A Fast Fault-Tolerant Architecture for Sauvola Local Image Thresholding Algorithm Using Stochastic Computing

M. Hassan Najafi and Mostafa E. Salehi

***Abstract*— Binarization plays an important role in document image processing, particularly in degraded document images. Among all local image thresholding algorithms, Sauvola has excellent binarization performance for degraded document images. However, this algorithm is computationally intensive and sensitive to the noises from the internal computational circuits. In this paper, we present a stochastic implementation of Sauvola algorithm. Our experimental results show that the stochastic implementation of Sauvola needs much less time and area and can tolerate more faults, while consuming less power in comparison with its conventional implementation.**

***Index Terms*— Fault-tolerant computing, image binarization, Sauvola image thresholding, stochastic computing (SC).**

I. Introduction

Document binarization as the first step in optical character recognition systems has been an active research area for many years. The simplest way to accomplish this binarization is thresholding, which selects a threshold value and then all pixel intensities above/below this threshold are set to 1 (background)/ 0 (foreground) [1]. Thresholding algorithms have been divided into global and local methods. In global methods, a single threshold is selected for the whole image, whereas for local methods, each pixel threshold is selected according to its neighbors in local region. Global methods, such as Otsu [2], are often very fast, and give good results for typical scanned documents. However, in the cases that illumination over the document is not uniform, global methods tend to produce marginal noise along the page borders. Local methods, such as Sauvola [3], try to solve the global method problems using local neighbors information. Local methods usually achieve good results even on severely degraded documents. Since in local methods extracting image features is performed for each pixel, these methods are often slow [4].

Local methods for degraded document images have been studied in [5] and it has been concluded that Sauvola [3] performs better than the others. Although some degraded results have been reported in using Sauvola in images where the gray values of text and nontext pixels are close to each other [6], this method, can usually achieve ideal segmentation performance by considering the influence of value variance of all pixels. Similar to the other local methods, processing time of the Sauvola binarization is usually much long [4].

Sauvola method can somehow tolerate image noises due to snow, rain, or camera shaking. However, it is still sensitive to the noises from the internal circuits, such as the noises due to soft errors, environmental noises, or process, voltage, and thermal variations. Although the conventional fault-tolerance techniques, such as triple modular redundancy (TMR), can increase the fault-tolerance ability of these kinds of methods, the overhead of these techniques is usually more hardware resources and consequently higher power consumption [7].

A solution for high computation time and fault-tolerance problem of the conventional Sauvola implementation is stochastic computing (SC) [8]. SC can gracefully tolerate a very large number of errors at lower cost compared with the conventional TMR techniques [9]. Since all bits have the same significance in stochastic representation, a single bit flip in a long bit stream will result in a small change in the value of the stochastic numbers. Therefore, stochastic circuits are inherently more fault-tolerant. Li and Lilja [7] present a new low-power fault-tolerant architecture based on SC for the kernel density estimation image segmentation algorithm. They extended their work to other image processing algorithms, such as edge detection and noise reduction algorithms in [9].

In this paper, we propose a novel high-speed and, yet, low-power and fault-tolerant stochastic architecture for Sauvola algorithm using SC. To implement this architecture, we introduce a new 9-to-1 and 81-to-1 stochastic mean circuit (SMC), a novel stochastic standard deviation circuit, and also a new accurate stochastic comparator. We exploit both correlated and uncorrelated streams in designing the standard deviation circuit using only a stochastic square root circuit, an XOR gate, and an AND gate. In contrast with other stochastic image processing architectures proposed in the literature, which could work only with the bipolar bit streams, we work only with the unipolar streams to increase the quality of results. Solving the latency problem and also reducing the sensitivity of this method to soft errors are other main contributions of this brief.

Section II introduces the background of Sauvola method and SC. Section III presents the selected parameters, and introduces both conventional and our proposed stochastic implementations. Finally, the experimental results and conclusion are discussed in Sections IV and V.

II. Background

A. Sauvola Local Thresholding Algorithm

In Sauvola, the threshold $t(x, y)$ is computed using the mean $m(x, y)$ and the standard deviation $s(x, y)$ of the pixel intensities in a $W * W$ window centered on the pixel (x, y)

$$t(x, y) = m(x, y) * \left[1 + K\left(\frac{s(x, y)}{R} - 1\right)\right] \tag{1}$$

where R is the maximum value of the standard deviation (often 128 for gray-scale documents), and K is a parameter, which takes a positive value in the range [0.2, 0.5] [4]. To compute $t(x, y)$, local mean and standard deviation have to be computed for all image pixels. Computing $m(x, y)$ and $s(x, y)$ in a traditional way results in a computational complexity of $O(W^2N^2)$ for an $N * N$ image in a $W*W$ window [4]. Our proposed architecture not only significantly improves the computation time of the conventional implementation but also improves the required area, power consumption, and capability of tolerating faults.

Manuscript received June 7, 2014; revised November 12, 2014 and March 1, 2015; accepted March 15, 2015. Date of publication May 8, 2015; date of current version January 19, 2016. This work was in part suported by the Institute for Research in Fundamental Sciences (IPM), Tehran, Iran, under Grant CS1393-4-14.

The authors are with the School of Electrical and Computer Engineering, University of Tehran, Tehran 14395-515, Iran, and also with the School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran 19395-5746, Iran (e-mail: mhassannajafi@ut.ac.ir; mersali@ut.ac.ir).

Digital Object Identifier 10.1109/TVLSI.2015.2415932

B. Stochastic Computing

In SC, computations in the deterministic Boolean domain are transformed into probabilistic real domain [7]. In this approach, numbers are represented by streams of random bits in two simple formats: 1) unipolar and 2) bipolar. Bipolar format can deal with negative numbers directly, while given the same stream length, the precision of the unipolar format is twice that of the bipolar format [8]. For accuracy purposes, in this brief, we will work only with the unipolar representation of stochastic bit streams.

1) Stochastic Architecture: The stochastic architectures are usually composed of three parts: 1) the randomizer unit (RU), which generates stochastic bit streams from deterministic input values; 2) the processing unit, which processes the generated bit streams and produces the results in stochastic format; and 3) the de-RU, which converts back the resulting stochastic bit streams to output values in deterministic format [10].

2) Stochastic Operations:

a) Scaled addition and multiplication: Since all numbers in the unipolar format are in [0, 1] interval, we use scaled addition, instead of normal addition. A simple MUX can do this operation. Besides that, in unipolar representation of stochastic bit streams, multiplication can be done just by a simple AND gate. Note that for correct functionality, the input streams of these operations should be uncorrelated [9], [10].

b) Stochastic mean circuit (SMC): To average 2^n input bit streams, we need a 2^n to 1 MUX with n uncorrelated select bit streams, each one representing the 0.5 value.

c) Square root: Toral *et al.* [11] presented a stochastic square root circuit that uses two different pulse streams to represent the same value with a different pattern of pulses. Their circuit is looking for a stochastic pulse stream that tends to the input stream when multiplied by itself [12].

d) Absolute valued subtraction: New studies on correlation between stochastic bit streams have shown that correlation in SC is not always harmful [13], [14]. An XOR gate with independent inputs performs the function $z = x_1(1 - x_2) + x_2(1 - x_1)$. However, when fed with correlated inputs where x_1 and x_2 have maximum overlap of 1s, the circuit computes $z = |x_1 - x_2|$ [13].

III. Implementation of the Sauvola Algorithm

A. Sauvola Parameters

Performance of Sauvola algorithm depends on three parameters: 1) R; 2) K; and 3) the window size [15]. Based on [4], $R = 128$ and $K = 0.5$ could have the best performance of Sauvola for most gray-scale document pictures. The computation cost and the quality of the resultant binary document images produced by Sauvola are very sensitive to the selected window size. Choosing larger than necessary window sizes just incurs higher computation cost while could not increase the quality of the binarized images [3].

Based on our experiments on several image data sets, as exemplified with a sample in Fig. 1, we claim that for most document images, such as the captured images from newspapers, selecting 9×9 as the window size not only can produce acceptable outputs but also can make our implementation more scalable. Scalability gives us the opportunity of changing the size of the window dynamically. Hence, for the rest of this paper, we set 9×9 as the window size.

B. Conventional Implementation

The main step of calculating $t(x, y)$ for each image pixel in Sauvola is to compute $m(x, y)$ and $s(x, y)$. Since we fixed the window size to $9 * 9$, we have to calculate the mean and the standard deviation of 81 local numbers using

$$\text{mean} = \frac{a_1 + a_2 + a_3 + \cdots + a_{81}}{81} \tag{2}$$

$$\text{standard deviation} = \sqrt{|\text{mean}(x^2) - \text{mean}(x)^2|}. \tag{3}$$

To implement the required square root function used in the standard deviation, we use Newton–Raphson, a method for finding successively better approximations to the root of a real value number [16]. Fig. 2 shows the implemented block diagram of the conventional architecture of Sauvola method.

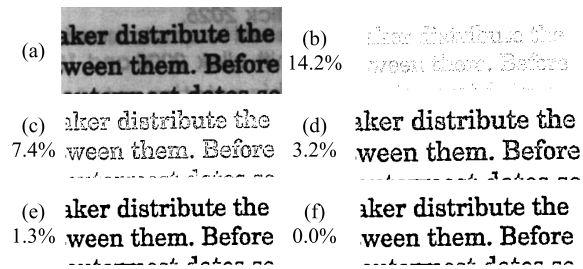


Fig. 1. Window size effect on the quality and quality reduction of binarization of a 360 × 160 document image using Sauvola. (a) Original image. Binarized image using (b) 3 × 3, (c) 5 × 5, (d) 7 × 7, (e) 9 × 9, and (f) 13 × 13 windows.

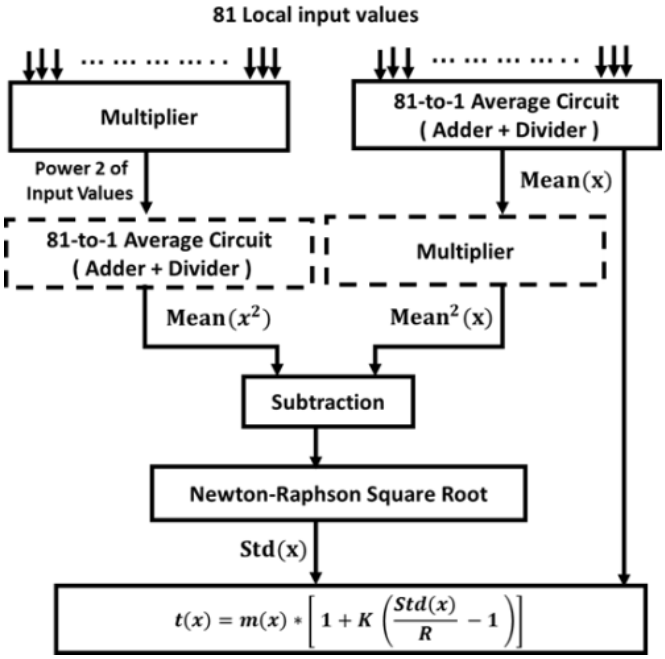


Fig. 2. Simple view of the conventional process of calculating threshold value for each pixel of an input image using Sauvola method.

C. Stochastic Implementation

In stochastic implementation, we have to scale down all pixel intensities from [0, 255] to [0, 1] interval. Therefore, the R constant in the Sauvola equation changes to 1. The new modified Sauvola equation for our stochastic design will be

$$t(x, y) = m \cdot [1 + 0.5(S - 1)] = m \cdot (s + 1)/2. \tag{4}$$

Now, to process all pixels, we need to do three steps: 1) converting pixel values into stochastic streams; 2) generating threshold streams; and 3) determining the output binary values.

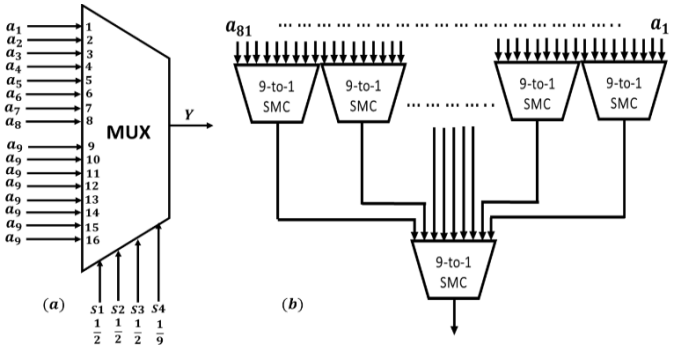


Fig. 3. Proposed (a) 9-to-1 SMC and (b) 81-to-1 SMC.

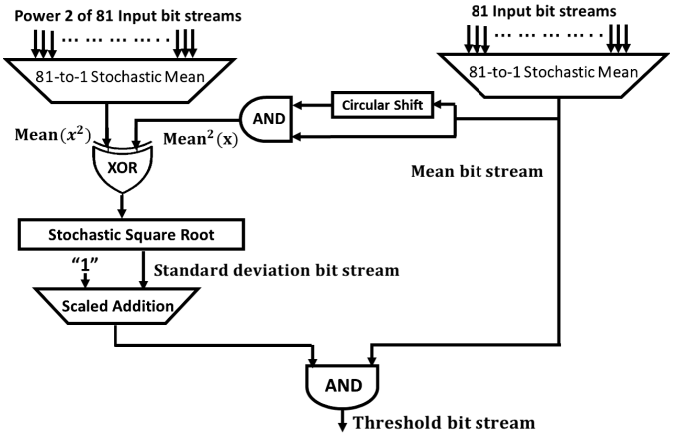


Fig. 4. Simple view of the process of calculating threshold bit stream stochastically for each pixel of an input image using Sauvola method.

1) Phase 1 (Generating Stochastic Bit Streams): To convert pixel intensities from binary format into stochastic streams, we use the stochastic number generator (SNG) presented in [8]. In generating pseudorandom numbers required in this SNG, we used different maximum period linear feedback shift registers (LFSRs) corresponding to each different lengths of streams (n-bit LFSR for 2^n stream).

2) Phase 2 (Generating Threshold Bit Streams):

a) First step (averaging local window bit streams): The SMC discussed in Section II can only be used when we have 2^n input streams, whereas we need to average 81 streams. To build an 81-to-1 SMC, we propose to combine nine 16-to-1 SMCs, each one as a 9-to-1 SMC. We use a simple technique to convert the existing 16-to-1 SMC to a new 9-to-1 SMC. We use a predefined 16-to-1 SMC, connect its first eight inputs to eight uncorrelated input streams, and then connect eight copies of the ninth bit stream, to the eight remaining inputs. Now, by connecting the most significant select line of MUX to a 0.11 corresponded bit stream, as shown in Fig. 3(a), all nine input bit streams will have the same worth for MUX, and their average will be produced at the output. Now, by combining nine separate 9-to-1 SMCs and averaging their results using another extra 9-to-1 SMC, we have the final stream that is exactly the average of 81 input streams. Fig. 3(b) shows our proposed 81-to-1 SMC.

Notice that selecting 16-to-1 MUX as the base of our 9-to-1 SMC gives the architecture the opportunity of increasing the size of local window to $11 \times 11, 13 \times 13$, and even 15×15 dynamically just by making a little change in the select bit streams and also in the number of instantiations required in building the final average circuit.

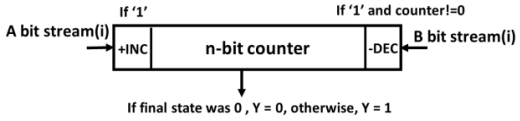
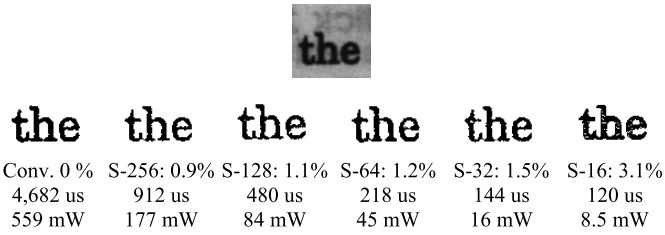


Fig. 5. Our proposed stochastic comparator.



Conv. 0 %	S-256: 0.9%	S-128: 1.1%	S-64: 1.2%	S-32: 1.5%	S-16: 3.1%
4,682 us	912 us	480 us	218 us	144 us	120 us
559 mW	177 mW	84 mW	45 mW	16 mW	8.5 mW

Fig. 6. Our sample (80×75) and output of binarization using the conventional and all stochastic implementations, in addition to their produced error rate in output images, and the required time and the dynamic power for binarization.

b) Second step (generating standard deviation bit stream): Based on (3), in order to determine the standard deviation of some input numbers, we need to have: 1) the average value of the squares and 2) the square of the average values. The proposed 81-to-1 SMC in Fig. 3 can average 81 input streams. Since this average is in stochastic form, generating its square will only need an AND gate with two uncorrelated versions of the input. The simplest way to have these two uncorrelated versions is to just shift the stream for one or a few bits. Besides that, to have the average value of the square of input numbers, we generate the average of the power two of all input streams right after converting pixel intensities to stochastic bit streams using an extra 81-to-1 SMC.

Having $\text{mean}(x^2)$ and $\text{mean}(x)^2$ bit streams, the next step is to calculate $(|\text{mean}(x^2) - \text{mean}(x)^2|)^{1/2}$ stochastically. Using a simple XOR gate, we could do the subtraction part of this function only if the two input bit streams of XOR gate had maximal correlation. We intuitively use the same patterns of select lines for both 81-to-1 SMCs, and hence, the outputs of these stochastic circuits will be almost correlated automatically. Finally, by connecting the outputs of XOR gates to the inputs of the stochastic square root circuit, we have a bit stream, which represents the standard deviation of local 81 input bit streams.

c) Third step (generating threshold bit stream): Considering (4), to generate threshold bit streams, we need to perform two other simple operations. First, we should do a simple scaled addition to convert the s bit stream to $(s + 1)/2$. Second, multiplying m and $(s+1)/2$ bit streams by a simple AND gate. Fig. 4 shows the process of producing the threshold bit stream for each pixel of image in Sauvola method.

3) Phase 3 (Generating Output Binary Values): Now we reach to the final step, the estimation of 0 or 1 output binary values by comparing the produced threshold bit stream with the corresponded pixel intensity bit stream. This function will be the responsibility of a specific circuit, stochastic comparator.

Li and Lilja [17] proposed a stochastic comparator, which uses an FSM-based stochastic tanh function developed in [18] in addition to a stochastic-scaled subtraction unit [9] to compare two bipolar streams and produce an approximate 0 or 1 stream. The comparator proposed by Li and Lilja can only produce a perfect 0 or 1 output stream when the difference between the two input streams is more than 0.2 and both the inputs are in the bipolar format. This would be a major problem for us if we wanted to use their comparator in our stochastic architecture. Therefore, we propose a new stochastic comparator

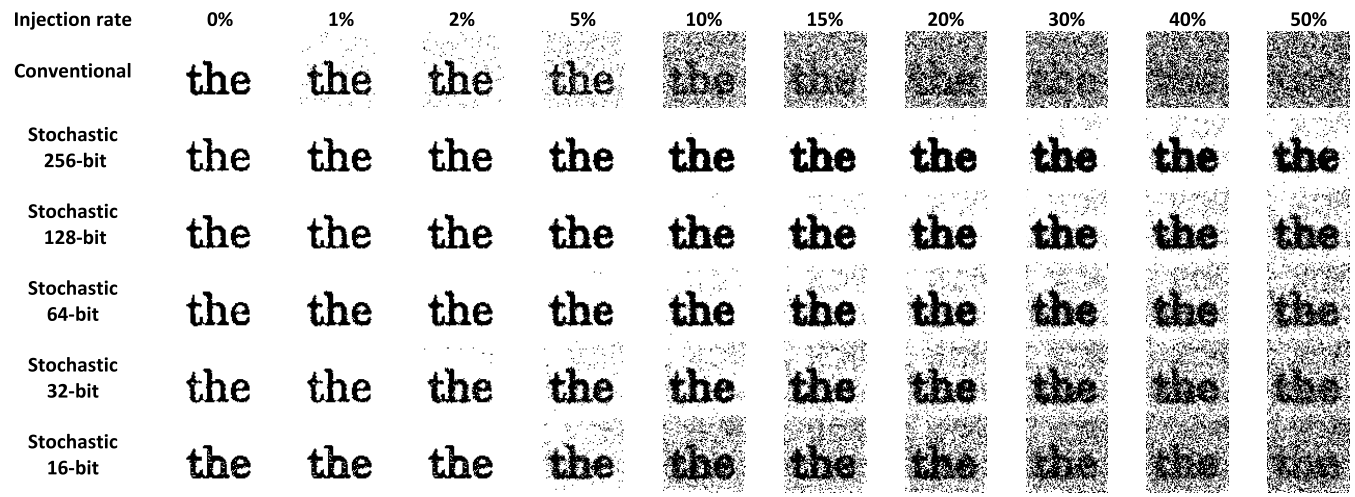


Fig. 7. Comparison of the fault-tolerance capabilities of different hardware implementations for the Sauvola image thresholding algorithm.

TABLE I
COMPARISON OF HARDWARE USAGE [NUMBER OF LOOKUP TABLE (LUT) FLIP-FLOP PAIRS USED] AND DELAY OF THE CONVENTIONAL TO THE STOCHASTIC IMPLEMENTATIONS

	# of LUT Flip Flop pairs used			Maximum Path
	Stream generator	Sauvola architecture	Total	Delay (ns)
Conventional	-	10,497	10,497	821.97
Stoch. 256	219	2,335	2,554	574.95
Stoch. 128	99	973	1,072	269.32
Stoch. 64	47	315	362	122.01
Stoch. 32	15	38	53	52.82
Stoch. 16	7	12	19	5.916

TABLE II
AVERAGE OUTPUT ERROR OF DIFFERENT STOCHASTIC IMPLEMENTATIONS WHEN FAULT INJECTION RATE CHANGES FROM 0% TO 50% IN BINARIZATION OF SELECTED SAMPLE DOCUMENT IMAGE

Inject rate (%)	0	1	2	5	10	15	20	30	40	50
Conventional	0	1.7	3.4	7.9	32	35	40	41	46.2	46.4
Stoch. 256	0.9	1.0	1.1	2.4	4.2	5.5	6.4	8.1	9.2	10.2
Stoch. 128	1.1	1.1	1.5	2.5	4.8	5.8	7.4	9.0	11.5	14.8
Stoch. 64	1.2	1.6	1.7	2.9	4.9	7.6	9.6	13.8	18.1	22.2
Stoch. 32	1.5	1.6	2.2	5.3	7.9	13	15	22.6	27	31.5
Stoch. 16	3.1	3.1	3.2	8.0	23	25	29	34	38	41

that: 1) can work with the unipolar presentation of bit streams and 2) has the ability of producing accurate 0 and 1 output streams even for nearly equal input bit streams.

Our proposed comparator (Fig. 5) is based on a simple counter. According to the length of the input streams (2^n), we choose an n bit counter. Now, by starting from the first bit of A stream, if it is 1, we increase the counter by one unit, and continue this process for all bits of A. After processing A, we start with the first bit of the second stream B. If this bit is 1 and the counter is not empty, we decrease the counter by one unit, and continue for all B bits. By this simple two stage process, we would know that which one of two input streams has more 1s and so is greater than the other one. In the last step, after processing A and B streams, if the counter was showing zero value, the output of the proposed comparator will be zero, otherwise the output will be a stream with all bits 1.

IV. EXPERIMENTAL RESULTS

To build the conventional implementation of Sauvola, we used MATLAB and Mathworks HDL coder to implement the algorithm and convert the codes to Verilog HDL. The stochastic implementation of Sauvola is also implemented using Verilog. To have a tradeoff between accuracy, hardware resource, and computation time, we implemented all 16-, 32-, 64-, 128-, and 256-bit stream stochastic architectures. Notice that in our implementations, we will reuse a single window circuit to move across all pixels of the image. All architectures have been synthesized and placed and routed on the Xilinx Virtex6 XC6VLX760-2FF1760 FPGA as the target device. After successful verification of all implementations, we selected a sample gray-scale degraded document image (Fig. 6) to compare different implementations from performance, area, power, and fault-tolerance points of view.

A. Performance Comparison

Fig. 6 shows the results of binarization, comparing the conventional and all stochastic implementations. As can be seen in this figure, the stochastic implementation, in the worst case, had just 3.1% error rate, which is a negligible error rate for human eyes [7]. The required execution times for binarization of our sample image are also shown in the figure. The stochastic implementation reduces the required execution time ∼40 times in the 16-bit stream architecture.

B. Power Consumption Comparison

To estimate the dynamic power consumption, we used Xilinx XPower Analyzer in addition with the value change dump files extracted from the post place and route simulations. As shown in Fig. 6, the conventional approach consumes ∼3 times more power than the 256-bit streams stochastic approach, and ∼60 times more than 16-bit streams for binarization of the sample image. Since the leakage power is proportional to area [13], the leakage power of the stochastic circuits is much lesser than the conventional implementation.

C. Hardware Resource Comparison

Implementation reports from the synthesis tool can give us enough information to compare the required hardware resources of all the approaches. As shown in Table I, the conventional implementation requires ∼500 times more hardware resources than the 16-bit stochastic implementation.

D. Fault-Tolerance Comparison

We injected noise in the same way, as proposed in [10]. Soft errors are simulated by independently flipping a given fraction

of the input or output bits of each computing element. For example, a soft error rate of 10% means that 10% of the total number of the signal bits is randomly chosen and flipped [7]. The images in Fig. 7 visually and the values in Table II statistically illustrate the fault-tolerance capabilities of the stochastic implementations. Note that when the injected soft error rate is $>2\%$, all stochastic implementations, even the one with 16-bit bit streams outperform the conventional implementation.

V. Conclusion

In this paper, we proposed a novel fault-tolerant low-power fast architecture for the computation intensive Sauvola local image thresholding algorithm based on the SC approach. Our proposed architecture not only is able to tolerate high rates of faults in noisy environments but also requires less area and consumes less power than the conventional implementation of this algorithm. To implement the Sauvola method using stochastic approach, we introduced a novel 9-to-1 and also an 81-to-1 SMC to average 9, and 81 input bit streams. Furthermore, we proposed a new stochastic standard deviation circuit and a new stochastic comparator for the unipolar stochastic bit streams. Implementation reports show that reducing the window size from 9×9 to 7×7 could reduce the hardware resource usage $\sim$52%; however, for the quality reasons, we made our architecture based on 9×9 window size.

References

[1] P. Stathis, E. Kavallieratou, and N. Papamarkos, "An evaluation technique for binarization algorithms," *J. Univers. Comput. Sci.*, vol. 14, no. 18, pp. 3011–3030, 2008.

[2] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Syst., Man, Cybern.*, vol. 9, no. 1, pp. 62–66, Jan. 1979.

[3] J. Sauvola and M. Pietikäinen, "Adaptive document image binarization," *Pattern Recognit.*, vol. 33, no. 2, pp. 225–236, 2000.

[4] F. Shafait, D. Keysers, and T. M. Breuel, "Efficient implementation of local adaptive thresholding techniques using integral images," in *Proc. 15th Doc. Recognit. Retr. Conf. (DRR-2008), Part IS&T/SPIE Int. Symp. Electron. Imag.*, vol. 6815. San Jose, CA, USA, Jan. 2008.

[5] E. Badekas and N. Papamarkos, "Estimation of proper parameter values for document binarization," in *Proc. 10th IASTED Int. Conf. Comput. Graph. Imag.*, 2008, pp. 202–207.

[6] K. Khurshid, I. Siddiqi, C. Faure, and N. Vincent, "Comparison of Niblack inspired binarization methods for ancient documents," *Proc. SPIE, Doc. Recognit. Retr. XVI*, vol. 7247, 2009.

[7] P. Li and D. J. Lilja, "A low power fault-tolerance architecture for the kernel density estimation based image segmentation algorithm," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Archit. Process. (ASAP)*, Sep. 2011, pp. 161–168.

[8] A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 2s, pp. 1–19, 2013.

[9] 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. Very Large Scale Integr. (VLSI) Syst.*, vol. 22, no. 3, pp. 449–462, Mar. 2014.

[10] W. Qian, X. Li, M. D. Riedel, K. Bazargan, and D. J. Lilja, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Trans. Comput.*, vol. 60, no. 1, pp. 93–105, Jan. 2011.

[11] S. L. Toral, J. M. Quero, and L. G. Franquelo, "Stochastic pulse coded arithmetic," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, Geneva, Switzerland, vol. 1. May 2000, pp. 599–602.

[12] B. R. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*, vol. 2. New York, NY, USA: Plenum, 1969.

[13] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Proc. 50th ACM/EDAC/IEEE Design Autom. Conf. (DAC)*, May/Jun. 2013, pp. 1–6.

[14] A. Alaghi and J. P. Hayes, "Exploiting correlation in stochastic circuit design," in *Proc. IEEE 31st Int. Conf. Comput. Design (ICCD)*, Oct. 2013, pp. 39–46.

[15] M. Cheriet, R. F. Moghaddam, and R. Hedjam, "A learning framework for the optimization and automation of document binarization methods," *Comput. Vis. Image Understand.*, vol. 117, no. 3, pp. 269–280, 2013.

[16] G. Upton and I. Cook, *A Dictionary of Statistics*, 3rd ed. New York, NY, USA: Oxford Univ. Press, 2014.

[17] P. Li and D. J. Lilja, "Using stochastic computing to implement digital image processing algorithms," in *Proc. IEEE 29th Int. Conf. Comput. Design*, Oct. 2011, pp. 154–161.

[18] B. D. Brown and H. C. Card, "Stochastic neural computation. I. Computational elements," *IEEE Trans. Comput.*, vol. 50, no. 9, pp. 891–905, Sep. 2001.