BFDLAC, A FAST LOSSLESS AUDIO COMPRESSION ALGORITHM FOR DRUM SOUNDS
or
SIZE ISN'T EVERYTHING

SKoT McDonald
FXpansion Audio (Australia)
skot@fxpansion.com
Perth, WA, Australia

ABSTRACT

BFDLAC is a fast-decoding, multi-channel, lossless audio compression algorithm developed by FXpansion for disk-streaming large drum sample libraries. It achieves near-FLAC file size reduction, but is over twice as fast to decode. These features are vital for use in real-time software instruments like FXpansion's *BFD*.

1. INTRODUCTION

BFDLAC is a new lossless audio compression algorithm developed by FXpansion for use in the 'BFD' acoustic drum software family. Our specific requirements were compressing multi-channel percussive sounds, disk-stream support, arithmetic simplicity allowing greater SIMD[1] CPU instruction use and parallel decoding.

Absolute priority is given to fast decode versus encoding times. We describe the BFDLAC algorithm, and compare its performance to other popular audio compression algorithms. BFDLAC is an algorithm that forgoes optimal performance in file size reduction and encoding time in return for a much improved decode performance.

1.1. Problem description

Sample-based software instruments have exploded in size over the past few decades. An Amiga "tracker" user in the early 1990s might have had their entire sound collection stored in a shoebox of 3.5" floppy disks, amounting to some tens of megabytes. With the ongoing "arms race" to provide ever-increasing detail and fidelity, it is now commonplace for a single software instrument to draw from hundreds of gigabytes. A typical musician may have many such behemoths installed.

The BFD family of drum software started with a 9GB core sound library in 2003. BFD2 hit the market with 55GB in 2007. In 2013, BFD3 launched with a 160GB wav-equivalent payload of sounds. A "drum kit" in BFD1 may have totaled 1GB; a BFD3 kit can span 40GB or more. BFD data is so large because every drum is exhaustively articulated: several playing styles, sometimes 100 or more velocity layers per style, and each hit recorded from 10-14 microphones. And then all that is repeated for different beater types – rods, mallets, brushes, etc.

Computer capabilities have improved in that time – most users' machines now run 64bit operating systems with 8 or more GB of RAM, and have fast solid state drives to deliver data. End-user performance expectations have also grown, possibly faster than their hardware; ever-more-sprawling expansion libraries appear, each demanding additional storage real estate. It is essential to find ways for a product to deliver more content of higher quality without massively increasing the proportion of a user's computing resources: storage, data bandwidth between storage and the CPU, and CPU power itself. Data compression can address the first two, but must do so without blowing out the CPU budget.

The product delivery model has also evolved with increasing connectivity and bandwidth. BFD1 was sold primarily through physical stores on 2 DVDs, uncompressed; BFD2 squeezed via data compression onto 5 DVDs. Nowadays, BFD3 is primarily sold as a direct download, despite being over 17 times larger than the first edition (when compared as uncompressed wav files). Digital downloads impose server and bandwidth costs on the manufacturer, as well as consuming customers' time and connection bandwidth. When dealing with payloads in the dozens of gigabytes going to thousands of customers, efforts to minimize payload size can quickly have real economic benefits. Compression is essential.

BFD Expansion Pack installers started using a customized version of the FLAC algorithm in 2005, with the launch of BFD Deluxe, a 55GB behemoth delivered on 5 DVDs. Unfortunately, FLAC decode was too slow on contemporary hardware for disk streaming, so BFD2 still required uncompressed WAVs. This might seem strange – FLAC decodes many times faster than real-time playback, but the sheer number of channels requiring simultaneous decoding in BFD can be taxing. A fairly uncomplicated drum pattern can cause 30-50 synthesis

[1] Single Instruction, Multiple Data instruction sets such as SSE and AVX allow modern CPUs to operate on several data simultaneously.

voices to play simultaneously, each with a dozen audio channels. BFD needs to be able to decode several hundred audio channels simultaneously. It must do so in a way that doesn't occupy every CPU core on a user's machine, particularly as musicians will insist on using several software instruments at once.

1.2. Solution requirements

- Get "close" to FLAC in file size reduction
- Be significantly faster than FLAC to decode.
- Disk-stream-able / random-access capable
- SIMD-instruction friendly
- Multi-core / parallel-processing friendly
- Support encoding for different target bit depths.

2. APPROACH

We are not trying to develop a general purpose audio encoder; rather observe and exploit features of the specific dataset in question.

Exponential decay: Single-hit drum samples are typically dominated by exponential decay envelopes. For cymbals and toms, a sample may extend for some tens of seconds. Sample producers additionally fade out samples over several seconds to minimise detection of an artificial end point. Large reductions are possible simply by variable encoding of bit depth.

"Simple" harmonics: Every sample is primarily a recording of a single resonating body being struck once. There are some mechanical and "through air" vibration coupling effects with other drums, usually at low levels.

Noise: Users have an ingrained perception that 24-bit audio equates to high quality. The human auditory system has a dynamic range of approximately 120dB, or 20 bits (Bregman 1994). It is rare for raw BFD recordings to achieve noise floors below -90dB (15 bits), due to the realities of sampling with analogue electronics. This is sometimes compounded by engineers recording through classic/vintage equipment that is also perceived as High Quality / Must Haves, by themselves or customers. Thus, we estimate that there is at best 8 bits of unavoidable noise, 4 bits of which are unperceivable to humans without dynamic range compressors. We may discreetly hold reservations whether The Customer Is Always Right, but they are the ones holding the money: hence vast effort is expended to encode and deliver electrical noise. Noise is definitely part of the natural realism any complete recording studio simulation must provide.

BFDLAC's encoding technique is built on the assumption that encoding and decoding the noise floor is where most effort should be spent. We break the signal into small blocks of 64 samples, and exhaustively search through a small set of simple approximation algorithm-plus-error-signal pairs, determining first which of these can actually fully encode the signal-block, and which results in the smallest encoded block.

The bit depth of the Error Signal is optimized per block, but is constant within the block, unlike variable bit rate Huffman-encoded Error Signals. All block encodings can be described as a pair of Sub-Algorithm and an Error Signal *E* of bit depth *b*, which result in a known, fixed block size for each pair.

This deterministic block structure and size allows for efficient SIMD assisted decoding, particularly of the error signal bit strings. For example, an Error Signal with bit depth of 6 stores 4 samples in 3 bytes; these 4 samples can be efficiently separated with SIMD bit shifting and masking operations applied in parallel.

3. FILE FORMAT

BFDLAC is a RIFF-like, chunk-based format derived from the Electronic Arts Interchange File Format, and described by Microsoft and IBM (Microsoft & IBM, 1991). An outer "BFDC" chunk contains a set of sub-chunks, as shown in Figure 1.

<table>
<tr><td rowspan="12">BFDC</td><td colspan="3">fmt</td></tr>
<tr><td colspan="3">Indx</td></tr>
<tr><td rowspan="10">data</td><td rowspan="4">Frame 0</td><td>Block 0</td></tr>
<tr><td>Block 1</td></tr>
<tr><td>…</td></tr>
<tr><td>Block n</td></tr>
<tr><td rowspan="4">Frame 1</td><td>Block 0</td></tr>
<tr><td>Block 1</td></tr>
<tr><td>…</td></tr>
<tr><td>Block n</td></tr>
<tr><td>…</td><td>…</td></tr>
<tr><td colspan="2">[custom]</td></tr>
</table>

Figure 1. The chunk hierarchy of a BFDLAC file encoding *n* audio channels.

3.1. Format chunk ("fmt")

The format chunk encodes the following properties of the sound file:

- Channel Count (up to 255)
- Target Bit Depth (16,20,24)
- Sample Rate (44100 for BFD)
- Duration (in sample frames)
- BFDLAC version (currently 2, 1 is obsolete)

Format chunks are required, and additionally must be the first chunk in the file.

3.2. Index chunk ("Indx")

The index chunk provides a lookup table of data offsets at a specified interval of sample frames, with the aim of speeding up read access to locations within a sample without having to decode all proceeding data stream up to that location.

Index locations are encoded as a 32bit offset, at intervals of 2048 sample frames – 2048 being a common denominator of BFD's user-configurable disk-streaming buffer size.

Index chunks are optional, but should appear before the Data chunk if used.

3.3. Data chunk ("data")

The encoded data is composed of a string of *Frames*. These are not to be confused with sample frames, which represent a single simultaneous sample across multiple channels. Data chunks are required.

3.3.1. Frame

A *Frame* is a set of *Blocks*, with one block per audio channel. A Frame commences with a single byte encoding the number of audio channels. Since the count of audio channels doesn't change from the number given in the format chunk, this is also used as a simple check for data corruption or errors during decoding.

3.3.2. Block

A *Block* encodes the samples for a single audio channel within a Frame. In BFDLAC v2, block length is fixed to 64 samples. Each block commences with a header byte that describes which sub-algorithm is used (4 bits, 16 possible algorithms) and the bit-depth of an Error signal (4 bits, 16 possible error bit-depths).

All sub-algorithm/error signal pairs describe encoded data of a predetermined length. This allows easier parallel decoding of blocks; or skipping over the data of channels that are not being used, such as a muted channel.

3.4. Other chunks

Like all RIFF files, other optional custom chunks can be included to encode non-standard metadata such as BFD-proprietary meta data, authoring information, copyright notices and so on. BFDLAC encodes such information in the same manner as the RIFF-WAV format for familiarity. Optional chunks should be skipped if not recognized.

4. BLOCK ENCODING

4.1. Sub-encoding algorithms

BFDLAC is focused on fast decode, and is unapologetic about exhaustively testing all algorithm-error pairs to find the smallest encoding for every block, if needed. That said, many options are discarded quickly, and the encoding time remains competitive. We describe the 9 current algorithms restoring a signal y from starting coefficients x with a error signal E^b, b being bit-depth. Time, t, is the block-relative sample position.

4.1.1. Zero

$$y_t = 0 \tag{1}$$

Zero blocks are a special case to encode a block entirely of zeroes. It has no signal. Whilst it may seem like an extravagance to devote a sub-algorithm ID to zero, the frequency of which zeroed signal data needs encoding in BFD sounds makes it worthwhile.

4.1.2. Constant

$$y_t = x_0 \tag{2}$$

A constant, non-zero block. The value is encoded at the target bit-depth.

4.1.3. Median

$$\mathrm{y_t} = \tilde{x} + \mathrm{E_t^b} \tag{3}$$

A block with an error signal with median value $\tilde{x}$. The median is encoded at the target bit depth at the start of the block.

4.1.4. Quad

$$\begin{aligned} y_t &= x_t && t = 4n \\ y_t &= y_{4n} + E_t^b && 4n{+}1 \le t \le 4n{+}3 \end{aligned} \tag{4}$$

Blocks of length *N* are subdivided into *n* micro-blocks of 4 samples, with the micro-block initial value followed by 3 error values array of length *3N/4* storing offsets. For a block size of 64 samples, there are 16 micro-blocks.

4.1.5. Oct

$$\begin{aligned} y_t &= x_t && t = 8n \\ y_t &= y_{8n} + E_t^b && 8n{+}1 \le t \le 8n{+}7 \end{aligned} \tag{5}$$

Similar to Quad, but with micro-blocks 8 samples long.

4.1.6. Diff

$$\begin{aligned} y_t &= x_t && t = 0 \\ y_t &= y_{t-1} + E_t^b && t > 0 \end{aligned} \tag{6}$$

A first order differential, with the value y_t equal to the previous value plus the error signal. A block is encoded with a starting value x_0 followed by *N-1* error values.

4.1.7. Diff2

$$\begin{aligned} y_t &= x_t && t = 0 \\ y_t &= y_{t-1} + E_t^b && t = 1 \\ y_t &= 2y_{t-1} - y_{t-2} + E_t^b && t > 1 \end{aligned} \tag{7}$$

A second order differential, where the gradient between two prior samples is used to predict the next sample position, and then corrected by the local error value.

4.1.8. HalfDiff2

$$\begin{aligned} y_t &= x_t && t = 0 \\ y_t &= y_{t-1} + E_t^b && t = 1 \\ y_t &= (3y_{t-1} - y_{t-2})/2 + E_t^b && t > 1 \end{aligned} \tag{8}$$

A second order differential, with a scaling factor of 0.5 applied to the prediction gradient.

4.1.9. Gradient

$$y_t = A\, x_t + B + E_t^b \tag{9}$$

Linear regression is used to identify the gradient *A* and a constant *B* that best fit the data.

4.1.10. Uncompressed

If none of the other algorithms are able to encode the data, samples are simply stored at the target bit rate. Due to the block header byte, this results in 0.521% inflation for 24-bit samples with a block size of 64. Uncompressed blocks typically occur during the high-dynamic range, "noisy" onsets of sounds.

4.2. Block algorithm comparison

Table 1 shows the byte usage of the sub-algorithms for encoding signal coefficients and an 8-bit residual. Table 2 shows the frequency of which various sub-algorithms are utilized when encoding BFD3's audio data. Note that commonly used algorithms Median, Diff and Diff2 have a very high percentage of their encoding bytes used for the Error signal. A competing algorithm that was able to encode the block with only 7 bit error could lead to a large saving, provided the signal coefficient overhead didn't increase more than the saved error bytes.

Sub-algorithm	Signal	Error	Err %	Total
Zero	0	0	0.0	1
Const	3	0	0.0	4
Median	3	63	94.0	67
Quad	48	48	49.5	97
Oct	24	56	69.1	81
Diff	3	63	94.0	67
Diff2	6	62	89.8	69
HDiff2	6	62	89.8	69
Gradient	6	64	91.4	70
Uncompressed	192	0	0.0	193

Table 1. E^8 sub-algorithm byte utilization

Sub-algorithm	%	Sub-algorithm	%
Zero	35.63	Diff	33.25
Const	0.03	Diff2	10.13
Median	13.32	HDiff2	1.36
Quad	0.03	Gradient	4.69
Oct	0.01	Uncompressed	1.55

Table 1. Total incidence of each sub-algorithm at all error-signal bit-depths when encoding the entire BFD3 audio library.

Bit Depth	%	Bit Depth	%
Zero / Const	35.66	9	11.10
1	0.35	10	6.97
2	0.56	11	2.91
3	0.98	12	1.86
4	1.67	13	1.53
5	2.94	14	1.30
6	5.33	15	1.09
7	10.15	16	0.87
8	13.18	Uncompressed	1.55

Table 2. Total incidence of each Error-Signal bit depth across all algorithms.

5. ALGORITHM PERFORMANCE COMPARISON

5.1. Test

We wish to compare the encoding and decoding times, and resulting data payload sizes, of different compression algorithms. Encoding trial times are I/O inclusive, meaning start and end times include disk read and write access. Not all algorithms allowed us to time the data encoding entirely in memory. Decoding times are I/O exclusive, meaning we timed transcoding of a block of compressed data in memory to a block of uncompressed data, also in memory. We measured the decode times of only those algorithms that allowed such operations, and exhibited a useful amount of data compression.

5.1.1. Data

The test data was BFD3's "Kit 1" dataset, comprising 37GB of 14 channel 24-bit wav files. Each sound file is a multi-microphone recording of a single hit of a drum or cymbal. Drums are hit in several styles, or articulations, and each articulation will contain 30-100 dynamics, or velocity layers. There are some 3206 samples in total, 5.5 hours of 14 channel audio, equivalent to over 3 days of single channel audio.

5.1.2. Machine

All data compression / decompression timing was performed on a machine with this specification:

CPU: Intel Core i7-4770K; 4 cores, 3.50GHz
RAM: 32 GB 1600MHz DDR3 RAM
OS: Windows 10
HDD: Western Digital "Green" 2TB 7200rpm SATA3

5.2. Comparison Algorithms

The following algorithms were selected for comparison:

5.2.1. Zip (Gailly and Alder 2013)

Zip is a popular implementation of the generic data compression algorithm LZ77 (Ziv and Lempel, 1977) and Huffman encoding. It is not specialized for audio; we include it as a baseline comparison.

5.2.2. RAR, RAR5 (Roshal 2015)

RAR is a proprietary generic data compression algorithm based on LZ77 and prediction by partial matching. It has optional "audio awareness"; a user can specify input data is comprised of a number of interleaved channels, which can aid encoding.

5.2.3. Tau True Audio (TTA) (Djourik and Zhilin 2004)

A block-based approach with inter-channel decorrelation, optimal selection of polynomial / linear predictive / adaptive filter signal modelling, and entropy (error) encoding.

5.2.4. *Shorten (Robinson 1994)*

Shorten uses blocked linear predictive coding with Huffman encoded error signal. Robinson suggests blocks of 256 samples for a 16kHz signal (6ms), compared to BFDLAC's 64 sample blocks for 44.1kHz (1.5ms).

Shorten is of interest as it uses a similar small-block approach with a variety of sub-algorithms, but unfortunately the current implementation of *Shorten* doesn't support 24bit wav files. *Shorten*'s 16bit encoding achieved similar compression ratios to *BFDLAC* 16bit.

5.2.5. *Monkey's Audio (MA) (Ashland 2015)*

Monkey's Audio converts stereo signals to a Mid-Side encoding, after which predictive coding is applied. The resulting signal and entropy is Rice encoded.

5.2.6. *FLAC (Xiph 2005)*

FLAC (Free Lossless Audio Compression) is a popular and patent-free codec using linear-prediction with Golomb-Rice encoding of any residual signal. It is our benchmark for best compression by absolute size. FLAC is also well-known as an easy-to-stream format. A custom modified version of FLAC is used by FXpansion to encode BFD audio data for installer payloads. FLAC is used because reducing the absolute size of data is paramount for economic data delivery. The modification we made allowed FLAC to support the large number of audio channels needed by BFD; FLAC is natively limited to 8 channels. BFD data payloads are transcoded during installation to WAV for older versions of BFD, or to BFDLAC for more resource-efficient use within BFD3.

5.2.7. *WavPack (Byrant 2015)*

WavPack uses an adaptive linear predictive coding model operating on stereo decorrelation, and a custom bit encoder for final compression.

5.2.8. *BFDLAC v2 (McDonald 2015)*

A block-based, sub-algorithm selective, bit-depth reduction focused algorithm that trades some compression size for greater speed and simplicity. Version 2, which appeared in BFD v3.1, has a fixed block size of 64 samples, 16 possible sub-algorithms (9 currently used), and supports error components encoded as simple PCM signals to all bit depths between 1 and 16.

6. RESULTS

6.1. Encoding BFD drum data

Table 3 shows the encoding time and resulting total file size of different lossless compression algorithms encoding BFD3's "Kit 1". Figure 2 shows the encoding time versus file size trade-off of the same algorithms. Table 4 shows the effectiveness of BFDLAC on the recordings of different types of drum. Note how effective BFDLAC is on "noisy" cymbals due to bit-depth reduction during very long decay tails.

Encoding	Time (s)	Size (GB)
Wav	-	37.975
Zip (best, 32kB)	927.38	16.041
RAR	1450.02	15.033
RAR5 (32MB cache)	1852.21	14.324
TTA	1746.00	9.460
Monkey's Audio (normal)	1482.03	8.872
BFDLAC v2	**522.67**	**9.510**
WavPack (extra)	1232.31	8.214
FLAC	910.07	8.020

Table 3. A comparison of encoding times and resulting data sizes of a variety of compression algorithms

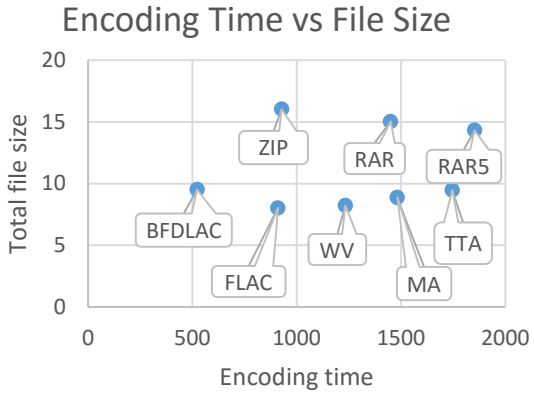

Figure 2. Encoding time versus compressed file size of various compression algorithms.

Class	Original GB	Encoded GB	%
Kick	1.767	0.515	29.10
Snare	5.267	1.492	28.33
Hihat	8.098	2.206	27.24
Tom	8.739	2.301	26.33
Cymbal	14.114	2.996	21.23

Table 4. BFDLAC encoding performance by Kit-Piece class of BFD3's Kit 1.

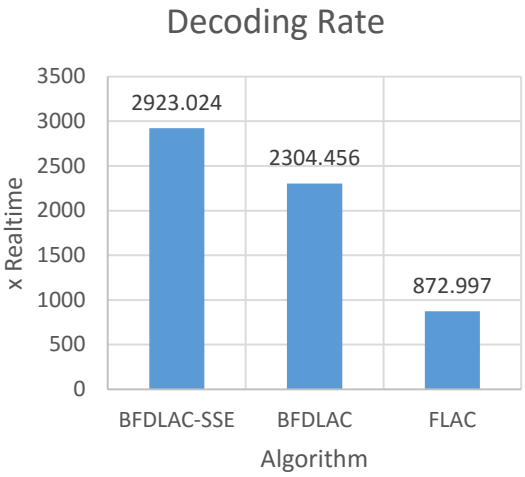

Figure 3. Comparing decode rate as a multiple of real-time playback of BFDLAC and FLAC encodings of BFD3's Kit 1.

Trade off (BFDLAC vs FLAC)

Algorithm	Disk space	Time	Real-time
Wav	100.0%	-	-
BFDLAC-SSE	25.25%	101.1	x2923
BFDLAC	25.25%	128.2	x2304
FLAC	21.12%	228.4	x873

Table 5. Resource use during playback BFDLAC versus Wav and FLAC. Time is in seconds (lower is better), Real-time is the number of channels one CPU core can decode simultaneously (higher is better).

6.1.1. Generic audio – "popular" music

BFDLAC is not designed to handle complex, mixed audio. Harmonically rich content defeats its limited ability to encode tonal sounds; its bit reduction features, which efficiently encode the long tails of percussive sounds, are little use on music with variable dynamics.

Song	WAV	FLAC	BFDLAC
Stairway To Heaven	135.6	62.1%	83.1%
Girl From Ipanema	80.9	35.4%	93.2%
Gangnam Style	56.6	49.2%	98.6%

Table 6. Comparing the encoding performance of FLAC and BFDLAC on "popular" music. Figures given are the final compressed file size as a percentage of the original wav file. WAV size is shown in MB.

7. CONCLUSION

We have introduced BFDLAC and shown it to be audio data compression algorithm that, while data-specific, yields a very beneficial compromise between optimizing for file size reduction and the time to decode data. In particular, BFDLAC-SSE trades a 4.13% reduction in wav compression performance for a 235% improvement in decode time versus FLAC.

BFDLAC is a core technology inside BFD, and crucial to BFD's delivery of industry-leading hyper-detailed and realistic drum sounds. BFDLAC enables finer-grained dynamics as well as additional drum articulations: thus more of the organic whole of an acoustic drum. On the engineering and production side, a larger palette of microphone channels gives engineers additional options for creative sound design. All this is achieved whilst reducing the computing resources needed, which of particular benefit for live performers such as e-Drummers using laptops.

Further performance improvements in both compression and speed are being actively realised, and more are anticipated. Correlation of stereo microphone pairs remains to be exploited; the work of converting more sub-algorithms and error signal decoding to full SIMD use is ongoing. Additionally, there is still scope in the file structure for deploying new block encoding sub-algorithms that result in reduced error signal bit-depth, and hence file size reduction.

8. REFERENCES

Ashland, M. 2015. *Monkey's Audio*. [Software]. Version 4.16. Available online at URL www.monkeysaudio.com. Accessed October 2015.

Bregman, A. 1994. *Auditory Scene Analysis*. MIT Press

Roshal, A. 2015. *WinRAR*. [Software]. Version 5.21, 64bit. Available online at URL www.win-rar.com. Accessed October 2015. win.rar GmbH, Berlin, Germany

Bryant, D. 2015. *WavPack*. [Software]. Version 4.75.0. Available online at URL www.wavpack.com. Accessed October 2015.

Djourik, A. and Zhilin, P. 2004. *Tau Producer*. [Software]. Available online at URL en.true-audio.com/. Accessed October 2015.

Gailly, J., and Adler, M. 2013. *ZLib*. [Software]. Version 1.2.8. Available online at URL www.zlib.net. Accessed October 2015.

Microsoft and IBM. 1991. *Multimedia Programming Interface and Data Specifications 1.0*. Available online at URL www-mmsp.ece.mcgill.ca/documents/ AudioFormats/WAVE/Docs/riffmci.pdf. Accessed October 2015.

McDonald, S. *et al*. 2015. *BFD3*. [Software]. Version 3.1.1.5. Available online at URL www.fxpansion.com/bfd3. Accessed October 2015. FXpansion Audio UK Ltd, London, UK.

Robinson, T. 1994. "SHORTEN: Simple lossless and near-lossless waveform compression". *Technical report CUED/F INFENG/TR.156*, Cambridge University

Xiph Org. 2005. *FLAC*. [Software]. Available online at URL xiph.org/flac/index.html. Accessed October 2015. Xiph Foundation

Ziv J., Lempel A. 1977. "A Universal Algorithm for Sequential Data Compression". *IEEE Transactions on Information Theory*, Vol. 23, No. 3, pp. 337-343.