Revolution SDK DSPADPCM

Version 1.01

The content of this document is highly confidential and should be handled accordingly.

Contents

Code Examples

Revision History

Version	Date Revised	Item	Description
1.01	2009/03/09	-	Added AIFF files to the explanation of `-<mode>` in Chapter 2 Usage. Added support for WAV files which include loop data. (See section 3.1 WAV Files.)
1.00	2006/03/01	-	First release by Nintendo of America Inc.

1 Overview

DSPADPCM is a data conversion utility for the Wii audio system. This tool converts standard WAV or AIFF files into the DSP-ADPCM-format files. This format is specific to the hardware decoder built into the Wii audio DSP, and provides good data compression while retaining high fidelity.

The DSPADPCM tool can also convert DSP-ADPCM data back into WAV- or AIFF-format data. The conversion process models the DSP decoder exactly and thus provides a convenient method for previewing compressed data without relying on Wii hardware.

2 Usage

DSPADPCM is a Win32 console application. It has the following command-line syntax and parameters:

```
DSPADPCM -<mode> <inputfile> [<outputfile>] [-<option><argument> ……]
```

-<mode>	<mode> must be either "e" (encode) or "d" (decode). This is a required parameter and specifies the operational mode of the tool. If encoding is requested, the tool will convert a WAV or AIFF file (as specified by <inputfile>) into a DSPADPCM file (as specified by [<outputfile>]). **Note:** The [<outputfile>] parameter is optional. If omitted, the default output file name will be the input file with a ".dsp" extension. If decoding is requested, the tool will convert a DSP-ADPCM file (as specified by <inputfile>) into a WAV or AIFF file (as specified by [<outputfile>]). Again, the [<outputfile>] parameter is optional. If omitted, the default output file name will be the input file with a .wav extension.
<inputfile>	Specifies the file to be converted. This is a required parameter.
[<outputfile>]	Specifies the file that will store the converted data. If omitted, the tool will generate a filename based on <inputfile> (see <mode> above). If the user has specified decode mode and the output file already exists, the tool will abort to prevent inadvertent destruction of source data.

The DSPADPCM tool also supports the following options:

-l<start>-<end>	For encode mode only; specifies the loop points for the sample data to be converted. The <start> parameter is the raw sample address at which the loop begins. The <end> parameter is the raw sample address at which the loop ends. Both addresses are expressed in decimal. For example, "-l100-232" means that the loop starts at sample 100, and ends at sample 232. Samples are counted from zero, meaning that "sample zero" is the first sample in the file; "sample 100" is actually the one hundred-first sample in the file.
-a<endaddr>	For encode mode only; this parameter is ignored if a loop has been specified. The <endaddr> specifies the last sample to be played by the DSP. If omitted, DSPADPCM uses the sample count (minus one) of the WAV file as a default value.
-c<textfile>	Instructs DSPADPCM to dump the ADPCM file's header information into <textfile>. If <textfile> is omitted, DSPADPCM will use <inputfile> with a ".txt" extension. If the text file already exists, its contents will be destroyed.
-v	Turns on verbose mode. The tool will dump header data and processing status to stdin.

`-f`	When decoding, generates an AIFF file. Loop points specified in the DSP header of the source file will be preserved.
`-w`	When decoding, generates a WAV file. Loop points specified in the DSP header of the source file will be preserved.
`-h`	Displays help information.

3 Data Formats

3.1 WAV Files

- `DSPADPCM` converts standard WAV files into `DSP-ADPCM` format. The WAV files must contain ***monaural***, 16-bit PCM data. Loop points specified in a WAV file's `SMPL` chunk are read automatically and programmed into the header of the `DSP-ADPCM` output file.

Use of the `SMPL` chunk has the following restrictions.

- Only one loop is supported. If multiple loops are specified in the chunk, the first specification is used.
- The only supported loop type is Forward (normal).
- The loop is handled as an infinite sustain loop.
- The only parameters in the chunk that are used are those for loop type and loop position.

3.2 AIFF Files

`DSPADPCM` can also convert AIFF files into `DSP-ADPCM` format. The AIFF files must contain ***monaural***, 16-bit PCM data.

Note: Loop points in the AIFF file will be read automatically and programmed into the header of the `DSP-ADPCM` output file.

3.3 DSP-ADPCM Files

When converting data into `DSP-ADPCM` format, the tool will preface the output data with a header. The structure of the header is defined in Code 3–1:

Code 3–1 DSPADPCM Header File

```
// all data in this structure is in BIG-ENDIAN FORMAT!!!!

typedef struct
{
// for header generation during decode
    u32 num_samples;       // total number of RAW samples
    u32 num_adpcm_nibbles; // number of ADPCM nibbles (including frame headers)
    u32 sample_rate;       // Sample rate, in Hz

// DSP addressing and decode context
    u16 loop_flag;    // 1=LOOPED, 0=NOT LOOPED
    u16 format;       // Always 0x0000, for ADPCM
    u32 sa;           // Start offset address for looped samples (zero for non-looped)
    u32 ea;           // End offset address for looped samples
    u32 ca;           // always zero
    u16 coef[16];     // decode coefficients (eight pairs of 16-bit words)

// DSP decoder initial state
    u16 gain;         // always zero for ADPCM
    u16 ps;           // predictor/scale
    u16 yn1;          // sample history
    u16 yn2;          // sample history

// DSP decoder loop context
    u16 lps;          // predictor/scale for loop context
    u16 lyn1;         // sample history (n-1) for loop context
    u16 lyn2;         // sample history (n-2) for loop context

    u16 pad[11];      // reserved

} DSPADPCM;

// Header is 96 bytes long
```

This header contains information needed by the Wii audio DSP to decode and play the associated sample.

Note: All data in the header is stored in big-endian format. This facilitates transfer of the data to a Nintendo Revolution runtime application. Much of the data may be used verbatim to program the DSP for sample playback. Consult the Audio Library document set in this guide for application details.

When decoding `DSP-ADPCM` data into WAV or AIFF format, the tool will assume that this header is present at the start of the `DSP-ADPCM` file. The `DSPADPCM` tool needs the first two parameters of the header to regenerate WAV header information during decode:

`num_samples`	Number of raw, uncompressed samples in the file. Used for WAV/AIFF header generation during decode.
`num_adpcm_nibbles`	Number of `ADPCM` nibbles (including frame headers) generated for this sample. **Note:** You must round this up to the next multiple of 8 bytes to get the actual length of the data in the file because `DSPADPCM` only generates complete frames.

`sampling_rate`	Sampling rate of the data, expressed in Hertz. Used for WAV/AIFF header generation during decode.

The remaining parameters are required by the Wii audio DSP to decode and play the associated `ADPCM` sample data:

`loop_flag`	Specifies whether or not the sample is looped. This parameter is stored in big-endian format and is used by the DSP for sample playback.
`format`	Specifies the data format of the sample. Always zero. Used by the DSP for sample playback.
`sa`	Loop start offset value (in nibbles). This value includes the frame header. If not looping, specify 2, which is the top sample. In the user application, the main memory address of the sample data must be added before DSP is used.
`ea`	Loop end offset value (in nibbles). This value includes the frame header. In the user application, the main memory address of the sample data must be added before DSP is used.
`ca`	Initial offset value (in nibbles). This value includes the frame header. Always specify 2, which is the first sample. In the user application, the main memory address of the sample data must be added before DSP is used.
`coef[16]`	Decoder coefficients. This coefficient corresponds to AXPBADPCM Structure member a`[][]` in the following way.

a[0][0] = coef[0];

a[0][1] = coef[1];

a[1][0] = coef[2];

a[1][1] = coef[3];

a[2][0] = coef[4];

a[2][1] = coef[5];

a[3][0] = coef[6];

a[3][1] = coef[7];

a[4][0] = coef[8];

a[4][1] = coef[9];

a[5][0] = coef[10];

a[5][1] = coef[11];

a[6][0] = coef[12];

a[6][1] = coef[13];

a[7][0] = coef[14];

a[7][1] = coef[15];

`gain`	Gain factor. Always zero for `ADPCM` samples.

`ps`	Predictor and scale. This will be initialized to the predictor and scale value of the sample's first frame.
`yn1`	History data; used to maintain decoder state during sample playback.
`yn2`	History data; used to maintain decoder state during sample playback.
`lps`	Predictor/scale for the loop point frame. If the sample does not loop, this value is zero.
`lyn1`	History data for the loop point. If the sample does not loop, this value is zero.
`lyn2`	History data for the loop point. If the sample does not loop, this value is zero.

Some notes about decoder addressing:

- When processing `ADPCM` samples, the DSP will address memory as 4-bit nibbles.
- The values for `sa`, `ea`, and `ca` generated by `DSPADPCM` are nibble-offsets which already account for the extra space needed for `ADPCM` frame headers. For example, the one hundredth sample does not refer to the one hundredth nibble in the sample data; the one hundredth sample would actually be the one hundred-sixteenth nibble.
- The `sa`, `ea`, and `ca` values are *offsets*. When encoding data, `DSPADPCM` cannot know where the sample will be placed in memory. The user application is therefore responsible for adding a main memory address (in nibbles) to these offsets before the DSP can access the sample.
- Note that individual `ADPCM` sound effects must start on 8-byte boundaries and must be at least a multiple of 8 bytes in length. Thus, when loading one or more `ADPCM` samples into memory, the samples must be packed such that the start of each sample falls on an 8-byte boundary.