06-3644-API-ZCH70

MARCH 31 , 2009

0.1

Application Programming Interface for FLAC Decoder

ABSTRACT:

Application Programming Interface for FLAC Decoder

KEYWORDS:

Multimedia codecs, FLAC, LPC, RICE

Revision History

VERSION	DATE	AUTHOR	CHANGE DESCRIPTION
0.1	24-July-2008	Guo Yue	Initial Draft.

Table of Contents

Introduction

1.1 Purpose

This document gives the details of the application programmer's interface of the FLAC Decoder.

1.2 Scope

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

1.3 Audience Description

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

1.4 References

1.4.1 Standards

- FLAC official website: http://flac.sourceforge.net/index.html

1.4.2 General references

- A. J. Robinson for his work on Shorten; FLAC trivially extends and improves the fixed predictors, LPC coefficient quantization, and Rice coding used in Shorten.
- S. W. Golomb and Robert F. Rice; their universal codes are used by FLAC's entropy coder.
- N. Levinson and J. Durbin; the reference codec uses an algorithm developed and refined by them for determining the LPC coefficients from the autocorrelation coefficients.

1.4.3 Freescale Multimedia References

- FLAC Decoder Application Programming Interface – flac_dec_api.doc
- FLAC Decoder Requirements Book - flac _dec_reqb.doc
- FLAC Decoder Test Plan - flac _dec_test_plan.doc
- FLAC Decoder Release notes - flac _dec_release_notes.doc
- FLAC Decoder Test Results – flac _dec_test_results.doc
- FLAC Decoder Performance Results – flac _dec_perf_results.doc
- FLAC Decoder Interface Header – flac_dec_interface.h
- FLAC Decoder Application Code – flac _dec_api.c

1.5 Definitions, Acronyms, and Abbreviations

TERM/ACRONYM	DEFINITION
API	Application Programming Interface
ARM	Advanced RISC Machine
DAC	Digital to Analog Converter
FSL	Freescale
IEC	International Electro-technical Commission
ISO	International Standards Organization
LC	Low Complexity
OS	Operating System
PCM	Pulse Code Modulation
RVDS	ARM RealView Development Suite
TBD	To Be Determined
UNIX	Linux PC x/86 C-reference binaries
FLAC	Free Lossless Audio Codec
LPC	Linear Predicting Coding

1.6 Document Location

docs/flac_dec

2 API Description

This section describes the steps followed by the application to call the FLAC decoder. 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 member variables inside the structure are prefixed as FLAC_ or app_ to indicate if that member variable needs to be initialized by the decoder or application.The FLAC decoder API currently support PULL mode..

Step 1: Allocate memory for Decoder parameter structure

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

```
/* Decoder  parameter  structure */
typedef struct FLACD_Dec_Config {
      FLAC_Mem_Alloc_Info      flacd_mem_info;
      void                     *flacd_decode_info_struct_ptr;
      int (*read_callback)(FLAC__byte** buffer, FLAC__uint32 *bytes, void
*context);
      int channel_no;
      int bit_per_sample;
      int sampling_rate;
      int total_sample;
      int block_size;
      void* context;
} FLACD_Decode_Config;
```

Description of the decoder parameter structure

flacd_mem_info

This is memory information structure. The application needs to call the function FSL_FLACD_query_memory() to get the memory requirements for decoder. The decoder will fill in this structure. This will be discussed in step 2.

flacd_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.

read_callback (used in PULL mode only)

Function pointer to swap buffers. It means to get the bit stream. The application has to initialize this pointer.

channel_no

After FSL_FLACD_initiate_decoder() is called, the variable of channel_no will be stored the number of channel of this bit stream. The Maximum number of channel is 8 for FLAC.

bit_per_sample

After FSL_FLACD_initiate_decoder() is called, the variable of bit_per_sample will be stored the number of bit per sample. The Maximum number of bit per sample is 32 for FLAC.

sampling_rate

After FSL_FLACD_initiate_decoder() is called, the variable of sampling_rate will be stored sampling frequency. The Maximum sampling frequency is 192kHz for FLAC.

block_size

After FSL_FLACD_initiate_decoder() is called, the variable of block_size will be stored the number of samples per block. The Maximum number of samples per block is 32768 for FLAC.

context

The application has to initialize this pointer, and it will be passed to read_callback() function to point a variable which is utilized for the application.

Example pseudo code for this step:

```
/* Allocate memory for the decoder parameter */
pDecoder_Config =
(FLACD_Decode_Config*)alloc_fast(sizeof(FLACD_Decode_Config));
```

Step 2: Get the decoder version information

This function returns the codec library version information details. It can be called at any time and it provides the library's information: Component name, supported ARM family, Version Number, supported OS, build date and time and so on.

The function prototype of *FSL_FLACD_decoder_version_info* is :

C prototype:

```
const FLAC__int8 * FSL_FLACD_decoder_version_info();
```

Arguments:

- None.

Return value:

- const char * - The pointer to the constant char string of the version information string.

Example pseudo code for the memory information request

```
{
// Output the version information of FLAC decoder.
printf( "%s\n", FSL_FLACD_decoder_version_info() );
}
```

Step 3: Get the decoder memory requirements

The FLAC decoder does not do any dynamic memory allocation. The application calls the function *FSL_FLACD_query_memory()* to get the decoder memory requirements. This function must be called before all other decoder functions are invoked.

The function prototype of *FSL_FLACD_query_memory* is :

C prototype:

```
FLACD_RET_TYPE FSL_FLACD_query_memory( FLACD_Decode_Config
*pDecoder_Config );
```

Arguments:

- pDecoder_Config - Decoder config pointer.

Return value:

- FLACD_OK - Memory query successful.
- FLACD_ERROR_INIT - Error (For other error codes refer to appendix).

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

<u>Memory information structure array</u>

```
typedef struct {
      /*  Number of valid memory requests */
      FLAC__int32             flacd_num_reqs;
      FLAC_Mem_Alloc_Info_Sub mem_info_sub[FLAC_MAX_NUM_MEM_REQS];
} FLAC_Mem_Alloc_Info;
```

Description of the structure *FLAC_Mem_Alloc_Info*

<u>*flac_num_reqs*</u>

The number of memory chunks requested by the decoder.

<u>*mem_info_sub*</u>

This structure contains each chunk's memory configuration parameters.

```
typedef struct {
      FLAC__int32       flacd_size;     /* Size in bytes */
      FLAC__int32       flacd_type;    /* Memory type Fast or Slow */
      FLAC_MEM_DESC   flacd_mem_desc; /* to indicate if it is scratch
memory */
      FLAC__int32      flacd_priority; /* In case of fast memory, specify
the priority */
      void         *app_base_ptr; /* Pointer to the base memory , which
will be allocated filled by the  application */
} FLAC_Mem_Alloc_Info_Sub;
```

Description of the structure *FLAC_Mem_Alloc_Info_sub*

<u>*flacd_size*</u>

The size of each chunk in bytes.

<u>*flacd_type*</u>:

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 request for a FAST_MEMORY for which the application allocates a SLOW_MEMORY, the decoder will still decode, but the performance (Mhz) will suffer.)

flacd_mem_desc

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

flacd_priority

In case, if the decoder 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.

app_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 assign the base pointer of this chunk of memory to *app_base_ptr*. The application should allocate the memory which is aligned to a 4 byte boundary in any case.

```
typedef enum
{
    FLAC_STATIC_MEM,        /* 0 for static memory */
    FLAC_SCRATCH_MEM        /* 1 for scratch memory */
} FLAC_MEM_DESC;
```

Example pseudo code for the memory information request

```
    /* Query for memory */
    retval = FSL_FLACD_query_memory( pDecoder_Config );
    if( retval != FLACD_OK )
    {
        printf("ERROR: FSL_FLACD_query_memory() failed\n");
        return 1;
    }
```

Step 4: Allocate Data Memory for the decoder

In this step the application allocates the memory as required by FLAC Decoder and fills up the base memory pointer *'app_base_ptr' of 'FLAC_Mem_Alloc_Info_sub*' structure for each chunk of memory requested by the decoder.

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

```
/* Number of memory chunks requested by the decoder */
    for(i = 0; i < pDecoder_Config->flacd_mem_info.flacd_num_reqs; i++)
    {
        FLAC_Mem_Alloc_Info_Sub* mem = NULL;
        mem = &(pDecoder_Config->flacd_mem_info.mem_info_sub[i]);

        if (mem->flacd_type == FLAC_FAST_MEMORY)
        {
            /* This function allocates memory in internal memory */
            mem->app_base_ptr = alloc_fast (mem->flacd_size);
            memset(mem->app_base_ptr, 0xfe, mem->flacd_size);
            if (mem->app_base_ptr == NULL)
                return 1;
        }
        else
        {
            /* This function allocates memory in external memory */
            mem->app_base_ptr = alloc_fast (mem->flacd_size);
            if (mem->app_base_ptr == NULL)
                return 1;
         }
    }
```

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

Step 5: Memory allocation for input buffer

The application has to allocate the memory needed for the input. It is desirable to have the input buffer allocated in FAST_MEMORY, as this may improve the performance (Mhz) of the decoder. There is no restriction on the size of the input buffer to be given to the decoder. The recommended minimum size would be 8K Bytes. The decoder, whenever it needs the FLAC bit-stream, shall call the function *file_read_callback_()* internally from the function *FSL_FLACD_decode_frame()*. *file_read_callback_()* should be implemented by the application. The application might have different techniques to implement this function. Sample code is given in section 5.1.1

Example pseudo code for allocating the input buffer

```
    /* Allocate memory for input buffer */
    input_buffer = (FLAC__uint8*)alloc_fast(FLAC_INPUT_PULL_BUFFER_SIZE);
    if (input_buffer == NULL)
    {
        return 1;
    }
```

Step 6: Initialization routine

All initializations required for the decoder are done in *FSL_FLACD_initiate_decoder()*. This function must be called before the main decoder function is called. The input buffer pointer is needed to be passed to the initialization function. This is required by the decoder to start decoding the bitstream to begin with.

C prototype:

```
FLACD_RET_TYPE FSL_FLACD_initiate_decoder( FLACD_Decode_Config
*pDecoder_Config, FLAC__uint8 *inbuf );
```

Arguments:

- `pDecoder_Config` Decoder parameter structure pointer.
- input_buffer Initial pointer to the input buffer

Return value:

- FLACD_OK - Initialization successful.
- Other codes - Initialization Error

<u>Example pseudo code for calling the initialization routine of the decoder</u>

```
/* Initialize the FLAC decoder. */
retval = FSL_FLACD_initiate_decoder( pDecoder_Config, input_buffer );
if( retval != FLACD_OK )
{
    printf("ERROR: FSL_FLACD_initiate_decoder() failed\n");
    return 1;
}
```

Step 7: Memory allocation for output buffer

The application has to allocate memory for the output buffers to hold the decoded stereo PCM samples for a maximum of one frame size. The pointer to this output buffer needs to be passed to the FSL_FLACD_decode_frame() function. The application can allocate memory for output buffer in external memory using alloc_fast. 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 fast memory.

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

```
    /* allocate for output buffer */
    /* No. of channels * Size of long word * FLACD_SUBFRAME_SIZE */
    outbuf = (FLAC__uint8 *)alloc_fast( pDecoder_Config-
>channel_no*pDecoder_Config->block_size*sizeof(FLAC__uint32) );
    if (outbuf == NULL)
          return 1;
```

Step 8: Call the frame decode routine

The main FLAC decoder function is *FSL_FLACD_decode_frame()*. This function decodes the FLAC bit stream in the input buffer to generate one frame of decoder output per channel in every call. The output buffer is filled with samples of different channels separately.

C prototype:

```
FLACD_RET_TYPE FSL_FLACD_decode_frame(
                  FLACD_Decode_Config *pDecoder_Config,
                  FLAC__uint32* poutlength,
                  FLAC__uint8* outbuf);
```

Arguments:

- pDecoder_Config — Decoder parameter structure pointer
- poutlength — Decoder output buffer length
- outbuf — Pointer to the output buffer to hold the decoded samples

Return value:

- FLACD_OK — Indicates decoding was successful.
- Others — Indicates error

When the decoder encounters the end of bitstream, the application comes out of the loop. In case of error while decoding the current frame, the application can just ignore the frame without processing the output samples by continuing the loop.

<u>Example pseudo code for calling the main decode routine of the decoder</u>

```
    while(1)
    {
        retval = FSL_FLACD_decode_frame( pDecoder_Config, &outlength,
outbuf );
        if( retval != FLACD_OK && retval != FLACD_CONTINUE_DECODING &&
retval != FLACD_COMPLETE_DECODING )
        {
            printf("ERROR: FSL_FLACD_decode_frame() failed\n");
            return 1;
        }
        else if( retval == FLACD_COMPLETE_DECODING )
        {
            fprintf(stderr, "decoding: %s\n", "succeeded");
            break;
        }
        else
        {
            /* write it */
            if(outlength != 0)
            {
                  if(audio_output_frame(pDecoder_Config, outbuf,
outlength, fout) != 0)
                        return 1;
```

```
            }
            continue;
        }
    }
```

Step 9: Free memory

The application releases the memory that it allocated to FLAC Decoder if it no longer needs the decoder instance.

```
    free (input_buffer);
    free (outbuf);
    for (i = 0; i < pDecoder_Config->flacd_mem_info.flacd_num_reqs; i++)
    {
        free (pDecoder_Config-
>flacd_mem_info.mem_info_sub[i].app_base_ptr);
    }
    free (pDecoder_Config);
```

2.1 Call back function usage

Call back function is only used when in PULL mode. It is called by the decoder to get a new input buffer for decoding. This function is called by the FLAC decoder within the *FSL_FLACD_decode_frame()* function when it runs out of current bit stream input buffer. The decoder uses this function to return the used buffer and get a new bit stream buffer. This function will be assigned to the pointer file_read_callback_() before the init is called.

This function should be implemented by the application. The parameter buffer is a pointer to pointer. This will hold the recently used buffer by the decoder when this function is called. The application can decide to free this or do any sort of arithmetic to get any new address. The application needs to put the new input buffer pointer in *buffer to be used by the decoder.

The interface for this function is described below:

C prototype:

```
int file_read_callback_(FLAC__byte** buffer, FLAC__uint32 *bytes, void
*context);
```

Arguments:

- *buffer* - Pointer to the new buffer given by the application.
- *bytes* - Pointer to length of the new buffer in bytes. (8bit)
- *context* - Point to the context needed by the application and framework.

Return value:

- 0 - Buffer allocation successful.
- 1 - End of bitstream
- 2 - Error for *byte==0

Example pseudo code

```
int file_read_callback_(FLAC__byte** buffer, FLAC__uint32 *bytes, void
*context)
{
  (void)context;
  if(*bytes > 0)
  {
    if (InBufLen < uFileSize)
    {
        /* Bytes available */
        pUsedBuf = *buffer + *bytes;
        *buffer = (pInBuf + InBufLen - (FLAC_INPUT_PULL_BUFFER_SIZE -
*bytes) );
        if ((InBufLen + *bytes) > uFileSize)
            *bytes = (FLAC__uint32) (uFileSize - InBufLen);
        else
            *bytes = *bytes;
        InBufLen += *bytes;
        return 0;
    }
    else
    {
        /* Exhausted the buffer */
        *bytes = 0;
        return 1;
    }

  }
  else
    return 2; /* abort to avoid a deadlock */
}
```