International Journal of Engineering and Advanced Technology (IJEAT)
ISSN: 2249-8958 (Online), Volume-1 Issue-1, October 2011

Voice Data Compression and Decompression

Anubhuti Khare, Manish Saxena, Bhagawati Patil

Abstract— ***An efficient and minimum hardware implementation for the Voice data compression and decompression will be presented in this paper. Voice data compression and decompression is about a process which reduces the data rate or file size of digital audio signals. This process reduces the dynamic range (withoutchanging the amount of digital data) of audio signals [1]. The Huffman coding is used to have lossless audio compression. Very High Speed Integrated Circuit Hardware Description Language (VHDL) is used for to code the Huffman encoder and decoder and Actel's ProASIC kit is used for hard ware implementation of it. This system is minimal model of real time audio compression and decompression system.***

Index Terms— ***FPGA, VLSI, ADC, Huffman Coding.***

I. Introduction

The importance of compression and decompression was established in discussions of Wireless Transport Layer Security (WTLS) [5]. After carefully going through the available lossy compression technique, it became clear that the decompressed data may be different from the original data. Typically, there is some distortion between the original and reproduced signal. As in lossy compression the decompression produces approximately the original data. It is suitable when exact accuracy is not required, like in image and audio processing. Decompression is deterministic (data is lost when compressing). Lossy compression produces smaller output. So we are going to recommend the use of lossless compression technique. [4]

A. *Compression: The aim of data compression is to represent an information source (e.g. a data file, a speech signal, an image, or a video signal) as accurately as possible as using the fewest number of bits. A stream is either a file or a buffer in memory.*

It is difficult to maintain all the data in an audio stream and achieve substantial compression. First, the vast majority of sound recordings are highly complex, recorded from the real world. As one of the key methods of compression is to find patterns and repetition, more chaotic data such as audio doesn't compress well. In a similar manner, photographs compress less efficiently with lossless methods than simpler computer-generated images do [9].But interestingly, even computer generated sounds can contain very complicated waveforms that present a challenge to many compression algorithms. This is due to the nature of audio waveforms, which are generally difficult to simplify without a (necessarily lossy) conversion to frequency information, as performed by the human ear. The second reason is that values of audio samples change very quickly, so generic data compression algorithms don't work well for audio, and strings of consecutive bytes don't generally appear very often. However, convolution with the filter [11] (that is, taking the first difference) tends to slightly whiten (décor-relation, make flat) the spectrum, thereby allowing traditional lossless compression at the encoder to do its job; integration at the decoder restores the original signal. Codecs such as FLAC, Shorten and TTA use linear prediction to estimate the spectrum of the signal. At the encoder, the estimator's inverse is used to whiten the signal by removing spectral peaks while the estimator is used to reconstruct the original signal at the decoder[11].

Lossless audio codecs have no quality issues, so the usability can be estimated by

Speed of compression and decompression
Degree of compression
Software and hardware support
Robustness and error correction

Two basic categories of data compression are lossless and lossy compression. Lossless data compression method will not cause any loss of data or errors while during a lossy type process some data are changed or lost.

B. *Types of compression:*

There are two compression techniques commonly used

1. Lossless compression
2. Lossy compression

C. *Huffman Codes:*

The Huffman procedure is based on two observations regarding optimum prefix codes:

1. In an optimum code, samples that occur more frequently (have a higher probability of occurrence) will have shorter codeword than samples that occur less frequently.

If samples that occur more often had codeword that were longer than the codeword for samples that occurred less often, the average number of bits per sample would be larger than if the conditions were reversed.
Therefore, a code that assigns longer codeword to samples that occur more frequently cannot be optimum.

2. In an optimum code, the two samples that occur least frequently will have the same length.

Manuscript published on 30 October 2011.
* Correspondence Author (s)
Dr. Anubhuti Khare*, Reader, Department of Electronics and Communication, University Institute of Technology, Rajeev Gandhi Technical University, Bhopal (M.P.), India, (Email:-anubhutikhare@gmail.com)
Manish Saxena, Head of Department of Electronics and Communication, Bansal Institute of Science and Technology Bhopal (M.P.), India, Mobile: +919826526247 (Email:- manish.saxena2008@gmail.com)
Bhagawati C.Patil, Mtech (Digital Communication), Bansal Institute of Science and Technology Bhopal (M.P.), India (email-bhagawati_patil8@rediffmail.com).

To check the validity of this condition consider two least probable codeword violating this condition i.e. the longer codeword is,k‟ bits longer than the shorter codeword.

Since this is a prefix code, the codeword cannot be a prefix of the longer codeword. This means that even if last k bits of the longer codeword are dropped, the two codeword will still be distinct.

As these codeword correspond to the least probable samples, no other codeword can be longer than these code words; therefore, there is no danger that the shortened codeword would become the prefix of some other codeword.

The necessary conditions for an optimal variable length binary code are as follows:

1. Given any two letters aj and ak, if P[aj] ≥ P[ak], then lj≤lk where lj is the number of bits in the codeword for a j.

2. The two least probable have code words with the same maximum length lm.

3. In the tree corresponding to the optimum code, there must be two branches stemming from each intermediate node.

All these conditions are satisfied by Huffman code and therefore it is optimal variable binary code.

D. *Length of Huffman code:*

The length of Huffman code depends on following things:

i) Size of the alphabet

ii) Probabilities of individual letters

The average code length „l' is bounded by following condition,

$H(S) \leq l < H(S) + 1$

where H(S) = entropy of sample S.

The condition for uniquely decodable code given by Kraft‟ s inequality is as follows:

$\Sigma\, 2^{-li} \leq 1$

Where i = 1, 2, 3……….. k

k = code words of length l

E. *Certain important properties of Huffman code:*

1. A fast and reasonable memory efficient Huffman coder
2. File storage and communications bandwidth have become less expensive and more available
3. No quality issues
4. 100% lossless coding technique.

Figure-1. Shows the following symbols listed with a probability of occurrence where A=30%, B=25%.C=205,D=15%, E=10%.

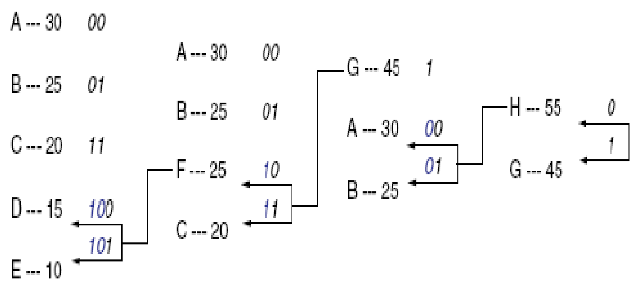


Fig. 1: Huffman Coding Procedure

F. *Steps for Huffman coding:*

1. Adding the two least probable symbols gives 25%. The new symbol is F

2. Adding the two least probable symbols gives 45%. The new symbol is G

3. Adding the two least probable symbols gives 55%. The new symbol is H

4. Write "0" and "1" on each branch of the summation arrows. These binary values are called branch binaries.

5. For each letter in each column, copy the binary numbers from the column on the right, starting from the right most columns (i.e., in column three, G gets the value "1" from the G in column four.) For summation branches, append the binary from the right-hand side column to the left of each branch binary. For A and C in column three append "0" from H in column four to the left of the branch binaries. This makes a "00" and B "01". Completing step 5 gives the binary values for each letter: A is "00", B is "01", C is "11", D is "100", and E is "101". The input with the highest probability is represented by a code word of length two, whereas the lowest probability is represented by a code word of length three.

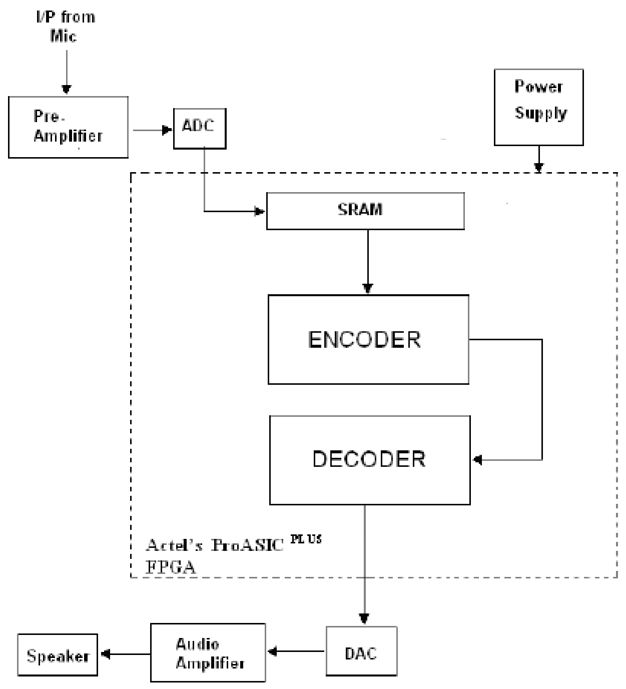


Fig.2.1 Block Diagram of Voice Data Comp. & Decomp.

II. Block Diagram Description

For voice data compression a speech template is taken as an input through mice. When sound is played into a microphone, it is converted into a voltage that varies continuously with time. This analog voltage is given as an input to the Pre-amplifier for boosting its level. To have compatibility with the FPGA used, this analog output is given to an Analog to Digital Converter to obtain corresponding Digital samples. To take digital samples in appropriate sequence, we store samples into a SRAM using VHDL coding. This data will be compressed using Huffman coding using the hardware developed in FPGA by writing the appropriate routines. The compression achieved will be indicated on 7 segment

LED display. The compressed data is decoded in FPGA. The decompressed samples are given as an input to DAC so as to get its analog equivalent. This block will accept a stream of bits transmitted by Encoder. As it receives next bit, it compares the bit pattern with that in the look-up table. If it finds that bit pattern, it decodes the sample else it accepts next bit. An Audio Amplifier is used for conditioning and amplifying the Analog output of DAC which is reproduced using Speaker.

2.1 ENCODER:

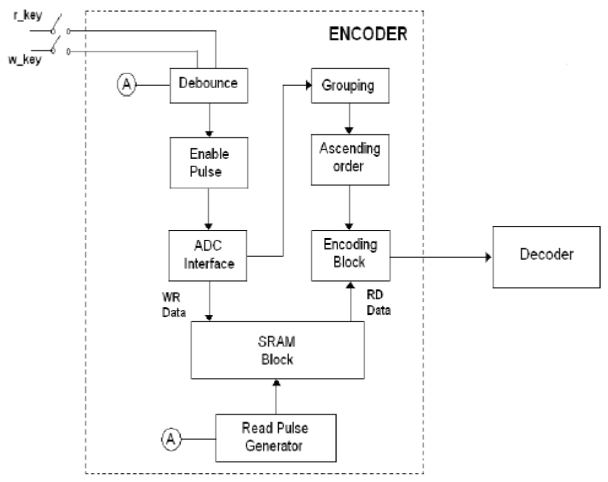


Fig.2.2 Encoder

2.1.1 Keys:

A. w_key : This key will trigger the subsequent blocks, after which the ADC will start taking the samples of the speech template uttered by the user.

B. r_key : After pressing this key, the actual process of decoding will start and we will hear the decoded speech template from the speaker.

2.1.2 Debounce:

This block avoids false triggering due to glitches. This block gives a pulse output after it detects high period of the input keys for a specified count.

2.1.3 Enable pulse:

As the duration of speech template is restricted to 1sec, this is decided by this block. This block will output a pulse of 1 sec during which the ADC will start sampling, and as the output of this block goes low after 1 sec, the ADC block will be deactivated.

2.1.4 ADC Interface:

This block deals with handshake signals with the actual ADC. It gives the start of conversion (convst) signal to the ADC and after detecting the end of conversion (EOC) pulse it will select the device (cs) and read the converted data (rd).

2.1.5 SRAM:

The SRAM we are using is Asynchronous SRAM. Mainly it contains two blocks:

2.1.5.1 SRAM: It will generate actual SRAM in the FPGA.

2.1.5.2 SRAM interface: This block will take care of writing and reading the samples to and from the generated SRAM. As shown in the figure,the interface block is divided into twoparts viz.,

 SRAM write: This block will write each sample value to the SRAM when ADC triggers this block after it finishes the conversion of each sample.

 SRAM read: This block will read back the sample values from the SRAM at an interval of 125µsec when read pulse generator block will triggers it.It will give these samples to Encoder to encode the samples.

2.1.6 Read pulse generator:

It will generate pulses for reading data from SRAM. This block is triggered by r_key

2.1..7 Grouping:

This block will count the number of times each sample occurs.

2.1.8 Ascending:

Input to this block will be the frequency of occurrence of each sample. This block will sort the frequencies in ascending order. It will output the samples sorted according to their frequencies.

2.1..9 Encoding:

It will encode each sample that it will read from SRAM i.e. it will assign a codeword according to the Huffman algorithm.

2.2 DECODER:

This block will accept a stream of bits transmitted by Encoder. As it receives next bit, it compares the bit pattern with that in the look-up table. If it finds that bit pattern, it decodes the sample else it accepts next bit.

Input Stream

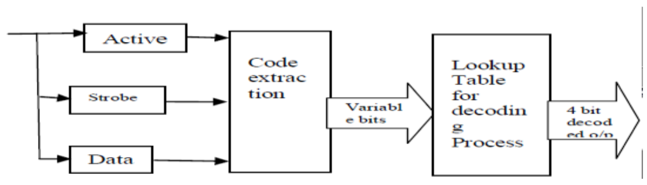
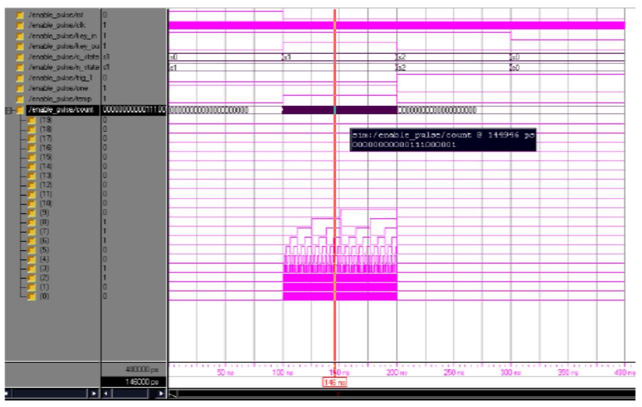


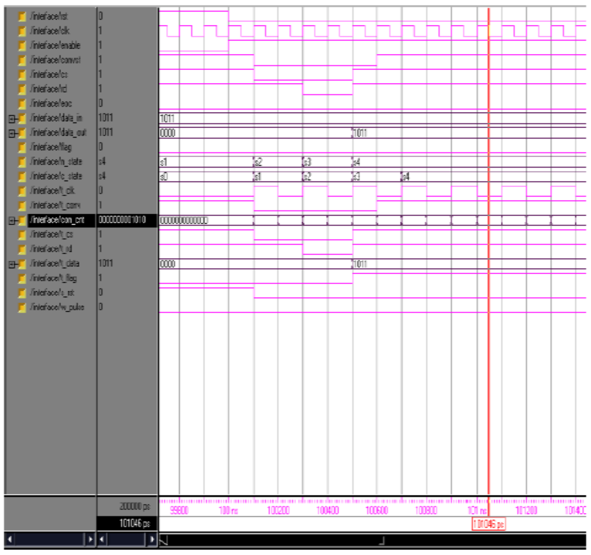
Fig.2.4 Decoder

III. RESULTS

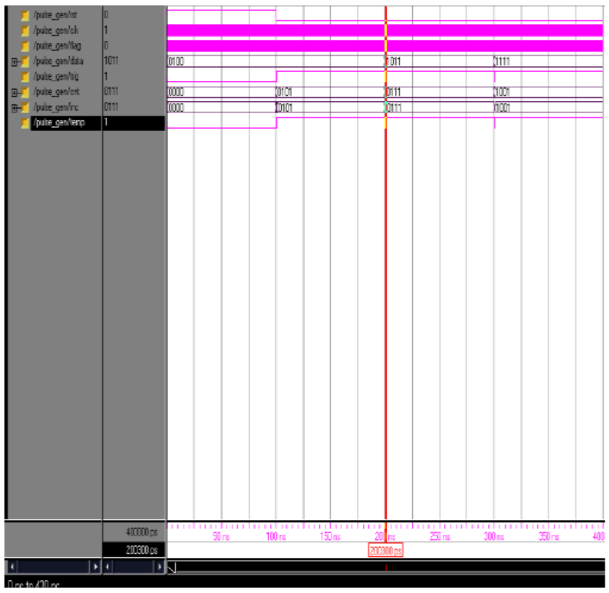


3.1. Enable Pulse:

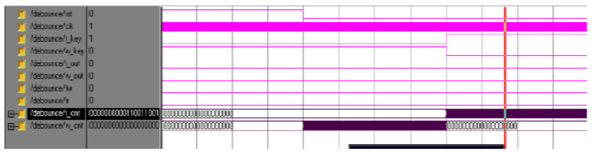
3.2. Interfacing of 7822:

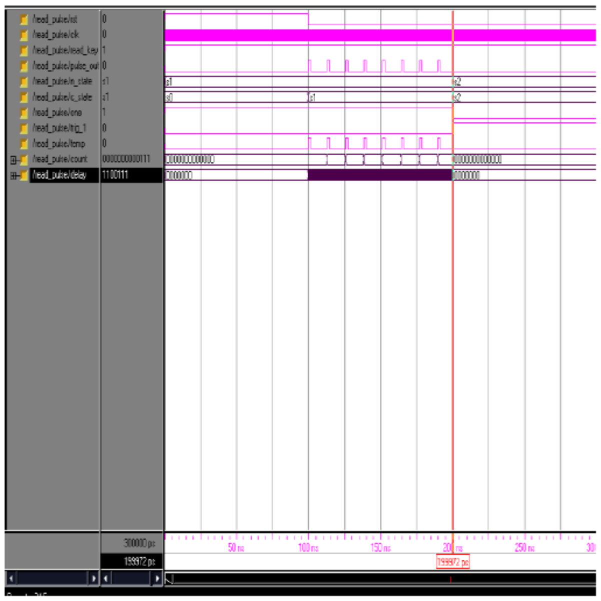


3.3.Pulse Generater :



3.4. Debounce:





3.5. Read pulse:

IV. CONCLUSION

This paper discussed the design and implementation of Huffman encoder- decoder in hardware for voice data compression and decompression. Previous research in this field has been limited to hardware implementations of Huffman encoders for Text data. We have described a compression technique that works by applying a Huffman coding to a sample of voice data to make redundancy in the input more accessible to simple coding schemes. This algorithm is general-purpose, in that it does well on both text and non-text inputs. The transformation uses sorting to group digital voice data together based on their probabilities; this technique makes use of the probabilities on only one side of design. The effectiveness of the algorithm continues to improve with increasing voice data storage block size. Our algorithm achieves compression comparable with good statistical model, our algorithm decompresses faster than it compresses.

Since this implementation is written in VHDL, it is fully portable to a variety of hardware architectures.As FPGA technology improves; the performance of the hardware implementation of Huffman will improve without the need to change the design. Also it utilizes full Actel FPGA.

ACKNOWLEDGMENT

Mrs. Anubhuti Khare one of the authors is indebted to Director UIT RGPV Bhopal for giving permission for sending the paper to the journal. Manish Saxena is also thankful to the Chairmen, Bansal Instituteof Science & Technology Bhopal for giving permission to send the paper for publication. Last but not least, I would also like to thanks our HOD and colleagues for supporting us.

REFERENCES

1. FPGA based architecture of MP3decoding core for multimedia systems.
2. C. Murthy and P. Mishra. Bit mask-based control word compression for NISC architectures. In Proceedings of ACM Great Lakes Symposium onVLSI (GLSVLSI), 2009.
3. K. Basu and P. Mishra. A novel test-data compression technique using application-aware bit mask and dictionary selection methods. In Proceedings of ACM Great Lakes Symposium on VLSI (GLSVLSI),pages 83–88, 2008.
4. S. Seong and P. Mishra. Bit mask-based code compression for embedded systems. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 27(4):673–685, April 2008.
5. B. Gorjiara and D. Gajski. FPGA-friendly code compression for horizontal micro coded custom IPs. In Proceedings of Field Programmable Gate Arrays (FPGA), 2007.
6. S. Seong and P. Mishra. An efficient code compression technique using application-aware bit mask and dictionary selection methods. In Proceedings of Design Automation and Test in Europe (DATE), pages 582–587, 2007.
7. M.-B. Lin, J.-F. Lee, and G. E. Jan, “A lossless data compression and decompression algorithm and its hardware architecture,” IEEE Trans.Very Large Scale Interation (VLSI) Syst., vol. 14, no. 9, pp. 925–936,Sep. 2006.
8. K. Pagiamtzis and A. Sheikholeslami, “Content-addressable memory (CAM) circuits and architectures: A tutorial and survey,” IEEE J. Solid-State Circuits, vol. 41, no. 3, pp. 712–727, Mar. 2006.

9. S. Seong and P. Mishra. A bit mask-based code compression technique for embedded systems. In Proceedings of International Conference on Computer-Aided Design (ICCAD), pages 251–254, 2006.
10. V. Sklyarov, I. Skliarova, B. Pimentel, J. Arrais, "Hardware/Software Implementation of FPGA-Targeted Matrix-Oriented SAT Solvers",Proceedings of the 14th International Conference on Field-Programmable Logic and Applications –FPL‟2004, Antwerp, Belgium,August/September, 2004, pp. 922-926.
11. V. Sklyarov, "Hierarchical Finite-State Machines and Their Use for Digital Control", IEEE Transactions on VLSI Systems, Vol. 7, No 2,1999, pp. 222-228.
12. B. Pimentel, J. Arrais, "Implementación de Algoritmo de Compressão Descompressão de Dados para Modelo de Coprocessamento basead on FPGA‟s", Electrónica e Telecomunicações, vol. 4, no. 10, Jan. 2004, pp. 215-220.
13. V. Sklyarov, "FPGA-based implementation of recursive algorithms",Microprocessors and Microsystems, Special Issue on FPGAs:Applications and Designs, 2004, vol. 28/5-6 pp 197-211.
14. M. Guthaus, J. Ringenberg, D. Ernst, T. Austin, T. Mudge, and R.Brown. MiBench: A free, commercially representative embedded benchmark suite. In Proceedings of International Workshop onWorkload Characterization (WWC), 2001.
15. NiosEmbeddedProcessor http://www.altera.com/products/devices/excalibur/excniosindex.html. http://www.xilinx.com/partinfo/databook.htm.
Xilinx Micro Blaze. http://www.xilinx.com/xlnx/xil product.jsp? Title=micro blaze.

Published By:
Blue Eyes Intelligence Engineering and Sciences Publication (BEIESP)
© Copyright: All rights reserved.