Spring 2017

Novel approaches for efficient stochastic computing

Ramu Seva

Department:

NOVEL APPROACHES FOR EFFICIENT STOCHASTIC COMPUTING

by

RAMU SEVA

A THESIS

Presented to the Graduate Faculty of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

2017

Approved by

Dr. Minsu Choi, Advisor
Dr. Daryl Beetner
Dr. Mihail Cutitaru

PUBLICATION DISSERTATION OPTION

This dissertation has been prepared in the form of two papers formatted to the specifications prescribed by Missouri University of Science and Technology:

Paper I, pages 1 to 23 are intended for submission to IEEE Transaction on Emerging Topics in Computing (IEEE TETC).

Paper II, pages 24 to 31 has been accepted to be published in 13th IEEE International SoC Design Conference (ISOCC).

ABSTRACT

This thesis is comprised of two papers, where the first paper presents a novel approach for parallel implementation of SC using FPGA (Field Programmable Gate Array). This paper makes use of the distributed memory elements of FPGAs (i.e., look-up-tables -LUTs) to achieve this. An attempt has been made to build the stochastic number generators (SNGs) by using the proposed LUT approach. The construction of these SNGs has been influenced by the Quasi-random number sequences, which provide the advantage of reducing the random fluctuations present in the pseudo-random number generators such as LFSR (Linear Feedback Shift Register) as well as the execution time by faster convergence. The results prove that the throughput of the system increases and the execution time is reduced by adopting the proposed technique.

The second paper of the thesis proposes a novel technique referred to as the approximate stochastic computing (ASC) approach focusing on image processing applications to reduce the lengthy computation time of SC with a trade-off in accuracy. The proposed technique is to truncate low-order bits of the image pixel values for SC for faster operation, which also causes an error in the binary to stochastic converted value. Attempts have been made to reduce this error by linearly increasing the clock cycles rather than exponentially. Experimental results from the well-known SC edge detection circuit indicate that the proposed technique is a promising approach for efficient approximate stochastic image processing.

ACKNOWLEDGMENTS

Foremost, I would like to express my sincere gratitude to my advisor Dr. Minsu Choi for the continuous support of my master's study and research, for his patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this thesis. I could not have imagined having a better advisor and mentor for my master's.

Besides my advisor, I would like to thank the rest of my thesis committee, Dr. Daryl Beetner and Dr. Mihail Cutitaru for their encouragement and insightful comments.

Last but not the least, I would like to thank my parents for supporting me throughout my life.

TABLE OF CONTENTS

Page

PUBLICATION DISSERTATION OPTION iii

ABSTRACT iv

ACKNOWLEDGMENTS v

LIST OF ILLUSTRATIONS viii

LIST OF TABLES ix

SECTION

1. INTRODUCTION 1

PAPER

I. FPQSC: FPGA BASED PARALLEL QUASI STOCHASTIC COMPUTING 2

ABSTRACT 2

1. INTRODUCTION 4

2. BACKGROUND 6

3. LUT-BASED METHOD 10

4. EFFICIENT IMPLEMENTATION FOR PARALLELISM 16

4.1. FIRST STAGE 17

4.2. SECOND STAGE 18

4.3. THIRD STAGE 20

5. SIMULATION RESULTS 22

5.1. ACCURACY OF THE PROPOSED SEQUENCE GENERATORS 22

5.2. EDGE DETECTION 24

6. CONCLUSION 27

II. APPROXIMATE STOCHASTIC COMPUTING (ASC) FOR IMAGE PROCESSING APPLICATIONS 28

ABSTRACT 28

1. INTRODUCTION 30

2. APPROXIMATE STOCHASTIC COMPUTING: EDGE DETECTION CASE STUDY 31

3. PROPOSED DESIGN 32

4. ADAPTIVE TRUNCATION FOR ERROR REDUCTION 34

4.1. PERFORMANCE VERIFICATION 35

5. CONCLUSION 38

SECTION

2. CONCLUSION 39

BIBLIOGRAPHY 40

VITA 43

LIST OF ILLUSTRATIONS

Figure Page

PAPER I

2.1 Basic circuits used in stochastic computation [1]. 7

2.2 a) Correlation effect in an AND gate (multiplier circuit in SC) when the both bit streams are the same; b) Correlation effect when both bit streams are inverse of each other. 8

3.1 Distribution of pseudo-random points (top) and LD points (bottom) in the unit square. 11

3.2 Basic block diagram of the proposed QSNG. 12

4.1 Parallel stochastic bit matrix processing. 17

4.2 Three stages of parallel implementation 18

4.3 First stage LD sequence generation. 19

4.4 Second stage - LD to stochastic bit conversion. 20

5.1 Edge detection using LFSR's 25

5.2 Edge detection using LD sequence genrators 26

PAPER II

3.1 (a) Proposed circuit for approximate stochastic edge detection; (b) Stochastic bit generation; (c) Edge detection circuit used [2]. 33

4.1 Edge detection implemented using 8-bit length pixel values 35

4.2 Edge detection implemented using 4-bit length pixel values 36

LIST OF TABLES

Table Page

PAPER I

3.1 Table showing the resource utilization 15

5.1 Minimum number of clock cycles needed for various input values in LD SNG and LFSR SNG. Average time saved is also shown. 23

5.2 Resource utilization comparison. 26

SECTION

1. INTRODUCTION

Deterministic computing has dominated the digital world for decades. Lately, new techniques are being adopted other than conventional deterministic approach such as SC (stochastic computing) technique. It is probabilistic in nature and has the advantages of lower power and area overhead. It is being implemented widely in image processing applications as well as in designing simple arithmetic circuits, but the main challenges it faces are longer computation time and lower accuracy. In this thesis, these two main challenges of the SC are studied and new approaches have been presented to address them.

The first part of the dissertation deals with the construction of new SNG (stochastic number generator) called as Quasi-SNG (QSNG) which tries to address the issue of accurate stochastic computing and proposes a possible parallel implementation using it to increase the throughput of the system as compared to the implementation using traditional SNG built using LFSRs. The design has been implemented using Xilinx Virtex 4 FPGAs.

The second part of the dissertation talks about minimizing the computation time of a SC in an image processing application, the edge detection by approximating a 8-bit length pixel value to a 4-bit length and converting it to stochastic numbers. This approach though introduces a certain amount of a known error attempts have been made to reduce this error to an acceptable limit by increasing the clock cycle linearly rather than exponentially. The acceptable quality of an image has been analyzed in terms of PSNR (peak signal-t-noise ratio) and MSE (mean square error) values.

PAPER

I. FPQSC: FPGA BASED PARALLEL QUASI STOCHASTIC COMPUTING

Ramu Seva[1] Prashanthi Metku[1], Kyung Ki Kim[2], Yong-Bin Kim[3] and Minsu Choi[1]

[1]*Dept of ECE, Missouri Univ of Science & Technology, Rolla, MO, USA,*

{pmcmc,rs2k6,choim}@mst.edu

[2]*Dept of Electronic Eng., Daegu University, Gyeongsan, Korea, kkkim@daegu.ac.kr*

[3]*Dept of ECE, Northeastern University, Boston, MA, USA, ybk@ece.neu.edu*

ABSTRACT[1]

High performance of FPGAs in image processing applications is justified by their flexible reconfigurability, inherent parallel nature and availability of large amount of internal memories. Lately, SC (stochastic computing) has been found to be significantly advantageous in image processing applications because of lower hardware complexity and power consumption. However, its viability is deemed to be limited due to excessive run-time requirement. In this paper, a novel method is introduced

[1]To be submitted to IEEE Transaction on Emerging Topics in Computing

where efficient parallel implementation of SC is accomplished using FPGA to address this issue. The proposed approach is to leverage the distributed memory elements of FPGAs (i.e., look-up-tables - LUTs) to achieve this. An attempt has been made to build the stochastic number generators (SNGs) using LUTs. The construction of these SNGs has been influenced by Quasi-random number sequences, which provide the advantage of reducing the random fluctuations present in the pseudo-random number generators such as LFSRs (Linear Feedback Shift Registers) as well the execution time via significantly faster convergence. The proposed design has been implemented on Virtex-4 FPGA and results have been compared with the parallel implementation of conventional stochastic computation for edge detection application. Results prove that by using this approach the throughput of the system increases and the execution time is reduced significantly.

1. INTRODUCTION

In general, many image processing applications deal with data words of less than 16-bits. Compared to application specific integrated circuits (ASICs), FPGAs can provide higher performance with lesser clock speed. This high performance of FPGAs is also supported by availability of large amounts of internal memory blocks, which provide parallel access to large data sets. Graphics processing units (GPUs) are another hardware platform where high performance can be achieved at higher clock rates, but they have been limited to fewer number of applications as they are designed for a specific set of operations and realizing them for various applications is difficult.

Stochastic computing (SC) is an alternative computing style which has recently proved to be advantageous in image processing applications, because of its potential area and power benefits compared to binary implementations. The performance benefits of parallel implementation of a stochastic circuit using FPGAs for an image processing application has not been analyzed in the prior literature until now. Taking advantage of parallel implementation of stochastic circuits (such as edge detection and multiplication) is possible by using the distributed memory elements of FPGAs. New SNGs are designed to utilize quasi-random numbers, which makes use of the distributed memory elements in this paper. Though the design alternative selected here is certainly not new [3], no prior work in SC has analyzed it in the FPGA context. While it is possible to use linear feedback shift registers (LFSRs) as random number generators in SNGs, making use of low-discrepancy sequences (LDS) or quasi-random numbers is advantageous, because they do not suffer from random

fluctuations and converge faster. This paper mainly focuses on the possibility of parallel stochastic computation for image processing applications using FPGAs. The main contributions of this paper are as follows:

1. Reduction of the random fluctuation errors present in the traditional pseudo random numbers by adopting a new way of constructing SNGs by using look-up table (LUT) based approach and low-discrepancy (LD) subrandom sequences.

2. Parallel implementation is accomplished to increase the throughput and decrease the execution time of SC.

2. BACKGROUND

SC has its roots in 1960's and it is used for probability representation using digital bit streams [4, 5]. SC has been successfully applied to many applications like image processing, neural networks, LDPC codes, factor graphs, fault-tree analysis, and in filters [2, 6, 7, 8, 9, 10]. Extensive use of stochastic computation is still limited, because of it's long run-time and inaccuracy. Recent improvements have mainly focused on improving the accuracy and performance of the stochastic circuits by sharing consecutive bit streams, sharing the stochastic number generators, exploiting the correlation, and using the spectral transform approach for stochastic circuit synthesis [11, 12, 13, 14]. This paper also explores new methods to improve the accuracy and performance of stochastic circuits.

Fig. 2.1 shows the basic SC circuits. The function implemented by these circuits varies with the number interpretation i.e., unipolar, bipolar or inverted bipolar (UP, BP, IBP) where unipolar format is used to represent real number x in the range of $[0, 1]$, bipolar is used to represent real number x between $[-1, 1]$, and IBP is inverted bipolar format which is an invert of BP ranging from $[-1, 1]$, where the boolean values 0 and 1 in the stochastic number (SN) represent 1 and -1 respectively rather than -1 and 1 in the case of BP format. One can refer to [13] for more details on different SN formats. In SC, a probability value is represented by a binary bit stream of $0s$ and $1s$ with specific length L. To represent a probability value of 0.5, the half of the bits in the bit stream of length L are represented by $1s$. For example, if 0.5 is to be represented by a bit stream of 10 bits, then 0101010101 bit stream can be used. This is just one way of representing it and the representation of a probability

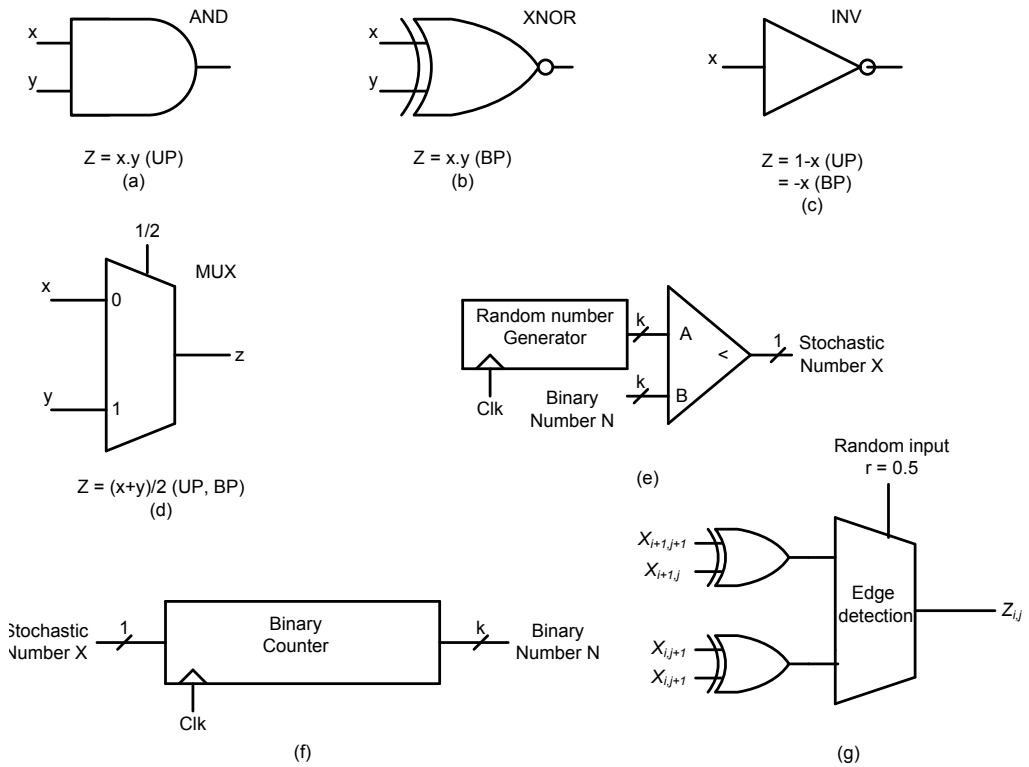


Figure 2.1. Basic circuits used in stochastic computation [1].

value in SC is not unique and not all real number's in the interval $[0, 1]$ can be exactly represented for a fixed value of L. Another considerably important factor when representing a stochastic number is the dependency or correlation between the inputs [15]. This is an important inherent nature of stochastic circuits which limits its performance over certain applications when compared to conventional binary implementations [16]. Fig.2.2 shows two examples where inaccurate results are caused by correlated inputs in the multiplication circuit. This correlation comes from the SNGs, where the SNs generated by SNGs happened to have the same set of sequences of $1s$ and $0s$ or with some relation among them as shown in Fig. 2.2. This causes inaccuracy in the output generated, so SNGs are always chosen in such a way they

produce uncorrelated SNs. LFSRs are known to be best-suited for SC and have been used for number generation in many SC designs [17]. However, the main disadvantages are the number of SNGs must be higher (i.e., for every independent input, the number of SNGs used increases by one) for uncorrelated inputs and longer time to operate for accurate and efficient SC [15]. When the circuit size, power, and computation time of

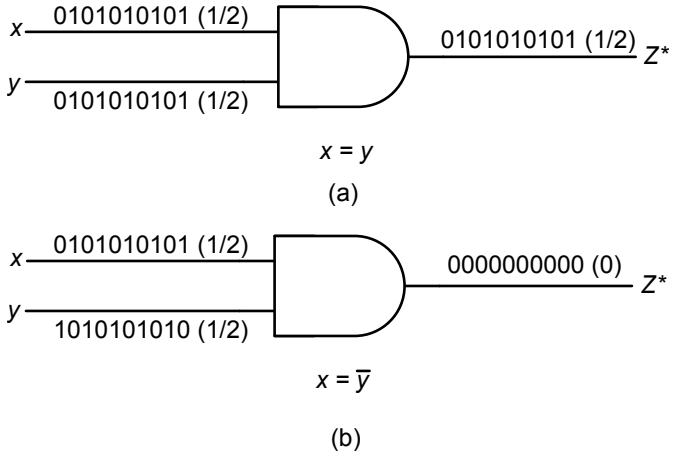


Figure 2.2. a) Correlation effect in an AND gate (multiplier circuit in SC) when the both bit streams are the same; b) Correlation effect when both bit streams are inverse of each other.

SC are considered, the main contributions for these factors to vary significantly are the SNGs. The number of SNGs is proportional to the area of a stochastic circuit, contributing to about 80% of the circuit area. The power consumed by SC mainly depends on the number of clock cycles the circuit uses for computation which, in turn, depends on the SNG properties. The computation time can be limited by SNGs due to their inherent properties such as random number fluctuations. The computation time increases exponentially with the linear increase in accuracy. Hence the need to address the basic questions such as: — What is the minimum number of clock cycles needed to run, so the probability value is represented correctly? What is the effect of random noise fluctuations in a sequence of stochastic operations? — Answers to

these questions may help in decreasing the computation time drastically. SC has another disadvantage over the binary implementation as all the operations in the SC are single staged; therefore, conventional techniques such as pipelining to improve the throughput cannot be applied [1].

This paper is organized as follows. Section 3 gives the background of LD sequences and LUT-based method implemented. Section 4 discusses the parallel implementation of SNGs and the different stages used in the parallel implementation of SNGs. Section 5 discusses the simulation results comparing the proposed SNG with the pseudo-random number generators (LFSRs). Analysis of the convergence rate of the proposed SNG with that of the LFSR. Discussion of the application of the proposed parallel SNG in edge detection and multiplication circuit and specification of the advantages over the LFSR implementation. Finally, Section 6 asserts the conclusion.

3. LUT-BASED METHOD

In the proposed approach, SNGs are designed to leverage LUTs, which are the distributed memory elements of the FPGAs. FPGAs are the target hardware implemented in this design for their abundant availability of LUTs and their inherent parallel nature. One important point to be noted here, since the target devices chosen are FPGAs for the implementation of a digital circuit, reduction in area is not the main concern. The primary focuses in this paper are improving the accuracy, reducing the number of random fluctuations, and reducing the execution time by parallel implementation using the proposed LUT-based Quasi-SNG (QSNG).

LUT is defined as a group of logic gates hard-wired on the FPGA. LUTs store a defined list of outputs for every set of inputs and provide a fast way to obtain the output of a logic operation. LUTs are single memory elements, where the inputs are the addresses, and the corresponding outputs are the data stored in the given addresses. Among various applications, LUTs are used in digital signal processing algorithms, where multiplication is done with a fixed set of coefficients which are already pre-computed and stored in the LUT so they can be used without computing them each time. The same concept is used in this paper, computing fixed direction vectors which have to be multiplied with a binary number to get the desired sequence. LUT-based method is used to develop stochastic bit numbers by using Quasi-Monte Carlo (QMC) methods. The LD sequences in the literature are used to develop these stochastic numbers [18]. The main advantage gained over use of LFSRs, is that LD sequences do not suffer from random fluctuations as the $0s$ and $1s$ are uniformly spaced [3]. This is unlike LFSRs, where the $0s$ and $1s$ are non-uniformly spaced. The

idea behind the LD sequences is to let the fraction of the points within any subset of $[0, 1]$ be as close as possible, such that the low-discrepancy points will spread over $[0, 1]$ as uniformly as possible, reducing gaps and clustering points. Fig. 3.1 shows the comparison of pseudo-random points and LD points in the unit square. LD points shows even and uniform coverage of the area of interest as shown and are to converge faster when applied to SC. The widely used sequences which fall under LD sequence

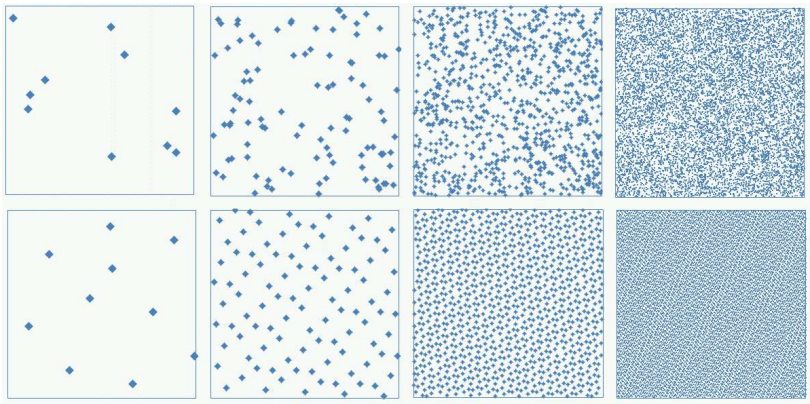


Figure 3.1. Distribution of pseudo-random points (top) and LD points (bottom) in the unit square.

category are the Halton sequence, Sobol sequence, Faure sequence, and Niederreiter sequence [18]. Generating these sequences is usually software based because hardware implementation of these sequences is not suited for SC due to their complexity in construction [3]. This disadvantage of LD sequences is mitigated fully by the proposed LUT-based approach. The main difference in generating the LD sequences lies in the construction of their direction vectors [18]. Each sequence has a specific type of algorithm to compute these and the uniformity of the sequence depends on the way these direction vectors are computed. In this paper, LUT-based SNGs were designed using three LD sequences including Halton, Sobol, and Niederreiter. The digital method was chosen to design these sequences, restricting the base value to binary

base 2. For detailed explanation about the sequences mentioned above, refer to [18]. General structure used for generation of LD sequence using binary base 2 is as shown in Fig. 3.2. It contains RAM to store the direction vectors, a multiplication circuit and bit-wise XOR gates. In the multiplication circuit, every bit from the counter

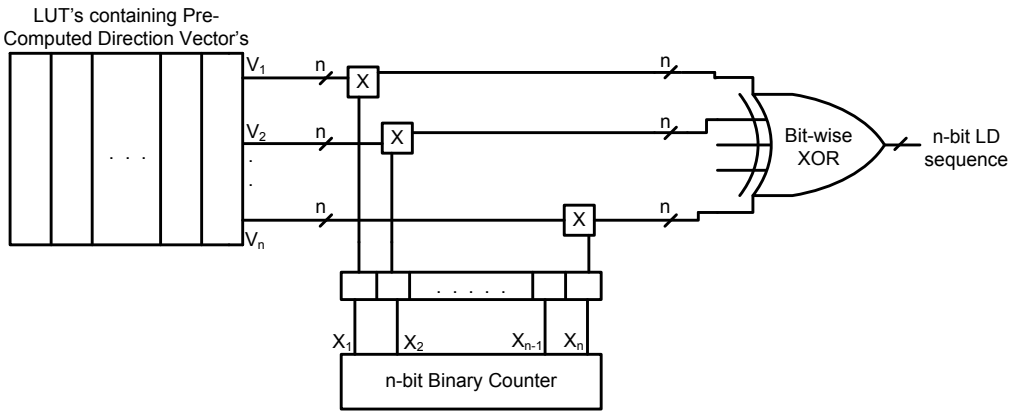


Figure 3.2. Basic block diagram of the proposed QSNG.

output is multiplied by each n-bit direction vector, stored in the RAM, to generate n-bit intermediate direction vectors. These n-bit intermediate direction vectors are then bit-wise XORed (i.e., modulo-2 addition) to generate a n-bit LD sequence in the unit interval $[0, 1]$.

This can be expressed by using a mathematical expression as shown in the equation below:

$$N = x_1(n-1)V_1 \oplus x_2(n-1)V_2 \oplus .... \tag{3.1}$$

where $\oplus$ denotes binary addition or XOR operation, $x_1(n-1)x_2(n-1)...$ is the binary representation of $(N-1)$, $V_1, V_2, ...V_n$ represents the direction vectors and N represents the Nth number in the respective LD sequence; for example $N = 8$ represents the $8th$ number in a Sobol sequence, which is computed by using n direction vectors and a n bit counter, when Sobol sequence direction vectors are used [19].

Sobol and Niederreiter sequences belong to the general class of digital sequences and their LD sequence generation can be expressed by the above digital method. The Halton sequences belong to the simplest form of LD sequences and their construction does not have a general form as mentioned above in Equation 3.1. In the above Equation 3.1, $V_1, V_2, ...V_n$ are called the direction vectors and are defined as the constant values which have to be multiplied with the counter output to generate the desired LD sequence as shown in Fig. 3.2. These values do not change throughout the operation of the circuit. Hence, for the generation of Halton LD sequences, defining the direction vectors to fit into the above equation of the general digital method of LD sequence is necessary to generalize the hardware structure for all of the LD sequences.

Halton sequences are defined as the generalized form of Van der Corput sequences which use a distinct prime base for every dimension. The k_{th} Halton point $H(k)$ is defined as $H(k) = \sum_{i=0}^{\infty} a_i(k-1)b^{-i-1}$ [20]. Upon closer inspection of the summation we define $a_i(k-1)$ is nothing but the base b representation of $k-1$ and b^{-i-1} is the base b term which has to be multiplied with $a_i(k-1)$ for generation of each sequence depending on the value of k. The term b^{-i-1} is a constant term, the value does not change with the change in the value of k. Hence, these terms are defined as direction vectors and fit into the general form represented above to generate a LD sequence by choosing the base $b = 2$.

Sobol and Niederreiter sequences have specific algorithms to calculate the direction vectors which fit into the equation above to generate the LD sequence. In this paper, algorithms reported in papers [21] and [18] are used to pre-calculate direction vectors. An important point to note in this implementation is that the number of sequences generated is limited by using only R base b direction vectors of R digits

which are capable of representing a value of $b^R - 1$ in base b using R digits [18] by depending on the bit length requirement L. For example, to generate a stochastic bit length of 256, generation of only initial 256 LD sequences is required. For this process, 8-bit length direction vectors, which are capable of generating an 8-bit length LD sequence every clock cycle are needed. The maximum value they can represent is $b^8 - 1 = 255$ limiting the size of the counter. For the above 256 initial sequences a 8-bit counter is needed to count from 0 to 255.

After generation, the LD sequence numbers are sent to the comparator where they are compared with the input value to generate an equivalent stochastic number. The size of the proposed SNG depends on the stochastic bit length L of the circuit as well as the number of inputs to the stochastic circuit. For a stochastic bit length of 256 it is necessary to use an 8 bit binary counter and a memory space of 64 bits to store 8 direction vectors each of 8 bit length. Independent stochastic inputs require different direction vectors; as the number of independent stochastic inputs increases the memory space required to store these direction vectors increases. LUT-based SNGs were implemented for 256, 512, 1024 and 2048 bit lengths on the Xilinx Virtex 4-SFFPGA (XC4VLX15) device and synthesized using Xilinx ISE tool. The resource utilization, speed, and throughput are shown in Table 3.1. Note the resource utilization shown in the Table 3.1 includes the SNG as well as the stochastic-to-binary (STB) conversion unit. In this paper, a general form of implementation was presented and further optimization of the circuit has been left for the future study. This table clearly shows the LD sequence generators make use of more hardware when compared to the LFSRs, but the convergence and the accuracy obtained from LD sequences are superior enough to justify this extra hardware utilization (explained in the following section).

Table 3.1. Table showing the resource utilization

Sequence	**Bit-stream length**	**Slices**	**Frequency (MHz)**	**Throughput (Gbits/s)**
LD Sequence	256	30	299.8	0.3
	512	31	298.9	0.3
	1024	33	297.94	0.3
	2048	35	297.90	0.3
Pseudo-Random	256	9	1134.2	1.13
	512	10	1133.46	1.12
	1024	11	1133.46	1.12
	2048	12	1133.46	1.12

The frequency mentioned in the table corresponds to the maximum frequency and the average computation time of a stochastic computation can be calculated simply by dividing the number of clock cycles needed for a successful stochastic computation by the maximum frequency value.

4. EFFICIENT IMPLEMENTATION FOR PARALLELISM

The proposed parallel implementation of the SNGs was designed to generate LD sequence numbers in parallel. These LD sequence numbers generated in parallel were used to generate stochastic bits in parallel. These stochastic numbers, generated parallelly, are termed as stochastic bit vectors (SBVs) and the parallel processing used to generate the sequence is termed as stochastic bit matrix (SBM) processing. Consider a 256 bit length stochastic bit matrix, this design generates p initial bits every clock cycle of the SBM, instead of generating one bit of the SBM. This is shown in a vector form in Fig. 4.1 which shows that for one stochastic bit generation using a single SBM processing unit (SBMPU) 256 clock cycles is needed to generate a 256 bit length SBM. By duplicating p SBMPUs in parallel, it is also possible to generate p stochastic bits of the SBM in just one clock cycle. Hence, $256/p$ clock cycles are needed for generating 256 bit length, thus saving execution time of the operation as p increases. The structure of the parallel implementation of the circuit is shown in Fig. 4.2. The parallel implementation of the proposed SNGs is done in three stages. First stage is where the LD sequence numbers are generated in parallel. The second stage is where the stochastic bit streams are generated parallelly using comparators. Finally, the third stage is where the stochastic bits are converted back to binary number by counting the number of ones. A combinational circuit is implemented for the conversion of stochastic to binary number by counting the number of $1s$ in the SN by making use of Hamming weight counter principle, which is capable of counting only the number of $1s$ in a bit stream [22].

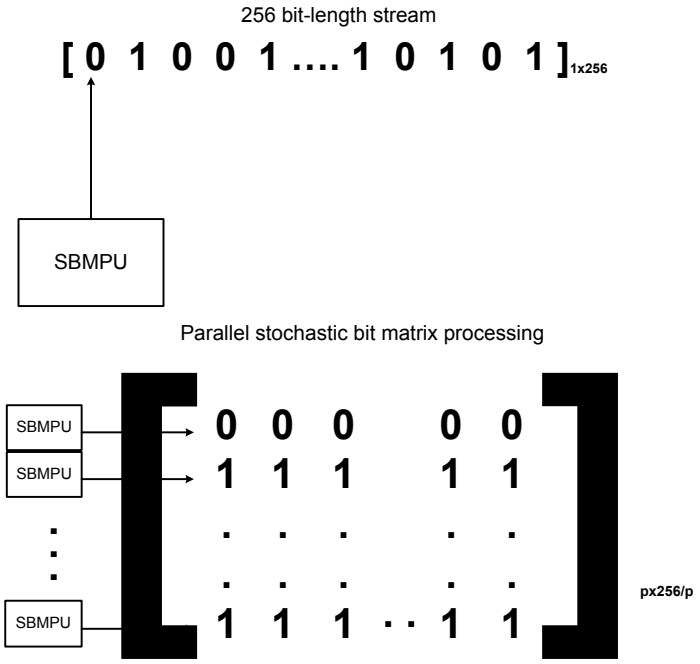


Figure 4.1. Parallel stochastic bit matrix processing.

4.1. FIRST STAGE

The first initial p LD sequence numbers are generated parallelly depending on the degree of parallelism. The general structure of the implementation is as shown in Fig. 4.3. Here the entire structure is not duplicated, but the part of the SNG which generates the LD sequence number is duplicated to significantly reduce area overhead. The degree of parallelism determines the amount of hardware utilized. Counters, which follow a specific sequence of counting, are used to implement the SNGs in parallel. For example, to generate the first initial 8 sub-sequences in parallel of a 256 bit stream length, use eight 8-bit counters which count by 8. The first counter follows the sequence $0, 8, 16, 32...$ and the second counter follows the sequence $1, 9, 17, 33...$ in the same way the 8^{th} counter follows the sequence $7, 15, 31....$ Therefore, in the

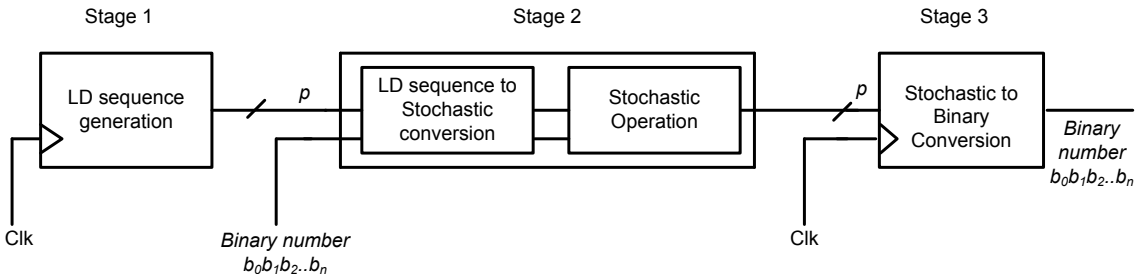


Figure 4.2. Three stages of parallel implementation

first clock cycle the eight counters hold the value from 0 to 7, which means that the first eight LD sequence numbers are generated in parallel. In the second clock cycle, the counters are incremented by 1 to hold the value 8 to 15 and the next 8 LD sequence numbers are generated. These generated sequences are then sent to the parallel comparator units where they are compared with the input probability value to generate the stochastic bits in parallel. This implementation generates a sequence for a single input parallelly. For multiple inputs, different direction vectors can be used while the circuit for the generation of the LD sequence is the same.

4.2. SECOND STAGE

The second stage consists of generation of the stochastic bit stream and the stochastic operation. For generation of the stochastic bit stream, the LD sequences generated parallelly are sent to the comparators, which are also in parallel, such that multiple sequences are compared simultaneously to generate a stochastic bit stream parallelly. For example, to generate the first initial 8-bits of the stochastic bit stream, use 8 comparators where each sequence is compared with the binary probability value to generate the first 8 bits of the SN at the same time. This is termed as 8-bit SBV

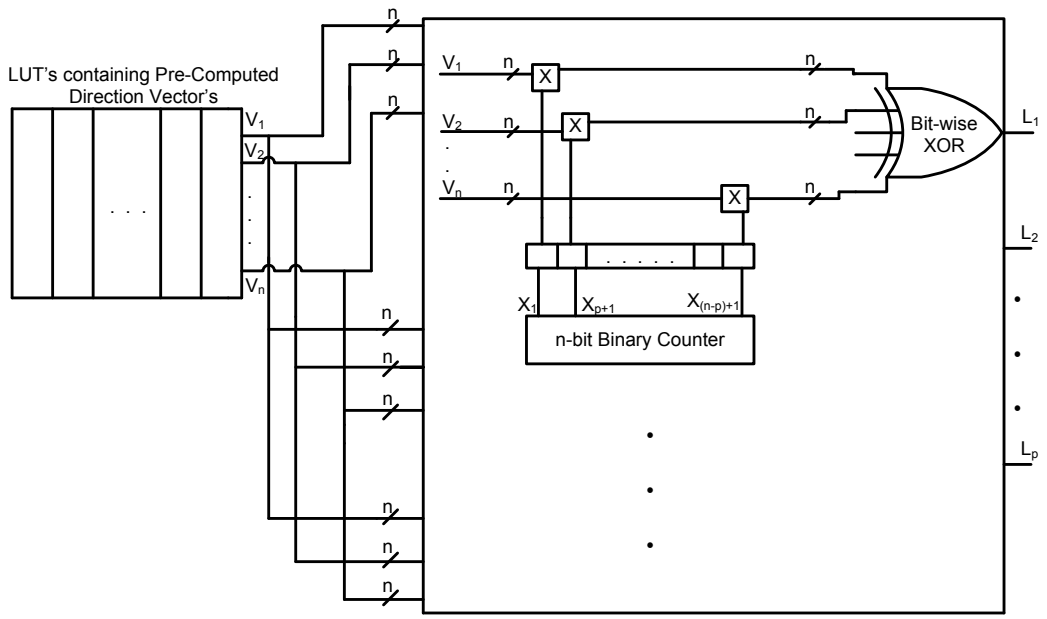


Figure 4.3. First stage LD sequence generation.

generation using a 8 SBM processing in one clock cycle by replicating the SBM circuit 8 times. Similarly, in order to generate 16 SBV's, 16 SBM processing is done in one clock cycle replicating the SBM circuit 16 times. SBM processing circuit involves only that part of the LD sequence generator capable of generating the sequence (i.e., the multiplication and the bit-wise XOR structure) and a LD sequence to stochastic conversion unit (i.e., comparator), LUTs used are shared among the parallel SBM processing units as they are constant values which do not change during the execution cycle. The generated stochastic bits by SBM processing are then sent for computation and then to the stochastic to binary conversion stage for final output. See Fig. 4.4 for the parallel structure of the stochastic bit stream generators. The number of comparators used depends on the degree of parallelism implemented. Hence, the degree of parallelism determines the hardware utilized.

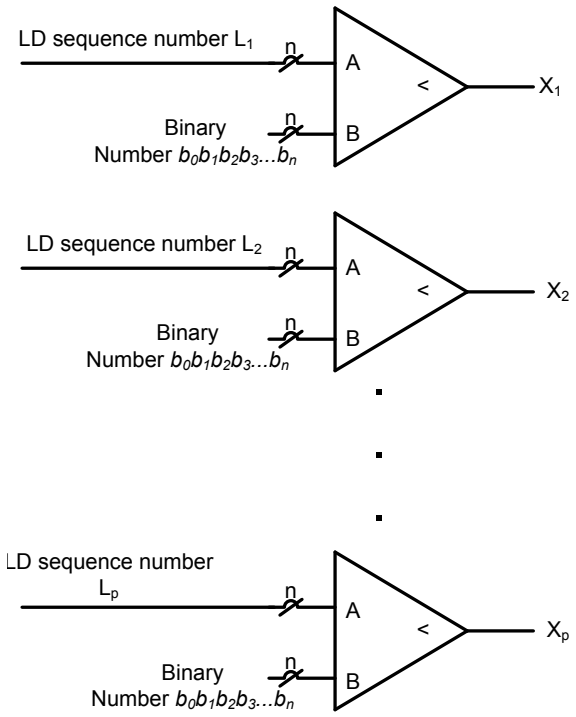


Figure 4.4. Second stage - LD to stochastic bit conversion.

4.3. THIRD STAGE

The final stage in a stochastic computation is to create the binary output, which is generated by using STB conversion units which comprise of a counter which counts the number of 1s in the stochastic bit-stream. If the output stochastic bits generated are 8-bits per clock cycle, it is necessary to count the number of 1s in the initial 8-bits within one clock cycle; this is not possible by using a single counter circuit with the same clock period. In this paper, a STB conversion unit is used which converts the parallel stochastic output into binary number by using simple adder circuits. This circuit is capable of counting the number of 1s in a parallel bit stream. Hamming weight counter principle is used in this work [22]. The structure of

the STB conversion unit for counting the number of 1s in first initial 8-bits consists of four half adders, two 2-bit adders, a 3 bit adder, a 4-bit register, and a 4-bit adder cascaded. A 4 bit register is used to store the previous count value and it is updated every clock cycle with the new value (i.e., the number of 1s in the stochastic bit stream). To count the number of 1s in 16 initial stochastic bits of a 256 bit stream length, eight half adders, four 2-bit adders, two 3-bit adders, an 8-bit register, and an 8-bit adder are required. It should be noted that the unused bits in the 8 bit-adder and the register are assigned zero value initially. Therefore, the size of STB conversion unit increases with the number of parallel bits generated. Scalability issue of the STB conversion unit may not be a major concern as the proposed approach mainly targets image processing applications where the word length for many operations is less than 16-bit. The next section presents the simulation results of both the parallel and the serial implementation of the LUT-based LD sequence SNGs.

5. SIMULATION RESULTS

Simulation results are organized into two groups. First, the accuracy of sequence generator proposed in this work is compared with that of the pseudo-random number generator. This demonstrates that the proposed SNG generators have better convergence when compared with LFSRs. Second, the LUT-based SNGs are used in image processing and arithmetic application of SC (i.e., edge detection and multiplication) to compare the results with the LFSR-based SNGs. The parallel implementation of LFSRs was done by extracting all the pseudo-random sequences and saving them in the RAM, then calling the initial sub-sequence every clock cycle, depending on the degree of parallelism. The main disadvantage of this implementation is saving all the pseudo-random sequences in RAM as the initial sub-sequence of a particular pseudo-random sequence cannot be generated in parallel using multiple LFSRs because of it's construction. Simulation results that will be presented in next sections show that LUT-based LD sequence generators give better results in terms of throughput and convergence and computation time.

5.1. ACCURACY OF THE PROPOSED SEQUENCE GENERATORS

Table 5.1 shows the convergence and amount of time possibly saved in computation by using the-LUT based SNGs. Simulations were carried out using LUT-based SNGs and LFSRs for comparison. Different bit stream lengths of 256, 512, 1024 and 2048 were generated and the convergence of the probability values in the order of 10^{-3} was extensively studied. Comparing with pseudo-random number generators (LFSRs), LD sequence SNGs outperformed them over the range of probability values

Table 5.1. Minimum number of clock cycles needed for various input values in LD SNG and LFSR SNG. Average time saved is also shown.

Value	Min # of clock cycles to converge			
	LD-Sequence	**LFSR**	**Average time saved**	**Stochastic bit-length**
0.5	16	64	25%	2048
0.125	64	256	25%	2048
0.015625	128	384	33.33%	2048
0.0078125	192	512	37.5%	2048
0.00390625	256	1024	25%	2048
0.001953125	512	1536	33.33%	2048
0.0009765625	1024	2048	50%	2048

in $[0, 1]$ space. They converge much faster than the LFSRs, which results in significant reduction in computation time. An important conclusion of the convergence results is, if a bit stream length to be generated is in the order of $\frac{1}{2^n}$, the minimum number of clock cycles needed to run is equal to 2^n using LD sequence SNGs. The stochastic bit length used for this analysis is 2048. The average time saved is calculated by giving same input values multiple times and by finding the average value of the required number of clock cycles used to represent each input value for both the LFSR and LD-sequence implementation.

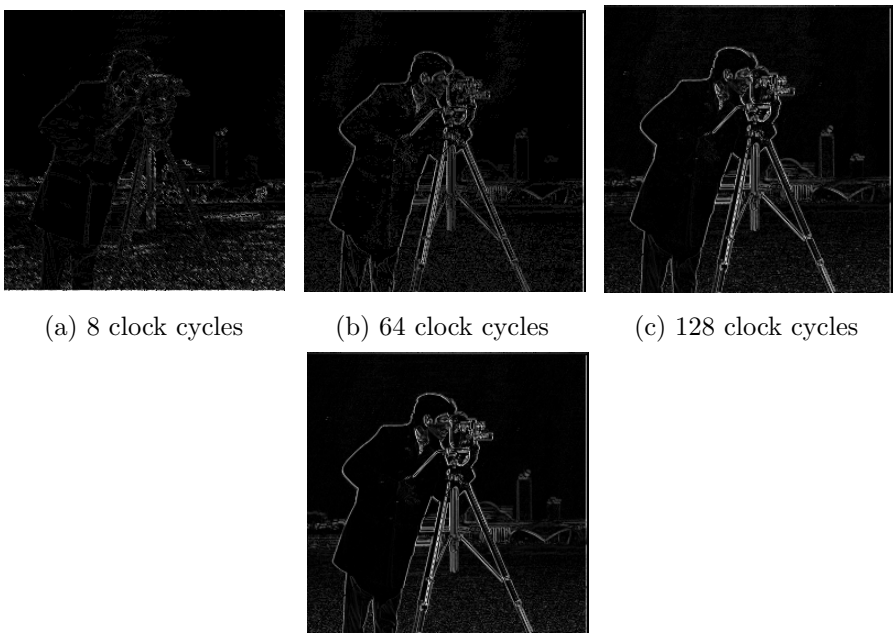
Results show that the average time saved is roughly around 25% for input values of 0.5 and 0.125. As the probability value decreases, the number of clock cycles increases exponentially, saving up to 50% of the computation time when compared to the LFSRs. Note, all these LUT-based SNGs were implemented using base-2 direction vectors. For independent SN generation, independent direction vectors are used to generate independent outputs to eliminate the inaccuracy caused by correlation. Another important thing to note about the LD sequence generators is that they never

deviate away from the input value. In other words, there is always a deterministic error bound between the actual value and the generated value whereas in the case of LFSRs the error bounds cannot be determined because of the random behavior and the inherent random noise fluctuations they possess.

5.2. EDGE DETECTION

The proposed SNGs were tested with edge detection as well as multiplication circuits for parallel implementations. In this work, edge detection circuit has been implemented using the stochastic circuit described in [2] as shown in Fig. 2.1(g). The simulation results of the edge detection circuit show that for initial sub-sequence of just 8 clock cycles, almost all the edges in case of LD sequences are clearly seen when compared to 64 clock cycles of the LFSR implementation as shown in Fig. 5.1 and Fig. 5.2. The run time of 256 clock cycles using LFSR implementation is equivalent to 64 clock cycles of LD sequence implementation. This shows the execution time can be reduced by almost 4 times using LD sequences.

The hardware utilization of parallel implementation of edge detection and multiplication circuits is shown in the Table 5.2. It is clear that since the convergence power of LD sequence generators is better than the LFSRs for the same circuit implementation LD sequences generators hardware utilization can be reduced by restricting it to generation of initial sub-sequences rather than complete sets of sequences. Therefore by using the proposed SNGs, better convergence will result. Also, higher throughput can be achieved as needed by implementing them in parallel.

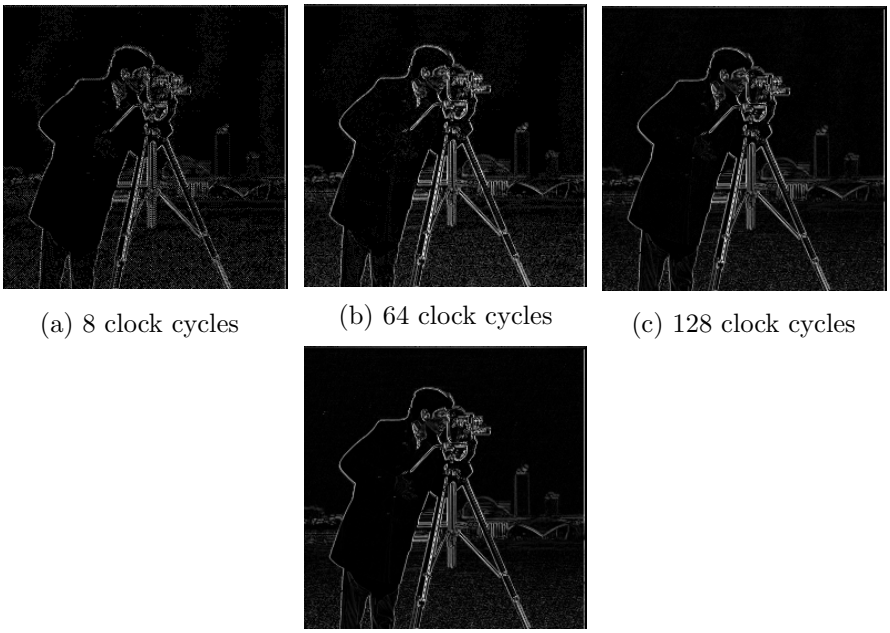


(a) 8 clock cycles (b) 64 clock cycles (c) 128 clock cycles

(d) 256 clock cycles

Figure 5.1. Edge detection using LFSR's

For a multiplication circuit simulations have been performed for a random set of 256 values to obtain an average value of computation time using LFSRs and LD seqeunce generators. The average computation time using LFSRs was around 512 clock cycles per input and the LD sequence generators was around 54 clock cycles per input. Similarly for the edge detection circuit it was 128 and 22 clock cycles using LFSRs and LD sequence generators.



(a) 8 clock cycles (b) 64 clock cycles (c) 128 clock cycles

(d) 256 clock cycles

Figure 5.2. Edge detection using LD sequence genrators

Table 5.2. Resource utilization comparison.

Sequence	**Application**	**Degree of Parallelism**	**Slices**
LD Sequence	Edge-Detection	4	537
		8	1069
		16	2125
LD Sequence	Multiplication	4	104
		8	199
		16	394
LFSR	Edge-Detection	4	190
		8	382
		16	757
LFSR	Multiplication	4	67
		8	134
		16	262

6. CONCLUSION

This paper has introduced a novel construction method to realize QSNGs on FPGA using LUTs. FPGA's superior reconfigurability was leveraged advantageously for parallel implementation of a stochastic circuit which outperforms the conventional LFSR-based stochastic circuit approach in terms of convergence and accuracy. Simulation results suggest that upto 50% of the computation time can be saved when dealing with the probability values less than 10^{-3}. Further, extensive simulation results justify that for faster and more accurate computation of image processing application making use of FPGA based parallel quasi-stochastic computing is a better option. The future scope of this work is to optimize the LD-sequence generator circuit with a combinational logic to generate the LD sequence to reduce the area occupied.

II. APPROXIMATE STOCHASTIC COMPUTING (ASC) FOR IMAGE PROCESSING APPLICATIONS

Ramu Seva[1] Prashanthi Metku[1], Kyung Ki Kim[2], Yong-Bin Kim[3] and Minsu Choi[1]

[1]*Dept of ECE, Missouri Univ of Science & Technology, Rolla, MO, USA,*

{pmcmc,rs2k6,choim}@mst.edu

[2]*Dept of Electronic Eng., Daegu University, Gyeongsan, Korea, kkkim@daegu.ac.kr*

[3]*Dept of ECE, Northeastern University, Boston, MA, USA, ybk@ece.neu.edu*

ABSTRACT[1]

SC (stochastic computation) has been found to be advantageous in image processing applications because of its lower area consumption and low-power operation. However, one of the major issues with the SC is its long run-time requirement for accurate results. In this paper, a new technique called the approximate stochastic computing (ASC) approach focusing on image processing applications is proposed to reduce the computation time of a SC with an acceptable trade-off in accuracy. The proposed technique is to truncate low-order bits of the image pixel values for SC for faster operation, which introduce an error in the binary to stochastic converted value. Attempts have been made to reduce this error by linearly increasing the clock cycles rather than exponentially. Experimental results from the well-known SC edge detection circuit

[1]Submitted to 13th IEEE International SoC Design Conference (2016)

indicate that this technique is a promising approach for efficient approximate image processing.

1. INTRODUCTION

SC has its roots from 1960's and it is used for probability representation using digital bit streams [4, 5]. SC has been successfully applied to many applications such as image processing, neural networks, LDPC (Low Density Parity Code) decoders, and factor graphs [15]. However, the extensive use of stochastic computation is deemed to be limited, because of it's long run-time requirement and inherent inaccuracy. Recent improvements have mainly focused on improving the accuracy and performance of the stochastic circuits by sharing consecutive bit streams, sharing the stochastic number generators, exploiting the correlation, and using the spectral transform approach for stochastic circuit synthesis [11, 12, 13, 14].

In this paper, a new approach called Approximate Stochastic Computing (ASC) to decrease the computation time has been proposed and analyzed. The proposed ASC is motivated by certain applications, such as audio, video and image processing, where an approximate or less-than-optimal solution is acceptable in lieu of smaller hardware circuit and faster operation. The proposed ASC technique has been validated by a specific image processing application called edge detection in this paper, although it can be used for various other applications, where a small amount of approximation is acceptable.

This paper is organized as follows. Section 2 gives the background of the proposed design approach. Then, section 3 discusses about new design and the image processing application implemented in this paper. Finally, Section 5 makes the conclusion.

2. APPROXIMATE STOCHASTIC COMPUTING: EDGE DETECTION CASE STUDY

Image processing belongs to the class of applications which demonstrate inherent error resilience, where approximate computing techniques can be used to design efficient digital systems [23]. Approximate computing (AC) is different from SC in a way that it does not involve assumptions and circuits involve deterministic designs rather than probabilistic designs implemented in SC. AC uses statistical properties of data and algorithms to trade quality for energy reduction and/or faster operation [24].

As a case study, both AC and SC approaches are combined to build an ASC edge detection circuit which provides area/speed efficient design with an acceptable error bound. As an input to the proposed ASC edge detection circuit, a grayscale bitmap image has been used where each pixel value is represented in 8 bit binary number. Then, an approximation of pixel value is initially done where the pixel value of an image (ranges from 0 to 255) represented by 8 bit length is truncated to 4 bits (ranges from 0 to 15) by considering the first four MSBs of the binary value.

By ignoring the first four LSBs an error percentage of 100% is recorded for smaller binary values with 4 MSBs given as 0000_2, but when the percentage contribution of the weight of 4 LSBs towards a pixel value ranging from 0 to 255 is considered, it's almost less than equal to 6% ($15 \div 255 \cong 6\%$) as the maximum weight of the last four LSBs is 15. In this paper these 4 bits are ignored and the first four MSBs are used for stochastic computation. The error introduced by ignoring the 4 LSBs is further reduced by considering the weight of the $5th$ bit in a 8 bit pixel value.

3. PROPOSED DESIGN

The proposed circuit for the stochastic edge detection is as shown in Fig. 3.1. Generally stochastic computation consists of three parts: firstly, the binary to stochastic conversion (BTS) which is done by using a random-number generator and a comparator; secondly, a stochastic circuit used for stochastic computation; and finally, a stochastic to binary conversion (SBC) unit used to convert the stochastic value back to binary. In most of the SC designs a counter is used for SBC and a LFSR is used for BTS conversion. The proposed design is similar to the general structure of stochastic computation, the only difference is the use of D-flip flop to delay the stochastic output generated from the stochastic circuit, a decision block and a MUX to initiate the counter value.

The decision block changes with the number of the inputs to the stochastic circuit and the operation of the stochastic circuit. In this work, edge detection unit has been implemented using the stochastic circuit described in [2] as shown in Fig. 3.1(c). It is based on well-known Robert's cross algorithm, where it computes a moving average on a window of size 2×2 for each pixel $x_{i,j}$ at row i and j of the image, and generates an output value $z_{i,j}$ according to the following formula [2].

$$z_{i,j} = 0.5 \times (|x_{i,j} - x_{i+1,j+1}| + |x_{i,j+1} - x_{i+1,j}|) \tag{3.1}$$

The circuit shown in the Fig. 3.1(c), is capable of performing this operation if the four inputs of the XOR gate are correlated. This implies that we need to use a single SNG or BTS conversion unit for this operation [2]. The select input to the MUX is generated by using a BTS conversion unit with a constant input binary value of

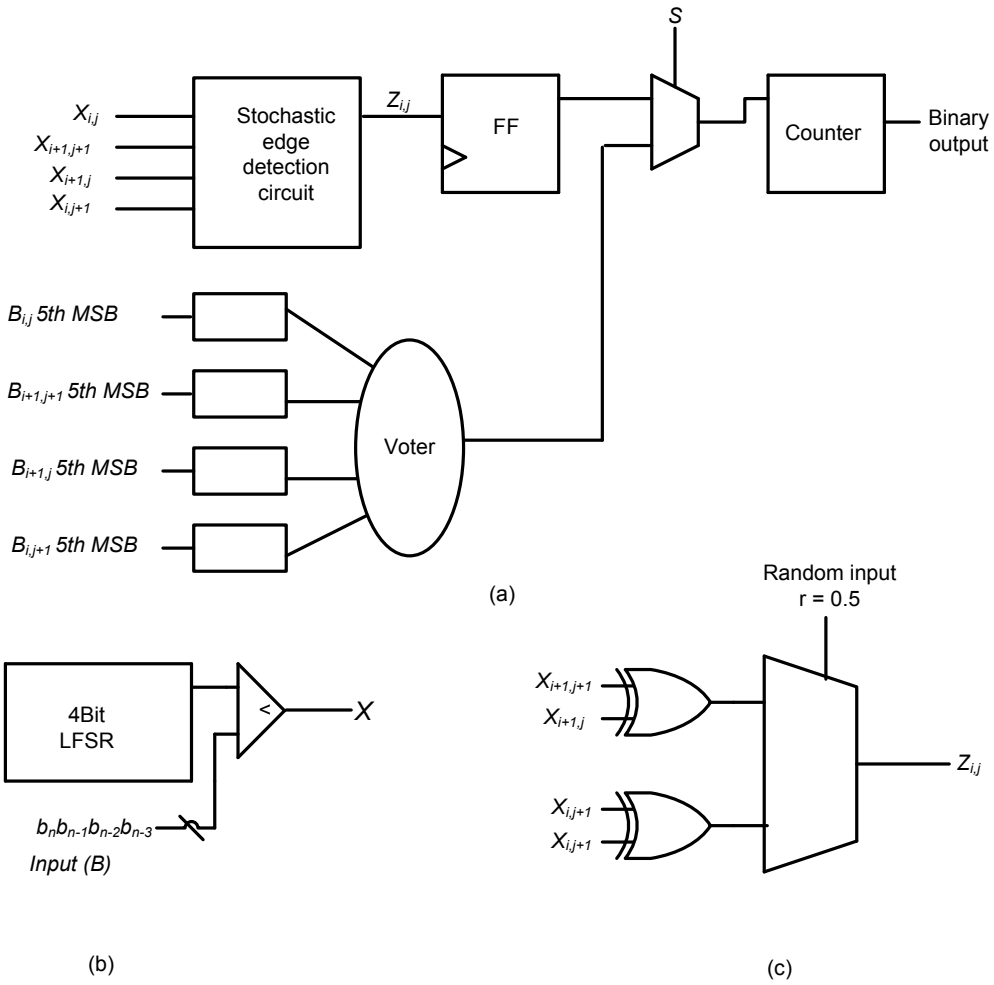


Figure 3.1. (a) Proposed circuit for approximate stochastic edge detection; (b) Stochastic bit generation; (c) Edge detection circuit used [2].

0.5 such that it generates a alternate 0s and 1s every clock cycle evaluating to a probability value of 0.5 by the end of stochastic operation.

4. ADAPTIVE TRUNCATION FOR ERROR REDUCTION

Error caused by the proposed approximation approach is solely determined by the weights of the truncated bits. If they are 0000_2, no error will be caused, since $0s$ are weightless. The maximum relative error will happen when MSBs and LSBs are given as 0000_2 (i.e., weightless) and 1111_2 (i.e., weights maximized), respectively. To reduce the maximum relative error bound, a novel adaptive truncation method has been proposed in this work.

In the proposed design shown in Fig. 3.1(a), a decision is made based on comparing the 5_{th} bits of the inputs, if any of the two inputs is '1' the counter starts counting from '1' and if not it starts from '0'. An additional delay of one clock cycle initially gets injected by using a flip flop to decide the initial stage of the counter. A MUX is controlled by a select signal in such a way that for the first clock cycle the select signal is high and for all the remaining clock cycles of the stochastic computation it remains low. In this way, the error caused from the approximation can be considerably reduced by adaptively taking 5^{th} bit when it is set. Just one extra count gets added to the counter and a weight equivalent to the truncated 4 LSBs is added. Hence, a linear increase in the clock cycles can result in significantly smaller error. The Fig. 4.1 and Fig. 4.2 show the simulation results of an edge detection circuit implemented on a standard image ("Pepper") using this approach. In this approach, the final binary value is scaled accordingly to have the pixel values in the range of 0 to 255 in decimal (i.e., the output from the counter is multiplied by a factor of 16).

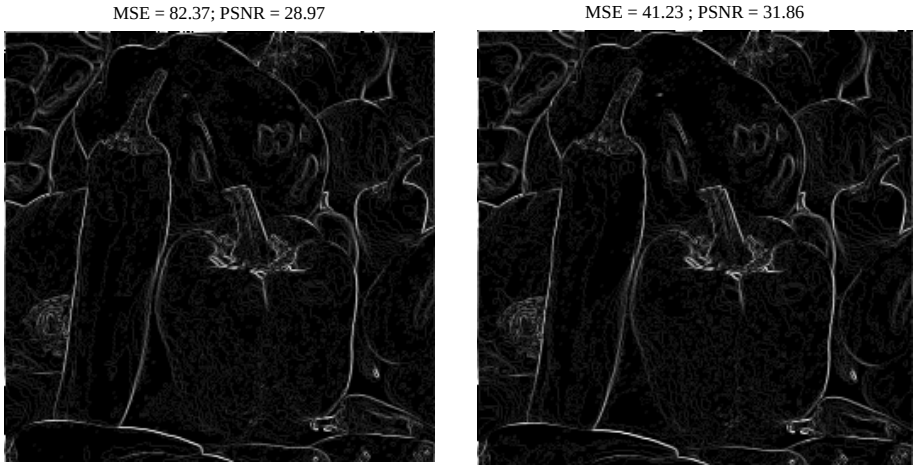
(a) Original image (b) Edge detection-original

Figure 4.1. Edge detection implemented using 8-bit length pixel values

4.1. PERFORMANCE VERIFICATION

To evaluate the acceptability and quality of the image generated by the proposed approach, PSNR (Peak Signal-to-Noise Ratio) value of the original image with respect to the generated image was calculated. PSNR is commonly adopted in the image processing field to quantify the acceptability of erroneous or noisy images [25]. PSNR value (usually in the unit of dB) can indicate the similarity of two different images. Here the edge detection image generated by using 8-bit length and the edge detection image generated by using the proposed ASC approach are used to compute the corresponding PSNR value. In the case that these two images are more similar, a higher PSNR value will be obtained. PSNR value can be calculated by the equation below.

$$PSNR = 10 \cdot \log_{10} \frac{MAX_I^2}{MSE} \tag{4.1}$$



(a) Edge detection-16 clock cycles (b) Edge detection-17 clock cycles

Figure 4.2. Edge detection implemented using 4-bit length pixel values

where $MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |I(i,j) - K(i,j)|^2$ is the mean square error of the error-free and the erroneous image, MAX_I is the maximum image pixel values (e.g., 255 in 8-bit grayscale image), m and n represent the width and height of the target image in terms of the number of pixels and $I(i,j)$ and $K(i,j)$ represent the pixel values of the error-free image and the erroneous/noisy image, respectively.

For PSNR-based numerical analysis, an open-source benchmark image called "Pepper" shown in Fig. 4.1(a) has been used. Its exact edge detection result is also presented in Fig. 4.1(b), which provides the basis of comparison. Fig. 4.2 shows the output image generated by the proposed ASC approaches including the fixed truncation and adaptive truncation, respectively. When the 4 LSBs are ignored and the error in the output image is calculated PSNR value of 28.97 dB and a mean square error (MSE) of 82.37 are recorded. When the 5_{th} bit is adaptively considered and the output of the counter is increased by '1', PSNR value increases to 31.86 dB and MSE decreases to 41.23. Hence, just by adding an additional clock cycle delay,

MSE was drastically decreased by almost 50%. When the quality of the output image is considered, both approaches have PSNR values which are in the acceptable range as all the edges of the original output image and the output images by the proposed approaches are well-matched visually.

5. CONCLUSION

In this paper, an approximate computing technique has been used to reduce the stochastic computation time drastically and still achieve acceptable results. From Fig. 4.1 and Fig. 4.2, it can be observed that the fixed 4 bit truncation yields a visually acceptable results with 16 times reduction in the total number of clock cycles. Furthermore, adding one more clock cycle by checking the weight of the 5_{th} MSB, even more accurate result can be generated.

The edge detection results suggest that this approach can be beneficial to design efficient circuits with smaller circuit size and faster operation for image processing applications, where 100% accuracy is not needed. Future work would be to implement the same design techniques to various image processing applications as well as arithmetic circuits to reduce the error percentage further by increasing the clock cycles linearly rather than exponentially.

SECTION

2. CONCLUSION

QSNGs have a better accuracy and convergence as compared to LFSRs. For a parallel implementation proposed approach using QSNGs show higher throughput values for multiplication and edge detection application. Future work to be done is to optimize the QSNG circuit for less area. The approximate stochastic computing approach used in the second part of the dissertation proved to be beneficial for edge detection application. Further analysis has to done on other image processing applications as well as on arithmetic circuits to achieve acceptable results with the linear increase in clock cycle rather than exponential.

BIBLIOGRAPHY

[1] Bert Moons and Marian Verhelst. Energy-Efficiency and Accuracy of Stochastic Computing Circuits in Emerging Technologies. *Emerging and Selected Topics in Circuits and Systems, IEEE Journal on*, 4(4):475–486, 2014.

[2] Armin Alaghi, Cheng Li, and John P Hayes. Stochastic circuits for real-time image-processing applications. In *Proceedings of the 50th Annual Design Automation Conference*, page 136. ACM, 2013.

[3] Armin Alaghi and John P Hayes. Fast and accurate computation using stochastic circuits. In *Proceedings of the conference on Design, Automation & Test in Europe*, page 76. European Design and Automation Association, 2014.

[4] Brian R Gaines. Stochastic computing. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 149–156. ACM, 1967.

[5] BR Gaines. Stochastic computing systems. In *Advances in information systems science*, pages 37–172. Springer, 1969.

[6] Ali Naderi, Shie Mannor, Mohamad Sawan, and Warren J Gross. Delayed stochastic decoding of LDPC codes. *Signal Processing, IEEE Transactions on*, 59(11):5617–5626, 2011.

[7] Hananeh Aliee and Hamid R Zarandi. Fault tree analysis using stochastic logic: A reliable and high speed computing. In *Reliability and Maintainability Symposium (RAMS), 2011 Proceedings-Annual*, pages 1–6. IEEE, 2011.

[8] Peng Li and David J Lilja. Using stochastic computing to implement digital image processing algorithms. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 154–161. IEEE, 2011.

[9] Yun-Nan Chang and Keshab Parhi. Architectures for digital filters using stochastic computing. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 2697–2701. IEEE, 2013.

[10] Naman Saraf, Kia Bazargan, David J Lilja, and Marc D Riedel. IIR filters using stochastic arithmetic. In *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, pages 1–6. IEEE, 2014.

[11] Peng Li and David J Lilja. Accelerating the performance of stochastic encoding-based computations by sharing bits in consecutive bit streams. In *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on*, pages 257–260. IEEE, 2013.

[12] Hideyuki Ichihara, Shin Ishii, Daiki Sunamori, Tsuyoshi Iwagaki, and Takeru Inoue. Compact and accurate stochastic circuits with shared random number sources. In *Computer Design (ICCD), 2014 32nd IEEE International Conference on*, pages 361–366. IEEE, 2014.

[13] Armin Alaghi and John P Hayes. A spectral transform approach to stochastic circuits. In *Computer Design (ICCD), 2012 IEEE 30th International Conference on*, pages 315–321. IEEE, 2012.

[14] Armin Alaghi and John Hayes. STRAUSS: Spectral Transform Use in Stochastic Circuit Synthesis. 2012.

[15] Armin Alaghi and John P Hayes. Survey of stochastic computing. *ACM Transactions on Embedded computing systems (TECS)*, 12(2s):92, 2013.

[16] Rajit Manohar. Comparing Stochastic and Deterministic Computing.

[17] PK Gupta and R Kumaresan. Binary multiplication with PN sequences. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 36(4):603–606, 1988.

[18] Paul Bratley, Bennett L Fox, and Harald Niederreiter. Implementation and tests of low-discrepancy sequences. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 2(3):195–213, 1992.

[19] Ishaan L Dalal, Deian Stefan, and Jared Harwayne-Gidansky. Low discrepancy sequences for monte carlo simulations on reconfigurable platforms. In *Application-Specific Systems, Architectures and Processors, 2008. ASAP 2008. International Conference on*, pages 108–113. IEEE, 2008.

[20] John H Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 2(1):84–90, 1960.

[21] IM Sobol. The distribution of points in a cube and the approximate evaluation of integrals, Zh. Vychisl. Mat. i Mat. Fiz. 7 (1967), 784–802.

[22] Behrooz Parhami. Efficient hamming weight comparators for binary vectors based on accumulative and up/down parallel counters. *Circuits and Systems II: Express Briefs, IEEE Transactions on*, 56(2):167–171, 2009.

[23] J. Han and M. Orshansky. Approximate computing: An emerging paradigm for energy-efficient design. In *2013 18th IEEE European Test Symposium (ETS)*, pages 1–6, May 2013.

[24] R. Venkatesan, A. Agarwal, K. Roy, and A. Raghunathan. Macaco: Modeling and analysis of circuits for approximate computing. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 667–673, Nov 2011.

[25] T. Y. Hsieh, Y. H. Peng, and C. C. Ku. An efficient test methodology for image processing applications based on error-tolerance. In *2013 22nd Asian Test Symposium*, pages 289–294, Nov 2013.

VITA

Ramu Seva was born in Hyderabad, India. After completing his schoolwork at International School in 2006, Ramu did his engineering from Institute of Aeronautical Engineering affiliated to Jawaharlal Nehru Technological University in Hyderabad. He received a Bachelor of Technology with a major in Electrical and Electronics Engineering from Jawaharlal Nehru Technological University in May 2012. In June 2012, he was employed as a graduate engineer at IFB Industries, Goa, India and then followed to start his own start-up Clique Media Solutions, Hyderabad, India in January 2013. In May 2017, he completed his MS degree from Computer engineering department of Missouri University of Science and Technology at Rolla, MO, USA.