07-5449-API-ZIN05 1.2
23/05/2008
1.2

Application Programmers Interface for SBC Encoder

ABSTRACT:

Application Programmers Interface for SBC Encoder

KEYWORDS:

Multimedia codecs, SBC

APPROVED:

Shang Shidong

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
1.0	07-June-2007	Milash James	Initial Version
1.1	07-April-2008	Tao Jun	Update doc ID
1.2	23-May-08	Tao Jun	Added SBCEncVersionInfo API

Table of Contents

1 Introduction

1.1 Purpose

This document gives the details of the application programmer's interface of SBC Encoder. The SBC Encoder implementation currently supports mono, dual channel and stereo encoding. It does not support dual channel stereo encoding.

1.2 Scope

This document describes only the functional interface of the SBC Encoder. It does not describe the internal design of the encoder. Specifically, it describes only those functions by which a software module can use the encoder.

1.3 Audience Description

The intended audience for this document is the development community who wish to use the SBC encoder in their systems. The reader is expected to have basic understanding of Audio Signal processing and SBC encoding.

1.4 References

1.4.1 Standards

- A2DP specification for Bluetooth, version 1.2

1.4.2 Freescale Multimedia References

- SBC Encoder Application Programming Interface – sbc_enc_api.doc
- SBC Encoder Requirements Book - sbc_enc_reqb.doc
- SBC Encoder Test Plan - sbc_enc_test_plan.doc
- SBC Encoder Release notes - sbc_enc_release_notes.doc
- SBC Encoder Test Results – sbc_enc_test_results.doc
- SBC Encoder Performance Results – sbc_enc_perf_results.doc
- SBC Encoder Interface header – sbc_api.h
- SBC Encoder Application Code – sbc_enc_test_handler.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
SBC	Sub Band Codec
API	Application Programming Interface
ARM	Advanced RISC Machine
FSL	Freescale
IEC	International Electro-technical Commission
ISO	International Standards Organization
OS	Operating System
RVDS	ARM Real View Development Suite
TBD	To be decided

1.6 Document Location

docs/sbc_enc

2 API Description

This section describes the steps followed by the application to call the SBC encoder. During each step the data structures used and the functions used will be explained. Pseudo code is given at the end of each step.
The SBC encoder API currently support PUSH mode. In the PUSH mode, the encoder will not call swap functions to read in input data as the original PULL mode does. It is the application's duty to supply correct data for the encoder to encode.

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 *SBCEncVersionInfo(void);
```

Arguments:

- None.

Return Value:

- Returns a sting which includes version information

Step 2: Allocate memory for encoder parameter structure

The application allocates memory for the structure mentioned below. This structure contains the decoder parameters and memory information structures.

```
/*Encoder Parameter Structure*/
typedef struct tSbcEncConfig {
      uint8 sbc_enc_status;
      uint8 control_word;
      void *sbc_info;
      void *x;
      sbc_mem_info mem_info;
      struct sbc_data_struct *sbc_data;

}SbcEncConfig;
```

Description of the encoder parameter structure *SbcEncConfig*

sbc_enc_status

If the main encode call fails, this is set to FAIL. This is set by the encoder.

control_word

The control word determines whether the codec is active or not. The SBC_ENABLE flag in the codec is used to activate the codec. The init call of the codec will enable this flag. This is used internally by the codec, and should not be modified by the application.

sbc_mem_info

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

x

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 a table used by the encoder. This is a variable used internally by the codec, and application should not modify this.

sbc_info

This is a void pointer. It will be initialized by the codec during the initialization routine. This will then point to an internal structure that contains information about the input bit stream and application should not modify this.

sbc_data

Pointer to sbc_data_struct structure, which contains the input stream information.

Step 3: Get the encoder memory requirements

The SBC encoder does not do any dynamic memory allocation. The application calls the function *sbc_enc_query_mem* to get the encoder memory requirements. This function must be called before all other encoder functions are invoked.

The function prototype of *sbc_enc_query_mem* is :

C prototype:

```
SBC_RET_TYPE sbc_enc_query_mem (SbcEncConfig * sbc_enc_config);
```

Arguments:

- sbc_enc_config - Encoder config pointer.

Return Value:

- Returns SBC_OK on successful execution of the function.
- Returns SBC_FAIL on failure.

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

Memory information structure array

```
typedef struct sbc_mem_info{
      int nNumMemReqs;
      sbc_mem_allocinfo_subtype sMemInfoSub[SBC_MAX_NUM_MEM_REQS];

}sbc_mem_info;
```

Description of the structure *sbc_mem_info*

nNumMemReqs

The number of memory chunks requested by the encoder.

sMemInfoSub

This structure contains each chunk's memory configuration parameters.

```
typedef struct
{
      int    nSBCSize;    /* Size in bytes */
      int  nSBCMemType;   /* Memory is STATIC or SCRATCH */
      int  nSBCMemTypeFs; /* Memory type FAST or SLOW */
      void *pSBCBasePtr;
      char cSBCMemPriority; /* Priority level */

} sbc_mem_allocinfo_subtype;
```

Description of the structure *sbc_mem_allocinfo_subtype*

nSBCSize

The size of each chunk in bytes.

nSBCMemTypeFs

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.)

nSBCMemType

The memory description field indicates whether requested chunk of memory is static or scratch.

cSBCMemPriority

In case, if the encoder requests for multiple memory chunks in the fast memory, the priority indicates the order in which the application has to prioritize placing the requested chunks in Fast memory

pSBCBasePtr

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 *pSBCBasePtr.* 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 */
if(sbc_enc_query_mem (&sbc_enc_config)!=SBC_OK)
{
      // query memory failed
      exit(2);
}
```


Step 4: Allocate Memory for the encoder

In this step the application allocates the memory as required by the SBC encoder and fills up the base memory pointer *'pSbcBasePtr' of* '*sbc_mem_allocinfo_subtype*' structure for each chunk of memory requested by the encoder.

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

```
sbc_mem_allocinfo_subtype *psMem;

/* allocate memory requested by the encoder*/
nNumReqs = sbc_enc_config.mem_info.nNumMemReqs;
for(nCounter = 0;nCounter<nNumReqs;nCounter++)
{
        psMem = &( sbc_enc_config.mem_info.sMemInfoSub[nCounter]);
   psMem->pSBCBasePtr = malloc(psMem->nSBCSize);
}
```

Step 5: Fill the input stream information

The application updates the data structure which holds information about the input data. The sbc_input_struct and sbc_output_struct will have pointers to their respective sbc_data_struct structures.

```
struct sbc_data_struct
{
      uint8 nrof_subbands;
      uint8 sampling_freq;
      uint8 nrof_blocks;
      uint8 channel_mode;
      uint8 allocation_method;
      uint8 bitpool;
      uint32 bitrate;
      uint32 super_frame_size;
      uint32 frame_size;
};
```

Description of sbc_data_struct

nrof_subbands

This is filled with the desired number of sub-bands for the encoded output stream. This can be 4 or 8.

sampling_freq

This refers to the sampling frequency to be used for encoding. It can be 16000, 32000, 41000 or 48000Hz.

nrof_blocks

Number of blocks to be used in encoding.

channel_mode

Refers to the mode of encoding. It can be 0(mono),1(dual channel) or 2(stereo).

allocation_method

This value will be set as 0 if psychoacoustics option is enabled and 1 if psychoacoustics is disabled. Default value is 1.

bitpool

This is the number that determines the bitrate for encoding. Can be a value between 2 to 250.

bitrate

Bitrate in bps. bitrate and bitpool are mutually exclusive options.

super_frame_size

Size of input data super frame. This can contain more than one frame.

frame_size

Size of input data frame. If super_frame_size is not specified, super_frame_size will be set to frame size.

```
struct sbc_enc_input_struct{
    void  *input_frame_buf;
    };
```

Description of sbc_enc_input_struct

input_frame_buf

pointer to one raw data frame, whose maximum size is defined as 4096 bytes.

```
struct sbc_output_struct{
    struct sbc_data_struct *sbc_data;
    void        *out_buffer;
    uint16      size;
    uint8       debug_info;
};
```

Description of sbc_output_struct

This structure is updated by the encoder, during each encode call.

sbc_data

Pointer to sbc_data_struct

out_buffer

pointer to buffer containing output data, whose maximum size is defined as 1012 bytes.

size

actual size of encoded data.

debug_info

If an error occurs during encode, the error code is updated in ‘status’ variable of the internal SBCInfo structure, and the same error code is copied to debug_info.

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 maximum size of the buffer has been defined as 4096 bytes.

Step 7: Initialization routine

All initializations required for the encoder are done in *sbc_enc_init*. This function must be called before the main encoder function is called.

C prototype:
*SBC_RET_TYPE sbc_enc_init(SbcEncConfig *sbc_enc_config);*
Arguments:

- Encoder parameter structure pointer.

Return Value:

- On success, returns SBC_OK
- On failure, returns corresponding error code.

Example pseudo code for calling the initialization routine of the decoder

```
/* Initialize the SBC encoder. */
 if(sbc_enc_init (sbc_enc_config)!=SBC_OK)
{
      printf("Init failed ");
      exit(2);
}
```

Step 8: Memory allocation for output buffer

The application has to allocate memory for the output buffers to hold the encoded data. The pointer to this buffer has to be passed to the main encode function. The maximum size of output buffer has been defined as maximum encoded frame length of 1012 bytes.

Step 9: Call the encode routine

The main SBC encoder function is *sbc_encoder_encode*. This function encodes the input buffer to generate one frame of encoder output in every call.

C prototype:

```
SBC_RET_TYPE sbc_encoder_encode(SbcEncConfig *sbc_enc_config struct,
                            sbc_enc_input_struct  *sbc_input,
                             struct sbc_output_struct *sbc_output,
```

```
                                    );
```

Arguments:

- sbc_enc_config pointer to encoder config structure
- sbc_enc_input_struct pointer to input structure
- sbc_output_struct pointer to output structure

Return value:

- Returns SBC_OK on successful execution, and appropriate error code on failure.

Example pseudo code for calling the main encode routine of the encoder

```
/* call the encoder N times ( N = super_frame_size/(nrof_blocks*nrof_channels*nrof_subbands)*/
for(i=0;i<N;i++)
{
if(sbc_encoder_encode(&sbc_enc_config ,&sbc_enc_input,&sbc_enc_output)!=SBC_OK)
      {
      printf("Encode failed");
      exit(2);
      }


/* advance the input pointer to the next frame in a super_frame */
      sbc_enc_input.input_frame_buf = ((uint32)
sbc_enc_input.input_frame_buf + 2 *sbc_enc_input.sbc_data->frame_size);

if(sbc_enc_config.sbc_enc_status != SBC_FAIL)
{
/* Output the encoded data */
fwrite(sbc_enc_output.out_buffer, 1, sbc_enc_output.size ,<file descriptor>);
}
else
{
      sbc_enc_config.control_word = SBC_DISABLE;
      /* last frame reached */
      break;
}

}
```

Step 10: Free memory

The application releases the memory that it allocated to SBC Encoder if it no longer needs the encoder instance.

```
for(nCounter = 0;nCounter<nNumReqs;nCounter++)
      free(sbc_enc_config.mem_info.sMemInfoSub[nCounter].pSBCBasePtr);

free(input buffer);
free(output buffer);
```

3 Example calling Routine

```
/* pseudo code to illustrate sequence of operations*/
SbcEncConfig sbc_enc_config;
int nNumReqs;
int nCounter;
sbc_mem_allocinfo_subtype *psMem;
struct sbc_enc_input_struct sbc_enc_input;
struct sbc_output_struct sbc_enc_output;
uint8 * in_buf;
uint8 * out_buf;
SBC_RET_TYPE eRetVal;

/* Get the version info of the sign-on SBC ENC */
fprintf(stderr,"Running %s\n",SBCEncVersionInfo());

/*query memory requirements of the codec */
eRetVal = sbc_enc_query_mem(&sbc_enc_config);
if(eRetVal !=SBC_OK)
      exit(2);

/* allocate memory requested by the codec */
nNumReqs = sbc_enc_config.mem_info.nNumMemReqs;
for(nCounter = 0;nCounter<nNumReqs;nCounter++)
      {
psMem = &( sbc_enc_config.mem_info.sMemInfoSub[nCounter]);
psMem->pSBCBasePtr = malloc(psMem->nSBCSize);
}
/* initialize and allocate input and output buffers*/
in_buf = (uint8 *)malloc(ENC_IN_BUF_SIZE);
out_buf = (uint8 *)malloc(ENC_OUT_BUF_SIZE);
sbc_enc_input.input_frame_buf = in_buf;
sbc_enc_output.out_buffer = out_buf;
/* fill the input stream information */
/* INITIALIZE THE ENCODER */
eRetVal = sbc_enc_init(& sbc_enc_config);
if(eRetVal !=SBC_OK)
```

```
      exit(2);
/* read data into the input buffer */
fread(in_buf, <size>, 1, <input fd>);
/* call the main encode function */
sbc_encoder_encode(&sbc_enc_config,& sbc_enc_input,& sbc_enc_output);
/* write the encoded output to a file */
fwrite(sbc_enc_output.out_buffer, 1,sbc_enc_output.size ,<fd>);

/* free the buffers once encoding is over */
nNumReqs = sbc_enc_config.mem_info.nNumMemReqs;
for(nCounter = 0;nCounter<nNumReqs;nCounter++)
      {
      free(psMem->pSBCBasePtr);
      }
free(in_buf);
free(out_buf);
```