SN Computer Science (2023) 4:99
https://doi.org/10.1007/s42979-022-01470-w

ORIGINAL RESEARCH

Development of Noise Tolerant Document Image Binarization Technique Employing an Accurate Square Root Circuit

Shyamali Mitra[1] · **Debojyoti Banerjee**[1] · **Mrinal K. Naskar**[2]

Received: 5 May 2022 / Accepted: 21 October 2022 / Published online: 17 December 2022

Abstract

Stochastic computing (SC) is a re-emerging paradigm that is inherently immune towards noise and shows low area and power implementation of conventional binary logic based approaches. A stochastic square root circuit is proposed, which shows faster convergence and less hardware area compared to the state-of-the-art methods and can be used for efficient real time implementation of various algorithms that employ the square root circuit. In this paper, the Nick's binarization algorithm is introduced as a case study to identify its promise using various statistical metrics. This paper also introduces structural modifications of some of the basic stochastic circuits obeying certain boundary conditions. A squarer circuit requiring the least number of delay elements is calculated to reduce the hardware cost.

Keywords Document image binarization · OCR · Stochastic logic elements · Nick algorithm · Square root circuit · Noise tolerant

Introduction

Image binarization is a critical preprocessing step in document processing applications, including Optical Character Recognition (OCR) and degraded document image recovery. Traditional binarization techniques such as Niblack, Sauvola, Nick etc. generally do not perform well on historical data as they are embedded with different types of noise. To address this problem, researchers are exploring an alternate paradigm for document image binarization to achieve improved noise tolerant behaviour [1].

We introduce stochastic computing as another paradigm for computing Nick's threshold estimation in this research, which has been shown to be more robust and area efficient for intrinsically noisy document datasets. As a proof of concept, we tested the proposal using the Bickley Diary and Monk Cuper datasets. Our stochastic pipelined architecture shows a huge reduction in design complexity, area, and power consumption while also vastly improving noise and fault tolerance capability. The computation of local threshold is done in a 9×9 pixel-sized window in an image of size $M \times N$ (for both conventional and Stochastic). Another issue is the calculation of the standard deviation within the local window that involves the calculation of square root using the Newton–Raphson method. This requires a large circuit size and computational burden [2]. As a result, increased hardware complexity is a major area of concern in the implementation of the local thresholding algorithms. So, we propose a stochastic square root circuit using the modified blocks that shows faster convergence, greater accuracy and less hardware area compared to the state-of-the-art methods.

The contributions in the present work can be highlighted as; (a) A stochastic square root circuit has been proposed showing faster convergence and greater accuracy compared to the state-of-the-art methods [3, 4], (b) Modified arithmetic logic blocks that give accurate results over the entire input range. The results are verified using suitable graphs,

This article is part of the topical collection "Advances in Applied Image Processing and Pattern Recognition" guest edited by K. C. Santosh.

✉ Shyamali Mitra
shyamalimitra.iee@jadavpuruniversity.in

Debojyoti Banerjee
dbanerjee181@gmail.com

Mrinal K. Naskar
mrinaletce@gmail.com

[1] Instrumentation and Electronics Engineering, Jadavpur University, Kolkata, West Bengal, India

[2] Electronics and Telecommunication Engineering, Jadavpur University, Kolkata, West Bengal, India

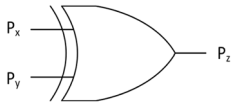

Fig. 1 An XOR gate as absolute subtractor implementing $p_z = |p_x - p_y|$

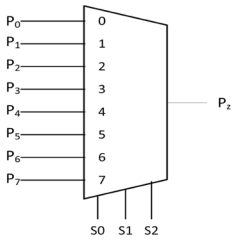

Fig. 2 Mean calculation of stochastic numbers using 8:1 MUX

(c) A stochastic architecture of Nick's binarization algorithm is proposed using the square root and squarer unit which is more robust and noise tolerant than the conventional implementation. Two standard datasets are tested using different statistical parameters.

Stochastic Logic Units (SLU)

In a paradigm proposed by Gaines [5] and Poppelbaum [6], logical computations are done on stochastic numbers. Each real valued number is represented using a stream of $1s$ and $0s$. In a stochastic number X, the probability of 1 in X is denoted as p_x. Then, p_x can be estimated from the ratio $\frac{n_1}{n}$, where n_1 is the number of $1's$ in X, e.g, $p_x = \frac{4}{8}$ may be represented by 01011001. Such form of representation of stochastic numbers is popularly known as Unipolar encoding [5]. Stochastic Logic Unit consists of combinational and sequential elements [5]. To design the square root circuit we use the modified stochastic blocks as well as some basic blocks to reduce the hardware resource utilization of the overall circuit.

Absolute Subtraction

Absolute subtraction of two numbers i.e, $p_z = |p_x - p_y|$ can be performed using an XOR gate acting on correlated inputs [7] (shown in Fig. 1). To generate correlated numbers, same LFSR can be used to generate two numbers or two different LFSRs with a correlator circuit.

Stochastic Mean Circuit (SMC)

2^n input bit streams can be averaged using a $2^n : 1$ MUX with n select bitstreams each representing a certain probability. Figure 2 represents an 8 : 1 MUX calculating the average of 8 inputs as $p_z = (p_0 + p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7)/8$ [8].

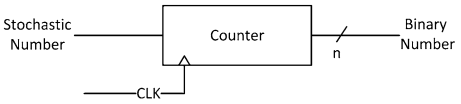

Fig. 3 The Stochastic to binary number converter

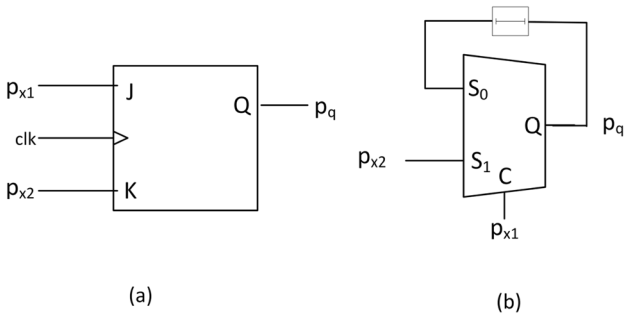

Fig. 4 Division circuits **a** JK Flip flop implementing division **b** CORDIV Block

Stochastic Number Storage

A 2^n-bit SN that is generated from an n-bit binary number can be stored effectively in an n-bit register in binary format for further processing (See Fig. 3). The result of the stochastic computation is stored using a counter that counts the number of 1s in a bitstream and stores it in a register. Thus, for a 256-bit sequence, we need only an 8-bit register for its effective storage [5].

Stochastic Division Circuit

There are various division circuits that are proposed in the literature [3, 5, 9]. Gaines [5] implemented an approximate division circuit using a single JK FF. The state equation that describes Boolean operation of a JK FF is given as $Q^+ = J \cdot \overline{Q} + Q \cdot \overline{K}$ where J and K are the two inputs to the JK flipflop. If the inputs are uncorrelated then using the Lemma 1 of [10], Q^+ is replaced by p_q and $\overline{Q}$ by $(1 - p_q)$. The input stochastic bit stream acting as the divisor (p_{x2}) is given to the K input and the dividend (p_{x1}) is given to the J input of the FF. The JK FF implements an approximate division given by Eq. 1.

$$p_q = \frac{p_{x1}}{p_{x1} + p_{x2}}. \tag{1}$$

However, the circuit in Fig. 4a has a limited accuracy when the input signal probabilities become comparable. Other circuits using the Gaines ADDIE block controlling

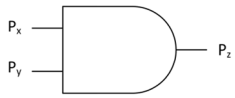

Fig. 5 AND gate implementing $p_z = p_x.p_y$

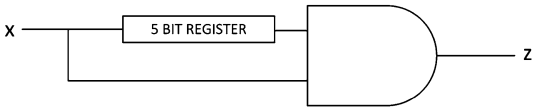

Fig. 6 A Squarer circuit

output in a closed loop negative feedback have also been suggested in the literature, e.g., [5], Chen and Hayes [9] etc.

Multiplication and Squaring Operation

Stochastic multiplication of two uncorrelated inputs as shown in Fig. 5 can be done using an AND gate [5] and is given by, $p_z = p_x p_y$. Two bitstreams are said to be uncorrelated iff, $p_{xy} = p_x p_y$. However, the results are not accurate if the numbers are positively or negatively correlated [7].

However, the squaring operation [11] cannot be performed accurately by simply giving the stochastic bitstream as inputs to the AND gate. Thus, instead of square of SN it produces the number itself [7], i.e, $p_x.p_x = p_x$

This happens due to high auto-correlation between inputs. Hence, it is necessary to decrease correlation between the given input sequence and then multiply using an AND gate (Fig. 6). This can be performed in many ways one of which is using two different LFSRs for generating the same input probabilities and then multiplying using AND gate [3]. This produces accurate results but the hardware cost increases considerably as separate LFSRs are required for generating the inputs.

An alternate solution to this problem was proposed by Gaines [5] using isolators to reduce the correlation between numbers and find square using an AND gate. The number of isolators required to perform the squaring operation with desired accuracy was found. It was observed that number of isolators depends upon the seed polynomial chosen in the LFSR and the progressive precision of the LFSR [12].

In order to estimate the correct and minimum number of decorrelator circuit uncorrelate a given number, an error analysis is conducted. The absolute values for different number of delay elements ($N = 1, 2, 3, 4, 5$) is calculated and is plotted in Fig. 7. It is observed that for $N = 5$, the sequences exhibit least value of *SCC* and hence the squaring operation can be done accurately. This also allows least number of decorrelator circuits thereby reducing the hardware complexity.

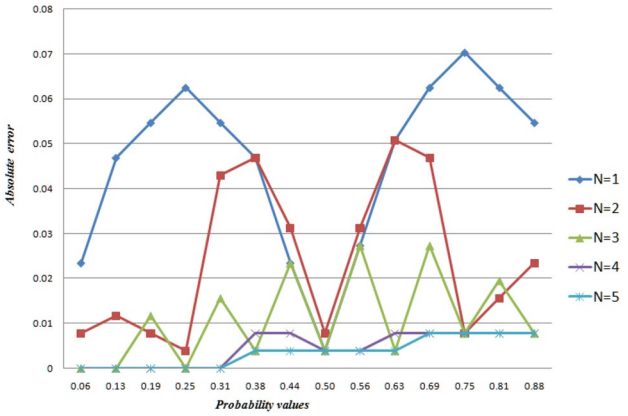

Fig. 7 Absolute error vs Probability values for different delays

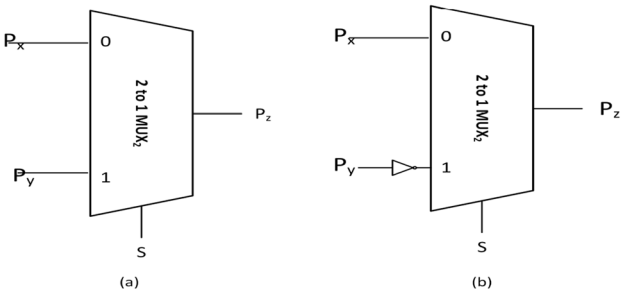

Fig. 8 2:1 MUX implementing **a** scaled addition $p_z = p_x(1-p_s) + p_y p_s$ **b** scaled subtraction $p_z = p_x(1-p_s) + (1-p_y)p_s$

Scaled Addition and Subtraction

Scaled addition and subtraction of correlated and uncorrelated numbers [13] can be done using a 2:1 MUX. The scaling is controlled by the select line bit stream. A constant scaling factor of 0.5 can be obtained by using a toggle flip-flop irrespective of the input probability, as suggested by Gaines [5]. For subtraction, only the subtrahend bitstream is inverted. If each of the bitstream is equally probable, then scaled addition is given by, $p_z = \frac{(p_x+p_y)}{2}$ and for scaled subtraction, output is $p_z = \frac{(p_x+1-p_y)}{2}$ as shown in the Fig. 8. This is elucidated with the help of Examples 1 and 2.

Example 1 Scaled Addition: Consider two SNs with probabilities $p_x = \frac{2}{8}$ and $p_y = \frac{4}{8}$. Three different cases of correlation between the two numbers are considered: (a) If p_x and p_y are positively correlated, i.e, $p_y = \frac{4}{8}$ is 11110000 and $p_x = \frac{2}{8}$

is 11000000 and if the select input $p_s = \frac{4}{8}$ is 10101010 then for scaled addition p_z =11100000 = $\frac{3}{8}$, which is equal to the desired value. (b) If p_x and p_y are negatively correlated, say, $p_y = \frac{4}{8}$ is 11110000 $p_x = \frac{2}{8}$ is 00000011 and the select input $p_s = \frac{4}{8}$ is given by 10101100, then the result of scaled addition $p_z = \frac{3}{8}$ is 01001100. (c) If p_x and p_y are uncorrelated, say, $p_y = \frac{4}{8}$ is 11110000 and $p_x = \frac{2}{8}$ is 10001000 and the select input is given by 10101100, then the output of scaled addition is given by $p_z = \frac{3}{8}$ is 10101000.

Example 2 Scaled Subtraction: Let us consider two SNs that are relatively uncorrelated, say, X and Y with the probabilities $p_x = \frac{5}{8}$ is given by 11111000 and $p_y = \frac{4}{8}$ is given by 10100101 the select input S of the multiplexer is given by 00111001, the output of the multiplexer Z is given by 01111010 whose probability is $p_z = \frac{5}{8}$, which is much closer to the expected result ,i.e, $\frac{p_x+1-p_y}{2}$ is $\frac{9}{16}$. (b) For correlated number the error increases and can be understood with the help following example with stochastic number X given by $p_x = \frac{5}{8}$ represented by 11111000 and Y given by $p_y = \frac{3}{8}$ represented by 11100000 and $p_s = \frac{4}{8}$ input is 11110000, the output Z is given by 11111111 which is $p_z = 1$, whereas the expected output was $\frac{5}{8}$.

Thus, in all the cases of correlation, the relation between the two inputs has been taken into account whereas the effect of the third input i.e., the select input has been disregarded. But in reality, the combinations between these three inputs dictate the condition for accurate addition and subtraction by MUX. If the condition is not satisfied, then the result of scaled addition and subtraction deviates from the desired value.

Adder Unit

MUX can be used to implement basic arithmetic functions like scaled addition and subtraction. But MUX implementing these basic functions produces inaccurate results [13]. There are various modified circuits found in literature that are designed to generate more accurate results irrespective of the input conditions [14]. Lee et al. [15] proposed an adder circuit using a Toggle flip flop that switches between 0 and 1 when the input is 1. A necessary condition guides the accurate operation of the adder circuit.

Theorem 1 *Scaled addition in the expression,* $Y_{sum} = \frac{p_A + p_B}{2}$ *between two inputs a, b in association with select line s of a multiplexer satisfies the probability expression,* $i_{001} + i_{110} = i_{010} + i_{101}$.

Proof Consider the three inputs a, b and s represented in stochastic numbers as A,B and S respectively. The output of the two input multiplexer [5, 6] is given by,

$$Y_{\text{sum}} = \frac{p_A + p_B}{2}. \tag{2}$$

Using the truth table representation of the MUX, Y can be given as

$$Y = i_{010} + i_{011} + i_{101} + i_{111}. \tag{3}$$

The probabilities of each of the input bitstreams are given by the number of 1's occurring in each bit stream. Thus probability of the input bitstreams A,B, S can be given by Eqs. 4, 5 and 6 respectively as

$$p_a = i_{010} + i_{011} + i_{110} + i_{111} \tag{4}$$

$$p_b = i_{001} + i_{011} + i_{101} + i_{111} \tag{5}$$

$$p_s = i_{100} + i_{101} + i_{110} + i_{111}. \tag{6}$$

The output of multiplexer can also be obtained using PTM through the matrix product of A and V_{MUX} and is given by

$$Y = i_{010} + i_{011} + i_{101} + i_{111}. \tag{7}$$

Substituting p_a and p_b from Eqs. 4 and 5 into Eq. 2.

$$Y_{sum} = \frac{i_{001} + i_{010}}{2} + i_{011} + \frac{i_{101} + i_{110}}{2} + i_{111}. \tag{8}$$

Now, comparing Eqs. 8 and 7 the following relation can be obtained.

$$i_{001} + i_{110} = i_{010} + i_{101}; \tag{9}$$

Equation 9 dictates that the condition that eventuates as a result of an association between input lines and the select input bitstream of Fig. 9. It is observed that the select line

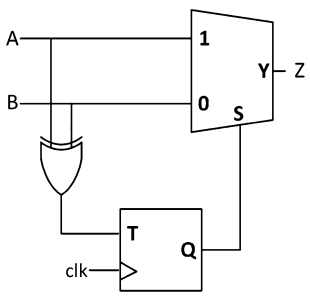

Fig. 9 The adder circuit

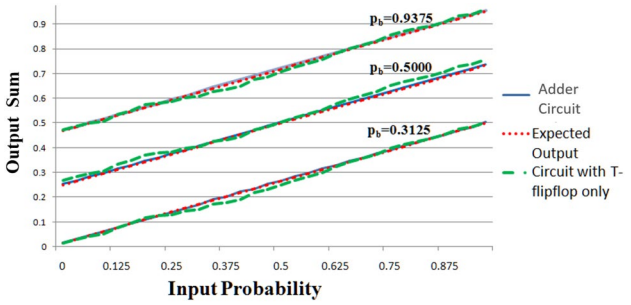

Fig. 10 The output comparison of the adder circuit

(*S*) of the multiplexer can be suitably implemented using the XOR gate along with a T-FF. The results of the simulation are shown in the Fig. 10. □

Subtractor Unit

MUX can be used to implement scaled subtraction [5] using Theorem 2.

Theorem 2 *Scaled subtraction in the expression,* $Y_{\text{sub}} = \frac{p_A + 1 - p_B}{2}$ *between two inputs a, b in association with select line s of a multiplexer satisfies the probability expression,* $\frac{i_{100} + i_{000}}{2} + i_{110} = i_{101} + \frac{i_{011} + i_{111}}{2}$.

Proof Let us consider that the three inputs a, b and s represented as A,B and S respectively. The expression for the scaled subtraction operation is given by

$$Y_{\text{sub}} = \frac{p_A + (1 - p_B).}{2} \tag{10}$$

The probability of finding a '0' in p_b can be given as $(1 - p_b)$ and can be written as

$$(1 - p_b) = i_{000} + i_{010} + i_{100} + i_{101}. \tag{11}$$

Substituting p_a and p_b from Eqs. 4 and 11 respectively in Eq. 10 we have,

$$Y_{\text{sub}} = \frac{i_{000} + i_{100}}{2} + i_{010} + \frac{i_{011} + i_{111}}{2} + i_{110} \tag{12}$$

Comparing Eq. 12 and 7 the following result is obtained.

$$\frac{i_{100} + i_{000}}{2} + i_{110} = i_{101} + \frac{i_{011} + i_{111}}{2}. \tag{13}$$

This condition is satisfied by using an XOR gate at the input of T-flip flop in the subtractor unit. The subtractor unit satisfying this condition is shown in Fig. 11. It is noted that the subtraction operation can be performed by replacing p_b with $1 - p_b$, which is obtained by passing p_b through an inverter circuit. □

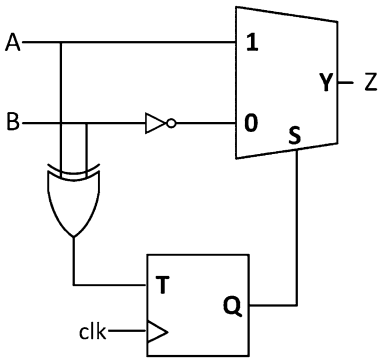

Fig. 11 The subtractor unit

Example 3 Let us consider two correlated numbers A given by $p_a = \frac{4}{8}$ represented by 1111000 and B given by $p_b = \frac{3}{8}$ represented by 11100000 and S input is given by 11110000 the output Z is given by 11111111 which is $p_z = 1$, whereas the expected output was $\frac{4}{8}$. Thus, $N_{010} = 0$, $N_{100} = 0$, $N_{111} = 3$, $N_{110} = 1$, and $N_{000} = 3$. If we place the above condition in the equation we would find $\frac{3}{2} + 1 \neq 0 + \frac{3}{2}$.

Square Root Circuit

Square root is one of the standard and essential blocks in various mathematical operations like solution of non-linear equations [16], calculation of standard deviation [17] in various image processing and machine learning applications [1, 18–22]. There are various square root circuits implemented in conventional binary logic [23] which have higher hardware complexity as well as consume larger area with high consumption of power [24–28]. Also, the number of iterations required to converge to the closest value is larger. So, there is a dire need to look for an alternative computing technique that could implement the same logic with low power and less hardware requirements. Several works exist in literature that implement the square root function making use of the probabilistic logic theory [3, 5].

The stochastic square root circuit was first proposed by Gaines [5] in 1969 exploiting the basic sequential logic element ADDIE that has a counter with negative feedback. An up-down counter integrates the error. But the circuit suffers from several disadvantages. It had slow convergence and

had a large latency as multiple iterations were required to get the square root value. Also, the squaring operation was performed by a single AND gate using a single D flip flop. That could not uncorrelate a given sequence producing inaccurate results in the end. The other drawback was the saturation of n-bit counter. These drawbacks restricted its use in real time applications.

To mitigate the issues with inaccuracy resulting from the squarer circuit, Toral et al. [3] proposed with multiple random number generators to fully eliminate the correlation between the numbers. Thus the squaring operation was not hardware efficient. The time square root $\tau_{\text{root}} = \frac{2^n}{y_2 \cdot F_{\text{clk}}}$, had an exponential relationship with the length of bitstream n. Thus, the circuit cannot be used in real time applications. The accuracy of circuit decreases for low values of input probability. As random number generators are the costliest to implement, the increased number of RNGs in the circuit resulted in an increased cost as well as power consumption.

Wu et al. [29] proposed a stochastic square root circuit by removing the saturating counter as in [3, 5]. Thus, hardware complexity was reduced. It showed accurate results in the higher range of inputs, i.e., 0.7–1. In the lower range of inputs, i.e., 0–0.2 the accuracy of the circuit was not commendable and also the number of clock cycles required to find the desired accuracy was significant.

More recently, an SSRC was proposed by Mitra et al. [4], that showed low latency and less hardware compared to its predecessors in computing the square root value. A variable length bitstream approach in representation of stochastic numbers was adopted that could save significant number of clock pulses. It was observed that in increasing the length of bitstream in every iteration terminates the process in three iterations requiring 448 clock pulses. However, the results obtained with this approach could not exceed a certain accuracy range. So, we propose a square root circuit that is accurate throughout the input range, i.e., the radicands ranging between [0,1] requiring less hardware.

Proposed Square Root Circuit

The stochastic square root computation begins with the binary number converted to a 256-bit stochastic number with the help of LFSR. We subdivided the circuit into two sections: (a) Error Detection Block and (b) Error Correction Block. A MUX is placed to select either the input stochastic number or the approximate square root value obtained as an output from the circuit. A single high going pulse generated by the control unit C, selects '1' input of MUX to give input SN sequence to the squarer circuit. At the trailing edge of the pulse, the output from the 8-bit register is selected, which gives the approximate square root value that needs to be refined further.

Error Detection Block

This is implemented using a squarer circuit, absolute subtractor unit and a 2-bit register as shown in Fig. 12. The squaring operation is performed using a 5-bit shift register as mentioned in Section 5. The output of the squarer and the estimated square root value is compared using an absolute subtractor. The output of absolute subtractor is stored in a 2-bit register, say, $D(D_1D_0)$, to count the number of $1's$. So register D can store a maximum value of 3 in binary. If D_1 bit is 1, then D contains a minimum value of 2, thus the obtained guess value needs to be refined further and this is accomplished when $D1$ bit of D register is 1. If $D1$ bit of register D is 0, it indicates the processing is complete and the guess value is obtained as output value. At the beginning of every iteration the register D is cleared using a control signal IT generated from the control unit C designed for this circuit. The circuit operation is stopped once it is found that $D1 = 0$ at the end of an iteration by disabling the LFSR through enable of the control signal, i.e, $DI = 1$. If $D1 = 1$, the error persists and hence carried over to the Error Correction Block.

Error Correction Block

The error correction block consists of an adder, a division block and an 8-bit register. The division circuit is implemented using the CORDIV block. One of the two inputs of CORDIV block comes from the input SN sequence and another one from the last iteration. The output of CORDIV block is given to one of the input of the MUX to produce the required square root value S_N. The obtained square root is stored in the binary format in an 8-bit register which can be used in next iteration for more accurate computation.

It has been found that the result of the square root value converges to its desired value in two iterations in the majority of the cases. This fact has been utilized and the circuit is implemented using a single 8-bit register. In the case of a single 8-bit register output stochastic number is not updated during the second iteration. This is controlled using the I_S control signal that is generated from the control unit C.

However, if higher accuracy or multiple iterations are required, another 8-bit register $R1$ can be used in conjunction with 8-bit register R that is connected to the output of the multiplexer. The output of R is connected to the register $R1$ which is used along a 8 bit comparator to generate the stochastic number for the next iteration. The output of the register $R1$ is given to the register R at the beginning of the

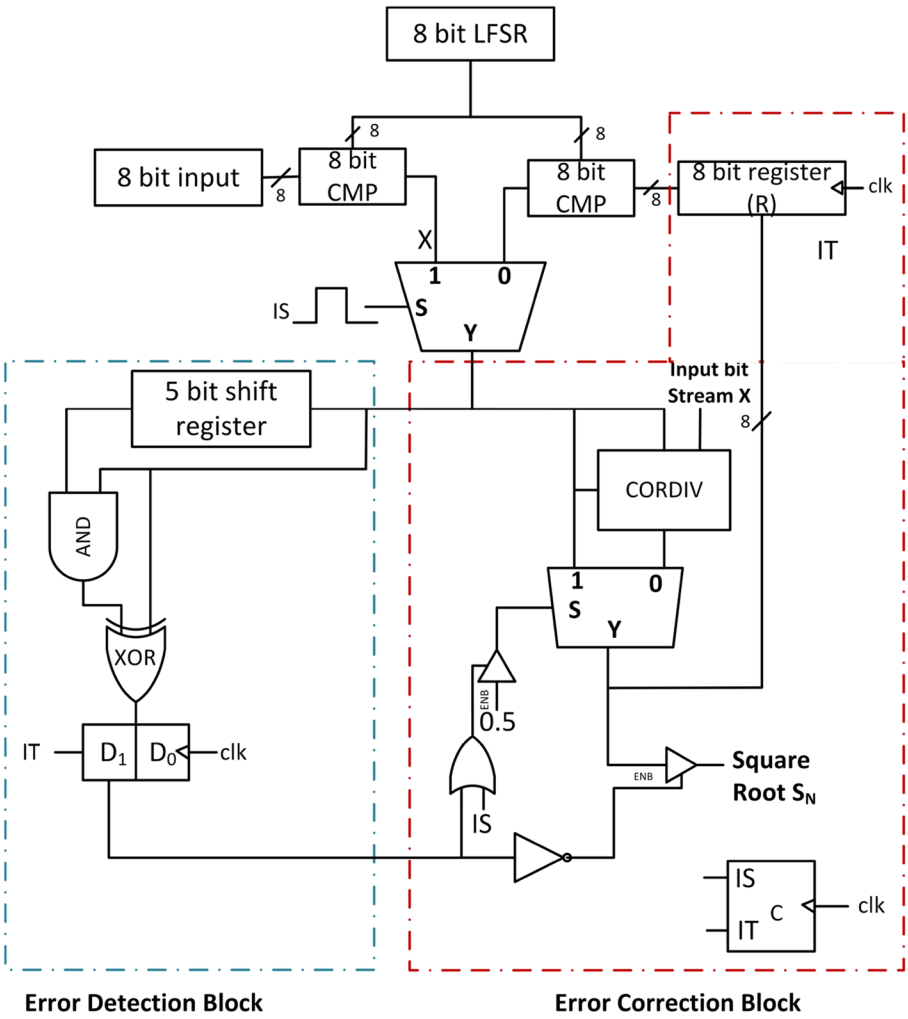

Fig. 12 Proposed square root circuit

next iteration using a control signal *IT* generated from the control circuit *C* as shown in Fig. 12. The circuit shows faster convergence and accurately tracks the actual square root value as shown in Fig.13.

Experimental Results and Analysis

In the proposed work, structure of some of the basic stochastic circuit components are modified and also a stochastic

Algorithm 1 Calculation of square root of a stochastic number

Input: X *//The 256 bit stochastic number generated by comparing 8 bit input binary number with 8 bit LFSR*

Output: R //the square root of the stochastic number

Initialisation : $q = X$;

1: $r = q << 5$ *//Left shift 5 times to uncorrelate the given* S_N *sequence*

2: $s = q * r$ *//Bit wise multiplication to get square of the number*

3: $t =\mid q - s \mid$ *//Bitwise comparison to check the error*

4: **if** $(t >= 2)$ *//if error exist need to iterate to get the exact value* $q = (q/2) + X/(2 * q)$ *//q is the approximate square root value* **then**

5: Go to Step 1

6: **end if**

7: R=q

8: **return** R

square root circuit has been proposed. While designing the squarer circuit we determined experimentally the number of isolators required to uncorrelate the given sequence. Referring to Fig. 7, the light blue line corresponding to $N(delay) = 5$ has the lowest absolute error. For inputs ranging from 0 to 0.31, the SN sequences are completely uncorrelated. From 0.31 to 0.88, though the input SN sequences could not be completely correlated, the absolute error lies within 0.01, indicating the numbers are almost uncorrelated. Reducing the number of isolators thereby to uncorrelate a given stochastic sequence resulted in a significant reduction of hardware complexity. All the circuits are synthesized using Xilinx Spartan 6 *xc*6*slx*4 family of FPGA.

In order to escalate the accuracy of scaled addition and subtraction, modified adder and subtractor have been proposed. Fig. 10 shows the addition operation of the proposed

Table 1 Comparison of accuracies of the proposed method with [3, 4]

SL#	Input	Desired output	Toral et al.	Deviation	Mitra et al. (256- bit)	Deviation	Proposed	Deviation	Improvement w.r.t [3] (in %)	Improvement w.r.t [4] (in %)
1	0.1	0.316	0.492	0.176	0.289	0.027	0.313	0.004	97.881	86.277
2	0.2	0.447	0.508	0.061	0.410	0.037	0.446	0.001	98.558	97.643
3	0.3	0.548	0.551	0.003	0.563	0.015	0.545	0.003	6.601	80.669
4	0.4	0.632	0.621	0.011	0.602	0.031	0.631	0.001	89.152	96.010
5	0.5	0.707	0.699	0.008	0.688	0.020	0.704	0.003	64.923	85.889
6	0.6	0.775	0.762	0.013	0.789	0.014	0.773	0.002	88.300	89.585
7	0.7	0.837	0.828	0.009	0.797	0.040	0.836	0.000	94.528	98.826
8	0.8	0.894	0.898	0.004	0.914	0.020	0.893	0.002	56.463	91.107
9	0.9	0.949	0.945	0.003	0.922	0.027	0.948	0.001	75.661	96.940
10	1	1.000	0.996	0.004	0.988	0.012	0.998	0.002	50.000	83.334

Table 2 Hardware comparison of the proposed method with the method of [3, 4]

Quantity	Number present	Toral et al.		Mitra et al.		Proposed Method	
		Used	% used	Used	% used	Used	% used
Slice registers	4800	45	0.94	17	0.35	17	0.35
Slice LUT's	2400	76	3.17	33	1.38	31	1.29
Used as Logic	2400	75	3.13	–	–	32	1.33
Occupied Slices	600	28	4.67	–	–	12	2.00
LUT-FF pair	76	35	46.05	–	–	14	18.42
Bonded IOB	102	45	44.12	27	26.47	19	18.6

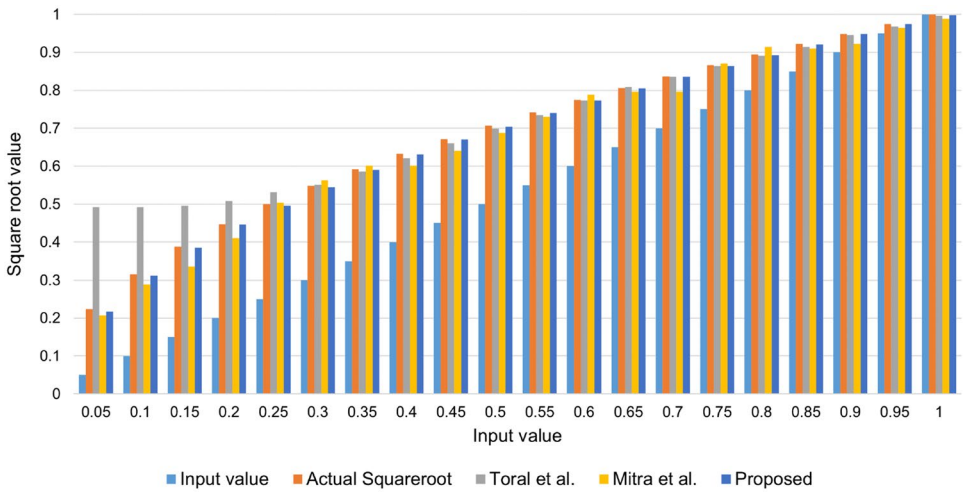

Fig. 13 Accuracy comparison of the proposed square root circuit, [3], and [4]

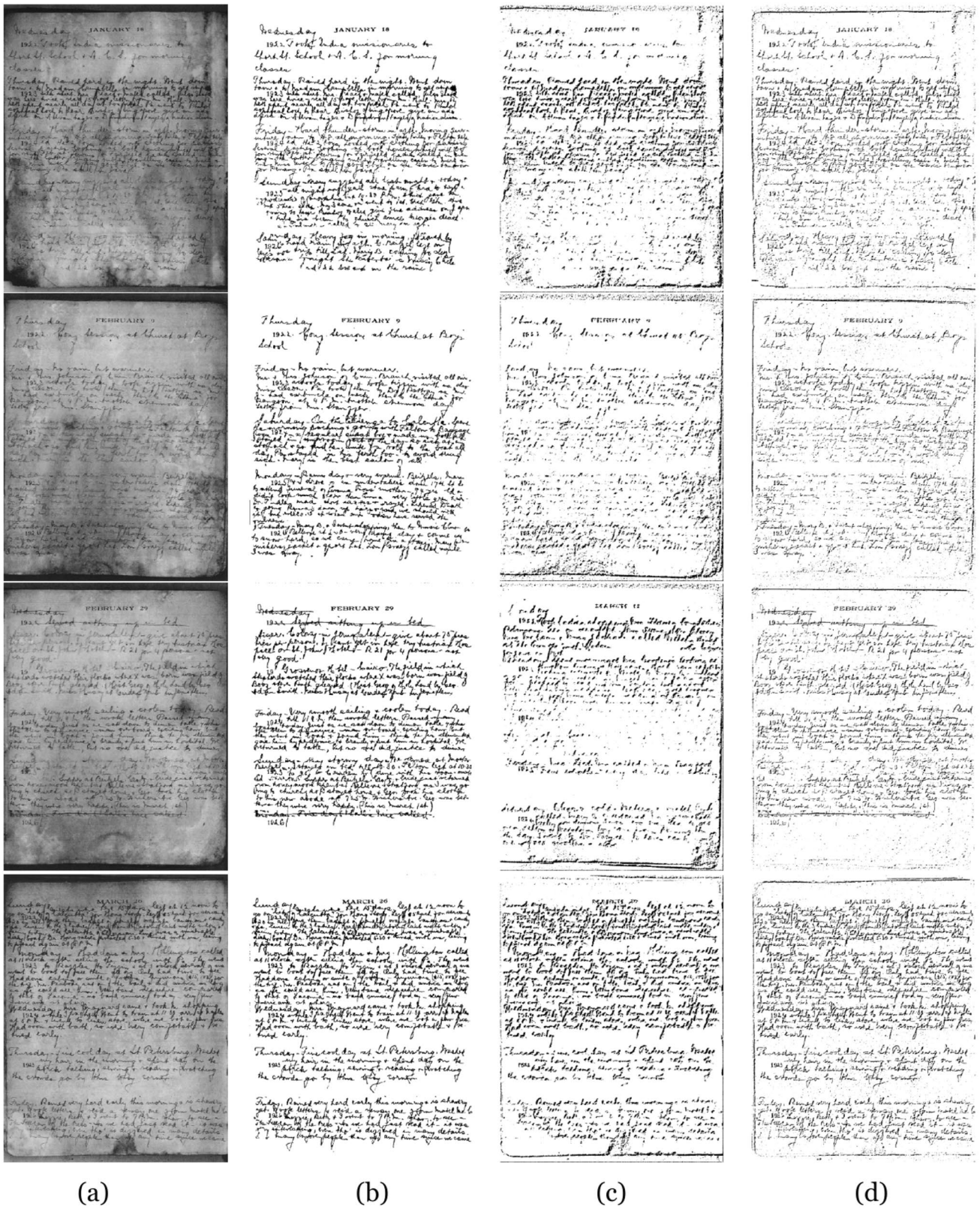

Fig. 14 Output images of Conventional and Stochastic implementation of Nick's algorithm on Bickley Diary dataset; **a** original images; **b** ground truths; **c** Conventional; **d** our method

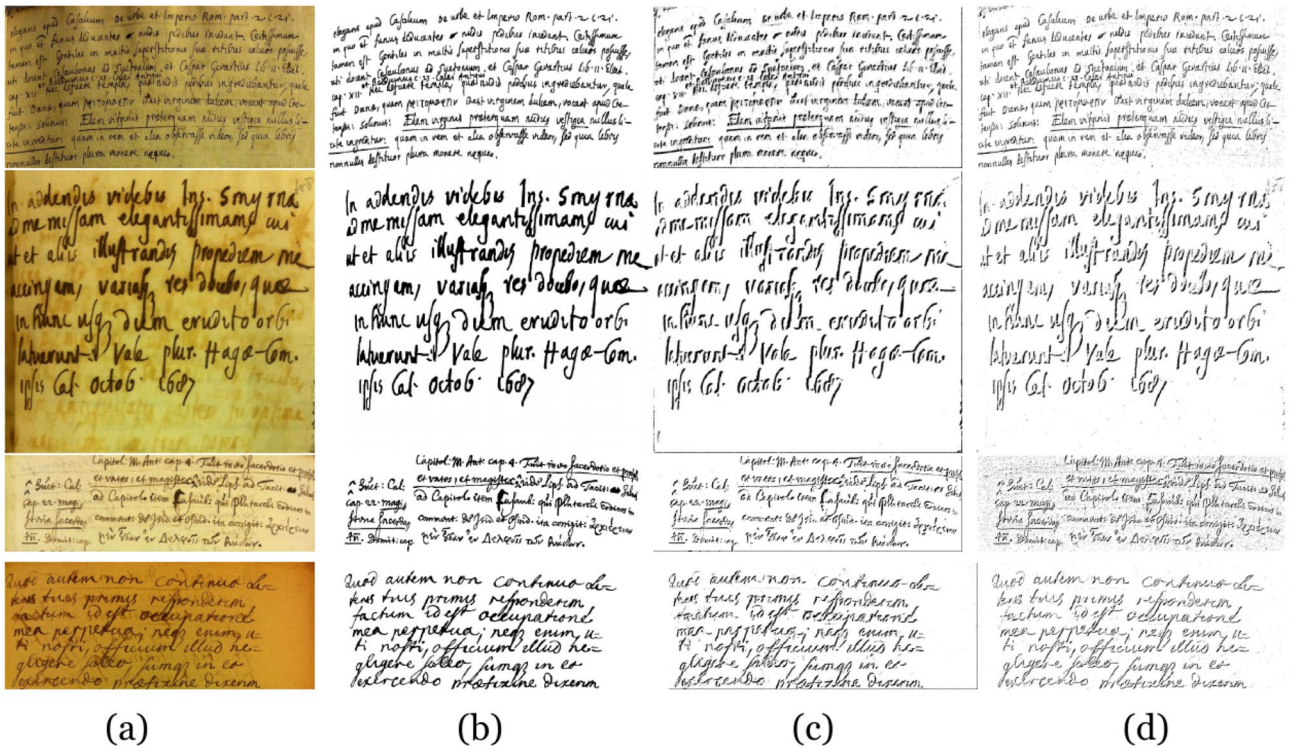

Fig. 15 Output images of Monk Cuper Set (MCS); **a** original images; **b** ground truths; **c** Conventional; **d** Our method

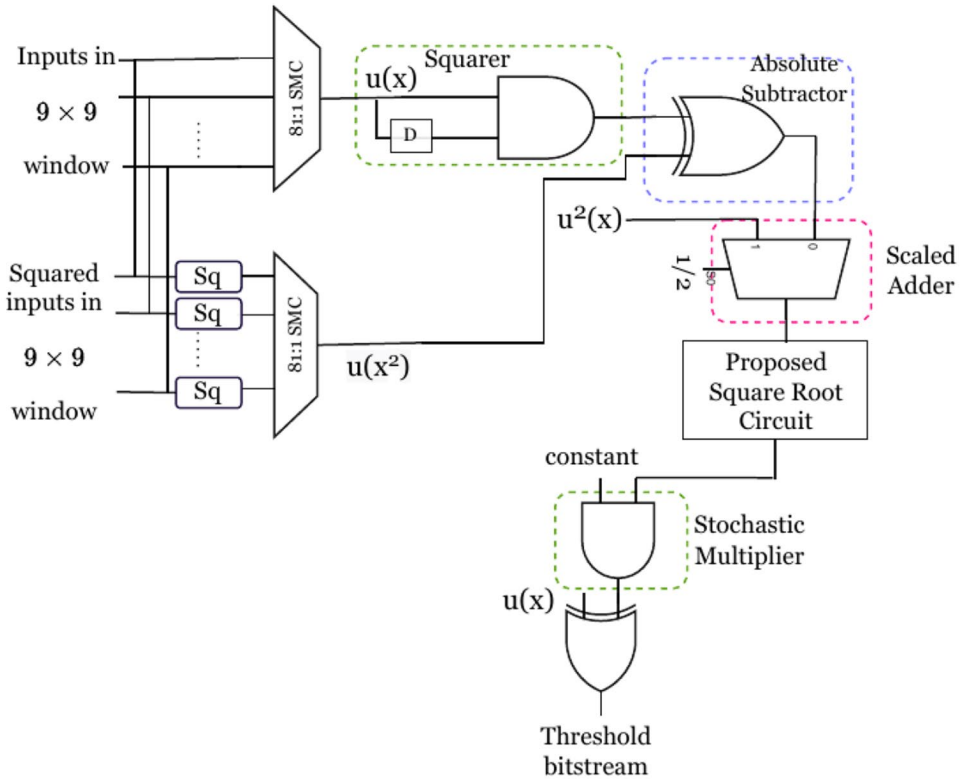

Fig. 16 Stochastic implementation Nick's algorithm

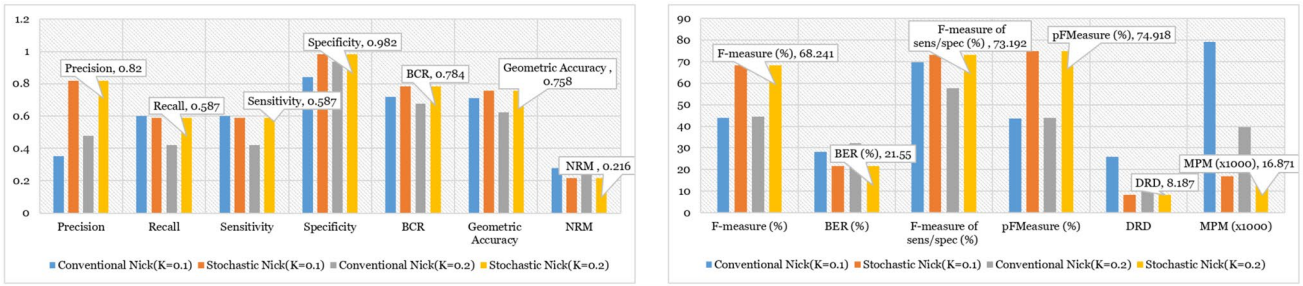

Fig. 17 Average performance metrics calculated on some selected images of Bickley Diary dataset for conventional and stochastic implementation of Nick's algorithm

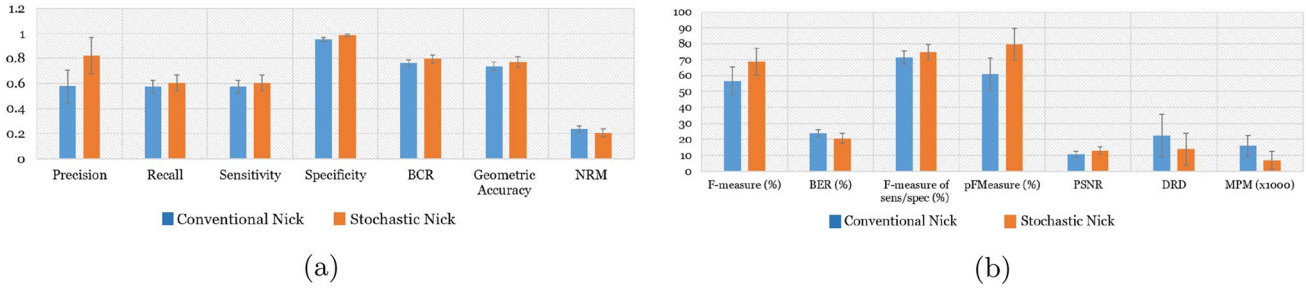

Fig. 18 Average Performance metrics calculated on all images of Monk Cuper Set (MCS) for conventional and stochastic implementation of Nick's algorithm

circuit which reveals much closer approximation to the exact calculation for a 256-bit input sequence compared to the state-of-the-art method [5].

Implementation of the square root circuit using the modified stochastic elements resulted in a more accurate computation of the square root value as shown in Table 1 compared to [3] and [4]. The advantage is that it is accurate throughout the entire range of inputs as shown in Fig. 13. The circuit shows an average improvement of 70.2 percent compared to [3] and 90.6 percent compared to [4] (fixed-bitstream length approach). The proposed circuit also shows faster convergence of the square root value compared to [3]. Also, the hardware utilization of the proposed circuit is less compared to [3] and is comparable with [4] as shown in Table 2. Figure 13 shows the comparison of accuracy of the proposed square root circuit and the circuit of [4] (with variable-bitstream length approach) and [3], which depicts that the proposed circuit is a better choice where accuracy is a concern.

Case Study: Nick's Binarization on Noisy Document Dataset

Stochastic computing is recently used in many image processing applications due to its inherent robustness towards noise. Many attempts can be seen in the literature [30–33]. In Nick, the threshold value is computed using the mean $u(x, y)$ and the variance of the pixel intensity values in a $W \times W$ sized window centered on (x, y) and is given as;

$$T(x, y) = u(x, y) + k\sqrt{\text{variance} + u^2(x, y)}. \quad (14)$$

The square root in Eq. 14 is calculated using the proposed square root circuit and the results are evaluated on two different standard binarization datasets: Bickley Diary dataset and Monk Cuper Set (MCS) [34] for validation. The stochastic architecture of the Nick's method is shown in Fig. 16.

Using several statistical tests and hardware usage metrics, we compare the experimental results of the conventional local binarization approach and the proposed approach. No image enhancing techniques such as noise removal, contrast enhancement, etc. were used in this study. We have chosen 14 different measures [35] namely; Precision, Recall, F-measure, Sensitivity, Specificity, BCR (Balanced classification rate), AUC (Area Under Curve), BER (Balanced error rate), SFmeasure (F-measure based on sensitivity and specificity), Accuracy, GAccuracy (Geometric mean of sensitivity and specificity), pF-Measure (pseudo F-Measure),

Table 3 Different performance metrics calculated (Bickley Diary dataset) for conventional and stochastic implementation of Nick's algorithm at $k = 0.1$

	SL#	Precision	Recall	*F*-measure (%)	Sensitivity	Specificity	BCR	BER (%)	*F*-measure of sens/spec (%)	Geometric Accuracy	p*F*Measure (%)	NRM	PSNR	DRD	MPM (x1000)
Conventional	Image#1	0.306	0.589	40.301	0.589	0.822	0.705	29.463	68.614	0.696	40.072	0.295	6.870	27.978	100.294
	Image#2	0.353	0.613	44.757	0.613	0.845	0.729	27.122	71.026	0.719	44.652	0.271	7.371	25.382	62.347
	Image#3	0.330	0.557	41.434	0.557	0.857	0.707	29.302	67.522	0.691	41.202	0.293	7.524	26.173	88.220
	Image#4	0.293	0.616	39.707	0.616	0.797	0.706	29.375	69.460	0.700	39.462	0.294	6.481	32.726	79.055
	Image#5	0.361	0.592	44.836	0.592	0.860	0.726	27.365	70.158	0.714	44.618	0.274	7.670	24.909	96.988
	Image#6	0.403	0.558	46.837	0.558	0.874	0.716	28.358	68.159	0.699	46.288	0.284	7.765	21.197	71.477
	Image#7	0.401	0.661	49.920	0.661	0.845	0.753	24.679	74.202	0.748	49.666	0.247	7.450	22.152	55.261
Stochastic	Image#1	0.821	0.600	69.318	0.600	0.983	0.791	20.879	74.496	0.768	74.648	0.209	12.035	7.481	20.331
	Image#2	0.821	0.586	68.362	0.586	0.982	0.784	21.596	73.386	0.759	74.988	0.216	11.826	8.300	18.590
	Image#3	0.755	0.462	57.332	0.462	0.981	0.722	27.838	62.839	0.673	65.348	0.278	11.121	10.294	20.015
	Image#4	0.789	0.598	68.049	0.598	0.978	0.788	21.177	74.250	0.765	74.735	0.212	11.701	8.637	17.936
	Image#5	0.859	0.553	67.286	0.553	0.988	0.771	22.947	70.921	0.739	75.951	0.229	11.999	8.134	9.492
	Image#6	0.825	0.576	67.816	0.576	0.981	0.779	22.148	72.561	0.752	74.105	0.221	11.421	8.530	17.119
	Image#7	0.871	0.732	79.524	0.732	0.983	0.857	14.267	83.891	0.848	84.650	0.143	12.918	5.935	14.614

Table 4 Different performance metrics calculated (Bickley Diary dataset) for conventional and stochastic implementation of Nick's algorithm at $k = 0.2$

	SL#	Precision	Recall	*F*-measure (%)	Sensitivity	Specificity	BCR	BER (%)	*F*-measure of sens/spec (%)	Geometric Accuracy	p*F*Measure (%)	NRM	PSNR	DRD	MPM (x1000)
Conventional	Image#1	0.425	0.407	41.552	0.407	0.926	0.667	33.347	56.518	0.614	40.908	0.333	8.703	16.270	50.672
	Image#2	0.508	0.434	46.821	0.434	0.942	0.688	31.182	59.463	0.640	46.621	0.312	9.225	14.625	27.328
	Image#3	0.437	0.321	37.027	0.321	0.948	0.634	36.557	47.977	0.552	36.681	0.366	9.113	16.278	46.116
	Image#4	0.429	0.484	45.507	0.484	0.912	0.698	30.196	63.243	0.664	44.846	0.302	8.556	18.143	34.000
	Image#5	0.486	0.384	42.898	0.384	0.946	0.665	33.508	54.597	0.603	42.627	0.335	9.214	15.737	49.633
	Image#6	0.509	0.361	42.237	0.361	0.947	0.654	34.604	52.250	0.585	41.177	0.346	8.853	15.194	39.835
	Image#7	0.538	0.546	54.191	0.546	0.926	0.736	26.369	68.726	0.711	53.645	0.264	9.025	13.839	29.372
Stochastic	Image#1	0.821	0.600	69.318	0.600	0.983	0.791	20.879	74.496	0.768	74.648	0.209	12.035	7.481	20.331
	Image#2	0.821	0.586	68.362	0.586	0.982	0.784	21.596	73.386	0.759	74.988	0.216	11.826	8.300	18.590
	Image#3	0.755	0.462	57.332	0.462	0.981	0.722	27.838	62.839	0.673	65.348	0.278	11.121	10.294	20.015
	Image#4	0.789	0.598	68.049	0.598	0.978	0.788	21.177	74.250	0.765	74.735	0.212	11.701	8.637	17.936
	Image#5	0.859	0.553	67.286	0.553	0.988	0.771	22.947	70.921	0.739	75.951	0.229	11.999	8.134	9.492
	Image#6	0.825	0.576	67.816	0.576	0.981	0.779	22.148	72.561	0.752	74.105	0.221	11.421	8.530	17.119
	Image#7	0.871	0.732	79.524	0.732	0.983	0.857	14.267	83.891	0.848	84.650	0.143	12.918	5.935	14.614

Table 5 Different performance metrics calculated (Monk Cuper Set (MCS)) for first five images for conventional and stochastic implementation of Nick's algorithm at $k = 0.2$

SL#	Method#	Precision	Recall	F-measure (%)	Sensitivity	Specificity	BCR	BER (%)	*F*-measure of sens/spec (%)	Geometric Accuracy	p*F*Measure (%)	NRM	PSNR	DRD	MPM (×1000)
Image#1	Conventional	0.565	0.577	57.083	0.577	0.931	0.754	24.596	71.229	0.733	58.493	0.246	9.354	14.276	23.056
	Stochastic	0.903	0.649	75.523	0.649	0.989	0.819	18.092	78.372	0.801	88.307	0.181	12.496	6.66	6.005
Image#2	Conventional	0.395	0.605	47.797	0.605	0.951	0.778	22.202	73.969	0.759	49.001	0.222	11.747	25.962	21.173
	Stochastic	0.584	0.666	62.246	0.666	0.975	0.821	17.949	79.153	0.806	68.847	0.179	13.885	16.172	11.822
Image#3	Conventional	0.587	0.611	59.873	0.611	0.933	0.772	22.802	73.849	0.755	63.409	0.228	9.561	18.872	25.946
	Stochastic	0.84	0.622	71.443	0.622	0.982	0.802	19.847	76.112	0.781	82.999	0.198	11.732	11.063	7.552
Image#4	Conventional	0.546	0.581	56.336	0.581	0.927	0.754	24.562	71.468	0.734	58.769	0.246	9.288	19.309	13.83
	Stochastic	0.828	0.686	75.021	0.686	0.979	0.832	16.772	80.659	0.819	84.12	0.168	12.238	9.498	9.446
Image#5	Conventional	0.664	0.564	61.011	0.564	0.917	0.741	25.923	69.871	0.719	66.796	0.259	7.904	19.21	33.896
	Stochastic	0.892	0.611	72.519	0.611	0.979	0.795	20.523	75.227	0.773	82.73	0.205	9.828	11.934	18.551

NRM (Negative rate metric), PSNR (Peak signal-to-noise ratio), DRD (Distance reciprocal distortion metric), MPM (Misclassification penalty metric) to compare the results.

We have experimented using two values of $k = 0.1$ and 0.2. Several metrics are calculated, that denote that the stochastic implementation in Tables 3, 4 and 5 are close to the ideal values of each. The graphs in Fig. 17a, b show the average values of the selected metrics of all the images of Bickley Diary dataset at $k = 0.1$ and $k = 0.2$. It is observed that all the metrics perform well for $k = 0.2$, so the experiment on the Monk Cuper Set (MCS) is done only on $k = 0.2$. In Fig. 18a, b, we have plotted the average values of the selected metrics of all the images and the standard deviation of the Monk Cuper Set. Some of the sample images of the Bickley Diary dataset and Monk Cuper Set along with the ground truth and binarization outcomes are shown in Figs. 14a–d and 15a–d.

Conclusion

The proposed stochastic square root circuit is used in this study to stochastically implement Nick's binarization method on noisy document images in order to better handle noise-related issues. We have also tweaked some logic blocks to improve individual accuracy and reduce the amount of hardware required. We showed that the performance of stochastic Nick is better than that of the current method. An array of datasets can benefit from the proposed approach because stochastic computing is able to handle both the extrinsic and intrinsic noise of an image. The key to this result is the incorporation of bitwise operations into a stochastic model, which can then be generalised to produce an efficient architecture for a wide range of applications.

Declarations

Conflict of Interest There is no conflict of interest with others.

References

1. Mitra S, Santosh KC, Naskar MK. Niblack binarization on document images: area efficient, low cost, and noise tolerant stochastic architecture. Int J Pattern Recognit Artif Intell. 2021;35(04):2154013. https://doi.org/10.1142/S0218001421540136.
2. Ypma TJ. Historical development of the Newton-Raphson method. SIAM Rev. 1995;37(4):531–51. https://doi.org/10.1137/1037125.
3. Toral SL, Quero JM, Franquelo LG. Stochastic pulse coded arithmetic. In: 2000 IEEE International Symposium on Circuits and Systems (ISCAS), 2000:1:599–6021 . https://doi.org/10.1109/ISCAS.2000.857166.
4. Mitra S, Banerjee D, Naskar MK. A low latency stochastic square root circuit. In: 2021 34th International Conference on VLSI

Design and 2021 20th International Conference on Embedded Systems (VLSID), 2021; p. 7–12 https://doi.org/10.1109/VLSID51830.2021.00006.
5. Gaines BR. In: Tou, JT. editors. Stochastic computing systems, Boston: Springer; 1969, p. 37–172. https://doi.org/10.1007/978-1-4899-5841-9_2.
6. Poppelbaum, WJ, Afuso C, Esch JW. Stochastic computing elements and systems. In: Proceedings of the November 14–16, 1967, Fall Joint Computer Conference. AFIPS '67 (Fall), ACM, New York, NY, USA, 1967; pp. 635–644. https://doi.org/10.1145/1465611.1465696.
7. Alaghi A, Hayes JP. Exploiting correlation in stochastic circuit design. In: 2013 IEEE 31st International Conference on computer design (ICCD), 2013; p. 39–46. https://doi.org/10.1109/ICCD.2013.6657023.
8. Ting P, Hayes JP. Stochastic logic realization of matrix operations. In: 2014 17th Euromicro Conference on digital system design, 2014; p. 356–364. https://doi.org/10.1109/DSD.2014.75.
9. Chen T, Hayes JP. Design of division circuits for stochastic computing. In: 2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2016; p. 116–121. https://doi.org/10.1109/ISVLSI.2016.48.
10. Parker KP, McCluskey EJ. Probabilistic treatment of general combinational networks. IEEE Trans Comput. 1975;C–24(6):668–70. https://doi.org/10.1109/T-C.1975.224279.
11. Ting P, Hayes JP. On the role of sequential circuits in stochastic computing. In: Proceedings of the on Great Lakes Symposium on VLSI 2017. GLSVLSI '17, ACM, New York, NY, USA, 2017; p. 475–478. https://doi.org/10.1145/3060403.3060453.
12. Alaghi A, Cheng Li, Hayes, J.P.: Stochastic circuits for real-time image-processing applications. In: 2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC), 2013; p. 1–6. https://doi.org/10.1145/2463209.2488901.
13. Alaghi A, Hayes JP. Survey of stochastic computing. ACM Trans Embed Comput Syst. 2013;12(2s):92–19219. https://doi.org/10.1145/2465787.2465794.
14. Ting P, Hayes JP. Eliminating a hidden error source in stochastic circuits. In: 2017 IEEE International Symposium on defect and fault tolerance in VLSI and nanotechnology systems (DFT), 2017; p. 1–6. https://doi.org/10.1109/DFT.2017.8244436.
15. Lee VT, Alaghi A, Hayes JP, Sathe V, Ceze L. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In: Design, Automation Test in Europe Conference Exhibition (DATE), 2017; p. 213–18 https://doi.org/10.23919/DATE.2017.7926951
16. Ambrosio V, Molica Bisci G, Repovš D. Nonlinear equations involving the square root of the Laplacian. Discrete Contin Dyn Syst S. 2019;12(2):151–70. https://doi.org/10.3934/dcdss.2019011.
17. Kumar V, Gupta P. Importance of statistical measures in digital image processing. Int J Emerg Technol Adv Eng. 2012;2(8):56-62.
18. Yang P, Song W, Zhao X, Zheng R, Qingge L. An improved Otsu threshold segmentation algorithm. Int J Comput Sci Eng. 2020;22(1):146–53.
19. Liu W, Hu E-W, Su B, Wang J. Using machine learning techniques for dsp software performance prediction at source code level. Connect Sci. 2021;33(1):26–41.
20. Chouder R, Benhamidouche N. New exact solutions to nonlinear diffusion equation that occurs in image processing. Int J Comput Sci Math. 2019;10(4):364–74.
21. Chen K, Wang G, Chen J, Yuan S, Wei G. Impact of climate changes on manufacturing: Hodrick-Prescott filtering and a partial least squares regression model. Int J Comput Sci Eng. 2020;22(2–3):211–20.
22. Liu A, Li P, Deng X, Ren L. A sigmoid attractiveness based improved firefly algorithm and its applications in iir filter design. Connect Sci. 2021;33(1):1–25.
23. Ren H, Hoang LB, Chen H-C, Wei BWY. Design of a 16-bit cmos divider/square-root circuit. In: Proceedings of 27th Asilomar Conference on Signals, Systems and Computers, 1993; p. 807–8111. https://doi.org/10.1109/ACSSC.1993.342633.
24. Kabuo H, Taniguchi T, Miyoshi A, Yamashita H, Urano M, Edamatsu H, Kuninobu S. Accurate rounding scheme for the Newton-Raphson method using redundant binary representation. IEEE Trans Comput. 1994;43(1):43–51.
25. Verma K. On the centroidal mean newton's method for simple and multiple roots of nonlinear equations. Int J Comput Sci Math. 2016;7(2):126–43.
26. Haridas SG, Ziavras SG. Fpga implementation of a Cholesky algorithm for a shared-memory multiprocessor architecture. Parallel Algorithms Appl. 2004;19(4):211–26.
27. Vázquez Á, Bruguera JD. Iterative algorithm and architecture for exponential, logarithm, powering, and root extraction. IEEE Trans Comput. 2012;62(9):1721–31.
28. Singh A. An efficient fifth-order iterative scheme for solving a system of nonlinear equations and pde. Int J Comput Sci Math. 2020;11(4):316–26.
29. Wu D, San Miguel J. In-stream stochastic division and square root via correlation. In: 2019 56th ACM/IEEE Design Automation Conference (DAC), IEEE, 2019; p. 1–6.
30. Li P, Lilja DJ. Using stochastic computing to implement digital image processing algorithms. In: Proceedings of the 2011 IEEE 29th International Conference on computer design. ICCD '11, IEEE Computer Society, USA, 2011; p. 154–161. https://doi.org/10.1109/ICCD.2011.6081391.
31. Xu W, Xie G, Wang S, Lin Z, Han J, Zhang Y. A stochastic computing architecture for local contrast and mean image thresholding algorithm. Int J Circ Theory Appl. 2022. https://doi.org/10.1002/cta.3320.
32. Lee VT, Alaghi A, Pamula R, Sathe VS, Ceze L, Oskin M. Architecture considerations for stochastic computing accelerators. IEEE Trans Comput Aided Des Integr Circ Syst. 2018;37(11):2277–89. https://doi.org/10.1109/TCAD.2018.2858338.
33. Baker TJ, Sun Y, Hayes JP. Benefits of stochastic computing in hearing aid filterbank design. In: 2021 IEEE Biomedical Circuits and Systems Conference (BioCAS), 2021; p. 1–5. https://doi.org/10.1109/BioCAS49922.2021.9645021.
34. He S, Schomaker L, Shi Z. Monk Cuper Set (MCS) for benchmarking historical document image binarization. https://doi.org/10.5281/zenodo.4767809.
35. Moghaddam RF. Objective evaluation of binarization methods for document images. 2013. https://in.mathworks.com/matlabcentral/fileexchange/27652-objective-evaluation-of-binarization-methods-for-document-images. Accessed 31 Mar 2022.