Stochastic Circuit Design and Performance Evaluation of Vector Quantization

Ran Wang, Jie Han, Bruce Cockburn, and Duncan Elliott
Department of Electrical and Computer Engineering
University of Alberta
Edmonton, AB T6G 2V4, Canada
{ran5, jhan8, cockburn, duncan.elliott}@ualberta.ca

***Abstract*—Vector quantization (VQ) is a general data compression technique that has a scalable implementation complexity and potentially a high compression ratio. In this paper, a novel implementation of VQ using stochastic circuits is proposed and its performance is evaluated. The stochastic and binary designs are compared for the same compression quality and the circuits are synthesized for an industrial 28-nm cell library. The effects of varying the sequence length of the stochastic design are studied with respect to the performance metric of throughput per area (TPA). When a shortened 512-bit encoding sequence is used to obtain a lower quality compression, the TPA is about 2.60 times that of the binary implementation with the same quality as that of the stochastic implementation measured by the L^1 norm error (i.e., the first-order error). Thus, the stochastic implementation outperforms the conventional binary design in terms of TPA for a relatively low compression quality. By exploiting the progressive precision feature of a stochastic circuit, a readily scalable processing quality can be attained by simply halting the computation after different numbers of clock cycles.**

Keywords—stochastic computing; vector quantization; sequence length; cost efficiency

I. INTRODUCTION

In some application areas, stochastic computation has been shown to have advantages with respect to important measures of circuit performance [1, 2]. These advantages include potentially simpler arithmetic hardware and inherent tolerance of transient signal errors. Because accuracy is closely related to the stochastic sequence length, applications for which some accuracy can be safely sacrificed, such as multimedia information displayed to humans, should be considered for implementation in stochastic circuits. Some image processing algorithms using stochastic methods have already been shown to match their binary counterparts in terms of hardware cost and speed while providing a similar experience to humans [3].

The use of combinational logic as stochastic elements can be traced back to the work of Gaines [1]. Several common arithmetic blocks, such as adders, multipliers, dividers and integrators, were proposed for both bipolar and unipolar stochastic representations. An arithmetic synthesis method based on Bernstein polynomials was investigated in [4]. Bernstein polynomials were found to be efficiently built with stochastic logic. An arbitrary polynomial can be expressed using Bernstein polynomials after proper scaling operations, thus any polynomial can be implemented stochastically. In [5], sequential stochastic computational elements were built using finite state machines (FSMs). Various arithmetic functions were built using state transition diagrams such as stochastic exponentiation, a stochastic *tanh* function, and a stochastic linear gain function [6]. These stochastic implementations were shown to be efficient compared with previous stochastic computing elements implemented with combinational logic.

Vector quantization based compression algorithms are useful in that the amount of stored and transmitted data can be reduced with a readily adjusted trade-off between compression ratio and implementation size. These features are important in multimedia processing and communications, such as for voice and image compression. VQ is a lossy data compression method, and the loss in the original information must be kept as low as possible. In VQ, the information loss can be reduced by simply using a larger suitably-designed codebook. The resulting extra search time can be minimized by using parallel computational elements [7].

In this paper, we explore the feasibility of using stochastic circuits to implement VQ. Typically, a longer sequence offers greater representational accuracy, but with a longer computation time. In a stochastic design, therefore, shorter sequences are used to provide faster computation but with less accurate results. Both the stochastic and binary designs are synthesized to measure important performance characteristics.

II. VECTOR QUANTIZATION

A. Background

Vector quantization (VQ) is a lossy digital compression technique. First the source data is partitioned into equal-length vectors [7]. Each vector is then replaced with the index of the closest matching codevector that is contained in a given codebook. This encoding process is shown in Fig. 1(a). Note that each input vector is more compactly represented using the index of the closest codevector. The indexes are then replaced with the corresponding codevectors during decompression, as shown in the decoding process in Fig. 1(b).

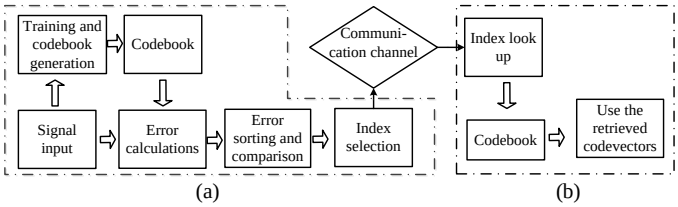


Fig. 1. The block diagram for the (a) encoding and (b) decoding in VQ.

B. Codebook Generation and the Encoding Process

In 1980's, Linde, Buzo, and Gray (LBG) proposed a VQ codebook generation algorithm [7]. Although other efficient codebook generation approaches have been developed, we selected the LBG algorithm to generate our codebook because of its efficiency. The LBG algorithm is similar to the K-means clustering algorithm with a large amount of feedback before convergence. Therefore, codebook design and generation can be computationally intensive. In this paper, however, we focus on implementing the VQ encoders using prepared codebooks.

The VQ encoding process is as follows:

1) A set of N_x source vectors $\{\boldsymbol{X_1}, \boldsymbol{X_2}, \dots, \boldsymbol{X_{N_x}}\}$ is to be compressed.

2) A codebook with N_c codevectors was generated previously:

$$\boldsymbol{C} = \{\boldsymbol{C_1}, \boldsymbol{C_2}, \dots, \boldsymbol{C_{N_c}}\}. \quad (1)$$

3) The codevector $\boldsymbol{C_i}$ $(i = 1, 2, \dots, N_c)$ that is the nearest to each of the source vector with respect to the corresponding errors E_i $(i = 1, 2, \dots, N_c)$ must be found. If function f maps the source vector $\boldsymbol{X}$ to its nearest codevector $\boldsymbol{C_i}$, we have

$$f(\boldsymbol{X}) = \boldsymbol{C_i} \text{ if } E_i \le E_{i'}, \forall i' = 1, 2, \dots, N_c. \quad (2)$$

The L^1-norm error E_i is defined as

$$L^1: E_i = |\boldsymbol{X} - \boldsymbol{C_i}| = \sum_{j=0}^{N_e-1} |X_j - C_{ij}|, i = 1, 2, \dots, N_c, \quad (3)$$

where N_c is the number of codevectors in the codebook, and N_e is the number of elements in a vector (any codevector or input vector $\boldsymbol{X}$). Then we compare and find the minimum error distance E_{min} using (3). If $E_i = E_{min}$, index i is used as the compressed encoding of the input vector $\boldsymbol{X}$.

4) Compression is obtained by mapping the N_x source vectors to the N_x corresponding indexes of the closest codevectors.

5) A decompressed approximation to the N_x source vectors is obtained from the compressed representation by replacing the indexes with the corresponding codevectors.

III. Required Stochastic Computing Elements

The arithmetic operations required by a VQ design follow the error calculations specified in (3) as well as the block diagram in Fig. 1. In this section the required stochastic computing elements, including adders, absolute subtractors and comparators, are described.

A. Combinational Stochastic Elements

In stochastic computing, a multiplexer is used as a scaled adder (see Fig. 2(a)). Interestingly, an XOR gate can be used to compute the absolute value of a subtraction [3], provided that there is an appropriate correlation between the two parallel input sequences (see Fig. 2(b)). If S1 and S2 are two statistically independent sequences containing N_s bits, then S1 and S2 are related as follows:

$$\sum_{i=0}^{N_s-1} S1(i) \cdot S2(i) = \frac{1}{N_s} \sum_{i=0}^{N_s-1} S1(i) \cdot \sum_{i=0}^{N_s-1} S2(i). \quad (4)$$

When an XOR gate is used to implement the absolute value of a subtraction, correlated sequences are generated by sharing the same linear feedback shift register (LFSR) and the same initial seed.

B. Sequential Stochastic Elements

A stochastic implementation of the *tanh* function is proposed in [6] using the state transition diagram in Fig. 3. This is essentially a saturating up-down counter with the sign bit as the output. The state machine starts at the central state and is always reset to that state before every new *tanh* calculation.

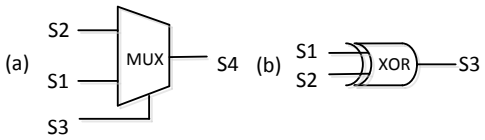


Fig. 2. Stochastic arithmetic units: (a) $S4 = \frac{1}{2} \cdot (S1 + S2)$ (S3 is a unipolar sequence encoding a probability of 0.5) and (b) $S3 = |S1 - S2|$ (S1 and S2 are correlated unipolar sequences).

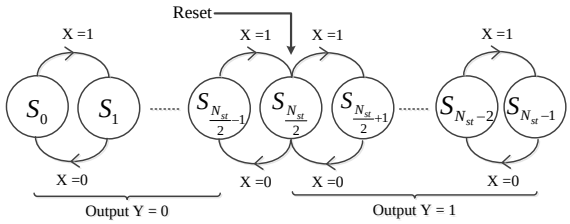


Fig. 3. A state transition diagram of stochastic *tanh* function [6].

The distance calculation and sorting operations are crucial in vector quantization. Ideally stochastic comparison operations in VQ require the Heaviside step function, which can be approximated using the stochastic *tanh* function and then realized using a finite state machine (FSM). When the number, N_{st}, of states is large, the *tanh* function behaves like a Heaviside function. The stochastic comparator can be implemented using the *tanh* function [6]. In Fig. 4, the architecture of a stochastic comparator is shown with two stochastic inputs P_X and P_Y. The output P_S of the stochastic comparator is a stochastic approximation to $min(P_X, P_Y)$. If it is not the last stage in the comparison tree, P_S will be passed on as a data input to the next comparator stage.

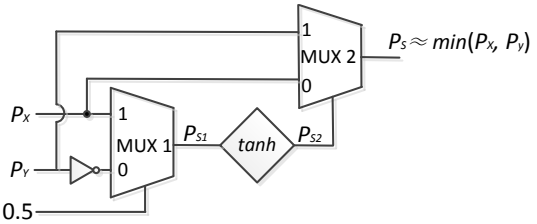


Fig. 4. Stochastic comparator based on the stochastic *tanh* function [6].

IV. Proposed Stochastic Circuit Design

A. Overall System Architecture

The stochastic VQ system can be abstracted as in Fig. 5. As an example, consider an image of 300 pixels by 300 pixels for performance evaluation and comparison. Each of the four-by-four square blocks is considered as a vector while the pixel values are the elements in the vector. There are thus $300 \times 300/16 = 5625$ four-by-four pixel blocks in this image. Fig. 6 illustrates how the 16-element input vectors are formed. Assume we have 256 codevectors in the codebook. Both the binary and stochastic implementations of VQ use errors calculated based on the L^1 norm. In addition to the gates for error calculation and comparison, the total hardware cost must also include memory cost because the iteratively updated errors are stored in indexed arrays.

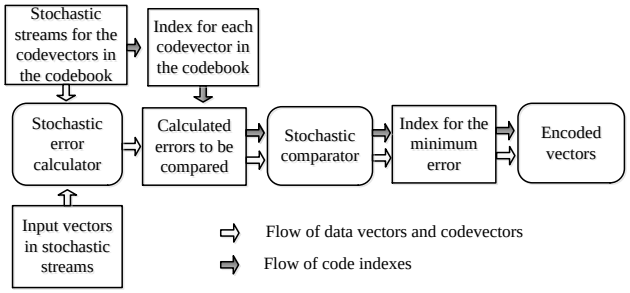


Fig. 5. Data flow in the encoding process of vector quantization.

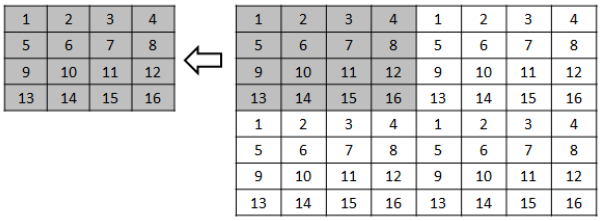


Fig. 6. An input vector has 16 entries, from 1 to 16 in the left block, forming a macro-pixel. In the right block, the array of 8×8 pixels can be divided into 4 macro-pixels.

B. Detailed Design for Stochastic VQ Using L^1-norm Errors

In the L^1-norm error calculation, we need to implement (3). A suitable architecture is shown in Fig. 7 with $N_e = 16$ elements in a vector. $X[i]$ and $H[i]$ represent the i^{th} elements in the input vector and one of the $N_c = 256$ codevectors. They are encoded by using stochastic number generators [2] from their original 8-bit binary values for a grey-scale image. S(Y) is the stochastic output, which must be stored prior to being converted back to a binary number at the final stage using a counter. In this stochastic VQ design, however, this conversion is not needed as the final result of the encoding process is a binary index that is embedded in the stochastic sequences. The XOR gates are used to implement the absolute subtractions in stochastic computing with correlated stochastic sequences, where the correlated sequences attain the maximum overlap of 1's [3]. The results are then added up by the 16-input multiplexer whose selecting signals Sel[0] to Sel[3] are four independent stochastic sequences encoding 0.5. Note that 256 copies of the L^1-norm error calculator in Fig. 7 are used to compute the 256 errors in parallel. The outputs of the L^1-norm error calculator are passed on to a parallel comparison tree (see Fig. 8) to find the minimum value.

C. Index storage and delivery

A source vector is encoded by the index of the codevector that produces the minimum error among all the calculated ones. The comparison results come naturally as stochastic streams that represent probabilities instead of deterministic Boolean values. Therefore the stochastic streams have to be converted to binary numbers by counters, which would add cost. To avoid this problem, we can embed the index in the last few bits of the stochastic sequences as a binary-encoded value. The error of the k^{th} codevector is labeled with index k. The last few bits in this stochastic error sequence are replaced with the binary number k. The preceding stochastic bits are left unchanged. If the sequences are long enough, giving up the last few bits will have little effect on the stochastic value. By doing so, only one shift register is required at the last stage

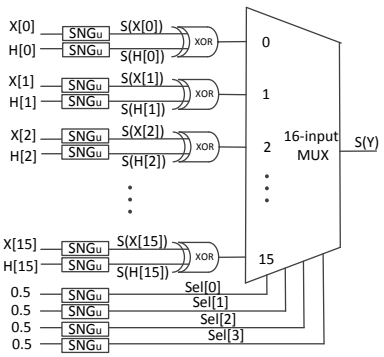
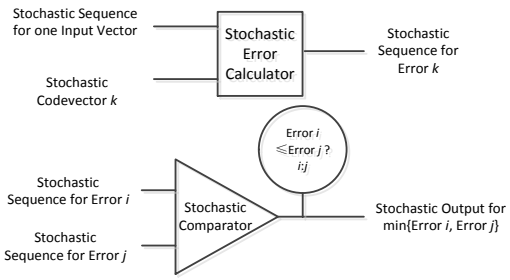
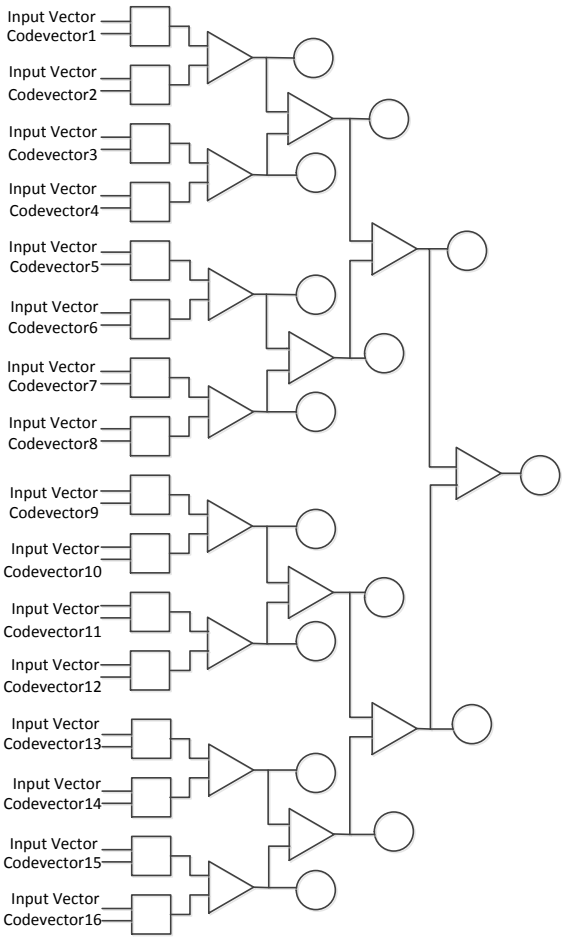


Fig. 7. Architecture of the L^1-norm error calculator.



(a)



(b)

Fig. 8. (a) Stochastic elements. (b) Stochastic comparison tree: a square represents an error calculator, a circle represents a comparison result and a triangle represents a stochastic comparator. Note that the codevector with the smallest error is chosen and forwarded as the comparison result.

to extract the index from the stochastic sequence. Hence, registers for index storage and counters used as the stochastic-to-binary converter for every comparator are saved to reduce hardware cost. A shorter delay also results as no extra time is needed to process the index, which is extracted easily from the output bit stream.

D. Synthesis of Stochastic Logic to Implement VQ

The stochastic VQ design can be synthesized easily by mapping the necessary operations and the corresponding stochastic elements. If we are given an input vector and a codebook, then a general synthesis method can be used to implement the stochastic VQ encoder as follows:

1) The number of XOR gates and the size of the multiplexer in the error calculator shown in Fig. 7 are determined by the vector length (i. e. the number of elements in a codevector or source vector).

2) The number of error calculators and the size of the stochastic comparison tree shown in Fig. 8 are determined by the size of the codebook. A codebook with N_c codevectors requires N_c error calculators and a comparison tree with $N_c/2$ stochastic comparators as leaf nodes.

3) The data precision can be adjusted by using different sizes of LFSRs when stochastic number generators are implemented.

V. Simulation and Discussion

A. Required Sequence Length

The loss in image quality caused by compression can be measured objectively as the total power in the error between the original image and the image reconstructed from the output of a VQ encoder. Assume that the image contains N_p=90,000 pixels. Let the pixel values in the original image be denoted by P_{Oi} $(i = 0, 1, \dots, N_p - 1)$, where the pixel values of the reconstructed image after compression are denoted by P_{Ci} $(i = 0, 1, \dots, N_p - 1)$. The average penalized error (APE) of the loss of quality is defined as

$$APE = \sqrt{\frac{1}{N_p}\sum_{i=0}^{N_p-1}\left(P_{Oi} - P_{Ci}\right)^2}. \quad (5)$$

The stochastic VQ circuit was implemented for the L^1-norm error calculations and various sequence lengths were investigated. The corresponding APEs are reported in Table I.

TABLE I. The APE values using L^1-norm error at different sequence lengths for stochastic VQ and different bit resolutions for conventional binary VQ.

Sequence Length (bits)	256	512	1024	2048	4096	9192
APE	27.4	19.9	10.9	7.3	7.1	7.1
Resolution (Bits)	4	6	7	8	9	10
APE	33.1	18.3	11.0	7.8	7.6	6.9

The APE decreases as the stochastic sequence length grows, as expected. Conventional binary implementations of VQ with the same experimental parameters are also simulated for comparison. It can be seen that 8-bit resolution produces good quality and that higher bit resolutions only improve the APE more slowly. The stochastic implementation using 2048 bits subjectively matches the 8-bit binary conventional implementation as the two designs produce similar APEs. Implementations with roughly equivalent compression performances are considered next in the comparison of circuit performance.

The sequence length implies an output latency that limits the performance of a stochastic circuit. Vector quantization, however, is already a lossy data compression method. We can in some cases accept quality deterioration to reduce the latency. In fact, the quality of the compression relies heavily on the comparison results of the errors. Hence, the accurate ranking of the errors is more important than the error accuracies. Finally, in streaming media applications, latency is often not an issue as it only affects the initial delay.

B. Functional Simulation Using Matlab

The classic Lena image was used as the input source and the LBG algorithm was used to generate a codebook. Fig. 9 shows the stochastic L^1-norm VQ simulation results. The input is a 300×300 pixel grey-scale image. Each pixel is represented by an 8-bit binary number. After using stochastic vector quantization to compress the original image, the image is re-constructed using codebook look-up and displayed for visual quality assessment. The image has 5625 input vectors, and each vector comprises 16 unsigned 8-bit pixel values. We use 2048 bits in a stochastic sequence, so it takes 2048 clock cycles to finish one round of calculation. The stochastic representation of 2048 bits can be generated by an 11-bit LFSR implemented in a Matlab function. To encode the 5625 input vectors in a fully-parallel architecture, a total of 5625 independent processor units are required and each unit includes 256 error calculators and a 256-input comparison tree.

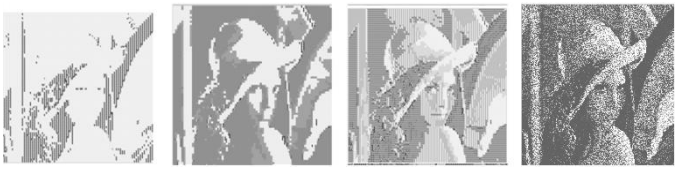


Fig. 9. The progressive improvement of image quality using L^1-norm stochastic VQ after 256, 512, 1024 and 2048 clock cycles.

The output images in Fig. 9 illustrate the progressive quality feature of stochastic computing. The reconstructed image after stochastic compression for the 256th, 512th, 1024th and 2048th clock cycles are shown for the L^1-norm error measure. The reconstructed output images are vague and only show a rough outline of the original image after compression using 256 clock cycles. However the reconstructed image becomes a clearer reproduction as the stochastic encoding time increases. Because 11-bit LFSRs are used to generate the stochastic sequences, the sequences repeat every 2048 cycles. The image quality stops improving after 2048 clock cycles.

C. Circuit Performance

The hardware area, power consumption and delay are investigated for an L^1-norm implementation using both the stochastic and conventional binary approaches. The auxiliary circuits such as stochastic number generators (implemented by LFSRs) and counters are all included. By using the Synopsys design compiler [8], we obtained the fastest clock and the corresponding power and silicon area.

Following the results in Table I, we compared (1) an 8-bit binary implementation with the stochastic implementation using 2048-bit sequences, (2) a 7-bit binary implementation with the stochastic implementation using 1024-bit sequences

and (3) a lower quality processing implementation using the 6-bit binary and the 512-bit stochastic designs. The synthesis reports are shown in Table II. The stochastic circuits have significantly lower hardware cost. Stochastic implementations only cost roughly 1% of the hardware of binary implementations. This also leads to savings in power consumption. Moreover, we can compress two input images using the 2048-bit stochastic VQ implementation and achieve the same compression quality as the 1024-bit stochastic VQ implementation. Thus only the 2048-bit stochastic VQ implementation is needed instead of two copies of the 1024-bit stochastic VQ circuits, further saving 32.3% of the area.

TABLE II. CIRCUIT PERFORMANCE OF L^1-NORM VECTOR QUANTIZATION: (1) 8-BIT BINARY (B) VS. 2048-BIT STOCHASTIC (S), (2) 7-BIT BINARY (B) VS. 1024-BIT STOCHASTIC (S) AND (3) 6-BIT BINARY (B) VS. 512-BIT STOCHASTIC (S).

	Area (μm^2)			Power (mW) @ Min Clock Period			Minimum Clock Period (ns)		
	B	S	Ratio: S/B	B	S	Ratio: S/B	B	S	Ratio: S/B
(1)	93294	1358	0.015	107.56	3.22	0.03	2.26	0.20	0.09
(2)	86231	1003	0.012	81.30	2.86	0.04	2.26	0.20	0.09
(3)	79177	641	0.008	50.09	2.47	0.05	2.26	0.21	0.09
	Energy per Operation (pJ/Operation)			Throughput per Area ($1/(\mu m^2 \cdot s)$)			Sequence Length (bits)		
	B	S	Ratio: S/B	B	S	Ratio: S/B	B	S	Ratio: S/B
(1)	243	1320	5.43	4743	1797	0.38	N/A	2048	N/A
(2)	184	586	3.19	5131	4869	0.95	N/A	1024	N/A
(3)	113	266	2.35	5588	14519	2.60	N/A	512	N/A

The time required for an encoding operation is determined by the product of the clock period and the stochastic sequence length. Because the structure of stochastic circuits is simpler, a shorter critical path delay is expected. In Table II the minimum stochastic clock periods are roughly 10% of the minimum binary clock periods.

The throughput per area (TPA) and the energy per operation (EPO) are two important generic metrics. In Table II, the stochastic approach shows significant advantages over the binary approach in terms of the area cost, power consumption and delay, as expected. When long sequences such as 2048 bits are considered, the ratio of the EPOs between stochastic and binary circuits is about 5.43, and the ratio of the TPAs is approximately 0.38. Therefore, the stochastic circuit using 2048-bit sequences consumes more energy and underperforms the conventional binary circuit using an 8-bit resolution. However, if some loss in quality is acceptable in the application, the stochastic implementation using 512-bit sequences shows only 2.35 times of the EPO and 2.60 times of the TPA with only 0.8% the total area compared to a 6-bit binary implementation. It can be seen that the stochastic implementation using 1024-bit sequences shows similar performance compared to the 7-bit binary implementation with respect to TPA. The stochastic VQ is thus not competitive for 7-bit or higher bit resolutions in terms of the TPA.

VI. CONCLUSIONS

Stochastic circuits were proposed to implement L^1-norm vector quantization (VQ). Implementations with similar compression qualities were compared with respect to average penalized error (APE) for a grey-scale image. Due to the compact stochastic arithmetic elements and an efficient index storage approach, the area for a stochastic implementation is less than 1.5% of the fully parallel binary design. Our results show that the 2048-bit stochastic VQ circuit underperforms the 8-bit binary circuit in terms of throughput per area (TPA) and energy per operation (EPO). However, the stochastic circuit can be more efficient for a lower-quality compression algorithm by using the 512-bit stochastic circuit. In the binary implementation, 256 codevectors are used to encode an image with 90000 pixels, so the 6-bit VQ provides an image quality of $256 \times 16 \times 6$ bits / 90000 pixels = 0.273 bit per pixel. Because the 512-bit stochastic implementation shows similar compression quality to the 6-bit binary implementation, it can be potentially used where videos are streamed with low-speed connections and possibly low-quality images [9]. The TPA of the 512-bit stochastic circuit is roughly 2.60 times as large as that of the 6-bit binary circuit. We found that the 7-bit binary implementation and 1024-bit stochastic implementation show roughly the same performance in terms of TPA. Thus, stochastic VQ is only competitive for bit resolutions lower than 7 bits.

Applications could benefit from a stochastic implementation of VQ if the extra energy per operation and the higher latency are not important issues. The stochastic design is so small that many multimedia streams can be encoded simultaneously in the same area as one conventional binary design. The inherent progressive quality of the stochastic VQ design is potentially useful. To obtain greater accuracy, one simply waits more clock cycles. The number of clock cycles can be reduced to save energy at the cost of less accuracy. Hence, a stochastic VQ implementation for high-quality compression can be easily applied to obtain lower compression quality by encoding multiple input images. Compared to lower-quality stochastic circuit implementations, the area can further be significantly reduced.

REFERENCES

[1] B. R. Gaines, "Stochastic computing systems." In *Advances in information systems science*, pp. 37-172. Springer US, 1969.

[2] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. on Embedded Computing Systems*, vol. 12, no. 2s, p. 92, 2013.

[3] A. Alaghi and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," *Proc. DAC*, pp. 136:1-6, June 2013.

[4] W. Qian and M. D. Riedel. "The synthesis of robust polynomial arithmetic with stochastic logic." In *Design Automation Conf.*, pp. 648-653. IEEE, 2008.

[5] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms." *Proc. 29th Int. Conf. on Computer Design*, IEEE, 2011.

[6] P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Computation on stochastic bit streams digital image processing case studies," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 3, pp. 449-462, 2014.

[7] Y. Linde, A. Buzo and R. M. Gray, "An algorithm for vector quantizer design." *IEEE Trans. on Communications*, 28.1 (1980): 84-95.

[8] D. W. Knapp, *Behavioral synthesis: Digital system design using the Synopsys behavioral compiler*, Prentice-Hall, Inc., 1996.

[9] J. Ozer. (2014, January/February). *How to Produce High-Quality H.264 Video Files* [Online]. Available: http://www.streamingmedia.com/Articles/Editorial/Featured-Articles/How-to-Produce-High-Quality-H.264-Video-Files-94216.aspx

[10] J. Han, H. Chen, J. Liang, P. Zhu, Z. Yang, F. Lombardi, "A stochastic computational approach for accurate and efficient reliability evaluation." *IEEE Trans. on Computers*, vol. 63, no. 6, pp. 1336 – 1350, June 2014.