

retval = wbamre_query_enc_mem (enc_config);

if (retval != WBAMRE_OK)
      return FAILUE;
```

Step 4: Allocate Data Memory for the encoder

In this step the application allocates the memory as required by the WB-AMR encoder and fills up the base memory pointer *'wbappe_base_ptr' of 'WBAMRE_Mem_Alloc_Info_sub'* structure for each chunk of memory requested by the encoder. These allocated memory chunks should be aligned to 4 byte boundary.

Example pseudo code for the memory allocation and filling the base memory pointer by the application is given below.

```
WBAMRE_Mem_Alloc_Info_sub *mem;

/* Number of memory chunks requested by the encoder */
nr = enc_config->wbamre_mem_info.wbamre_num_reqs;

for(i = 0; i < nr; i++)
{
      mem = &(enc_config-> wbamre_mem_info.mem_info_sub[i]);
      if (mem->wbamre_type_fs == WBAMR_FAST_MEMORY)
```

```
        {
                /* This function allocates memory in internal memory */
                mem->wbappe_base_ptr = alloc_fast(mem->wbamre_size);
        }
        else
        {
                This function allocates memory in external memory */
                mem->wbappe_base_ptr = alloc_slow(mem->wbamre_size);
        }
}
```

The functions alloc_fast and alloc_slow are required to allocate the memory aligned to 4 byte boundary.

Step 5: Initialization routine

All initializations required for the encoder are done in wbamre*_encode_init*. This function must be called before the main encode frame function is called.

C prototype:
`WBAMRE_RET_TYPE wbamre_encode_init (WBAMRE_Encoder_Config *);`

Arguments:

- Pointer to encoder configuration structure

Return value:

• WBAMRE_OK	- Initialization successful.
• Other codes	- Initialization Error

Example pseudo code for calling the initialization routine of the decoder

```
enc_config->dtx_flag = 1; /* enable DTX mode */

/* Set the requested mode */
config->wbappe_mode = 0;

/* Initialize the WB-AMR encoder. */
retval = wbamre_encode_init (enc_config);

if (retval != WBAMRE_OK)
        return WBAMR_FAILURE;
```

Step 6: Memory allocation for input buffer

The application has to allocate the memory needed for the input buffer. It is desirable to have the input buffer allocated in FAST_MEMORY, as this may improve the performance (MHz) of the encoder. The size of input buffer should be equal to speech frame size i.e. 320 words. Pointer to the input buffer needs to be passed to encode frame routine.

Example pseudo code for allocating the input buffer

```
/* allocate memory for input buffer */
in_buf = alloc_fast(WBAMR_L_FRAME * sizeof (WBAMR_S16));
```

<u>Special Consideration</u>

14-bit uniform PCM input sample should be left aligned in 16 bit word boundary with remaining 2 bits set to zero as shown below.

MSB	LSB	zeroes

Step 7: Memory allocation for output buffer

The application has to allocate memory for the output buffers to hold the encoded bit stream corresponding to one frame of speech sample at maximum supported bitrate, i.e. 23.85 kbps. The pointer to this output buffer needs to be passed to the wbamre_encode_frame function. The application can allocate memory for output buffer in external memory using alloc_slow. Allocating memory in internal memory using alloc_fast will improve the performance (MHz) of the encoder marginally.

<u>Example pseudo code for allocating memory for output buffer</u>

```
/* allocate memory for output buffer */

/* Allocate WBAMR_SERIAL_FRAMESIZE=480 Words */
out_buf = alloc_slow(WBAMR_SERIAL_FRAMESIZE * sizeof (WBAMR_S16));
```

<u>Special Consideration</u>

Encoder output will be in one bit per byte format and that bit will be placed at least significant bit position as shown below.

LSB	Encoded output

Step 8: Call the frame encode routine

The main WB-AMR encoder function is wbamre_*encode_frame*. This function encodes the 14-bit uniform PCM sample and writes bitstream to output buffer.

C prototype:

```
WBAMRE_RET_TYPE         wbamre_encode_frame (
                              WBAMRE_Encoder_Config *enc_config,
                              WBAMR_S16 *in_buf,
                              WBAMR_S16 *out_buf);
```

Arguments:

- enc_config - Pointer to encoder config structure

• in_buf	- Pointer to input speech buffer
• out_buf	- Pointer to output (encoded) buffer

Return value:

• WBAMRE_OK	- Indicates encoding was successful.
• **Others**	**- Indicates error**

Example pseudo codes for calling the main encode routine of the encoder.

```
while (WBAMR_TRUE)
{
      enc_config->wbappe_mode = 0;

      /* Please note that homing frame detection and required action
         will be taken inside the encode frame function */
      retval=wbamre_encode_frame(enc_config, in_buf, out_buf);
      if (retval  != WBAMRE_OK)
            return WBAMR_FAILURE;
}
```

Step 9: Free memory

The application should release all the memory it allocated before exiting the encoder.

```
free (out_buf);
free (in_buf);
for (i=0; i<nr; i++)
{
      free (enc_config->wbamre_mem_info.mem_info_sub[i].wbappe_base_ptr);
}
free (enc_config);
```

2.2 Decoder API Description

The member variables inside the structure are prefixed as wbamrd_ or wbappd_ to indicate if that member variable needs to be initialized by the decoder or application calling the decoder.

Step 1: Print version information

The application can get version information such as version number and build time by calling the below function.

C prototype:

```
const char *WBAMR_get_version_info(void);
```

Arguments:

- None.

Return Value:

- Returns a sting which includes version information

Step 2: Allocate memory for Decoder config parameter structure

The application allocates memory for below mentioned structure.

/* decoder parameter structure */
```
typedef struct
 {
      WBAMRD_Mem_Alloc_Info        wbamrd_mem_info;
      Void                        *wbamrd_decode_info_struct_ptr;
      WBAMR_U8                    *wbappd_initialized_data_start;
      WBAMR_U8                     bitstreamformat;
} WBAMRD_Decoder_Config;
```

Description of the decoder parameter structure *`WBAMRD_Decoder_Config`*

wbamrd_mem_info

This is a memory information structure. The application needs to call the function wbamrd_query_dec_mem to get the memory requirements from decoder. The decoder will fill this structure with its memory requirements. This will be discussed in **step 2**.

wbamrd_decode_info_struct_ptr

This is a void pointer. This will be initialized by the decoder during the initialization routine. This will then be a pointer to a structure which contains the pointers to tables, buffers and symbols used by the decoder.

wbappd_initialized_data_start

The application has to assign this pointer with the symbol supplied in the header file. This symbol is the start address of the initialized data that the decoder uses. The decoder needs to know this as the OS can relocate the data tables of the WB-AMR decoder every time the application is invoked.

bitstreamformat

This is added to the Decoder Config to provide IF2/MIME/ITU format support by the decoder library.

Example pseudo code for this step:

```
/* allocate memory for the decoder parameter */
   WBAMRD_Decoder_Config *dec_config;
dec_config = (WBAMRD_Decoder_Config *)
                              alloc (sizeof(WBAMRD_Decoder_Config);
/* Fill up the Relocated data position */
dec_config->wbappd_initialized_data_start = BEGIN_WBAMRD_DATA;
dec_Config->bistreamformat = bitstreamformat;
      /* Here bit stream comes as the input argument for the decoder.
      Default (ETSI format) is 0, ITU is 1, MIME is 2, and IF2 is 3. */
```

Step 3: Get the decoder memory requirements

The WB-AMR decoder does not do any dynamic memory allocation. The application calls the function *wbamrd_query_dec_mem* to get the decoder memory requirements. This function must be called before any other decoder functions are invoked.

The function prototype of *wbamrd_query_dec_mem* is:

C prototype:
```
WBAMRD_RET_TYPE wbamrd_query_dec_mem (
                    WBAMRD_Decoder_config * dec_config);
```

Arguments:

• dec_config	- Decoder configuration pointer.

Return value:

• WBAMR_OK	- Memory query successful.
• Other codes	- Error (For other error codes refer to appendix).

This function populates the memory information structure, which is described below:

Memory information structure array
```
typedef struct
{
      /* Number of valid memory requests */
      WBAMR_S32                        wbamrd_num_reqs;
      WBAMRD_Mem_Alloc_Info_Sub mem_info_sub[WBAMR_MAX_NUM_MEM_REQS];
} WBAMRD_Mem_Alloc_Info;
```

Description of the structure *WBAMRD_Mem_Alloc_Info*

wbamrd_num_reqs

The number of memory chunks requested by the decoder.

mem_info_sub

This structure contains each chunk's memory configuration parameters.

```
typedef struct
{
      WBAMR_S32          wbamrd_size;          /* Size in bytes */
      WBAMRMemType       wbamrd_mem_type;  /* Memory is static or scratch */
      WBAMR_S32          wbamrd_type_fs;       /* Memory type Fast or Slow */
      WBAMR_U8           wbamrd_priority;    /* Priority level */
      void               *wbappd_base_ptr;
                   /* Pointer to the base memory, which will be
                             allocated and filled by the application*/
} WBAMRD_Mem_Alloc_Info_sub
```

```
enum {
      WBAMR_STATIC,
      WBAMR_SCRATCH,
} WBAMRMemType;
```

Description of the structure *WBAMRD_Mem_Alloc_Info_sub*

wbamrd_size

The size of each chunk in bytes.

wbamrd_mem_type

The type of the memory indicates if the requested chunk of memory is static or scratch. This is indicated by the enum *WBAMRMemType.*

wbamrd_type_fs

The type of the memory indicates if the requested chunk of memory needs to be allocated in external or internal memory. The type of memory can be SLOW_MEMORY or external memory, FAST_MEMORY or internal memory. In targets where there is no internal memory, the application can allocate memory in external memory.
(Note: If the decoder requests for a FAST_MEMORY for which the application allocates a SLOW_MEMORY, the decoder will still decode, but the performance (MHz) will suffer.)

wbamrd_priority

This indicates the priority level of the memory type, *wbamrd_type_fs*. The type of memory can be SLOW_MEMORY or external memory, FAST_MEMORY or internal memory. In case the type of memory is FAST_MEMORY then the field *wbamrd_priority* indicates the importance or the priority of the request. A priority value of zero indicates highest priority and 255 indicates lowest priority.

wbappd_base_ptr

This will be initialized by the application. The application will allocate the memory for each chunk depending on the requested size and the type, and then assign the base address of this chunk of memory to *wbappd_base_ptr.* The application should allocate the memory that is aligned to a 4 byte boundary in any case.

Example pseudo code for the memory information request

```
/* Query for memory */
retval = wbamrd_query_dec_mem (dec_config);

if (retval != WBAMRD_OK)
      return WBAMR_FAILURE;
```

Step 4: Allocate Data Memory for the decoder

In this step the application allocates the memory as required by the WB-AMR decoder and fills up the base memory pointer *'wbappd_base_ptr' of 'WBAMRD_Mem_Alloc_Info_sub'* structure for each chunk of memory requested by the decoder. These allocated memory chunks should be aligned to 4 byte boundary

Example pseudo code for the memory allocation and filling the base memory pointer by the application is given below.

```
WBAMRD_Mem_Alloc_Info_sub *mem;

/* Number of memory chunks requested by the encoder */
nr = dec_config->wbamrd_mem_info.wbamrd_num_reqs;
```

```
for(i = 0; i < nr; i++)
{
      mem = &(dec_config-> wbamrd_mem_info.mem_info_sub[i]);
      if (mem->wbamrd_type_fs == WBAMR_FAST_MEMORY)
      {
            /* This function allocates memory in internal memory */
            mem->wbappd_base_ptr = alloc_fast(mem->wbamrd_size);
      }
      else
      {
            This function allocates memory in external memory */
            mem->wbappd_base_ptr = alloc_slow(mem->wbamrd_size);
      }
}
```

The functions alloc_fast and alloc_slow are required to allocate the memory aligned to 4 byte boundary.

Step 5: Initialization routine

All initializations required for the decoder are done in *wbamrd_decode_init*. This function must be called before the main decode frame function is called.

C prototype:

```
WBAMRD_RET_TYPE wbamrd_decode_init (WBAMRD_Decoder_Config *);
```

Arguments:

- Pointer to decoder configuration structure

Return value:

• WBAMRD_OK	- Initialization successful.
• Other codes	- Initialization Error

Example pseudo code for calling the initialization routine of the decoder

```
/* Initialize the WB-AMR decoder. */
retval = wbamrd_decode_init (dec_config);
if (retval != WBAMRD_OK)
      return WBAMR_FAILURE;
```

Step 6: Memory allocation for input buffer

The application has to allocate the memory needed for the input buffer. It is desirable to have the input buffer allocated in FAST_MEMORY, as this may improve the performance (MHz) of the decoder. Pointer to the input buffer needs to be passed to decode frame routine.

Example pseudo code for allocating the input buffer

```
/* allocate SERIAL_FRAME_SIZE = 480 words */
in_buf = alloc_fast (WBAMR_SERIAL_FRAMESIZE * sizeof (WBAMR_S16));
```

Step 7: Memory allocation for output buffer

The application has to allocate memory for the output buffers to hold the decoded PCM sample corresponding to one speech frame. The pointer to this output buffer needs to be passed to the wbamrd_decode_frame function. The application can allocate memory for output buffer in external memory using alloc_slow. Allocating memory in internal memory using alloc_fast will improve the performance (MHz) of the decoder marginally. It would be desirable to allocate the buffer in the slow memory.

Example pseudo code for allocating memory for output buffer

```
/* allocate memory for output buffer,
 WBAMR_L_FRAME=320 as defined in Appendix */
out_buf = alloc_slow(WBAMR_L_FRAME * sizeof (WBAMR_S16));
```

Special Consideration

Output of decoder is 14-bit uniform PCM sample left aligned in 16 bit word boundary with remaining 2 least significant bits set to zero as shown below.

MSB	LSB	zeroes

Step 8: Call the frame decode routine

The main WB-AMR decoder function is wbamrd_*decode_frame*. This function decodes the WB-AMR bitstream and writes bitstream to PCM play out buffer.

C prototype:

```
WBAMRD_RET_TYPE        wbamrd_decode_frame (
                              WBAMRD_Decoder_Config *dec_config,
                              WBAMR_S16 *in_buf,
                              WBAMR_S16 *out_buf);
```

Arguments:

• dec_config	- Pointer to decoder configuration structure
• in_buf	- Pointer to WB-AMR bitstream buffer
• out_buf	- Pointer to output (decoded) buffer

Return value:

• WBAMRD_OK	- Indicates decoding was successful.
• **Others**	**- Indicates error**

Example pseudo codes for calling the main decode routine of the decoder.

```
while (WBAMR_TRUE)
```

```
{
      /* Please note that homing frame detection and required action
        will be taken inside the decode frame function */
      retval=wbamrd_decode_frame (dec_config, in_buf, out_buf);
      if (retval  != WBAMRD_OK)
            return WBAMR_FAILURE;
}
```

Step 9: Free memory

The application should release memory before exiting the decoder.

```
free (out_buf);
free (in_buf);
for (i=0; i<nr; i++)
{
      free(dec_config->wbamrd_mem_info.mem_info_sub[i]. \
                                    wbappd_base_ptr);
}
free (dec_config);
```

3 Example calling Routine

3.1 Example calling routine for WB-AMR Encoder

Below example code gives a guideline for calling WB-AMR encoder.

```
/*******************************************************************
 *                         INCLUDE FILES
 *******************************************************************/

#include "wbamr_enc_interface.h"
#include "wbamr_common_interface.h"


/*******************************************************************
 *        MAIN PROGRAM
 *******************************************************************/

int main (void)
{
      /* Pointer to new speech data*/
      WBAMR_S16 *in_buf;
      /* Output bitstream buffer */
      WBAMR_S16 *out_buf;
      WBAMR_S32 i;
      WBAMRE_Mem_Alloc_Info_sub *mem;

      WBAMRE_Encoder_config *enc_config;

      /* Get the version info of the WB AMR */
      fprintf(stderr,"Running %s\n", WBAMR_get_version_info());

      /* allocate memory for encoder configuration structure */
      enc_config=(WBAMRE_Encoder_config *)
                       alloc_fast(sizeof(WBAMRE_Encoder_config));

      /* Fill up the Relocated data position */
      enc_config->wbappe_initialized_data_start = BEGIN_WBAMRE_DATA;
      /*
       * Symbol BEGIN_WBAMRE_DATA is defined in wbamr_enc_interface.h
       */

      enc_config->wbamre_encode_info_struct_ptr =NULL;
      /* Initialize config structure memory to NULL;
      for (i=0; i<WBAMR_MAX_NUM_MEM_REQS; i++)
      {
            enc_config->wbamre_mem_info.mem_info_sub[i].\
                                          wbappe_base_ptr =NULL;
      }
      /* Query for memory */
      retval = wbamre_query_enc_mem (enc_config);
      if (retval != WBAMRE_OK)
```

```
        {
              /* deallocate memory allocated for encoder config */
              free(enc_config);
              return WBAMR_FAILURE;
        }
        /* Number of memory chunk requests by the encoder */
        nr = enc_config->wbamre_mem_info.wbamre_num_reqs;

        /* allocate memory requested by the encoder*/
        for(i = 0; i < nr; i++)
        {
              mem = &(enc_config->wbamre_mem_info.mem_info_sub[i]);
              if (mem->wbamre_type_fs == WBAMR_FAST_MEMORY)
              {
                    mem->wbappe_base_ptr = alloc_fast(mem->wbamre_size);
              }
              else
              {
                    mem->wbappe_base_ptr = alloc_slow(mem->wbamre_size);
              }
        }
        /* enable DTX mode here */
        enc_config->dtx_flag = 1;
        /* initialize encoder */
        retval = wbamre_encode_init (enc_config);
        if (retval =! WBAMRE_OK)
        {
              /* free all the memory allocated for encoder
              config param */
              for (i=0; i<nr; i++)
              {
                    free (enc_config->wbamre_mem_info.\
                          mem_info_sub[i].wbappe_base_ptr);
              }
              free (enc_config);
              return WBAMR_FAILURE;
        }
        /* allocate memory for input buffer */
        if ((in_buf = alloc_fast(WBAMR_L_FRAME * sizeof (WBAMR_S16))) ==
NULL)
        {
              /*free all the memory allocated for encoder config param*/
              for (i=0; i<nr; i++)
              {
                    free (enc_config->wbamre_mem_info.\
                          mem_info_sub[i].wbappe_base_ptr);
              }
              free (enc_config);
              return WBAMR_FAILURE;
        }

        out_buf = alloc_fast (WBAMR_SERIAL_FRAMESIZE * sizeof (WBAMR_S16));

        /* allocate memory for output buffer (unpacked) */
```

```
if (out_buf == NULL)
{
      /* free all the memory allocated for encoder config param
      */
      for (i=0; i<nr; i++)
      {
            free (enc_config->wbamre_mem_info.\
                  mem_info_sub[i].wbappe_base_ptr);
      }
      free (enc_config);

      free(in_buf);
      return WBAMR_FAILURE;
};

/***************************************************************
 * Encode speech frame by frame till end of frame
 ***************************************************************/
while (WBAMR_TRUE)
{
      enc_config->wbappe_mode = 0;

      Get PCM sample in input buffer;
      /* increment frame number */

      /* zero flags and parameter bits */
      for (i = 0; i < WBAMR_SERIAL_FRAMESIZE; i++)
            out_buf[i] = 0;

      /* encode speech */
      retval=wbamre_encode_frame (enc_config, in_buf, out_buf);
      if (retval != WBAMRE_OK)
      {
            free (out_buf);
            free (in_buf);
            for (i=0; i<nr; i++)
            {
                  free (enc_config->wbamre_mem_info.\
                              mem_info_sub[i].wbappe_base_ptr);
            }
            free (enc_config);
            exit (-1);
      }
}
/***************************************************************
 *Closedown speech coder
 **************************************************************/
free (out_buf);
free (in_buf);
for (i=0; i<nr; i++)
{
      free (enc_config-> wbamre_mem_info.\
                  mem_info_sub[i].wbappe_base_ptr);
}
```

```
        free (enc_config);
        return WBAMR_SUCCESS;
}
```

3.2 Example calling routine for WB-AMR Decoder

Example calling guidelines for calling the WB-AMR decoder is given below.

```
/***********************************************************************
* Include Files
***********************************************************************/
#include "wbamr_dec_interface.h"
#include "wbamr_common_interface.h"

/***********************************************************************
* Main Program
***********************************************************************/
int main (void)
{
      WBAMR_S32 i;

      WBAMRD_Mem_Alloc_Info_sub *mem;

      WBAMRD_Decoder_config *dec_config;
      WBAMR_S16 *in_buf; /* bitstream buffer */

      WBAMR_S16 *out_buf; /* decoded PCM sample buffer */

      /* Get the version info of the WB AMR */
      fprintf(stderr,"Running %s\n", WBAMR_get_version_info());

      /* allocate memory for decoder configuration structure */
      dec_config=(WBAMRD_Decoder_config *)
                        alloc_fast(sizeof(WBAMRD_Decoder_config));

      /* Fill up the relocated data position */
      dec_config->wbappd_initialized_data_start = BEGIN_WBAMRD_DATA;

      /* Symbol BEGIN_WBAMRD_DATA is defined in wbamr_dec_interface.h */


      dec_config->wbamrd_decode_info_struct_ptr =NULL;

      /* Initialize config structure memory to NULL;
      for (i=0; i<WBAMR_MAX_NUM_MEM_REQS; i++)
      {
            dec_config->wbamrd_mem_info.mem_info_sub[i].\
                                          wbappd_base_ptr =NULL;
      }


      /* Query decoder for its memory requirement */
      retval = wbamrd_query_dec_mem (dec_config);
      if (retval != WBAMRD_OK)
      {
            /* de-allocate memory allocated to enc_config */
```

```
        free(enc_config);
        return WBAMR_FAILURE;
    }

    /* Number of memory chunk requested by the decoder */
    nr = dec_config->wbamrd_mem_info.wbamrd_num_reqs;

    /* allocate memory requested by the decoder*/
    for(i = 0; i < nr; i++)
    {
        mem = &(dec_config->wbamrd_mem_info.mem_info_sub[i]);
        if (mem->wbamrd_type_fs == WBAMR_FAST_MEMORY)
        {
            mem->wbappd_base_ptr = alloc_fast(mem->wbamrd_size);
        }
        else
        {
            mem->wbappd_base_ptr = alloc_slow(mem->wbamrd_size);
        }
    }

    /* initialize decoder */
    retval = wbamrd_decode_init (dec_config);
    if (retval =! WBAMRD_OK)
    {
        for (i=0; i<nr; i++)
        {
            free (dec_config-> wbamrd_mem_info.\
                                    mem_info_sub[i].wbappd_base_ptr);
        }
        free (dec_config);
        return WBAMR_FAILURE;
    }

    /* allocate memory for input buffer */
    in_buf = alloc_fast (WBAMR_SERIAL_FRAMESIZE *sizeof (WBAMR_S16));

    if (in_buf == NULL)
    {
        /* free all the memory allocated for decoder config param
        */
        for (i=0; i<nr; i++)
        {
            free (dec_config->wbamrd_mem_info.\
                mem_info_sub[i].wbappd_base_ptr);
        }
        free (dec_config);
        return WBAMR_FAILURE;
    };

    /* allocate memory for output buffer */
    out_buf = alloc_fast (WBAMR_L_FRAME * sizeof (WBAMR_S16));
    if (out_buf == NULL)
    {
            /* free all the memory allocated for decoder config param
            */
            for (i=0; i<nr; i++)
            {
                  free (dec_config->wbamrd_mem_info.\
                        mem_info_sub[i].wbappd_base_ptr);
            }
            free (dec_config);
            free(in_buf);
            return WBAMR_FAILURE;
      };

      /**************************************************************
       * Decode bitstream to generate one frame of speech till end
       * of bitstream
       **************************************************************/
      while(WBAMR_TRUE)
      {
            /* decode speech */
            retval=wbamrd_decode_frame (dec_config, in_buf,out_buf);
            if (retval != WBAMRD_OK)
            {
                  free(in_buf);
                  free(out_buf);
                  for (i=0; i<nr; i++)
                  {
                        free (dec_config->wbamrd_mem_info.\
                                    mem_info_sub[i].wbappe_base_ptr);
                  }
                  exit (-1);
            }
      }
      /**************************************************************
       * Closedown speech coder
       **************************************************************/
      free (out_buf);
      free (in_buf);
      for (i=0; i<nr; i++)
      {
        free (dec_config->wbamrd_mem_info.mem_info_sub[i].\
                                                wbappe_base_ptr);
      }
      free (dec_config);
      return WBAMR_SUCCESS;
}
```