ARM MPEG-2 Audio Layer III Decoder

Version 1

Programmer's Guide

ARM

ARM DUI 0121B

Release Information

The following changes have been made to this document.

Change history

Date	Issue	Change
May 1999	A	First release
June 1999	B	Second release, minor changes

Proprietary Notice

Preface

This preface introduces the ARM *Moving Pictures Experts Group* (MPEG)-2 Audio Layer III (MP3) Decoder. It contains the following sections:

About this guide

This guide is provided with the ARM MP3 Decoder. It describes the *Application Program Interface* (API) to the MP3 decoder library.

Intended audience

This document has been written for programmers who want to integrate the ARM MP3 Decoder into an embedded system.

Organization

This book is organized into the following chapters:

Chapter 1 *Introduction*

This chapter describes the bitstream input format required by the ARM MP3 Decoder.

Chapter 2 *ARM MP3 Decoder Types and Constants*

This chapter describes the types and constants defined by the ARM MP3 Decoder.

Chapter 3 *ARM MP3 Decoder Functions*

This chapter describes the functions provided by the ARM MP3 Decoder.

Chapter 4 *Example Use of ARM MP3 Decoder API*

This chapter contains an example C program that uses the API of the ARM MP3 Decoder.

Typographical conventions

The following typographical conventions are used in this document:

bold	Highlights interface elements, such as menu names. Also used for emphasis in descriptive lists, where appropriate.
italic	Highlights special terminology, denotes internal cross-references, and citations.
`typewriter`	Denotes text that may be entered at the keyboard, such as commands, file and program names, and source code.
`type`writer	Denotes a permitted abbreviation for a command or option. The underlined text may be entered instead of the full command or option name.
`typewriter italic`	Denotes arguments to commands and functions where the argument is to be replaced by a specific value.
`typewriter bold`	Denotes language keywords when used outside example code.

Further reading

The following documents are referenced in this guide or may prove useful as reference material.

Reference

ISO/IEC 11172-3, *Information technology-Coding of moving pictures and associated audio for digital storage media at up to about 1.5Mbit/s-Part 3: Audio*, 1993.

ISO/IEC 13818-3, *Information technology-Generic coding of moving pictures and associated audio information-Part 3: Audio*, 1998.

Feedback

ARM Limited welcomes feedback on the ARM MP3 Decoder and this documentation.

Feedback on this document

If you have any comments or suggestions about this document, please send an email to `errata@arm.com` giving:

- the document title
- the document number
- the page number(s) to which your comments apply
- a concise explanation of your comments.

General suggestions for additions and improvements are also welcome.

Feedback on the ARM MP3 Decoder

If you have any problems with the ARM MP3 Decoder (AS022), and you have a valid support contract, please contact your supplier. To help us provide a rapid and useful response, please submit error reports in the form specified in the support agreement, giving:

- details of the release you are using
- details of the platform you are running on, such as the hardware platform, operating system type, and version
- a small stand-alone sample of code that reproduces the problem
- a clear explanation of what you expected to happen, and what actually happened
- the commands you used, including any command-line options
- sample output illustrating the problem.

Contents
Programmer's Guide

Chapter 1
Introduction

This chapter introduces the ARM MP3 Decoder, and describes the bitstream input format it requires. This chapter contains the following sections:

1.1 About the ARM MP3 Decoder

The ARM MP3 Decoder is an optimized library, designed to efficiently decode MP3 on the ARM processor family. MP3 is a widely-used audio compression standard, designed for generic mono or stereo audio. At 128kbps, with a compression ratio of about 11:1, the quality is subjectively similar to compact disc. At 64kbps, the quality is similar to FM radio.

The ARM MP3 Decoder is compliant with:

- audio layer III ISO standards:
  - ISO/IEC 11172-3:1993 (MPEG-1)
  - ISO/IEC 13818-3:1998 (MPEG-2 LSF).
- the MPEG-2.5 extensions.

Multichannel audio (MPEG-2 MC) is not supported.

All output rates are supported. These include:

- 32kHz, 44.1kHz, and 48kHz (MPEG-1)
- 16kHz, 22.05kHz, and 24kHz (MPEG-2)
- 8kHz, 11.025kHz, and 12kHz (MPEG-2.5).

The output format is non-interleaved 16-bit stereo *Pulse Code Modulation* (PCM).

All input rates are supported. These include:

- 32kbps-320kbps (MPEG-1)
- 8kbps-160kbps (MPEG-2 and MPEG-2.5)
- free-format bitrate.

The ARM MP3 Decoder requires ARMv4.

1.2 Bitstream input format

The bitstream must be presented to the decoder as an array of 32-bit **`int`**s, each with the left (most significant) bit first. For example, the stream:

```
1111 1111 1111 1010 1001 0100 0111 1100...
```

would be represented by:

```
0xfffa947c,...
```

If the input from the hardware is right-bit-first, the user code will have to *bit-wise reverse* the input. The ARM Applications Library contains an efficient macro, `BITREVC`, which bit-wise reverses a 32-bit register in 12 cycles. Using this macro, on the maximum bitrate of 320kbps, the penalty is about 0.1MHz, ignoring load/store overhead. Please refer to the ARM website, `http://www.arm.com`, for details on products such as the ARM Applications Library.

Chapter 2
ARM MP3 Decoder Types and Constants

This chapter describes the types and constants defined by the ARM MP3 Decoder.

This chapter contains the following sections:

2.1 Enumerations and structures

This section describes the C types that are used to interface to the following ARM MP3 Decoder functions, described in Chapter 3 *ARM MP3 Decoder Functions*:

- *InitMP3Audio()* on page 3-2
- *MP3SearchForSyncword()* on page 3-4
- *MP3DecodeInfo()* on page 3-5
- *MP3DecodeData()* on page 3-7.

2.1.1 The tSampleRate enumeration

The `tSampleRate` enumeration codes the sampling frequency of the decoded PCM. It is defined as follows:

```
typedef enum tagSampleRate
{
       SR_11_025kHz,
       SR_12kHz,
       SR_8kHz,
       SR_ReservedMPEG2_5,
       SR_22_05kHz,
       SR_24kHz,
       SR_16kHz,
       SR_ReservedLSF,
       SR_44_1kHz,
       SR_48kHz,
       SR_32kHz,
       SR_Reserved
} tSampleRate ;
```

You may assume that the order of the elements of this enumeration will not change.

2.1.2 The tMPEGStatus enumeration

The `tMPEGStatus` enumeration codes the return status of each ARM MP3 Decoder function. Refer to the function descriptions in Chapter 3 *ARM MP3 Decoder Functions* to find out which values may be returned by each function. The `tMPEGStatus` enumeration is defined as follows:

```
typedef enum tagMPEGStatus
{
       eNoErr,
       eNoSyncword,
       eCRCError,
       eBrokenFrame,
       eEndOfBitstream,
       eDataOverflow,
       eCantAllocateBuffer,
       eUnsupportedLayer,
        eFrameDiscarded,
       eReservedSamplingFrequency,
       eForbiddenBitRate
} tMPEGStatus ;
```

You may assume that the success status code, `eNoErr`, will be element zero in all releases, but the order of the other elements may change.

Note

The elements `eEndOfBitstream`, `eDataOverflow`, and `eCantAllocateBuffer` are never returned by the ARM MP3 Decoder functions. They have been included so that user functions may return the `tMPEGStatus` type to flag errors in addition to those that may be returned by the ARM MP3 Decoder.

2.1.3 The tMPEGBitstream structure

The `tMPEGBitstream` structure points to the current location in the MPEG bitstream. It is defined as follows:

```
typedef struct tagMPEGBitstream
{
     unsigned int *bufptr;
     unsigned int bitidx;
} tMPEGBitstream;
```

where:

bufptr is a pointer to the current 32-bit word of the bitstream.

bitidx is an index within the 32-bit word. The value can range from 0-31, inclusive, where:

0 the next bit is bit 31, the *Most Significant Bit* (MSB) of this word, followed by bit 30

1 the next bit is bit 30 of this word

...

31 the next bit is bit 0 of this word, followed by bit 31 of the next word.

2.1.4 The tMPEGHeader structure

The `tMPEGHeader` structure contains information from the MPEG audio header. It is written by the `MP3DecodeInfo()` function. This structure is defined as follows:

```
typedef struct tagMPEGHeader
{
       tSampleRate    sample_rate;
       unsigned int   samplesperchannel;
       unsigned int   numchans;
       unsigned int   packed_info;
       unsigned int   bits_required;
       unsigned int   free_format;
} tMPEGHeader;
```

where:

`sample_rate`
: is the PCM sampling frequency.

`samplesperchannel`
: is the number of samples-per-channel that will be returned by `MP3DecodeData()`.

`numchans` is the number of channels (one or two).

`packed_info`
: is the header information packed into a 32-bit word. The format of *`packed_info`* is shown in Table 2-1:

Table 2-1 Format of packet_info

Bit number	Description
20	`1` if MPEG 1 or MPEG2. `0` if MPEG 2.5
19	`ID`
17, 18	`layer`: `11`—layer 1 `10`—layer 2 `01`—layer 3 `00`—reserved.
16	`protection_bit`
12-15	`bitrate_index`

Table 2-1 Format of packet_info (continued)

Bit number	Description
10, 11	`sampling_frequency`
9	`padding_bit`
8	`private_bit`
6, 7	`mode`
4, 5	`mode_extension`
3	`copyright`
2	`original/copy`
0, 1	`emphasis`

See section 2.4.2.3 of ISO/IEC 11172-3 for a complete description of the above header information.

`bits_required`

is the number of bits required by the next call to `MP3DecodeData()`. This does not include the 32 bits of header already read.

`free_format`

must be set to one if the bitstream is in free-format mode. See *Notes* on page 3-6 in the *MP3DecodeInfo()* section.

2.1.5 The tMPEGInstance type

The `tMPEGInstance` type should be regarded as an opaque type, used as a temporary workspace by the ARM MP3 Decoder. A pointer to an instance of this type is passed to each of the API functions. This is the only RAM used by the decoder.

For convenience, where the decoder is used in simple applications, the library contains an instance of this type called `MPEGInstance`. For most applications, this will be the only instance required.

Where an application needs to support multiple decoder formats, or where multiple instances of the MP3 decoder are required, an instance of type `MPEGInstance` should be declared externally, and a pointer to this instance should be passed to each of the API functions. For more information, refer to the header file `mpgdata.h`. This file specifies the minimum size and alignment restrictions of the datatype. The easiest way to allocate a block of RAM with the necessary alignment is to define it in a small ARM assembly language file using the `ALIGN` area attribute.

2.2 Constants

This section describes the constants defined by the ARM MP3 Decoder.

2.2.1 MP3_MAX_PCM_LENGTH—maximum length of PCM returned

This constant defines the maximum number of PCM samples per channel returned by a single call to `MP3DecodeData()`. It is the maximum value of the *`samplesperchannel`* element of the `tMPEGHeader` structure written by `MP3DecodeInfo()`.

The parameters *`left`* and *`right`* in the function `MP3DecodeData()` must both point to **`short`** arrays with at least `MP3_MAX_PCM_LENGTH` elements.

2.2.2 MP3_MAX_BITS_REQUIRED—maximum number of bits required

This constant defines the maximum number of bits required by the `MP3DecodeData()` function.

The constant is defined as follows:

```
#define  MP3_MAX_BITS_REQUIRED     11520
```

Note

The value 11520 corresponds to the maximum bit rate of 320kbps and the minimum sampling frequency of 32kHz. It is calculated as follows:

320000 (bits per second) * 1152 (samples per channel per frame) / 32000 (samples per channel per second) = 11520 (bits per frame).

2.2.3 MP3_NINFOBITS—size of MPEG audio frame header

This constant defines the number of bits required by the function `MP3DecodeInfo()`. It is defined as follows:

```
#define  MP3_NINFOBITS     32
```


Chapter 3
ARM MP3 Decoder Functions

This chapter describes the functions provided by the ARM MP3 Decoder.

This chapter contains the following section:

3.1 Functions

This section describes the functions provided by the MP3 decoder library:

- *InitMP3Audio()*
- *MP3SearchForSyncword()* on page 3-4
- *MP3DecodeInfo()* on page 3-5
- *MP3DecodeData()* on page 3-7.

3.1.1 InitMP3Audio()

This function initializes the MPEG audio decoder. It must be called at system reset and must also be called before decoding a new MPEG bitstream.

It clears the `main_data` buffer so the new frame cannot refer backwards into the previous frame from the wrong bitstream (see Figure 3-1). If it is not called, there may be loud clicks on the PCM output, though the decoder will always remain stable.

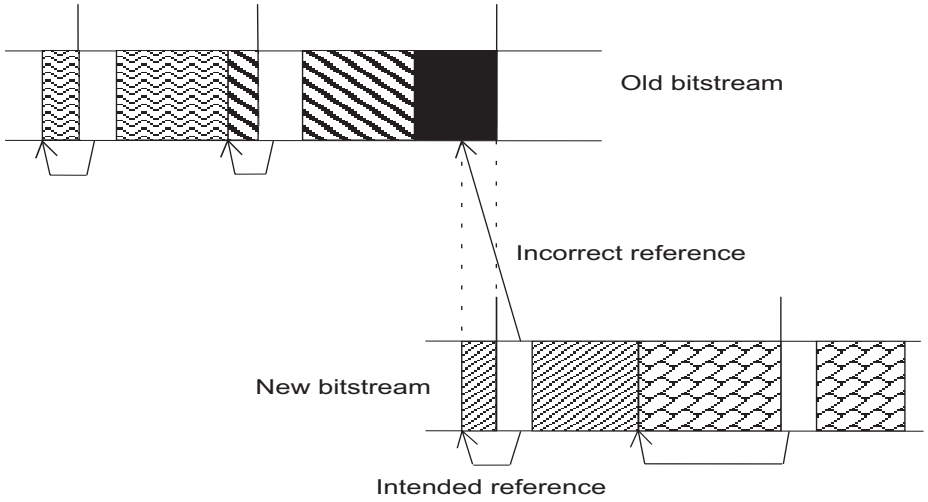

Figure 3-1 Invalid reference—MP3 bitstream

Refer to Figure A.7 in ISO/IEC 11172-3 for complete details on this diagram.

This function also clears the `filterbank` and `IMDCT` histories, so output from the previous bitstream does not *bleed* into the new one.

Syntax

```
void InitMP3Audio(tMPEGInstance *inst)
```

where:

inst is a pointer to the instance data for the decoder.

3.1.2 MP3SearchForSyncword()

This function searches for the *synchronization word* (syncword) that marks the beginning of the next MPEG audio frame.

Syntax

```
tMPEGStatus MP3SearchForSyncword(tMPEGInstance *inst,
                                 tMPEGBitstream *bs,
                                 unsigned int length)
```

where:

inst is a pointer to the instance data for the decoder.

bs is a pointer to the structure that holds the incoming bitstream buffer. The buffer is updated to:

- the end of the buffer minus 11 bits, if the syncword is not found
- the beginning of the syncword, if the syncword is found.

length is the number of new bits that are valid.

Return value

`tMPEGStatus` the function's return status:

`eNoErr` the syncword was found

`eNoSyncword`
: there is no syncword.

Usage

This function may not be required if the syncword is known to be at the start of the buffer, which is typically the case for `.mp3` files or MP3 embedded within some multiplex.

Notes

The syncword in the bitstream must be byte-aligned (see section 2.3 of ISO/IEC 11172-3). This function makes the assumption that byte alignment in the MP3 stream is equivalent to byte alignment in memory.

3.1.3 MP3DecodeInfo()

This function decodes the header from the MPEG frame, returning the length of the MPEG audio frame.

There must be at least `MP3_NINFOBITS` valid bits of incoming bitstream available.

Syntax

```
tMPEGStatus MP3DecodeInfo(tMPEGInstance *inst,
                          tMPEGBitstream *bs,
                          tMPEGHeader *pmpeg_hdr)
```

where:

`inst` is a pointer to the instance data for the decoder.

`bs` is a pointer to the structure that holds the incoming bitstream buffer. The buffer is moved forward `MP3_NINFOBITS` bits to point to the beginning of the audio data, as defined in 2.4.1.7 of ISO/IEC 11172-3.

`pmpeg_hdr` is a pointer to a structure that will be updated with MPEG audio header information. For a description of the `tMPEGHeader` structure, see *The tMPEGHeader structure* on page 2-5.

Return value

`tMPEGStatus` the function's return status:

`eNoErr` header decoded with no errors

`eNoSyncword`
: no syncword

`eBrokenFrame`
: header is inconsistent

`eUnsupportedLayer`
: unsupported or illegal (00) MPEG audio layer

`eReservedSamplingFrequency`
: undefined sampling frequency (11)

`eForbiddenBitRate`
: illegal bit rate (1111).

Notes

If the bitstream is in *free-format* mode, the size of the frame is not indicated. For more information on free-format mode, see section 2.1.67 in ISO/IEC 11172-3.

In this case, `bits_required` will contain the maximum value corresponding to the maximum bit rate of 320kbps. It is safe to provide this length of valid bitstream, and use the updated bitstream structure from `MP3DecodeData()` to find out how much data was actually used.

In some cases, the frame size may be known to the application, such as when the frame is embedded within a multiplex that indicates the length of the frame. In these cases, you should ignore `bits_required` and provide the complete frame.

3.1.4 MP3DecodeData()

This function decodes a frame of PCM samples from the MP3 stream.

Syntax

```
tMPEGStatus MP3DecodeData(tMPEGInstance *inst, short *left,
                          short *right, tMPEGBitstream *bs)
```

where:

inst is a pointer to the instance data for the decoder.

left is a pointer to the output buffer for left-channel PCM.

right is a pointer to the output buffer for right-channel PCM.

bs is a pointer to the structure that holds the incoming bitstream.

Note

The number of elements written to *left* and *right* is given by the element `samplesperchannel` of the `tMPEGHeader` structure. This value is written by the `MP3DecodeInfo()` function. The maximum value that can be written is given by `MP3_MAX_PCM_LENGTH`, described in *MP3_MAX_PCM_LENGTH—maximum length of PCM returned* on page 2-8.

Return value

`tMPEGError` the function's return status:

`eNoErr` the frame decoded with no errors

`eCRCError`
: *cyclic redundancy check* (CRC) error

`eBrokenFrame`
: the frame is inconsistent

`eFrameDiscarded`
: insufficient main data to decode the frame.

Usage

The bitstream must contain at least the number of valid bits requested by `pmpeg_hdr->bits_required`, returned by the previous call to `MP3DecodeInfo()`. The function `MP3DecodeData()` has no way of ascertaining whether this is true, so if insufficient data is available, the PCM output is undefined, though the decoder will remain stable.

The bitstream structure will be updated to reference the first bit in the bitstream immediately after the audio frame.

If required, the external program may inspect the bitstream structure to check that frame decoding has not advanced beyond the end of the data. If it has, you should ignore the PCM output from this function, and call `InitMP3Audio()` to flush the `filterbank` histories and `main_data` buffer.

Chapter 4
Example Use of ARM MP3 Decoder API

This chapter contains an example C program that uses the API of the ARM MP3 Decoder. It contains the following section:

- *Example C program* on page 4-2.

4.1 Example C program

Example 4-1 shows a simple example program using the ARM MP3 Decoder. The example is provided for reference purposes only.

Example 4-1

```
#include <stdio.h>
#include <stdlib.h>

#include "mpgdata.h"
#include "mpgaudio.h"

/* external prototypes */

tMPEGStatus InitAudio( unsigned int buffsize ) ;
unsigned int LoadData( FILE *file ) ;
unsigned int PlayData( void ) ;

/* local globals */

static int              *lgBuffer = NULL ;
static unsigned int     lgBufferSize = 0 ;
static unsigned int     lgLengthDataInBuffer = 0 ;
static tMPEGBitstream   lgMPEGBitstream = { NULL, 0 } ;

/* macros */

#define GetNBitsRemaining( ) \
   ( lgLengthDataInBuffer*32 - \
      /* current bit position in buffer = */       \
      ( ( ( lgMPEGBitstream.bufptr - lgBuffer )*32 ) + lgMPEGBitstream.bitidx ) )

/* local functions */

static void ClearBuffer( void )
{
    memset( lgBuffer, 0, lgBufferSize * sizeof( int ) ) ;
    lgLengthDataInBuffer = 0 ;
}
```

```
static tMPEGStatus GetDecodedData( short *left, short *right,
                                   tMPEGHeader *pmpeg_hdr )
{
    tMPEGStatus mpg_error ;

    if( MP3SearchForSyncword( &MPEGInstance, lgMPEGBitstream,
                              GetNBitsRemaining( )) != eNoErr )
    {
        return eEndOfBitstream ;
    }

    if( ( mpg_error = MP3DecodeInfo( &MPEGInstance, lgMPEGBitstream, pmpeg_hdr ) )
                                     != eNoErr )
    {
        return mpg_error ;
    }

    return MP3DecodeData( &MPEGInstance, left, right, lgMPEGBitstream ) ;
}

static unsigned int InterleaveChannels( short *left, short *right,
                                        short *interleaved,
                                        unsigned int samples_per_channel )
{
    unsigned int sample;

    for( sample = samples_per_channel ; sample /* > 0 */ ; sample -= 1 )
    {
        *interleaved++ = *left++ ;
        *interleaved++ = *right++ ;
    }

    return( samples_per_channel * 2 ) ;
}
```

```
/* functions */

tMPEGStatus InitAudio( unsigned int buffsize )
{
    if( lgBuffer != NULL )
    {
        /* checks if buffer has been allocated previously and frees it */
        /* if it has */
        free( ( void * )lgBuffer ) ;
        lgBuffer = NULL ;
    }

    if( ( lgBuffer = ( int * )malloc( buffsize * sizeof( int ) ) ) == NULL )
    {
        return eCantAllocateBuffer ;
    }
    lgBufferSize = buffsize ;

    return eNoErr ;
}

unsigned int LoadData( FILE *file )
{
    ClearBuffer( ) ;
    return fread( ( void * )lgBuffer, sizeof( int ), lgBufferSize, file ) ;
}

unsigned int PlayData( void )
{
    short           left[ MP3_MAX_PCM_LENGTH ] ;
    short           right[ MP3_MAX_PCM_LENGTH ] ;
    short           interleaved[ MP3_MAX_PCM_LENGTH * 2 ] ;
    tMPEGStatus     mpg_error ;
    unsigned int    num_samples_interleaved ;

    InitMP3Audio( &MPEGInstance ) ;

    /*
    initialise the tMPEGBitstream structure with the pointer to the start of
    the buffer containing the MP3 data and set the bit index as the
    most significant bit
    */
    lgMPEGBitstream.bufptr = lgBuffer ;
    lgMPEGBitstream.bitidx = 0 ;
```



```
    while( ( mpg_error = GetDecodedData( left, right, &mpeg_hdr ) )
                                != eNoErr )
    {
        num_samples_interleaved = InterleaveChannels( left, right, interleaved,
                                                      mpeg_hdr.samplesperchannel ) ;
        /*
        the function

        void ProcessPCM( short *pcm, unsigned int nsamples ) ;

        below is not given here and is any function that processes the interleaved
        pcm data post-MP3 decoding
        */

        ProcessPCM( interleaved, num_samples_interleaved ) ;
    }
    if( mpg_error != eEndOfBitstream )
    {
        return 0 ;
    }
    return 1 ;
}

/* main routine */

int main( void )
{
    FILE         *file ;
    unsigned int error ;

    if( ( file = fopen( "filename.mp3", "rb" ) ) == NULL )
    {
        return 1 ;
    }

    if( InitAudio( 1024 * 1024 ) != eNoErr )/* 1Mb of samples */
    {
        return 1 ;
    }
    error = 0 ;
```

```
    if( LoadData( file ) )
    {
        error = PlayData( ) ;
        error = !error ;                /* invert sense for return from main() */
    }

    fclose( file ) ;

    return error ;
}
```

Index

The items in this index are listed in alphabetic order, with symbols and numerics appearing at the end. The references given are to page numbers.

P

R

S

T

