Asia-pacific Journal of Multimedia Services Convergent with Art, Humanities, and Sociology
Vol.2, No.1, June (2012), pp. 17-24
http://dx.doi.org/10.14257/AJMAHS.2012.06.02

Hardware Implementation of Pulse Code Modulation Speech Compression Algorithm

Geetam Singh Tomar[1)]

Abstract

Voice-over Internet Protocol (VoIP) telephony is increasingly becoming a variant telecommunication technology that one day may surpass the old analog and digital telephone systems. The Quality-of-Service (QoS) factor is an important parameter to be considered when measuring the performance of a VoIP system. Many factors may influence the QoS of a given VoIP system. Some of these factors are delay, jitter and packet loss. This paper served to demonstrate that the algorithmic delay (latency) of the Pulse Code Modulation (PCM) speech compression algorithm which has originally been implemented in software can be significantly reduced by implementation in hardware. The PCM speech compression algorithm is first implemented, verified to demonstrate equivalence and then validated by comparing the latency of the hardware-implemented speech compression algorithm with that of an existing software implementation.

Keywords : Pulse Code Modulation; Hardware implementation; Speech compression algorithm.

1. Introduction

A speech signal is a sound formed by humans in order to communicate. Like all other sound, it is a perturbation of the atmospheric pressure that travels in the form of a sound wave [1-4]. To transport speech over communication networks, the speech signal must be transformed from a sound wave to an electrical signal. This is usually done with a microphone and the result is an electrical signal where the voltage varies with the pressure of the speech signal. VoIP is the transmission of real-time voice traffic over an Internet Protocol (IP) data network [2]. VoIP is an alternative to the Public Switched Telephone Network (PSTN) [5][6] when placing a telephone call between two or more parties. Since both sides of a telephone conversation can speak and listen, each side must be able to code and decode voice data. Coding and decoding of speech is done using speech compression algorithms. Waveform-based speech compression algorithms are the simplest form of speech compression algorithms. They represent individual samples from the analog speech waveform in a digital form.

Speech compression algorithms can be implemented in three ways: 1) software; 2) hardware-software

Received(March 15, 2012), Review request(March 16, 2012), Review Result(1st: April 05, 2012)
Accepted(June 10, 2012)

[1]Dept of Electrical and Computer Engg, University of West Indies, St. Augustine, Trinidad and Tobago
email: marcus.george99@yahoo.com

ISSN: 2383-5281 AJMAHS

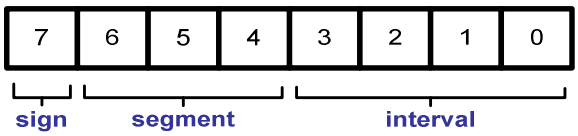
co-design; and 3) hardware. The software implementations of speech compression algorithms such as in [5] have provided users with high speech quality and low terminal price. However the software implementations of these speech compression algorithms require a significant amount of time for their execution. The implementations of speech compression algorithms that take advantage of the hardware-software co-design model such as in [5] have also provided users with high speech quality and low terminal price. Although these implementations of speech compression algorithms required less time for their execution than the speech compression algorithms implemented entirely in software, they still require too much time for their execution.

It can be assumed from the information previously presented that the implementation of the speech compression algorithms which utilize the hardware-software co-design model may have been speeded up because of the fact that some components have been implemented in hardware. This also indicates to us that that algorithmic delay of speech compression algorithms that were implemented entirely in software can be accelerated by implementation in hardware. To further solve the problem of large delays required for the execution of speech compression algorithms, this paper will present the implementation of the PCM speech compression algorithm entirely in hardware and demonstrate that the hardware implementation significantly reduces the algorithmic delay of the PCM speech compression algorithm originally implemented in software.

2. Overview of PCM Speech Compression Algorithm

PCM is a waveform based speech compression algorithm. The international standard for PCM is ITU-T G.711 [3]. The PCM speech compression algorithm uses logarithmic scalar quantization and represents each speech sample with 8 bits. Logarithmic scalar quantization places most of the quantization steps at lower amplitudes by using the non-linear function logarithm function. The PCM algorithm converts speech in 16-bit uniform PCM format to 8-bit μ-law or A-law PCM format.

The μ-law/A-law PCM byte (Figure 1) comprises of three components: sign bit, segment and interval. The sign of the input PCM sample is represented by bit 7 (MSB) of the uniform PCM byte, the segment to which the input PCM sample belongs to is represented by bits 4 to 6, and the position within the segment that the input PCM sample resides, known as the interval value is represented by the lower nibble. PCM is widely used due to its simplicity. This results in low algorithmic delay and high decoded speech quality but also a relatively high bit rate.



[Fig. 1] Structure of a μ-law/A-law PCM byte

3. Profiling of Software Implementations Of PCM

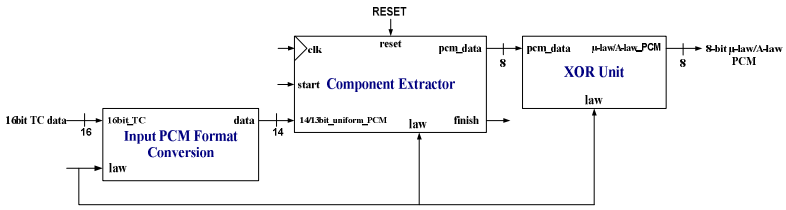
The latency of the software implementation of PCM from Sun Microsystems was obtained using the procedure outlined in [7-9]. Because of the GPROF time resolution each encoder/decoder was run over 1,000,000 inputs. To discount the effects of other programs/overhead the process was repeated three times for each encoder/decoder considered. The total was divided by 3,000,000 to give an estimate of the latency. Table I gives the results of the software profiling for the latency (in cycles) of the software implementation of PCM encoders and decoders. [9] demonstrates how latency in nanoseconds was converted to cycles.

[Table 1] Latency of Software Implemented PCM Speech Compression Algorithm from Sun Microsystems

Sun Microsystems		
Name of function	end-to-end execution	
	time / ns	cycles
PCM Encoder	33	79200
PCM Decoder	36	86400

4. Design of The PCM Speech Compression Algorithm

The PCM encoder (Fig 2) converts data in 14/13-bit uniform PCM format to 8-bit μ-law/A-law PCM format, and consists of three units. The input PCM format conversion unit converts speech in 16-bit two's complement format to 14-bit two's complement format (14-bit uniform PCM). The component extractor unit along with the XOR operation convert data in 14-bit two's complement format to 8-bit μ-law/A-law PCM format. Among these three units, only the operation of the component extractor must be initiated and controlled. However a controller unit was not included because the latency of the input PCM format conversion and XOR units were very short and their operation does not need to be initiated or controlled.



[Fig. 2] Datapath block diagram of the PCM Encoder

The component extractor performs logarithmic scalar quantization on the 14-bit uniform PCM sample.

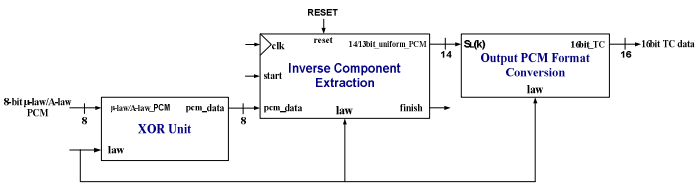
Initially the sign bit (MSB) is extracted and installed as the sign bit of the μ-law/A-law PCM byte. The remaining 13/12-bits are input to the Segment LUT which represents in 3 bits, the segment to which the 14-bit uniform PCM sample belongs. The 3-bit segment is installed as part of the μ-law/A-law PCM byte. The section of component extractor responsible for the computation of the interval was designed by applying the hardware design technique outlined in [9] to equation (1).

$$\text{int}erval = \frac{\mu\_law/A\_law - lower\_boundary}{upper\_boundary - lower\_boundary} \times 1111_2 \quad (1)$$

An integer multiplier and divider were used instead of floating point units were used in the computation of the interval because the quantized version of the interval has to be stored in the lower nibble of the μ-law/A-law PCM byte. The integer multiplier was placed before the divider to ensure that output values less than 1 are not obtained, hence eliminating the need for floating-point units to ensure correct computation of the interval. Another reason for choosing integer units is that they require less hardware than floating point units.

The PCM encoder could have been designed using a LUT instead of the approach of Figure 2. This would have yielded a faster system. However a large amount of hardware would have been required to represent all 16-bit uniform PCM values. Since speech is being sampled at 8kbit/s, the time between successive encode processes is relatively long compared to the latency of the PCM encoder (Figure 2). Therefore a faster system (LUT approach) is not advantageous in this case.

The PCM decoder (Figure 3) converts speech in 8-bit μ-law/A-law PCM format to 16-bit uniform PCM format. This system consists of three units. The inverse-component extractor unit along with the XOR operation convert the μ-law/A-law PCM byte to 14-bit two's complement format. The output PCM format conversion unit converts speech in 14-bit two's complement format to 16-bit two's complement format. A controller unit was not included for the same reason one was not included in the PCM encoder.



[Fig. 3] Datapath block diagram of the PCM Decoder

The inverse-component extractor first extracts the sign bit (MSB) installed it as the sign bit of the 14-bit two's complement sample. The segment and interval were used in computation of the least significant 13-bit of the two's complement output. This section was designed by applying the hardware design technique outlined in chapter [9] to equation (2).

$$\mu\_law / A\_law = A + [\frac{\text{int}\, erval\_value}{(10000)_2} \times B]$$
$$A = lower\_boundary$$
$$B = upper\_boundary - lower\_boundary \quad (2)$$

Integer units were used over floating point units for the same reason they were chosen in the design of the component extractor. Like the PCM encoder, the PCM decoder could have been designed using a LUT. However this approach was not used for the same reasons why it was not selected for the design of the PCM encoder.

5. Hardware Implementation of The PCM Speech Compression Algorithm

The hardware implementation of the PCM speech compression algorithm was done in VHDL using the Xilinx 11.1 ISE environment. VHDL was selected for the advantages presented in [9]. Integer arithmetic units obtained from [1] and the Xilinx IPcore library of the ISE environment were used in the hardware implementation of the PCM speech compression algorithm. Integer arithmetic was used in the implementation of the PCM encoder and decoder instead of floating point and fixed point units because the outputs of the PCM encoder and decoder were quantized values and hence units that can handle fractional values are not necessary since the outputs would be rounded-off to obtain the quantized values.

The hardware units were port-mapped together using knowledge gained in [7] to implement the PCM encoders and decoders. FSM control units designed as demonstrated in [9] were used in the control of the component extraction and inverse component extraction units. After implementing the PCM speech compression algorithms, they were synthesized in the Xilinx 11.1 ISE environment in preparation for verification and validation.

6. Verification and Validation of The PCM Speech Compression Algorithm

Test cases for the verification tests consisted of input samples belonging to each of the following categories: small, intermediate and large values, positive and negative values. The expected outputs for the PCM encoder were obtained from [3]. The actual outputs of the hardware-implemented PCM encoder were obtained through simulation using Modelsim XE 6.4b using the chosen test cases. The actual outputs were compared with the expected outputs to verify that the hardware-implemented PCM encoder correctly compressed the input test sequences.

The expected outputs of the hardware-implemented PCM encoder were used as the inputs to the

hardware-implemented PCM decoder. The expected outputs for the PCM decoder were obtained from [3]. The actual outputs of the hardware-implemented PCM decoder were obtained through simulation using Modelsim XE 6.4b. The actual outputs of the hardware-implemented PCM decoder were compared to the input samples to the hardware-implemented PCM encoder to verify that the hardware-implemented PCM decoder correctly decompressed selected inputs.

The verification test results (Table II and Table III) indicate that the actual outputs of the PCM encoder and decoder in μ-law and A-law modes corresponded to the expected outputs from [3]. This demonstrated that hardware-implemented PCM encoder and decoder correctly compressed and decompressed the input samples respectively.

[Table 2] Verification Tests Results for PCM Encoder and Decoder in U-Law Mode

u-law PCM Encoder			**u-law PCM Decoder**		
Input Sample	**Expected Output**	**Actual Output**	**Input Sample**	**Expected Output**	**Actual Output**
14	11111111	11111111	11111111	14	14
198	11010111	11010111	11010111	198	198
-198	01010111	01010111	01010111	-198	-198
710	10111111	10111111	10111111	710	710
2246	10011111	10011111	10011111	2246	2246
-2246	00011111	00011111	00011111	-2246	-2246
4294	10001110	10001110	10001110	4294	4294
7923	10000000	10000000	10000000	7923	7923

[Table 3] Verification Tests Results For Pcm Encoder And Decoder In A-Law Mode

A-law PCM Encoder			**A-law PCM Decoder**		
Input Sample	**Expected Output**	**Actual Output**	**Input Sample**	**Expected Output**	**Actual Output**
14	11010101	11010101	11010101	14	14
198	11111101	11111101	11111101	198	198
-198	01111101	01111101	01111101	-198	-198
710	10010101	10010101	10010101	710	710
2246	10110111	10110111	10110111	2246	2246
-2246	00110111	00110111	00110111	-2246	-2246
3270	10111101	10111101	10111101	3270	3270

The latency of both hardware-implemented PCM encoder and decoder were obtained by applying the technique outlined in [9]. The results of the validation tests (Table IV) indicate that the latency of hardware PCM encoder was 1760 times shorter than the latency of software PCM encoder, while the latency of the hardware PCM decoder was 1878 times shorter than the latency of the software PCM decoder.

[Table 4] Performance Comparison of Hardware and Software Implementations of PCM

PCM Functional Module	Average End-to End Execution Time / Cycles	
	Hardware Implementation	Software Implementation
Encoder	45	79,200
Decoder	46	86,400

7. Discussion

From the verification results it was deduced that the actual outputs of the PCM encoder and decoder in both μ-law and A-law modes correspond to the expected outputs from [3]. This demonstrated that hardware-implemented PCM encoder and decoder correctly compressed and decompressed their input samples respectively and hence can be considered equivalent implementations of the PCM encoder and decoder respectively.

The validation results indicated that the latency (algorithmic delay) of both the hardware-implemented PCM encoder and decoder in both μ-law and A-law modes were 1760 and 1878 times shorter than the latency of their software counter-parts from Sun Microsystem respectively. This demonstrates that the algorithmic delay of the PCM speech compression algorithm which has originally been implemented in software was significantly reduced by implementation in hardware.

The main advantage of using hardware is speed as these algorithms use a great deal of processing power for computation. Software has the disadvantage of being slower but however has the benefit of flexibility. However with the advent of reconfigurable computation, FPGAs are capable of providing this advantage to embedded systems designers. The speech compression algorithms could also have been implemented in hardware using analog circuits instead of digital circuits. However the area used for the implementation of the systems would have been larger and because of the great complexity of the several of the speech compression algorithms the implementation process would have taken more time than if the system was implemented with digital circuits using an HDL. In addition the implementation of the speech compression algorithm in hardware using an HDL was accelerated by the availability of floating-point libraries. Required floating-point units were selected from these libraries for use in the implementation of these speech compression algorithms.

8. Conclusions

This paper presented the implementation, verification and validation of the PCM speech compression algorithm [3]. A future development for this system is the creation of a prototype to work with the existing VoIP software which can replace the existing software-implementation of PCM. This will require the

development of an ASIC for each of these speech compression algorithms. ASICs have several advantages over FPGAs. ASICs are relatively faster and more power efficient than FPGAs [8]. FPGA power and delay overheads are due to their programmable switch elements [8]. ASICs however have a reduced production volume compared to FPGAs [8].

References

[1] D. Bishop, “Parametized Floating Point Package”, **(2008)** February 27, http://www.vhdl.org/fphdl/vhdl.html (accessed July 17 2009).

[2] F. Halsall, “Data Communications, Computer Networks and Open Systems", 4th. ed., Wales: Addison-Wesley Publishing Company, **(1998)**.

[3] International Telecommunication Union (ITU), **(1972)**, Pulse Code Modulation (PCM) of Voice Frequencies. G.711.

[4] G. R. Juan, “Introduction to the Physics and Psycholophysics of Music”, 3rd ed. New York: Springer Verlag, **(1995)**.

[5] S. Mishra and A. Balaram, “Efficient hardware-software co-design for the G.723.1 algorithm targeted at VoIP applications”, **(2000)**.

[6] C. Montminy, “A Study of Speech Compression Algorithms for Voice over IP. Msc Thesis”, Ottawa-Carleton Institute for Electrical and Computer Engineering, University of Ottawa, Canada, **(2000)**.

[7] D. Perry, "VHDL", 3rd ed. New York: McGraw-Hill, **(1998)**.

[8] R. Tessier, "Fast Place and Route Approaches for FPGAs", PhD Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, United States of America, **(1999)**.

[9] V. A. Pedroni, “Digital Electronics and Design with VHDL”, 1st ed. Boston: Elsevier Science, **(2008)**.

