Project Document

Team member	Student Number
Ilia Sosner	49425259
Cole Shanks	54950860
Guoyu Zhao	65442865
Reuben Singh Joginder	89291884

Team Number: 8
Instructors: Dr. Paul Lusina and Dr. Cristian Grecu
Teaching Assistants: Hamed Noori and Fredy Alves
Course: ELEC 391
Document: Project Report
Total Number of Pages: 13
Date: 17th June, 2020

Table of Contents

Objectives, Requirements and Constraints

Objectives

- To transmit data over a simulated channel and recover the data on the receiving end.
- To achieve high data transfer reliability.
- To achieve low latency, especially for audio transmission.
- To keep complexity low where possible.

Functional Requirements

- Transmit 2 data types: typical speech audio, and a text file.
- Use compansion[1] techniques to reduce the amount of data going over the channel.
- Use an error encoding/correcting algorithm to detect and correct errors that have occurred in the transmission.
- Modulate data bits into symbols prior to transmission in a way that is relatively resistant to errors caused by channel noise.
- Simulate a multi-path channel with AWGN noise and attenuation.
- Use a transmitter/receiver design that archives the required bitrate.
- Recover data from multipath received signal.
- Store the received text data in memory and playback the received audio data through the onboard audio codec.

Non-functional requirements

- Transmit audio with no noticeable latency and sufficient quality to understand typical human speech.
- Transmit text data files with an industry standard bit error rate.

Constraints

- Stay within the provided power spectral mask and total output power.
- Stay within the resources available on the DE1-SOC development board.

Stretch Goals

- Huffman Encoding for text compression and decompression.
- Optimizations to reduce latency.

[1] Companson is commonly used to express “Compression” and “Expansion” or “Decompression”.

Communication System Design

System Overview

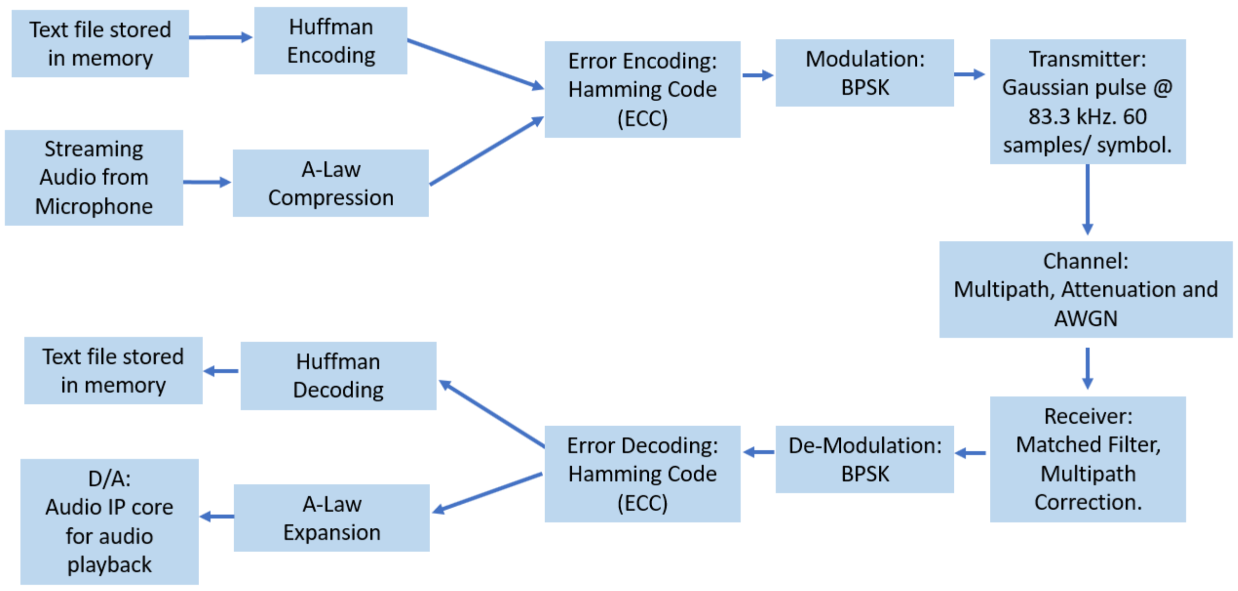

Text file stored in memory
Huffman Encoding
Streaming Audio from Microphone
A-Law Compression
Error Encoding: Hamming Code (ECC)
Modulation: BPSK
Transmitter: Gaussian pulse @ 83.3 kHz. 60 samples/ symbol.
Channel: Multipath, Attenuation and AWGN
Receiver: Matched Filter, Multipath Correction.
De-Modulation: BPSK
Error Decoding: Hamming Code (ECC)
Huffman Decoding
Text file stored in memory
A-Law Expansion
D/A: Audio IP core for audio playback

Fig-1: High-level system overview.

In order to achieve our objective of reliable data transfer we have made a few high level design choices. For transmission reliability (measured in Bit Error Rate[2]) there are two main design considerations. The first of these is the modulation scheme, for which we have chosen to implement Bit Phase Shift Keying[3]. This type of modulation provides the largest euclidean distance between symbols on the complex plane and is therefore more resistant to noise when compared to alternatives such as 4 PSK, 8PSK and other similar methods. This choice comes at the expense of fewer bits per transmitted symbol.

The second consideration for data reliability is error encoding/decoding. We chose to use a Hamming Code type error encoding that is able to detect and correct corrupted bits. The tradeoff of using this method is that we need to add redundant data to the transmission packet, thereby reducing useful data throughput. To minimize the data that needs to be sent over the channel, we chose to use A-law compression for audio due to its simplicity and relatively good compression ratio (3:1). For text data we chose Huffman encoding due to its efficacy in compressing ascii text compared to other common compression algorithms such as Lempel-Ziv or Run Length Encoding. For transmitter and receiver we chose to use a gaussian pulse and matched filter because it was the only

[2] Abbreviated as “BER”.
[3] Abbreviated as “BPSK”.

pulse shape that could achieve our required data rate while staying under the power spectral mask (see FigA-3 in Appendix).

Taking into account all of the above design choices and system considerations we believe our system will be able to achieve the following specifications shown in table 1.

Target Bit Error Rate @ Eb/No of 3dB	10^-9. This is a commonly accepted BER for telecommunications.
Bandwidth Resources	Gaussian pulse at 83.3 khz fills our allotted bandwidth perfectly and allows for 83.3 kbps overall transmission rate.
Transmission rate (Useful data)	64 kbps (compressed audio rate).
Power Usage	Transmitted pulse power is recovered from multiple paths. Transmitter only enabled when data is available
Latency	0.25ms seconds for audio transmission. Text data latency depends on file size but is transferred at a minimum of 8k characters per second with a minimum latency of 0.25 ms
Audio Bandwidth	0-4000 HZ

Table-1:Overall system specifications.

Source Modelling

Audio Source

Our audio signal is supplied from an outside source such as mobile phone or computer. The audio file is preprocessed with a low pass filter at 4khz to ensure it fits within the specified audio transmission bandwidth. The signal is fed into the onboard codec and sampled by the audio IP core at 84 kHz (2 channels at 24-bits). We then down sample it to 8.4 kHz by transmitting every tenth sample. The two channels provided by the codec are divided by 2 and summed into a single channel for compression, error encoding and transmission. Some specifications are highlighted in the table below.

Target BER at sink	10*E-4 (Commonly accepted BER for audio)
Tolerable Latency	2.5*E-4 seconds
Source Signal Bandwidth	20Hz to 4kHz (typical speech)
Format	Analog signal at the line-in port

Table-2:Audio specifications.

Text Source

The source text is stored in a memory block on the fpga. It’s contents are initialized from a Memory Initialization File on compilation. The memory block holds 4096 8-bit ascii characters. Some of the specifications are highlighted in the table below.

Target BER at sink	10*E-9% (Commonly accepted BER for telecommunications)
Tolerable Latency	0.6 seconds for the 4096 character message.
File Format	.txt file converted to hex and then to .mif

Table-3: Text specifications.

Compression Encoding/Decoding

Audio Compansion

We considered A-law, Mu-law and Linear Predictive Coding[4]. The implementation of A-law/Mu-law has lower complexity compared to LPC, in both simulation and FPGA, provided a linear approximation is used. A-law causes lesser distortion on low amplitude signals compared to Mu-law and is suitable for human speech. This compression algorithm works by re-quantizing the audio sample in a non-linear (logarithmic) fashion in order to preserve a greater dynamic range in fewer bits. Choosing to use A-law allows us to compress 24-bit audio samples down to 8 bits for transmission, thereby providing a compression ratio of 3:1. This compression ratio helps us to meet our audio latency goals and transmission bit-rate constraints.

Text Compansion

Our text compression algorithm had to be lossless. We narrowed down to the following algorithms: Lempel-Ziv, Run-length encoding, and Huffman Encoding.

The Huffman code is a prefix-free, variable-length code which is capable of achieving the shortest possible code word length for a particular symbol. This algorithm is done by assigning the most occurring symbol by the least number of bits up to the least occurring symbol with the most number of bits. Since Huffman Encoding produces binary codes that are variable in length (unlike Lempel-Ziv), higher compression ratios can be achieved, which generates smaller packet sizes. This allows for a higher useful data throughput. Compared to Run Length Encoding, some simplifying assumptions (such as a fixed number of buffered values) allow for a lower latency implementation with lesser FPGA memory resources used.

Huffman Compansion becomes difficult to decode due to variable lengths of data, resulting in lower compression ratios and slower compression speeds compared to some lossy encoding techniques. However in the context of our project, the advantages outweigh the disadvantages.

[4] Abbreviated as “LPC”.

Error Correction Encoding / Decoding

The error correction code that we decided to implement in our design is the ALTECC IP core included within the Quartus library. This is a type of forward error correction which means that the receiver can detect and correct errors without having to send information back to the sender.

The coding scheme used to encode the data is the Hamming Code scheme. Hamming encoding is a method of attaching redundant bits to the data in order to tell when an error has occurred. It is more sophisticated than a simple parity check but it applies the same idea. In a simple parity check, a parity bit is appended to the end of a string of bits and is either 0 or 1 depending on the even or oddness of the bits. If the previous bits have an odd number of ones, this bit will be set to 1. If the previous bits have an even number of ones, then it will be set to 0. In this sense, if the receiver receives an odd number of ones it immediately knows that an error has occurred. This breaks down when there is more than a single bit error in the encoded message. However, the receiver does not know where the error has occurred and has no way to correct the message. Hamming codes are a way of placing parity bits strategically throughout the message so that the receiver can identify that an error has occurred and also where it has occurred. This is done by placing the parity bits at the powers of 2 throughout the message.

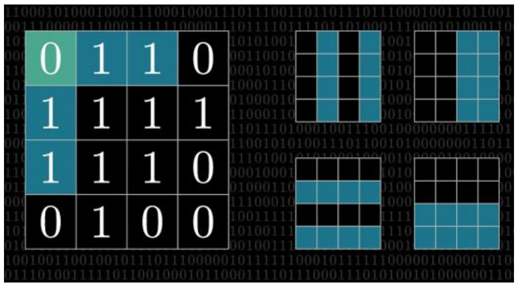


Fig-2: Hamming code Placement

So if the message is treated as a square block of bits, parity bits would be placed at 1, 2, 4, 8, 16, and so forth along the columns and the rows. Then, by a simple check of the parity corresponding to the sectioning off of the columns and rows based on these locations, a single bit error can be located anywhere in the message. This scales so that the number of redundant bits attached to large messages is very low. This is referred to as the code rate.

We chose this method of error correction for a number of reasons.

- It's forward-error correcting. It does not require communication back to the transmitter and can correct errors automatically.
- Simple and efficient. We don't expect large burst errors as a consequence of noise. Small distributed errors can be corrected effectively.

- Good relative code rate. Relatively low number of redundant bits being transmitted across the channel.
- Minimal latency. The ALTECC core is combinational with some buffering before the input.

We are using a Hamming (72, 64) code. This means that a 64-bit message is sent to the encoder. The encoder takes this message, attaches 8 parity bits, and sends the 72-bit encoded message across the channel. 7 of these bits allow for correction of single-bit errors and the 8th bit placed at location zero allows us to detect 2-bit errors although we cannot correct them. This gives us a code rate of 0.11 or 11% redundant bits transmitted over the channel.

Modulation and Demodulation

We chose to use BPSK encoding for two major reasons. First, out of all the alternatives it is the most resistant to corruption by noise as it has the greatest euclidian distance between symbols [5]. Secondly, it reduces the complexity of an fpga implementation by a large amount as compared to QPSK, 8-PSK, as the symbols lie on the real axis and therefore imaginary number representations do not need to be implemented on the fpga. This comes at a cost of a much lower bit rate, all other things being equal. A BPSK symbol can only represent one bit as compared to 4 bits for 8-PSK. After optimizing our transmitter for the highest possible pulse rate, our overall maximum bit-rate over the channel with BPSK is 83.3 kbps. This meets our requirements of 72 kbps for an error-encoded and compressed audio stream. An alternate method (perhaps 8-PSK) could be used to increase the bit rate and improve audio quality at the expense of data transfer reliability.

		Advantage	Disadvantage
Current System	BPSK	Better noise handling	Fewer bits/symbol, worse audio quality.
Possible Alternative	8-PSK	More bits/symbol, better audio quality.	Worse noise handling

Table-4: Comparison Between BPSK and 8-PSK

[5] Due to this, BPSK is a power efficient modulation technique as less power is needed to transmit the carrier with less number of bits.

Transmitter and Receiver

We did a frequency analysis of three different pulse types (rectangular, triangular and Gaussian[6]). The goal was to find the maximum pulse frequency for which the frequency content of the pulse still fit under the power spectral mask. We determined that the maximum rates are as shown in Table-5, and that the gaussian pulse was the best choice.

Pulse Shape	Max symbols/sec
Rectangular	1,400
Triangular	24,000
Gaussian	83,300

Table-5: Maximum Symbols/Sec for Pulse Shapes

For the receiver we use a matched filter (same shape as the attenuated transmit pulse) to filter out noise and recover the original signal. The output of the matched filter is sampled at the symbol frequency and processed with delays and some scaling to recover the original signal from the multipath combination. We are using 60 samples to represent the pulse and running 10 clock cycles per sample (50 MHZ clock). This gives us a real time pulse rate of 83.3 kHz.

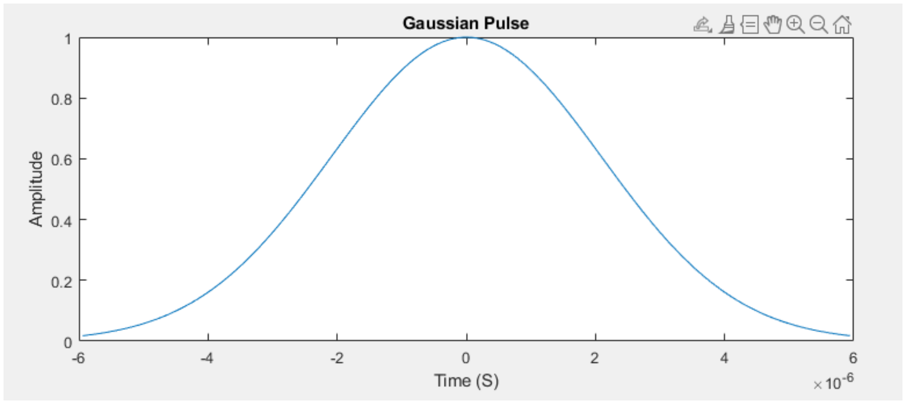


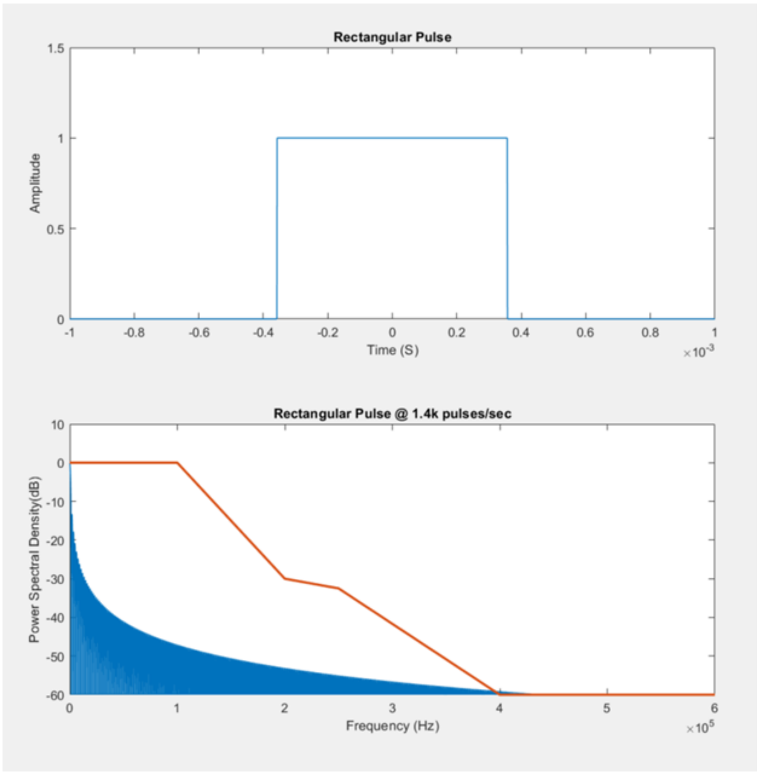
Fig-3: The gaussian transmit pulse.

[6] See appendix FigA-1, FigA-2 and FigA-3 for frequency power spectrum analysis for the three Pulse types.

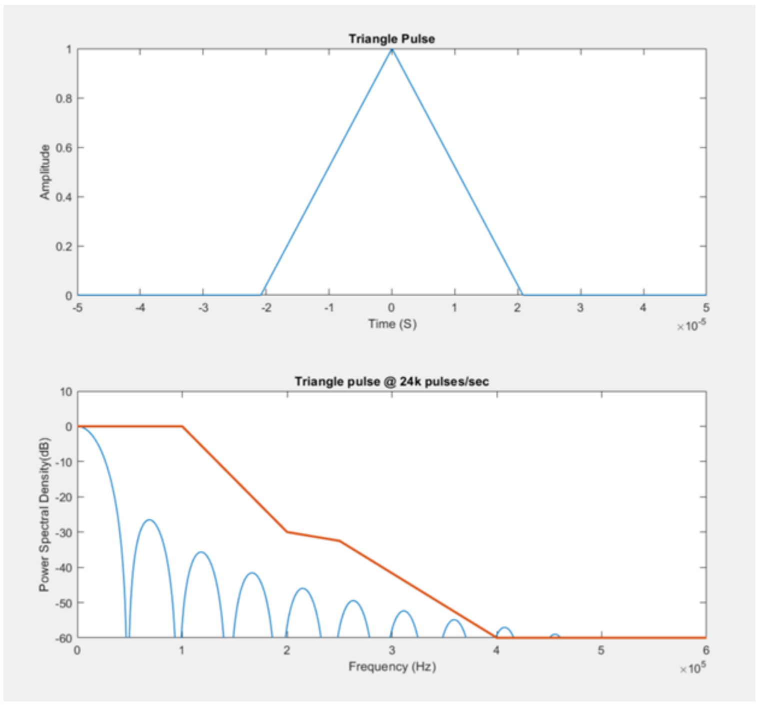
Appendix A

Deliverable	File name	Notes (optional)
Product Document	Product_Document.pdf	
Product Presentation	Product_Presentation.pdf	
Simulink System File	Final.slx	
HDL source code, including testbenches and readme files.	Folder Named HDL	

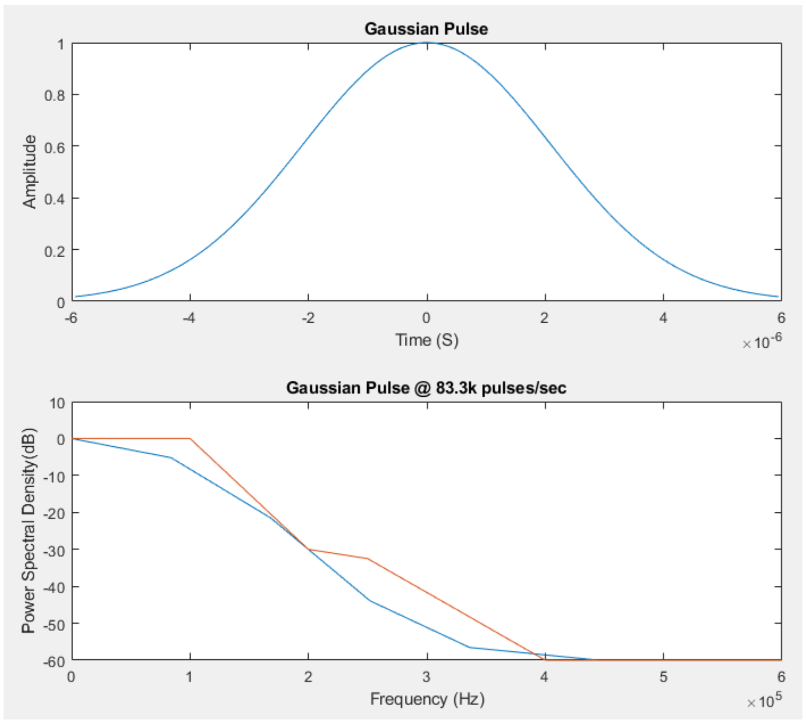
Technical Appendix



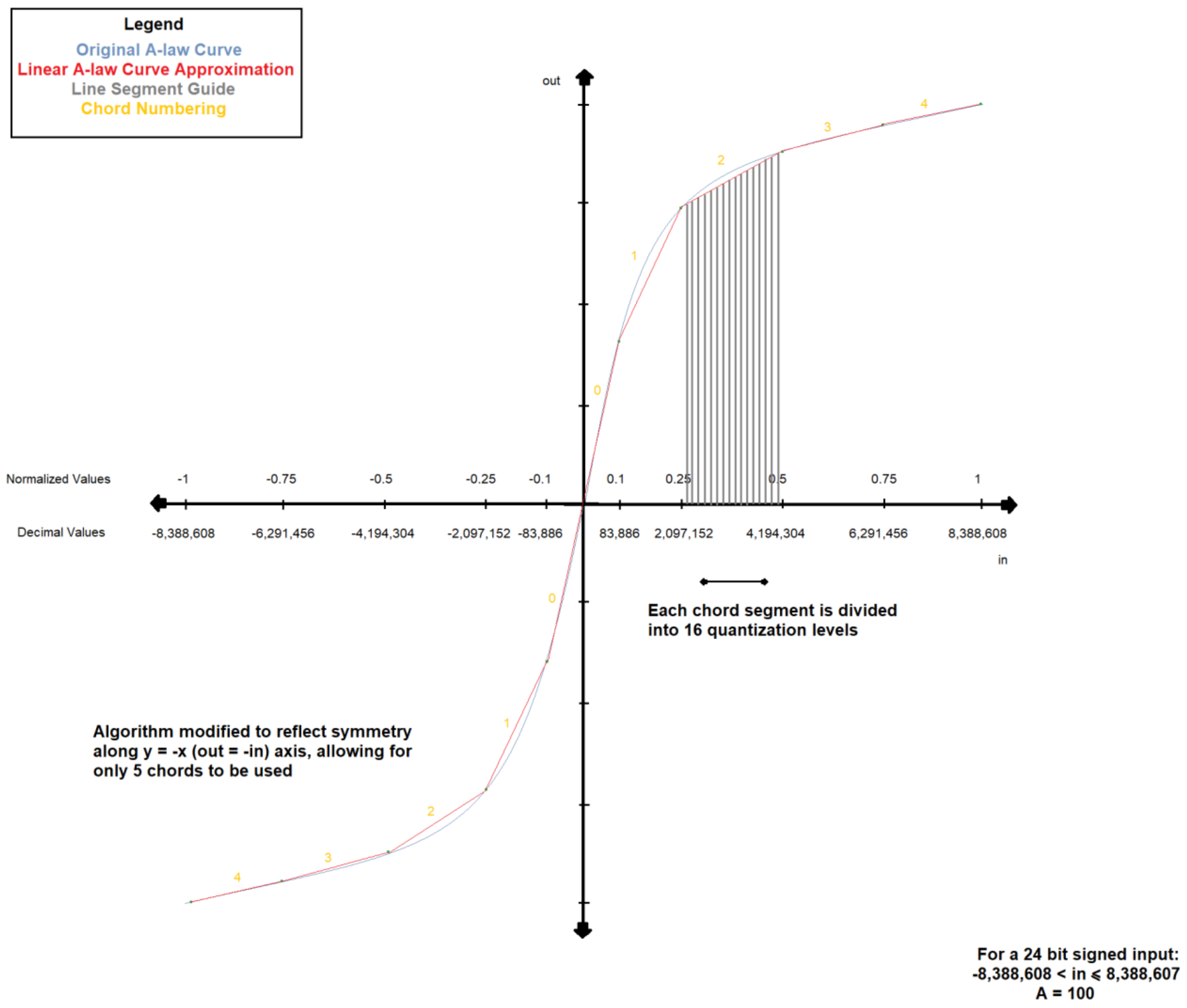
FigA-1: Power Spectrum Analysis of a Rectangular Pulse



FigA-2: Power Spectrum Analysis of a Triangle Pulse.



FigA-3: Power Spectrum Analysis of a Gaussian Pulse.



FigA-4: A-law Compansion Linear Approximation Curve.