fLaCPGA - FPGA fLaC Decoder

Progress Report

Emmanuel Jacyna - 24227498

May 22, 2016

Contents

Abstract

The purpose of this document is to describe the objectives of the fLaCPGA project so far, explain the progress achieved to date, and to set out the means by which the project goals will be achieved.

1 Introduction

The goal of my final year project is to investigate hardware optimization techniques through the implementation of a Free Lossless Audio Codec (henceforth, fLaC) decoder and encoder in Verilog. fLaC is a lossless audio compression codec that is very popular as a method of distributing high quality audio recordings and for compressing large audio archives[1].

Whilst there are a number of software implementations of the fLaC encoder and decoder, targeted to both traditional microprocessors and digital signal processors (DSPs), there are no freely available ASIC or FPGA implementations. An efficient hardware implementation would be of great use for a number of reasons. Many nations are now in the process of digitising large national audio archives. These audio archives require data to be compressed losslessly in order to preserve content in a format faithful to the original, however, uncompressed lossless data consumes large amounts of space. Currently, uncompressed formats such as WAVE and BWF_WAVE are quite popular for audio archiving[2].

Compressed audio has the major benefit of reducing space usage by 50% or more, potentially doubling an archive's potential storage space. In order to convert large (terabytes) amounts of audio data to a compressed format, a lot of computing power is required. A hardware decoder that performed better than a software decoder would reduce encoding time and potentially reduce power consumed by the encoding process. fLaC is also gaining popularity as a medium for portable audio players. These players are very sensitive to power consumption, thus a hardware implementation would be of great use in reducing the power load of the decoding process. Another potential use case of a hardware encoder would be in a recording studio. Instead of recording audio to an uncompressed format, high quality audio could be encoded in real time as it comes in.

2 Audio Compression Overview

Just as with general data compression, there are two main methodologies for compressing audio data, lossy and lossless. Lossy audio encoding most often includes techniques such as psychoacoustic compression, where sounds that humans cannot perceive or differentiate are removed from the audio. These techniques will not be the focus of this project. Lossless compression, by comparison, encodes audio perfectly, that is, the decoded audio samples are identical to the original input samples. Lossless techniques are still affected by problems inherent to the digitization process such as quantisation noise and recording device noise, however they provide a faithful encoding of the original data.

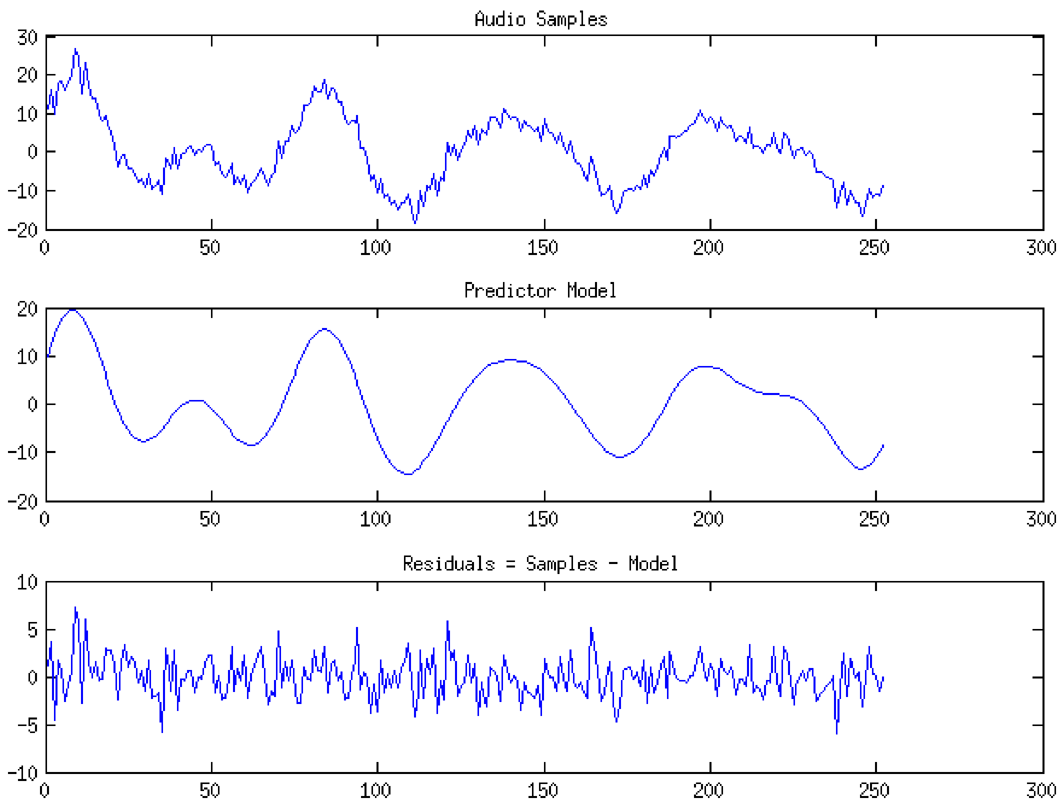


Figure 1: Predictor Concept

The main goal of any compression algorithm is to remove redundancy from data, thereby reducing the amount of information needed to reproduce the data. Audio data is often highly redundant, samples of data that are close to each other will usually have very similar patterns, for example, samples of a clarinet playing the same note for a number of seconds will clearly have a similar spectral pattern over the period of time the note is held, thus offering redundancy to be exploited. The most popular method for lossless compression of audio data is to find an accurate model of the audio (a *predictor*), find the error (often called *residuals*) between the predictor and the true audio, then to encode the residuals using a variable bit length encoding in order to increase the entropy of the signal. The coefficients of the model can then be used with the residuals to reconstruct the orginal signal.

The fLaC lossless audio compression algorithm borrows heavily from prior work in lossless encoding including the Shorten algorithm[3], and the AudioPAK algorithm[4]. These algorithms use a technique called *linear prediction* to produce their audio model, and use Golomb-Rice encoding to perform the entropy encoding phase of the compression. The advantage of these algorithms is the ease with which they can be translated into hardware, as the linear prediction step consists of a number of multiplies and adds, whilst the Golomb-Rice encoding consists of bit-shifting and unary encoding.

2.1 fLaC Overview

An overview of the fLaC decoding and encoding processes is provided to aid in the reader's understadning!!!!! !!!!!!!.

2.1.1 fLaC Encoding Process

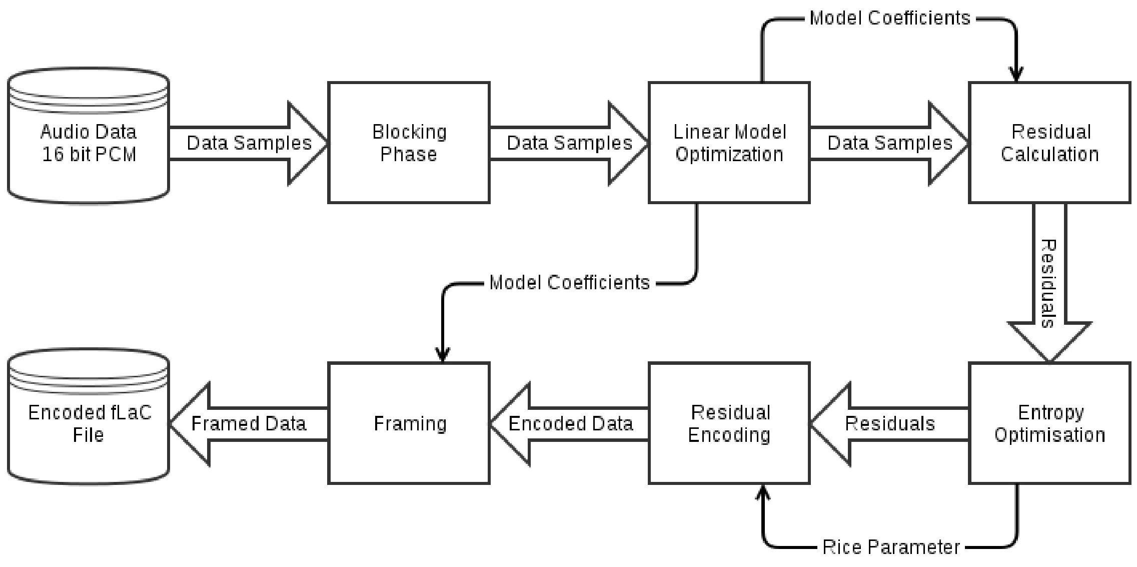


Figure 2: Overview of fLaC Encoding Process

The fLaC encoding process consists of two major stages. Finding a linear model that minimises the prediction error, and finding the best set of rice parameters to encode a block of residuals. As well as this, the output data must be framed appropriately and encoded using these optimal parameters before being output to a file/stream. Figure 2 gives an overview of the entire encoding process.

Finding the best linear model

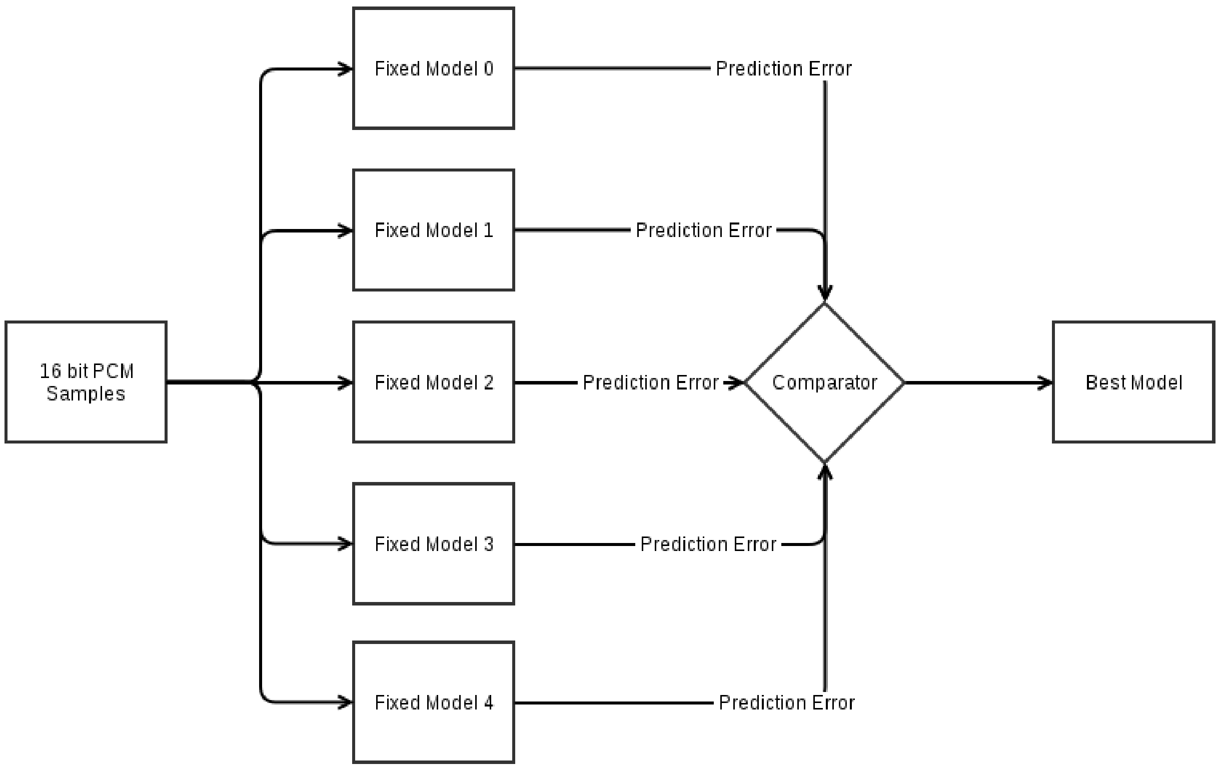


Figure 3: Finding the best fixed model

fLaC has two options for finding the best linear model. It can either choose from four precalculated predictors, the "fixed" predictors, or it can find a more accurate model using the autocorrelation of the audio signal and calculating the optimal coefficients using the Levinson-Durbin recursion method[?]. The fixed model has the advantage that it requires less computation to calculate, and it also results in a much smaller frame header size. The fixed model usually achieves around 50% compression for most audio data. The Levinson-Durbin method has the advantage that the models it calculates are almost always more accurate than the fixed models, and thus will result in better compression ratios. Choosing the best model is very simple. In the case of the fixed models, since the model coefficients are already provided, the encoder simply runs the audio through the models and calculates the total absolute error for each. Since the quality of the entropy encoding is directly related to the total absolute error, the encoder selects the model which produces the lowest error.

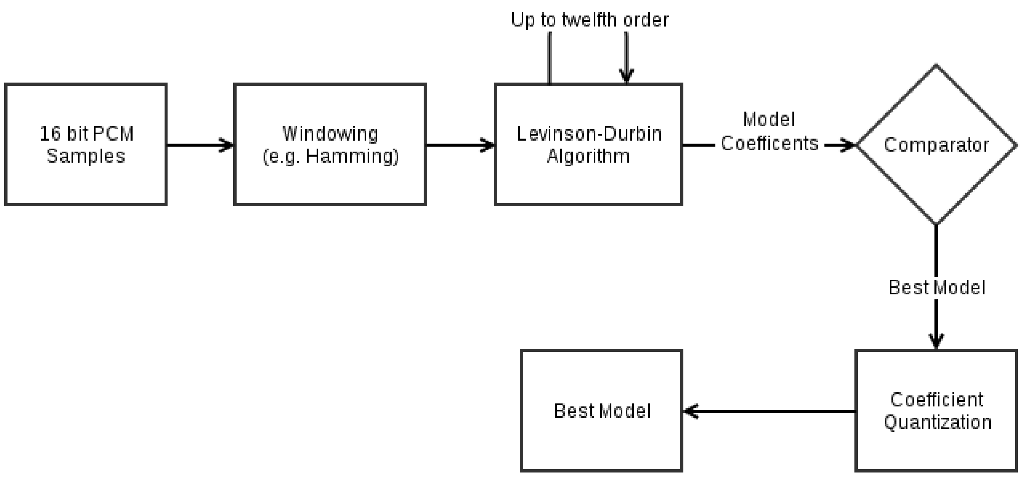


Figure 4: Finding the best lpc model

In the case of the Levinson-Durbin method, more effort is required. First, the input data must be windowed in order for it to be correctly modeled in the frequency domain[?]. The window selected tends to have a significant effect on the quality of the resulting model. The autocorrelation of the windowed data is then calculated. Next, the Levinson-Durbin algorithm is used to find the model coefficients for orders from the minimum to the maximum order (the fLaC specification allows up to twelfth order predictors[?]). The total absolute error is then calculated for each model similar to the fixed method, and the optimum model is selected. The encoder must then quantise the model coefficients, since the Levinson-Durbin method does not produce integer coefficients.

Finding the best entropy encoding

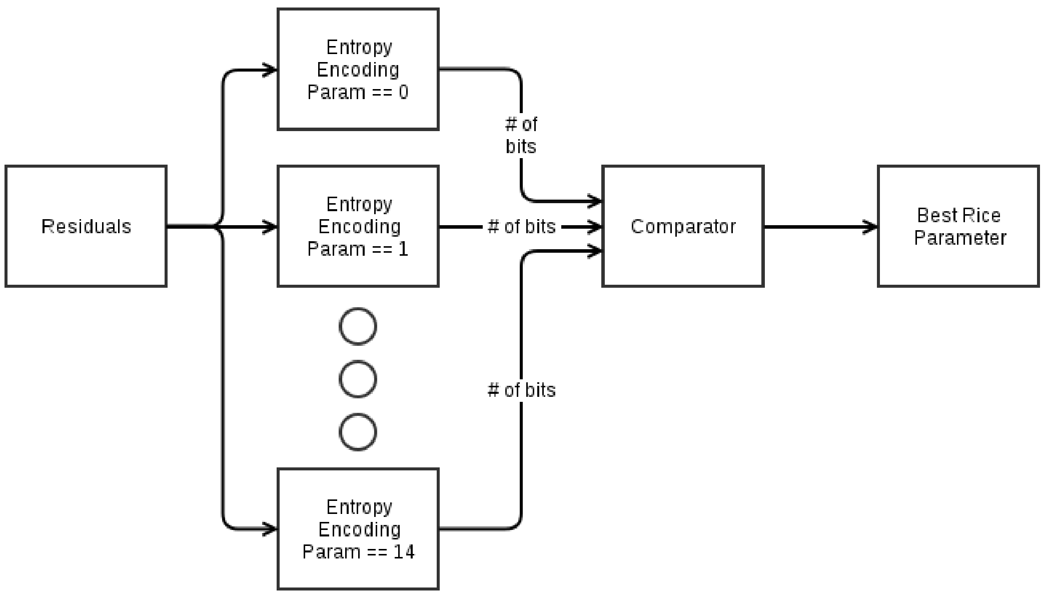
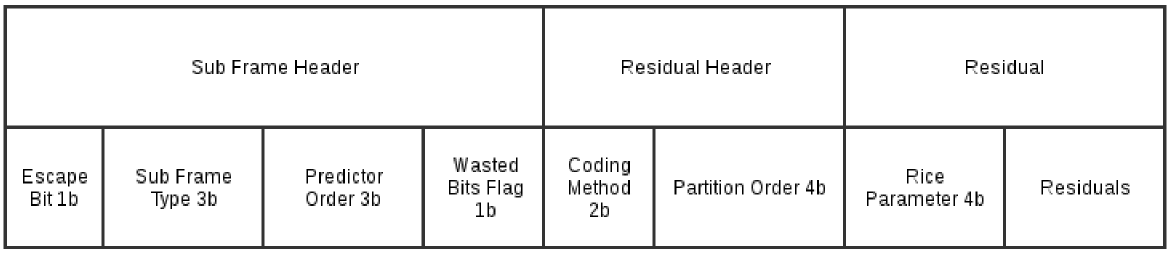

Residuals
Entropy Encoding Param == 0
Entropy Encoding Param == 1
Entropy Encoding Param == 14
# of bits
# of bits
# of bits
Comparator
Best Rice Parameter

Figure 5: Finding the best entropy coding

Once the best linear predictor has been found, the encoder calculates the residuals (the difference between the model and the actual audio samples) and sends them through to the entropy encoder. Whilst there are a number of different entropy encoding methods available, including Huffman coding (used in JPEG[?]), arithmetic coding, and Golomb codes, fLaC uses a subset of the Golomb codes known as Rice coding. Rice coding is used because it is very easy to implement on binary machinery, as it consists only of bit shifts. Rice coding consists of taking a binary number x, dividing it by the rice parameter K, encoding the quotient in unary (n zeros followed by a one), and encoding the remainder as K binary bits. The advantage of this method is that so long as the majority of the residuals are less than the rice parameter K, a net reduction in encoded residual size will result.

In order to optimize the rice parameter, the fLaC algorithm allows the parameter to vary within a block of residuals. Each *partition* within a block may have a different rice parameter, with as many partitions as there are residuals allowed. A prefix sum algorithm is used to find the optimum combination of partition size and rice parameter allocation.

Encoding and Framing



Sub Frame Header				Residual Header		Residual	
Escape Bit 1b	Sub Frame Type 3b	Predictor Order 3b	Wasted Bits Flag 1b	Coding Method 2b	Partition Order 4b	Rice Parameter 4b	Residuals

Figure 6: Typical framing of a Subframe header

The final stage of the encoder is to encode and frame the audio data. The data is run through the best discovered model to produce the residuals. The residuals are then encoded using the best rice parameters. Finally, the resulting data, parameters, model coefficients, and audio file information (e.g. sample rate) is framed according to the fLaC format and written out to a file buffer, usually with significant compression having been achieved.

2.1.2 fLaC Decoding Process

The decoding process is much easier than the encoding process, as all the appropriate model parameters are given and the decoder simply has to run the given data through the appropriate model to get the audio samples out. The decoding process is done frame by frame.

3 Requirements

3.1 Changes to original requirements

Whilst the original requirements specified a fully fLaC compatible encoder and decoder, I have decided to reduce the goal to a subset of the fLaC specification. The new target is to decode single channel 16 bit fLaC files. Whilst it would not be a major leap to add stereo support, I believe that there is little to be gained from a theoretical perspective by doing so and would prefer to limit the problem domain in order to focus on the interesting areas, rather than concern myself with implementing extra but not necessary features.

4 Progress To Date

4.1 Software fLaC Decoder

A C++ fLaC decoder was written over the course of the semester. The decoder can successfully decode 16 bit stereo fLaC encoded files. Implementing the decoder provided valuable knowledge about the internal workings of the fLaC decoding process which proved very useful when implementing the hardware decoder.

4.2 Software fLaC Encoder

A C++ fLaC encoder was written over the course of the semester. This encoder can successfully encode 16 bit mono WAVE format files. It is significantly slower than the fLaC reference encoder, but since the software encoder was only written to provide

4.3 Hardware fLaC Decoder

A hardware fLaC decoder was written using Verilog and tested in Modelsim. It did take me a while to get an initial version of the hardware decoder working due to my unfamiliarity with Verilog and writing hardware description code for FPGAs. Implementing the hardware decoder gave me insight into the bottlenecks of the decoding algorithm in hardware. The main bottleneck turned out to be applying the linear model to the residual data. This consists of a multiply accumulate block. However, pipelining this block solves the problem

4.4 Potential Speedups

4.4.1 Hardware Decoder

- Pipeline the Fixed and LPC decoding modules
- Handle more bits per cycle with the residual decoding module
- Decode multiple frames per cycle

4.4.2 Hardware Encoder

- Pipeline the Fixed and LPC encoding modules
- Run Fixed and LPC models in parallel
- Encode multiple residuals each cycle
- Encode multiple frames per cycle

References

[1] Kevin De Vorsey and Peter McKinney. Digital preservation in capable hands: Taking control of risk assessment at the national library of new zealand, Spring 2010. Copyright - Copyright National Information Standards Organization Spring 2010; Document feature - Illustrations; ; Last updated - 2015-09-03.

[2] Library of Congress. Wave audio file format, 2013.

[3] Tony Robinson. Shorten: Simple lossless and near-lossless waveform compression. *Cambridge University Engineering Department*, 1994.

[4] M. Hans and R. W. Schafer. Lossless compression of digital audio. *IEEE Signal Processing Magazine*, 18(4):21–32, Jul 2001.