ARM AS030—Dolby Digital Decoder

Version 1

Programmer’s Guide

ARM

ARM DUI 0128A

Release Information

The following changes have been made to this document.

Change history

Date	Issue	Change
September 1999	A	First release

Proprietary Notice

Contents
Programmer's Guide

Preface

This preface introduces the ARM Dolby Digital Decoder. It contains the following sections:

About this book

This book is provided with the ARM Dolby Digital Decoder. It describes the *Application Program Interface* (API) to the Dolby Digital Decoder library.

Intended audience

This book is written for programmers who want to integrate the ARM Dolby Digital Decoder into an embedded system.

Using this book

This book is organized into the following chapters:

Chapter 1 *Introduction*

Read this chapter for an introduction to the ARM Dolby Digital Decoder.

Chapter 2 *The ARM Dolby Digital Decoder API*

Read this chapter for a description of the interface to the ARM Dolby Digital Decoder.

Chapter 3 *ARM API and Dolby SIP Differences*

Read this chapter to learn all the differences between the interfaces of the ARM Dolby Digital Decoder and the Dolby Software Interface Protocol.

Typographical conventions

The following typographical conventions are used in this book:

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.

`typewriter` 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

This section lists publications from both ARM Limited and third parties that provide additional information on developing the ARM Dolby Digital Decoder.

ARM periodically provides updates and corrections to its documentation. See `http://www.arm.com` for current errata sheets and addenda.

See also the ARM Frequently Asked Questions list at:
`http://www.arm.com/DevSupp/Sales+Support/faq.html`

ARM publications

This book contains reference information that is specific to the ARM Dolby Digital Decoder. For additional information, refer to the following ARM publications:

- *Fixed Point Arithmetic on the ARM Application Note* (ARM DAI 0033)
- *ARM Application Library Programmer's Guide* (ARM DUI 0081).

Other publications

For other reference information relating to the ARM Dolby Digital Decoder, please refer to the following:

- *Digital Audio Compression Standard*, ATSC Document A/52, 1995.
- *Dolby DSP Software Interface Protocol* (S96/10549/10816), Dolby Laboratories Inc.

  To obtain this document, you should contact Dolby Laboratories.

Feedback

ARM Limited welcomes feedback on both the ARM Dolby Digital Decoder, and its documentation.

Feedback on the ARM Digital Decoder

If you have any problems with the ARM Dolby Digital Decoder, please contact your supplier. To help us provide a rapid and useful response, please give:

- 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 standalone 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
- the version string of the tool, including the version number and date.

Feedback on this book

If you have any comments on this book, please send 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 the problem.

General suggestions for additions and improvements are also welcome.

Chapter 1
Introduction

This chapter provides an overview of the ARM Dolby Digital Decoder. It includes the following sections:

1.1 Overview

This guide is provided with the ARM Dolby Digital Decoder, an optimized library that is designed to efficiently decode Dolby Digital AC-3 for the ARM processor family. The ARM Dolby Digital Decoder has been submitted for evaluation by Dolby Laboratories and fully complies with Dolby Laboratories decoder *Integrated Circuit* (IC) requirements.

This *Application Programmer Interface* (API) is as similar as possible to the Dolby *Software Interface Protocol* (SIP). This guide describes the API in detail and indicates exactly how the API differs from the Dolby SIP.

1.1.1 Features supplied by the ARM Dolby Digital Decoder

This section describes the basic features included with the ARM Dolby Digital Decoder. Some are provided with the source, while others do not contain the sources. The features provided are:

- *Dolby Digital Decoder library—without source*
- *Floating point support—with source*
- *Command-line application—with source* on page 1-4
- *Real-time demonstration software—with source* on page 1-4.

Dolby Digital Decoder library—without source

The Dolby Digital Decoder library is available without source files. It is built with:

- either little-endian or big-endian
- 1.31 fixed-point format for all fractional values
- no software stack checking
- ARM-code only (no Thumb)
- no unaligned load or store instructions.

The decoder is karaoke-capable.

Please contact ARM if you require different options than those above.

Floating point support—with source

The support library provides a floating-point interface to the Dolby Digital Decoder library. Its usage is illustrated in Figure 1-1 on page 1-3.

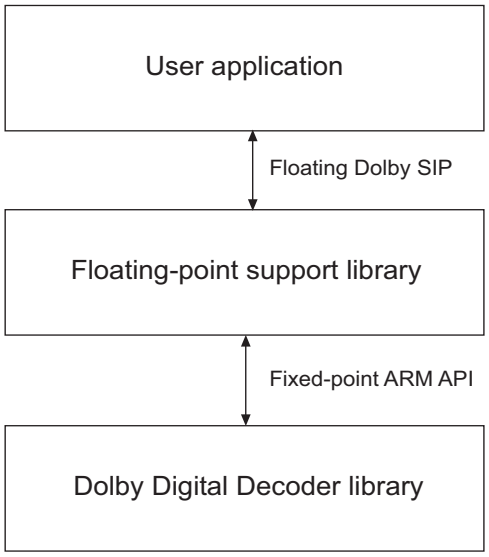

Figure 1-1 Using the optional floating point support

If you have code written to use the Dolby Simulator, version 3.11, it will work immediately when linked with the support library and the Dolby Digital Decoder, provided the code conforms to the Dolby SIP.

Note

All references to the Dolby Simulator throughout this book refer to version 3.11.

It is unlikely that you will use any of the support code in the final product because it is very slow. On a typical ARM processor without floating-point hardware, the support code takes much longer to run than the decoder.

Each fractional value in the Dolby SIP may be independently set to a floating-point or fixed-point value. This will enable a working floating-point implementation to be gradually ported to fixed-point. If a fractional value is naturally held as a floating-point value in your application, it may be beneficial to leave the floating-point conversion in place. However, it is usually preferable to move the conversion overhead so that it is not executed on every AC-3 block.

Command-line application—with source

The command-line application consists of the following files from the Dolby Simulator:

- `USR_COM.H` (unmodified)
- `USR_EQU.H` (unmodified)
- `DOLBY_PL.H` (unmodified)
- `DECODE.C` (unmodified)
- `UDATA_D.C` (unmodified)
- `USR_SIM.H` (modified to include only ANSI headers if built for ARM).

The project file, `original.apj` is also supplied.

This project links with the support library and the Dolby Digital Decoder library to build the Dolby Digital command-line decoder. The command line option `-h` returns the usage information.

Real-time demonstration software—with source

ARM supplies real-time demonstration software that runs on the EBSA285 development board. Refer to the `readme.txt` file in the root directory that is created when you install the software.

1.2 Reference decoder version

The ARM Dolby Digital Decoder is based on the Dolby AC-3 Decoder Simulation.

The following version numbers returned by the decoder library are the same as the reference code:

```
AC3I_REV      0x0300      (AC-3 Frame Information)
CRC_REV       0x0300      (CRC Calculation)
AC3D_REV      0x030B      (AC-3 Decode)
```

1.3 Terms and conventions

In this document, the following terms are used:

Application Programmer Interface (API)
is the interface between the decoder library and the user program.

audio block is part of a synchronization frame, containing 256 samples per channel.

executive is a term used in Dolby documentation to refer to the systems software calling the decoder library.

Software Interface Protocol (SIP)
is the Dolby Laboratories term for the interface between the decoder library and the user program.

synchronization frame
is one frame of AC-3 compressed audio, containing 1536 samples per channel. A synchronization frame contains six audio blocks.

syncword is a 16-bit constant that marks the beginning of a synchronization frame.

Conventions used throughout the document, such as the use of bold or italic font, are explained in *Typographical conventions* on page vii.

Chapter 2
The ARM Dolby Digital Decoder API

This chapter describes the API to the ARM Dolby Digital Decoder. It is designed to be read alongside *Dolby DSP Software Interface Protocol*.

This chapter contains the following sections:

2.1 Fractional values

All fractional values are passed in 1.31 fixed-point form.

A fractional value, *x*, between -1.0 (inclusive) and 1.0 (exclusive), is represented by the integer *n*, given by:

$$n = x.2^{31}$$

A full list of the fractional values represented in this way is:

- dynamic range scale factors
- *Pulse Code Modulation* (PCM) output scale factor
- user-specified downmix table
- user-specified karaoke level/pan table
- decoded PCM output.

The application note *Fixed Point Arithmetic on the ARM* shows how to write efficient fixed-point code on the ARM. The type `APIfract` is used to pass 1.31 fixed-point values. It is defined as follows:

```
typedef int APIfract;
```

2.2 Bitstream input format

The bitstream must be presented to the decoder as an array of 16-bit **`short`**s, each with the left (most significant) bit first.

For example, the following stream:

```
0000 1011 0111 0111 1101 0010 1000 0101…
```

would be represented by:

```
0x0b77, 0xd285…
```

On a little-endian machine, this is equivalent to an integer array beginning with:

```
0xd2850b77
```

On a big-endian machine, this is equivalent to an integer array beginning with:

```
0x0b77d285
```

If the input from the hardware is right-bit-first, the user code will have to bit-wise reverse the input. An example of such code can be found in the ARM Applications Library, which contains an efficient macro, `BITREVC`. This macro bit-wise reverses a 32-bit register in 12 cycles. Using this macro, on a 320kbps stream, the penalty is about 0.1MHz, ignoring load/store overhead.

2.3 Interface to the Dolby Digital Decoder

The header file `dolbyapi.h` contains the complete API definitions for the ARM Dolby Digital Decoder. The interface to the decoder is designed to be as similar as possible to the Dolby *Digital Signal Processing* (DSP) described in *Dolby DSP Software Interface Protocol*.

The decoder library is accessed using the `dolby_api()` function defined in `dolbyapi.h`:

```
typedef struct dolby_api {
    int funcnum;      /* function number */
    int status;       /* error status */
    void *param_ptr;  /* pointer to parameter list structure */
    } DOLBY_API;

__value_in_regs DOLBY_API dolby_api(DOLBY_API input_sip);
```

There are no other functions or global variables available to the main program.

This file also declares structures suitable for the data pointed to by `param_ptr`. This file is similar to the `dolby_pl.h`, supplied with the Dolby Simulator, except for the following differences:

- All fractional values are 1.31 fixed-point integer.
- Several function and structure names have been changed to draw attention to the differences (see *Changes to names of structures and functions* on page 3-3).
- The return type of `dolby_api` is `__value_in_regs`. This instructs the ARM compiler to pass this structure more efficiently.
- The elements `funcnum` and `status` of `DOLBY_API` are **`int`**s rather than **`short`**s. This allows `__value_in_regs` to use the processor registers more efficiently.
- The header is guarded against multiple inclusion.

The filename has been changed to draw attention to these differences.

The *Dolby DSP Software Interface Protocol* document refers to the processor registers FR, SR, and LPR for subroutine input and output. The ARM implementation, like the Dolby Simulator, uses the structure defined in Table 2-1.

Table 2-1 FR, SR, and LPR registers

Register name	Description	Element of DOLBY_API structure
Function Register (FR)	The function number	`int funcnum`
Status Register (SR)	Error status for the routine	`int status`
List Pointer Register (LPR)	Pointer to parameter list structure. This will point to an instantiation of the appropriate structure defined in `dolbyapi.h`.	`void *param_ptr`

To allow for platform independence, all 16-bit values in the structures pointed to by LPR are of type `DSPshort` in the original Dolby code. The type is defined as:

```
typedef short DSPshort;
```

2.4 Dolby Digital Decoder functions

Table 2-2 provides a list of the functions supported by the Dolby Digital Decoder library.

Table 2-2 Dolby Digital Decoder supported functions

Function number	Name	Description
0	`DD_SYS_INIT`	System initialization
1	`DD_AC3_INFO`	AC-3 frame information
2	`DD_CRC_CALC`	*Cyclic Redundancy Check* (CRC) calculation
3	`DD_AC3_DEC`	AC-3 decode
4	`DD_AC3_AUX`	AC-3 auxiliary data decode

This section provides an overview of each function provided by the Dolby Digital Decoder library. Refer to *Dolby DSP Software Interface Protocol* for specific details of each function. You can refer to *Specific deviations from the Dolby SIP* on page 3-4 for differences between the ARM API and the API described in *Dolby DSP Software Interface Protocol*.

2.4.1 Function 0: system initialization

The initialization function must be called once on system start up, before any other function is called. This function is compatible with the corresponding function in the Dolby Simulator.

This function does not perform a significant amount of processing. It also does not cause any function called afterward to take significantly longer than usual.

Inputs

FR 0 (`DD_SYS_INIT`).
SR 0.
LPR 0.

Outputs

FR Dolby subroutine package version.
SR 0.
LPR 0.

2.4.2 Function 1: AC-3 frame information

This function accepts the first 20 bytes of an AC-3 synchronization frame and provides information from the header of the frame.

Inputs

FR 1 (`DD_AC3_INFO`).

SR 0.

LPR Pointer to the input parameter structure:

```
typedef struct ac3_info_pl {
  DSPshort size; /* input parameter list size */
  DSPshort *iptr;/* input buffer pointer */
  DSPshort ioff; /* input offset pointer */
  DSPshort imod; /* input modulo pointer (ignored) */
  DSPshort icfg; /* input buffer config (ignored) */
} AC3API_INFO_PL;
```

Outputs

FR AC-3 frame information version.

SR Return status:

- **0** No errors.
- **1** Invalid frame syncword.
- **2** Invalid sample rate.
- **3** Invalid data rate.

LPR Pointer to the return parameter structure:

```
typedef struct ac3_info_rl {
  DSPshort size;    /* return parameter list size */
  DSPshort bscfg;   /* bitstream configuration */
  DSPshort frmsize; /* frame size */
  DSPshort crcsize; /* first CRC buffer size */
  DSPshort bsinfo;  /* bitstream info */
  DSPshort dialnorm;/* dialog normalization values */
  DSPshort langcod; /* language code values */
  DSPshort audprod; /* audio production values */
  DSPshort timecod1;/* time code values, 1st half */
  DSPshort timecod2;/* time code values, 2nd half */
} AC3API_INFO_RL;
```

The output parameter structure is encoded as described in *Dolby DSP Software Interface Protocol*.

If an error is returned, the pointer to the return parameter structure is undefined.

2.4.3 Function 2: CRC calculation

This function calculates the CRC of the input data stream. This function should be called twice per synchronization frame:

- Before decoding the first audio block of the sync frame:

  Perform a CRC on the first 5/8 of the sync frame, **`short`**s numbered 1 through *`crcsize`* of the frame buffer, inclusive. The *`crcsize`* parameter is returned by function one (see *Function 1: AC-3 frame information* on page 2-7).

  Note

  The first **`short`** (numbered 0) is the sync word, which is not covered by the CRC.

- Before decoding the third audio block of the synchronization frame:

  Perform a CRC on the final 3/8 of the sync frame, **`short`**s numbered *`crcsize`*+1 through *`frmsize`*-1 of the frame buffer, inclusive. The *`crcsize`* and *`frmsize`* parameters are returned by function one (see *Function 1: AC-3 frame information* on page 2-7).

The processing time taken by this function is very small, but you can distribute the computational load across several calls, if required.

Inputs

FR 2 (`DD_CRC_CALC`).

SR Input CRC syndrome. This should be zero on the first call.

LPR Pointer to the input parameter structure:

```
typedef struct crc_calc_pl {
  DSPshort size; /* input parameter list size */
  DSPshort *iptr;/* input packed buffer pointer */
  DSPshort ioff; /* input packed buffer offset */
  DSPshort imod; /* input packed buffer modulo */
                 /* (ignored) */
  DSPshort icfg; /* input buffer config (ignored) */
  DSPshort count;/* CRC word count */
} A3API_CRC_CALC_PL;
```

Outputs

FR CRC calculation version.

SR Output CRC syndrome.

At the end of the calculation, this value will be zero if the data has passed the CRC-check, otherwise it will be nonzero. It is acceptable to divide the CRC calculation into more than one CRC call, but only if the output syndrome becomes the input syndrome of the next call.

Note

The specific nonzero value returned in case of failure may be different than that returned by the Dolby Simulator.

LPR 0.

2.4.4 Function 3: AC-3 decode

This function decompresses AC-3 audio data into PCM samples. This function must be called six times per sync frame, once for each 256-sample-per-channel audio block.

Inputs

FR 3 (`DD_AC3_DEC`).

SR Input block error status:

- **0** No errors.
- **positive** Known errors. Repeat previous audio block.
- **negative** Known errors. Mute outputs.

LPR Pointer to the input parameter structure:

```
typedef struct ac3_dec_pl {
  DSPshort size;    /* input parameter list size */
  DSPshort *iptr;   /* input buffer pointer */
  DSPshort ioff;    /* input offset pointer */
  DSPshort imod;    /* input modulo pointer */
                    /* (ignored) */
  DSPshort icfg;    /* input buffer config */
                    /* (ignored) */
  APIfract **optr;  /* output packed buffer */
                    /* pointer */
  DSPshort *ooff;   /* output packed buffer offset */
  DSPshort *omod;   /* output packed buffer modulo */
                    /* (ignored) */
  DSPshort ocfg;    /* output buffer config */
                    /* (partly ignored) */
  DSPshort blknum;  /* current block number */
  APIfract dynsclh; /* dynamic range scale high */
                    /* value */
  APIfract dynscll; /* dynamic range scale low */
                    /* value */
  APIfract pcmscl;  /* pcm scale factor */
  DSPshort rptmax;  /* maximum repeat value before */
                    /* muting */
  APIfract *dnmxptr;/* user-specified downmix */
                    /* table */
  APIfract *krkptr; /* karaoke-capable mix/pan */
                    /* parameters */
  DSPshort debug;   /* Dolby internal use only */
} AC3API_DEC_PL;
```

Refer to *Dolby DSP Software Interface Protocol* for a full description of the input parameters.

Outputs

FR AC-3 decoder version.

SR Output block status:

- **0** No errors.
- **1** Input status nonzero. Outputs were repeated.
- **2** Input status nonzero. Outputs were muted.
- **3** Unsupported bit stream identification revision.
- **4** Unsupported number of channels in input stream.
- **5** Unsupported number of input streams.

LPR 0.

2.4.5 Function 4: AC-3 auxiliary data decode

This function copies the auxiliary data within the AC-3 stream into a user buffer.

Inputs

FS 4 (`DD_AC3_AUX`).

SR 0.

LPR Pointer to the input parameter structure:

```
typedef struct ac3_aux_pl {
  DSPshort size;    /* input parameter list size */
  DSPshort *iptr;   /* input packed buffer pointer */
  DSPshort ioff;    /* input packed buffer offset */
  DSPshort imod;    /* input packed buffer modulo */
                    /* (ignored) */
  DSPshort icfg;    /* input buffer config */
                    /* (ignored) */
  DSPshort maxcnt;  /* max aux data count */
  DSPshort *auxptr; /* aux data buffer pointer */
  DSPshort auxoff;  /* aux data buffer offset */
  DSPshort auxmod;  /* aux data buffer modulo */
                    /* (ignored) */
  DSPshort auxcfg;  /* aux data buffer config */
                    /* (ignored) */
} AC3API_AUX_PL;
```

Refer to *Dolby DSP Software Interface Protocol* for a full description of the input parameters.

Outputs

FR AC-3 auxiliary data decode version.

SR Lost data count. This is the number of bits that could not be returned because the output buffer was too small. If your application permits dynamic allocation, for example, you can try to increase the size of the buffer and call this function again.

LPR Pointer to the output parameter structure:

```
typedef struct ac3_aux_rl {
  DSPshort size;    /* output parameter list size */
  DSPshort auxcnt;  /* number of aux bits copied */
  DSPshort *auxptr; /* updated aux buffer pointer */
  DSPshort auxoff;  /* updated aux buffer offset */
} AC3API_AUX_RL;
```

Chapter 3
ARM API and Dolby SIP Differences

This chapter describes the differences between the ARM Dolby Digital Decoder API and the Dolby SIP. It is intended to assist in porting a Dolby-interface application to the ARM Dolby Digital Decoder.

This chapter contains the following sections:

3.1 Summary of deviations from Dolby SIP

The main differences between the ARM API and the Dolby SIP are:

- No modulo addressing. This would impose a large overhead on execution time with marginal benefit.
- Buffer data width is not adjustable.
- PCM rounding control is not adjustable.
- Fractional values are represented as 1.31 fixed-point integers. The support library implements a slower floating-point API.

These changes are reflected in the structures defined in `dolbyapi.h`.

3.2 Changes to names of structures and functions

The C header file `dolbyapi.h` is based on the header file `dolby_pl.h` that is supplied with the Dolby Simulator, version 3.11. The structure and function names have been modified to draw attention to the differences, as shown in Table 3-1.

Table 3-1 Changes from dolby_pl.h to dolbyapi.h

Old name in `dolby_pl.h`	**New name in** `dolbyapi.h`	**Description**
`dolby_sub()`	`dolby_api()`	Entry point to the Dolby Digital Decoder
`DOLBY_SIP`	`DOLBY_API`	Arguments for the entry point function
`AC3_INFO_PL`	`AC3API_INFO_PL`	Parameter list for header decode
`AC3_INFO_RL`	`AC3API_INFO_RL`	Return list for header decode
`CRC_CALC_PL`	`AC3API_CRC_CALC_PL`	Parameter list for CRC calculation
`AC3_DEC_PL`	`AC3API_DEC_PL`	Parameter list for decode data
`AC3_DEC_RL`	`AC3API_DEC_RL`	Return list for decode data

3.3 Specific deviations from the Dolby SIP

This section describes specific differences, by function, between the ARM Dolby Digital Decoder API and the Dolby SIP.

3.3.1 Function 0: system initialization

There is no deviation of this function from the Dolby SIP.

3.3.2 Function 1: AC-3 frame information

This function deviates from the Dolby SIP as follows:

- Input parameter 3 (input modulo pointer) is ignored. There is no modulo addressing.
- Input parameter 4 (input buffer configuration information) is ignored. Bits 15-14 are assumed to contain binary value 01.

3.3.3 Function 2: CRC calculation

There is no deviation of this function from the Dolby SIP.

As described in *Dolby DSP Software Interface Protocol*, this function returns zero if, and only if, the CRC passes. If the CRC fails, the value returned is nonzero, but it may not be the same nonzero value that would be returned by the Dolby Simulator, version 3.11.

3.3.4 Function 3: AC-3 decode

This function deviates from the Dolby SIP as follows:

- Input parameter 3 (input modulo pointer) is ignored. There is no modulo addressing.
- Input parameter 4 (input buffer configuration information) is ignored. Bits 15-14 are assumed to contain the binary value 01.
- Input parameter 7 (pointer to the array of output buffer modulos) is ignored. There is no modulo addressing.
- Input parameter 8 (output buffer configuration information) is limited in scope:
  - Bits 15-14 (Output buffer data width) are ignored. They are assumed to contain binary value 00.
  - Bits 11-10 (PCM rounding control) are ignored. The decoder always returns 1.31 fixed-point integers with maximum accuracy.
- Input parameter 10 (dynamic range scale factor, high) is a 1.31 fixed point integer.
- Input parameter 11 (dynamic range scale factor, low) is a 1.31 fixed point integer.
- Input parameter 12 (PCM scale factor) is a 1.31 fixed point integer.
- Input parameter 14 (user-specified downmixing table pointer) is a pointer to a 6x6 array of 1.31 fixed-point integers.
- Input parameter 14 (user-specified karaoke level/pan table) is a pointer to an array of 1.31 fixed-point integers.

3.3.5 Function 4: AC-3 auxiliary data decode

This function deviates from the Dolby SIP as follows:

- Input parameter 3 (input modulo pointer) is ignored. There is no modulo addressing.
- Input parameter 4 (input buffer configuration information) is ignored. Bits 15-14 are assumed to contain the binary value 01.
- Input parameter 8 (auxiliary modulo pointer) is ignored. There is no modulo addressing.
- Input parameter 9 (auxiliary buffer configuration information) is ignored. Bits 15-14 are assumed to contain binary value 01.

Index

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

K

L

O

P

R

S

U

