ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

STOCHASTIC BITSTREAM-BASED VISION AND LEARNING MACHINES

Ph.D. THESIS

Sercan AYGÜN

Department of Electronics and Communication Engineering

Electronics Engineering Programme

OCTOBER 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

STOCHASTIC BITSTREAM-BASED VISION AND LEARNING MACHINES

Ph.D. THESIS

Sercan AYGÜN
(504152207)

Department of Electronics and Communication Engineering

Electronics Engineering Programme

Thesis Advisor: Prof. Dr. Ece Olcay GÜNEŞ

OCTOBER 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

STOKASTİK BİT AKIŞI TABANLI GÖRÜ VE ÖĞRENME MAKİNELERİ

DOKTORA TEZİ

Sercan AYGÜN
(504152207)

Elektronik ve Haberleşme Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Ece Olcay GÜNEŞ

EKİM 2022

Sercan AYGÜN, a Ph.D. student of ITU Graduate School student ID 504152207, successfully defended the dissertation entitled "STOCHASTIC BITSTREAM-BASED VISION AND LEARNING MACHINES", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Ece Olcay GÜNEŞ** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Mürvet KIRCI** Istanbul Technical University	.....
	Assoc. Prof. Dr. Bülent BOLAT Yildiz Technical University	.....
	Prof. Dr. İsmail Serdar ÖZOĞUZ Istanbul Technical University	.....
	Asst. Prof. Dr. M. Hassan NAJAFI University of Louisiana at Lafayette	.....

Date of Submission : 25 September 2022
Date of Defense : 21 October 2022

To my beloved family,

FOREWORD

Obtaining a Ph.D. is a long journey, and many people have helped me during my research. First, I share my loads of appreciation to my dear thesis supervisor Prof. Dr. Ece Olcay GÜNEŞ, who has supervised my research since my master's degree and helped with all the issues related to science and life. Her technical feedback, patience, and understanding allowed this research to be completed successfully. She will always be a role model for me.

I would like to express my gratitude to the thesis committee, which always provided valuable feedback. I would also like to thank the Department of Electronics and Communication Engineering of Istanbul Technical University for the courses that contributed to my dissertation. I have been lucky to be a member of the Istanbul Technical University SMART Laboratory, led by Assoc. Prof. Dr. Mürvet KIRCI, and share the same workspace with my fellow researchers, Halil DURMUŞ, M.Sc., Bilgi Görkem YAZGAÇ, M.Sc., and Dr. Lida KOUHALVANDI.

From the very beginning, since 2014, every single person has encouraged me positively at the Department of Computer Engineering, Yildiz Technical University, where I worked as a research assistant. I am grateful for all of the people in my department and university. On this occasion, I would like to thank my undergraduate students, Akif Berkay GÜRCAN, Muhammad ARSLAN, Mecit AHMETOĞLU, Bahattin Cihan ÜNAL, and Aizhan MAKSATBEK KYZY, for their discussions and efforts on stochastic computing.

I appreciate Prof. Dr. Christophe DE VLEESCHOUWER, who directed me for my dissertation during my stay as an exchange student at the Université Catholique de Louvain - ICTEAM, Belgium. Thanks to his vast knowledge and feedback on neural networks, he gave a new perspective to my dissertation. I also thank Dr. Julien MOREAU, then a postdoctoral fellow with whom I shared my office during this visit, for encouraging me to do my research.

I am thankful to Asst. Prof. Dr. M. Hassan NAJAFI for accepting me as a research scholar in his research group in the Emerging Computing Technologies (ECT) Laboratory at the University of Louisiana at Lafayette, LA, USA. Dr. Najafi's feedback made my research time very efficient and enabled my thesis to take its final form. His deep knowledge and expertise in stochastic computing gave my dissertation new directions, especially in stochastic computing-based vision machines and emerging neural network technologies. I am also thankful to the members of our research group, with whom I have had a lot of great discussions at the University of Louisiana at Lafayette, LA, USA.

Special thanks go to the native speaker Jake SUNDBERG, M.Sc., M.A. from the University of Louisiana at Lafayette, LA, USA, for his help with the English language editing of this dissertation and technical advice. I also thank Savaş OKYAY, M.Sc., for his final reading and checking of the thesis.

Last but not least, I want to share my gratitude to my mother and father. They supported me in all sorts of manners during my life. My beloved family always listened to and consoled me with any issues in my academic journey.

This thesis was financially supported by Istanbul Technical University (ITU), Scientific Research Project Unit under Grant MDK-2018-41532. The ITU Erasmus+ program supported my research visit to Belgium, Université Catholique de Louvain. My other research visit to the University of Louisiana at Lafayette, LA, USA, under the supervision of Asst. Prof. Dr. M. Hassan NAJAFI, has been supported by NSF Grant #2019511, Louisiana Board of Regents Support Fund #LEQSF(2020-23)-RD-A-26, and generous gifts from Cisco, NVIDIA, and Xilinx. I thank them all for their support.

October 2022

Sercan AYGÜN
(Electronics & Computer Engineer, M.Sc.)

TABLE OF CONTENTS

Page

ABBREVIATIONS

1D : One-Dimensional
2D : Two-Dimensional
3D : Three-Dimensional
abs. : Absolute
AC : Approximate Computing
ADDIE : Adaptive Digital Element
ANN : Artificial Neural Network
APC : Approximate Parallel Counter
ASC : Approximate Stochastic Computing
ASIC : Application-Specific Integrated Circuit
B : Background (image)
BitSAD : Bitstream Synthesizer and Designer
BNN : Binarized Neural Network
BPE : Bipolar Encoding
BS : Batch Size
BSBNN : Bitstream Processing Binarized Neural Network
C : Composited (image)
CCD : Charge-Coupled Device
CLR : Clear (pin of a flip-flop)
CMOS : Complementary Metal-Oxide Semiconductor
CNN : Convolutional Neural Network
CONVN : Conventional (binary, deterministic processing)
CORDIC : COordinate Rotation DIgital Computer
CORDIV : Correlated Division
CT : Contingency Table
CT0 : Contingency Table Zero Correlation
CTLFSR : Contingency Table Linear-Feedback Shift Register
CTRAND : Contingency Table Random (i.e., binomial distribution)
DC : Deterministic Computing
Det. : Deterministic
DI : Dice (similarity)
DNN : Deep Neural Network
DSP : Digital Signal Processing
ESL : Extended Stochastic Logic
F : Foreground (image)
ffs : Flip-flops
FIR : Finite Impulse Response
FPGA : Field-Programmable Gate Array
FPNN : Full-Precision Neural Network
FSM : Finite State Machine
GND : Ground

GPU : Graphics Processing Unit
HL : Hidden Layer
HR : Hit Rate
HRU : Hardware Resource Utilization
IIR : Infinite Impulse Response
JAC : Jaccard (similarity)
JPEG : Joint Photographic Experts Group
K2 : Kulczynski 2 (similarity)
KDE : Kernel Density Estimation
LD : Low-Discrepancy (sequence)
LDPC : Low-Density Parity-Check
LFSR : Linear-Feedback Shift Register
LLR : Log Likelihood Ratio
LR : Likelihood Ratio
LSB : Least Significant Bit
MAA : Maximum Achievable Accuracy
MAC : Multiply and Accumulate
MAE : Mean Absolute Error
MATLAB : Matrix Laboratory
MEM : Memory
MLP : Multi-Layer Perceptron
MNIST : Modified National Institute of Standards and Technology
MNIST-C : Modified National Institute of Standards and Technology-Corrupted
MOD : Modulus
MSB : Most Significant Bit
MSD : Multiplication Smooth Design
MSE : Mean Squared Error
MSSC : Mixed-Signal Stochastic Computing
MUX : Multiplexer
N/A : Not Available
NaN : Not a Number
NN : Neural Network
nohl : Number of Hidden Layers
OC : Ochiai (similarity)
ODE : Ordinary Differential Equation
PC : Pearson Correlation
PD : Plain Design
PH : Pure Hardware
PS : Pure Software
PSNR : Peak Signal-to-Noise Ratio
PTM : Probabilistic Transfer Matrix
PWM : Pulse Width Modulation
QNN : Quantized Neural Network
QR : Quick Response
QSNG : Quasi-Stochastic Number Generator
RBF : Radial Basis Function
Ref. : Reference(s)
ReLU : Rectified Linear Unit
RGB : Red-Green-Blue

RMSE : Root-Mean-Square Error
RNG : Random Number Generator
RNN : Recurrent Neural Network
RT : Runtime
RV : Random Variable
SAD : Sum of the Absolute Differences
SC : Stochastic Computing
SCC : Stochastic Cross-Correlation
SCLfsr : Stochastic Computing Linear-Feedback Shift Register
SC-MAC : Stochastic Computing Multiply and Accumulate
SCNN : Stochastic Computing-based Neural Network
SCRand : Stochastic Computing Random (i.e., binomial distribution)
SCSobol : Stochastic Computing Sobol (sequence)
Sel : Selection (multiplexer-MUX)
SNN : Spiking Neural Network
SPD : Semi-Plain Design
SSD : Sum of the Squared Differences
SSIM : Structural Similarity
Sto. : Stochastic
StoPE : Stochastic Processing Elements
StoRM : Stochastic Recognition and Mining (processor)
STRAUSS : Spectral Transform Use in Stochastic Circuit Synthesis
SVM : Support Vector Machine
TC : Total Count (of logic-1s)
TCO : Total Count of logic-1s at the Output
TMR : Triple Modular Redundancy
UPE : Unipolar Encoding
VCC : Voltage Common Collector

SYMBOLS

$\boldsymbol{N}$	: Bitstream size
$\boldsymbol{\mu}$**W**	: Microwatt
$\boldsymbol{\mu}$**m**	: Micrometer
ns	: Nanosecond
MHz	: Megahertz
nm	: Nanometer
dB	: Decibel
sec	: Second
$\boldsymbol{\mu}$	: Mean
$\boldsymbol{\sigma}$	: Standard deviation
X	: Scalar to be represented in a bitstream
$\boldsymbol{X}$	: Bitstream (probability: P_Y)
$\hat{X}$	: Integer scalar to be represented in a bitstream
$X_m^{(b)}$, x_m	: Single bit of a bitstream, m^{th} element of $\boldsymbol{X}$, i.e., $\boldsymbol{X}(m)$
$\mathbb{B}$	: Boolean set, $\{0, 1\}$
$\boldsymbol{\kappa}$	: Bit precision of image representation
$\boldsymbol{G}$	: Logic gate
$\boldsymbol{L}$	: Multiplexer input size (2^m)
$\boldsymbol{\beta}$	: Logic input operand count
$\boldsymbol{S}$	: Multiplexer selection port (alternatively pre-activation in Chapter 5)
$\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{d}$	: Contingency table primitives
$\wedge$, $\vee$, $\neg$	: AND, OR, NOT
$\oplus$, $\odot$	: XOR, XNOR
$\boldsymbol{a}_{min}$, $\boldsymbol{a}_{max}$	: Minimum correlation, maximum correlation (via a primitive)
$\boldsymbol{a}_{mid}$	: Zero correlation with midpoint approximation
$\boldsymbol{a}_{poly}$	: Zero correlation with polynomial fitting
$\boldsymbol{a}_{best}$	: Zero correlation with the best optimization
$\boldsymbol{a}_{ref}$	: Reference a primitive guarantees the zero correlation
$\mathcal{X}_1 \times \mathcal{X}_2$	: Cartesian product
$\boldsymbol{O}_{CT}$	: Occurrence of a contingency table
$\boldsymbol{C}$, $\boldsymbol{P}$	: Combination, Permutation
$\mathbb{E}$	: Expectation operator
$\boldsymbol{I}$, $\boldsymbol{K}$	: Image, Kernel
$\boldsymbol{r}$	: Row size of an image
$\boldsymbol{c}$	: Column size of an image
$\boldsymbol{k}$	: Kernel size
$\boldsymbol{K}_{any}$	: Image kernel coefficient
$\boldsymbol{w}_k$	: Kernel scaling weight
$\boldsymbol{\nabla}$	: Gradient operator
$\boldsymbol{G}_{\mathrm{x}}$, $\boldsymbol{G}_{\mathrm{y}}$	: Image gradient partial derivatives
$\boldsymbol{\Gamma}$	: Scale factor effector (for MUX)

$\tilde{\times}$: Approximate multiplication
$\boldsymbol{M}$: Temporary matching image (template matching)
$\hat{\boldsymbol{I}}$, $\hat{\boldsymbol{K}}$: Noisy image, Noisy kernel
$\boldsymbol{\rho}$: Image scaling factor in bilinear interpolation
$\bar{\boldsymbol{I}}$: Resized image after bilinear interpolation
$\boldsymbol{d}_x$, $\boldsymbol{d}_y$: Relative positions in bilinear interpolation
$\boldsymbol{\alpha}$: Foreground image opacity
$\mathbf{x}$: x-axis in the xy plane
$\mathbf{y}$: y-axis in the xy plane
$\mathbf{x}_1$: Point 1 in the x-axis
$\mathbf{y}_2$: Point 2 in the y-axis
$\boldsymbol{n}$: Sample size
$\boldsymbol{Y}$, $\boldsymbol{y}_i$: Expected (Target, Exact) output (probability: P_Y)
$\hat{\boldsymbol{Y}}$, $\hat{\boldsymbol{y}}_i$: Obtained (Estimated, Predicted) output (probability: $\hat{P}_Y$)
$\boldsymbol{x}$, $\boldsymbol{X}$: Neuron input
$\boldsymbol{w}$: Weight
$\boldsymbol{b}$, $\boldsymbol{w0}$, $\boldsymbol{w}_{j,0}$: Bias
$\boldsymbol{y}$: Neuron output
$\boldsymbol{f}$, $\boldsymbol{g}$: Activation functions
$\boldsymbol{\phi}_j$: Nonlinear basis function
$\boldsymbol{\eta}$: Learning rate
$\boldsymbol{m}$: Slope of a line (in Chapter 5)
$\boldsymbol{D}$, $\boldsymbol{D}'$: Data, Noisy data
$\boldsymbol{E}_{total}$: Total error
$\boldsymbol{\delta}$: Weighted sum of the errors
$\boldsymbol{T}$: Predefined hard-coded counter threshold

LIST OF TABLES

Page

LIST OF FIGURES

Page

STOCHASTIC BITSTREAM-BASED VISION AND LEARNING MACHINES

SUMMARY

Stochastic computing (SC), a paradigm that dates back to the 1960s, has re-emerged in the last decade by being a tool for emerging technology development. SC adopts unconventional data representation encoding scalars (X) into binary streams. Unlike conventional binary radix, the cumulative values of the *logic-1*s and *logic-0*s in the bitstream with a probability P_X are evaluated free from the bit position. Thus, SC provides simple circuits of complex functions (e.g., multiplication with a single AND gate) and soft error-tolerant (e.g., robust to bit-flips) systems. However, latency inevitably occurs because SC expresses bitstreams in long sequences size of N (512 bits, 1024 bits, etc.) for high accuracy. Although several solutions for latency in SC hardware systems have been described in the literature, the software simulation of the SC framework falls behind. Therefore, this doctoral thesis proposes the general framework of software-based SC simulations considering both latency and memory issues. This study also discusses the systematic view of SC-based image processing and proposes a new concept, namely the bitstream processing binarized neural network.

The dissertation begins with an introduction presenting a short literature check, the purpose of the thesis, and the hypothesis with the major and minor contributions. Then, the background part presents basic SC concepts such as bitstream structure, scalar encoding techniques, correlation, random bitstream generation, SC building block elements, and arithmetic. A cascaded multiplexer (MUX) optimization algorithm is proposed for scaled additions of multiple operands. Also, a comprehensive survey on vision and learning machines is presented, examining previous efforts and exploring the dissertations in the last decade.

The software-driven SC is further discussed by proposing the utilization of a contingency table (CT). The generation and processing of the lengthy SC bitstreams pose the simulation runtime and memory occupation problems. Considering that applications require intensive arithmetic operations such as artificial neural network (ANN), the problems become significant. To tackle these, scalar-only processing of CT is proposed. CT is set by two input scalars ($X1$, $X2$), bitstream cross-correlation, and bitstream length (N). The main objective is to reach the desired logic output using only the scalar values instead of generating bitstreams and processing bit-by-bit with logical operators. The CT holds the cumulative values of four logic pairs, 11, 10, 01, and 00, for any overlapping bits of the two bitstream operands. These cumulative values denoted as a, b, c, and d, respectively, are the CT primitives. The correlation value of the two non-generated bitstream operands sets the prior CT primitive, a, based on the stochastic cross-correlation (SCC) metric. CT is established for maximum ($SCC = 1$,

a is maximum), minimum ($SCC = -1$, a is minimum), or near-zero ($SCC \approx 0$, a is based on the proposed algorithm) correlation. Zero correlation is vital for the accuracy of some SC-based arithmetic operations (e.g., multiplying bipolar encoded bitstreams by XNOR). Therefore, three methods are proposed to set the a prior primitive for near-zero correlation with an algorithm. After the determination of a, the proposed formulas define b, c, and d. The linear combination of CT primitives obtains each logical operator. (e.g., XNOR is $a + d$.) The CT emulates the entire hardware system in software via the proposed model of random number generators by including SC's built-in random fluctuation error. The random source models imitate Sobol low-discrepancy sequences, linear-feedback shift register (LFSR), and the binomial distribution. Also, CT can simulate all 2^{2N} Cartesian combinations of two input bitstreams; therefore, there is no need for random sampling like in the Monte Carlo simulation.

Next, several image processing techniques on behalf of SC-based vision machines are discussed. The first is the reinterpretation of SC-based mean filtering for noise removal. The second is the Prewitt edge detector, a case study for inspecting the different levels of hardware approximations based on the MUX scaling factor. The plain design (PD) exhibits remarkable edge detection performance in the case of excessive noise. The third technique is template matching to detect finder patterns of quick response (QR) codes in a noisy environment. Pattern matching is accomplished by feeding a single AND gate with bitstreams, and SC achieves slight outperformance compared to the deterministic counterpart. As the first study of its kind, to the best of our knowledge, the two other algorithms, bilinear interpolation and image compositing, are synthesized with SC. Bilinear interpolation is a method of scaling the dimensions of images. It is proven that the hardware equivalent can be a simple 4-to-1 MUX fed by bitstreams. The last technique, image compositing, outputs a new composited image by combining the background and foreground images. It is proposed to obtain the composited image utilizing a simple 2-to-1 MUX via SC. Both techniques are verified with the help of CT. Bilinear interpolation and image compositing highlight another contribution in validating different random number generator models (Sobol, LFSR, and binomial distribution) on behalf of CT.

Finally, this dissertation focuses on the adaptation of SC to learning systems. First of all, a single stochastic neuron is designed, and higher classification accuracy is achieved in early epochs when the neuron is trained with noisy data. Then, the mathematics behind the learning procedure of conventional multi-layer neural network architecture is reviewed. A fully-bitstream processing binarized neural network (BSBNN) is proposed in comparison with the traditional binarized neural network (BNN) architecture. BNNs express network weights and neuron activations with one bit; however, this causes a fragile structure against soft errors such as bit-flips occurring on emerging hardware and memory technologies. In traditional BNN, the neuron activation is $+1$ or -1, decided by subtraction from a threshold value ($+1$ is *logic-1* and -1 is *logic-0* in hardware). In our proposal, the power of bipolar encoding is used, and the neuron output is decided by checking the majority-minority balance of *logic-1*s or *logic-0*s in the pre-activation (S) bitstream. Since this control is performed simultaneously during accumulation (counter) via a masking logic, an additional activation module is not required. Thus, less hardware resource utilization is achieved (30% per neuron basis). In addition, a more efficient architecture against bit-flip errors is provided. The proposed architecture proves superior robustness over

the conventional fragile BNN regarding image- and weight-based bit-flips. All four different networks are then tested: BNN, BSBNN, stochastic computing-based neural network (SCNN), and full-precision neural network (FPNN), having no quantization. Considering image-based corruptions (contrast, Gaussian blur, fog, speckle noise, zoom blur, etc.), different training scenarios are compared with and without the awareness of corruption in network training. The proposed BSBNN architecture exhibits comparable classification accuracy, and the importance of error-sensitive training in binary networks (BNN and BSBNN) is underscored. In the last section, the performance of the CT-based network simulation is finally unveiled. SC-based XNOR multiplications are present in the classifier part of a convolutional neural network (CNN) architecture. Emulating multiplication via CT results in a faster training runtime than the counterpart with actual bitstream processing. Training a bitstream processing neural network with actual bitstreams (with bit-by-bit processing) shows an exponentially increasing runtime as N increases. Conversely, CT-based simulation provides linear training runtime independent of bitstream length.

STOKASTİK BİT AKIŞI TABANLI GÖRÜ VE ÖĞRENME MAKİNELERİ

ÖZET

İlk olarak 1960'larda sunulan stokastik hesaplama (stochastic computing, SC) son yıllarda yeniden ilgi odağı olmuş ve gelişmekte olan yeni nesil teknolojilerde (emerging technologies) kullanılmaya başlanmıştır. Bu hesaplama paradigması, verileri (X) geleneksel ikili sayı tabanı (conventional binary radix) kullanarak değil, alışılmadık temsil ile; *lojik-1* ve *lojik-0*'ların bit akışındaki birikimli değerlerine ve bit olasılığına (P_X) göre bit pozisyonundan bağımsız olarak kodlar. Böylece karmaşık fonksiyonların basit devreleri (örneğin tek bir AND kapısı ile çarpma işlemi) ve yumuşak hatalara (örneğin bit çevirme) karşı toleranslı sistemler elde edilir. Stokastik hesaplama bit akışlarını N boyutunda (512 bit, 1024 bit vb.) uzun dizilerle ifade ettiğinden, gecikmeler (latency) kaçınılmaz olarak meydana gelmektedir. Literatürde, gerçek donanım sistemleri açısından bu soruna çözüm aranmışsa da yazılım benzetim çerçevesi geri planda kalmaktadır. Bu sebeple, bu doktora tezinde stokastik sistemlerin yazılım tabanlı benzetimine ve gecikmeler ile hafıza kullanım sorunlarına ilişkin çözümler sunulmaktadır. Bu tez ayrıca stokastik hesaplama tabanlı görüntü işlemeye sistematik bakışı tartışmakta ve yeni bit akış işlemeli ikili sinir ağlarını önermektedir.

İlk bölümde literatüre hızlı bakış ve tezin amacının devamında sunulan hipotez, katkılar ile birlikte verilmektedir. Ardından, ikinci bölümde, temel tasarım ögeleri tanıtılmaktadır. Bu bölüm tezin arka planını oluşturmaktadır. Bit akışı yapısı, sayı kodlama teknikleri, korelasyon, rasgele bit akış üreteçleri, stokastik hesaplama yapıtaşı ögeleri ve aritmetik operasyonlar ile ideal olmayan durumlardan gürültü ve yumuşak hata kavramları tanıtılmaktadır. Stokastik hesaplama gürültüye karşı gürbüz bir hesaplama metodu olduğundan, tezin ilerleyen bölümleri için önemli olan bit çevirme (bit-flip) hataları gibi yumuşak hataların ön tanımı verilmiştir. Ayrıca bu bölümde, çok girişli toplama devresi için kademelenmiş mimaride optimizasyon sunulmaktadır. Bu bölüm ilk bölüme göre genişletilmiş bir literatür derlemesi alt bölümüne sahiptir. Literatürde yer alan özellikle görü ve makine öğrenmesi konularındaki çalışmalar irdelenmiş, son on yıldaki doktora tezleri, sunulan tez çalışmasının literatürdeki yerini belirlemek adına incelenmiştir.

Tezin ilerleyen bölümlerinde, yazılım tabanlı benzetim için literatüre sunulan stokastik hesaplamada olumsallık tablolarının (contingency table, CT) kullanımı anlatılmaktadır. Bu hesaplama paradigmasında verilerin işlenmesi, gecikmelere sebep olan uzun bit akışları üzerinden gerçekleştirildiği için özellikle yoğun aritmetik işlemlerin olduğu yapay sinir ağı (artificial neural network, ANN) gibi uygulamalarda, benzetim zamanı açısından darboğaz meydana gelmektedir. Öte yandan, sistem benzetimi için değişken olarak tanımlanan uzun bit akışları (özellikle 128 bit ve daha uzun boyutlu), yazılım ortamında kapladığı hafıza alanı bakımından sorun teşkil etmektedir. Monte Carlo gibi benzetim yöntemleriyle çoklu verilerin testi, hafıza

ve benzetim zamanı açısından problem yaratmaktadır. Bu tez çalışmasının önemli katkılarından biri olan yazılımda hafıza ve gecikme sorununun çözümü için şu amaç güdülür: *Girdi bit akışlarının lojik operasyon sonuçları için, bit akışı üretmeden ve lojik operatörler kullanmadan, sadece skaler değerler üzerinden lojik benzetim yapabilmek*. Üçüncü bölümde önerilen yaklaşımla bunun nasıl mümkün olabileceğine dair ön sonuçlar sunulmuş ve ilgili benzetim yaklaşımının görü ve sinir ağları açısından uygulama sonuçları sırasıyla dördüncü ve beşinci bölümde genişletilmiştir (*New perspectives toward CT simulation-I*, *II* ve *III* alt başlıklarında). Böylece, hem hafıza ve çalışma zamanı açısından verimli bir yaklaşım sunulmuş hem de N boyutlu iki bit akışı işleneninin tüm 2^{2N} durumlu kartezyen çifti benzetime dâhil edilmiştir; yani diğer benzetimler gibi veriler arasından rasgele örneklemeye gerek kalmamıştır.

Olumsallık tablosu; iki giriş skaleri ($X1$, $X2$), bit akışı çapraz korelasyonu ve bit akışı uzunluğu (N) ile kurulur. Korelasyon kontrolü, bit akışları olmadan hesaplama yapabilmek için gereklidir. Stokastik hesaplama için ise üç temel korelasyon noktası kritiktir: *maksimum*, *minimum* ve *sıfır* korelasyon. İlgili bölümde bu korelasyon değerleri için olumsallık tablosunun kurulumu verilmektedir. Özellikle sıfır korelasyonlu (korelasyonsuz - uncorrelated) bit akışları ile işlem yapmak bazı stokastik tabanlı aritmetik işlemlerin doğruluğu açısından çok önemlidir. Örneğin, bipolar kodlama (bipolar encoding, BPE) ile kodlamış iki sayıya ait bit akışlarının çarpılması, XNOR lojik işleci ile yapılmaktadır. Bu iki sayıya ait bit akışlarının birbirleri ile olan korelasyonu çarpma işleminin doğruluğunu etkilemektedir. Literatürde var olan "stokastik çapraz korelasyon" (stochastic cross-correlation, *SCC*) metriği üzerinden kurgulanan benzetim yöntemi ile sıfır korelasyon için olumsallık tablosunun kurulumuna ait üç farklı yöntem önerilmektedir. *SCC* metriği 1 değerini vererek bit akışları arasındaki maksimum korelasyonu ölçerken, -1 değeri ile minimum korelasyonu belirtir. Sıfır korelasyon ise $SCC \approx 0$ olduğunda, yani birbirinden bağımsız bit akışları söz konusu ise elde edilir. Olumsallık tablosu iki işlenen bit akışının herhangi üst üste denk gelen bitleri için lojik 11, 10, 01 veya 00 çakışmalarına ait dört çiftin akış boyunca birikimli değerlerini tutar. Bu değerler sırasıyla a, b, c ve d ile anılmaktadır ve olumsallık tablosu ilkelleri (CT primitives) olarak isimlendirilir. Burada a değeri, yani iki bit akışının herhangi m'inci bitine ait lojik değerin her ikisinde de 1 olması durumu, *öncelikli olumsallık tablosu elemanı* olarak nitelendirilmiştir. Diğer tablo elemanları, b, c ve d ise a belirlendikten sonra önerilen formüller ile hesaplanır. CT ilkellerinin doğrusal kombinasyonu, lojik işleçlere bağlı sonucu verir (örneğin, XNOR $a+d$'dir.). Maksimum korelasyon (a değerinin maksimum ve $SCC = 1$ olması) ve minimum korelasyon (a değerinin minimum ve $SCC = -1$ olması) kolayca a öncelikli elemanına göre kurulmaktadır. Sıfıra yakın korelasyon ise önerilen üç farklı yöntemde bu a değerinin ayarlanması ile sağlanmaktadır. İlk yöntem maksimum ve minimum a değerlerinin aritmetik ortalamasını alırken, ikinci yöntem polinom yakıştırma yaklaşımı kullanır. Üçüncü ve en başarılı yöntem ise *SCC* metriğinin optimizasyonu ile elde edilmektedir.

Diğer yandan olumsallık tabloları, hata ölçümü ve benzetimi açısından da oldukça önemli sonuçlar ortaya koymaktadır. Farklı ikili korelasyon metrikleri (Jaccard, Dice, Kulczynski 2, Pearson ve Ochiai) ile hata benzetimleri, tüm 2^{2N} kartezyen çiftleri üzerinden çarpma aritmetiği için verilir. Metrik değerinin dağılımına (popülasyon) karşı aritmetik hata grafikleri tüm giriş kombinasyonları üzerinden sunulabilir; çünkü olumsallık tablosuna bağlı tezde önerilen *ortaya çıkma* (occurrence, O_{CT}) formülü her eşsiz korelasyon değerinin ortaya çıkış sayısını hızlıca hesaplayabilir. Önemli bir katkı

olarak rasgele dalgalanma hatasının (random fluctuation error) benzetime eklenmesi de literatüre sunulan diğer öneridir. Stokastik hesaplama için kurulacak olan bir öykünüm aracı için gerekli olabilecek rasgele kaynak modelleri; Sobol düşük tutarsızlık dizi modeli, doğrusal geri beslemeli kaydırmalı yazmaç modeli (linear-feedback shift register, LFSR) ve binom dağılım modeli, yine ilgili bölümde olumsallık tablosu kapsamında önerilmiştir.

Görüntü işleme açısından bit akış tabanlı hesaplama dördüncü bölümde verilmektedir. Görüntü pikselleri bit akışları ile ifade edilir. Görüntü işleme algoritması basit lojik sistemlerle (AND kapısı ve çoklayıcı - multiplexer, MUX) gerçekleştirilir. Bir yeniden yorumlama ile ilk öneri gürültü giderici ortalama filtresi (mean filtering) üzerine kurulmuştur. Ardından, literatürde önerilen yöntemlere de atıfta bulunarak görüntü toplama gürültüsü (image acquisition noise) için farklı mimarilerin görüntü kenar bulma (edge detection) analizi sunulmaktadır. Bu mimarilerde "yaklaşık hesaplama" (approximate computing, AC) çerçevesinde sistem karmaşıklığı üç farklı model ile ele alınır; temel motivasyon, ölçekli toplayıcının (MUX) ölçek faktörüne müdahale etmektir. Ölçek faktörü toplama işlemi sonucunu MUX seçim bağlantı noktasının (selection port) sahip olduğu bit akışı olasılık değerine (P_S) göre ölçekler; sonuç beklenen değerden sapar. Bu örnek çalışma için kenar tespitinde kullanılan filtre Prewitt filtresidir. Önerilen üç farklı model arasında, daha yaklaşık (approximate) olmasına rağmen sade tasarım (plain design, PD) gürültü gürbüzlüğü açısından daha iyi tepe sinyal-gürültü oranı (peak signal-to-noise ratio, PSNR) sunmuştur. Sonuçların yapısal benzerlik ölçüsü (structural similarity, SSIM) ile incelemesi de verilmektedir. Görüntü işleme açısından stokastik hesaplamanın şablon eşleştirme (template matching) ile kullanımı bir diğer katkı olarak sunulmaktadır. Tek bir AND kapısının bit akışları ile beslenmesiyle şablon eşleştirme gerçekleştirilir. Test için kare kodların (quick response code, QR code) gürültülü ortamda tespit şablonu üzerinden analizi verilmektedir. Evrişim (convolution) işlemi ile aranan şablon, kare kod üzerinde gezdirilerek kodda bulunan üç adet noktanın (finder pattern - bulucu desen) tespit doğruluk yüzdesi hesaplanmaktadır. Gauss gürültüsü ve alacalı gürültüye sahip görüntülerin testi, gürbüzlüğü açısından stokastik hesaplamanın gerekirci (deterministic) hesaplamaya olan hafif üstünlüğü ile sonuçlanmıştır.

Bit akışı tabanlı görü makineleri kapsamında diğer iki algoritma, çift doğrusal aradeğerleme (bilinear interpolation) ve görüntü birleştirme (image compositing), bilgimiz dâhilinde stokastik hesaplama ile ilk kez bu tez çalışmasında sentezlenmektedir. Çift doğrusal aradeğerleme, görüntülerin boyutlarını ölçekleyen bir yöntemdir; bu çalışmada 4 girişli ve 1 çıkışlı MUX ile nasıl basit donanım karşılığı olabileceği kanıtlanmıştır. Görüntü birleştirme ile arka plan ve ön plan görüntüleri bir araya getirilerek yeni birleşik görüntü elde edilir. Bu algoritma için ise bit akışları kullanarak sadece 2 girişli ve 1 çıkışlı MUX yardımıyla nasıl birleşik görüntü elde edilebileceği kanıtlanmıştır. Her iki algoritma da olumsallık tabloları yardımı ile doğrulanır. Buradaki diğer önemli katkı, farklı rasgele sayı üreteç modellerinin (Sobol, doğrusal geri beslemeli kaydırmalı yazmaç ve binom dağılımı), benzetimde neredeyse bire bir taklit edilerek uygulamada doğrulanmasıdır. Özetle, gerçek donanım olmadan, bit akışları yazılımda üretilmeden ve işlenmeden, gerçek donanım varmış ve bit akışları işleniyor gibi sonuçlar elde edilmiştir. Çalışma zamanı ve bellek kullanımı ise önemli ölçüde azaltılmıştır.

Öğrenme sistemlerine stokastik hesaplamanın uyarlanmasına beşinci bölümde yer verilmiştir. Öncelikle, tek stokastik nöron önerilmekte olup, erken öğrenme dönemlerinde iki sınıflı doğrusal sınıflamada gerekirci yaklaşıma göre üstünlüğü vurgulanmaktadır. Gürültülü veri ile nöron eğitimi sırasında erken epoklarda daha yüksek sınıflama başarısı elde edilmiştir. İlgili bölümde çok katmanlı sinir ağı mimarisine ilişkin bir hatırlatma bölümüne de yer verilmiştir. Bu bölümdeki derleme, sinir ağlarına aşina olmak isteyen okuyucular için hazırlanmıştır. İleri yön, geri yayılım ve ağ ağırlıklarının güncellemesiyle öğrenme işleminin nasıl olduğuna ilişkin bir dizi denklem derlenerek geleneksel ağ sistemleri hatırlatılmıştır. Ardından, stokastik hesaplamanın çok katmanlı mimarideki uyarlaması ve genel çerçevesi (stochastic computing-based neural network, SCNN) anlatılmakta, bu sırada literatüre sunulan yeni bir ağ mimarisi verilmektedir: *Bit akış işlemeli ikili sinir ağı* (bitstream processing binarized neural network, BSBNN). Geleneksel ikili sinir ağı (binarized neural network, BNN), ağ ağırlıklarını ve nöron aktivasyonlarını 1 bit ile ifade etmekte; ancak bu durum hafıza hataları gibi yumuşak bit çevirme hatalarına karşı kırılgan bir yapıya neden olmaktadır. Verilerin 1 bit ile ifade edilmesi çarpma ve toplama gibi temel ağ operasyonları açısından son derece verimli bir donanım yapısı sunmasına rağmen, hataya meyilli yeni donanım ve hafıza teknolojilerinde sorun teşkil etmektedir. Geleneksel ikili ağlar çarpma işlemi için XNOR lojik kapılarını, toplama için ise basit sayıcı devrelerini kullanmaktadır. Bu açıdan stokastik hesaplama için kullanılabilecek çarpma ve toplama devre elemanları ile örtüşmektedir. Aynı basit donanım elemanları, bit çevirme hatalarına daha gürbüz olan bit akış işlemeli ağ için kullanılarak daha verimli sonuçların elde edilmesi sağlanır. Literatüre sunulan makalemizde bu avantajın yanı sıra, nöron aktivasyon fonksiyonu için toplayıcı devresinde lojik maskeleme ile ek aktivasyon modülüne gerek olmadan hafif bir mimari önerilmektedir. Geleneksel ikili ağlarda aktivasyon sonucuna, bir eşik değerine göre çıkarma işlemiyle $+1$ veya -1 nöron çıktısı alınarak karar verilir ($+1$ donanımda *lojik-1* ile, -1 ise *lojik-0* ile kaydedilir.). Bizim önerimizde, bipolar kodlamanın gücü ile nöron çıktısına, ön-aktivasyon (pre-activation, S) bit akışındaki *lojik-1* veya *lojik-0*'ların çoğunluk-azınlık dengesi kontrol edilerek karar verilir. Bu kontrol toplama sırasında eş zamanlı yapıldığı için ek bir aktivasyon modülüne gerek olmaz. Böylelikle hem daha az kaynak kullanımı sağlanır (nöron bazında %30 daha az) hem de bit çevirme hatalarına karşı daha verimli bir mimari sunulur. Bit çevirme varlığında ağ eğitimi ile görüntü ve ağ ağırlıklarına enjekte edilen yumuşak hatalar, ilgili hata enjeksiyon değerinin test ağında kullanılmasıyla nasıl gürbüz ağ doğruluğu elde edildiğini kanıtlamıştır. Önerilen mimarinin, geleneksel kırılgan ikili ağa göre üstün sağlamlığı ispatlanmıştır.

Son olarak, farklı sinir ağı mimarilerinin karşılaştırılması ve olumsallık tablosunun bit akış tabanlı sinir ağı benzetimi için oyuna dâhil edilmesi tartışılmıştır. Herhangi bir nicemleme içermeyen, sabit veya kayan virgüllü aritmetik ile, tam hassas sinir ağı da (full-precision neural network, FPNN) dikkate alınarak, dört ağ yapısı; BNN, BSBNN, SCNN ve FPNN kıyaslanmıştır. Ağ giriş görüntülerinde bozulmalar (kontrast, Gauss bulanıklığı, sis, alacalı gürültü, yakınlaştırma bulanıklığı vb.) söz konusu olduğunda hataya duyarlı ağ eğitimi, duyarlı olmayana göre kıyaslanarak verilmiştir. Görüntü sınıflama doğrulukları açısından, önerilen bit akış işlemeli ikili sinir ağı mimarisinin kıyaslanabilir performansı sunulmuş; ikili ağlarda (BNN ve BSBNN) hataya duyarlı eğitimin önemi gösterilmiştir. Son bölümde ise olumsallık tablosu ile ağ eğitimi sırasında stokastik bit akış tabanlı XNOR çarpmasının benzetimi

sunulmuştur. Evrişimsel sinir ağı (convolutional neural network, CNN) mimarisinin sınıflayıcı kısmındaki çarpma işlemlerinde kullanılan XNOR öykünmesi, olumsallık tabloları ile sağlanır. Gerçek bit akış işlemeli modele göre hızlı ağ eğitimi sunulur. Fiziksel bit akışlarını kullanan yazılım benzetimi, akış uzunluğuna bağlı üstel olarak artan bir çalışma zamanı gösterirken, olumsallık tablosu varlığında ağ eğitim süresi bit akış uzunluğundan bağımsız; doğrusaldır.

1. INTRODUCTION

Stochastic computing (SC) is a next-generation computing paradigm that gained attention starting in the 1960s [1]. Current efforts on data processing techniques are geared toward hardware-friendly solutions in emerging platforms, and unconventional calculations such as probabilistic computing are required. Especially with the new era in machine learning systems, the devices for learning purposes, named learning machines, have become significant as von Neumann emphasized the importance of learning devices in his paper about probabilistic logics [2]. SC treats any scalar value with bitstream probability describing an unconventional environment. In conventional computing, 1-bit inputs are applied to the logic circuits, and the output is obtained. Instead of a single bit, an N-bit stream is used to represent value in the stochastic scenario by generating a binary stream with randomness. In Figure 1.1, one of the representations of a random bitstream is exemplified. The position of *logic-1* and *logic-0* is random and can occur in different bit positions. The relationship between the bitstream size, N, and the total number of *logic-1*s, $\sum 1s$, is used to denote a probability value, $\frac{\sum 1s}{N}$. The extended background on encoding details is given in Section 2.1.

$$(00010011)_{\mathrm{SC}} = \frac{\sum 1s}{N} = \left(\frac{3}{8}\right)_{10}$$

Figure 1.1 : Bitstream example for $N = 8$-bit size.

Using the power of *logic-1* and *logic-0* only, generating probability values, and using encoding techniques, complex arithmetic operations are reduced to simple logic circuits; thereby, lightweight systems emerge. However, since these systems are stochastic, operations are performed within the scope of approximate calculation, and the interpretation of accuracy on the application side is critical. Adapted to vision and learning machines, SC will be evaluated from hardware design and machine learning perspectives in this thesis. Although hardware-oriented solutions are presented in the literature, a perspective from computer science with software-driven aspects, such as optimization, simulation, hardware approximation, robust neural network (NN) issues,

various dataset performances, etc., will be given in this dissertation. In that respect, an interdisciplinary study is presented, which melts electronics and computer engineering into a single pot.

1.1 Literature Review

In this section, an overview of the literature will be presented. An extended literature survey underscoring vision and learning machines will be given in Section 2.6.

SC is an unconventional computation technique that recently appeared in research papers in image processing and machine learning. Number representation is based on the stochastic bitstreams, and the hardware design is handled considering the probabilistic logic operations. In their paper on sequential logic-based SC, Najafi et al. explain the motivation behind the overall SC systems as follows: "*Computations based on stochastic bitstreams have several advantages compared to deterministic binary radix computations, including low power consumption, low hardware cost, high fault tolerance, and skew tolerance.*" [3] Overall, from the different efforts in the literature, the following key points can be listed as the advantages of this re-emerging computation:

i.) Less-area footprint and low-power consumption

ii.) Noise resilience and robustness to faults (or soft errors)

iii.) Massive parallelism and scalability

Future technologies are expected to be in lower power consumption and less area. The mitigated size can be possible either using nanoscale devices or by making some truncation or approximation on the current devices. In both cases, stochastic behavior may occur, and the robust modeling of these devices becomes more crucial. In the nanoscale, particles behave in a probabilistic fashion, and stochastic modeling may be an issue. In approximated designs, stochastic algorithms may be required to validate circuit behavior. A possible solution, SC, is a popular approach that presents a less-area footprint and low-power consumption as an approximate computing technique.

Many SC-based papers have a strong motivation for efficient power consumption and area. Numerically speaking, the following efforts in the literature highlight the importance of SC for this motivation. Sato et al. showed the area and power usage

for their neurochip designed using stochastic logic. They reported that 5 times less area consumption was achieved compared to the closest competitor in the market, and a 10-fold advantage was provided for power [4]. In addition to the power, Seva et al. explained the energy-efficient SC approach together with the resource utilization in field-programmable gate array (FPGA) for multiplication and image processing applications. Multiplication and edge detection operations were handled by utilizing the quasi-stochastic number generators (QSNGs) and parallel stochastic bitstream processing. Low-discrepancy (LD) sequences and pseudo-random sequences were used. The average power consumption for the LD-based multiplication was 0.3 μW, and for the edge detection, it was 0.12 μW. The average power consumption of the pseudo-random sequence processing was 1.34 and 0.41 μW, respectively, in the application of multiplication and edge detection. Moreover, for the resource utilization, the occupied slices in FPGA were 70 and 140 for LD-based multiplication and edge detection, and 17 and 48 for pseudo-random bitstream-based multiplication and edge detection, respectively [5]. Li and Lilja proved that kernel density estimation-based (KDE) image segmentation was performed with 1.041 times higher energy consumption with conventional implementation and 3.125 times higher energy consumption with the triple modular redundancy (TMR) implementation compared to SC. They paraphrased that not only less hardware and energy-power consumption but also higher tolerance to the errors were experienced for SC implementation [6]. There are also different applications of SC, like power electronics. Li et al. implemented a stochastic artificial neural network (ANN) on the FPGA device to be used in power electronic applications on wind turbine systems. FPGA resource utilization was 75% for the stochastic method; however, 98% resource utilization was monitored for the traditional look-up table-based ANN [7]. Onizawa et al. designed the parallel stochastic Gabor filters with a treatment for latency issues that could occur in SC systems in the presence of longer streams. As their application-specific integrated circuit (ASIC) design for the Gabor filter was compared with coordinate rotation digital computer (CORDIC), 68 parallel filters in their own design outperformed in terms of delay and the number of gates assigned. In their 0.13 μm complementary metal-oxide semiconductor (CMOS) design, a 194 ns/pixel delay was recorded, while CORDIC had a 196 ns/pixel delay. The numbers of gates in use were approximately 30 times

less in their design compared to CORDIC [8]. Fick et al. proved how efficient area consumption could be obtained to design image processing operations (edge detection, noise filtering) by using mixed-signal stochastic computing (MSSC). When compared to the conventional digital synthesis implementation, 2.75 times lower power and 5 times lower area were obtained [9].

Another important key point of SC is the built-in noise robustness. Built-in probabilistic behavior brings an ability to resist the noise, and stochastic systems can compensate for the noise. The noise issue is discussed in the image processing-related papers [10, 11] and in the image compression-related papers [12, 13]. Canals et al. focused on the noise-tolerant probabilistic logic devices by exemplifying the electromagnetic noise immunity in SC. Their proposal for statistical pattern recognition applications employed extended stochastic logic (ESL) encoding, and it was shown that the SC environment still survives with the presence of 90% noise with an approximate error of 10% for 24-bit architectures [14]. As a different noise example, Alaghi et al. defined the input noise as a Gaussian random variable and added it to the voltage signal produced by the sensor. SC-based edge detectors could perform efficiently even though a 20% noise injection was applied. It was underlined that noise impact appears as a gray background in the images when SC designs were utilized [10].

In addition to noise, fault tolerance (or soft error robustness) is the other appealing advantage of SC. Fault tolerance in stochastic modeling is crucial for improving the accuracy of an application. Qian et al. created an architecture for fault-tolerant computation with stochastic logic [15], and their effort has been state-of-the-art in the SC community since 2011. They applied an SC-based design for image processing targeting the gamma correction function. Considering a one-time fault not occurring repeatedly, a soft error is generally treated as a bit-flip in SC (an error on a bit, flipping like $1 \rightarrow 0$ or $0 \rightarrow 1$) [16–18]. Qian et al. in [15] underlined and reported that for 15% soft error injection as bit-flips, SC-based design outperformed deterministic one. Before giving the encoding details in Section 2.1, assume that the total number of *logic-1*s is exactly equal to the scalar value to be encoded as an SC-based bitstream. The case is different than conventional computing, where base-2 representation causes severe issues due to the bit significance, such as the most significant bit (MSB) and

the least significant bit (LSB). Thus, a 1-bit error may cause a considerable deviation of the scalar value represented when the bit-flip is closer to the MSB. However, in SC, the scalar changes by only $+1/-1$ for single-bit errors, regardless of the bit position. Li et al. created soft errors manually via bit-flips for noise modeling of transmission gates, which are non-restoring devices. Regarding noise robustness, they tested bitstream-based quantized neural network (QNN) applications, with a certain percentage flipping the bits seen at the output. The recognition error rate in the network was recorded as less than 5% for injected errors [19]. The simulation platform gains importance for bulk tests addressing the soft error in NNs. Preliminary noise and fault performance can be measured in the simulations without fully-designed hardware. In this respect, the necessity for a memory and latency-aware simulation environment is obvious, and it constitutes one of the motivations of this doctoral thesis. A pioneer simulation environment with an experimental setup was presented in the literature by Canals et al. They gave the implementation details of the stochastic computing-based neural network (SCNN) application in their paper [20]. They explained block-by-block hardware elements, including the random pulse generator, pulse-to-binary converter, stochastic arithmetic elements, and activation function. They measured the noise performance of the proposed system to show its noise tolerance limit. The noise immunity of the learning system for injected bit-flips was high [20].

In the design of SC systems, using small gates and systems instead of complex arithmetic units enables the designer to make the computations in parallel by applying gate arrays. Chippa et al. indicated several limitations related to SC, such as stream length versus precision trade-off; therefore, they used two-dimensional arrays of stochastic processing elements (StoPE) for parallelization. Their stochastic recognition and mining processor (StoRM) in [21] supplied up to triple the energy-delay improvements compared to the conventional approaches. On the other hand, Zhang et al. proposed a parallel hybrid stochastic-binary NN system, providing 255 and 49 times faster runtime compared to its pseudo-random and deterministic-based counterparts, respectively [22]. For parallel computation in [23], Sim et al. presented the SC convolutional neural network (CNN) accelerator together with the highly optimized bit-parallel SC multiply and accumulate (SC-MAC) module. With this network design, the speed tripled compared to the fastest NN accelerator they compared.

Xie et al. reported that the binary radix-based NN design (with two hidden layers, 784-100-200-10) needs a 6-stage pipeline, while the fully-parallel SC-based deep neural network (DNN) with the same size can operate at a clock frequency of 450 MHz without the pipeline [24].

A few previous studies proposed SC circuit synthesis and simulation in software. The scsynth [25], as an open source synthesis tool, generates Verilog design of architectures based on stochastic logic, considering the spectral transform use in stochastic circuit synthesis (STRAUSS) [26]. In [27–29], bitstream synthesizer and designer (BitSAD) is a domain-specific language for bitstream processing systems performing complex algorithms in limited-resource environments. BitSAD presents the rapid generation of synthesizable bitstream computing circuits; however, it still utilizes actual bitstreams in software. This dissertation differentiates at this point by using scalar-only values while still generating the stochastic logic result in the software without bitwise operations and being able to control correlation, random fluctuations, and random source generators. The model of random sources, such as linear-feedback shift registers (LFSR), Sobol LD sequences, and binomial distribution, is proposed by targeting SC's built-in random fluctuations.

1.2 Purpose of Thesis

This doctoral thesis serves several purposes. The first (in Chapter 2) is to create a background on SC for the reader, as well as provide an accumulator optimization for known SC topology. A cascaded adder for multiple inputs is an available structure set up via cascaded multiplexers (MUXs). However, it is unclear which input width for each stage's MUX should be selected. Hence, it is aimed to construct an architecture with a rule-based algorithm for optimization. The background section hierarchically defines all concepts, such as bitstream, encoding, decoding, correlation, and non-idealities, in order to familiarize reader with rudimentary concepts.

The second aim (in Chapter 3) is to simulate a hardware-based system in a software environment. A digital design must be validated before it is physically implemented. This situation becomes complicated, especially in the digital design of complex applications, such as NNs, and software-hardware co-design is needed. Although bitwise logic operations on the software platform seem to be sufficient for the software

simulation of hardware design, SC's dependency on the long bitstreams causes a problem that needs to be solved in terms of memory and runtime constraints. For this reason, an exploration of a simulation environment that meets the SC properties is required. Specifically, this thesis focuses on SC properties such as correlation settings, random fluctuation errors, and random source models.

The other aim (in Chapter 4) is to interpret well-known facts during SC's adaptation to image processing. It is shown how the digital design of basic box filter structure can be made lightweight under SC. The noise robustness is given attributing to the previous studies. The innovation presented at this point concerns the performance of comparative digital designs at different levels of approximation. The purpose is to underscore that the image acquisition noise issue is better solved with simplified digital architectures in SC. In addition, the goal is to extend previous SC-based vision algorithms; therefore, template matching, bilinear interpolation, and image compositing as brand new applications are targeted.

In terms of learning systems (in Chapter 5), it is aimed to eliminate the deficiencies of traditional NNs considering the hardware-aware concepts (bit-flips, early training, activation function simplification, etc.). To tackle this, SC-based enhancements are targeted. An SC system is created to provide early learning in the linear classification in premature epochs. Another aim of this thesis is to explain the basic learning steps from a single neuron to multi-layered architecture. By reading the fifth chapter, the reader will thoroughly understand how a traditional NN learns. Understanding the NN equations presented in the literature may be tedious even though there are many resources related to conventional learning systems; nevertheless, the learning machines chapter clarifies all the concepts systematically. In addition, SC superiority is aimed against some faults that may occur in NNs, and an update to rival architectures (binarized neural networks - BNNs) is desired. Thus, both robust (in terms of accuracy) and efficient (in terms of hardware) results are targeted. Furthermore, in terms of learning systems, it is aimed to measure the effect of different training scenarios on network accuracy. Additionally, the impact of noise and errors in the presence of various image corruptions and soft errors is measured.

1.3 Contributions and Hypothesis

Three main perspectives motivate the hypothesis of this dissertation. These fundamental contributions differ from the prior art:

(1) A software-driven SC framework is proposed for runtime- and memory-efficient simulations with a high ability to emulate actual hardware.

(2) New vision approaches considering SC-based image processing design are presented. New perspectives are proposed related to bitstream processing vision machines for mean filtering, edge detection, template matching, bilinear interpolation, and image compositing.

(3) A new hardware-aware NN design is proposed with a lightweight and soft-error-robust solution. A bitstream processing network architecture without an extra activation function module is designed. A hardware utilization gain of 30% is achieved for a single neuron.

In addition to these major contributions, the first part of the thesis conducts a literature summary (Section 1.1) and a comprehensive survey (Section 2.6) by listing last decade's dissertations and research papers on vision and learning machines. By reading the related sections, a prospective reader can benefit from understanding the recent trends in bitstream processing systems. Moreover, the cascade MUX tree is optimized for multi-operand accumulation, and a rule-based algorithm is proposed for the multi-stage scaled adder, thereby obtaining less resource utilization and logic delay. Regarding contribution (1), the proposed CT simulation is unveiled for random source emulation and random fluctuation error. This work also contributes to the simulation of the random number generators based on the binomial distribution, Sobol sequences, and LFSRs. Extending contribution (2), CT validation is shown by performing experiments related to bilinear interpolation and image compositing. Thus, the hardware model of the vision machine systems is perfectly imitated in software with a fast and memory-aware solution. Besides the main contribution in (3), a single neuron design is presented, and the outperformance of bitstream processing neuron in two-class noisy data classification for early epochs is proven. Furthermore, different

NN architectures are investigated considering the soft errors and data corruptions. Hence, the proposed hardware-aware NN architecture is underscored, which is referred to as a bitstream processing binarized neural network.

After all major and minor contributions above, the hypothesis can be summarized as *"creating a runtime- and memory-aware SC simulation framework, developing new SC-based image processing algorithms, and blending the lightweight binarized NN architecture with bitstream processing paradigm for an efficient and robust design."*

2. BITSTREAM COMPUTING

Fundamental terminology, as well as basic notations and definitions, are presented in this background chapter. In addition, optimization and interpretation of some known topologies, such as arithmetic operations, are presented.

Generally, a system related to SC has three basic sub-parts: inputs, the system (maybe as a black-box), and the output. The first is related to the bitstreams applied to the system input. Correlated and uncorrelated bitstreams affect the performance (operation accuracy) directly. The probabilistic bitstreams can be encoded in different ways. Encoding scheme, correlation, and bitstream size are essential parameters for the SC-based system input. The second sub-part of SC architectures is the system itself. The bottom-level logic gates (AND, XNOR, OR, NOT, etc.) and the well-known top-level digital systems (*combinational* or *sequential*) are considered in the frame of SC, but the inputs are no longer a single bit [30]. As the final part of the SC system, the bit-processed output is more than bitwise-only results and is considered in the frame of arithmetic operations. Thus, instead of complex arithmetic circuits, a low-area and power-effective digital design is gathered, which is also robust to noise and prone to being processed in parallel. Nevertheless, since the operations are in the probabilistic domain, the obtained output may differ from the expected results. Optimizing the inaccurate results should be considered at the application level more than the hardware level itself.

In the next section, the first part of the SC system, bitstream, is defined initially, including encoding techniques.

2.1 Bitstreams

In SC systems, numerical values are represented in a bitstream. The bitstream is denoted in $\boldsymbol{bold}$, while the *italic* fonts are used to indicate scalar numbers. Let us first define the bitstream as follows.

Definition 2.1 (Bitstream)**.** $\exists X$ such that $-\infty < (X \in \mathbb{Q}) < +\infty$ can be defined in a bitstream $\boldsymbol{X}$, which is composed of N logic elements, i.e., *logic-1* (*VCC*, *High*) and *logic-0* (*GND*, *Low*). The bitstream representation is composed of N binary elements $\boldsymbol{X} = \{x_1, \ldots, x_N\}$ where $(N \geq 2) \in \mathbb{Z}^+$, defining the *bitstream size*.

In Definition 2.1, $\exists X$ is intentionally chosen instead of $\forall X$. This is because "*not every*" but "*some of the*" X scalars can be represented in $-\infty < X < +\infty$. First of all, the large enough bitstream size restricts the implementable range to $-N \leq X \leq N$. In theory, N can approximate to infinity; however, it has a limit based on the processing environment in real applications. Therefore, in practice, N is taken as 2^n, where $n \in \{1, 2, 3, \ldots\}$. According to literature, the greatest previously experienced stream size is no more than $N = 32768$ ($n = 15$) [31].

On the other hand, $X \in \mathbb{Q}$ as defined in Definition 2.1 is further processed to obey $(X \to \hat{X} \in \mathbb{Z}) \in \mathbb{Q}$. The implementable range is $-N \leq (\hat{X} \in \mathbb{Z}) \leq +N$. If the scalars fall in a range $[-N, N]$, the rounding to the nearest integer operation ($\lfloor \ \rceil$) is applied to have $\lfloor X \in \mathbb{Q} \rceil \to \hat{X} \in \mathbb{Z}$. However, if the scalars are lying in the $[-1, 1]$ range, $-1 \leq (X \in \mathbb{Q}) \leq +1$, then, the up-scaling by N is utilized first, such a way that $N \times (-1 \leq (X \in \mathbb{Q}) \leq +1)$ yields $-N \leq \lfloor N \times (X \in \mathbb{Q}) \rceil \leq +N \to -N \leq \hat{X} \in \mathbb{Z} \leq +N$. These explanations are given so one can establish the relationship between the *logic-1* count in a bitstream and the value of a scalar.

Furthermore, the notation of bitstream elements in a stream is required to be defined. In a bitstream $\boldsymbol{X}$, any m^{th} element out of N is binary *1* (*logic-1*) or *0* (*logic-0*). In Definition 2.2, notations for a bitstream element are defined.

Definition 2.2 (Notations of a Bitstream Element)**.** For a single bitstream, any m^{th} element may be denoted as one of the notations: $\boldsymbol{X}(m) \in \mathbb{B}$, $X_m^{(b)} \in \mathbb{B}$, or $x_m \in \mathbb{B}$.

After defining the bitstream and its elements, the *bitstream probability* should be defined before giving the *encoding* and *decoding* operations. Across any bitstream, the **T**otal **C**ount of *logic-1*s denoted as $TC = \sum_{m=1}^{N} x_m$ is linked to a probability value as determined in Definition 2.3. Thus, a relation is now set between *logic-1* count (TC) and N.

Definition 2.3 (Bitstream Probability)**.** X is defined in a bitstream represented as $\boldsymbol{X}$ where the probability of $\boldsymbol{X}$ is $P(\boldsymbol{X})$ (sometimes denoted as $P_{\boldsymbol{X}}$, or P_X skipping the bold notation) indicates how likely the *logic-1* can be obtained from the N-element bitstream. The probability is given as:

$$P(\boldsymbol{X}) = \frac{\sum_{m=1}^{N} x_m}{N} = \frac{TC}{N}. \tag{2.1}$$

The bitstream representation in stochastic devices can be treated in several ways [32]. In bitstream processing SC systems, the numbers are represented by a binary encoding scheme. Based on whether $X \leq 0$ or $X \geq 0$, polarity affects the binary encoding. Two encoding schemes are utilized: Unipolar Encoding (UPE) and Bipolar Encoding (BPE). For $X \leq 0$, BPE is used for encoding the scalar into a bitstream, while for $X \geq 0$, both schemes can be employed. UPE and BPE are generally considered well-known encoding schemes; however, there are some alternatives like inverted bipolar encoding proposed by Alaghi [33], likelihood ratio (LR), and log likelihood ratio (LLR) stochastic number representations [34].

2.1.1 Unipolar encoding (UPE) scheme

When there are only positive scalars and 0 in the set of values to be represented, UPE can be utilized. Especially in image processing applications, UPE is frequently used since grayscale or red-green-blue (RGB) image pixel values always fall into the range $0 \leq X \leq (2^{\kappa-bit}-1)$. $(2^{\kappa-bit}-1)$ is the maximum pixel value as $\kappa \in \mathbb{N}^{+}$, and $2^{\kappa-bit} = N$ is applied.

In Definition 2.4 and Definition 2.5, encoding and decoding of the unipolar representation are given.

Definition 2.4 (Unipolar Encoding)**.** $\forall X$ s.t. $0 \leq (X \in \mathbb{Z}) \leq +N$ or $0 \leq (X \in \mathbb{Q}) \leq +1$, UPE-based bitstream $\boldsymbol{X}$ is represented either via $P(\boldsymbol{X}) = \frac{(X \in \mathbb{Z})}{N} = \frac{TC}{N}$ or $P(\boldsymbol{X}) = \frac{\lfloor (N \times X \in \mathbb{Q}) \rceil}{N} = \frac{TC}{N}$.

Definition 2.5 (Unipolar Decoding)**.** $\forall \boldsymbol{X}$ as UPE-based bitstream is decoded either via $\sum_{m=1}^{N} x_m = X \in \mathbb{Z}$ or $\frac{\sum_{m=1}^{N} x_m}{N} \approx X \in \mathbb{Q}$.

2.1.2 Bipolar encoding (BPE) scheme

In the BPE scheme, the signed number representation is an essential factor. Negative scalars require the sign information throughout the stream, which should be treated by the encoding scheme. BPE scheme perfectly handles scalars obeying $X \leq 0$. Bitstreams representing negative numbers have *logic-0*s in the majority, while for the positive numbers, bitstreams have *logic-0*s in the minority. This majority-minority will be exploited in Chapter 5 during the design of bitstream processing binarized neural networks (BSBNNs). Overall, the importance and need for the negative numbers in the binary streams emerge, especially in the NN design [35,36], since some of the network weight values may be negative scalars during the NN training. In Definition 2.6 and Definition 2.7, encoding and decoding of the bipolar representation are given.

Definition 2.6 (Bipolar Encoding)**.** $\forall X$ s.t. $-N \leq (X \in \mathbb{Z}) \leq +N$ or $-1 \leq (X \in \mathbb{Q}) \leq +1$, BPE-based bitstream $\boldsymbol{X}$ is represented either via $P(\boldsymbol{X}) = \frac{(\frac{(X\in\mathbb{Z})+N}{2})}{N}$ or $P(\boldsymbol{X}) = \frac{\lfloor(\frac{(N\times X\in\mathbb{Q})+N}{2})\rceil}{N}$. For the former probability, if $(X \in \mathbb{Z})+N$ is odd, $(X+N) \in \{2n+1 : n \in \mathbb{Z}\}$, floor $\frac{\lfloor(\frac{(X\in\mathbb{Z})+N}{2})\rfloor}{N}$ or ceiling $\frac{\lceil(\frac{(X\in\mathbb{Z})+N}{2})\rceil}{N}$ functions may be required.

Definition 2.7 (Bipolar Decoding)**.** $\forall \boldsymbol{X}$ BPE-based bitstream is decoded either via $\left[\left(\sum_{m=1}^{N} x_m\right) - \left(N - \left(\sum_{m=1}^{N} x_m\right)\right)\right] = X \in \mathbb{Z}$ or $\left[\frac{\left(\sum_{m=1}^{N} x_m\right) - \left(N - \left(\sum_{m=1}^{N} x_m\right)\right)}{N}\right] \approx X \in \mathbb{Q}$.

The relation between UPE and BPE can be denoted using the probability value. In UPE, the bitstream probability is directly utilized for decoding as $P(\mathbf{X}) = \frac{\sum_{m=1}^{N} x_m}{N}$. To show the similar interpretation in BPE, the expression $\left[\left(\sum_{m=1}^{N} x_m\right) - \left(N - \left(\sum_{m=1}^{N} x_m\right)\right)\right]/N$ from Definition 2.7 is rewritten in equation (2.2).

$$\frac{\left(\sum_{m=1}^{N} x_m\right) - \left(N - \left(\sum_{m=1}^{N} x_m\right)\right)}{N} = \frac{2\times\sum_{m=1}^{N} x_m - N}{N} = 2\times\frac{\sum_{m=1}^{N} x_m}{N} - 1 = 2\times P(\boldsymbol{X}) - 1 \tag{2.2}$$

Thus, in BPE-based decoding, the probability value can be related via the $2\times P(\boldsymbol{X}) - 1$ expression. The connection between UPE and BPE is explained with this notation [35]. The following lemma is noted down:

Lemma 2.1 (UPE-BPE Relation)**.** *Encoded scalar directly relates to $P(\boldsymbol{X})$ in UPE, and in BPE, it is noted as $2\times P(\boldsymbol{X}) - 1$.*

In Table 2.1, the example bitstreams for $N = 8$ are given. For each UPE and BPE scheme, the decoded values are indicated using fractions and decimals. Moreover, Figure 2.1 presents decoding examples and BPE-related sign encoding.

Table 2.1 : Number representations using UPE and BPE ($N = 8$-bit).

Example Bitstream*	UPE		BPE	
00000000	$0/8 =$	0	$-8/8 =$	-1
00000001	$1/8 =$	0.125	$-6/8 =$	-0.75
00000011	$2/8 =$	0.25	$-4/8 =$	-0.5
00000111	$3/8 =$	0.375	$-2/8 =$	-0.25
00001111	$4/8 =$	0.5	$0/8 =$	0
00011111	$5/8 =$	0.625	$2/8 =$	0.25
00111111	$6/8 =$	0.75	$4/8 =$	0.5
01111111	$7/8 =$	0.875	$6/8 =$	0.75
11111111	$8/8 =$	1	$8/8 =$	1

*The occurrence of 1s and 0s is expected to be random.

$$(00010011)_{SC} = \frac{\sum 1s}{N} = (\frac{3}{8})_{10} \quad \text{SC UPE}$$

$$(00010011)_{SC} = \frac{\sum 1s - \sum 0s}{N} = (-\frac{2}{8})_{10} \quad \text{SC BPE}$$

BPE encodes the negative sign using the majority of zeros.

Figure 2.1 : Decoding examples of UPE and BPE.

2.2 Bitstream Correlation and Random Process

The correlation is a critical concept affecting operation accuracy in SC [37]. Let us first visit the most frequent arithmetic example in the literature, the AND gate, used as a multiplier of UPE-based bitstreams. The accurate multiplication is achieved when independent and uncorrelated bitstreams are generated [38, 39]. Besides, different operations may be obtained at different correlation levels between input operands. For example, if the bitstreams are positively correlated, the AND gate returns the probability of the smallest bitstream from inputs. If the bitstreams are negatively correlated, then the probability value of the output bitstream by the AND gate is $max(0,\ P_{\boldsymbol{X1}} + P_{\boldsymbol{X2}} - 1)$, where ***X1*** and ***X2*** are the UPE-based input bitstreams for AND gate [32, 40]. Therefore, the definition of correlation is significant for SC, and the randomization procedure should be defined accordingly. These topics are detailed in the following subsections.

2.2.1 Cross-correlation

The cross-correlation between bitstreams is concerned with the logical states of the bits that overlap in the two streams. Maximally or minimally similar bitstreams may occur, but there is one more important condition between these extremes. This exactly-in-the-middle case is called *zero correlation* and refers to bitstreams independent of each other (uncorrelated bitstreams) [40, 41]. A metric that measures cross-correlation has been proposed before, and a corresponding hardware design has been presented [33, 38, 42]. In equation (2.3), this metric, named *stochastic cross-correlation* (*SCC*) (or stochastic computing correlation), is presented. *SCC* is computed with the aid of the AND operation and is denoted as $P_{X1 \wedge X2}$, where P_{X1} and P_{X2} are the probabilities of two stochastic inputs. $P_{X1 \wedge X2} - P_{X1}P_{X2}$ denotes the *difference* between the obtained AND output probability and the expected result [40]. The value range of *SCC* is $[-1, 1]$. *SCC* returns values approaching 0 in case of uncorrelated bitstreams while returning values converging to 1 for maximum correlation and converging to -1 for minimum correlation.

$$SCC(\boldsymbol{X1}, \boldsymbol{X2}) = \begin{cases} \frac{P_{X1 \wedge X2} - P_{X1}P_{X2}}{min(P_{X1},\, P_{X2}) - P_{X1}P_{X2}} & , \; if \; P_{X1 \wedge X2} > P_{X1}P_{X2} \\ \frac{P_{X1 \wedge X2} - P_{X1}P_{X2}}{P_{X1}P_{X2} - max(P_{X1} + P_{X2} - 1,\, 0)} & , \; else \end{cases} \tag{2.3}$$

Examples of three distinct correlations using the AND operation are presented below.

Example 2.1. Examples for the correlation of bitstreams. (i) An example showing maximum correlation; the maximum overlaps of *logic-1*s. (ii) An example showing minimum correlation; the minimum overlaps of *logic-1*s. (iii) Uncorrelated example.

(i)	(ii)	(iii)
$\boldsymbol{X1} = 11000000$	$\boldsymbol{X1} = 11000000$	$\boldsymbol{X1} = 10000010$
$\boldsymbol{X2} = 11110000$	$\boldsymbol{X2} = 00001111$	$\boldsymbol{X2} = 10101001$
AND 11000000	AND 00000000	AND 10000000

For the *SCC* metric, the first case (i) gives $SCC = 1$, (ii) has $SCC = -1$, and the uncorrelated example, (iii), has $SCC = 0$.

On the other hand, when probability values of the two operands in Example 2.1, $P_{\boldsymbol{X1}} = 2/8$ and $P_{\boldsymbol{X2}} = 4/8$, are considered, AND results should be interpreted. Regarding case (i) in Example 2.1, the minimum of $P_{\boldsymbol{X1}}$ or $P_{\boldsymbol{X2}}$ is seen at the output ($min(P_{\boldsymbol{X1}} = 2/8,\, P_{\boldsymbol{X2}} = 4/8) \rightarrow 2/8$). That positive correlation presents $min(P_{\boldsymbol{X1}},\, P_{\boldsymbol{X2}})$

operation. In case (ii), i.e., minimum (or negative) correlation, the AND output probability is 0, and it is formulated via $max(0,\ P_{\boldsymbol{X2}} + P_{\boldsymbol{X2}} - 1)$ [40]. Most importantly, case (iii) exemplifies uncorrelated $\boldsymbol{X1}$ and $\boldsymbol{X2}$ bitstreams. The AND gate works as a multiplier, and the output probability is $1/8$ ($P_{\boldsymbol{X1}} = 2/8 \times P_{\boldsymbol{X2}} = 4/8 \rightarrow 1/8$).

Regarding the recent efforts for the correlation in SC, the latest trend is on correlation manipulation, which is required in the transitions of cascade-connected SC modules. If an output bitstream is an input for the subsequent module, in-stream manipulation will be required when there is no bitstream generation and correlation tuning between the two modules. Sina et al. [43] try to manipulate the correlation by directly interfering with the bitstreams. Random number generators (RNGs) are not always preferred between modules because it requires binary conversion and an extra cost. Sina et al., in the LD sequence-based system [44–46], offer cost-effective stream manipulation by processing bitstreams within the scope of Sobol sequences. Thus, random number generation becomes independent of scalars. Instead, bitstreams are manipulated in-stream without regenerating random numbers, thereby being correlated or uncorrelated. The case studies like sorting and median filtering are presented in the paper. They also show the efficient division example. Wu and Miguel discuss another bitstream manipulation and division operation example in [47].

2.2.2 Bitstream randomization

In the literature, the concept of "randomization" for stochastic bitstream processing, which can also be encountered as a random bitstream or random pulse processing [48–50], is explained in this section.

The power of SC stems from the built-in probabilistic behavior coming from the random bitstreams. Noise and fault robustness are two of the most acclaimed rewards from SC utilization in vision and learning machines. The *logic-1*s and *logic-0*s occur randomly in a bitstream obeying the bitstream probability, $P_{\boldsymbol{X}}$.

Within the scope of this work, three methods will be discussed: LFSR random source, Sobol LD sequence, and the binomial distribution. The first, LFSR, is the most frequently used RNG in the literature [51–55]. LFSR provides pseudo-randomness with binary numbers that are generated in a circular manner. That is, the generated numbers observed during a cycle will be obtained in the next cycle in the same order. In

Figure 2.2, an example is depicted for 3-bit LFSR. It is shown how *initial seed* and *taps* are utilized, together with the cycle of $2^3 - 1$ states. Next, Sobol sequences are the ones most recently proposed in the literature, especially for the high-accurate SC arithmetic. Considering the design of efficient Sobol generators in the literature [44,45,56], Sobol LD sequences have become a game-changer. Figure 2.2 illustrates MATLAB-based first two Sobol sequences (the first seven elements).

Finally, the third approach, which is easy to implement in software like Sobol LD, is presented. Stochastic bitstream probability, P, can be thought of based on a set of samples from a random variable (RV) having a Bernoulli distribution with P success probability [33, 57]. The Bernoulli distribution is employed to produce the bitstream having independent bits [12, 58]. Using N different Bernoulli trials, the binomial distribution is obtained. The probability formula is a single function, and one can easily simulate it in any software environment. Kuehnel explains how to achieve the binomial distribution from random numbers, which are independent and uniformly distributed in the $[0, 1]$ range [59]. In Figure 2.2, each random source method shown has the "comparison" operation. For instance, LFSR is coupled with a comparator circuit to compare the randomly generated number and binary scalar, X, to be encoded in a bitstream [60, 61]. When a random number is generated, the scalar value is compared with it. If the scalar is greater than the random number, then the corresponding bit is *logic-1* in a bitstream; otherwise, it is *logic-0*.

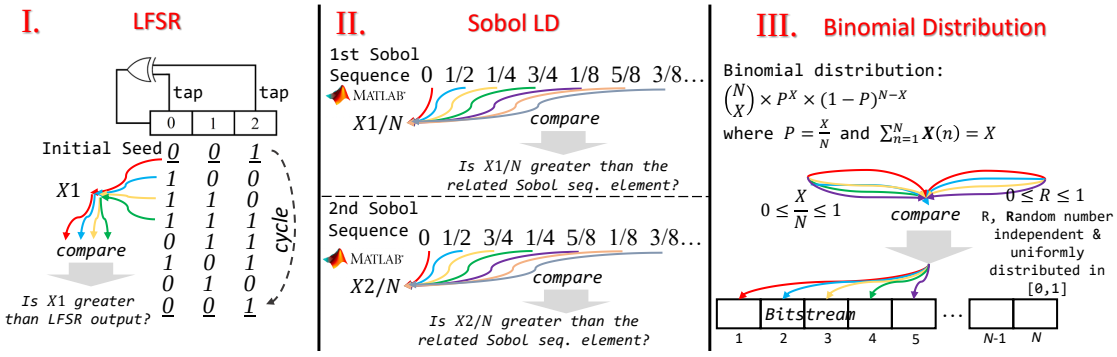


Figure 2.2 : LFSR, Sobol LD, and binomial distribution for random bitstream generation.

2.3 Stochastic Computing (SC) Primitives

SC primitives are simple gates and some well-known logic systems. NOT, AND, OR, XOR, NAND, NOR, and XNOR are deterministic logic gates used in SC. Some combinational and sequential logic circuits are also utilized in SC architectures. SC primitives are sourced with UPE- or BPE-based "input" bitstreams ($\boldsymbol{X}$), and the "output" bitstream ($\boldsymbol{Y}$) is obtained. The relation between input and output represents some arithmetic operations based on the bitstream probabilities. Any logic gate (except NOT gate) or a system can have multiple inputs as indicated in Definition 2.8.

Definition 2.8 (Multi-Input Logic Gate/System)**.** Let G be a logic gate or any cascaded logic system. There are β input terminals, and each can be sourced with a synchronized bitstream from $\{\boldsymbol{X1}, \boldsymbol{X2}, ..., \boldsymbol{X\beta}\}$ set of inputs.

Based on the β size, the logic system may require optimization, especially for the bulk arithmetic operations. In SC, 4-arithmetic operations are performed using simple circuitry such as AND, XNOR, MUX, and JK flip-flop. Let us first give the formal definition of AND gate multiplier by first defining the NOT logic in SC via Lemma 2.2 [62].

• Multiplication

Lemma 2.2 (NOT Logic in SC)**.** *Let G be a NOT gate having $\beta = 1$ input terminal. From the bitstream $\boldsymbol{X1}$ applied to input having $P(\boldsymbol{X1})$ probability value, the output probability $P(\boldsymbol{Y}) = 1 - P(\boldsymbol{X1})$ is obtained.*

Proof. For any m^{th} element, $P(\boldsymbol{Y}) = P(\boldsymbol{Y}(m) = 1) = P(\neg\boldsymbol{X1}(m) = 1) = P(\boldsymbol{X1}(m) = 0) = 1 - P(\boldsymbol{X1})$. □

NOT logic in SC is free from the encoding type. However, other logic gates differ from UPE to BPE in terms of the SC expression. [32] presents deep research on encoding-based analyses, which is highly-recommended for the reader of this dissertation.

With respect to the following Theorem 2.1, Corollary 2.1.1 and Corollary 2.1.2 arise when considering the two encoding schemes.

Theorem 2.1 (AND Logic)**.** *Let G be an AND gate having β input terminals. At any single bit operation coming from β inputs, the output is denoted as $Y^{(b)} = X1^{(b)} \wedge X2^{(b)} \wedge \cdots \wedge X\beta^{(b)}$. At the output, $Y^{(b)} = 1 \in \mathbb{B}$ is obtained if and only if $\forall X^{(b)} = 1 \in \mathbb{B}$.*

Corollary 2.1.1 (AND Logic in UPE-Based SC)**.** *UPE-based bitstreams $\{\boldsymbol{X1}, \boldsymbol{X2}, ..., \boldsymbol{X\beta}\}$ applied to G yield $P(\boldsymbol{Y}) = P(\boldsymbol{X1}) \times P(\boldsymbol{X2}) \times ... \times P(\boldsymbol{X\beta})$.*

Proof. With a reminder of the *independence property* of the inputs, each bitstream probability is independent from each other s.t. $P(\boldsymbol{X1} = 1)$ *AND* $P(\boldsymbol{X2} = 1)$ *AND* ... *AND* $P(\boldsymbol{X\beta} = 1) \rightarrow P(\boldsymbol{X1}) \times P(\boldsymbol{X2}) \times \cdots \times P(\boldsymbol{X\beta})$. □

Corollary 2.1.2 (AND Logic for UPE-Based Inputs BPE-Based Outputs)**.** *UPE-based input bitstreams $\{\boldsymbol{X1}, \boldsymbol{X2}, ..., \boldsymbol{X\beta}\}$ are applied to G, and regarding the BPE-based output, the following is obtained:* $2 \times (P(\boldsymbol{X1}) \times P(\boldsymbol{X2}) \times ... \times P(\boldsymbol{X\beta})) - 1$.

Proof. Via Lemma 2.1, Corollary 2.1.1 is denoted as a BPE-based version of the output. □

Encoding type affects the expected SC operation. Like AND multiplier in UPE bitstream processing, BPE inputs are processed via XNOR for multiplication [32].

• Division

Division is another basic arithmetic operation implemented in the stochastic domain. In the literature, the first version of the SC-based dividers was rooted in Gaines's adaptive digital element (ADDIE) design [1]. ADDIE, as the simplest example of an integrator, was considered by Ananth's division circuit for bitstream processing [63]. Chen and Hayes also worked on the stochastic division architecture exploiting the correlation. Correlated division (CORDIV) presents the lower area cost, utilizing RNGs, comparator, MUX, and D-type flip-flop [64]. Moreover, a simple JK flip-flop can also be used as an approximate divider [65, 66]. As $\boldsymbol{X1}$ and $\boldsymbol{X2}$ input bitstreams are applied to the J and K ports of a JK flip-flop (with $P_{\boldsymbol{J}}$ and $P_{\boldsymbol{K}}$ probabilities, respectively), the output bitstream, $\boldsymbol{Y}$, is obtained from the Q port yielding $P_{\boldsymbol{Y}} = \frac{P_{\boldsymbol{J}}}{P_{\boldsymbol{J}}+P_{\boldsymbol{K}}}$ [66,67]. This equation is quite approximate and the recent efforts in the literature target mitigating the approximation level of JK flip-flop by converging to $P_{\boldsymbol{Y}} = \frac{P_{\boldsymbol{J}}}{P_{\boldsymbol{K}}}$ [68–70].

• Addition and Subtraction

Last but not least, the accumulation and subtraction in SC are presented before proposing the scaled adder optimization in Section 2.4. The primitive addition operation is performed by OR gate approximately in SC. For UPE-based $\boldsymbol{X1}$ and $\boldsymbol{X2}$ input bitstreams, the OR gate expression is $P_{\boldsymbol{Y}} = P_{\boldsymbol{X1}} + P_{\boldsymbol{X2}} - P_{\boldsymbol{X1}} \times P_{\boldsymbol{X2}}$ [32]. The OR gate is an adder provided that the input operands hold small probability values. Therefore, $P_{\boldsymbol{X1}} \times P_{\boldsymbol{X2}}$ becomes negligibly small in the OR expression. Besides, approximate parallel counters (APCs) are frequently used for high accuracy in the literature [71]. APC accepts input in the bitstream format; however, the output is a deterministic binary value [72]. As a result, using APC necessitates both deterministic and stochastic conversions, i.e., multiple encoding. Unlike APC, the MUX circuit keeps the operands' stochastic bitstream format throughout SC execution and presents a "scaled adder" [41, 71, 73]. In addition, it is proposed to utilize modulus (MOD) counters for accumulation during the population count in SCNN applications. MOD counters will be discussed in Chapter 5.

Let us explain the frequently used adder architecture: MUX scaled adders. In Figure 2.3, an example is depicted. MUX input ports accept the input operands as bitstreams. The selection port, S, has the third bitstream with 0.5 probability. Thus, half of the bits are randomly chosen from the first input of MUX, and the other half are selected from the second port. The selection port probability brings a "scaling factor" that halves the total sum. Thus, this architecture is called a "scaled adder" and might be considered limited for some applications due to its accuracy [74].

Figure 2.3 presents a bitstream processing MUX example for scaled addition. Two input scalars, $6/8$ and $6/8$, are encoded into two random bitstreams using UPE, together with a selection port bitstream having a $1/2$ value. The output is half of the total sum due to the scaling factor. Finally, a 2-to-1 MUX implements equation (2.4) when $P_{\boldsymbol{S}}$ is set to $1/2$ [15]. In a more theoretical way, output probability is obtained via $P_{\boldsymbol{Y}} = P_{\boldsymbol{X1}}(1 - P_{\boldsymbol{S}}) + P_{\boldsymbol{X2}}P_{\boldsymbol{S}}$ by recalling AND, OR, and NOT operations in SC.

$$P_{\boldsymbol{Y}} = \frac{P_{\boldsymbol{X1}} + P_{\boldsymbol{X2}}}{2} \tag{2.4}$$

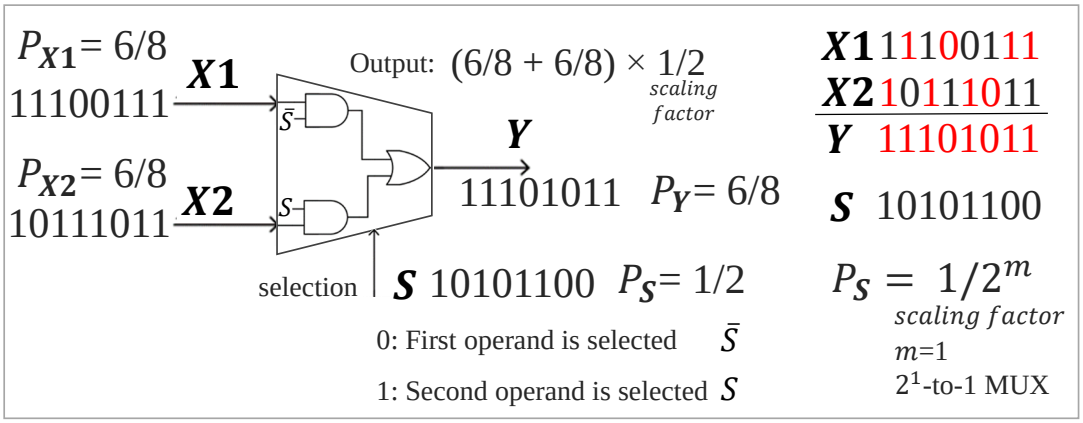


Figure 2.3 : Example of a MUX-based scaled adder.

Furthermore, the scaled subtractor is another unit among SC architectures [75]. By applying the NOT to the input being subtracted, the traditional MUX architecture is used for subtraction. Figure 2.4 depicts how to subtract $6/8$ from $6/8$. MUX-based subtraction has one more term other than the scaling factor. The output is written as $P_Y = (1-P_{X1})(1-P_S)+P_{X2}P_S$. For $P_S = 1/2$, the equity turns into equation (2.5). The extra $+\frac{1}{2}$ term comes from $(1-P_S)$, and scaled subtraction has a more approximate output.

$$P_Y = \frac{P_{X2}-P_{X1}}{2}+\frac{1}{2} \tag{2.5}$$

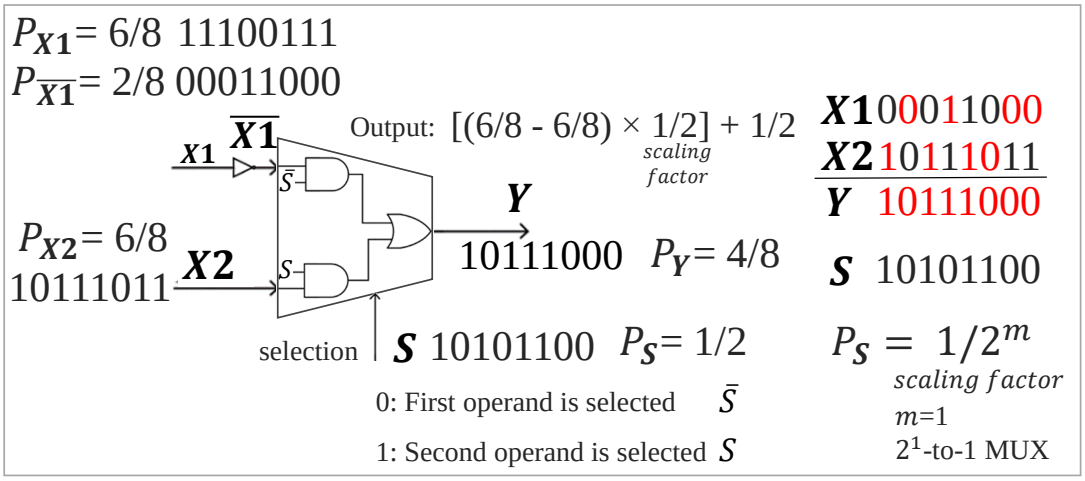


Figure 2.4 : Example of a MUX-based scaled subtractor.

Both the MUX scaled adder and subtractor work for UPE- and BPE-based inputs, but the selection is required to be encoded in UPE.

2.4 *Optimization*: Cascaded Scaled Adder

Accumulation of multiple inputs via MUX requires cascaded MUXs yielding a MUX tree. Since different-input-sized MUX architectures (such as 2-to-1, 4-to-1, 8-to-1, etc.) can be selected at each stage, an optimization algorithm is proposed in this section. The approach optimizes the stage-wise selection of MUXs and reduces the logic-gate resource utilization and latency.

Let the set of addition operands be $\{X1, X2, \ldots, X\beta\}$, where the total operand count equals to β. If β is much greater than the MUX input size (2^m), the use of single-stage scaled adders results in low operational accuracy. These adders perform well for integer m having $1 \leq m \leq 4$. However, the MUX selection port bitstream probability (P_S) limits the large-input MUX usage due to the scaling factor, $\frac{1}{2^m}$ [24]. An optimization algorithm is proposed to set up cascaded MUX architecture giving similar accuracy but with better hardware resource utilization (HRU) and latency. The optimization is based on the dynamic decision on each stage's MUX input size as defined in Definition 2.9.

Definition 2.9 (Optimized MUX Tree)**.** A MUX tree, with an input operand count β, is dynamically treated at each cascaded stage for optimization. The β at each stage is divided by the *MUX input size*, thereby yielding new intermediate operands for use in the subsequent stage. Let denote the *MUX input size* (2^m) as L. As accumulation proceeds along subsequent stages, the new intermediate term count in the $(n+1)^{th}$ stage is given by $\beta_{n+1} = \frac{\beta_n}{L_{n+1}}$. Since L is generally fixed, Algorithm 1 dynamically tunes L in the MUX tree. Thus, an optimized MUX tree with the dynamically cascaded sub-MUXs is obtained.

Algorithm 1 starts operation by handling the initial addition operands, β_0. The first-stage MUX-input size L_1 is a user-defined parameter. Since the MUX tree reduces the L_1 value in subsequent stages, based on our observations of the simulations, it is recommended to assign L_1 up to 16.

In Algorithm 1, the `setMUXStage` procedure is called until the output bitstream is obtained from the adder. The L-tuning operation is given between lines 7 and 11 of the

algorithm. If the new intermediate term count falls below twice the current L_{n+1} value, the MUX size in the next stage is given by half of L_{n+1}; i.e., $L_{n+2} = \frac{L_{n+1}}{2}$. Furthermore, for the β values not divisible by L, an additional MUX with zero-input padding is deployed via the `padding` procedure. The *remainder* via (β_n *mod* L_{n+1}) defines the inputs of this additional MUX for padding purposes. Zero padding is utilized for the remaining MUX inputs, i.e., ($L_{n+1} - remainder$).

Algorithm 1 MUX-tree Construction

1: **function** MUXTREE(β_0, L_1)
▷ β_0: initial operand count, L_1: the first-stage MUX-input size
2: **from** $n = 0$ **do** SETMUXSTAGE(β_n, L_{n+1}) , $n \leftarrow n + 1$
▷ proceed step-by-step
3: **while** $\beta_n \neq 1$ ▷ until the resultant bitstream
4: **end function**
5: **procedure** SETMUXSTAGE(β_n, L_{n+1})
6: $\beta_{n+1} \leftarrow \beta_n / L_{n+1}$
7: **if** $(\beta_{n+1} \geq 2 \times L_{n+1})$ || $(L_{n+1} = 2)$ **then** ▷ L-tuning starts
8: $L_{n+2} \leftarrow L_{n+1}$ ▷ the same L
9: **else if** $(\beta_{n+1} < 2 \times L_{n+1})$ && $(L_{n+1} \neq 2)$ **then**
10: $L_{n+2} \leftarrow L_{n+1}/2$ ▷ divide L by 2
11: **end if** ▷ L-tuning ends
12: $[remainder,\ extraMUX] =$ PADDING(β_n, L_{n+1}) ▷ extra MUX?
13: $\beta_{n+1} \leftarrow ((\beta_n - remainder)/L_{n+1}) + extraMUX$ ▷ extra MUX
14: **end procedure**
15: **procedure** PADDING(β_n, L_{n+1})
16: $remainder \leftarrow (\beta_n\ mod\ L_{n+1})$
17: **if** $remainder \equiv 0$ **then** $extraMUX \leftarrow 0$ ▷ no extra MUX
18: **else** $extraMUX \leftarrow 1$ ▷ one extra MUX with padding
19: **end if**
20: **end procedure**

Figure 2.5 depicts a sample construction for the MUX tree with $\beta_0 = 744$. The above algorithm is initiated by setting $L_1 = 8$ not exceeding 16 as recommended, and $\frac{\beta_0=744}{L_1=8}$ feeds the second stage with $\beta_1 = 93$ intermediate terms. Since the dynamic optimization condition $\beta_1 \geq (2 \times L_1)$ is satisfied for L-tuning in line 7 of Algorithm 1, L_2 is defined similar to L_1. When it comes to the second stage, $\beta_1 = 93$ is indivisible by $L_2 = 8$ with *remainder* $= 5$; hence, an additional padding MUX is deployed with five inputs. The remaining ($L_2 - 5 = 3$) idle inputs are padded by the zero bitstreams. The rest of the MUX construction keeps the same fashion by applying L-tuning. Where necessary, extra MUX is added with padding inputs.

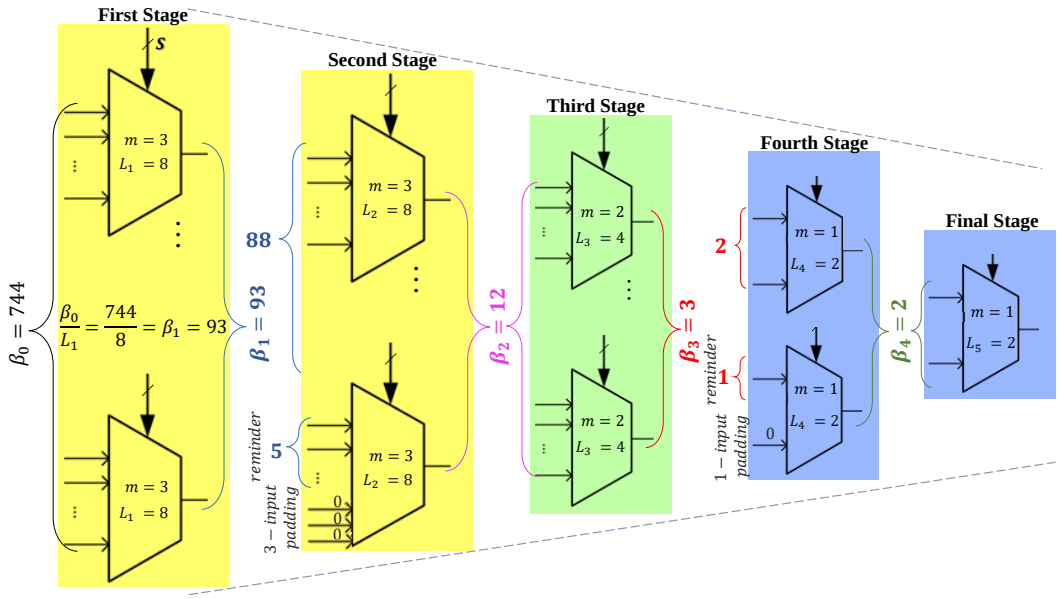


Figure 2.5 : Example of MUX tree optimization.

Considering the proposed algorithm, a simulation was performed utilizing the FPGA platform and the MATLAB-Simulink-Vivado cooperation feature. Table 2.2 shows three alternative designs for a 100-input accumulator: deterministic (with full adders), MUX-based stochastic adder, and the proposed via Algorithm 1. As previously stated, SC-based techniques result in lower HRU, which includes look-up tables and registers. Compared to the typical SC-based technique (2-to-1 MUX-only architecture), Algorithm 1 also presents a delay-efficient solution.

Table 2.2 : Accumulator comparison of deterministic versus SC approaches.

	Deterministic†	2-to-1 MUX$^{\blacktriangle}$	Algorithm 1$^{\blacktriangle,\ddagger}$
HRU	27644	4452	3193
Logic Delay	9.768 ns	17.086 ns	10.252 ns

Each candidate design was simulated on the FPGA in a fully-parallel manner by setting the target device as Xilinx Zynq-7000. †Addition of 100 operands in 8-bit. $^{\blacktriangle}N = 256$-bit LFSRs were utilized for random bitstream generation. $^{\ddagger}L_1 = 8$ initially.

2.5 Non-Idealities

SC is acclaimed in the literature for noise-robust and fault-aware applications [6, 9, 14–16, 76–80]. Accuracy-fragile applications, such as NNs, easily get affected by the non-ideal cases (such as faults, defects, error, noise, etc.) that occur in a system. If the

non-ideal cases occur on inputs (X), NN weights (w), or activation functions (f), the model accuracy decays significantly [81–84].

Cilasun and Altun explain the terms defect, fault, or error as a generic concept. These all are definitions (may be used interchangeably) for the abnormalities. It is underlined that if a fault is a one-time problem that does not occur repeatedly, then it is considered a soft error [16, 17]. The numbers created by SC, an unconventional paradigm, are expressed in an unusual way with no regard for the bit importance. As a result, SC outcomes are unaffected by bit placement. The bit-flips and any kind of faults like stuck-at-0 or 1 can be vulnerable in conventional binary number representation. MSB - LSB hierarchy is prone to cause severe accuracy drops in applications. In Example 2.2, the flip in the MSB in the presence of any soft error causes significant changes in conventional representation. In SC representation, 4 becomes 5 in the presence of soft error; however, 4 in the traditional binary (base-2) case becomes 132 in the MSB-related error of the conventional representation.

Example 2.2. Example of bit-flips that occur in SC-based and binary representations.

$(01100110)_{SC} = (4)_{10} \quad 0 \rightarrow 1 \text{ flip } (\mathbf{1}1100110)_{SC} = (5)_{10}$

$(00000100)_2 = (4)_{10} \quad 0 \rightarrow 1 \text{ flip } (\mathbf{1}0000100)_2 = (132)_{10}$

In Figure 2.6, three important non-idealities are depicted considering the application level. The first is the bit-flips on the data. For applications like vision and learning machines, image data in hardware may have some bit-level problems [85]. The noisy texture is depicted for different bit-flip percentages applied. Second, the bit-flips can occur on NN weights, which is the cause of severe accuracy drops. The details of the "image" and "weight" bit-flips of the NN are discussed in Chapter 5. In Figure 2.6, there is also another example of non-idealities for image data. MNIST (Modified National Institute of Standards and Technology) presents a database for handwritten digits in grayscale format. Recently, Mu and Gilmer released a corrupted image collection version of MNIST [86], named MNIST-C [87]. MNIST-C includes a variety of corruption types, such as *defocus blur, glass blur, shot noise, saturate, motion blur, brightness, pixelate, JPEG compression, spatter, impulse noise,* etc. Together with image- and weight-based soft errors, the performance of this third non-ideal case for network applications will be discussed in Chapter 5. Besides these non-idealities, the Gaussian image noise is another non-ideal case in which SC can best fit corresponding

applications [78]. From the vision machine perspective, kernel-based image processing applications and noise performance will be tested in Chapter 4.

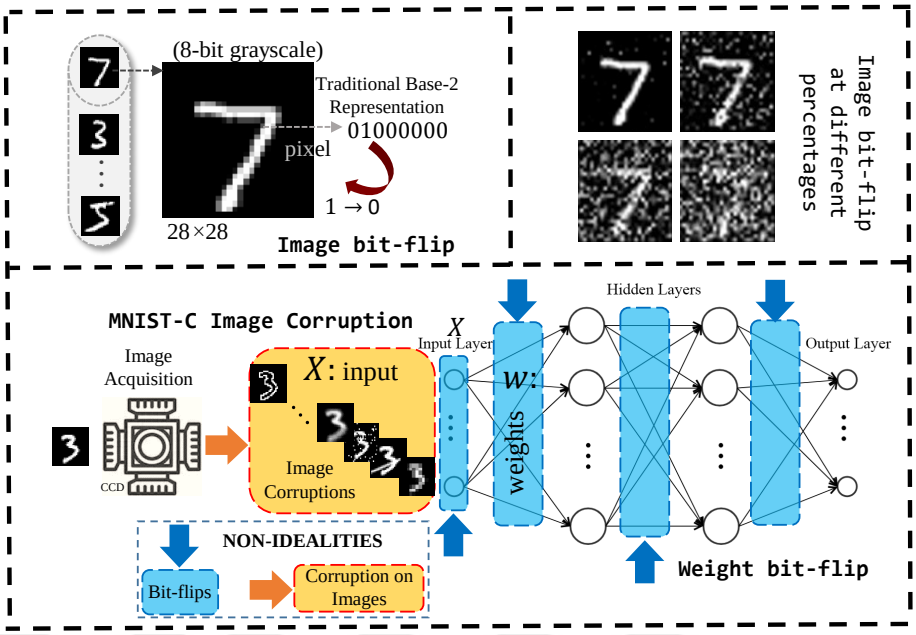


Figure 2.6 : Non-idealities: Bit-flips and corruption in network weights and data.

2.6 Survey on SC with Emphasis on Vision and Learning Machines

SC is an unconventional computation technique recently applied to computer vision and machine learning. Scalar representation is based on the probabilistic bitstreams, and simplified hardware is sufficient enough to supply intended arithmetic operations. In this section, a survey of SC will be presented to shed light on the current efforts on vision and learning machines.

There have been several survey papers on SC so far [88–92]. Alaghi and Hayes explained the basics of SC in [88], a leading survey paper in the SC community published in 2013. Especially underlining the applications, such as low-density parity-check (LDPC) and noise-robust image processing, basic hardware blocks were reported in the paper. In 2014, there was an effort to survey SC with an emphasis on probabilistic transfer matrix (PTM) in logic circuits [89]. The year 2014 was important for SC in terms of taking part in another survey on different topics like circuit reliability [93] by Xiao and Chen. Another extensive survey entirely on SC was authored by Alawad and Lin in 2019 [90]. Alawad and Lin reviewed the randomization issues, noise sources, pure electronic and corresponding emerging devices, addressing

probabilistic computing. Two other surveys presented SC studies that focus entirely on NNs. Liu et al., in their survey in 2021, hierarchically introduced the basic hardware components that build an SCNN in terms of arithmetic operations [91]. From that survey, one can benefit from the SC network inference accuracy and hardware efficiency table, which was prepared in a detailed and informative way. Speaking of NN, another survey in the literature only dealt with CNN architectures in SC [92].

Looking at the doctoral studies in the last ten years within the scope of SC in the literature, the general trend is toward computer vision and NNs. In Table 2.3, the prominent doctoral theses brought to the literature within the scope of SC for the last ten years are given.

Table 2.3 : The leading doctoral theses on SC in the last ten years.

Author	Title	Year
Qian [94]	*Digital yet deliberately random: Synthesizing logical computation on stochastic bit streams*	2011
Li [95]	*Analysis, design, and logic synthesis of finite-state machine-based stochastic computing*	2013
Alaghi [33]	*The logic of random pulses: Stochastic computing*	2015
Chen [31]	*Designing accurate and low-cost stochastic circuits*	2016
Bai [96]	*Stochastic-based computing with emerging spin-based device technologies*	2016
Liu [97]	*Digital signal processing and machine learning system design using stochastic logic*	2017
Barceló [98]	*Methodologies for hardware implementation of reservoir computing systems*	2017
Li [99]	*Distributed edge computing infrastructure with low hardware cost, performance evaluation and reliability*	2018
Najafi [100]	*New views for stochastic computing: From time-encoding to deterministic processing*	2018
Liu [101]	*Design and evaluation of stochastic computing neural networks for machine learning applications*	2019
Ardakani [102]	*Complexity reduction of deep neural networks for efficient hardware implementations*	2019
Lee [103]	*Towards practical stochastic computing architectures for emerging applications*	2019
Ting [104]	*Design of sequential stochastic computing systems*	2019
Liu [105]	*Toward energy-efficient, dynamic stochastic computing*	2019
Prado [106]	*Development of neuromorphic hardware and non-iterative learning designs for edge computing applications*	2020
El-Derhalli [107]	*Design of stochastic computing architectures using integrated optics*	2021
Najafi [108]	*Systematic design of low-power processing elements using stochastic and approximate computing techniques*	2021

The first study referring to image processing in the recent past was the 2011 Ph.D. thesis by Qian. In [94], Qian covers the realization of polynomial functions with SC hardware synthesis and their simple application, such as color space transformation, gamma correction, including noisy image issues. With the thesis in 2013, Li explained the design of SC elements via finite state machine (FSM) and presented block designs of special functions (stochastic tanh, exponentiation, absolute value, and linear gain) within the scope of SC [95]. The dissertation presented edge detection, median filter, contrast stretching, and image segmentation applications regarding image processing, including fault tolerance issues. The new age started with Alaghi's 2015 thesis [33], in which he presented a study targeting image processing. This work has been a reference guide many times in this dissertation. It is based on fundamental topics such as correlation and error definitions, stochastic circuit design, and their application to image processing, such as edge detection. It is a significant study introducing cross-correlation measurement among bitstreams and emphasizing the superiority of SC in terms of error robustness. Considering the edge detector application in image processing as a case study, Chen presented the doctoral research to the literature in 2016, especially with the proposal on stochastic divider (CORDIV) [31]. CORDIV has brought a new perspective to the division circuit literature.

With the year 2016, SC designs began to re-emerge conventional NN hardware. In fact, although SC and NNs have come together historically before [109, 110], their popularity has increased again with the acceleration of DNNs in 2015 and beyond. Bai demonstrated the use of SC in spin-based technology, one of the emerging designs, in his doctoral thesis in 2016 [96]. With the system called stochastic-based ANN, advantages of SC were underlined, such as performing the multiplication with the simple logic gates and built-in robustness. In 2017, machine learning notions finally appeared in SC, and concepts such as radial basis function (RBF) and support vector machine (SVM) were implemented using SC. Liu presented not only the machine learning concepts in the doctoral thesis in 2017 but also the use of SC in digital filter design [97]. With the acceleration of DNN architectures by conventional computing, they have also taken place in non-deterministic computing technologies. Barceló presented SC with a new perspective, combining the concepts of reservoir computing and spiking neural networks (SNNs) with SC [98]. In 2018, Li presented a study that

opened the door to unary computing by combining the reduced hardware output of SC with the QNNs [99]. That same year, Najafi introduced new perspectives from stochastic bitstreams to deterministic bitstreams. This was groundbreaking research proving how LD sequences increase SC performance with the aid of Sobol sequences [100]. In 2019, Liu used NNs to examine machine learning from a broader perspective [101]. Contributions were presented regarding SC adaptation to recurrent neural network (RNN) and realization of network training with the SC system. In [102], Ardakani introduced a new approach to SC with the adaptation of SC to XNOR RNNs and the well-known binary/ternary weight concepts. Lee, in [103], focused on design, simulation, and application co-design related to SC circuit components. On the other hand, that thesis introduced the structure of "hybrid stochastic-binary encoded NN" to the literature; therefore, SC and NN were brought together again in line with the trend of recent years. "Full taxonomy of stochastic computing circuits," presented with the appendix at the end of the thesis, is a resource that every researcher in the SC field should examine [103]. Ting touched upon the random fluctuations and correlation issues under the title of sequential SC systems. Considering Ting's work, that work included error concepts that clarify the randomness issue in SC [104]. In [105], Liu showed the parallel Sobol sequences, SC-based digital signal processing (DSP) applications, and SC-based network training in connection with ordinary differential equation (ODE) solvers. In 2020, Prado paved the way for the leap of SC applications to SNN [106].

In 2021, El-Derhalli became the first to combine SC with integrated optics. She designed the basic SC blocks as an optical system, and an SC edge detector within the scope of optical computing [107]. Recently, Najafi presented a doctoral study targeting low power with inaccurate multipliers and adders that includes both approximate and stochastic computing [108].

In addition to these surveys and doctoral theses, let us glance at SC-based vision and learning papers. The motivation for surveying SC stems from the increased scientific efforts in the literature on SC systems applied to vision and learning machines from 2015 to 2022. The prior efforts on image processing operations such as filtering and convolution have led to extending SC into the neural nets. There are numerous examples of image processing- and computer vision-related designs in

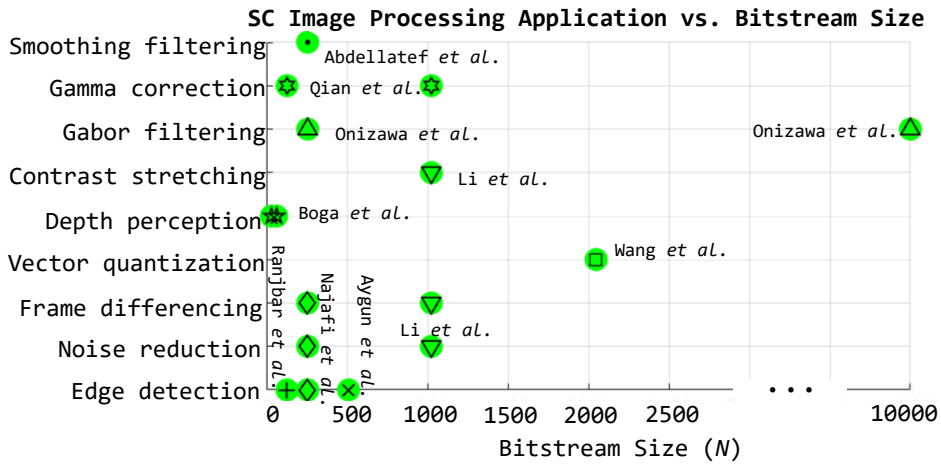


Figure 2.7 : Examples from the literature for SC image processing applications and applied bitstream size.

the literature. Noise reduction, edge detection, contrast stretching, frame difference, vector quantization, and depth sensing are examples of SC-based image processing [11, 33]. Figure 2.7 depicts different SC-based image processing applications and the applied bitstream size, N, for proper computing ($\rightarrow$ [111], $\rightarrow$ [15], $\rightarrow$ [112], $\rightarrow$ [113], $\rightarrow$ [114], $\rightarrow$ [115], $\rightarrow$ [8, 116], $\rightarrow$ [117], $\rightarrow$ [118].) This observation is also crucial for the long-running simulation bottleneck of SC image processing.

Especially when there are soft errors and a noisy environment, SC outperformance can be exploited significantly. Qian et al. primarily showed the efficiency of SC-based image processing applications in the presence of faults such as bit-flips, i.e., soft errors. They presented gamma correction examples and proved how smooth results could be obtained in the presence of soft image errors [15]. Najafi and Salehi proposed another fault-tolerant architecture to measure the effect of the noisy environment during the Sauvola local image thresholding. The design platform of their study is similar to our work, such as a collaboration of MATLAB and FPGA environments [119].

Li et al. introduced SC-based image operations such as edge detection, frame difference, noise reduction, contrast stretching, and KDE. They discussed hardware resource comparison at the level of two-input NAND gates [115]. Li and Lilja thoroughly drew the mainframe of the stochastic image processing for the noise-robustness [11]. From the hardware perspective, stochastic edge detection circuit details were shared by Alaghi et al. [10, 120]. They utilized Gaussian noise with zero

mean, $\mu = 0$, and variable standard deviation, σ, which they added to the voltage signal generated by the sensor. Thus, calculating the full voltage swing of the sensor in their vision chip, some percentages over the sensor voltage range were utilized as the injected noise values. Gaussian noise over an image in the test was also employed. In Chapter 4 of this dissertation, the selection of image acquisition noise parameters, μ and σ^2, are performed in the flexible software-only platform.

Speaking of noise, Najafi and Lilja proposed a pulse width modulation (PWM) signals-based design that helps to reduce the latency of SC-based systems in general. Their proposed low-cost PWM generator performed well in median filter noise reduction, edge detection, and frame difference-based image segmentation. Thanks to highly overlapping PWM signals, highly correlated pulses were utilized, and robust results were obtained [111]. Seva et al. proposed the variable bit truncation for approximate stochastic computing (ASC). They performed the median filter-based noise reduction and edge detection. It should be noted that edge detection peak signal-to-noise ratios (PSNRs) were in the range of $[23dB,\ 39dB]$ based on different clock cycle lengths [121].

Differently, an image compression application using SC was developed by Moons and Verhelst. They explained the multi-stage stochastic architectures with compression case studies. It was underlined that the number of scaled adders in a cascaded fashion affects the overall performance. Increasing the stream size was a way to compensate for the performance drops; however, they underlined how it could be drastic in terms of power consumption [12, 13]. Therefore, they proposed the optimum number of MUX stages (s) in cascade, which was inferred as $s = 3$. There are other image processing efforts in the frame of SC, addressing depth-sensing [117] and circuit aging-aware image processing [122]. Moreover, the distinguished efforts in a higher-level perspective targeted pure signal processing operations from finite impulse response (FIR) [123] to infinite impulse response (IIR) [124] in a broad view of digital signals [125].

Gaines, who presented the preeminent SC studies starting in the 1960s, recently underlined the impact of numerous research efforts in SC [126]. After data- [21, 127–130] and image processing-related SC applications [10–13] were proposed, SCNNs have become a focal point [20, 71, 76–78, 92, 131, 132]. Since stochastic bitstream

processing systems provide hardware-efficient [26], highly parallel [24], low-power consuming [131], and noise-robust [11] solutions, the design of SC-based lightweight NN has become a center point. In previous efforts, the hardware efficiency of SC-based architectures was very well explained; however, a wide perspective of image corruption and soft error robustness inspection (especially for binarized NN architectures) is required.

The current demand for power- and area-efficient NN solutions has inspired renewed interest in SC-related research during the last decade [133–137]. Recent efforts to overcome challenges encountered in the construction of NNs that display human brain-like behavior were described in the literature [24, 71, 72, 78, 128, 138–140]. SC provides a vast, parallel environment for performing arithmetic operations due to its small hardware footprint [33, 109, 110, 141], thereby being an appealing solution for NNs. In the frame of SC and approximate computing (AC) [142, 143], there has been much interest in building efficient hardware that can handle arithmetic operations with high precision [19, 99, 144–148]. Sim and Lee [144, 145] suggested a low-cost SC multiplier and accumulator for NN applications. Li et al. presented a low-cost stochastic multiplier for quantized NNs [19, 99].

For the SCNN applications, various interesting studies have inspired this thesis. Li et al., in [134], measured the effect of non-optimized feature extraction on their deep stochastic CNN. Using two feature extraction layers, non-optimization could cause a test error rate up to 27.83%. On the other hand, Kim et al. worked on the structure of an NN considering the network dynamics. They addressed the near-zero values during SC multiplication and early decision termination on the network training [71]. The noise issues in the network, including stream processing, were offered by Babu et al. with the usage of nanoscale devices. With a stochastic weight update, the $O(1)$ time complexity was proposed, and the total test accuracy of handwritten digit recognition was achieved at 88.1%. They simulated the memristive devices at the synapse and presented the stochastic update on the training phase [78]. Liu et al. also proposed SC usage in the training phase of an NN [72]. Recent efforts on SCNN can be found for each sub-module design of a network, such as activation, normalization, dropout, softmax regression, pooling, and gradient descent amongst others. Liu et al. gave the details of two activation functions, hyperbolic tangent and sigmoid, via stochastic logic in [149].

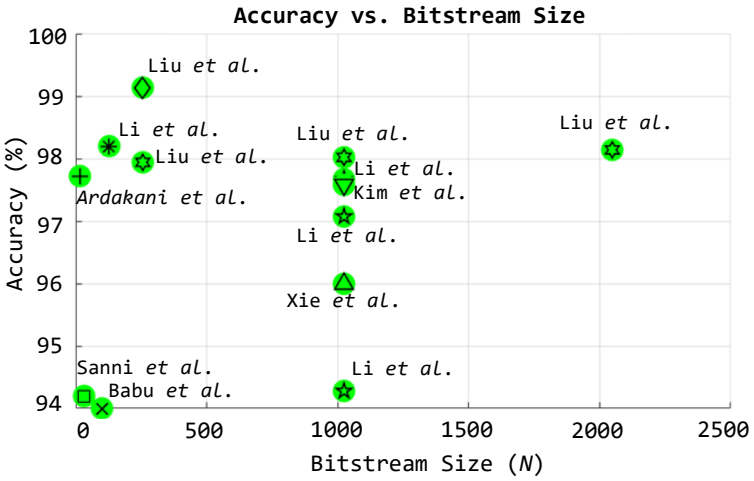


Figure 2.8 : Previous efforts on the stochastic bitstream processing NNs.

Li et al. designed and showed comparatively the tanh, logistic, and ReLU activation function performances on the stochastic system. In terms of area, power, and energy, tanh outperformed for longer bitstreams; however, for the reduced N, ReLU brought less accuracy degradation [150]. A design of RBF in the hardware for stream-based NN was given in [151] by Ji et al. The normalization- and dropout-related efforts on SC systems were made by Li et al. [152]. They proposed pooling and enhanced near-max pooling design with stream-based computation elements. Another vital module in deep architectures, the softmax regression design for stochastic networks, was completed by Yuan et al. in [153].

One of the main challenges in SCNN architectures is the bitstream length. In Figure 2.8, the SC-based multi-layer perceptron (MLP) architectures and performing N values are depicted ($\rightarrow$ [139], $\rightarrow$ [19], $\rightarrow$ [72], $\rightarrow$ [154], $\rightarrow$ [140], $\rightarrow$ [78], $\rightarrow$ [71], $\rightarrow$ [24], $\rightarrow$ [128].) Figure 2.8 presents the accuracy obtained on the MNIST digit dataset by previous bitstream-based NN architectures as a function of the bitstream length. Most studies in the literature use an MLP with two hidden layers, as will be considered in this study. Moreover, Chapter 5 shows that efficient and robust binarized architecture with $N = 8$ or 16 bitstream length is sufficient for high classification accuracy. Figure 2.8 is an essential indication of the required N, which is then a bottleneck (as indicated in Figure 2.7) targeted by the agile SC simulation explained in the following chapter.

3. SOFTWARE-DRIVEN STOCHASTIC COMPUTING

SC systems are used in different applications from communication systems to NNs. Even though the hardware design side of SC has been visited by most of the studies before, the simulation aspects and the software applications in the vision and learning machines are blurry. In this chapter, a fast simulation approach is presented to manipulate input scalars without generating bitstreams. Considering the bitstream correlation, SC operations are proposed to be performed by only manipulating the scalar values of inputs without the bitstream encoding. One-to-one imitation of bitstreams is considered, and the error issues related to the random fluctuations are also discussed. To conclude, a fast simulation environment is constructed in a software platform using the proposed contingency table (CT) approach.

3.1 Contingency Table (CT) in Stochastic Computing

One of this thesis' contributions is the agile software simulation of SC systems. It is proposed to adjust bitstream correlation and simulate expected logic operations without using bitwise operators. In this context, first of all, the definition of the CT concept is given in Definition 3.1.

Definition 3.1 (Contingency Table and its Primitives)**.** Assuming $X1 \in \mathbb{Z}_0^+ \leq N$ and $X2 \in \mathbb{Z}_0^+ \leq N$ represent the number of *logic-1*s in their prospective bitstreams (recalling TC notation), the overlapping bits on the same bit positions of input streams are cumulatively recorded in a table, defined as CT. Each unique bit overlapping of bitstreams (such as 11, 10, 01, and 00 for two input operands) is called "CT primitive" or "CT element." The number of total primitives is equal to $2^{number\_of\_input\_operands}$.

CT primitives are the essential elements in the CT construct. Like a confusion matrix in data mining, CT can be thought of a table based on the occurrences of event counts, i.e., overlapping bits, in the input bitstreams [155]. As described in Chapter 2, input scalars such as $X1$ and $X2$ can be represented in bitstreams, $\boldsymbol{X1}$ and $\boldsymbol{X2}$, using any

encoding scheme. At this point, an "overlapped bitstream" concept is defined to clarify the idea of CT primitives. Assume two binary vectors A and B of equal size. At any m^{th} element of vectors, there are four possible logic couples: $\big(A(m)=1,\ B(m)=1\big)$ *or* $\big(A(m)=1,\ B(m)=0\big)$ *or* $\big(A(m)=0,\ B(m)=1\big)$ *or* $\big(A(m)=0,\ B(m)=0\big)$. The two-input CT is given in the form of possible event counts, having a, b, c, and d values attributed to set operations as $|A\cap B|$, $|A-B|$, $|B-A|$, and $|\bar{A}\cap\bar{B}|$. In terms of binary vectors, a, b, c, and d hold the counts of unique logic couples $\big(A(m)B(m)\big)$; 11, 10, 01, and 00, respectively [155,156]. All these logic couples form an overlapped bitstream that constitutes a, b, c, and d CT primitives. For example, from the bitstreams $\begin{matrix}X1\rightarrow 0\ 1\ 0\ 1\ 0\ 1\ 1\ 0\\ X2\rightarrow 0\ 1\ 1\ 1\ 1\ 1\ 0\ 1\end{matrix}$, the CT primitives are obtained as $a=3$, $b=1$, $c=3$, and $d=1$ via blue, green, magenta, and orange color overlapping. This yields an overlapped bitstream as $dacacabc$; however, N-bit physical bitstream (neither input nor output) is never generated and saved, instead only the scalars (and the numerical values of CT primitives) are dealt with. Therefore, the repetitions of each primitive are saved and all the logic operations, correlation manipulation, random fluctuation error imitation are performed using these numerical values.

In Figure 3.1, the above example is depicted. Scalars $X1$ and $X2$ are represented in bitstreams using bold font $\boldsymbol{X1}$, $\boldsymbol{X2}$, and each overlap in the input bitstreams is represented by a, b, c, and d CT primitives. The total count of each CT element is shown, where $\boldsymbol{X1}(i)$ and $\boldsymbol{X2}(i)$ denote any i^{th} element (by recalling Definition 2.2). At this point, it should be noted that: $a,b,c,d\in\mathbb{Z}_0^+$.

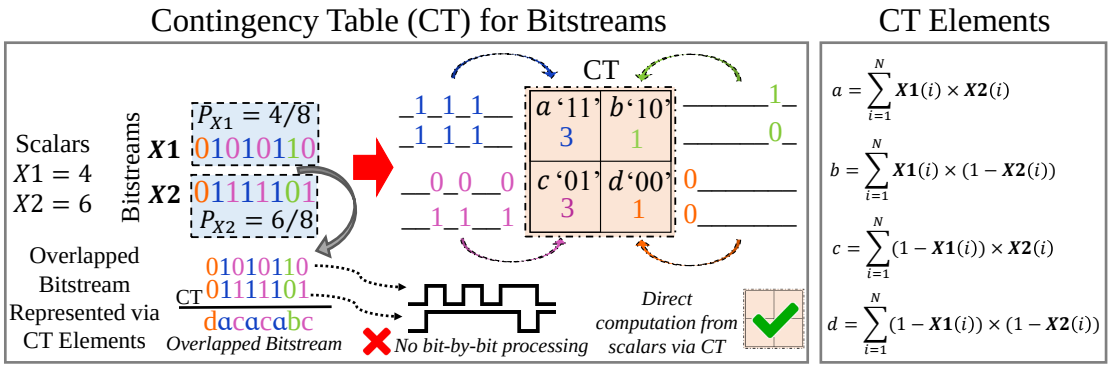


Figure 3.1 : Definition of CT and its primitives.

3.2 Scalar-Only Logic Operations: *CT in Stochastic Logic*

The relation between CT and the SC logic operations is explained in this section. CT primitives are evaluated to define logic gate operations such as AND, OR, XNOR, etc. This proposal is to address decreasing the lengthy bit-by-bit simulation time (latency) and memory issues. Mapping each logic function into the combinations of a, b, c, and d lets us set the simulation environment efficiently in terms of runtime. The inference between CT primitives and the logic operations can be explained using a two-input AND operation. The CT primitive a is the '11' count in the overlapped bitstream, and the AND gate output bitstream relates to a. This relation can be explained in the frame of total *logic-1*s at the output bitstream; thus, the **T**otal **C**ount of 1s at **O**utput (TCO) is defined. After the SC logic operation, each *logic-1* at the output is counted to decode the resultant bitstream. This TCO value of the AND gate is exactly equal to the CT primitive a. In Table 3.1, the other logic operations are defined, and numerical examples are included to clarify the concept. Finally, after the CT is obtained from $X1$ and $X2$, the logic operations can be performed using the linear combinations of CT primitives. The CT setting is based on the correlation between two operands, and the next section discusses correlation measurement.

Table 3.1 : Relation between logic operations and CT primitives.

	$X1 = 4$	$\rightarrow \boldsymbol{X1} = 10001101$	$a = 3 \quad b = 1$
	$X1 = 6$	$\rightarrow \boldsymbol{X2} = 11011110$	$c = 3 \quad d = 1$
$TCO = a + b + c$	Logic OR:	$\vee$	$11011111 \rightarrow TCO = 7$
$TCO = b + c$	Logic XOR:	$\oplus$	$01010011 \rightarrow TCO = 4$
$TCO = b + c + d$	Logic NAND:	$\neg\wedge$	$01110011 \rightarrow TCO = 5$
$TCO = d$	Logic NOR:	$\neg\vee$	$00100000 \rightarrow TCO = 1$
$TCO = a + d$	Logic XNOR:	$\odot$	$10101100 \rightarrow TCO = 4$

3.3 The Measures of Correlation

As a measure of the relationship between input bitstreams, "correlation" has an important place in previous studies [38, 40–43, 57, 157, 158] and in this dissertation since the correlation between bitstreams influences the accuracy of the SC arithmetic (by recalling Section 2.2.1).

In terms of the CT method, correlation plays an important role in determining primitives based on scalar values. The *SCC* that has been proposed in the literature specifically for SC will be the main correlation observation criterion used in this thesis. In Table 3.2, *SCC* and other binary similarity equations used for observation are given in terms of CT primitives. The *SCC* proposed by Alaghi et al. tops the list. In [38, 42], they presented the properties of the *SCC*, including an efficient digital hardware design for it, which was clearly explained as state-of-the-art. The *SCC* has a coefficient range of $[-1,1]$. For positive (maximum) correlation, *SCC* converges to $+1$. For negative (minimum) correlation, *SCC* converges to -1. Most importantly, uncorrelated, i.e., with zero-correlation, input bitstreams have an *SCC* value converging to 0 ($SCC \approx 0$). This case indicates that some digital logic gates operate at the lowest error for the expected SC arithmetic [34].

On the other hand, other binary similarity equations are included in Table 3.2 for comparison purposes. Considering the tests in the next sections, a performance comparison of *SCC* will be performed via these equations. *Pearson Correlation* (*PC*) [159] and *Jaccard* (*JAC*) [160] are listed as the most well-known similarity equations in the literature [161]. *Dice* (*DI*) [162], similar to *JAC* with a scalar multiplier of 2, is also included in Table 3.2. *DI* evaluates a's diverse dominance in a CT, i.e., 11 logic couples in bitstream inputs. This is important to monitor since 'a' will be announced as a prior primitive in the following section. In addition to these similarities, *Kulczynski 2* (*K*2) [163] and *Ochiai* (*OC*) [164] are considered the arithmetic and geometric mean of the *precision* $(a/(a+b))$ and *recall* $(a/(a+c))$ [156].

Table 3.2 : Similarity equations of the two bitstream inputs.

Equations		
Stochastic Cross-Correlation (*SCC*) $\begin{cases} \frac{ad-bc}{N\times min(a+b,a+c)-(a+b)\times(a+c)} & , if\ ad > bc \\ \frac{ad-bc}{(a+b)\times(a+c)-N\times max(a-d,0)} & , else \end{cases}$		
Jaccard (*JAC*) $\frac{a}{a+b+c}$	**Dice** (*DI*) $\frac{2a}{2a+b+c}$	**Kulczynski 2** (*K*2) $\frac{1}{2}(\frac{a}{a+b}+\frac{a}{a+c})$
Pearson Correlation (*PC*) $\frac{ad-bc}{\sqrt{(a+b)\times(a+c)\times(b+d)\times(c+d)}}$		**Ochiai** (*OC*) $\frac{a}{\sqrt{a+b}\times\sqrt{a+c}}$

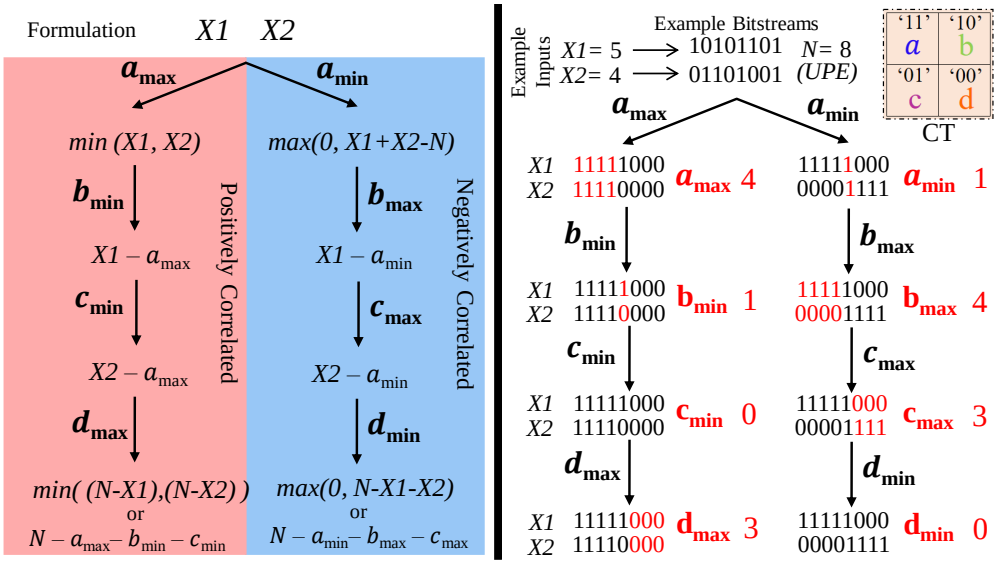


Figure 3.2 : Maximum and minimum correlation: The formulation and an example.

3.4 CT-Based Manipulation

In SC systems, the representation of bitstreams is affected by the correlation. Even though the random procedure makes the bitstreams probabilistic, there is also a way to generate bitstreams via controlled correlation like unary bitstreams [165–167]. The manipulation of the bitstreams, which obey a predefined correlation value, determines the operation of any logic system. Let us look first at the positive and negative correlations and then the zero correlations for CT.

3.4.1 Positive and negative correlation

The operation of any SC logic system is determined by manipulating bitstreams according to a preset correlation value. Positive and negative correlations can be achieved by controlling the bitstream encoding of the $X1$ and $X2$ input scalars. Through CT generation, a flow of extreme examples is shown in Figure 3.2, either as the maximum similarity $(a_{max} \dashrightarrow b_{min} \dashrightarrow c_{min} \dashrightarrow d_{max})$ or the minimum similarity $(a_{min} \dashrightarrow b_{max} \dashrightarrow c_{max} \dashrightarrow d_{min})$. When encoding bitstreams, the maximum and the minimum number of overlapping 11s exemplify the two extreme cases: positive and negative correlations. It is noted that the primitive a is the first to be decided while generating the CT. With the formulation given in Figure 3.2, a is determined whether the correlation represented by CT is maximum or minimum. All other CT primitives are determined by first deciding the starting point, a. In general, when a is determined,

$b = X1 - a$, $c = X2 - a$, and $d = N - (a + b + c)$ can be provided since $X1$, $X2$, and $N = a + b + c + d$ are already known beforehand. Correspondingly, a is highlighted as the most *prior* CT primitive.

3.4.2 Near-zero correlation

After the positive and negative correlations, the most crucial point, the zero correlation, is explained in this subsection. The proposed contribution on how to achieve CT-based near-zero correlation, i.e., uncorrelated CT, is explained below.

Three approaches are proposed in this study for the CT model of uncorrelated streams. The first is "*midpoint approximation*," as a heuristic and simple solution given in Definition 3.2. In this method, the a CT primitive is chosen as the midpoint of a_{max} and a_{min} that adjust the CT to represent the maximum and minimum correlation, respectively. Roughly assuming the midpoint is a near-zero correlation, a_{mid}, in Definition 3.2, is utilized while setting a.

Definition 3.2 (Midpoint Approximation Approach)**.** As $a_{max} \in \mathbb{Z}_0^+ \leq N$ and $a_{min} \in \mathbb{Z}_0^+ \leq N$, CT primitive a is set as $a_{mid} \leftarrow \lfloor (a_{max} + a_{min})/2 \rfloor$ to make CT presented uncorrelated input bitstreams roughly.

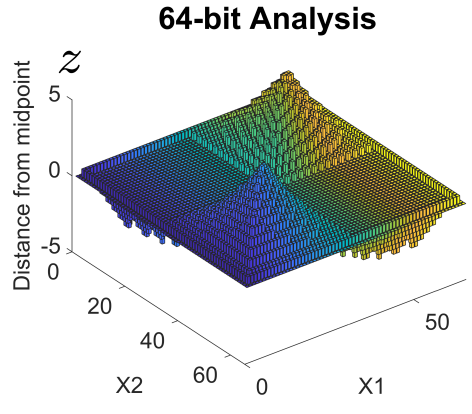
The second approach is "*polynomial fitting*," presented in Definition 3.3, which is based on statistical observations of exhaustive analysis. The fine-tuned version of the roughly determined a values by the first method is adjusted via the polynomial function, which is based on observations. These observations are based on the Cartesian combinations of $X1$ and $X2$ scalars.

Definition 3.3 (Polynomial Fitting Approach)**.** Let $\mathcal{X}1$ and $\mathcal{X}2$ represent the sets s.t. $\mathcal{X}1 = \{0, 1, 2, ..., N\}$ and $\mathcal{X}2 = \{0, 1, 2, ..., N\}$. Cartesian product, $\mathcal{X}1 \times \mathcal{X}2$, is the ordered pairs $(x1, x2)$, where $x1 \in \mathcal{X}1$ and $x2 \in \mathcal{X}2$. On $|\mathcal{X}1 \times \mathcal{X}2| = (N+1) \times (N+1)$ total elements, the roughly approximated a CT primitive for each case is first recorded and then compared with respect to the reference a values (a_{ref}) that guarantee the most optimal near-zero correlation, $SCC \approx 0$. Then, the analytical plots based on $\mathcal{X}1 \times \mathcal{X}2$ are rendered, placing the $a_{ref} - a_{mid}$ onto the z-axis. The obtained data are then fitted into the 4^{th}-order polynomial to be used as a single inline function while setting a.

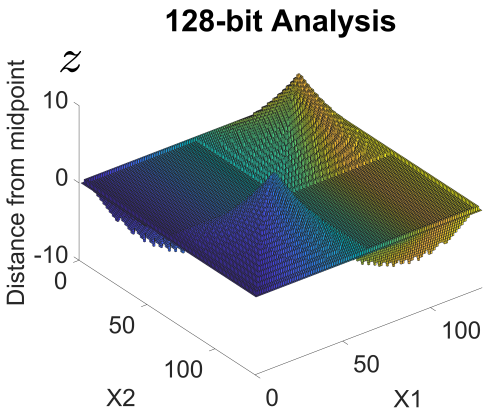
Figure 3.3 shows the statistical analysis of "all" possible bitstream couples using exhaustive lookup and the respective three-dimensional (3D) polynomial fitted version. $X1$ and $X2$ still represent the number of *logic-1*s in their prospective bitstream ($X1 \in \mathcal{X}1$ and $X2 \in \mathcal{X}2$). Based on the statistical observations from the Cartesian, the bar plots in Figure 3.3 (a) ($N = 64$) and Figure 3.3 (b) ($N = 128$) represent how to fine-tune a_{mid} values for $SCC \approx 0$ ($SCC = 0$ if possible). For each combination of $X1$ and $X2$ scalars to be represented in bitstreams, the superior CT construction via 'a' prior primitive is guaranteed for zero-correlated stream emulation. The z-axis on the plots represents the "compensation deviation" from a_{mid}, i.e., the middle point of maximum (a_{max}) and minimum (a_{min}) correlation. Therefore, any value in the z-axis guarantees to set CT for the ideal $SCC \approx 0$ via $a_{ref} \leftarrow z + a_{mid}$, where a_{ref} is the most optimal a value for $SCC \approx 0$. However, since this observation, including a lookup table, is costly in memory and runtime, the bar plots are fitted into the polynomials to supply single inline functions that return the compensation values to fine-tune a_{mid} as $a_{poly} \leftarrow z + a_{mid}$. Figures 3.3 (c) and 3.3 (d) represent fitted polynomial versions of Figures 3.3 (a) and 3.3 (b). The polynomial functions for $N = \{8, 16, 32, ..., 1024\}$ are obtained after the statistical observations and conversions via *MATLAB Curve Fitting Toolbox*.

In addition to these two heuristic approaches, the final method is based on optimizing the *SCC* formula, i.e., "*correlation coefficient optimization*." In Definition 3.4, this last approach is defined as an optimization-oriented method working on the *SCC* formula. Lemma 3.1 and the related proof show how to optimize and obtain an equation for a. Based on the total count of each a, b, c, and d CT element, given in Figure 3.1, the equity is set: $[(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i)) \times (\sum_{i=1}^{N} (1 - \boldsymbol{X1}(i)) \times (1 - \boldsymbol{X2}(i)))] - [(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times (1 - \boldsymbol{X2}(i))) \times (\sum_{i=1}^{N} (1 - \boldsymbol{X1}(i)) \times \boldsymbol{X2}(i))] = 0$, i.e., *SCC* numerator in Table 3.2 $ad - bc = 0$, to be solved for optimization.

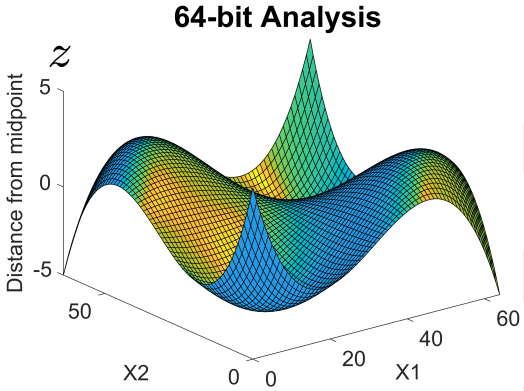
Definition 3.4 (Correlation Coefficient Optimization Approach)**.** By recalling *SCC* from Table 3.2, its numerator value, $ad - bc$, is to be optimized for $SCC \approx 0$. The a after the optimization is the required value to set CT for near-zero correlation and is expected to be the closest value to a_{ref} among the two previous methods, thereby assigning the "a_{best}" name for it. Hence, CT primitive a is designated as $a_{best} \leftarrow \lfloor \frac{X1 \times X2}{N} \rceil$ to get CT to emulate uncorrelated input bitstreams.



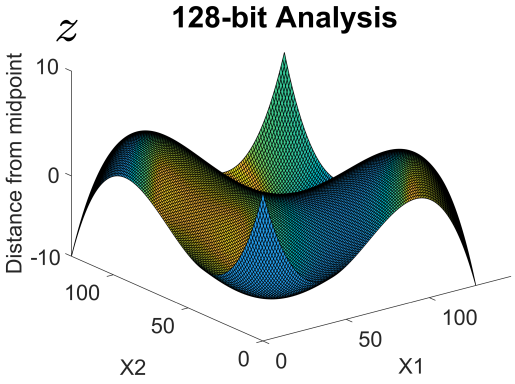
(a) Fine-tuning for a_{mid}, $N = 64$ bits.



(b) Fine-tuning for a_{mid}, $N = 128$ bits.



(c) Fitted polynomial of Figure 3.3 (a), $N = 64$ bits.



(d) Fitted polynomial of Figure 3.3 (b), $N = 128$ bits.

Figure 3.3 : Fine-tuning for a_{mid} values, and polynomial fitting. (a) and (b) are statistical analyses for the best-uncorrelated bitstreams of $X1$ and $X2$. The deviation from a_{mid} in the z-axis direction indicates the required compensation to fine-tune a_{mid} values. Each bar plot in (a) and (b) holds all combinations of input scalars on behalf of N-bit SC representation, yielding $(N+1)\times(N+1)$ total cases from the Cartesian product. CT can easily calculate these values and quickly test all possibilities. Evaluation of all Cartesian couples becomes impossible for long sequences in a software environment when actual bitstreams are processed. The approximations of (a) and (b) using an analogy of elliptic paraboloids are depicted in (c) and (d). The 4^{th}-order fitted polynomial in the form of $z = c_1 + c_2X1 + c_3X2 + c_4X1^2 + c_5X1X2 + c_6X2^2 + c_7X1^3 + c_8X1^2X2 + c_9X1X2^2 + c_{10}X2^3 + c_{11}X1^4 + c_{12}X1^3X2 + c_{13}X1^2X2^2 + c_{14}X1X2^3 + c_{15}X2^4$ (any c_n is a coefficient) is obtained using the *MATLAB Curve Fitting Toolbox*. The fitted polynomial is utilized to roughly fine-tune the CT primitive, $a_{poly} \leftarrow z + a_{mid}$, as an inline function.

Lemma 3.1 (Near-Zero SCC Condition)**.** *Let $SCC = 0$; therefore, the SCC numerator is also $ad - bc = 0$, which yields $a = \lfloor \frac{X1 \times X2}{N} \rceil$.*

Proof. Assuming bold notations represent the bitstreams of $X1$ and $X2$ as $\boldsymbol{X1}$ and $\boldsymbol{X2}$ (noting that $\sum_{i=1}^{N} \boldsymbol{X1}(i) = X1$ and $\sum_{i=1}^{N} \boldsymbol{X2}(i) = X2$):

CT primitives have been defined in Figure 3.1:

$a = \sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i)$

$b = \sum_{i=1}^{N} \boldsymbol{X1}(i) \times (1 - \boldsymbol{X2}(i))$

$c = \sum_{i=1}^{N} (1 - \boldsymbol{X1}(i)) \times \boldsymbol{X2}(i)$

$d = \sum_{i=1}^{N} (1 - \boldsymbol{X1}(i)) \times (1 - \boldsymbol{X2}(i))$.

To optimize SCC around zero, the numerator, $ad - bc$, in the following formula:

$$SCC = \begin{cases} \frac{ad-bc}{N \times min(a+b,a+c) - (a+b) \times (a+c)} & , if\ ad > bc \\ \frac{ad-bc}{(a+b) \times (a+c) - N \times max(a-d,0)} & , else \end{cases}$$

which has the same numerator on both piecewise counterparts, is taken into account. That is, $ad - bc = 0$ is compelled for the $SCC = 0$. By recalling the notations above for a, b, c, and d; the numerator is rewritten:

$ad - bc = \left(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) \right) \times \left(\sum_{i=1}^{N} (1 - \boldsymbol{X1}(i)) \times (1 - \boldsymbol{X2}(i)) \right) - \left(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times (1 - \boldsymbol{X2}(i)) \right) \times \left(\sum_{i=1}^{N} (1 - \boldsymbol{X1}(i)) \times \boldsymbol{X2}(i) \right)$.

For the ad part, by performing the multiplication related to d:

$$\begin{aligned} ad = & \left(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) \right) \times \\ & \left(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) - \sum_{i=1}^{N} \boldsymbol{X1}(i) - \sum_{i=1}^{N} \boldsymbol{X2}(i) + \sum_{i=1}^{N} 1 \right) \end{aligned} \tag{3.1}$$

For the bc part, by performing the multiplication both in b and c:

$bc = \left(\sum_{i=1}^{N} \boldsymbol{X1}(i) - \sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) \right) \times \left(\sum_{i=1}^{N} \boldsymbol{X2}(i) - \sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) \right)$

Multiplying elements by distributing over parentheses:

$bc = \left(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \sum_{i=1}^{N} \boldsymbol{X2}(i) \right) - \left(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) \right) - \left(\sum_{i=1}^{N} \boldsymbol{X2}(i) \times \sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) \right) + \left(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) \right)^2$

Simplifying this expression using the common term $\left(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) \right)$:

$$bc = \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \sum_{i=1}^{N} \boldsymbol{X2}(i)\Big) + \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i)\Big) \times \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) - \sum_{i=1}^{N} \boldsymbol{X1}(i) - \sum_{i=1}^{N} \boldsymbol{X2}(i)\Big) \tag{3.2}$$

Now performing $ad - bc$ (i.e., equation (3.1) - equation (3.2)) as:

$$\begin{aligned} ad - bc = {} & \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i)\Big) \times \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) - \sum_{i=1}^{N} \boldsymbol{X1}(i) - \sum_{i=1}^{N} \boldsymbol{X2}(i) + \sum_{i=1}^{N} 1\Big) - \\ & \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \sum_{i=1}^{N} \boldsymbol{X2}(i)\Big) - \\ & \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i)\Big) \times \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) - \sum_{i=1}^{N} \boldsymbol{X1}(i) - \sum_{i=1}^{N} \boldsymbol{X2}(i)\Big) \end{aligned} \tag{3.3}$$

In equation (3.3), the first line and the third line are almost identical except for the last term in the first line, $\sum_{i=1}^{N} 1$. Let us multiply this term by $\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i)$, yielding $\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) \times \sum_{i=1}^{N} 1$, where $\sum_{i=1}^{N} 1 = N$. Except for this multiplication and $\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \sum_{i=1}^{N} \boldsymbol{X2}(i)$ in the second line of equation (3.3), the rest cancels out:

$$\Big(\Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i)\Big) - \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) - \sum_{i=1}^{N} \boldsymbol{X1}(i) - \sum_{i=1}^{N} \boldsymbol{X2}(i)\Big)\Big) -$$
$$\Big(\Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i)\Big) - \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i) - \sum_{i=1}^{N} \boldsymbol{X1}(i) - \sum_{i=1}^{N} \boldsymbol{X2}(i)\Big)\Big) = 0$$

Finally, what is leftover:

$$ad - bc = \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i)\Big) \times N - \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \sum_{i=1}^{N} \boldsymbol{X2}(i)\Big)$$

By forcing $ad - bc = 0$ for $SCC \approx 0$, and placing the negative term into the other side of the equation:

$$\Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i)\Big) \times N = \Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \sum_{i=1}^{N} \boldsymbol{X2}(i)\Big)$$

Since $\Big(\sum_{i=1}^{N} \boldsymbol{X1}(i) \times \boldsymbol{X2}(i)\Big) = a$, $\sum_{i=1}^{N} \boldsymbol{X1}(i) = X1$, and $\sum_{i=1}^{N} \boldsymbol{X2}(i) = X2$, it is:

$$a = \frac{X1 \times X2}{N}. \tag{3.4}$$

Since each CT element $a, b, c, d \in \mathbb{Z}_0^+$, equation (3.4) is finally rounded to the nearest integer, $a_{best} = \lfloor \frac{X1 \times X2}{N} \rceil$.

□

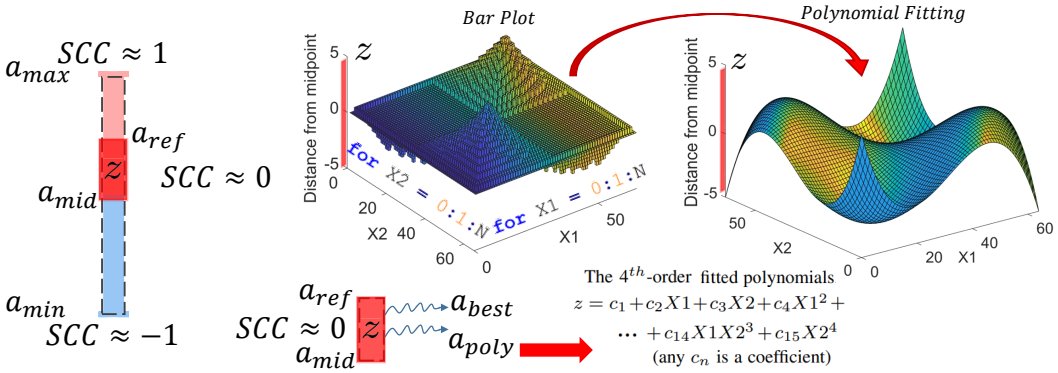


Figure 3.4 : Significant points between $SCC \approx 1$ and $SCC \approx -1$.

Algorithm 2 CT Model of Uncorrelated Bitstreams

Require: $X1$, $X2$, N $\{X1 = a + b,\ X2 = a + c,\ N = a + b + c + d\}$
Ensure: CT $\{CT = a, b, c, d\}$
1: $a_{max} \leftarrow MIN(X1, X2)$
2: $a_{min} \leftarrow MAX(0, X1 + X2 - N)$
3: $a_{mid} \leftarrow \lfloor (a_{max} + a_{min})/2 \rfloor$
4: **if** *midpoint approximation approach* **then**
5: $\quad a \leftarrow a_{mid}$ $//a_{mid}$
6: **else if** *polynomial fitting approach* **then**
7: $\quad a \leftarrow \lfloor 3D\ polynomial\ function(X1, X2, N)\ +\ a_{mid} \rceil$ $//a_{poly}$
8: **else if** *correlation coefficient optimization approach* **then**
9: $\quad a \leftarrow \lfloor \frac{X1 \times X2}{N} \rceil$ $//a_{best}$
10: **end if**
11: $b \leftarrow X1 - a;\quad c \leftarrow X2 - a;\quad d \leftarrow N - (a + b + c)$
12: **return** CT

Finally, Figure 3.4, which summarizes all methods, and Algorithm 2 with hierarchical computation flow, are presented. The vertical reference line in Figure 3.4 summarizes the critical points between the two extremes, a_{max} and a_{min}, and their relations with each other. The exact midpoint is specified as a_{mid}, and the point z away from a_{mid} (can be up or down) is designated a_{ref} (i.e., the a CT primitive that confirms the best near-zero *SCC* result). Figure 3.4 illustrates that the z-distance in red contains the values of a_{poly} and a_{best} other than a_{mid}. It is reminded how the z distance is located in both bar-plot and fitted polynomial, including how the x- and y-axis are sourced according to the Cartesian of input scalars. In addition, the hierarchical flow of the parameters calculated step by step in Algorithm 2 is given to understand how they affect each other. After obtaining a_{max} and a_{min} from the input scalars according

to the formulation in Figure 3.2, the a CT primitive assignment for each method is summarized, respectively.

3.5 Memory- and Runtime-Aware Calculation

The CT enables SC-based operations over CT primitives, thereby reducing memory consumption and bit-based operation latency. As there are no bitwise logic operations in the proposed simulation approach, only scalars are processed. There are 2^{2N} possible bitstream pair combinations, which is the count of all possible two input bitstreams' combinations from $0000\ldots0$ to $1111\ldots1$ (i.e., $\boldsymbol{X1}$ and $\boldsymbol{X2}$ overlapping from $0000\ldots0$ & $0000\ldots0$ to $1111\ldots1$ & $1111\ldots1$). Nevertheless, many overlapped pairs form the same CT. The total number of all bitstream combinations yielding the same CT is required for error operations and population calculations. Therefore, the "occurrence calculation" of any CT is proposed. The proposal for the occurrence of a CT (O_{CT}) is to answer: "How many other bitstream combinations does the same CT emulate?" The answer to this question, i.e., the determination of O_{CT}, helps to shed light on three purposes in the simulations:

PURPOSE – i: No physical bit-by-bit combinations, but a simple formula.

PURPOSE – ii: The population of each correlation value is calculated easily during error inspection.

PURPOSE – iii: During "all bitstreams combinations," O_{CT} is a tool to measure each CT's error proportion via its population: How many combinations does any CT bring over 2^{2N} total combinations?

The first purpose highlights the necessity of CT as a memory-efficient approach. Without the requirement of the physical bitstream and its bit-by-bit combinations, the total number of combinations is calculated rapidly. All the combinations in the software environment must be formulated for rapid calculation since bit-by-bit combinations take too long. Figure 3.5 shows how CT removes the penalty of this "combination calculation" by just calculating a, b, c, and d. Second, the "multiplication error" and "correlation population (occurrence)" plots (will be given in Section 3.6) require the O_{CT} during comprehensive experiments. Any correlation formula is nothing but the linear arrangement of a, b, c, and d; that means the

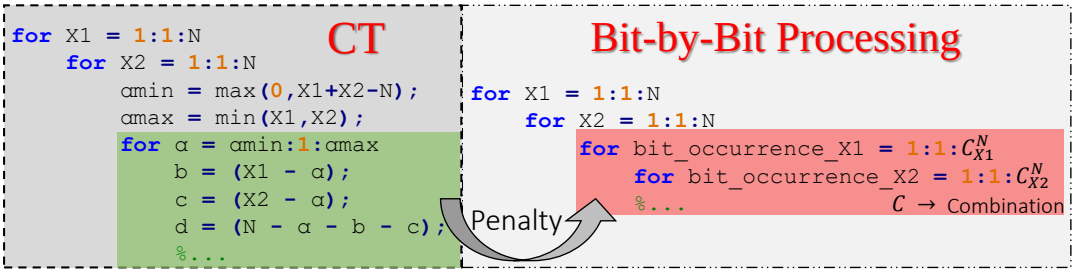


Figure 3.5 : Penalty avoidance thanks to CT.

correlation and *CT* relate to each other. For any CT, correlation frequency is monitored and this helps to understand the relationship between error and population for any SC-based arithmetic. Without O_{CT}, calculations become impossible for "all" 2^{2N} physical bitstream combinations, especially for $N \geq 32$ bits. There exists a bottleneck in terms of runtime and memory overhead. Lastly, for stochastic circuit error measurement, the proportion of each CT is important in the error calculations. All bitstream combinations (2^{2N}) are sourced to the circuit (not like Monte Carlo simulations), and the effect of each CT in terms of its contributing error should be known. Thankfully, O_{CT} gives this value for each CT during all combinations.

The formulation of O_{CT} is based on the combination of each CT primitive. Depending on the cumulative values of each bit intersection, i.e., a, b, c, and d, their possible combinations in an overlapped bitstream should be calculated. By utilizing the combination (C), each CT primitive is evaluated in order. In Figure 3.6, the $C(N,a) \times C(N-a,b) \times C(N-a-b,c) \times C(N-a-b-c,d)$ expression is the starting formula based on the combinations. This is derived by assigning positions to each primitive starting from a. Since some primitives (those whose *value* > 1, such as $a = 2$) may repeat in an overlapped bitstream, permutations with repetition are considered. Repeated primitives do not affect the combination since the interchanges between them still define the same combination. After assigning primitive a to the possible positions of the overlapped stream, the leftover primitives are also assigned to the remaining positions. When the starting formula in Figure 3.6 is expanded, the final notation is derived as $O_{CT} = \frac{N!}{a! \times b! \times c! \times d!}$, which Meyer refers to as "*permutations when not all objects are different*" in his book [168, p. 30]. In other words, the total permutation of the N bits is divided by the possible repetitions of each CT primitive. The example presented in Figure 3.6 shows that three possible bitstream combinations from a given

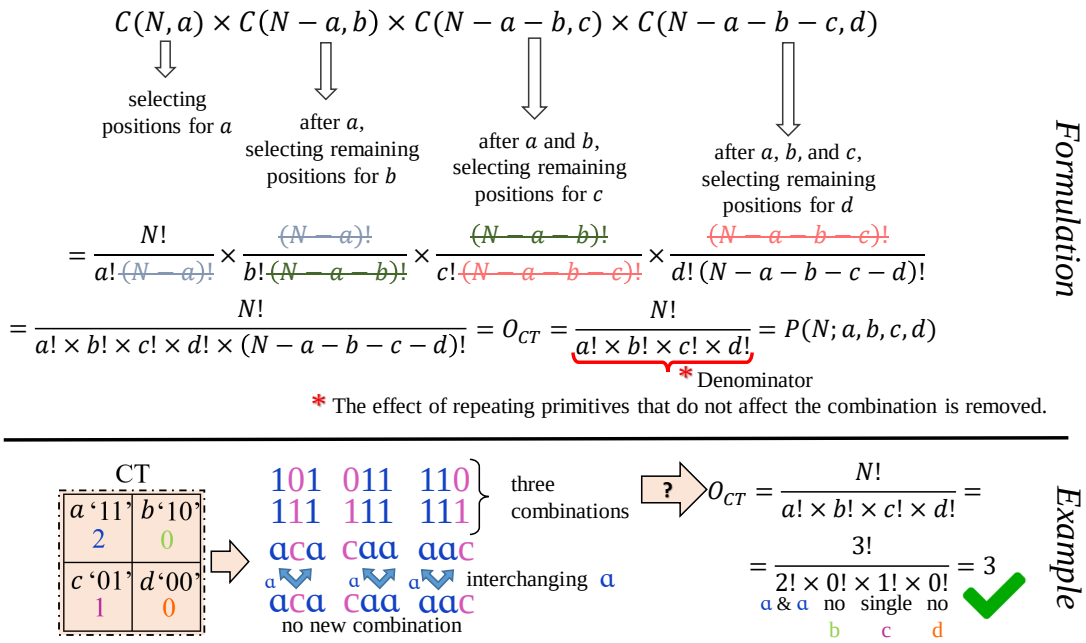


Figure 3.6 : Formulation of O_{CT} and an example.

CT can occur, which is calculated with O_{CT}. It should be noted that the interchange between a and a does not introduce a new combination, and this situation is considered in the formula.

3.6 The Error, Occurrence, and Efficiency Experiments via CT

Furthermore, the experiments focus on the error analysis for SC-based multiplication using AND and XNOR by addressing all input combinations. The CT has less algorithm complexity than the bit-by-bit processing and quickly calculates the logic value of the relevant inputs via TCO over the unique cases of a, b, c, and d. Correlation values based on the binary similarity equations in Table 3.2 are determined. O_{CT} calculates the number of different binary pairs that indicate the same correlation value (i.e., the same CT) and cumulatively contributes to the error. Thus, the relationship between the population of the correlation coefficient and the error also emerges. For the error metric, the root-mean-square error: $RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{y_i})^2}$ is utilized, where n is the total count of any coefficient occurrence from different CTs. The TCO of the SC-based logic gate operation is retrieved from Table 3.1. This obtained value is denoted by $\widehat{y_i}$. The expected value is based on the deterministic arithmetic operation. This conventional multiplication of input probabilities is denoted as y_i.

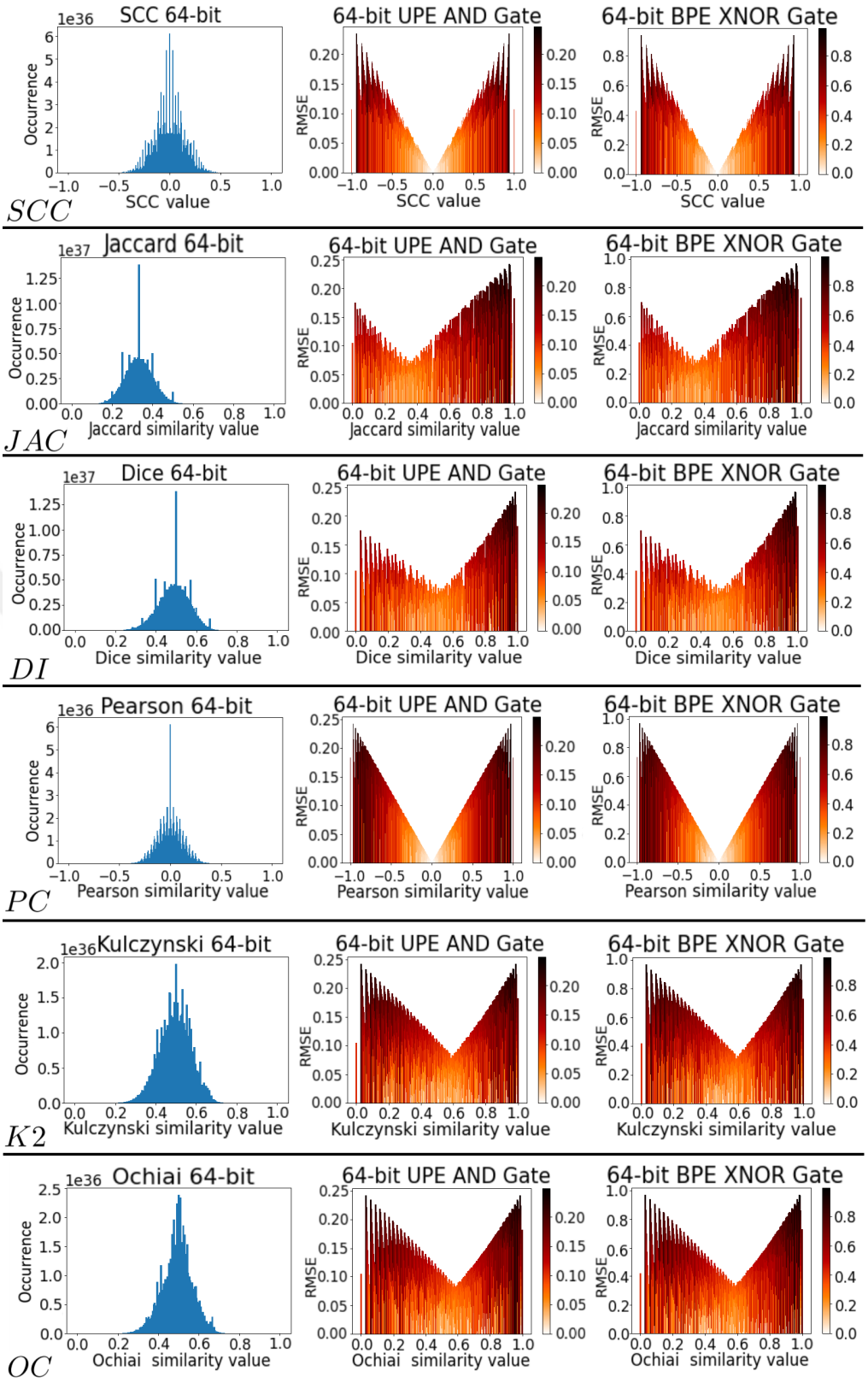


Figure 3.7 : Occurrence of the correlation coefficients and SC-based multiplication error for $N = 64$. The similarity metrics in Table 3.2 are utilized. The occurrence distribution of each metric is shown in the first blue plot. Then, the multiplication error is depicted using the RMSE in the red heat-map plots. For each, the first is the UPE-based AND gate multiplier, and the second is the BPE-based XNOR gate multiplier. CT is utilized in the application of all input combinations.

Table 3.3 : MAE of the proposed CT approaches for zero correlation.

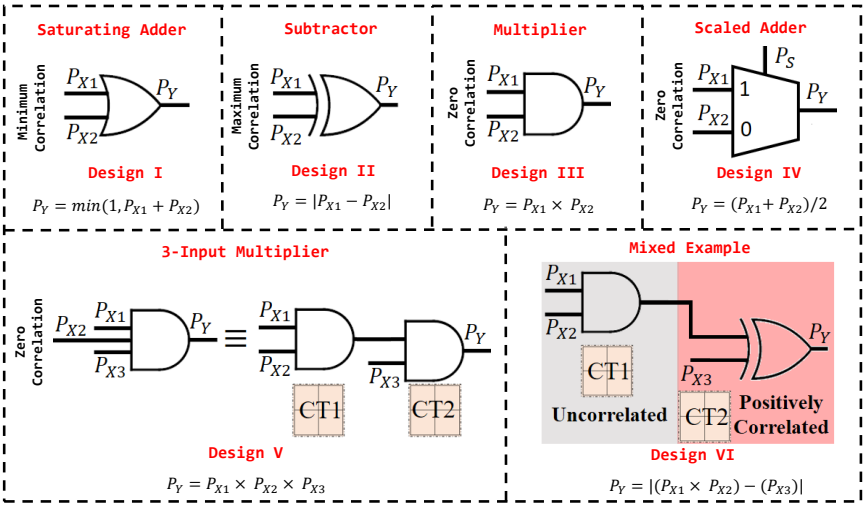
N	8	16	32	64	128	256
$MAE_{a_{mid}}$	0.681	0.542	0.409	0.325	0.277	0.253
$MAE_{a_{poly}}$	0.489	0.456	0.332	0.263	0.224	0.205
$MAE_{a_{best}}$	0.288	0.283	0.229	0.160	0.105	0.065

Figure 3.7 presents the analyses for each correlation equation. The RMSEs of both UPE-based AND and BPE-based XNOR multipliers are calculated and presented as red heat-map plots. The blue plots show the total number of contributing input combinations calculated via O_{CT} corresponding to each correlation value. Except for $K2$ and OC, a larger correlation coefficient frequency from binary pairs corresponds to a smaller SC-based arithmetic operation error. Moreover, SCC is underlined as a commendable metric for symmetric distribution and zero convergent error behavior. This result and the hardware-efficient model of SCC-based bitstream generators by Alaghi and Hayes [38] make the SCC a research-worthy topic. SCC and PC, which have a $[-1,1]$ coefficient range, are preferable in terms of SC performance monitoring. By contrast, other measures with a $[0,1]$ coefficient range fall behind, with asymmetry in terms of the high-frequency and low-error coefficient observations [169].

The proposed algorithm is also measured for efficiency. The ideal a_{ref} values, which are depicted in Figure 3.4, are first utilized to calculate $SCC_{a_{ref}}$ for each scalar input combination. Then, the methods proposed in Algorithm 2 are performed to calculate all SCC values of $n = (N+1) \times (N+1)$ scalar combinations. In Table 3.3, the error performance of the proposed methods evaluating $SCC_{a_{mid}}$, $SCC_{a_{poly}}$, and $SCC_{a_{best}}$, is revealed using the mean absolute error: $MAE_{a_{proposed}} = \frac{1}{n}\sum_{i=1}^{n} |SCC_{a_{ref}}^{(i)} - SCC_{a_{proposed}}^{(i)}|$. As expected, the values obtained from a_{best} are less erroneous.

In addition, some basic SC-based arithmetic operations [40] are analyzed for appropriate $SCC \approx 1$, -1, or 0. In Table 3.4, the behavior of the related correlation for each proposed circuit structure in the literature is tested for all input scalar combinations using CT. Our proposed simulation offers comparable error performance related to the references, which suggest *mixed-integer programming* [103] and *stochastic synthesis* [170]. Table 3.4 exemplifies multi-input gate simulation and mixed correlation arrangement in a cascaded logic structure.

Table 3.4 : MAE performance of the CT approaches for SC-based arithmetic.



Design	**I**	**II**
N	a_{min}	a_{max}
8	0	0
16	0	0
32	0	0
64	0	0
128	0	0
256	0	0
512	0	0
1024	0	0
Ref.	[103] MAE=0 → N=16...128	[170] MAE=0, N=N/A

Design	**III**			**IV**		
N	a_{mid}	a_{poly}	a_{best}	a_{mid}	a_{poly}	a_{best}
8	0.03086	0.03086	0.02160	0.05555	0.05555	0.05555
16	0.02400	0.02400	0.01297	0.02941	0.02941	0.02941
32	0.02140	0.01525	0.00711	0.01515	0.01515	0.01515
64	0.02074	0.01233	0.00372	0.00769	0.00769	0.00769
128	0.02067	0.01102	0.00190	0.00387	0.00387	0.00387
256	0.02071	0.01070	0.00096	0.00194	0.00194	0.00194
512	0.02076	0.01061	0.00048	0.00097	0.00097	0.00097
1024	0.02079	0.01058	0.00024	0.00048	0.00048	0.00048
Ref.	[103] MAE=0.016, 0.009, 0.005, 0.003 → N=16, 32, 64, 128 [170] MAE=0.021, N=N/A			[103] MAE=0.016, 0.008, 0.004, 0.002 → N=16, 32, 64, 128 [170] MAE=0.027, N=N/A		

Design	**V**			**VI**		
N	a_{mid}	a_{poly}	a_{best}	a_{mid}	a_{poly}	a_{best}
8	0.03800	0.03800	0.02647	0.03086	0.03086	0.02160
16	0.02867	0.02867	0.01498	0.02391	0.02391	0.01297
32	0.02454	0.01741	0.00800	0.02116	0.01520	0.00711
64	0.02341	0.01447	0.00414	0.02043	0.01226	0.00372
128	0.02328	0.01288	0.00211	0.02033	0.01094	0.00190
256	0.02335	0.01246	0.00107	0.02037	0.01061	0.00096
512	0.02343	0.01235	0.00053	0.02042	0.01052	0.00048
1024	0.02348	0.01232	0.00027	0.02045	0.01050	0.00024
Ref.	Our own example			Our own example		

3.7 Random Source Emulation via CT

The random fluctuations occur in SC depending on the process of the probabilistic bitstreams. The motivating problem is the inclusion of fluctuation error only over CT-based scalars without physically generating the bitstreams. In the literature, Alaghi clarifies the error categories in SC [33]. The random fluctuation is explained based on the random procedure applied during the bitstreams generated. In [171], Baker and Hayes underline how bit-by-bit processing is important as other analytic solutions underestimate several circuit characteristics like expected output error. Pure CT simulation around zero correlation can cause the same underestimation without the inclusion of random fluctuation error. In the literature, Ma et al. propose a generalized method to increase the expected accuracy of output by considering unknown distributions of incoming bitstreams [172]. They call these bitstreams "black bitstreams," and only the expected value and variance are known. By using hypergeometric decomposition, Ma et al. propose a new variance transfer to better simulate SC. Similarly, in this dissertation, three different random source generators are emulated using CT, considering the random error deviations.

Binomial Distribution. First, the random fluctuation error of binomially distributed bitstreams is included in the CT simulation approach. A more realistic model is obtained as if physical bit-by-bit processing is performed. Alaghi defines the random fluctuation errors using the *estimated* and *exact* values of stochastic logic. By many of the previous efforts [33, 57], a stochastic bitstream having probability P is considered as a set of samples from RV with a Bernoulli distribution having a success probability, P. Thus, considering the *Independent and Identically Distributed* (i.i.d.) Bernoulli RV, a stochastic number is represented yielding binomial distribution with a variance $\sigma^2 = P(1-P)/N$. Random fluctuations happen intrinsically as a built-in outcome during the stochastic number generation [104,173]. Random errors adversely influence the stochastic arithmetic accuracy. The *estimated* values differ from the *exact* values leading to some accuracy drops. Therefore, even though the near-zero correlation is simulated, the inclusion of random fluctuation error is a must. Considering an AND gate multiplier with two bitstream inputs, $\boldsymbol{X1}$ and $\boldsymbol{X2}$, the *exact* value of the

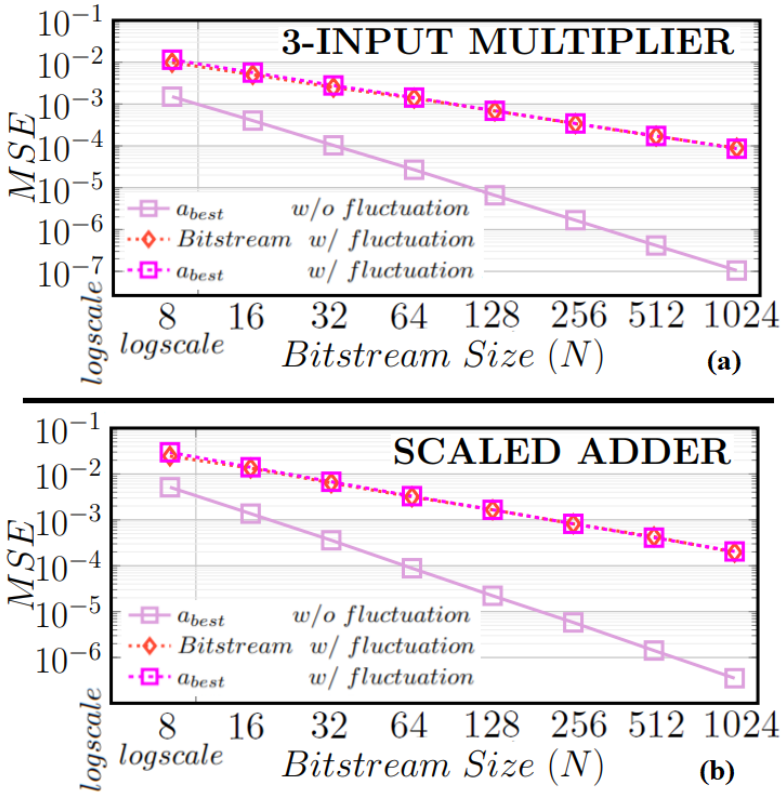


Figure 3.8 : MSE comparison of bit-by-bit bitstream processing and CT approach considering the random fluctuation error: (a) 3-input multiplier and (b) scaled adder.

multiplication, $P_Y = \mathbb{E}[\boldsymbol{Y}]$, is given as $\mathbb{E}[\boldsymbol{X1} \times \boldsymbol{X2}] = \mathbb{E}[\boldsymbol{X1}] \times \mathbb{E}[\boldsymbol{X2}] = P_{X1} \times P_{X2} = P_Y$, where $\mathbb{E}$ is the expectation operator and the inputs are assumed to be independent variables. Nevertheless, the *estimated* value at the output, $\hat{P}_Y$, may have been altered due to random fluctuations, and the difference between P_Y and $\hat{P}_Y$ is counted as a random fluctuation error, evaluated by the use of mean squared error (MSE) $err_Y = \mathbb{E}[(P_Y - \hat{P}_Y)^2]$ yielding $err_Y = P_Y(1 - P_Y)/N$ when it comes to Bernoulli RVs [33]. Advantageously, the error does not depend on the circuit type and is only based on the exact value of the output and N. For the inclusion of random errors in CT simulation, err_Y is considered. For example, the MSE of an AND gate, together with the random fluctuation error, forms into $MSE = \frac{1}{n}\sum_{i=1}^{n}((P_Y - a_{best}/N)^2 + err_Y)$. The a_{best}/N (i.e., TCO/N) term is the obtained value from CT in the case of $SCC \approx 0$, and n is the number of random experiments.

In Figure 3.8, experiments related to the 3-input multiplier and scaled adder circuitry are presented based on $n = 5000$ trials. The effect of random fluctuation is observed according to the deviation from a_{best} (a_{best} w/o $fluctuation$). The inclusion

of random fluctuation error in CT (a_{best} $w/$ $fluctuation$) perfectly matches the physical bit-by-bit processing of bitstreams ($Bitstream$ $w/$ $fluctuation$) having binomial distribution. Figure 3.8 also underscores how the error in stochastic circuits generally decreases depending on the increasing value of N.

Sobol. Second, it is proposed that a_{best} is an actual model of the Sobol sequence-based quasi-random bitstream generation using the LD sequence approach. In the literature [174, 175], Sobol-based random number generation is a recent method that yields high accuracy in terms of stochastic arithmetic. Najafi et al. propose a novel architecture for "accurate" bitstream generators using LD sequences. Their state-of-the-art paper [56] shows how rapidly stochastic operations can be performed with accurate results. Sobol-based bitstreams present deterministic-like arithmetic accuracy if long enough bitstreams are utilized. For example, the accurate results of multiplying two n-bit precision data can be achieved using $N \times N$ Sobol bitstreams where $N = 2^n$. Since $a_{best} = \lfloor \frac{X1 \times X2}{N} \rceil$ is obtained from $SCC = 0$ optimization in the near-zero CT-based approach that guarantees high accuracy, Sobol-based results are expected to be similar.

LFSR. Third, LFSR via CT is modeled considering the hypergeometric distribution indicated by Baker and Hayes [171]. They show that LFSR-based bitstreams fit better to hypergeometric RV bitstream generation than the binomial distribution. They define the output deviation of the bitwise AND operation on LFSR-based bitstreams as $\sigma = \sqrt{\frac{P_{X1} \times P_{X2} \times (1-P_{X1}) \times (1-P_{X2})}{N-1}}$. Since AND output is related to the a primitive, this output deviation is used after setting up the CT via a. For the near-zero CT, a_{best} must be updated, so the output probability model for bitwise ANDing with LFSR-based bitstreams becomes $\frac{a_{best}}{N} + \sqrt{\frac{P_{X1} \times P_{X2} \times (1-P_{X1}) \times (1-P_{X2})}{N-1}}$.

Tests for these three random source emulations will be given at the image processing application level in Section 4.2.4 and Section 4.2.5, considering runtime and memory.

4. HARDWARE-INSPIRED IMAGE PROCESSING

SC, as a re-emerging technique, has been explained in previous chapters for the simulation perspectives. With the aid of the probability definition, SC is an unconventional way to represent the processing operands approximately; hence, it is well-suited to vision applications, for which human perception is not very sensitive to approximations. Besides, the built-in probabilistic behavior of SC brings promising outcomes such as image acquisition noise resistance. In this chapter, an SC-based, i.e., hardware-inspired, image processing framework is presented. A broader perspective on SC-based image processing from the simulation aspect is drawn with respect to the prior art. Mainly, kernel-based image processing operations are deepened, and (i) mean filtering, (ii) edge detection, (iii) template matching, (iv) bilinear interpolation, and (v) image compositing are defined in the frame of SC. Proposed filter models, simulations, and noise robustness of SC-based image processing are demonstrated. Tests show that hardware-inspired image processing mostly outperforms in the case of noise injection.

4.1 From Deterministic Pixels to Randomized Pulses

Within the scope of the image processing concept, the data must be defined first. Grayscale images are used in this thesis and have $\kappa = 8$-bit precision unless otherwise stated. In Definition 4.1, the *random pixel* concept is presented.

Definition 4.1 (Random Pixel)**.** Let $I_{(r\times c)}$, size of $r \times c$, be a grayscale image (or a single channel of an RGB image) to be processed. The two-dimensional (2D) I at each *r*ow-*c*olumn position is further processed to be transformed into the 3D data structure, s.t. $I_{(r\times c)} \rightarrow \boldsymbol{I}_{(r\times c\times N)}$, where N is the bitstream size. This conversion is performed using probability $P = \frac{pixel\ value}{N}$, and UPE (unless otherwise stated) can simply be utilized since the value of a pixel at (x,y) position meets $I(\mathrm{x},\mathrm{y}) \in \mathbb{Z}_0^+$ condition in Definition 2.4 via $I(\mathrm{x},\mathrm{y}) \in \{0,...,2^{\kappa-bit}-1\},\ \kappa \in \mathbb{N}^+$. Finally, $\boldsymbol{I}$ is obtained considering each of the pixel randomized using any of the methods presented in Section 2.2.2.

The newly represented $\boldsymbol{I}$ has a binary value at any row-column position, (x,y), and at any bit, m, s.t. $\boldsymbol{I}(\mathrm{x},\mathrm{y},m) \in \mathbb{B}$. Regarding Definition 4.1, the probability P from any pixel value should be carefully and separately treated for *maximum pixel value* $> N$ or *maximum pixel value* $< N$ cases. For the latter, pixel values are up-scaled, while for the former, pixel values are down-scaled. The min-max normalization is performed by assigning $2^{\kappa-bit} \leftarrow N$ for both cases.

After having defined $\boldsymbol{I}$, the data are ready to be processed for bitwise operations. The digital circuit model of the desired operation is determined using bitwise operations in software. Therefore, SC-based model performance is measured using the bitwise circuit model, as illustrated in Figure 4.1. There are two more processing environments, pure software, MATLAB, and pure hardware, FPGA, for comparison purposes. Throughout this chapter, their corresponding results are used as references.

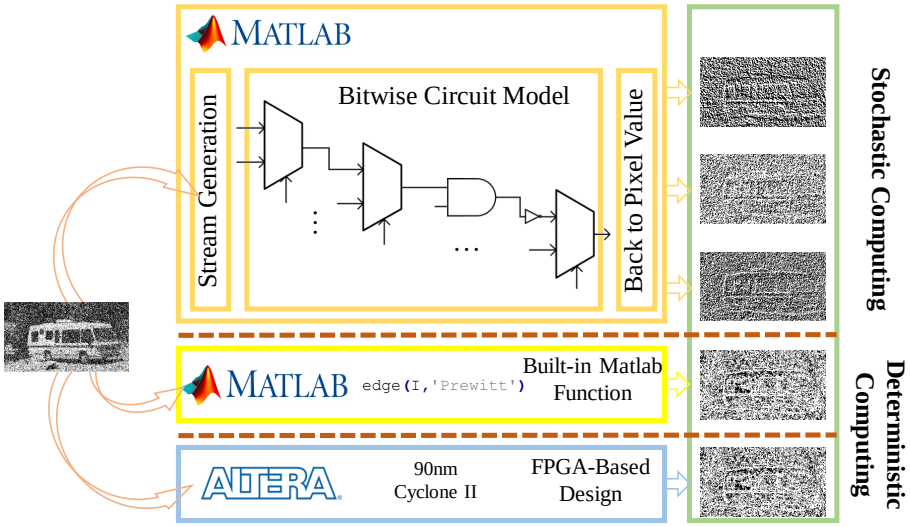


Figure 4.1 : The methodology to process images using logical operators with the idea of SC.

4.2 Stochastic Computing Image Processing

Lately, SC has been applied to image processing applications. This dissertation uses software-driven SC to demonstrate how to interfere with the hardware design concerns for the image processing applications. The bottom-up design process employs atomic building components to create kernel-based image processing operators. The kernel, alternatively known as convolution or mask matrix, is a small matrix used to slide over the main image for applications such as noise removal, edge detection, template

matching, image scaling, etc. In Figure 4.2, an example kernel is shown for $k = 3$ size. It is noted that K_9 is illustrated as a center pixel of the kernel, and $K_1 ... K_8$ are the neighboring pixels.

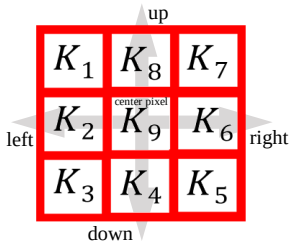


Figure 4.2 : An example of 3×3 kernel.

The K_{any} value, shown in Figure 4.2, represents the filter coefficients and is based on the desired operation. The kernel, K, which is slid over an image considering the central pixel, can be interpreted in the frame of a convolution operation such as $I * K$. I is an image in the process. The kernel has up-down and left-right movements leading to the 2D convolution. In Figure 4.3, the conventional convolution operation and the corresponding SC update are given. In addition to kernel size (k) and kernel coefficients (K_{any}), the kernel scaling weight, w_k, is a parameter to multiply kernel coefficients. For instance, the kernel values may be considered in the frame of an averaging operation by setting w_k as a fractional value: $w_k = 1/k^2$, for k-to-k kernel.

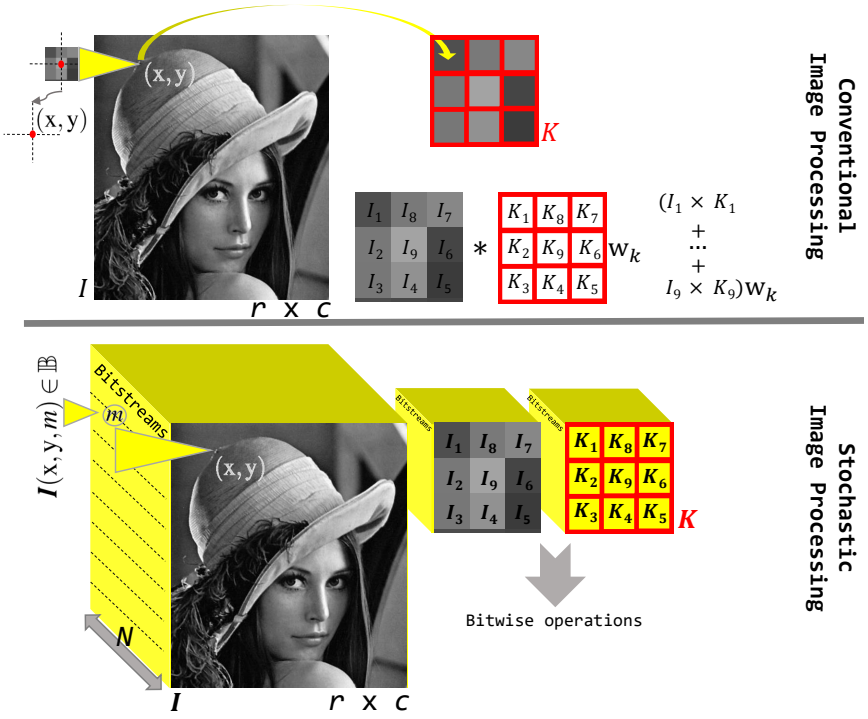


Figure 4.3 : From conventional to stochastic image processing.

4.2.1 Noise removal: Reinterpretation of mean filtering

Random bitstream representation of data supports the noise-robust solutions. In this section, a brief introduction to SC-based image filtering is given. Mean filtering is known for smoothing the noise, such as acquisition noise on the images. In Definition 4.2, the definition of conventional mean filtering is given.

Definition 4.2 (Mean Filtering)**.** Several pixel values on a sliding window, i.e., kernel, captured from an image, are averaged using summation and division. The $k \times k$ kernel takes the neighboring pixels into account at any x, y row-column position on an image. Single channel values (or simply grayscale pixel values) are then processed considering the convolution. Setting K with "1" scalar kernel coefficients and $w_k = 1/(k \times k)$, the new value of a center pixel is obtained as $\frac{(I_1 \times 1)+(I_2 \times 1)+...+(I_{(k \times k)} \times 1)}{(k \times k)}$.

Since the frequently utilized filter size in the literature is $k = 3$, after this point, the definitions/tests will be set accordingly.

When it comes to the SC-based mean filtering, an architecture is given in Figure 4.4. Representing circuit inputs via $\boldsymbol{X}s$ (which are fed by pixel values), nine-pixel inputs for $k = 3$ are denoted using $\boldsymbol{X}_{(1..9)}$ for UPE-based random bitstreams. The SC-based mean filter uses scaled adders (explained in Chapter 2), which have built-in scaling factors from the selection input. The design presented in Figure 4.4 is a 2-level MUX-based architecture, and the selection inputs ($\boldsymbol{S}$) need to be cleverly set. All in all, the expected output is as in equation (4.1) in terms of output probability.

$$P_{\boldsymbol{Y_2}} = \frac{(P_{\boldsymbol{X_1}} + P_{\boldsymbol{X_2}} + P_{\boldsymbol{X_3}} + P_{\boldsymbol{X_4}} + P_{\boldsymbol{X_5}} + P_{\boldsymbol{X_6}} + P_{\boldsymbol{X_7}} + P_{\boldsymbol{X_8}} + P_{\boldsymbol{X_9}})}{9} \tag{4.1}$$

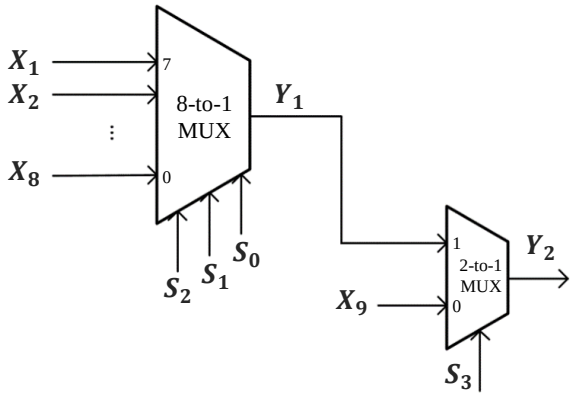


Figure 4.4 : The architecture of 3×3 stochastic mean filter.

Proof. (of equation (4.1)). From the 8-to-1 MUX, the probability-related SC-based definition is as follows:

$P_{Y_1} = (1 - P_{S_2})(1 - P_{S_1})(1 - P_{S_0})P_{X_8} + (1 - P_{S_2})(1 - P_{S_1})(P_{S_0})P_{X_7} + (1 - P_{S_2})(P_{S_1})(1 - P_{S_0})P_{X_6} + (1 - P_{S_2})(P_{S_1})(P_{S_0})P_{X_5} + (P_{S_2})(1 - P_{S_1})(1 - P_{S_0})P_{X_4} + (P_{S_2})(1 - P_{S_1})(P_{S_0})P_{X_3} + (P_{S_2})(P_{S_1})(1 - P_{S_0})P_{X_2} + (P_{S_2})(P_{S_1})(P_{S_0})P_{X_1}$

The selection probabilities, P_{S_0}, P_{S_1}, and P_{S_2}, are set to $1/2$ regarding the first level MUX architecture in Figure 4.4, and the following is obtained:

$P_{Y_1} = (\frac{1}{8})P_{X_8} + (\frac{1}{8})P_{X_7} + (\frac{1}{8})P_{X_6} + (\frac{1}{8})P_{X_5} + (\frac{1}{8})P_{X_4} + (\frac{1}{8})P_{X_3} + (\frac{1}{8})P_{X_2} + (\frac{1}{8})P_{X_1}$ $= (\frac{1}{8})(P_{X_8} + P_{X_7} + P_{X_6} + P_{X_5} + P_{X_4} + P_{X_3} + P_{X_2} + P_{X_1})$ After that, $P_{S_3} = 8/9$ is set for the "division to 9" using the denominator, and canceling 8 on the numerator in P_{Y_1}. Thus, the following is written for the second level of MUX structure considering the 2-to-1 architecture:

$P_{Y_2} = P_{Y_1}P_{S_3} + P_{X_9}(1 - P_{S_3}) = \frac{1}{8}(P_{X_1} + P_{X_2} + P_{X_3} + P_{X_4} + P_{X_5} + P_{X_6} + P_{X_7} + P_{X_8})\frac{8}{9}$ $+P_{X_9}(1 - \frac{8}{9}) = \frac{P_{X_1}+P_{X_2}+P_{X_3}+P_{X_4}+P_{X_5}+P_{X_6}+P_{X_7}+P_{X_8}}{9} + \frac{P_{X_9}}{9}$

Finally equation (4.1) is concluded as follows:

$P_{Y_2} = \frac{(P_{X_1}+P_{X_2}+P_{X_3}+P_{X_4}+P_{X_5}+P_{X_6}+P_{X_7}+P_{X_8}+P_{X_9})}{9}$

□

In Figure 4.5, the result of 3×3 SC-based mean filtering is shown. The original image in Figure 4.5 (a) is processed by injecting noise (Gaussian noise with $\mu = 0$ and $\sigma^2 = 0.008$), and the result is Figure 4.5 (b). When the architecture in Figure 4.4 is applied for noise reduction, Figure 4.5 (c) is the output image giving a 23.39 PSNR value.

(a) **(b)** **(c)**

Figure 4.5 : (a) Original grayscale image, (b) applied Gaussian noise with $\mu = 0$ and $\sigma^2 = 0.008$, and (c) output of the stochastic mean filter applied to the noisy image with $N = 512$ bitstream size.

4.2.2 Edge detection: On the decision of approximation

In order to give a different perspective to the SC-based edge detection literature, an approximation comparison of a custom kernel is presented using the power of software simulation. The conventional (deterministic, binary) design in MATLAB platform (software-based) and in FPGA (hardware-based) is performed for reference purposes. Therefore, there are three main design approaches: (i) pure software-PS (conventional) design of a filter in MATLAB, (ii) pure hardware-PH (conventional) design of a filter in FPGA, and (iii) the SC-based filter (unconventional) design considering different approximations. It is not very cost-effective to have physical hardware at the very beginning of design steps regarding the accuracy-aware solutions. Therefore, simulation as a cost-free choice is a unique alternative to see the design effects beforehand, especially in applications such as image processing. At this point, the overall motivation is to see different design approximations of the SC-based kernel design for edge detection. The edge detection case study, which considers a noisy environment, is presented with three different hardware architectures showing the effect of the scaling factor of a MUX. In the results, detailed outcomes are comparatively given by utilizing PSNR and structural similarity (SSIM) [176], together with HRU.

The SC-based edge detection design effort is handled for the Prewitt filtering, so the conventional version of this filter type is defined first. Different conventional edge detection methods have been named for their own inventors, such as Canny [177], Deriche [178], Sobel [179], Roberts cross [180], and Prewitt [181]. These methods depend on the filter size (k), and coefficients ($K_{1...k^2}$). In the literature, SC-based Prewitt filter design is a relatively less studied approach [112] compared to the other methods such as the Sobel edge detector [114, 182, 183] and the Roberts cross operator [10, 75, 100, 115, 184, 185]. Therefore, the Prewitt kernel is contextualized from a wide perspective. Considering the differentiation property (derivatives) between image pixels, the sliding windows (namely kernels or spatial masks) are utilized horizontally and vertically. In Figure 4.6, the conventional Prewitt filter kernel is shown, and explained in Definition 4.3.

Definition 4.3 (Prewitt Filtering)**.** The Prewitt filter is defined using horizontal and vertical image gradients. Let the first kernel, $K^{(1)}$, have the coefficients, $K_1^{(1)} = -1$, $K_2^{(1)} = -1$, $K_3^{(1)} = -1$, $K_4^{(1)} = 0$, $K_5^{(1)} = 1$, $K_6^{(1)} = 1$, $K_7^{(1)} = 1$, $K_8^{(1)} = 0$, $K_9^{(1)} = 0$ as in Figure 4.6 (a), and let the second kernel, $K^{(2)}$, be $K_1^{(2)} = -1$, $K_2^{(2)} = 0$, $K_3^{(2)} = 1$, $K_4^{(2)} = 1$, $K_5^{(2)} = 1$, $K_6^{(2)} = 0$, $K_7^{(2)} = -1$, $K_8^{(2)} = -1$, $K_9^{(2)} = 0$ in Figure 4.6 (b). The kernel scaling weight is $\mathrm{w}_k = 1$ for both. Sliding these Ks separately on an image, I, yields horizontal and vertical gradients. Thus, the Prewitt filter captures the edges on an image in both x, y directions.

−1	0	1
−1	0	1
−1	0	1

(a)

−1	−1	−1
0	0	0
1	1	1

(b)

Figure 4.6 : Prewitt filter kernels: (a) mask for the vertical edges, and (b) mask for the horizontal edges.

To extend the definition above in terms of the image gradients, the grayscale image, I, is defined using the spatial x, y notation of pixel positions, s.t. $I = f(\mathrm{x}, \mathrm{y})$. This 2D data, in (x, y) coordinates, is processed via defined kernels. Thus, image gradient, ∇f, is noted using partial derivatives, $G_\mathrm{x} = \frac{\partial f}{\partial \mathrm{x}}$ and $G_\mathrm{y} = \frac{\partial f}{\partial \mathrm{y}}$, regarding the applied kernels in Figure 4.6 (a) and Figure 4.6 (b), respectively. Hence, G_x and G_y can be defined in equation (4.2) and equation (4.3), considering the relative pixel positions $\mathrm{x} \pm 1$, $\mathrm{y} \pm 1$:

$$\begin{aligned} G_\mathrm{x} = {} & [f(\mathrm{x}+1, \mathrm{y}+1) + f(\mathrm{x}+1, \mathrm{y}) + f(\mathrm{x}+1, \mathrm{y}-1)] \\ & -[f(\mathrm{x}-1, \mathrm{y}+1) + f(\mathrm{x}-1, \mathrm{y}) + f(\mathrm{x}-1, \mathrm{y}-1)] \end{aligned} \tag{4.2}$$

$$\begin{aligned} G_\mathrm{y} = {} & [f(\mathrm{x}-1, \mathrm{y}-1) + f(\mathrm{x}, \mathrm{y}-1) + f(\mathrm{x}+1, \mathrm{y}-1)] \\ & -[f(\mathrm{x}-1, \mathrm{y}+1) + f(\mathrm{x}, \mathrm{y}+1) + f(\mathrm{x}+1, \mathrm{y}+1)]. \end{aligned} \tag{4.3}$$

From gradient calculation, the combined results are obtained by applying the magnitude of ∇f denoted as in equation (4.4).

$$|\nabla f| = \sqrt{(G_\mathrm{x}{}^2 + G_\mathrm{y}{}^2)} \tag{4.4}$$

In [186], Kho et al. underline that sum of absolute values of the gradients is approximately utilized without square root and the squares, as also stated in [187–189].

Therefore, $|\nabla f| \approx |G_x| + |G_y|$ is used instead of equation (4.4) since the squaring and square root implementation add more complexity to the hardware design (regarding both conventional and SC-based design [47]).

On the other hand, for SC-based Prewitt kernel design, three design perspectives using different approximation scales are set. Separating from the previous studies, measuring the effect of *multiplication approximation* versus *exactness* is the main motivation. The concepts of "accurate" and "approximate" terms refer to the degree of multiplication approximation.

In Figure 4.7, proposed design approaches are presented. At each design, the single gradient calculation is aimed using three-level cascaded MUX structures. In Figure 4.7 (a), the first design is the plainest approach without having a multiplier. The second approach, in Figure 4.7 (b), stands for interfering with the scaled adder scaling factor by using a well-known SC-based multiplier, the AND gate. The accurate scaling factor multiplication is finally applied in the third design by having an exact multiplier for SC-based bitstream on the utilization of shifter & repeater modules, as shown in Figure 4.7 (c). Vahapoğlu and Altun proposed an accurate multiplier with shifter & repeater modules. They explained how to have an exact multiplication when SC bitstreams are employed for the AND gate [190]. Before sourcing the AND, input bitstreams (size of N) are pre-processed for exact multiplication. One bitstream is copied and concatenated repeatedly by N times, while the other bitstream is shifted and concatenated N times. Then, both are processed in an AND gate, having an $N \times N$-size output bitstream. Actually, the proposal of Vahapoğlu and Altun supports the idea of Sobol sequence-based accurate operations [44]. The exact multiplication result via shifter and repeater is also obtained in Sobol sequences when the bitstreams of two operands are generated from the first and second Sobol sequences. The generated bitstreams are simulated as if coming from the shifter and repeater [44, 45, 56].

All three designs affect the scaling factor in the cascaded MUXs in different ways: either it is not intervened at all, or the scaling factor effect is emphasized with approximate multiplication or exact multiplication. Let us now define each approach by presenting the related notations and formulations in Definition 4.4, Definition 4.5, and Definition 4.6.

Definition 4.4 (Plain Design (PD))**.** The cascaded MUX construct in Figure 4.7 (a) is a model for the gradient value, $(X_5 + X_6 + X_7) - (X_1 + X_2 + X_3)$, where each X indicates the scalar value of a corresponding neighbor pixel in I. Like in Figure 4.4, the outputs in the second stage, Y_1 and Y_2, are defined considering their related probability values (thereby, the bitstreams denoted: $\boldsymbol{Y_1}$, $\boldsymbol{Y_2}$), and MUX equations become (see equation (2.4) again) $P_{\boldsymbol{Y_1}} = [P_{\boldsymbol{S_1}} \times \big((P_{\boldsymbol{S_0}} \times P_{\boldsymbol{X_1}}) + (1 - P_{\boldsymbol{S_0}}) \times P_{\boldsymbol{X_2}}\big)] + [(1 - P_{\boldsymbol{S_1}}) \times P_{\boldsymbol{X_3}}]$ and $P_{\boldsymbol{Y_2}} = [P_{\boldsymbol{S_1}} \times \big((P_{\boldsymbol{S_0}} \times P_{\boldsymbol{X_5}}) + (1 - P_{\boldsymbol{S_0}}) \times P_{\boldsymbol{X_6}}\big)] + [(1 - P_{\boldsymbol{S_1}}) \times P_{\boldsymbol{X_7}}]$.

By choosing the probabilities as $P_{\boldsymbol{S_0}} = 1/2$ and $P_{\boldsymbol{S_1}} = 2/3$, the second stage MUX probabilities become:

$$P_{\boldsymbol{Y_1}} = \tfrac{2}{3}\big(\tfrac{(P_{\boldsymbol{X_1}} + P_{\boldsymbol{X_2}})}{2}\big) + \tfrac{P_{\boldsymbol{X_3}}}{3} = \tfrac{(P_{\boldsymbol{X_1}} + P_{\boldsymbol{X_2}})}{3} + \tfrac{P_{\boldsymbol{X_3}}}{3}$$

$$P_{\boldsymbol{Y_1}} = \tfrac{P_{\boldsymbol{X_1}} + P_{\boldsymbol{X_2}} + P_{\boldsymbol{X_3}}}{3}$$

$$P_{\boldsymbol{Y_2}} = \tfrac{(P_{\boldsymbol{X_5}} + P_{\boldsymbol{X_6}} + P_{\boldsymbol{X_7}})}{3}$$

At the final stage of the PD-based gradient calculation, even though the bitstreams are in UPE, the scaled subtraction is applied, and the following result is obtained:

$$P_{\boldsymbol{Y_3}} = \tfrac{(P_{\boldsymbol{Y_2}} - P_{\boldsymbol{Y_1}})}{2} + \tfrac{1}{2} = P^{(PD)}_{\boldsymbol{Y_3}} = \Big(\tfrac{\big(\frac{(P_{\boldsymbol{X_5}} + P_{\boldsymbol{X_6}} + P_{\boldsymbol{X_7}}) - (P_{\boldsymbol{X_1}} + P_{\boldsymbol{X_2}} + P_{\boldsymbol{X_3}})}{3}\big)}{2}\Big) + \tfrac{1}{2}.$$

Definition 4.5 (Semi-Plain Design (SPD))**.** By considering the outcome of Definition 4.4, the scale factor effector, Γ, (as a UPE bitstream, $\boldsymbol{\Gamma}$) is utilized, affecting the probability: $P^{(SPD)}_{\boldsymbol{Y_3}} = \Big(\tfrac{\big(\frac{(P_{\boldsymbol{X_5}} + P_{\boldsymbol{X_6}} + P_{\boldsymbol{X_7}}) - (P_{\boldsymbol{X_1}} + P_{\boldsymbol{X_2}} + P_{\boldsymbol{X_3}})}{3}\big)}{2}\Big) \tilde{\times} P_{\boldsymbol{\Gamma}} + \frac{1}{2}$ as in Figure 4.7 (b). The multiplication has multiplicands: $P_{\boldsymbol{\Gamma}}$ and $P_{\boldsymbol{Y}}$. They are fed into the AND gate that approximately multiplies the inputs. Since the bitstream correlation is not controlled, the operation is kept approximate. $P_{\boldsymbol{\Gamma}}$ is set roughly to increase the effect of "3" at the denominator of the numerator in $P^{(SPD)}_{\boldsymbol{Y_3}}$.

Definition 4.6 (Multiplication Smooth Design (MSD))**.** Finally, the exact multiplier (still working on SC bitstreams) for MSD is utilized in Figure 4.7 (c). Revisiting the proposal of Vahapoğlu and Altun [190], the AND gate in UPE-based multiplication takes the shifted-and-repeated version of the input bitstreams. Thus, an exact multiplication is supplied with a single gate. The final output bitstream probability is again expected as $P^{(MSD)}_{\boldsymbol{Y_3}} = \Big(\tfrac{\big(\frac{(P_{\boldsymbol{X_5}} + P_{\boldsymbol{X_6}} + P_{\boldsymbol{X_7}}) - (P_{\boldsymbol{X_1}} + P_{\boldsymbol{X_2}} + P_{\boldsymbol{X_3}})}{3}\big)}{2}\Big) \times P_{\boldsymbol{\Gamma}} + \frac{1}{2}$, while increasing the effect of "3" at the denominator more precisely.

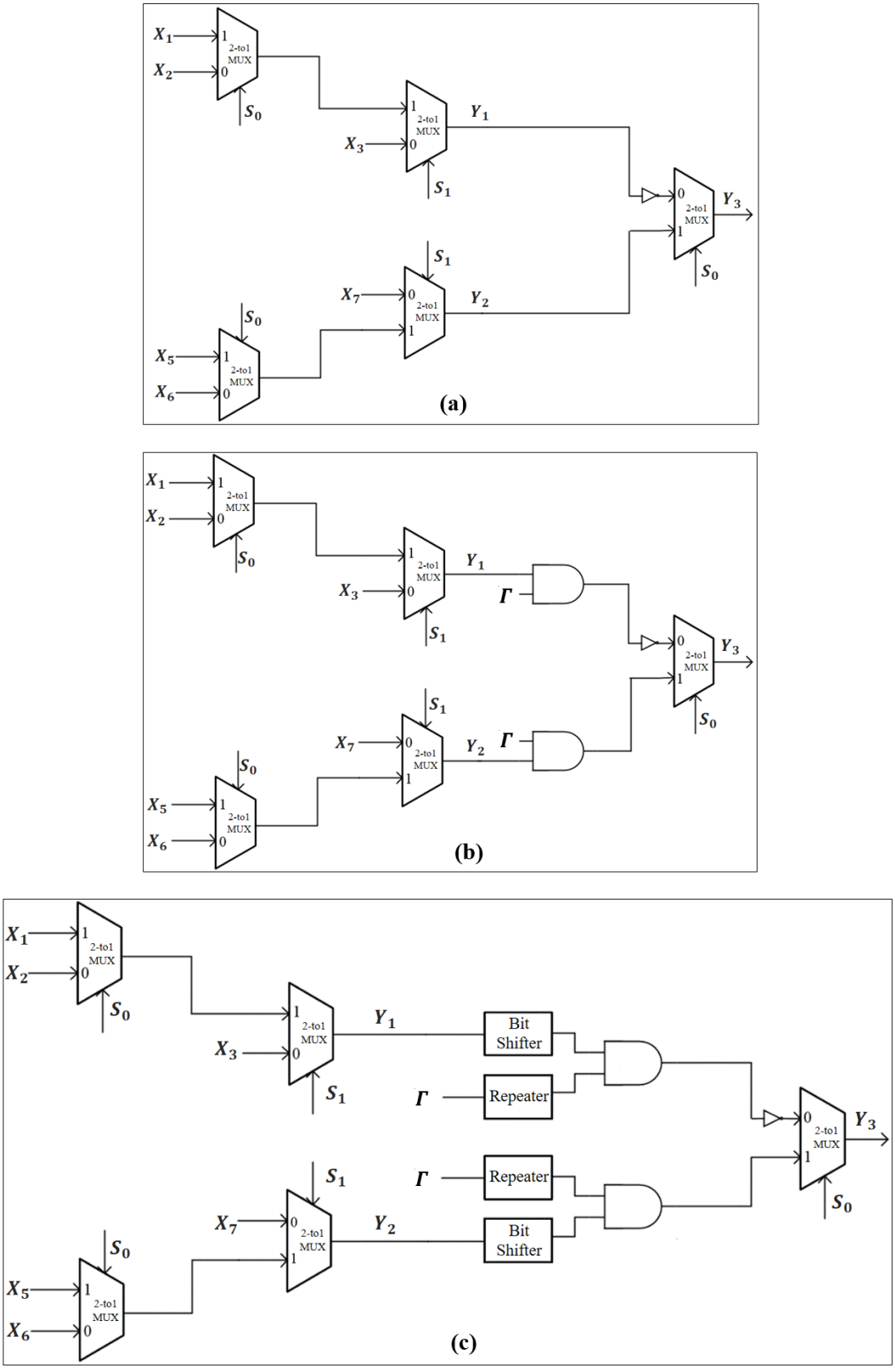


Figure 4.7 : Comparative proposed SC designs for single gradient calculation (G_x, i.e., $[(X_5+X_6+X_7)-(X_1+X_2+X_3)]$). (a) PD without any multiplier, (b) SPD with an approximate SC-based multiplier, and (c) MSD design with an exact SC-based multiplier.

In Figure 4.8, the details of designs described in Definition 4.4, Definition 4.5, and Definition 4.6 are summarized together with depictions of the fundamental differences between architectures.

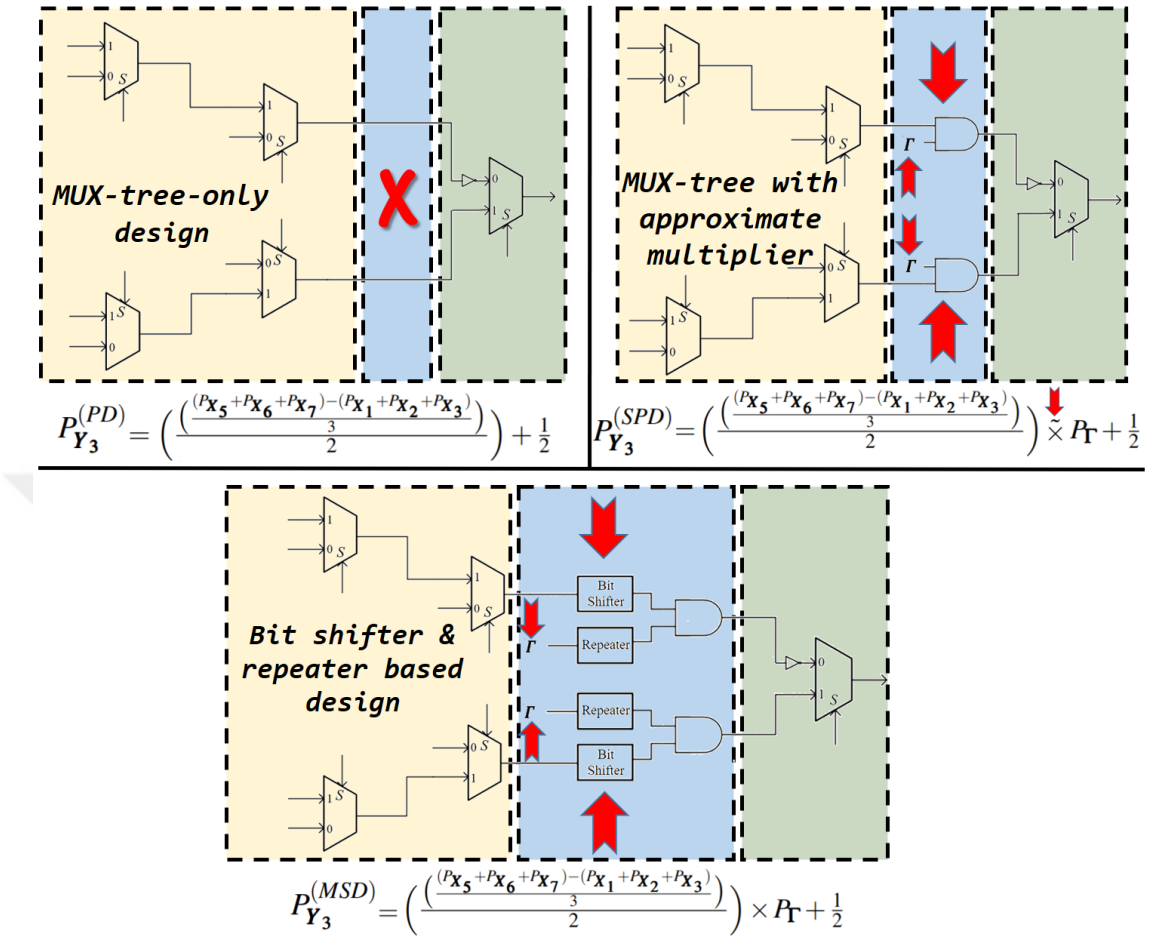


Figure 4.8 : Summary of the SC-based Prewitt edge detector architectures.

The visual output of each design proposal is presented in Figure 4.9. After separately calculating each gradient, G_x, G_y, via the architectures given in Figure 4.8, the final-level 2-to-1 MUX is applied to obtain $|\nabla f| \approx |G_x| + |G_y|$. The final scaled adder yields approximately $(G_x + G_y)/2$. In Figure 4.9, comparative visual results are given on the different scales of injected Gaussian noise. Images are presented after optimal thresholding for binarization to underline the relative performance. Software- (PS) and hardware-related (PH) binary computing approaches offer similar outputs as expected and are used as the reference for performance monitoring. SC-based PD, SPD, and MSD are treated as software-driven stochastic circuits in the frame of unconventional computing.

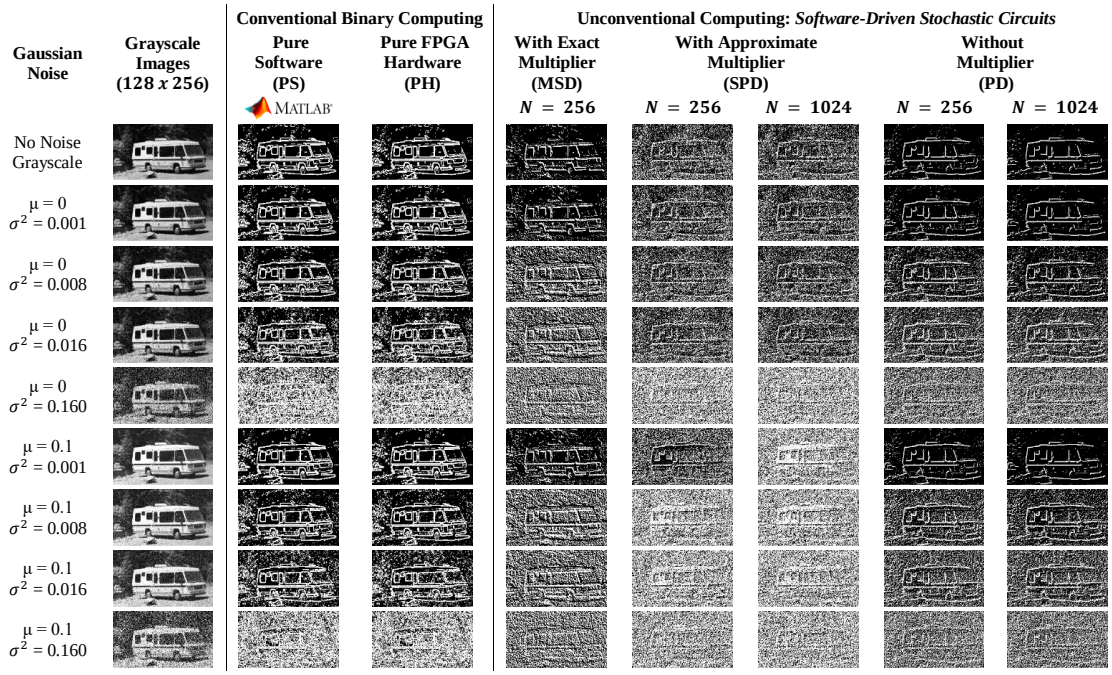


Figure 4.9 : Prewitt edge detection visual results on different noise levels.

In this part, comparison metrics are performed on the visual results in Figure 4.9. PSNR (like in previous efforts [5,183,191,192]) and SSIM are utilized for performance monitoring. The reference comparison is performed using the deterministically processed reference no-noise image. In Figure 4.10, the SSIM and PSNR results are plotted. The primary motivation is to compare two metrics in SC-based system design. SSIM is a measure to compare two images to indicate the affinity in between by outputting a value in the $[0,1]$ range (with 0 meaning they are different and with 1 meaning completely the same). In Figure 4.10 (a), PS and PH perform slightly higher than the SC-based architectures, except at the high noise injection. PD is the leading among SC architectures and gives the best SSIM when excessive noise is injected. In Figure 4.10 (b), conventional binary computing results, PS and PH, have lower performance (at each noise level) than the PD and MSD SC architectures. While SPD design is relatively stable and has low performance, PD outperforms compared to MSD with a similar noise response. As a result, SSIM and PSNR metrics may present different results. The similar behavior of the no-multiplier and with-smooth-multiplier designs underlines the insignificance of the MUX scaling factor for image processing regarding the PSNR.

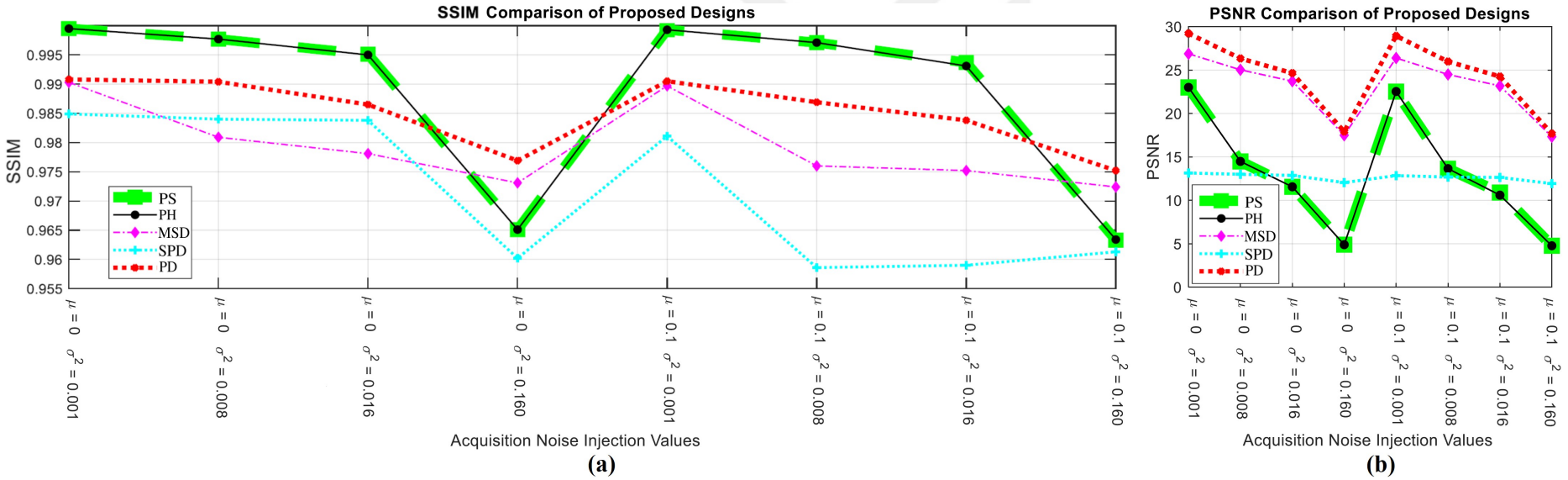


Figure 4.10 : Prewitt edge detection plots for (a) SSIM and (b) PSNR (dB) metrics.

Table 4.1 : Logic cell and gate counts.

PH	Software-Driven SC		
	MSD	SPD	PD
245 Logic Cells	44 Atomic Gates + 4 Shifters + 4 Repeaters	44 Atomic Gates	40 Atomic Gates

The final performance measure is more about hardware utilization. Table 4.1 compares the proposed SC-based edge detectors with the FPGA-based design (PH), accounting for both the G_x and G_y gradient calculation. PH is designed by customizing the logic cells to the minimum possible usage, and Altera DE2, Cyclone II, 90nm is the experiment platform. It should be noted that in SC designs, the stage-wise selection ports of MUXs are shared (see Figure 4.7). The lightest design, as expected, is PD with 40 atomic gates.

Furthermore, resource utilization in previous works is investigated in Table 4.2. For different kernel designs, such as Roberts cross, Gaussian, Sobel, and Prewitt, previous studies are listed considering the HRU for the single kernel (for instance, only for G_x). Roberts cross is lightweight if correlated inputs are in use, and Sobel has slightly more cost than Prewitt. This work presents an alternative design via PD having 5 $\times$ 2-to-1 MUX (with shared selection ports) and a NOT gate. Regarding Ranjbar et al.'s proposal [112], PD adds one more MUX but does not utilize XOR. Their proposal addresses a different Prewitt kernel, thereby having different kernel coefficient placement.

Table 4.2 : Single kernel (for one gradient) HRU comparison with the literature.

Ref.	**Implemented Kernel**	**HRU**
Li et al. [11, 184]	Roberts cross operator	2 $\times$ NOT gate, 3 $\times$ 2-to-1 MUX, 2 $\times$ SC abs. value function
Abdellatef et al. [118]	Gaussian	8 $\times$ 2-to-1 MUX
Najafi and Lilja [111]	Roberts cross operator (using correlated inputs)	2 $\times$ NOT, 2 $\times$ XOR, 1 $\times$ 2-to-1 MUX
Li et al. [115]	Roberts cross operator	110 NAND
Ranjbar et al. [112]	Prewitt	4 $\times$ 2-to-1 MUX, 1 $\times$ XOR gate
Joe and Kim [183]	Sobel	4 $\times$ 2-to-1 MUX, 1 $\times$ XOR gate, 2 $\times$ AND gate
This dissertation, PD	Prewitt	5 $\times$ 2-to-1 MUX, 1 $\times$ NOT gate

4.2.3 Template matching: Pattern detection of quick response codes

In SC systems, there are different efforts on kernel-based image processing [10, 11, 15, 41, 115, 193]; however, this section introduces SC-based template matching for the first time in the literature, to the best of our knowledge. This study shows how SC can be adapted for pattern detection and can also be a tool for noisy environments with an application using quick response (QR) codes. A template, K, is looked up throughout an image, I, using a sliding window. The formal definition of conventional template matching is given in Definition 4.7.

Definition 4.7 (Template Matching)**.** Let $I_{(r\times c)}$ be a grayscale image to be processed. The kernel $K_{(r_K\times c_K)}$ is a subset of I as a one-dimensional (1D) or 2D window, where $r_K < r$, $c_K < c$ (Either r_K or c_K is equal to 1 for a 1D window, but this work follows the 2D kernel). Like in edge detection, K is applied on I by processing each intersecting pixel of I and K. The objective is to catch the highest similarity and to get the exact template, i.e., kernel, match on I. The conventional operation is defined using $M = \sum f(I,K)$, where $f(I,K)$ can be any of the functions: (i) $f = |I - K|$, (ii) $f = (I-K)^2$, or (iii) $f = I \times K$.

In Definition 4.7, three of the conventional template matching approaches are given. The first is an absolute difference, and the temporary matching image, M, is set as the sum of the absolute differences (SAD). The second is the MSE-related formula, the sum of the squared differences (SSD) calculation, and the third is a cross-correlation-based calculation [194, 195]. This thesis uses (iii) for both conventional binary computing and SC-based simulation. Based on each method, M is processed to find minimum (valid if f in (i) or (ii) is utilized) or maximum-valued (valid if f in (iii) is applied) pixel positions that indicate the template coordinates. In (iii), the correlation is similar to the convolution without spatially flipping the template. Nevertheless, brighter pixels may cause a wrong template assignment in (iii), and "normalization" is proposed as a solution in the literature [195, 196]. If pixel operations are kept in the $[0, 255]$ range, the brighter pixel values (due to overflow of summation having > 255 values) may be erroneously assigned as the template, and the $[0, 1]$ normalized range is recommended. Lande et al. explain the correlation

property of SC using the AND operation [157]. The cross-correlation-based approach (iii) fits SC using the AND multiplier. Besides, in SC-based template matching while using (iii), a single AND operation, performing on probabilities approximately, brings the normalized fashion. Stefano et al. explain that normalization is performed by dividing the $\sum I \times K$ term with the multiplication of the main image and template L_2 norms: $|I|_2 \times |K|_2$ [197]. These terms are approximated as $N \times N$ coming from the denominator of $P_{\boldsymbol{I}} \times P_{\boldsymbol{K}}$ in SC.

Now, a new perspective is built on SC-based template matching for a visual QR code considering noise issues. QR codes are widely used in daily life, and the textual information is embedded using 2D visual images. Each code has three visually located finder patterns (identical) with a constant structure [198]. It can be underlined that the generated QR code is initially with black and white pixels [199] and is produced as a binary image. However, when captured by a camera, QR is obtained in RGB/grayscale format (colorful QR code generation is also possible [200]). After generating a binary QR code like in Step I of Figure 4.11, Gaussian noise is injected at Step II ($I \longrightarrow \hat{I}$, $K \longrightarrow \hat{K}$). Thus, a grayscale structure is obtained with a controlled noise, and more realistic images are provided as if captured with a camera [201]. This way of representation keeps us from dealing with geometric rectification and affine transform operations since the captured images require geometric pre-processing, which is out of our scope.

In Figure 4.11, Step III, algorithm steps are presented for SC. The AND operation is applied for multiplication ($P_{\hat{\boldsymbol{I}}} \times P_{\hat{\boldsymbol{K}}}$) after getting UPE-based bitstreams of $\hat{I}$ and $\hat{K}$. M is calculated to find the top-3 maximum values since each QR has three finder patterns (`FP`). In Step 3 of the algorithm, the maximum value at each iteration is substituted with the `NaN` value to be able to find out the second and the third maximums. Each matched pattern coordinate is recorded in `FP`. In Step 4, the constant pattern coordinates are compared with those obtained from the while-loop, `FP`, and at each "hit," the total pattern counter is increased, `pattern_Count++`.

Multiple QR codes are generated and evaluated for the algorithm in the testing phase. In Figure 4.11, the testing part shows how the *hit rate* (HR) is calculated for the accuracy measure. Over $n = 100$ samples, the matched pattern count is calculated in terms of percentages.

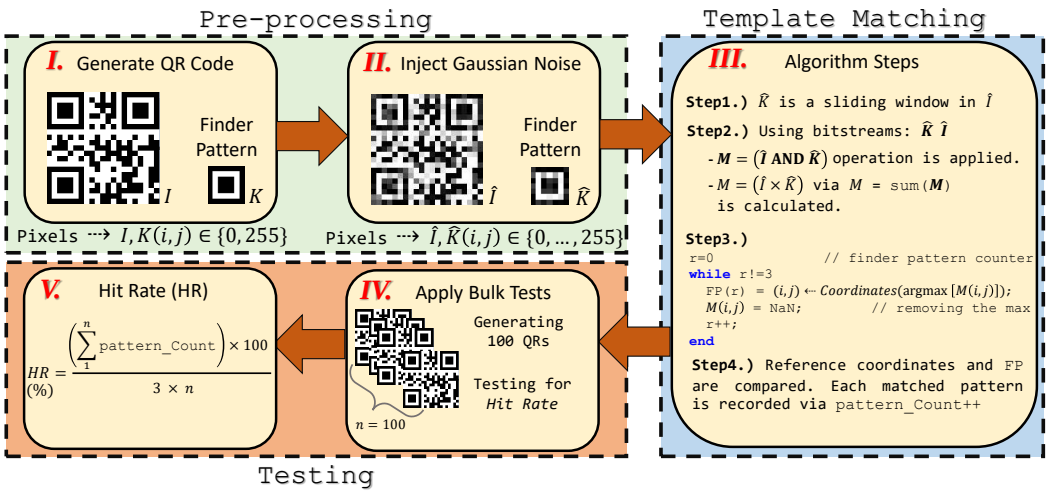


Figure 4.11 : The flow of the SC-based template matching for QR codes.

The test results in Figure 4.12 are given in terms of HR metric. The tests are discussed in two parts: (i) Deterministic (Det.): binary computing, and (ii) Stochastic (Sto.): random bitstreams in UPE. I and K images are tested with three different combinations of noise: $f(\hat{I},K)$, $f(I,\hat{K})$, and $f(\hat{I},\hat{K})$. Noise is applied in two types: Gaussian noise and speckle noise, considering noise parameters (μ,σ^2). Slight improvement in SC-based QR FP matching accuracy compared to the deterministic case is monitored.

$I_{21\times21}$	$K_{7\times7}$	Noise Type	$\hat{I}_{21\times21}$	$\hat{K}_{7\times7}$		Patterns Matched, HR (%) $\hat{I}, K$	$I, \hat{K}$	$\hat{I}, \hat{K}$
No Noise $I =$ 	No Noise $K =$	Gaussian Noise	μ=0.1 σ^2=0.3 $\hat{I} =$ 	μ=0.01 σ^2=0.03 $\hat{K} =$ 	Sto.	63 %	100 %	45.33 %
					Det.	60.33 %	100 %	41.66 %
			μ=0.1 σ^2=0.5 $\hat{I} =$ 	μ=0.01 σ^2=0.08 $\hat{K} =$ 	Sto.	41 %	74 %	19.66 %
					Det.	39 %	72.33 %	16 %
		Speckle Noise	σ^2=0.3 $\hat{I} =$ 	σ^2=0.6 $\hat{K} =$ 	Sto.	83 %	100 %	64.33 %
					Det.	79.66 %	98 %	61.33 %
			σ^2=0.5 $\hat{I} =$ 	σ^2=0.8 $\hat{K} =$ 	Sto.	64.66 %	97 %	53.33 %
					Det.	62.66 %	97 %	44.66 %

Sto. Stochastic (N=1024)

Det. Deterministic (Conventional Binary)

Figure 4.12 : SC-based QR FP matching results with Gaussian and speckle noise.

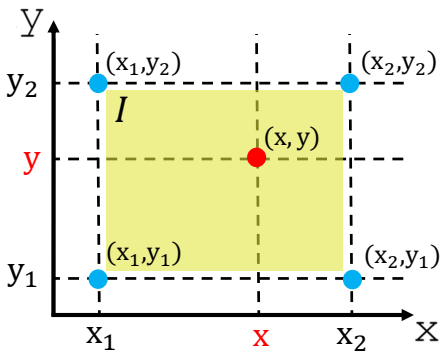


Figure 4.13 : Points in the coordinate system essential for bilinear interpolation.

4.2.4 Bilinear interpolation: New perspectives toward CT simulation-I

This section discusses the bilinear interpolation technique in image processing and the adaptation of SC. Meanwhile, the CT method proposed in Chapter 3 is reconsidered within the scope of image processing.

Bilinear interpolation is based on linear interpolations in both the `x` (width) and `y` (height) directions in the `xy` plane. With repetitive linear interpolations for `x` and `y` due to the row-column format for single-channel (grayscale) or multi-channel (RGB) images, bilinear interpolation, i.e., bilinear filtering, is obtained [202]. In Definition 4.8, the analytical definition of bilinear interpolation is presented.

Definition 4.8 (Bilinear Interpolation)**.** Let I be a 2D matrix with x and y row-column structure. Having defined 4 points in the coordinate system for I rectangular region in Figure 4.13, a new point, lying inside this rectangular region (region has vertices (x_1,y_1), (x_2,y_1), (x_1,y_2), and (x_2,y_2)) is denoted as (x,y). Based on the vertices of 2D I, whose coordinate positions hold any values (such as pixel values), $I(x,y)$ is subjected to estimation. After `x`- and `y`-related linear interpolations, the formula based on *bilinear interpolation* is denoted as:

$I(x,y) \approx a_{11}I(x_1,y_1)+a_{21}I(x_2,y_1)+a_{12}I(x_1,y_2)+a_{22}I(x_2,y_2)$, where

$a_{11} = [(x_2-x)(y_2-y)]/[(x_2-x_1)(y_2-y_1)]$,

$a_{21} = [(x-x_1)(y_2-y)]/[(x_2-x_1)(y_2-y_1)]$,

$a_{12} = [(x_2-x)(y-y_1)]/[(x_2-x_1)(y_2-y_1)]$, and

$a_{22} = [(x-x_1)(y-y_1)]/[(x_2-x_1)(y_2-y_1)]$ [203, 204].

By mapping I into the unit square via normalization of the values, the vertices are now $(\mathrm{x}_1 = 0, \mathrm{y}_1 = 0)$, $(\mathrm{x}_2 = 1, \mathrm{y}_1 = 0)$, $(\mathrm{x}_1 = 0, \mathrm{y}_2 = 1)$, and $(\mathrm{x}_2 = 1, \mathrm{y}_2 = 1)$ [205], thereby finally obtaining $I(\mathrm{x}, \mathrm{y})$ as:

$$\begin{aligned} I(\mathrm{x},\mathrm{y}) \approx \\ (1-\mathrm{x})(1-\mathrm{y})I(\mathrm{x}_1,\mathrm{y}_1)+ \\ (\mathrm{x})(1-\mathrm{y})I(\mathrm{x}_2,\mathrm{y}_1)+ \\ (1-\mathrm{x})(\mathrm{y})I(\mathrm{x}_1,\mathrm{y}_2)+ \\ (\mathrm{x})(\mathrm{y})I(\mathrm{x}_2,\mathrm{y}_2). \end{aligned} \tag{4.5}$$

Finally, with $I_{(r\times c)}$, representing a grayscale (or 3-channel RGB) image with r rows and c columns, the enlarged image, $\bar{I}_{(\rho r\times \rho c)}$, is obtained after the bilinear filtering, where ρ is the scaling factor. In Definition 4.9, bilinear interpolation-based image filtering is defined.

Definition 4.9 (Bilinear Interpolation-Based Image Filtering)**.** Let $I_{(r\times c)}$ be a grayscale image (or a single channel of an RGB image). A new image, $\bar{I}_{(\rho r\times \rho c)}$, is generated with ρ times updated size compared to I. Let us revisit and update equation (4.5), $I(\mathrm{x},\mathrm{y}) = (1-dx)(1-dy)I_{11} + (1-dx)(dy)I_{12} + (dx)(1-dy)I_{21} + (dx)(dy)I_{22}$, where $I_{11}, I_{12}, I_{21}, I_{22}$ are neighboring pixel values, and dx, dy are the relative positions (with respect to vertices) of the new pixel $I(\mathrm{x},\mathrm{y})$ being estimated. The relative positions are in multiples of $\frac{1}{\rho}$: $n \times \frac{1}{\rho}$, where positive integer $n \in \{1, ..., \rho\}$.

After these conventional definitions, the stochastic bilinear interpolation for image filtering is defined in Definition 4.10.

Definition 4.10 (Stochastic Bilinear Interpolation-Based Image Filtering)**.** Let $I_{(r\times c)}$ be a grayscale image (or a single channel of an RGB image). A new image, $\bar{I}_{(\rho r\times \rho c)}$, is generated with ρ times updated size compared to I. Using UPE-based bitstream representation, each pixel value and relative dx, dy positions are represented in bitstreams. The probabilities of neighboring pixels and relative positions are denoted as $P_{\boldsymbol{I_{11}}}$, $P_{\boldsymbol{I_{12}}}$, $P_{\boldsymbol{I_{21}}}$, $P_{\boldsymbol{I_{22}}}$, $P_{\boldsymbol{dx}}$, and $P_{\boldsymbol{dy}}$. By using the updated formula in Definition 4.9, the stochastic expression is yielded as $P_{\boldsymbol{I}(\mathrm{x},\mathrm{y})} = (1-P_{\boldsymbol{dx}})(1-P_{\boldsymbol{dy}})P_{\boldsymbol{I}_{11}}$ $+(1-P_{\boldsymbol{dx}})(P_{\boldsymbol{dy}})P_{\boldsymbol{I}_{12}} + (P_{\boldsymbol{dx}})(1-P_{\boldsymbol{dy}})P_{\boldsymbol{I}_{21}} + (P_{\boldsymbol{dx}})(P_{\boldsymbol{dy}})P_{\boldsymbol{I}_{22}}$.

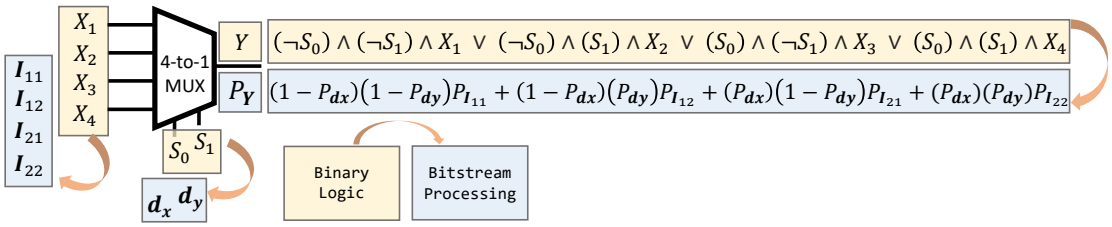


Figure 4.14 : From binary logic to the bitstream processing of 4-to-1 MUX.

At this point, a topology is proposed with an interesting analogy between hardware and the image processing operation: "Implementing bilinear interpolation with 4-to-1 MUX." The stochastic expression concluded in Definition 4.10 has the same formula with 4-to-1 MUX, where dx and dy are connected to the MUX selection ports, as shown in Figure 4.14. Thus, complex interpolation operations can be implemented in hardware with a simple SC MUX.

Figure 4.15 shows an example of how the stochastic bilinear filter is applied to an image. The processing of neighboring pixels within the scope of linear interpolation for $2\times$ enlargement ($\rho = 2$) of a 4×4 image is indicated. The image pixels are considered according to the stochastic expression given in Definition 4.10. The dx, dy relative distances are evaluated in the formula, and $\bar{I}$ is estimated after I is subjected to convolution with the kernels having "1"s coefficient: $K_{any} = 1$ (also $w_k = 1$).

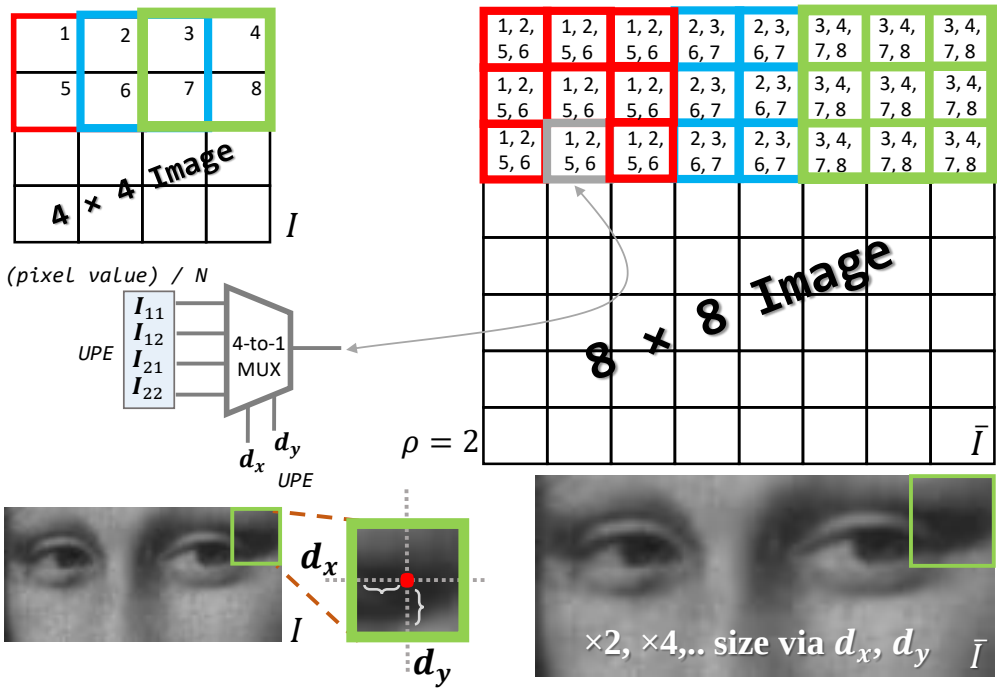


Figure 4.15 : Example of the proposal: Stochastic bilinear filter.

Figure 4.16 shows the results of conventional and stochastic bilinear filters for $\rho = 2$ (size of the image from 110×110 to 220×220). The SC-based result has PSNR and SSIM metrics for $N = 256$, 512, and 1024 with reference to the conventional result via deterministic scalars from conventional multiplication and addition. Both PSNR and SSIM results offer remarkable performance.

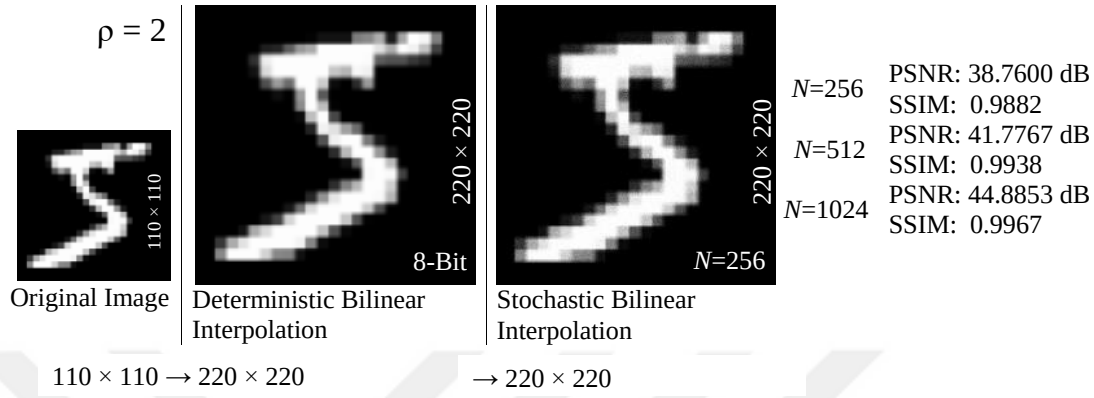


Figure 4.16 : The results of conventional and stochastic (SC-based) bilinear filtering.

On the other hand, Figure 4.17 presents the results of bilinear filters for noisy images (Gaussian and salt & pepper noise). SC-based and deterministic approaches show very close PSNR results, and the SC-based approach is slightly ahead in PSNR performance, even though the overall performance of both filters is low.

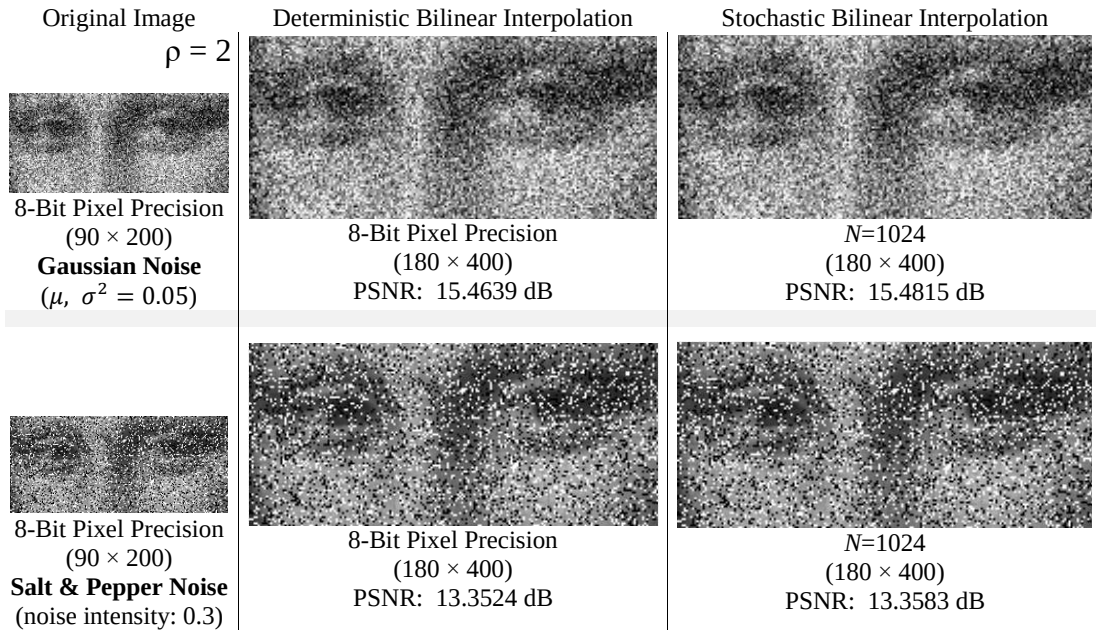


Figure 4.17 : Noise performance of both deterministic and stochastic bilinear filtering.

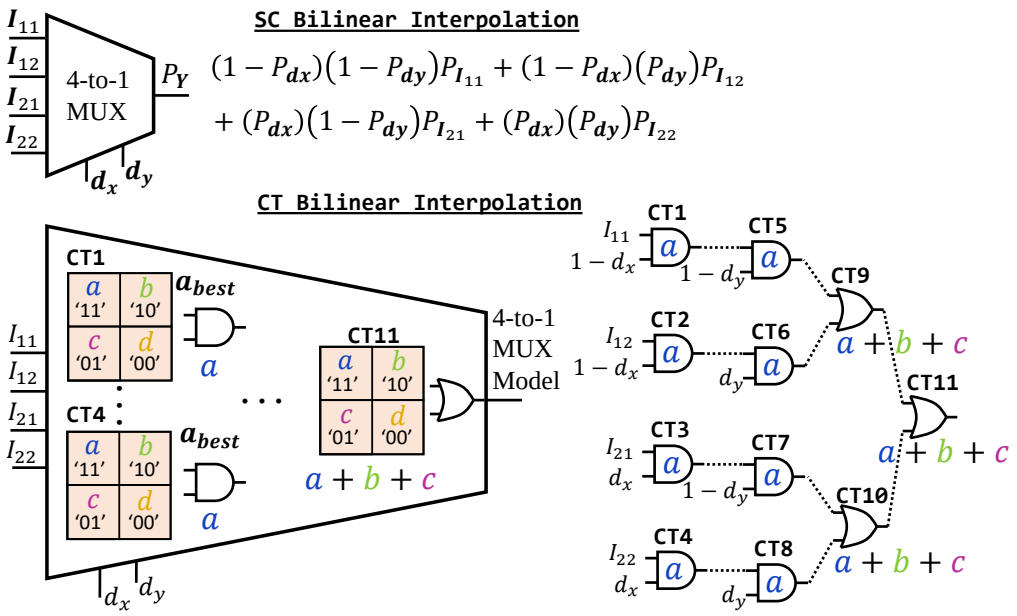
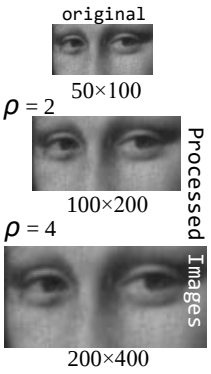


Figure 4.18 : CT-based 4-to-1 MUX construction for SC bilinear interpolation.

Last but not least, the proposed filtering is now evaluated within the scope of the CT-based simulation presented in Chapter 3. Considering the logic model of 4-to-1 MUX, the related CT generation for each logic primitive is performed, and the logic operation through the TCO is obtained. In Figure 4.18, the input for each logic gate is a scalar value presented in the probability calculation indicating the total 1s count. Each CT construct in Figure 4.18 is evaluated based on the targeted operation. For multiplication, the AND gates in the first and second stages are fed with the uncorrelated CT model setting each a via a_{best}. At the third and the fourth stage, CT-9, 10, and 11 are set using previous CTs, and OR gate model is considered via TCO = $a+b+c$. The prior a CT primitive at each gate is to be fine-tuned via variational random fluctuation error, as explained in Section 3.7.

Finally, the proposed SC bilinear interpolation is tested over two scaling values ($\rho = 2$ and 4). As with other tests, both the actual bitstream processing and CT-based approach are evaluated. CT-based random source emulation presented in Section 3.7 is also validated during the tests. The binomial distribution, Sobol-based bitstream processing, and LFSR model are emulated dynamically. In Figure 4.19, `SCRand`, `SCSobol`, and `SCLfsr` are the actual bitstream processing results using the binomial distribution, Sobol LD sequence, and LFSR, respectively. The actual random bitstream is generated in software, considering the model of each random source and bitwise processing.

ρ	SCRand	CTRAND	SCSobol	CT0	SCLfsr	CTLFSR	CONVN
2	RT: 8.186 PSNR: 30.496	RT: 0.024 PSNR: 29.857	RT: 9.325 PSNR: 45.778	RT: 0.015 PSNR: 44.661	RT: 7.707 PSNR: 35.927	RT: 0.025 PSNR: 34.665	RT: 0.009 PSNR: ref.
	MEM: ×22.792		MEM: ×22.803		MEM: ×22.799		
4	RT: 34.417 PSNR: 30.450	RT: 0.074 PSNR: 29.782	RT: 37.778 PSNR: 41.533	RT: 0.045 PSNR: 41.540	RT: 29.205 PSNR: 33.165	RT: 0.1411 PSNR: 33.321	RT: 0.032 PSNR: ref.
	MEM: ×23.506		MEM: ×23.520		MEM: ×23.508		



Tests are performed on a computer with Intel(R) Core(TM) i7-7700HQ CPU @2.80GHz, 16GB RAM, MATLAB 2017b. Results are based on the average of 1000 different rounds.

Figure 4.19 : Bilinear interpolation results based on both bitstream processing (SC) and CT. ρ is set to double and quadruple the images.

Furthermore, the CT-based scalar-only simulation approach is also tested. Referencing the two-input ($P_{\boldsymbol{X_1}}$, $P_{\boldsymbol{X_2}}$) AND gate model with $P_{\boldsymbol{Y}}$ exact output probability, CTRAND with binomial distribution fine-tunes a as $\frac{a}{N} + \sqrt{P_Y(1-P_Y)/N}$, where $\frac{a}{N}$ is the output probability of an AND gate having no random error, and $\sigma = \sqrt{P_Y(1-P_Y)/N}$ is the deviation from exact value due to random fluctuation. CT0 signifies Sobol LD, which is simple and equal to a_{best} since $SCC = 0$ optimization in Lemma 3.1 guarantees high accuracy, and actual Sobol bitstream results are expected to be similar. For CT-based LFSR emulation, CTLFSR is a hypergeometric distribution-based LFSR model in Section 3.7. The output deviation of the bitwise AND operation on LFSR-based bitstreams is defined as $\sigma = \sqrt{\frac{P_{X1}\times P_{X2}\times(1-P_{X1})\times(1-P_{X2})}{N-1}}$; thereby, AND operation has output probability $\frac{a}{N} + \sqrt{\frac{P_{X1}\times P_{X2}\times(1-P_{X1})\times(1-P_{X2})}{N-1}}$.

In Figure 4.19, the results show PSNR accuracy (in dB), runtime (RT, in seconds), and memory (MEM) consumption. The bitstream size is $N = 256$. The results presented in Figure 4.19 are the average of 1000 independent trials. According to the PSNR results obtained with reference to the images produced by the conventional binary calculation (CONVN), all bitstream processing results can be emulated very closely by CT. As expected, the Sobol method achieves the best performance, followed by LFSR and binomial random. CT provides approximately 22 times more efficient memory usage than the actual bitstream processing for each technique. When the scaling factor is increased from $\rho = 2$ to $\rho = 4$, CT can even simulate the slight decrease in the PSNR values. As can be seen, the CT runtime is even comparable to the runtime of the CONVN approach, and that promises CT's next-generation computing potential.

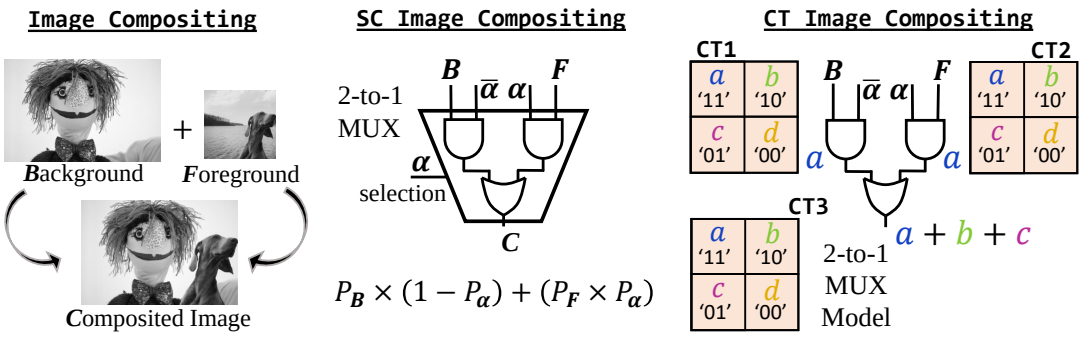


Figure 4.20 : Image compositing: Proposed SC-based actual bitstream processing with 2-to-1 MUX for image compositing and efficient simulation model via CT.

4.2.5 Image compositing: New perspectives toward CT simulation-II

For the first time in the literature, a 2-input MUX is used for compositing two images: background B and foreground F. In conventional image processing, a composited image (C) is denoted by the formula $C = B(1-\alpha) + F\alpha$, where α is the foreground image opacity [206]. A MUX with $\boldsymbol{X1}$ and $\boldsymbol{X2}$ input and $\boldsymbol{S}$ select bitstreams implements $P_{X1}(1-P_S) + P_{X2}P_S$ [15]. The composited image C and the MUX formula are well-matched by rewriting $\boldsymbol{C} = P_B(1-P_\alpha) + P_F P_\alpha$. As shown in Figure 4.20, connecting $\boldsymbol{B}$, $\boldsymbol{F}$, and $\boldsymbol{\alpha}$ to a MUX yields the compositing operation by recalling the scaled adder formula from equation (2.4). Figure 4.20 also illustrates the CT model for a MUX. Like in bilinear interpolation, multiple CTs are connected in cascade, considering the corresponding TCO formula for each logic.

Next, the performance of CT is evaluated for the SC image compositing. Figure 4.21 shows two pairs of Background (B) and Foreground (F) images as two different examples (Deer and Dog), the runtime in sec, and the quality of the outputs in PSNR as the performance metrics. The bitstream size is $N = 256$. PSNR values prove that the CT approach successfully emulates different bitstream generation methods (i.e., `SCRandom`, `SCSobol`, `SCLfsr`) and provides comparable performance to the bitstream processing approach. Comparing different random sources, the Sobol-based methods achieve the best PSNRs, then the LFSR method and the binomial random source follow. In the two examples, CT significantly reduces the runtime compared to actual bitstream processing.

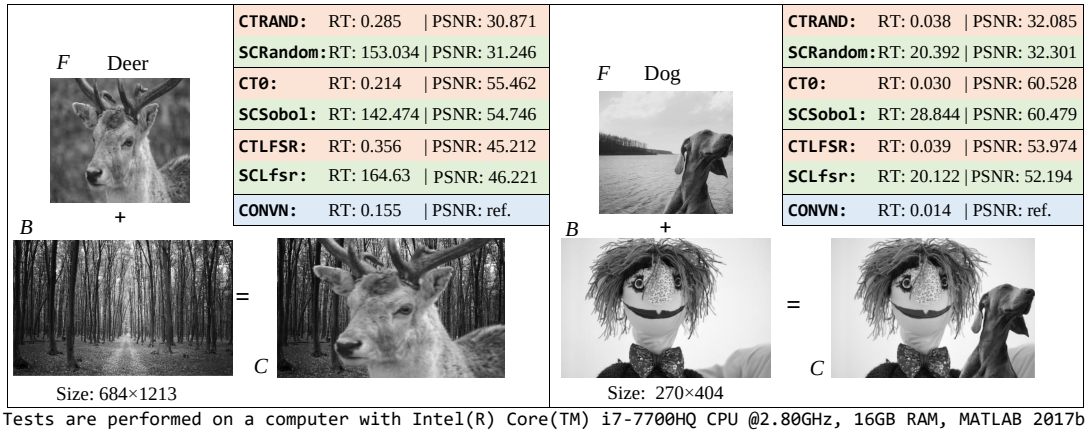


Figure 4.21 : Image compositing results for two examples: Deer and Dog. Monitoring random source emulation, PSNR (dB) accuracy, and runtime (in sec).

All in all, the common stochastic bitstream generation approaches using the binomial distribution, hypergeometric or pseudo-random, and LD are modeled with CT. Our experimental results for simulation of the proposed SC image processing methods prove that CT performs more efficiently than the actual bitstream processing in runtime and memory usage. Furthermore, in terms of the accuracy metrics such as PSNR, CT produces results with the same accuracy level as the results from the traditional bitstream processing.

5. LEARNING MACHINES

The SC paradigm is adapted to learning systems by presenting new views in terms of NNs. As surveyed in Section 2.6, there are numerous efforts on machine learning and NNs to make lightweight and effective systems in terms of hardware. This chapter presents how it is possible to update the “learning” phenomenon considering SC. First, the single neuron design is presented, and then, the network structures on a larger scale are given together with background information on the conventional NN systems. In this chapter, new perspectives regarding the design of SCNNs are shown, thereby simulating hardware-aware machine learning systems: “learning machines.”

SC-based learning machines require attention in terms of performance monitoring considering the accuracy of a system. Many previous efforts surveyed in Section 2.6 underline the hardware efforts on SC. Nevertheless, emerging NNs require optimization for concepts like network structures, training scenarios, various noise types (or corruptions), and soft-error-aware training strategies. Regarding these concepts, this study differs from the previous works while contributing to the machine learning aspects in addition to the hardware design issues. First, the following sections give the single neuron model for traditional and SC-based design targeting noisy data classification. It is proven that SC-based neurons having the stochastic training cycle outperform conventional design and offer higher accuracy in early epochs. Next, deeper architectures are presented by constructing the background on the traditional NN systems. A reader of this chapter gets the advantage of understanding "how a neural network system learns" and comprehending the formulas behind the scenes. Furthermore, BSBNN is proposed as a new network structure during the SC adaptation to deeper architecture. It is shown how efficient and robust architecture can be performed when SC is used in BNNs. The last section presents fast training of stochastic bitstream processing NNs via CT.

5.1 Overview of a Single Stochastic Neuron

Previous chapters proved that the primary motivation in SC is to exploit probabilistic behavior providing simplified arithmetic and fault tolerance. This section inspects an overview of a single neuron and the advantages of SC utilization in a neuron design. The one-layer-one-neuron problem for classification purposes is tested.

For the beginning, let us first recall the error function to be used as a reference point and a measure of success in NN systems. MSE is one of a cost function and frequently used in NNs: $MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$ [207]. The obtained ($\hat{Y}$) and expected (Y) values are evaluated in the sum of squares function, representing the error from n sample data points. $\hat{Y}$ is the predicted output, namely the obtained output from a decision. The expected output, i.e., the label/class of the sample data, is Y_i. The difference, $Y_i - \hat{Y}_i$, is important to measure the deviation from the correct value.

For simplicity, the term for averaging, $\frac{1}{n}$, can be written as $\frac{1}{2n}$. Hence, the '2' coming from the derivative of the squared expression cancels the one in the denominator. For the single-sample case, $n = 1$, the function can be written as $\frac{1}{2}(Y_i - \hat{Y}_i)^2$, which will be called MSE loss function. The E will be used to define the corresponding *error* via the loss function, $E = \frac{1}{2}(Expected - Obtained)^2 = \frac{1}{2}(Y_i - \hat{Y}_i)^2$.

The single neuron model satisfies the linear equation ($y = m \times x + b$, where x and y are independent variables, m is the slope of a represented line, and b is the y-intercept of described line). In [208], Bishop explains that linear models for regression and classification are based on linear combinations of fixed nonlinear basis functions, $\phi_j(\mathbf{x})$, yielding to ($\mathbf{x}$, $\mathbf{w}$ indicate having multiple observations):

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \times \phi_j(\mathbf{x})\right) \tag{5.1}$$

In equation (5.1), $f(\cdot)$ is denoted as a nonlinear activation function for the classification case (it is the identity if regression is the case), and M is the total number of parameters. Bishop explains that the goal is to make the basis function $\phi_j(\mathbf{x})$ depend on parameters to be adjusted. The parameters (*weights*: w_j) can be adjusted during the training. Considering the NN case, the basis function is a nonlinear function of a linear

combination of the inputs. The coefficients in the linear combination are adaptive parameters. By first defining the linear combination of two inputs (x_1 and x_2), equation (5.2) is given for the NN case considering equation (5.1):

$$S_j = \sum_{i=1}^{2} (w_{j,i} \times x_i) + w_{j,0}. \tag{5.2}$$

For a single neuron case, j is equal to 1. $w_{j,0}$ is a *bias* parameter that allows a fixed offset. The S is the activation (pre-activation) and is transformed using a differentiable function, f, such as $f(x) = tanh(x)$, $sigmoid(x)$, $sign(x)$, etc. [209]. The neuron output is finally denoted as $y_j = f(S_j)$ [208].

From the conventional neuron description, the definition of a stochastic neuron is given in Definition 5.1.

Definition 5.1 (Stochastic Neuron)**.** Considering equation (5.2), the stochastic neuron is defined based on bitstreams $\boldsymbol{w}$ and $\boldsymbol{x}$ in BPE format. A stochastic neuron with $\boldsymbol{x}$ inputs has the multiplication based on XNORs, and accumulation is performed using MUXs, i.e., scaled adders.

The stochastic neuron shown in Figure 5.1 performs multiply and accumulate (MAC) operations in the SC domain. Even though there are examples of SC-based activation functions in the literature [134, 139, 149–151], this work keeps the activation function deterministic. A new proposal for binarized networks in Section 5.3.2 will simplify activation operation. The bitstream encoding is performed using BPE since weights may be negative values. Figure 5.1 presents the forward pass of the neuron; however, there is also a backward pass during the learning phase (see Section 5.1.2).

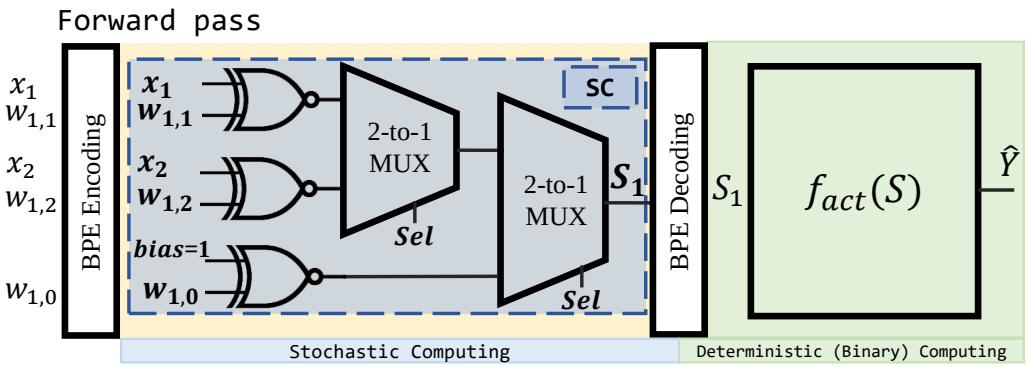


Figure 5.1 : Hardware-inspired stochastic neuron model.

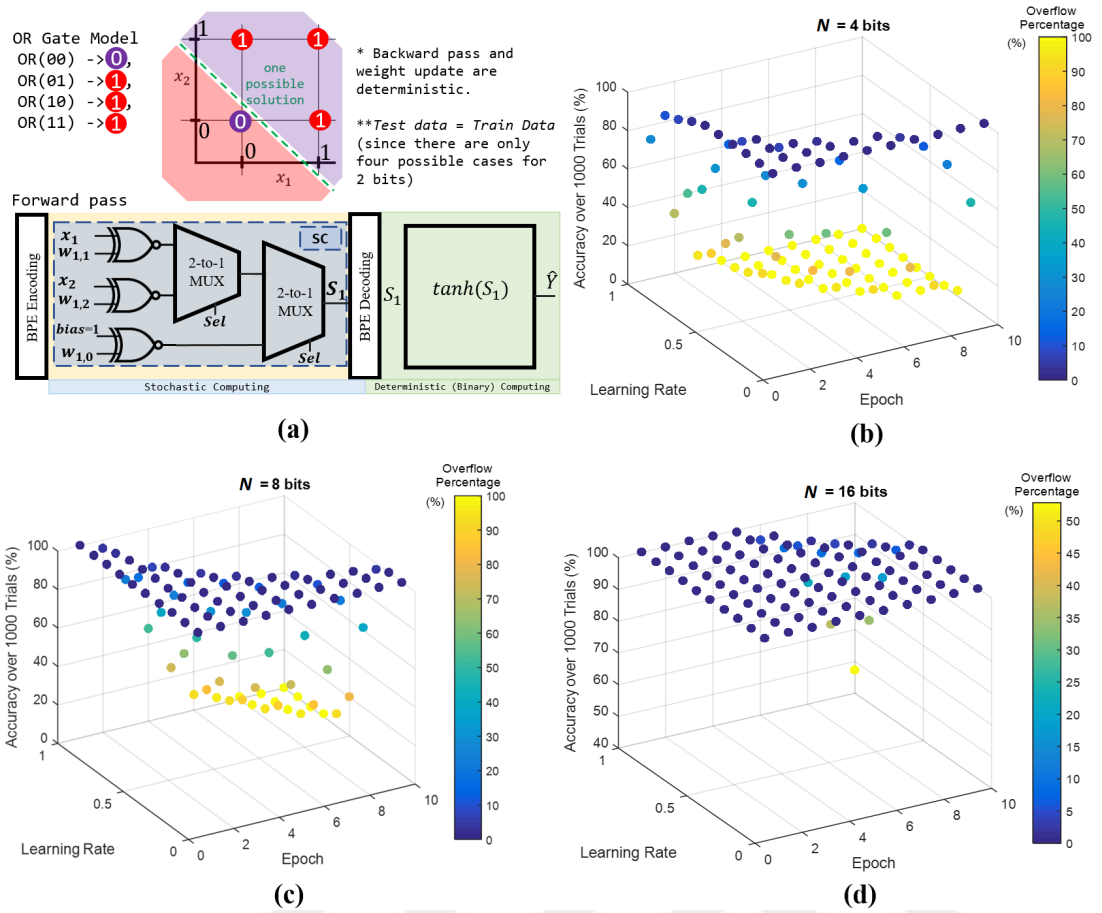


Figure 5.2 : Single-neuron OR model: (a) OR model using architecture in Figure 5.1, tests with (b) $N = 4$ bits, (c) $N = 8$ bits, and (d) $N = 16$ bits.

5.1.1 Single-neuron OR model

Let us start the analysis of the SC-based neuron with the OR gate linear classification example, which is usually given as the first example in the literature [210]. Our model aims to linearly classify logic cases: binary 00, 01, 10, and 11 inputs result in 0 or 1 ($OR(00) \rightarrow 0$, $OR(01) \rightarrow 1$, $OR(10) \rightarrow 1$, $OR(11) \rightarrow 1$). The operations are performed using the BPE equivalents of the base-10 numbers $(0)_{10}$ and $(1)_{10}$, which are manipulated as scalars. The forward pass MAC in Figure 5.1 (blue-shaded region) is used during the training phase, keeping all the rest of the operations (such as activation function and backpropagation) in the deterministic domain. The test neuron in Figure 5.1 and the trained model are utilized for testing, as indicated in Figure 5.2 (a). Each model for $N = 4$, 8, and 16 is considered using $\times 1000$ random bitstream generation of $\boldsymbol{x}$ and $\boldsymbol{w}$. At each epoch of the obtained model, random bitstreams

(having binomial distribution) are performed for test purposes to validate the model; thereby, accuracy is recorded as an average of the 1000 trials. In Figures 5.2 (a), (b), and (c), the corresponding results are shown. Except for $N = 4$, the other test cases perform robust classification at early epochs. With increasing epochs, the training decreases the performance. The overflow percentage with the heat-map bar indicates the inefficiency of the bitstream size that is not capable of representing $\boldsymbol{w}$s.

These plots highlight two crucial points. First, there is a solution to the simple problem in the first epoch in the SC-based classification, and second, this is related to the proper weight initialization. In the following epochs, the performance decreases because N cannot represent the weight values due to the overflow of weight scalars ($w >> N$). After all, the accuracy drops in the plots show parallelism with the change of overflow colors in the bars on the right.

5.1.2 Binary classification in premature epochs

In conventional machine learning, one of the fundamental problems is the linear separation between two clusters. As mentioned in the previous subsection, this problem can be achieved using a single neuron in a conventional learning system. However, the problem becomes more complex if the two linearly separable clusters are close to each other or have interpenetrating samples, i.e., having noise [211].

Before going into details, let us first have a conceptual look at the learning phase of a traditional neuron (details will be presented in Section 5.2). The learning phase of a neuron consists of three main steps: *forward pass*, *backward pass*, and *weight update*. As presented previously, there are three primary operations in the forward pass: multiplication, accumulation, and activation. Backward pass is an optimization-based step to reduce errors from the neuron output to the input. At the neuron output, the error can be written as $E = \frac{1}{2}(Expected - Obtained)^2 = \frac{1}{2}(Y_i - \hat{Y}_i)^2$ (by remembering the MSE loss function from Section 5.1). In the case of binary data classification (classifying two clusters with a single line, i.e., with a single neuron), each class label is the expected value and is compared via E with the obtained neuron output. After the error calculation, the error optimization is performed with respect to the w. The well-known optimization technique, *gradient descent*, is utilized to minimize the error concerning the weights by using $\frac{\partial E}{\partial w}$ [86]. For each of the ws, the gradients are

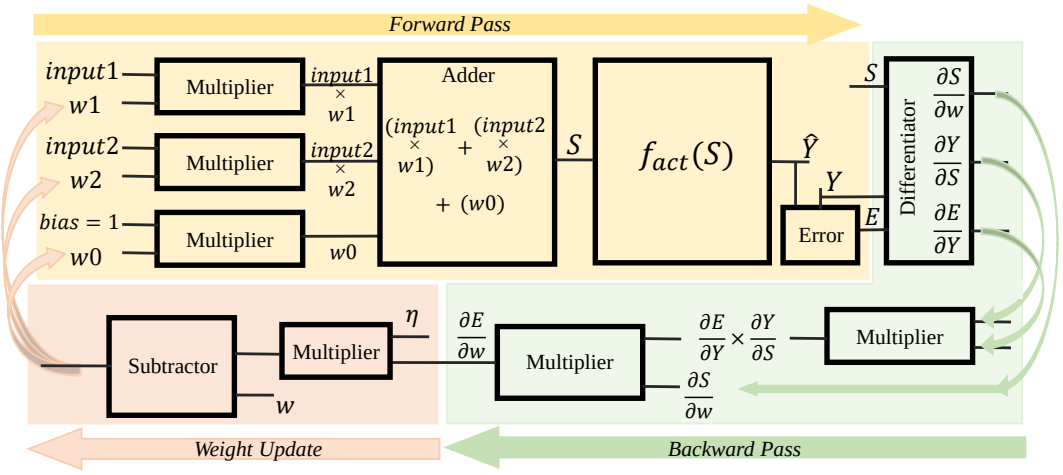


Figure 5.3 : Single neuron training architecture in the deterministic domain.

calculated using the chain rule, $\frac{\partial E}{\partial w} = \frac{\partial E}{\partial Y} \times \frac{\partial Y}{\partial S} \times \frac{\partial S}{\partial w}$. Lastly, the weight update phase adjusts the ws depending on the delta rule: $w^{(new)} = w^{(old)} - \eta \times \frac{\partial E}{\partial w}$, where η is the learning rate [86]. In Figure 5.3, the binary computation-based conventional neuron architecture is depicted.

Back to two-class real data classification: a single neuron is now utilized to separate more complex data than the OR gate model. Two separate data clusters lie in the 2D coordinate system having positional features coming from the x and y axes. The features coming from each cluster are the inputs of a neuron. Let $input1$ and $input2$ be the neuron inputs (x_1, x_2), representing the x and y positions of each point in a cluster, respectively. The pre-activation value, $S = (input1 \times w1) + (input2 \times w2) + w0$, resembling the line equation, represents a classifier line. For the SC-based operations, the neuron inputs are the BPE bitstreams $\boldsymbol{input1}$ and $\boldsymbol{input2}$. The pre-activation in SC is written as $\boldsymbol{S} = MUX\left(XNOR(\boldsymbol{input1}, \boldsymbol{w1}), XNOR(\boldsymbol{input2}, \boldsymbol{w2})\right)$ (ignoring bias for simplicity), considering the hardware blocks of the forward pass. Furthermore, the backward pass is also targeted to be in the SC domain. Liu et al. first explained the gradient descent implementation in SC by using the stochastic integrator [212]. Their proposal worked well for the least mean square error. In [72], Liu et al. proposed a backward propagation algorithm in the SC domain. Their design was based on ESL [20] and binarized networks [213]. This dissertation looks up the answer to which part of the backward pass can best fit in the SC domain.

Figure 5.4 shows the learning processes of the single stochastic neuron and the proposed structure. In the continuation of the forward pass architecture from

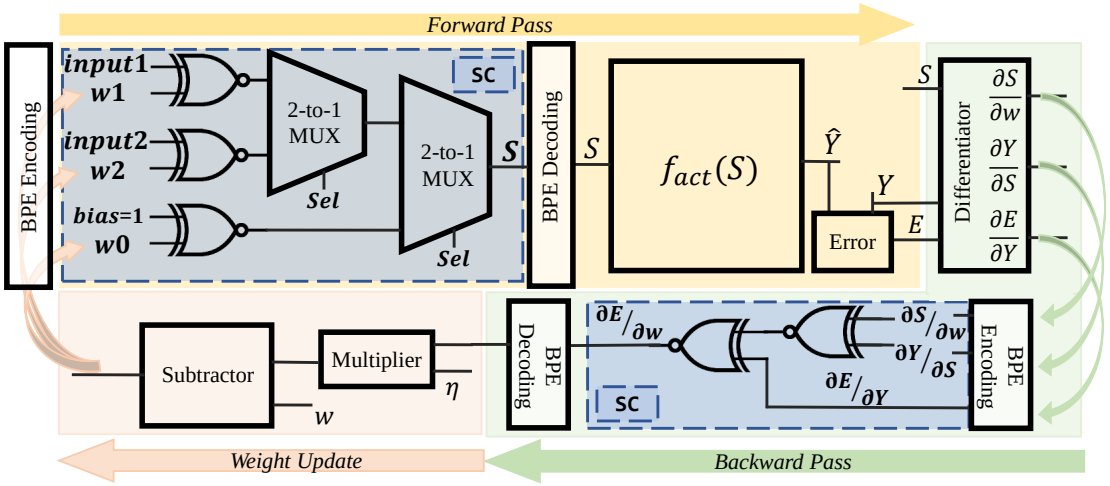


Figure 5.4 : Single stochastic neuron proposed training architecture.

Figure 5.1, the error calculation is performed, and the gradient calculations are shown in the backward pass. After pre-activation, the activation phase is kept in the deterministic domain together with the error calculation for ease of processing. The stochastic and deterministic parts of the hybrid architecture was determined with different experiments. The target is to propose the best-performing architecture that shows rapid and efficient binary classification in noisy data. The quantized activation function with a look-up table is investigated. Regarding the early-epoch classification, it is recommended not to involve the activation function for the outperformance of SC. In addition, weight update is not held in the SC domain. In software experiments, it is observed that keeping the weight update in SC fashion prolongs the learning phase. The backpropagation is handled in the SC domain to make gradient descent optimization robust to the noisy data. This representation of training is the key point of the proposal. Each partial derivative in the chain rule is converted to the BPE streams from their deterministic values. Multiplications are performed using XNORs.

For the validation and test of architecture in Figure 5.4, two different linearly separable clusters are randomly created on the xy-coordinate system. The sample points above and below the classifier that conforms to the $y = m \times x + b$ line equation are separated into two classes. The points of the two separate clusters are targeted to be close that almost destroying a margin shown in Figure 5.5. To tackle this, a large number of random points are determined. Then, the noise is injected into a two-class D dataset such that it changes the value of each sample's x and y positions. Hence, the margin region is exceeded, thanks to noise injection with the Gaussian distribution. Let Z be a

standard normal deviate, i.e., normally distributed deviate, which is an RV with $\mu = 0$ and $\sigma^2 = 1$. $X = Z\sigma + \mu$ is obtained as a normal deviate with (μ, σ); thus, X has Gaussian distribution denoted as $X \sim N(\mu, \sigma^2)$. Using μ and σ as parameters while generating noise, 100% linearly separable data, D, becomes noisy, and two classes get mixed up yielding a new dataset D'. Maximum achievable accuracy (MAA) for D' classification is $MAA < 100\%$. In Figure 5.5, overall data preparation (I), noise injection (II), sample selection (III), and testing (IV) steps are presented.

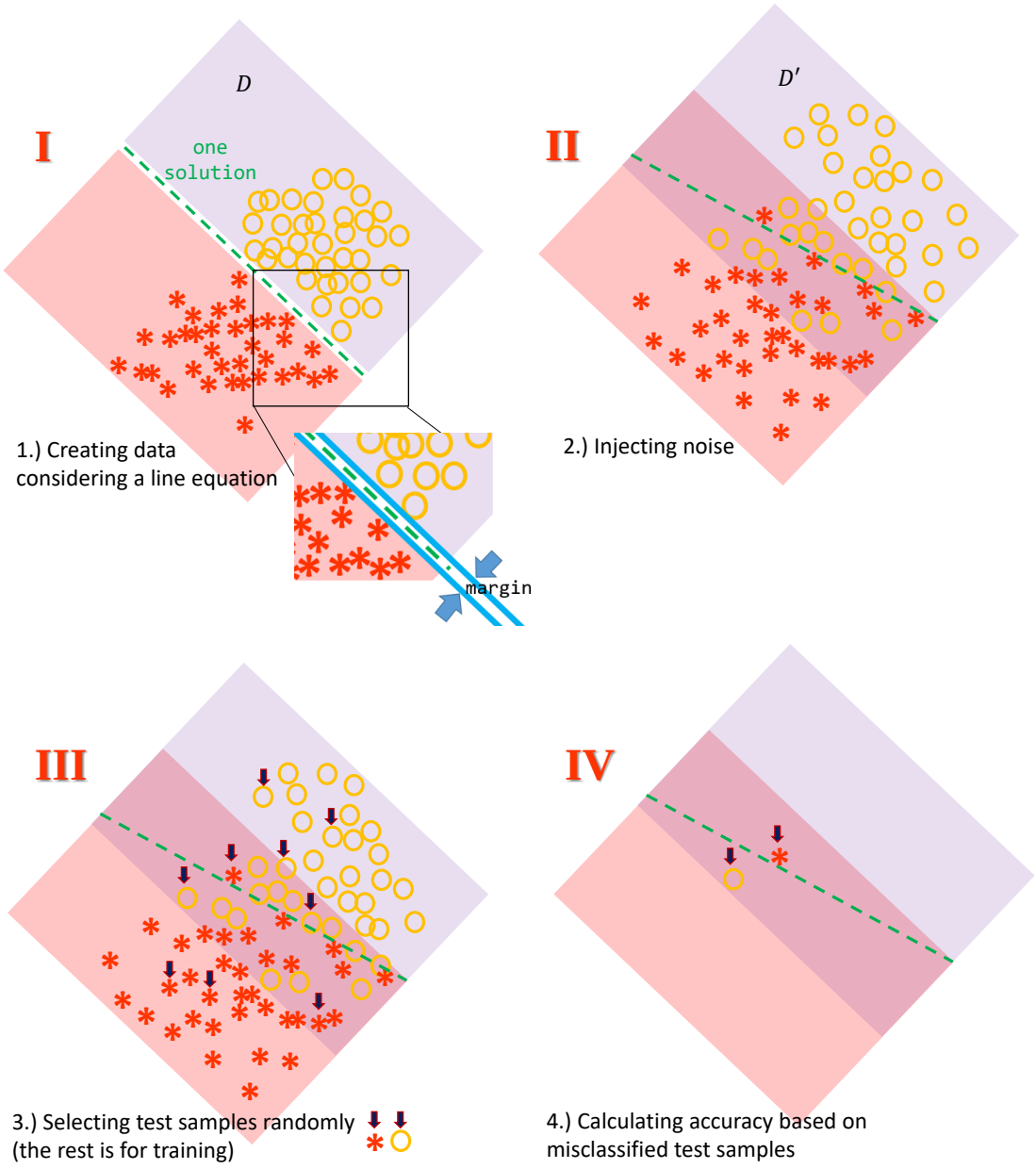


Figure 5.5 : (I) Data preparation, (II) noise injection, (III) sample selection, and (IV) testing steps.

As shown in Figure 5.5, the noise injected data, D', is divided into two parts for training and testing. Test data are not included in the training. During the experiments, μ and σ are selected so that the *MAA* is 98.11%. This accuracy is the highest value that can be achieved deterministically with a linear classifier. The data then begins to be trained such that test set performance is recorded at each epoch. The training operation is performed both in deterministic (using binary operations) and stochastic fashion (using architecture in Figure 5.4). Tests are performed using binary operations for the deterministic method and using the architecture in Figure 5.1 for the SC-based approach. On the other hand, η is incremented from 0.01 to 0.05 (the best-performing range for the data) with 0.01 steps to observe the learning rate effect.

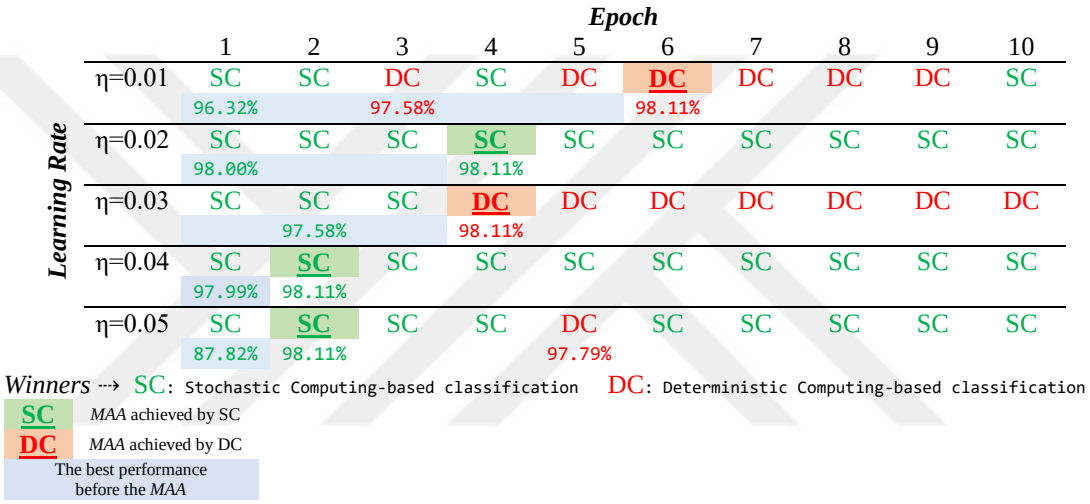


Figure 5.6 : Early-epoch winners of the noisy data classification.

In Figure 5.6, classification test results are shown. The one with higher classification accuracy from SC or deterministic computing (DC) is recorded in each epoch. As can be seen, SC occurs as a winner in early epochs. On the other hand, SC or DC are also underlined (shaded green and red, respectively) to indicate the epoch in which *MAA* was captured. In the epochs before *MAA* (blue-shaded region), the ones with the highest accuracy for SC and DC are written down by noting their numerical values. In addition to SC's majority, close values to *MAA* are generally obtained in the blue-shaded areas. To our knowledge, these results are new findings that have not been demonstrated in the literature before. Only Kim et al. present a study for early NN training efforts in SC with timely training termination [71].

5.2 From Traditional to Bitstream Processing Networks

This section takes a look at the MLP architecture from the conventional and bitstream processing system perspective. The term 'bitstream processing' is used more generally as it also includes unary computing [100]. The systematic transition from conventional DNN systems to emerging networks starts in this section. For this reason, a systematic overview of traditional NNs and step-by-step formulations are presented. Next, the simulation-feasible general structure of an SCNN is shown. The core idea is to create routine design procedures for a bitstream processing NN that is efficient for computer-aided simulation.

Let us first define some parameters and remind the audience of others:

- x (or X): input (image pixel or feature), x_u is the u^{th} neuron output in the input layer
- w: weight, $w_{u,p}^{(r)}$ is the weight in between the u^{th} neuron in the $(r)^{th}$ layer, and p^{th} neuron in the $(r-1)^{th}$ layer
- b: bias (sometimes written as w_0, or $b \times w_0$, where $b = 1$ or $w_0 = 1$)
- S: pre-activation value, $S_u^{(r)}$ is for the u^{th} neuron of the $(r)^{th}$ layer
- f, g: activation functions (g can be different than the previous activation function (f) special to the output layer)
- y: output of a neuron in hidden or output layer, $y_u^{(r)}$ is the u^{th} neuron output of the $(r)^{th}$ layer
- E_{total}: total error as $E_1 + ... + E_t$, where t is the number of output neurons

An example of MLP with two hidden layers is shown in Figure 5.7. Considering the deep network architecture, the parameters used during the forward and backward pass and their connections are presented. Here, the 'deep' keyword applies to networks with two or more hidden layers (*nohl*: number of hidden layers). As mentioned earlier in a conceptual manner, the backward pass optimizes the total error back over all weights on the network via backpropagation. The obtained outputs from the output layer and the expected class (or value) of the input sample are evaluated regarding the error function. The total error from all output neurons, E_{total}, is optimized based on the weights in the network. Exemplifying a two-neuron output system ($t = 2$), E_{total} can be written as in equation (5.3):

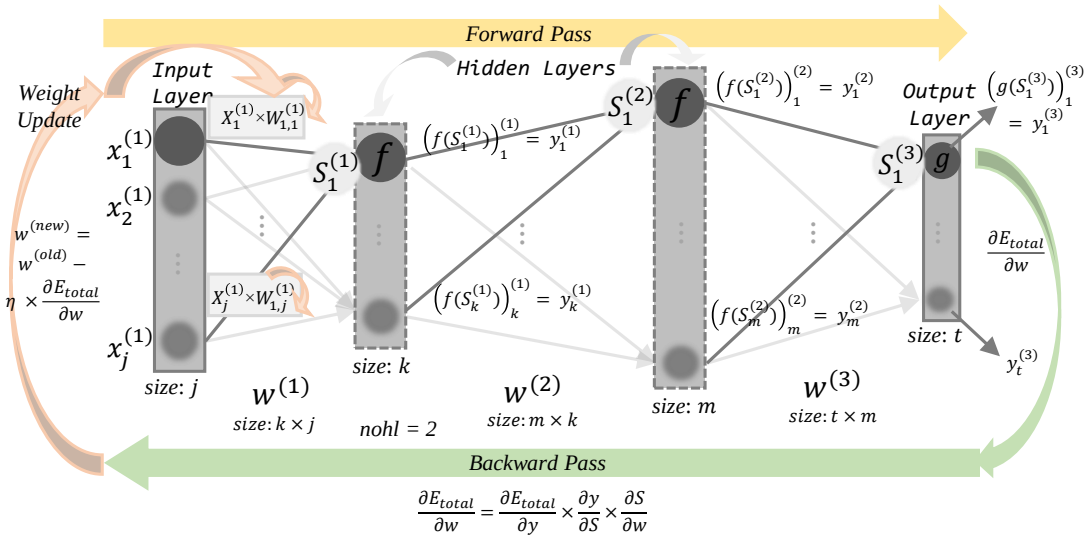


Figure 5.7 : An example of conventional MLP training with two hidden layers.

$$E_{total} = E_1 + E_2. \tag{5.3}$$

E_1 and E_2 are the errors related to the two neurons of the output layer. Recalling MSE, it is noted that $E_1 = \frac{1}{2}(y_{expected_1} - y_1^{(nohl+1)})^2$ and $E_2 = \frac{1}{2}(y_{expected_2} - y_2^{(nohl+1)})^2$. The obtained ($\hat{Y}$) outputs, $y_1^{(nohl+1)}$ and $y_2^{(nohl+1)}$, have superscripts $(nohl+1)$ indicating the relationship to the last layer (see Figure 5.7 example of $nohl = 2$, and $nohl + 1 = 3$ marks the last-layer-related parameters of w, S, y). Now, equation (5.3) turns into

$$E_{total} = \frac{1}{2}(y_{expected_1} - y_1^{(nohl+1)})^2 + \frac{1}{2}(y_{expected_2} - y_2^{(nohl+1)})^2. \tag{5.4}$$

The chain rule is applied for the optimization of $w_{1,1}^{(nohl+1)}$ considering $\frac{\partial E_{total}}{\partial w_{1,1}^{(nohl+1)}}$, which yields

$$\frac{\partial E_{total}}{\partial w_{1,1}^{(nohl+1)}} = \frac{\partial E_{total}}{\partial y_1^{(nohl+1)}} \times \frac{\partial y_1^{(nohl+1)}}{\partial S_1^{(nohl+1)}} \times \frac{\partial S_1^{(nohl+1)}}{\partial w_{1,1}^{(nohl+1)}}. \tag{5.5}$$

The first term in equation (5.5), $\frac{\partial E_{total}}{\partial y_1^{(nohl+1)}}$, i.e., the partial derivative of total error with respect to the obtained value of the first neuron in the last layer, is evaluated for the derivative regarding the equation (5.4), giving

$$\frac{\partial E_{total}}{\partial y_1^{(nohl+1)}} = 2 \times \frac{1}{2} \times (y_{expected_1} - y_1^{(nohl+1)})^{2-1} \times -1 + 0 \tag{5.6}$$

$$\frac{\partial E_{total}}{\partial y_1^{(nohl+1)}} = -(y_{expected_1} - y_1^{(nohl+1)}) = y_1^{(nohl+1)} - y_{expected_1}. \tag{5.7}$$

By selecting an example activation function, $g(x) = tanh(x)$, the output of the first neuron from the last layer is written as:

$$y_1^{(nohl+1)} = tanh(S_1^{(nohl+1)}). \tag{5.8}$$

Recalling equation (5.5), the second term is related to S, $\frac{\partial y_1^{(nohl+1)}}{\partial S_1^{(nohl+1)}}$, which is performed considering the derivative of $tanh(x)$:

$$\frac{\partial y_1^{(nohl+1)}}{\partial S_1^{(nohl+1)}} = 1 - tanh^2(S_1^{(nohl+1)}). \tag{5.9}$$

The pre-activation $S_1^{(nohl+1)}$ holds the previous layer neuron outputs, $y^{(nohl)}$ (the final hidden layer outputs multiplied by the output-layer-related weights, $w^{(nohl+1)}$), as given in equation (5.10):

$$S_1^{(nohl+1)} = y_1^{(nohl)} \times w_{1,1}^{(nohl+1)} + y_2^{(nohl)} \times w_{1,2}^{(nohl+1)} + ... + b. \tag{5.10}$$

The partial derivative of S with respect to $w_{1,1}^{(nohl+1)}$ (which is the third term in equation (5.5)) is then written in equation (5.11):

$$\frac{\partial S_1^{(nohl+1)}}{\partial w_{1,1}^{(nohl+1)}} = y_1^{(nohl)} + 0 + ... + 0 = y_1^{(nohl)}. \tag{5.11}$$

If the results from equation (5.7), equation (5.9), and equation (5.11) are combined for equation (5.5), the following equation (5.12) is obtained:

$$\frac{\partial E_{total}}{\partial w_{1,1}^{(nohl+1)}} = \frac{\partial E_{total}}{\partial y_1^{(nohl+1)}} \times \frac{\partial y_1^{(nohl+1)}}{\partial S_1^{(nohl+1)}} \times \frac{\partial S_1^{(nohl+1)}}{\partial w_{1,1}^{(nohl+1)}} = \\ [y_1^{(nohl+1)} - y_{expected_1}] \times [1 - tanh^2(S_1^{(nohl+1)})] \times [y_1^{(nohl)}]. \tag{5.12}$$

Using the delta rule, equation (5.13) is written to update the weight in interest, $w_{1,1}^{(nohl+1)}$, with the help of equation (5.12).

$$\hat{w}_{1,1}^{(nohl+1)} = w_{1,1}^{(nohl+1)} - \eta \times \frac{\partial E_{total}}{\partial w_{1,1}^{(nohl+1)}}. \tag{5.13}$$

On the other hand, the δ is defined to indicate the weighted sum of the errors as:

$$\delta_1^{(nohl+1)} = \frac{\partial E_{total}}{\partial y_1^{(nohl+1)}} \times \frac{\partial y_1^{(nohl+1)}}{\partial S_1^{(nohl+1)}} = \frac{\partial E_{total}}{\partial S_1^{(nohl+1)}}. \tag{5.14}$$

Equation (5.13) can now be rewritten as:

$$\hat{w}_{1,1}^{(nohl+1)} = w_{1,1}^{(nohl+1)} - \eta \times \delta_1^{(nohl+1)} \times y_1^{(nohl)}. \tag{5.15}$$

The backpropagation and weight update are performed for each of the weights from the output to the input layer. Therefore, the last hidden layer ($nohl^{th}$) is evaluated after the last layer. Regarding the first neuron of the last hidden layer, the first-weight-related optimization is written as in equation (5.16):

$$\frac{\partial E_{total}}{\partial w_{1,1}^{(nohl)}} = \frac{\partial E_{total}}{\partial y_1^{(nohl)}} \times \frac{\partial y_1^{(nohl)}}{\partial S_1^{(nohl)}} \times \frac{\partial S_1^{(nohl)}}{\partial w_{1,1}^{(nohl)}}. \tag{5.16}$$

For the total error, the term $\frac{E_{total}}{y_1^{(nohl)}}$ is then expanded as in equation (5.17). The equation takes all t neurons at the output into account (see Figure 5.7):

$$\frac{\partial E_{total}}{\partial y_1^{(nohl)}} = \frac{\partial E_1}{\partial y_1^{(nohl)}} + \frac{\partial E_2}{\partial y_1^{(nohl)}} + ... + \frac{\partial E_t}{\partial y_1^{(nohl)}}. \tag{5.17}$$

Any term in equation (5.17) can be written as follows, which establishes a relation to S. For instance,

$$\frac{\partial E_1}{\partial y_1^{(nohl)}} = \frac{\partial E_1}{\partial S_1^{(nohl+1)}} \times \frac{\partial S_1^{(nohl+1)}}{\partial y_1^{(nohl)}}. \tag{5.18}$$

The term $\frac{\partial E_1}{\partial S_1^{(nohl+1)}}$ can be calculated as:

$$\frac{\partial E_1}{\partial S_1^{(nohl+1)}} = \frac{\partial E_1}{\partial y_1^{(nohl+1)}} \times \frac{\partial y_1^{(nohl+1)}}{\partial S_1^{(nohl+1)}}. \tag{5.19}$$

After rewriting the $\frac{\partial E_1}{\partial y_1^{(nohl)}}$, it turns into equation (5.20):

$$\frac{\partial E_1}{\partial y_1^{(nohl)}} = \frac{\partial E_1}{\partial y_1^{(nohl+1)}} \times \frac{\partial y_1^{(nohl+1)}}{\partial S_1^{(nohl+1)}} \times \frac{\partial S_1^{(nohl+1)}}{\partial y_1^{(nohl)}}. \tag{5.20}$$

Considering the total error, E_{total}, the following equation (5.21) is written from equation (5.20) and other terms ($\frac{\partial E_2}{\partial y_1^{(nohl)}}, ..., \frac{\partial E_t}{\partial y_1^{(nohl)}}$):

$$\frac{\partial E_{total}}{\partial y_1^{(nohl)}} = \frac{\partial E_1}{\partial y_1^{(nohl+1)}} \times \frac{\partial y_1^{(nohl+1)}}{\partial S_1^{(nohl+1)}} \times \frac{\partial S_1^{(nohl+1)}}{\partial y_1^{(nohl)}} + \frac{\partial E_2}{\partial y_2^{(nohl+1)}} \times \frac{\partial y_2^{(nohl+1)}}{\partial S_2^{(nohl+1)}} \times \frac{\partial S_2^{(nohl+1)}}{\partial y_1^{(nohl)}} + \ldots + \frac{\partial E_t}{\partial y_t^{(nohl+1)}} \times \frac{\partial y_t^{(nohl+1)}}{\partial S_t^{(nohl+1)}} \times \frac{\partial S_t^{(nohl+1)}}{\partial y_1^{(nohl)}}. \tag{5.21}$$

On the other hand, recalling equation (5.16), the chain rule for the partial derivative of E_{total} with respect to the w: $\frac{\partial E_{total}}{\partial w_{1,1}^{(nohl)}} = \frac{\partial E_{total}}{\partial y_1^{(nohl)}} \times \frac{\partial y_1^{(nohl)}}{\partial S_1^{(nohl)}} \times \frac{\partial S_1^{(nohl)}}{\partial w_{1,1}^{(nohl)}}$, an update for $\frac{\partial E_{total}}{\partial y_1^{(nohl)}}$ via equation (5.21) is performed:

$$\frac{\partial E_{total}}{\partial w_{1,1}^{(nohl)}} = \left(\frac{\partial E_1}{\partial y_1^{(nohl+1)}} \times \frac{\partial y_1^{(nohl+1)}}{\partial S_1^{(nohl+1)}} \times \frac{\partial S_1^{(nohl+1)}}{\partial y_1^{(nohl)}} + \frac{\partial E_2}{\partial y_2^{(nohl+1)}} \times \frac{\partial y_2^{(nohl+1)}}{\partial S_2^{(nohl+1)}} \times \frac{\partial S_2^{(nohl+1)}}{\partial y_1^{(nohl)}} + \ldots + \frac{\partial E_t}{\partial y_t^{(nohl+1)}} \times \frac{\partial y_t^{(nohl+1)}}{\partial S_t^{(nohl+1)}} \times \frac{\partial S_t^{(nohl+1)}}{\partial y_1^{(nohl)}} \right) \times \frac{\partial y_1^{(nohl)}}{\partial S_1^{(nohl)}} \times \frac{\partial S_1^{(nohl)}}{\partial w_{1,1}^{(nohl)}}. \tag{5.22}$$

Equation (5.22) can be rewritten using a ∂E_{total} term, instead of using ∂E_1, ∂E_2, ..., ∂E_t since using E_{total} still gives the same result for each, and $\frac{\partial E_1}{\partial y_1^{(nohl+1)}} = \frac{\partial E_{total}}{\partial y_1^{(nohl+1)}}$, $\frac{\partial E_2}{\partial y_2^{(nohl+1)}} = \frac{\partial E_{total}}{\partial y_2^{(nohl+1)}}$,...,$\frac{\partial E_t}{\partial y_t^{(nohl+1)}} = \frac{\partial E_{total}}{\partial y_t^{(nohl+1)}}$. This yields the following:

$$\frac{\partial E_{total}}{\partial w_{1,1}^{(nohl)}} = \left(\frac{\partial E_{total}}{\partial y_1^{(nohl+1)}} \times \frac{\partial y_1^{(nohl+1)}}{\partial S_1^{(nohl+1)}} \times \frac{\partial S_1^{(nohl+1)}}{\partial y_1^{(nohl)}} + \frac{\partial E_{total}}{\partial y_2^{(nohl+1)}} \times \frac{\partial y_2^{(nohl+1)}}{\partial S_2^{(nohl+1)}} \times \frac{\partial S_2^{(nohl+1)}}{\partial y_1^{(nohl)}} + \ldots + \frac{\partial E_{total}}{\partial y_t^{(nohl+1)}} \times \frac{\partial y_t^{(nohl+1)}}{\partial S_t^{(nohl+1)}} \times \frac{\partial S_t^{(nohl+1)}}{\partial y_1^{(nohl)}} \right) \times \frac{\partial y_1^{(nohl)}}{\partial S_1^{(nohl)}} \times \frac{\partial S_1^{(nohl)}}{\partial w_{1,1}^{(nohl)}}. \tag{5.23}$$

Afterward, equation (5.14) is recalled: $\delta_1^{(nohl+1)} = \frac{\partial E_{total}}{\partial y_1^{(nohl+1)}} \times \frac{y_1^{(nohl+1)}}{S_1^{(nohl+1)}}$, which allows us to update equation (5.23) as:

$$\frac{\partial E_{total}}{\partial w_{1,1}^{(nohl)}} = \left(\delta_1^{(nohl+1)} \times \frac{S_1^{(nohl+1)}}{y_1^{(nohl)}} + \delta_2^{(nohl+1)} \times \frac{S_2^{(nohl+1)}}{y_1^{(nohl)}} + ... + \delta_t^{(nohl+1)} \times \frac{S_t^{(nohl+1)}}{y_1^{(nohl)}} \right) \times \frac{\partial y_1^{(nohl)}}{\partial S_1^{(nohl)}} \times \frac{\partial S_1^{(nohl)}}{\partial w_{1,1}^{(nohl)}}. \tag{5.24}$$

The $\frac{\partial S}{\partial y}$ terms are also reconsidered using the following equations:

$$S_1^{(nohl+1)} = y_1^{(nohl)} \times w_{1,1}^{(nohl+1)} + y_2^{(nohl)} \times w_{1,2}^{(nohl+1)} + ... + y_m^{(nohl)} \times w_{1,m}^{(nohl+1)} \tag{5.25}$$

$$S_2^{(nohl+1)} = y_1^{(nohl)} \times w_{2,1}^{(nohl+1)} + y_2^{(nohl)} \times w_{2,2}^{(nohl+1)} + ... + y_m^{(nohl)} \times w_{2,m}^{(nohl+1)} \tag{5.26}$$

$$S_t^{(nohl+1)} = y_1^{(nohl)} \times w_{t,1}^{(nohl+1)} + y_2^{(nohl)} \times w_{t,2}^{(nohl+1)} + ... + y_m^{(nohl)} \times w_{t,m}^{(nohl+1)} \tag{5.27}$$

where, $\frac{\partial S_1^{(nohl+1)}}{\partial y_1^{(nohl)}} = w_{1,1}^{(nohl+1)}$, $\frac{\partial S_2^{(nohl+1)}}{\partial y_1^{(nohl)}} = w_{2,1}^{(nohl+1)}$, and $\frac{\partial S_t^{(nohl+1)}}{\partial y_1^{(nohl)}} = w_{t,1}^{(nohl+1)}$.

Then, equation (5.24) can be rewritten as:

$$\frac{\partial E_{total}}{\partial w_{1,1}^{(nohl)}} = \left(\delta_1^{(nohl+1)} \times w_{1,1}^{(nohl+1)} + \delta_2^{(nohl+1)} \times w_{2,1}^{(nohl+1)} + ... + \delta_t^{(nohl+1)} \times w_{t,1}^{(nohl+1)} \right) \times \frac{\partial y_1^{(nohl)}}{\partial S_1^{(nohl)}} \times \frac{\partial S_1^{(nohl)}}{\partial w_{1,1}^{(nohl)}} \quad , \tag{5.28}$$

which turns into:

$$\frac{\partial E_{total}}{\partial w_{1,1}^{(nohl)}} = \left(\sum_{\alpha=1}^{t} \delta_\alpha^{(nohl+1)} \times w_{\alpha,1}^{(nohl+1)} \right) \times \frac{\partial y_1^{(nohl)}}{\partial S_1^{(nohl)}} \times \frac{\partial S_1^{(nohl)}}{\partial w_{1,1}^{(nohl)}}. \tag{5.29}$$

By recalling the delta rule:

$$\hat{w}_{1,1}^{(nohl)} = w_{1,1}^{(nohl)} - \eta \times \left[\left(\sum_{\alpha=1}^{t} \delta_\alpha^{(nohl+1)} \times w_{\alpha,1}^{(nohl+1)} \right) \times \frac{\partial y_1^{(nohl)}}{\partial S_1^{(nohl)}} \times \frac{\partial S_1^{(nohl)}}{\partial w_{1,1}^{(nohl)}} \right]. \tag{5.30}$$

The weight updating is performed for all hidden layers until the input layer. With a generalized form, any $w_{i,j}^{(\varsigma)}$ update in between the $(\varsigma-1)^{th}$ and $(\varsigma)^{th}$ layers is written in equation (5.31). υ is the $(\varsigma+1)^{th}$ layer size. $f^{'}$ is the activation function derivative.

$$\hat{w}_{i,j}^{(\varsigma)} = w_{i,j}^{(\varsigma)} - \eta \times \left[\left(\sum_{\alpha=1}^{\upsilon} \delta_{\alpha}^{(\varsigma+1)} \times w_{\alpha,i}^{(\varsigma+1)} \right) \times f^{'}(S_i^{(\varsigma)}) \times y_j^{(\varsigma-1)} \right] \tag{5.31}$$

Now, the learning concept for the conventional MLP has been explained. Let us present the basic structure of stochastic MLPs (or SCNNs in general) utilizing bitstreams. From the software simulation point of view, the implementation details are given by considering the physical (actual) bitstream processing (until Section 5.4). Therefore, first, the concepts based on physical bitstreams are discussed; however, Section 5.4 proposes the CT-based simulation of SCNNs with an efficient runtime.

Figure 5.8 shows the main operations of SCNN architecture for the input layer. The first difference from the conventional NN architectures is the data representation. The input layer is composed of the image pixels, which are normalized and represented in BPE bitstreams. Let X be a grayscale image pixel value and N be the bitstream size. By using bold font for stochastic bitstreams, $\boldsymbol{x}$ denotes the bitstream representation of a normalized pixel value, $\frac{X}{N}$, where $P_{BPE} = \frac{(\lfloor \frac{X+N}{2} \rceil)}{N}$. Network weights are also represented in stochastic bitstreams by denoting $\boldsymbol{w}$.

In SCNN, the multiplication is obtained using the XNOR logical operator, simplifying the multiplier hardware in physical architecture via BPE-based bitstreams. For instance, neuron output ($\boldsymbol{x_1}$) and weight ($\boldsymbol{w}_{1,1}^{(1)}$) bitstreams are multiplied as ($\boldsymbol{x_1} \odot \boldsymbol{w}_{1,1}^{(1)}$). Figure 5.8 also presents the preliminary information on the accumulation operation. In addition to APC, MUX, and counters, which are well-known hardware-based

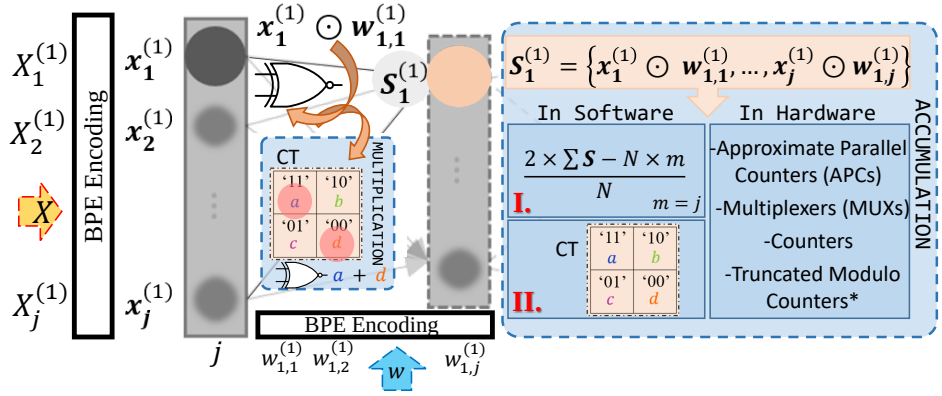


Figure 5.8 : SCNN: Software- and hardware-related implementation details.

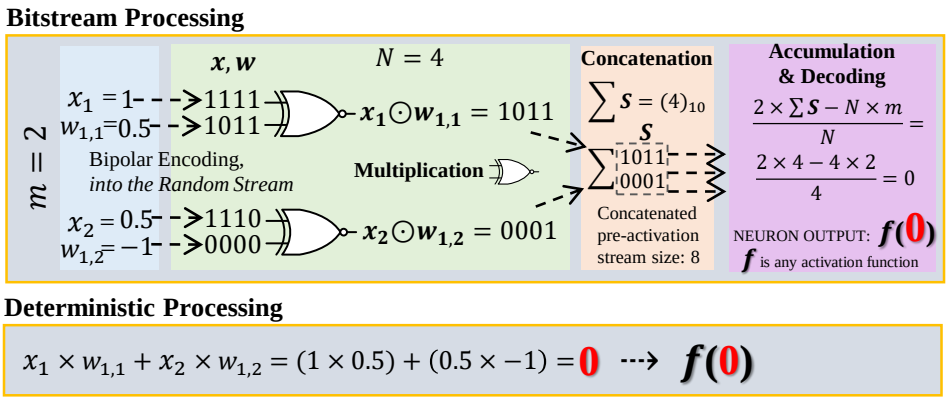


Figure 5.9 : Bitstream and deterministic processing examples for neuron arithmetic.

accumulators, the utilization of truncated MOD counters is proposed for SC-adapted BNNs, as will be discussed in Section 5.3.2.1. Regarding the software simulation of SC accumulator, two methods are discussed: Method I. Physical bitstream emulation considering the BPE decoding, and Method II. CT-based approach. The latter is discussed in Section 5.4. For the former, I, the multiplied values are concatenated to create a pre-activation bitstream, $\boldsymbol{S}$. This is not a final accumulation value; however, a decoding procedure is applied to obtain the integer sum. The proposed addition of bitstreams in simulation is based on the network type. Suppose the network *ws* are in full-precision. Then, the final addition of multiplied values from the concatenation is obtained as $\frac{2\times\sum \boldsymbol{S}-N\times m}{N}$, where m is the number of the adding terms (with an inspiration of integer stochastic streams by [154]). This expression is derived from the BPE, where $\sum 1s = \lfloor \frac{X+N}{2} \rceil$ in each of the N-sized bitstreams. The total count of *logic-1s* in the concatenated stream size of $N \times m$ is calculated considering $\sum 1s$, which is already an integer value; $\sum 1s = \sum \boldsymbol{S} = \frac{sum+(N\times m)}{2}$, yielding $sum = 2 \times \sum \boldsymbol{S} - N \times m$. The normalized value is obtained by dividing N: $\frac{2\times\sum \boldsymbol{S}-N\times m}{N}$.

In Figure 5.9, the neuron operations are exemplified by comparing bitstream and deterministic processing. Using the accumulation Method I in Figure 5.8, multiplied bitstreams are concatenated and processed. The deterministic processing result is the same as in bitstream processing. The activation is not discussed and is kept in the deterministic domain since the BNN-related bitstream processing will present a design without an activation module (Section 5.3.2.1). Beforehand, let us define emerging NNs, starting with the traditional BNNs.

5.3 Emerging Neural Networks

This section discusses efficient and robust architectures, which are referred to as "emerging neural networks." First, the foundations of conventional binarized networks are given to set a background. Then, the BSBNN architecture proposed within this thesis's scope is presented.

5.3.1 Binarized neural networks (BNNs)

NN implementations vary from platform to platform when observed from the hardware perspective. Even though the training environment can be a powerful graphics processing unit (GPU), the test network may be binding to the limited hardware resources in real-life applications. BNN quantizes weights and neuron activations into one bit; hence, network architecture alters with substantial improvements in terms of complexity. Conventional BNN aims to create a lightweight architecture by only processing the single-bit data of the *w*s. This is advantageous both for memory and operations that now require gate-only hardware resources.

Recalling the definition of S for any deterministic NN, $S_m^{(n)}$ denotes the m^{th} neuron pre-activation related to the n^{th} hidden layer (HL). Pre-activation, $S_m^{(n)} = (x_1 \times w_{m,1}^{(n)}) + \cdots + (x_j \times w_{m,j}^{(n)})$, is written using the multiplied and accumulated values of preceding neuron (or input) values, x, and the weights, w, for j neurons in the $(n-1)^{th}$ layer. In the hidden layers of the deterministic BNN, scalar weight and neuron output values are ± 1 regularized during the training. Then, they are mapped into 1-bit binary values such as $(+1)_{10} \equiv (1)_2$ and $(-1)_{10} \equiv (0)_2$. Implementing bitwise multiplication using a single XNOR gate is easy, just like in SC. Table 5.1 compares scalar and bitwise multiplications. It is noted how scalar multiplication is reduced to the simple logic operation via XNOR gates.

Table 5.1 : Multiplication in scalar and binary values.

Scalar	Binary	Scalar versus Binary
$+1 \times +1 = +1$	$1 \odot 1 = 1$	$(+1)_{10} \equiv (1)_2$
$-1 \times -1 = +1$	$0 \odot 0 = 1$	
$-1 \times +1 = -1$	$0 \odot 1 = 0$	$(-1)_{10} \equiv (0)_2$
$+1 \times -1 = -1$	$1 \odot 0 = 0$	

Courbariaux et al. explained the binarization procedure of BNNs using *deterministic* and *stochastic* approaches. The former considers a sign function, $f(x) = sign(x)$, which returns $+1$ if $x \geq 0$, and -1 otherwise. The second binarization procedure is a stochastic process that utilizes the *hard sigmoid* function, $f(x) = max(0, min(1, \frac{x+1}{2}))$. The piecewise function gives $+1$ with a probability $p = f(x)$ or gives -1 with probability $1 - p$. Nonetheless, Courbariaux et al. underlined that the necessity of random bit requirement makes the latter binarization method harder during quantization [213–215]. As a result, the simple deterministic binarization procedure (`SIGN(w,b)` = `BINARIZE(w,b)` in Algorithm 3) is applied in this work.

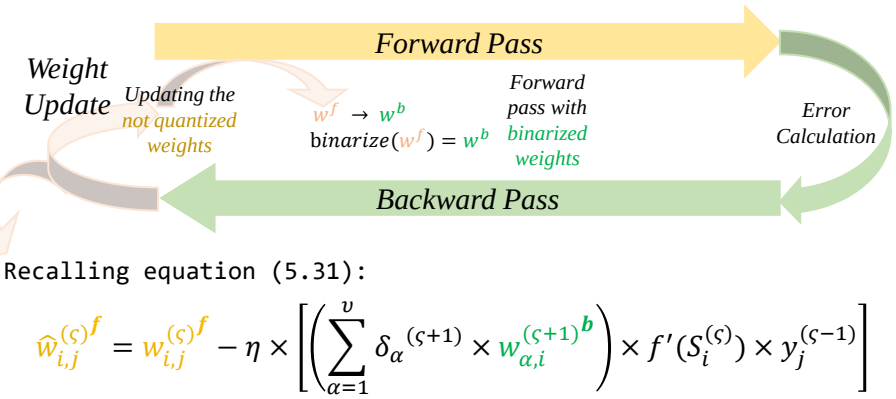


Figure 5.10 : The training strategy in BNN.

Figure 5.10 summarizes the main steps of the BNN training. The binarized weights are utilized in the forward pass, and the total error is obtained accordingly. The backward pass also requires the binarized weights and activations, while the weight update is applied over the full-precision version of the weights. Therefore, two copies of the weights: (i) fully precise, w^f, and (ii) binarized, w^b (or simply w, ± 1), scalars are considered during the training. Figure 5.10 also shows how the generalized weight update in equation (5.31) from the previous section is updated for BNN [216].

Algorithm 3 presents the flow of BNN training. The weight (w^f) and bias (b^f) values are initialized as full-precision. Xavier initialization [217] is used in this step. The binarization for the forward pass is shown at the beginning of each batch processing, i.e., after each weight update. Binarized scalars, w and b, are considered for forwarding error calculation. After batches are processed at the end of the epoch, binarized weights are utilized for validation accuracy. The training considers batch processing for the proper utilization of stochastic gradient descent. For simplicity, batch normalization,

Algorithm 3 BNN Training Flow

Require: input data: X ▷ $inputSize = size(X)$
weights and biases: w, b ▷ full-precision: w^f, b^f ▷ binarized scalars: w, b
pre-activation: S, neuron output: y
$totalEpoch$, $batchSize$, $hiddenLayers$

Ensure: w, b

$w^f, b^f \leftarrow Initialization$
function TRAINBNN(w^f, b^f)
  for $epoch = 1$ to $totalEpoch$ **do**
    for $batch = 1$ to $(inputSize/batchSize)$ **do**
      procedure BINARIZE(w^f, b^f)
        if $w^f \geq 0$, $b^f \geq 0$ **then** ▷ for each item
          $w \leftarrow +1$
          $b \leftarrow +1$
        else
          $w \leftarrow -1$
          $b \leftarrow -1$
        end if
      end procedure
      for $batchElement = 1$ to $batchSize$ **do**
        for $layer = 1$ to $hiddenLayers + 1$ **do**
          DOFORWARD($X, w, b, layer$) ▷ binarized neuron outputs
                ▷ except at the input
        end for
      end for
      for $layer = hiddenLayers$ to 1 **do**
        $\frac{\partial E_{total}}{\partial w} \leftarrow$ DOBACKWARD(x, S, y, w, w^f, b, b^f)
      end for
      w^f, $b^f \leftarrow$ DOWEIGHTUPDATE($x, S, y, w, w^f, b, b^f, \eta, \frac{\partial E_{total}}{\partial w}$)
    end for
    CROSSVALIDATION(w, $b \leftarrow$ BINARIZE(w^f, b^f))
  end for
return $w, b \leftarrow$ BINARIZE(w^f, b^f)
end function

dropout, and learning rate decay are not included in Algorithm 3. They are denoted as high-standard training options, which ensure significant accuracy improvements.

Forward pass keeps the binarization for the neuron outputs using the *sign* function, except for the input layer (input data). The proposed scheme in the next section will treat inputs in the stochastic bitstream representation, which fits the SCNN architecture. On the other hand, [213] underlines the problem of the *sign* function related to its derivative during backpropagation. The same solution via a "straight-through estimator" is followed.

5.3.2 Bitstream processing binarized neural networks (BSBNNs)

A few studies thoroughly investigated the use of bitstream processing in the context of NNs [19, 20, 154, 218]. Most of the preceding bitstream processing NNs operated at either full-precision or 2-or-more bit quantization of the weights and activations. Nonetheless, Hirtzlin et al. [219] first proposed the SC-based BNN with an improved training strategy by introducing stochastic binarized input data into the network during training. They enhanced stochastic network accuracy closer to the deterministic one for the Fashion-MNIST dataset [220]. The main processing units were XNOR gates for multiplication, counters for accumulation, and subtractors for the activations of neurons. This work proposes a different approach that uses bitstream processing for all layers except for the output, achieving an entirely bitstream-based binarized network, i.e., BSBNN. The output layer is not binarized since the classification requires an activation function giving values different than 1 and -1. In the literature, the SVM output layer and softmax-based solutions are present [213, 219]. This thesis sets a computational framework that derives the neuron activation from the accumulation unit and thereby avoids the subtraction implemented in the neuron activation unit. Hence, an efficient hardware architecture with reduced HRU is obtained. Furthermore, the robustness of BSBNN is underlined by showing the fragility of conventional BNN [77].

Next, the bitstream operations of the proposed BSBNN are presented. As already known from previous sections, the multiplier is either an AND gate for UPE bitstreams or an XNOR gate for BPE bitstreams. Since the weights can be $+1$ or -1 in the binarized network case, BPE streams are adopted in BSBNN; the XNOR gate implements the multiplication.

Different from the prior art [219], bitstreams are utilized for image pixel values, neuron outputs, and network weights. Bitstream processing is kept throughout all hidden layers. Thus, a -1 weight value in deterministic BNN is converted into $\boldsymbol{w} = \{0, 0, \ldots, 0_N\}$ for bitstream processing. Likewise, $+1$ is converted into $\boldsymbol{w} = \{1, 1, \ldots, 1_N\}$ for bitstream processing. This way of representing weights and hidden values improves the bit-flip soft error robustness. Accumulation in BSBNN is a new proposal, inherently incorporating activation operation thanks to SC, as explained in the following subsection.

5.3.2.1 Activation module-free network model

In a deterministic BNN, accumulation is performed by counting the population of 1s [221]. Recent efforts of in-memory computing-based BNNs show that considering the threshold operation while counting can merge activation into the accumulation [222, 223]. Efficient implementation of counter and thresholding circuitry is essential. To tackle this in SC-based binarized architectures, the utilization of masking logic with a counter is proposed, thanks to BPE-based bitstreams. Below, let us first define the simple asynchronous counter in Definition 5.2.

Definition 5.2 (Asynchronous Counter)**.** Let T be the total number of states of the counter. Including a *zero* initial accumulation value, the counter can count up to the final state, $(T-1)$. It naturally resets to zero when the total number of 1s presented to the input has reached a value that is the number of flip-flops (ffs) to the power of two, i.e., $2^{(\#\ of\ ffs)}$.

Figure 5.11 depicts an example of an asynchronous counter based on the D-type ffs for population count. The term *popcount*, short for population count, is generally used to denote the basic counter logic hardware [213, 219, 224]. Binary inputs are fed sequentially to the first flip-flop clock input, and the base-2 output, $(Q_3Q_2Q_1)_2$, gives the total number of 1s in the input binary word.

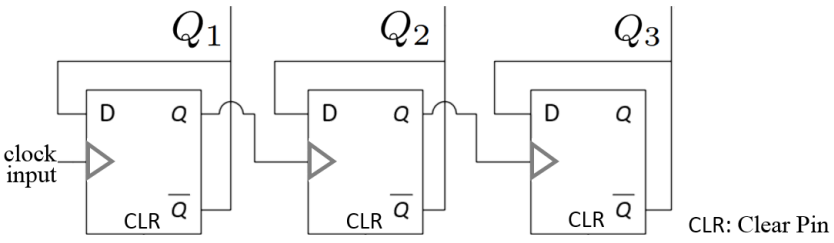


Figure 5.11 : $T = 2^3 = 8$ states asynchronous counter example utilizing 3 D-type ffs.

In BSBNN, a similar hardware to the popcount is utilized. Ultimately, the neuron activation is obtained directly from the accumulation. The concatenated pre-activation bitstream, $\left[(\boldsymbol{x_1}\odot\boldsymbol{w}^{(n)}_{m,1}),\ \ldots,(\boldsymbol{x_p}\odot\boldsymbol{w}^{(n)}_{m,p})\right]$ is considered, where $(\boldsymbol{x_i}\odot\boldsymbol{w}^{(n)}_{m,i})$ represents the multiplication bitstream related to the i^{th} neuron from the preceding layer. The concatenated bitstream is inputted to a counter, and a custom architecture not only counts the 1s but also decides the neuron output. Considering the BPE, since half of the bitstream size with *logic-1*s is a critical point that is equal to a zero (recalling

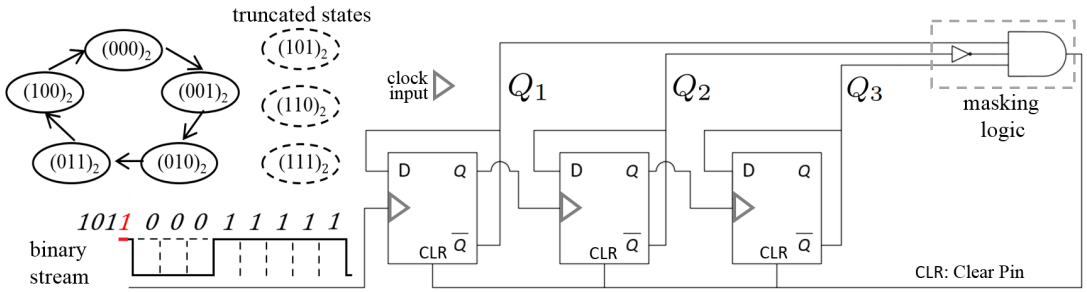


Figure 5.12 : Popcount presented in Figure 5.11 is truncated from $T = 8$ states into $T = 5$.

Table 2.1), $\frac{N}{2} + 1$ *logic-1*s in a bitstream indicates $+1$ scalar, and $\frac{N}{2} - 1$ *logic-1*s in a bitstream indicates -1 scalar. This helps to use a simple MOD counter to determine whether or not to trigger the neuron output without using an activation function. Our proposed solution uses a counter that resets itself and forwards a positive neuron activation as soon as the number of 1s observed in the concatenated bitstream has attained a threshold (i.e., 1s in majority), corresponding to the half size of this concatenated bitstream. This structure is called a *truncated MOD-T counter* and is presented in Definition 5.3.

Definition 5.3 (Truncated MOD-T Counter)**.** The counter combines a popcount with a specific masking logic to reset the popcount once it reaches the predefined hard-coded value T [225] since the total number of states is not always a power of two. This concept is denoted as the *truncated MOD-T counter*. The term *truncated* indicates that the counter resets before the maximal value of the popcount is reached, while *MOD-T* indicates that T is the modulo value of the counter. At every increment, the current total sum (in binary) is presented to the bit masking logic to check whether the threshold is attained.

The counter truncation is exemplified in Figure 5.12 for the popcount presented in Figure 5.11. After the output, $(Q_3Q_2Q_1)_2$ would have gone through 5 states, from $(000)_2$ to $(100)_2$, and it resets itself by setting the clear pin. To reset after 5 states, the output of the popcount is masked by a gate that outputs a $(1)_2$ only if $Q_1 = (1)_2$, $Q_2 = (0)_2$, $Q_3 = (1)_2$. This can be implemented with an AND gate, as shown in Figure 5.12. As 1s constitute a majority in the input binary stream, whenever the 6^{th} high value is seen on the input, the masking logic outputs $(1)_2$.

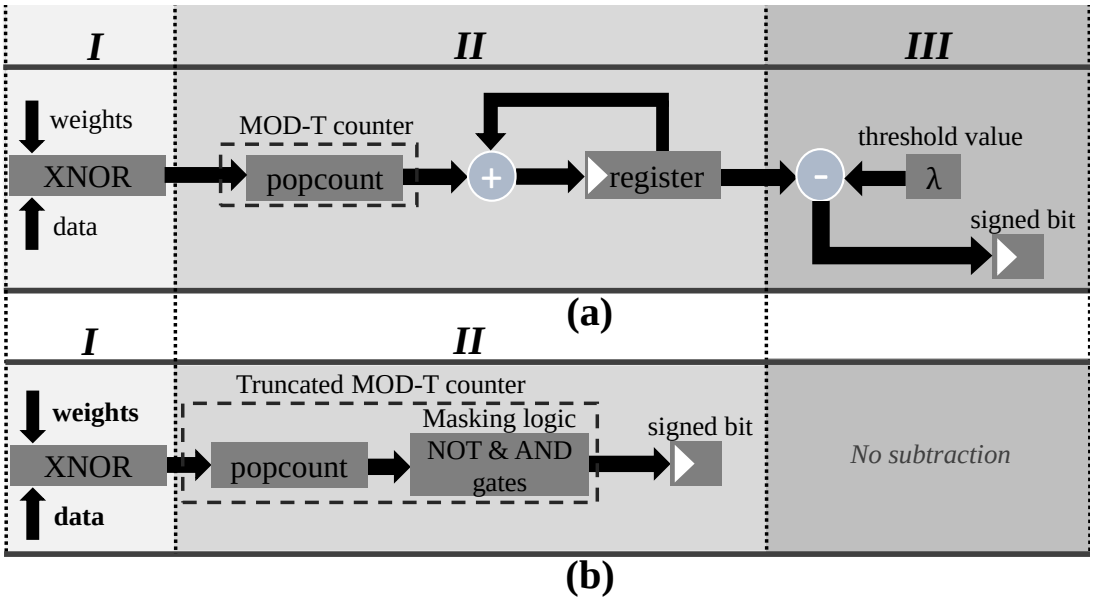


Figure 5.13 : Architecture of prior bitstream processing BNN and the proposed model: (a) Hardware units of prior art by Hirtzlin et al. Region *I*: multiplication, Region *II*: population count and cumulative addition, Region *III*: decision of activation by subtraction. (b) Architecture of our proposed framework. The signed bit is 0 by default and is set to 1 if and only if the counter reaches a predefined threshold that has been hard-coded in the masking logic. Region *I*: multiplication, Region *II*: population count & masking logic = truncated MOD-T counter.

As depicted in Figure 5.13, our proposal to use a truncated MOD-T counter in BSBNN simplifies the hardware in charge of turning the concatenated bitstream into a neuron activation. Former solutions [219], given in Figure 5.13 (a), simply use a register to accumulate the number of 1s counted in each of the input products and then rely on a subtraction to compare the accumulated sum with the pre-encoded threshold. In contrast, our proposed solution, shown in Figure 5.13 (b), parses the concatenated products bitstream and embeds the pre-encoded threshold T directly in the truncated MOD-T counter. It does not need any extra register nor subtraction other than the masking logic.

Figure 5.14 provides a numerical example to clarify the overall proposal. By assuming this is the input layer of an NN, input values are exemplified as 1, 0.5, 0.5, and weight values are 1, -1, 1. The overall concatenated bitstream size is 12 bits because there are three neurons, and each stream is 4 bits. The threshold is $T = 5$, including the $zero^{th}$ state, indicates $(101)_2$ hard-coded value in base-2 representation. To control the neuron activation bit, i.e., signed bit, the masking logic is performed at the output

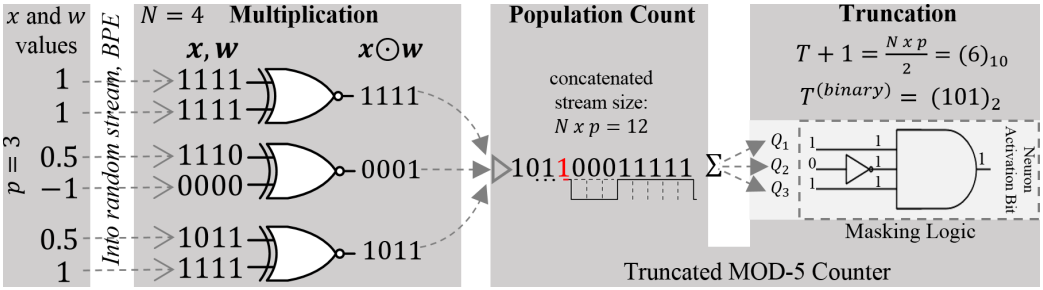


Figure 5.14 : Input layer example of BSBNN with $p = 3$ neurons.

Table 5.2 : Hardware units in BNN and BSBNN.

	Deterministic BNN	BSBNN
Multiplication	XNOR Gates	XNOR Gates
Addition	Popcount	Truncated MOD-T Counter
Activation	Subtraction Unit	-

of the truncated MOD-5 counter that has been shown in Figure 5.12. To conclude, hardware units in deterministic BNN and BSBNN are compared in Table 5.2. Our proposal degrades the activation operation into the accumulation itself with simple logical operators, thereby having efficient and reduced hardware. Table 5.2 reveals that BSBNN does not require subtraction for the activation operation while using bitstream processing.

For hardware validation, a hardware simulation framework was built in MATLAB Simulink with design primitives from Xilinx System Generator to compare the complexity of the conventional and bitstream-based computational paradigms. The input layer data for the BNN and BSBNN hardware simulations were transferred from the MATLAB workspace. On the software side, BPE-based random bitstreams having a binomial distribution were created and used for BSBNN input data as described in [20]. A parallel co-simulation was conducted by feeding the hardware model associated with the forward pass hidden layers. The overall count of the HRU (look-up tables and registers) for the single BSBNN neuron hardware proposed in Figure 5.13 (b) was 101 for N = 8-bit and 147 for the traditional deterministic BNN neuron in Figure 5.13 (a); thereby, representing a gain of around 30%. The design synthesis was carried out with the Vivado tool, having set the Zynq-7000 as the target device. BSBNN and BNN neuron primitives were pre-designed and tested in the MATLAB Simulink before transferring each unit-under-test module into the Vivado simulation.

5.3.2.2 Soft error- and data corruption-aware neural network model

This section extends the proposed methodology to the training strategies and inference networks for the non-idealities mentioned in Section 2.5. Network training strategies are discussed, offering efficient and robust architecture against soft errors applied to input data (images) and network weights [77], as well as image-based corruptions. While the previous efforts use analog or digital simulation/design environments with a limited range of datasets, this study exploits the power of the software for bulk simulations. It is underlined that powerful software simulation is required, especially for retraining a network with dense datasets having sub-classes (MNIST-C [87]).

First, the soft error performance of binarized architectures is inspected, considering bit-flips (by recalling Figure 2.6). Figure 5.15 (a) shows that the original MNIST dataset is used in the experiments, and MNIST images are pre-processed for pixel-wise bit-flips in the test phase. Figure 5.15 (b) shows that after clean training, the model weights have bit-flips in the test network via randomly flipping binary weights as $1 \rightarrow 0$ & $0 \rightarrow 1$. Conversely, the training in Figure 5.15 (c) consists of input images having pixel-wise bit-flips. The test network has image-based bit-flips, as well. Similarly, Figure 5.15 (d) shows that both training and testing networks have independent random bit-flips on the network weights. From a data representation point of view, traditional binary data in BNN is known to be quite vulnerable to bit-flipping errors [226]. This is attributed to it being quite sensitive to errors affecting the MSBs of input data and the 1-bit-only weights. In contrast, increased robustness is expected from the bitstream processing paradigm. Hence, bit-flip error injection has been considered to measure the robustness of the proposed bitstream processing framework to errors that can happen during memory accesses of images and weights [20, 219, 227, 228].

A reference model using deterministic BNN with no error training is considered first during simulations. The forward pass of the network uses binarized weights. However, the weight update is performed considering a floating-point version of the weights, which are then binarized to define the forward pass as explained in Algorithm 3. The network comprises four layers, each of them including 784, 1024, 1024, and 10 neurons, respectively. This network structure with $nohl = 2$ having 1024 neurons in each hidden layer is comparable and fair with respect to the previous MNIST-based classifiers targeting SCNN, as presented in Figure 2.8. Squared hinge loss is chosen,

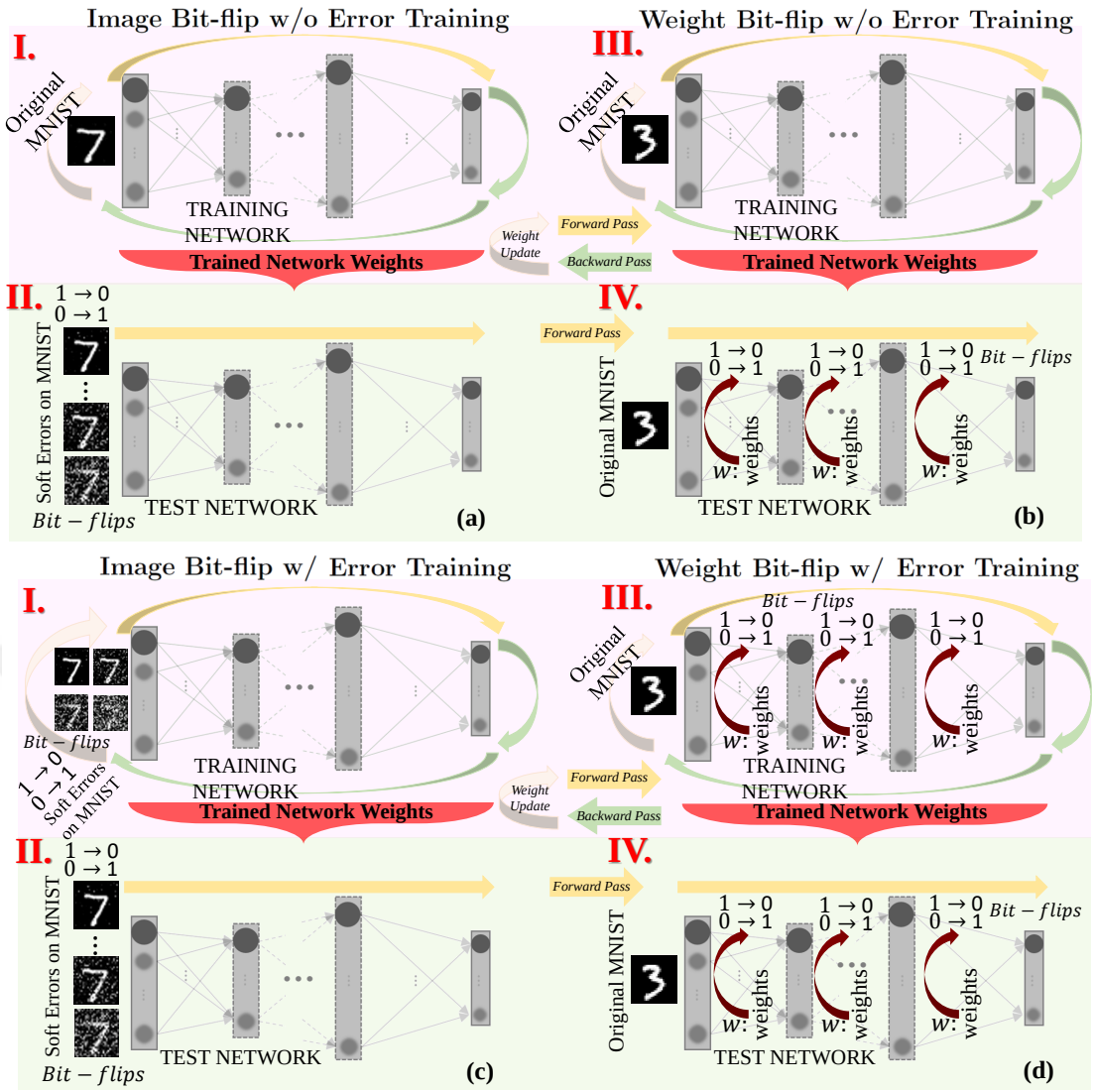


Figure 5.15 : Training and testing strategies of image and weight bit-flips in BNN architecture: (a) image bit-flips applied only in the test phase, (b) weight bit-flips applied only in the test phase, (c) image bit-flips applied both in training and testing, and (d) weight bit-flips applied both in training and testing.

which outperforms cross-entropy loss in terms of validation accuracy. An exponential learning rate decay is applied in the range of $[3 \times 10^{-7},\ 3 \times 10^{-3}]$. The batch size is set to 100. Dropout is implemented with $p = 0.2$ for the input layer, and with $p = 0.5$ for hidden ones. The best-performing model is saved by considering the best validation accuracy, 98.43%, at the 968^{th} epoch over 1000 epochs. The deterministic BNN test accuracy is 98.27%. Our training, validation, and test sets are always different from each other for all experiments regarding BNN and BSBNN.

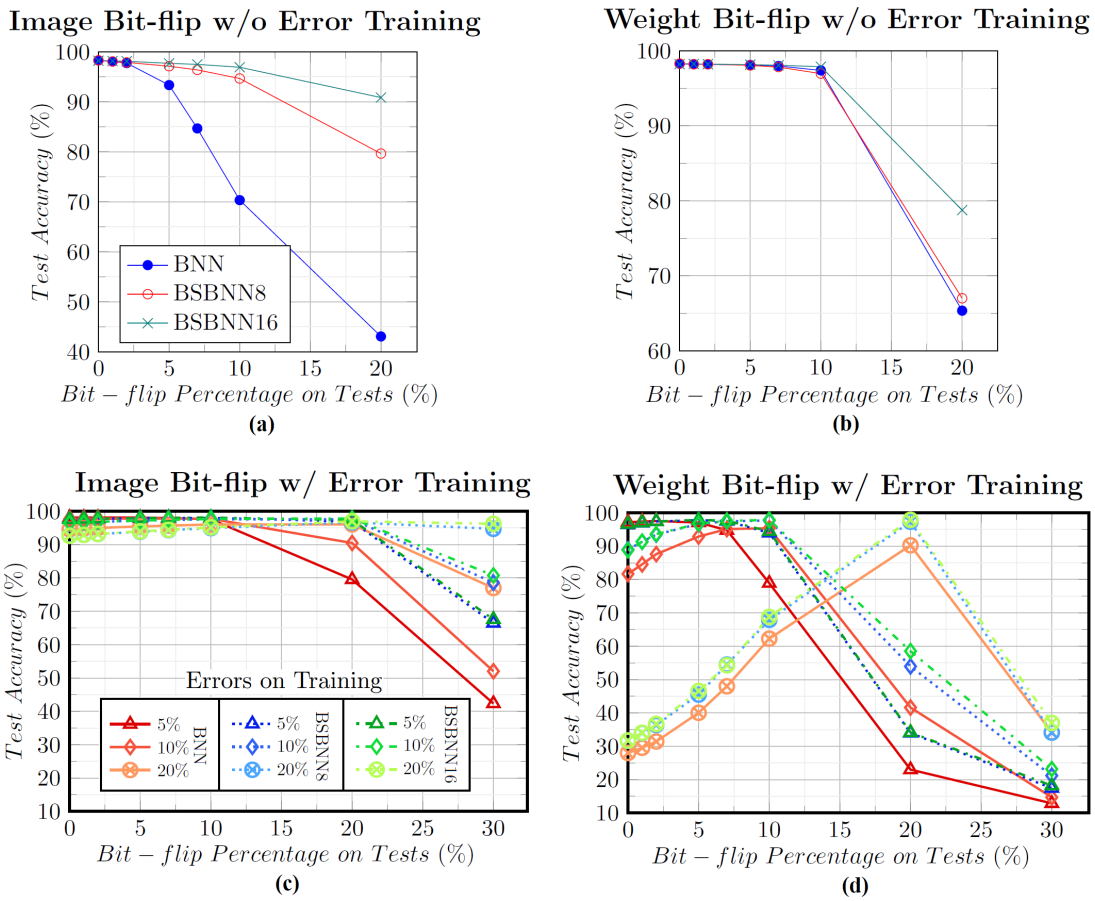


Figure 5.16 : Bit-flip soft error injections applied to BNN, BSBNN8, and BSBNN16. Error injection is defined in terms of the percentage of flipped binary symbols. (a) Models trained without error are tested for image bit-flips. (b) Models trained without error are tested for weight bit-flips. (c and d) Models are trained in the presence of (image or weight) errors, with the processing paradigm used at test time (dataset sizes: training set: 50000, validation set: 10000, test set: 10000 images).

The experiments considering cases in Figure 5.15 (a) and Figure 5.15 (b) are further conducted. The pixels of test images are manipulated in 8-bit traditional binary representation for deterministic BNN. For BSBNN, the image data are represented in $N = 8$ (BSBNN8) or $N = 16$ (BSBNN16) sized streams. Figure 5.16 (a) depicts how network accuracy decreases with the percentage of randomly flipped symbols. It is monitored that the conventional processing paradigm suffers from dramatic accuracy losses while the BSBNN achieves remarkable robustness. Regarding binary weight flipping, it is observed in Figure 5.16 (b) that BNN and BSBNN8 achieve close performance, while BSBNN16 demonstrates a non-negligible but significantly reduced sensitivity to weight bit errors.

During the second round of simulations, considering cases in Figure 5.15 (c) and Figure 5.15 (d), it is investigated how accounting for the errors and the processing paradigm at training time affects the robustness to errors at the test time. Therefore, different models are trained with different bit-flipping rates (5, 10, or 20%). This is done using either the conventional (BNN) or the bitstream processing computational paradigm (BSBNN), and the errors affect either the image pixels or the weights. The resulting models are tested with the same computational paradigm as the one used at training, and their test accuracy is reported in Figure 5.16 (c) and Figure 5.16 (d), as a function of the bit-flip percentage at test time. Figure 5.16 (c) shows that injecting errors on image pixels at training increases the robustness to errors at test time by a large margin (about 30% accuracy improvement at 30% error rate), and the bitstream processing paradigm is significantly more robust to errors than the conventional one. With regard to the errors affecting weights, Figure 5.16 (d) indicates that accounting for errors is largely beneficial when they are also present at test time but the performance is penalized in absence of errors at test time.

Moreover, the performance of NNs is monitored when image corruptions are present. In addition to binarized architectures, SCNN and full-precision NN (FPNN) are also tested. Hence, four types of neural networks are monitored:

(i) **BNN:** Conventional approach in Figure 5.13 (a).

(ii) **BSBNN:** Proposed approach in Figure 5.13 (b).

(iii) **SCNN:** Software model of stochastic bitstream processing with binomially distributed actual bitstreams of inputs, weights, and neuron outputs in BPE (proposed emulation approach in Figure 5.8 as a general framework) [229].

(iv) **FPNN:** Standard NN with fixed or floating point representation (No binarization, quantization, or stochastic representation applied).

In Figure 5.17, the motivation of the training strategy is depicted to summarize the idea of corruption training. The test set includes image corruptions that may occur on the sensor, during the data transfers, or on the storage. The training procedure is performed with two perspectives: (i) In Figure 5.17 (a), a dataset without corruptions (MNIST) is utilized, and (ii) in Figure 5.17 (b), corrupted images (MNIST-C) are introduced in training for each corruption subset.

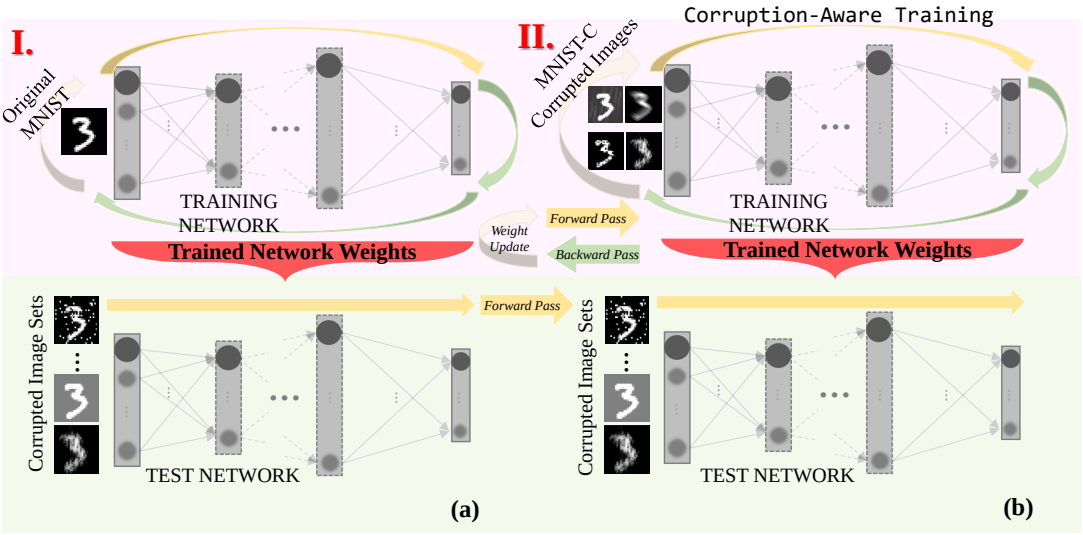


Figure 5.17 : Image corruptions from MNIST-C: (a) Corruptions only in the test, and (b) Corruption-aware training.

Table 5.3 : Details of deterministic networks.

	BNN	**FPNN**
Network Structure	784-1024-1024-512-10	784-200-100-10
Hidden Layer Activation	*Sign*	*Sigmoid*
Last Layer Activation	*Softmax* with Squared Hinge Loss	*Softmax* with Cross-Entropy Loss
Batch Size	100	200
The Best Epoch	784	615
Exponential Learning Rate Decay Range	$[3 \times 10^{-7},\ 3 \times 10^{-3}]$	$[7 \times 10^{-7},\ 7 \times 10^{-2}]$

While testing the four networks, the structure of deterministic networks is decided first. At this stage, the architecture is determined to achieve the best-generalized results for the cases in Figure 5.17 (a) and Figure 5.17 (b). Parameters in Table 5.3 for these universal architectures are also used for BSBNN (Figure 5.13 (b)) and SCNN (Figure 5.8), which are bitstream processing counterparts of BNN and FPNN.

In Figure 5.18 (a), the results of MNIST-C corruption types for the independently trained model with the MNIST standard dataset are presented. Since the model has not learned the corruptions yet, several subsets such as *contrast*, *fog*, *frost*, *inverse*, *stripe*, and *snow* severely suffer from the accuracy drops in all network types. On the other hand, surprisingly, the accuracy performance of subsets such as *dotted line*, *shot noise*, *JPEG compression*, *spatter*, *pixelate*, *quantize*, *speckle noise*, and *zoom blur* is remarkably high for all NN types.

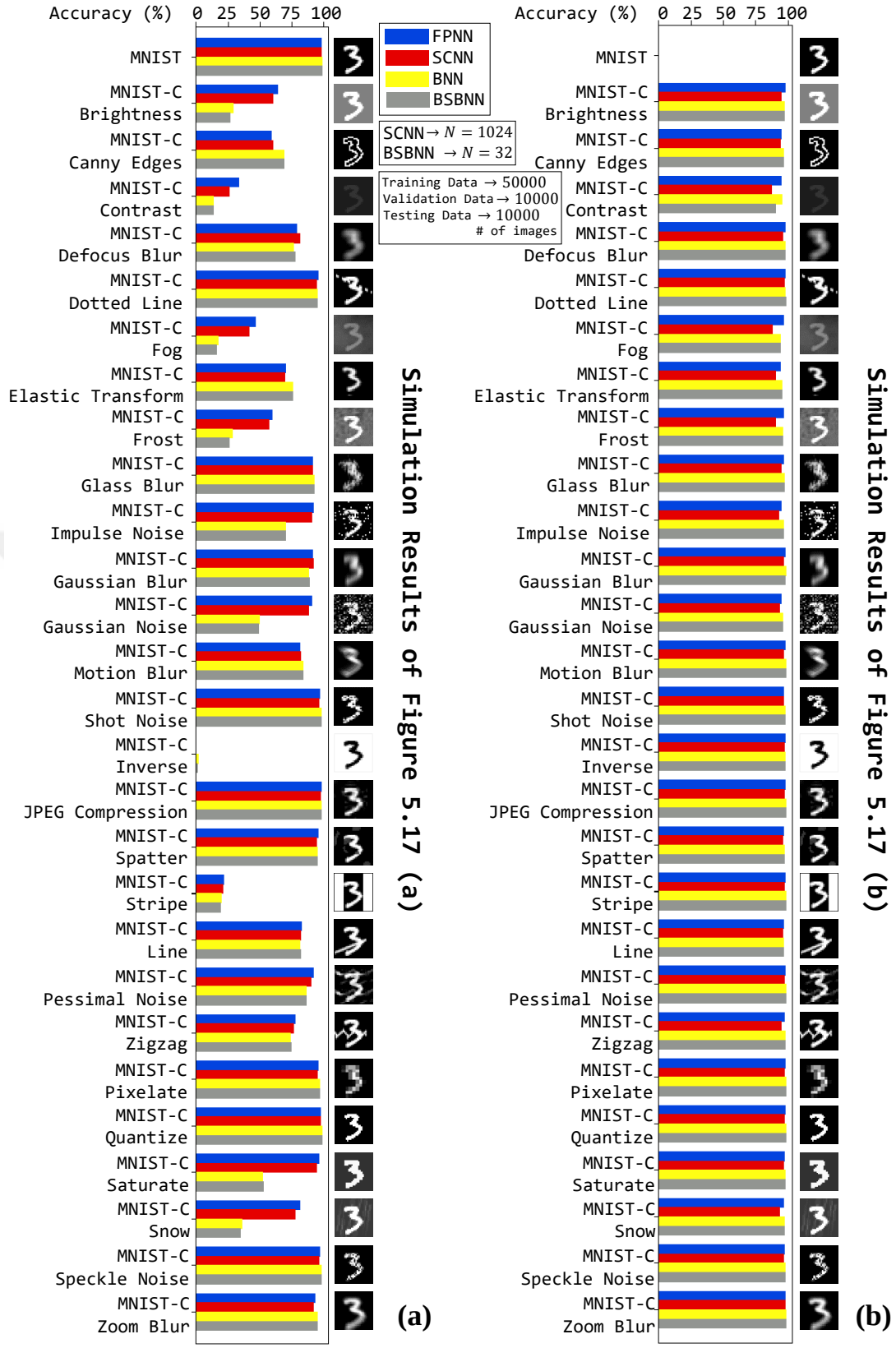


Figure 5.18 : Experiments with MNIST-C: (a) no corruption awareness during training, and (b) corruption-aware training.

In the corruption-aware training, all independent corruptions are presented in training separately. When independent tests are applied, the best over-learning-free accuracy results are shared in Figure 5.18 (b) for each corruption type. The severe accuracy drops in previous experiments in Figure 5.18 (a) improve for all subsets. Last but not least, in gray bar plots, it is underscored how BSBNN performs well in each corruption case showing comparable performance to the FPNN.

5.4 Fast Training of Stochastic Computing-Based Neural Network (SCNN): New Perspectives Toward CT Simulation-III

This section presents new perspectives of CT-based SC simulation for NNs. Accuracy and runtime simulations with CT are conducted to diversify our experiments for CNN, deeper network architecture with convolutional layers. (Here, SCNN is the general stochastic NN, not stochastic convolutional NN.) Plus, an MNIST-like dataset (Kuzushiji-MNIST [230]) is utilized with a more challenging classification task.

The runtime and accuracy performance of (i) direct bitstream processing and (ii) Algorithm 2 for an SCNN are investigated. The network architecture in Figure 5.19 (a) is tested, particularly its classifier part [86]. The convolutional feature extraction part is conventionally handled without a stochastic approach. The obtained features are sourced into the fully connected classifier. The data are represented in a stochastic manner for SC-based multiplication when needed. BPE-based multiplication, i.e., the XNOR logic operation, is performed for neuron outputs and the network weights as $x \times w$. Different from multiplication, addition and activation operations are conducted in a deterministic manner. Therefore, the direct bitstream processing method requires *from-stochastic-to-deterministic* data conversion and vice versa, while the CT method does not. The CT holds only a, b, c, and d, preserving information related to the *correlation* and *resultant TCO*. This is another advantage of the CT technique since the independent circuit operations and their effects can still be monitored while no data conversion is required by setting the essential deterministic processes aside. In our analyses, all operations are executed in a GPU. In Figure 5.19 (b), the per epoch classifier runtime of the SCNN training is depicted. The numbers of neurons in each HL and the batch size (BS) are arranged in four different ways. For $N > 128$, CT approaches (a_{mid}, a_{poly}, and a_{best}) are much faster than direct

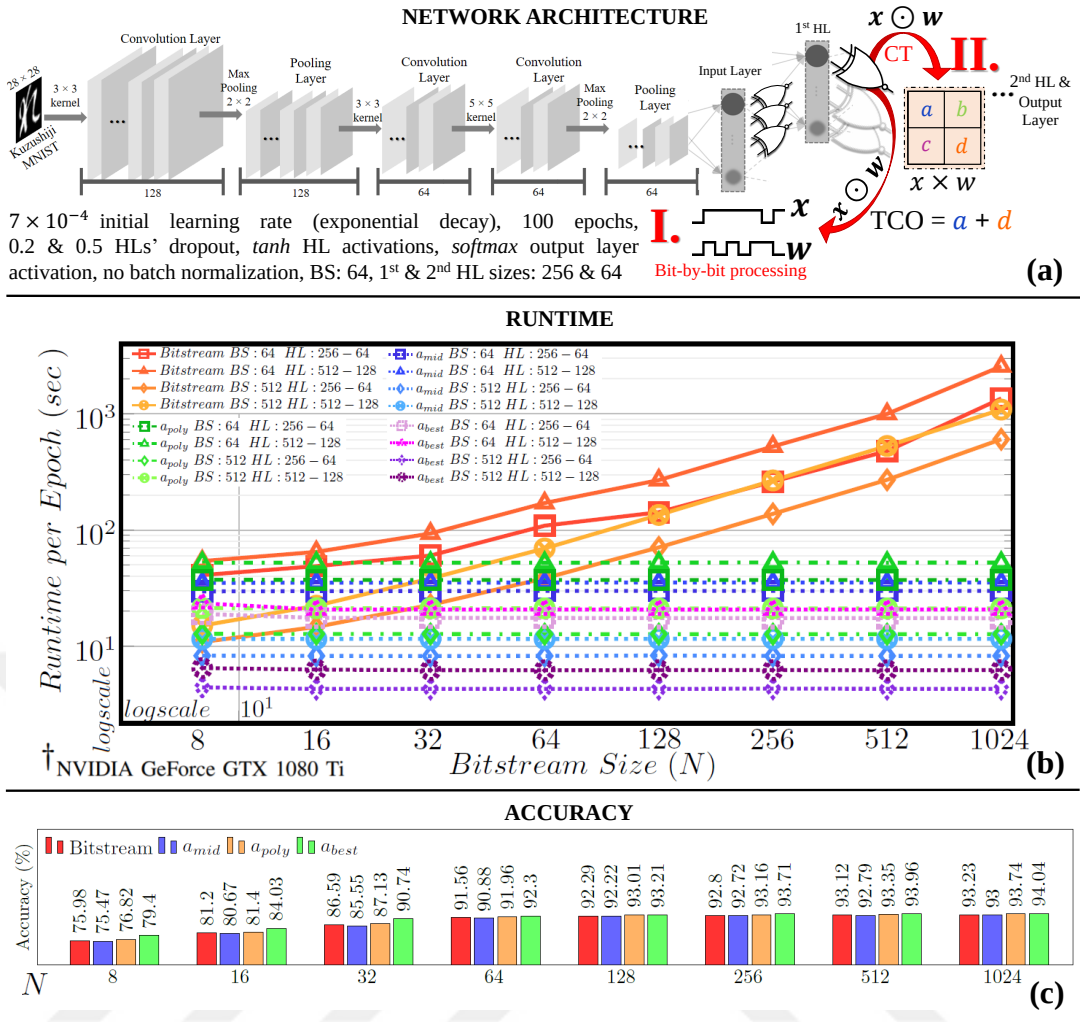


Figure 5.19 : (a) SCNN architecture, (b) classifier runtime, and (c) accuracy performance. A dataset of Japanese characters, Kuzushiji-MNIST, is utilized (48000 training, 12000 validation, and 10000 test images).

bitstream processing, which exponentially depends on N. It is further inspected the performance of NN accuracy for the proposed CT methods in Algorithm 2. The bar plot of Figure 5.19 (c) shows the SCNN test accuracy performance for both correlation-uncontrolled direct bitstream and near-zero correlation-driven CT approaches (a_{mid}, a_{poly}, and a_{best}). In Bitstream processing, the *logic-1*s in a stream are permuted (*cupy.random.shuffle* [231]) without a correlation control. In the literature, studies using the same dataset (Kuzushiji-MNIST [230]) and a similar convolutional network structure (deterministic) report 93.13% [232] and 95.12% [230] test accuracy. Therefore, in general, the accuracy of each SCNN approach is in the acceptable range for $N \geq 128$ compared to the ideal deterministic networks. Overall, the *correlation coefficient optimization approach* (a_{best}) yields the top accuracy performance for each N, similar to the runtime results.

6. CONCLUSIONS

This dissertation first provided introductory information on the bitstream processing paradigm, including our purpose, contributions, and hypothesis. Second, it conveyed the background knowledge on stochastic bitstream systems. Starting from the bitstream concept, encoding schemes, cross-correlation, bitstream randomization, SC hardware primitives for arithmetic, and non-idealities such as soft errors were discussed. While presenting the background, the design optimization issues for scaled adder arithmetic using cascaded MUXs were unveiled. Background also comprehensively surveyed SC-based vision and learning machine systems by listing previous works chronologically in the last decade. Hence, it was possible to shed light on a question: "What remains untouched in the literature regarding bitstream processing as a re-emerging computing paradigm?"

After observing the literature, the CT-based simulation approach was proposed, which makes bitstream processing efficient and fast in the software environment. Software simulation, as a flexible and virtual platform of a designed system, is essential in determining irreversible design errors before real hardware implementation. However, processing actual bitstreams and performing logic operations consume time and memory in the software environment. It was proposed to use only scalar values of operands, cross-correlation, and bitstream size to tackle this problem by constructing CT. Thus, only the table scalars were utilized to determine the logical operation. In other words, the logical results could be generated with the arithmetic of scalar values without using bitstreams and bitwise operations in the software.

The CT-based method was targeted for simulating positively and negatively correlated bitstreams, as well as for simulating uncorrelated bitstreams, which is a more challenging problem. The three different approaches for uncorrelated bitstream emulation were presented based on the *SCC* metric. The a_{best} approach, which was obtained with an optimization considering the *SCC* formula, was the best-performing method in terms of error. Besides *SCC*, different binary correlation measurement

metrics were tested; however, the best metric was found as *SCC* for error monitoring. For CT to simulate all combinations of circuit inputs, the CT occurrence formula, O_{CT}, was built. O_{CT} can calculate different combinations that make up the same unique CT. O_{CT} supplied fast simulation for considering all input combinations (different from Monte Carlo simulation), which was proven while obtaining *error-population* graphs. Finally, this study discussed incorporating random fluctuation error, an important concept in the SC paradigm, into the CT simulation. In this respect, it was shown how random errors could be emulated with the AND reference logic gate utilizing the MSE metric. Thereby, the built-in error handling was conducted when generating bitstreams via Sobol LD and LFSR, as well as the binomial distribution. Thus, this research proved that random errors due to the bitstream generators could also be simulated in addition to circuit operations and specific correlation.

After the contributions to the simulation environment in the third chapter, it was time to focus on the two most important applied areas of SC: image processing and machine learning. With the vision machines under the *Hardware-Inspired Image Processing* title in Chapter 4, the design suggestions were presented for mean filtering, edge detection, template matching, bilinear interpolation, and image compositing algorithms. The first was a reinterpretation and an example to present the combination of image processing and SC concepts. The Prewitt edge detector has been the subject of a few previous studies among Sobel and Roberts cross edge detector approaches in SC literature. This section showed how different approximations are compared using PSNR and SSIM metrics, considering the MUX scaled adder. Template matching was discussed considering three conventional approaches: SAD, SSD, and cross-correlation-based calculation. The template matching operation performed with a single logic gate was tested on the sample dataset of QR codes. The performance of SC in the presence of Gaussian and speckle noise was emphasized. Then, bilinear interpolation and image compositing algorithms were crucial for application-level validation of CT simulation. SC adaptation of both algorithms was proposed for the first time in this study. The SC-based bilinear interpolation approach that scales images showed how a 4-to-1 MUX simply performs the operation when fed with bitstreams. SC-based image compositing was also successfully performed with a straightforward hardware module using only a 2-to-1 MUX that combines two different input images.

Foreground and background images were combined, and the composited image was compared with the reference image using the PSNR metric. As a new perspective for CT, both methods proved the success of the CT-based emulation of the binomial distribution, LFSR, and Sobol LD-based bitstreams. The results were as if physical hardware modules were generating and processing actual bitstreams; however, the simulation processed only scalars and completed operations in a much shorter runtime with substantially less memory usage.

Finally, this dissertation focused on the new opportunities of bitstream processing in machine learning. The conventional (deterministic) and stochastic versions of each design were discussed, from a single neuron to the multi-layered and binarized network (BNN) architectures hierarchically. The superiority of the model with stochastic bitstreams over the conventional one was underscored when a single neuron works as a linear classifier. It was experimentally demonstrated that the stochastic neuron outperforms the conventional one in the early epochs on noisy bi-class data classification. Due to the complexity of mathematics presented in the literature for conventional multi-layered architectures, this thesis first demonstrated the mathematical background for training MLPs. Then, SC adaptation was proposed by switching to lightweight binarized networks (BSBNN). Unlike the literature, it was emphasized the importance of keeping the inputs and network weights in short dimensional bitstreams for hardware efficiency and soft error robustness. Thanks to the bitstream-based data representation and BPE scheme, 30% less hardware usage than the prior art was achieved by using a neuron architecture with only multipliers and an accumulator, without any additional activation modules. By keeping the weights and data in short bitstreams, superior error robustness was captured against bit-flip errors over deterministic BNNs. Meanwhile, a framework was presented for the general simulation of SC-based NNs (SCNN). Thus, all simulation approaches up to that point allowed us to compare four different network structures: BNN, BSBNN, SCNN, and FPNN. In addition to the bit-flip error, the comparative performance of different NN architectures was revealed for input data corruptions and corruption-aware training via the MNIST-C dataset. Ultimately, BSBNN architecture was validated with its remarkably comparable performance. In the final section, the CT was visited under the NN concept, and its runtime and accuracy performance was compared to the bitstream

processing architecture. After all, CT did not cause an exponential runtime as actual bitstreams do; on the contrary, it was experimentally proven that CT was a simulation method independent of the bitstream length, N.

In conclusion, the recommendations for future studies and other researchers are stated. To the best of our knowledge, SC has not been brought together yet with ① hyperdimensional computing, ② in-memory computing, and ③ quantum computing at the level of a doctoral thesis. As far as future avenues of research are concerned, it is recommended to blend SC with these areas of study for potentially fruitful results. For the first, data representation in high-dimensional vectors (hypervectors) broadly matches with SC bitstreams. Second, beyond von Neumann architecture, in-memory computing is a trending topic, and prior works show that bitstream processing can be a beneficial tool. Finally, combining the third with a built-in stochastic environment can fit stochastic bitstream processing with a relation of qubit and bitstream concepts. Therefore, conducting inter-computing research by combining two different computing approaches is highly recommended, which would benefit future studies remarkably.

REFERENCES

[1] **Gaines, B.R.**, (1969). *Stochastic Computing Systems*, Springer US, Boston, MA, pp.37–172, https://doi.org/10.1007/978-1-4899-5841-9_2.

[2] **von Neumann, J.** (1952). Probabilistic logics and the synthesis of reliable organisms from unreliable components, *Lectures, The Institute for Advanced Study Princeton*.

[3] **Najafi, M.H.**, **Li, P.**, **Lilja, D.J.**, **Qian, W.**, **Bazargan, K. and Riedel, M.** (2017). A reconfigurable architecture with sequential logic-based stochastic computing, *J. Emerg. Technol. Comput. Syst.*, *13*(4), https://doi.org/10.1145/3060537.

[4] **Sato, S.**, **Nemoto, K.**, **Akimoto, S.**, **Kinjo, M. and Nakajima, K.** (2003). Implementation of a new neurochip using stochastic logic, *IEEE Transactions on Neural Networks*, *14*(5), 1122–1127.

[5] **Seva, R.**, **Metku, P. and Choi, M.** (2017). Energy-efficient FPGA-based parallel quasi-stochastic computing, *Journal of Low Power Electronics and Applications*, *7*(4), 29, http://dx.doi.org/10.3390/jlpea7040029.

[6] **Li, P. and Lilja, D.J.** (2011). A low power fault-tolerance architecture for the kernel density estimation based image segmentation algorithm, *ASAP 2011 - 22nd IEEE International Conference on Application-specific Systems, Architectures and Processors*, Santa Monica, CA, USA, pp.161–168.

[7] **Li, H.**, **Zhang, D. and Foo, S.** (2006). A stochastic digital implementation of a neural network controller for small wind turbine systems, *IEEE Transactions on Power Electronics*, *21*(5), 1502–1507.

[8] **Onizawa, N.**, **Katagiri, D.**, **Matsumiya, K.**, **Gross, W.J. and Hanyu, T.** (2015). Gabor filter based on stochastic computation, *IEEE Signal Processing Letters*, *22*(9), 1224–1228.

[9] **Fick, D.**, **Kim, G.**, **Wang, A.**, **Blaauw, D. and Sylvester, D.** (2014). Mixed-signal stochastic computation demonstrated in an image sensor with integrated 2D edge detection and noise filtering, *Proceedings of the IEEE 2014 Custom Integrated Circuits Conference*, San Jose, CA, USA, pp.1–4.

[10] **Alaghi, A.**, **Li, C. and Hayes, J.P.** (2013). Stochastic circuits for real-time image-processing applications, *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX, USA, pp.1–6.

[11] **Li, P. and Lilja, D.J.** (2011). Using stochastic computing to implement digital image processing algorithms, *2011 IEEE 29th International Conference on Computer Design (ICCD)*, Amherst, MA, USA, pp.154–161.

[12] **Moons, B. and Verhelst, M.** (2014). Energy-efficiency and accuracy of stochastic computing circuits in emerging technologies, *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, *4*(4), 475–486.

[13] **Moons, B. and Verhelst, M.** (2014). Energy and accuracy in multi-stage stochastic computing, *2014 IEEE 12th International New Circuits and Systems Conference (NEWCAS)*, Trois-Rivieres, QC, Canada, pp.197–200.

[14] **Canals, V.**, **Frasser, C.**, **Alomar, M.**, **Morro, A.**, **Oliver, A.**, **Roca, M.**, **Isern, E.**, **Martínez-Moll, V.**, **Garcia-Moreno, E. and Rosselló, J.** (2017). Noise tolerant probabilistic logic for statistical pattern recognition applications, *Integr. Comput. Aided. Eng.*, *24*(4), 351–365.

[15] **Qian, W.**, **Li, X.**, **Riedel, M.D.**, **Bazargan, K. and Lilja, D.J.** (2011). An architecture for fault-tolerant computation with stochastic logic, *IEEE Transactions on Computers*, *60*(1), 93–105.

[16] **Cilasun, M.H. and Altun, M.** (2017). A novel reversible fault tolerant microprocessor design in AMS 0.35um process, *Istanbul University, IU-JEEE ELECTRICA*, *17*(1), 3147–3154.

[17] **Nicolaidis, M.**, (2011). *Soft Errors in Modern Electronic Systems*, Springer, New York, USA, https://doi.org/10.1007/978-1-4419-6993-4.

[18] **Chen, T.H.**, **Alaghi, A. and Hayes, J.P.** (2014). Behavior of stochastic circuits under severe error conditions, *it - Information Technology*, *56*(4), 182–191, https://doi.org/10.1515/itit-2013-1042.

[19] **Li, B.**, **Najafi, M.H. and Lilja, D.J.** (2019). Low-cost stochastic hybrid multiplier for quantized neural networks, *J. Emerg. Technol. Comput. Syst.*, *15*(2), https://doi.org/10.1145/3309882.

[20] **Canals, V.**, **Morro, A.**, **Oliver, A.**, **Alomar, M.L. and Rosselló, J.L.** (2016). A new stochastic computing methodology for efficient neural network implementation, *IEEE Transactions on Neural Networks and Learning Systems*, *27*(3), 551–564.

[21] **Chippa, V.K.**, **Venkataramani, S.**, **Roy, K. and Raghunathan, A.** (2014). StoRM: A stochastic recognition and mining processor, *2014 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, La Jolla, California, USA, pp.39–44, http://doi.org/10.1145/2627369.2627645.

[22] **Zhang, Y.**, **Wang, R.**, **Zhang, X.**, **Wang, Y. and Huang, R.** (2020). Parallel hybrid stochastic-binary-based neural network accelerators, *IEEE Transactions on Circuits and Systems II: Express Briefs*, *67*(12), 3387–3391.

[23] **Sim, H.**, **Nguyen, D.**, **Lee, J. and Choi, K.** (2017). Scalable stochastic-computing accelerator for convolutional neural networks, *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Chiba, Japan, pp.696–701.

[24] **Xie, Y.**, **Liao, S.**, **Yuan, B.**, **Wang, Y. and Wang, Z.** (2017). Fully-parallel area-efficient deep neural network design using stochastic computing, *IEEE Transactions on Circuits and Systems II: Express Briefs*, *64*(12), 1382–1386.

[25] **Alaghi, A. and Gaffney, E.**, (2019), scsynth, `https://github.com/arminalaghi/scsynth`, (Date Retrieved: 01.09.2022).

[26] **Alaghi, A. and Hayes, J.P.** (2015). STRAUSS: Spectral Transform Use in Stochastic Circuit Synthesis, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *34*(11), 1770–1783.

[27] **Daruwalla, K.**, **Zhuo, H. and Lipasti, M.** (2019). BitSAD: A domain-specific language for bitstream computing, *Proceedings of the First ISCA Workshop on Unary Computing (WUC'19)*, Phoenix, Arizona, USA.

[28] **Daruwalla, K.**, **Zhuo, H.**, **Shukla, R. and Lipasti, M.** (2019). BitSAD v2: Compiler optimization and analysis for bitstream computing, *ACM Trans. Archit. Code Optim.*, *16*(4), `https://doi.org/10.1145/3364999`.

[29] **Daruwalla, K.**, (2021), BitSAD, `https://github.com/UW-PHARM/BitSAD`, (Date Retrieved: 01.09.2022).

[30] **Li, P.**, **Lilja, D.J.**, **Qian, W.**, **Riedel, M.D. and Bazargan, K.** (2014). Logical computation on stochastic bit streams with linear finite-state machines, *IEEE Transactions on Computers*, *63*(6), 1474–1486.

[31] **Chen, T.H.** (2016). Designing accurate and low-cost stochastic circuits, *Ph.D. Dissertation*, University of Michigan, Ann Arbor, USA, `http://hdl.handle.net/2027.42/133255`.

[32] **Parhi, K.K.** (2017). Analysis of stochastic logic circuits in unipolar, bipolar and hybrid formats, *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, Baltimore, MD, USA, pp.1–4.

[33] **Alaghi, A.** (2015). The logic of random pulses: Stochastic computing, *Ph.D. Dissertation*, University of Michigan, Ann Arbor, USA, `https://homes.cs.washington.edu/~armin/PhD_Thesis.pdf`.

[34] **Winstead, C.**, (2019). Tutorial on Stochastic Computing, Springer International Publishing, Cham, pp.39–76.

[35] **Wang, Y.**, **Zhan, Z.**, **Zhao, L.**, **Tang, J.**, **Wang, S.**, **Li, J.**, **Yuan, B.**, **Wen, W. and Lin, X.** (2019). Universal approximation property and equivalence of stochastic computing-based neural networks and binary neural networks, *Proceedings of the AAAI Conference on Artificial Intelligence*, pp.5369–5376.

[36] **Faraji, S.R.**, **Hassan Najafi, M.**, **Li, B.**, **Lilja, D.J. and Bazargan, K.** (2019). Energy-efficient convolutional neural networks with deterministic bit-stream processing, *2019 Design, Automation and Test in Europe Conference Exhibition (DATE)*, Florence, Italy, pp.1757–1762.

[37] **Schober, P.**, **Estiri, S.N.**, **Aygun, S.**, **TaheriNejad, N. and Najafi, M.H.** (2022). Sound source localization using stochastic computing, *ICCAD 2022, The 41st IEEE/ACM International Conference on Computer-Aided Design*, San Diego, California, USA.

[38] **Alaghi, A. and Hayes, J.P.** (2013). Exploiting correlation in stochastic circuit design, *2013 IEEE 31st International Conference on Computer Design (ICCD)*, Asheville, NC, USA, pp.39–46.

[39] **Ishikawa, R.**, **Tawada, M.**, **Yanagisawa, M. and Togawa, N.** (2018). Stochastic number duplicators based on bit re-arrangement using randomized bit streams, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, *E101A*(7), 1002–1013.

[40] **Lee, V.T.**, **Alaghi, A. and Ceze, L.** (2018). Correlation manipulating circuits for stochastic computing, *2018 Design, Automation and Test in Europe Conference Exhibition (DATE)*, Dresden, Germany, pp.1417–1422.

[41] **Budhwani, R.K.**, **Ragavan, R. and Sentieys, O.** (2017). Taking advantage of correlation in stochastic computing, *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, Baltimore, MD, USA, pp.1–4.

[42] **Alaghi, A.**, **Ting, P.**, **Lee, V. and Hayes, J.**, (2019). Accuracy and correlation in stochastic computing, Springer International Publishing, Cham, pp.77–102.

[43] **Asadi, S.**, **Najafi, M.H. and Imani, M.** (2021). CORLD: In-stream correlation manipulation for low-discrepancy stochastic computing, *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, Munich, Germany, pp.1–9.

[44] **Liu, S. and Han, J.** (2017). Energy efficient stochastic computing with Sobol sequences, *Design, Automation and Test in Europe Conference Exhibition (DATE), 2017*, Lausanne, Switzerland, pp.650–653.

[45] **Liu, S. and Han, J.** (2018). Toward energy-efficient stochastic circuits using parallel Sobol sequences, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *26*(7), 1326–1339.

[46] **Lin, Z.**, **Xie, G.**, **Xu, W.**, **Han, J. and Zhang, Y.** (2021). Accelerating stochastic computing using deterministic Halton sequences, *IEEE Transactions on Circuits and Systems II: Express Briefs*, *68*(10), 3351–3355.

[47] **Wu, D. and Miguel, J.S.** (2019). In-stream stochastic division and square root via correlation, *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, Las Vegas, NV, USA, `https://doi.org/10.1145/3316781.3317844`.

[48] **Zhao, L.**, (1998), Random pulse artificial neural network architecture, `http://hdl.handle.net/10393/8489`, (Master Thesis, Ottawa-Carleton Institute for Electrical and Computer Engineering).

[49] **Erten, G. and Goodman, R.** (1992). A digital neural network architecture using random pulse trains, *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 1, Baltimore, MD, USA, pp.190–195 vol.1.

[50] **Ribeiro, S.T.** (1967). Random-pulse machines, *IEEE Transactions on Electronic Computers*, *EC-16*(3), 261–276.

[51] **Anderson, J.H.**, **Hara-Azumi, Y. and Yamashita, S.** (2016). Effect of LFSR seeding, scrambling and feedback polynomial on stochastic computing accuracy, *2016 Design, Automation and Test in Europe Conference Exhibition (DATE)*, Dresden, Germany, pp.1550–1555.

[52] **Parol, M.**, **Dabal, P. and Szplet, R.** (2016). Pseudo-random bit generators based on linear-feedback shift registers in a programmable device, *Measurement Automation Monitoring*, *62*(6), 184–186.

[53] **Seva, R.**, (2017), Novel approaches for efficient stochastic computing, `https://scholarsmine.mst.edu/cgi/viewcontent.cgi?article=8659&context=masters_theses`, (Master Thesis, Missouri University of Science and Technology).

[54] **Karadeniz, M.B. and Altun, M.** (2017). Sampling based random number generator for stochastic computing, *2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Batumi, Georgia, pp.227–230.

[55] **Salehi, S.A.** (2020). Low-cost stochastic number generators for stochastic computing, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *28*(4), 992–1001.

[56] **Najafi, M.H.**, **Lilja, D.J. and Riedel, M.** (2018). Deterministic methods for stochastic computing using low-discrepancy sequences, *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, San Diego, CA, USA, pp.1–8.

[57] **Parhi, M.**, **Riedel, M.D. and Parhi, K.K.** (2015). Effect of bit-level correlation in stochastic computing, *2015 IEEE International Conference on Digital Signal Processing (DSP)*, Singapore, pp.463–467.

[58] **Ting, P. and Hayes, J.P.** (2019). Exploiting randomness in stochastic computing, *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, Westminster, CO, USA, pp.1–6.

[59] **Kuehnel, R.** (2002). Binomial logic: extending stochastic computing to high-bandwidth signals, *Conference Record of the Thirty-Sixth Asilomar Conference on Signals, Systems and Computers, 2002.*, volume 2, Pacific Grove, CA, USA, pp.1089–1093.

[60] **Li, B.**, **Hu, J.**, **Najafi, M.H.**, **Koester, S. and Lilja, D.J.** (2019). Low cost hybrid spin-CMOS compressor for stochastic neural networks, *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, GLSVLSI'19, Tysons Corner, VA, USA, pp.141–146, https://doi.org/10.1145/3299874.3317989.

[61] **Frasser, C.F.**, **de Benito, C.**, **Skibinsky-Gitlin, E.S.**, **Canals, V.**, **Font-Rosselló, J.**, **Roca, M.**, **Ballester, P.J. and Rosselló, J.L.** (2021). Using stochastic computing for virtual screening acceleration, *Electronics*, *10*(23), https://www.mdpi.com/2079-9292/10/23/2981.

[62] **Parker, K. and McCluskey, E.** (1975). Analysis of logic circuits with faults using input signal probabilities, *IEEE Transactions on Computers*, *C-24*(5), 573–578.

[63] **Ananth, R.**, (2003), Programmable supervisory circuit and applications thereof, United States Patent, No: 6,618,711.

[64] **Chen, T.H. and Hayes, J.P.** (2016). Design of division circuits for stochastic computing, *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Pittsburgh, PA, USA, pp.116–121.

[65] **Xianjun, J.** (2012). An improved stochastic decoding algorithm of LTE turbo codes, *Wireless Algorithms, Systems, and Applications*, Springer, Berlin, Heidelberg, pp.301–308.

[66] **Tehrani, S.S.**, **Mannor, S. and Gross, W.J.** (2007). Survey of stochastic computation on factor graphs, *37th International Symposium on Multiple-Valued Logic (ISMVL'07)*, Oslo, Norway.

[67] **Faix, M.**, **Mazer, E.**, **Laurent, R.**, **Abdallah, M.O.**, **Le Hy, R. and Lobo, J.** (2017). Cognitive computation: An exact Bayesian inference stochastic machine, *International Journal of Software Science and Computational Intelligence (IJSSCI)*, *9*(3), 37–58.

[68] **Chu, S.I.** (2020). New divider design for stochastic computing, *IEEE Transactions on Circuits and Systems II: Express Briefs*, *67*(1), 147–151.

[69] **Zhou, Y.**, **Xie, G.**, **Han, J. and Zhang, Y.** (2021). Absolute subtraction and division circuits using uncorrelated random bitstreams in stochastic computing, *2021 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, AB, Canada, pp.1–6.

[70] **Wang, S.**, **Xie, G.**, **Han, J. and Zhang, Y.** (2022). Highly accurate division and square root circuits by exploiting signal correlation in stochastic computing, *International Journal of Circuit Theory and Applications*, *50*(4), 1375–1385.

[71] **Kim, K.**, **Kim, J.**, **Yu, J.**, **Seo, J.**, **Lee, J. and Choi, K.** (2016). Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks, *2016 53nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX, USA, pp.1–6.

[72] **Liu, Y.**, **Liu, S.**, **Wang, Y.**, **Lombardi, F. and Han, J.** (2018). A stochastic computational multi-layer perceptron with backward propagation, *IEEE Transactions on Computers*, *67*(9), 1273–1286.

[73] **Li, J.**, **Ren, A.**, **Li, Z.**, **Ding, C.**, **Yuan, B.**, **Qiu, Q. and Wang, Y.** (2017). Towards acceleration of deep convolutional neural networks using stochastic computing, *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Chiba, Japan, pp.115–120.

[74] **Aygun, S. and Gunes, E.O.** (2021). On the limit of multiplexers in stochastic computing, *International Journal of Multidisciplinary Studies and Innovative Technologies*, *5*(1), 94 – 97.

[75] **Najafi, M.H.**, **Lilja, D.J.**, **Riedel, M. and Bazargan, K.** (2016). Polysynchronous stochastic circuits, *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, Macao, China, pp.492–498.

[76] **Liu, Y.**, **Liu, L.**, **Lombardi, F. and Han, J.** (2019). An energy-efficient and noise-tolerant recurrent neural network using stochastic computing, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *27*(09), 2213–2221.

[77] **Aygun, S.**, **Gunes, E.O. and De Vleeschouwer, C.** (2021). Efficient and robust bitstream processing in binarised neural networks, *Electronics Letters*, *57*(5), 219–222, `https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/ell2.12045`.

[78] **Babu, A.**, **Lashkare, S.**, **Ganguly, U. and Rajendran, B.** (2018). Stochastic learning in deep neural networks based on nanoscale PCMO device characteristics, *Neurocomputing*, *321*, 227–236, `https://www.sciencedirect.com/science/article/pii/S0925231218310804`.

[79] **Duarte, R.**, **Véstias, M.**, **Carvalho, C. and Casaleiro, J.** (2018). Stochastic theater: Stochastic datapath generation framework for fault-tolerant IoT sensors, *i-ETC : ISEL Academic Journal of Electronics Telecommunications and Computers*, *4*(1).

[80] **Haselmayr, W.**, **Wiesinger, D. and Lunglmayr, M.** (2020). High-accuracy and fault tolerant stochastic inner product design, *IEEE Transactions on Circuits and Systems II: Express Briefs*, *67*(3), 541–545.

[81] **El-Mhamdi, E.M.**, **Guerraoui, R.**, **Kucharavy, A. and Volodin, S.**, (2019), The probabilistic fault tolerance of neural networks in the continuous limit, `https://openreview.net/forum?id=rkl_f6EFPS`.

[82] **Ho, K.I.J.**, **Leung, C.S. and Sum, J.** (2010). Convergence and objective functions of some fault/noise-injection-based online learning algorithms for RBF networks, *IEEE Transactions on Neural Networks*, *21*(6), 938–947.

[83] **Pajarinen, J.**, **Peltonen, J. and Uusitalo, M.** (2011). Fault tolerant machine learning for nanoscale cognitive radio, *Neurocomputing*, *74*(5), 753–764, https://www.sciencedirect.com/science/article/pii/S0925231210004108.

[84] **Hacene, G.B.**, **Leduc-Primeau, F.**, **Soussia, A.B.**, **Gripon, V. and Gagnon, F.** (2019). Training modern deep neural networks for memory-fault robustness, *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*, Sapporo, Japan, pp.1–5.

[85] **Kim, S.**, **Howe, P.**, **Moreau, T.**, **Alaghi, A.**, **Ceze, L. and Sathe, V.S.** (2018). Energy-efficient neural network acceleration in the presence of bit-level memory errors, *IEEE Transactions on Circuits and Systems I: Regular Papers*, *65*(12), 4285–4298.

[86] **Lecun, Y.**, **Bottou, L.**, **Bengio, Y. and Haffner, P.** (1998). Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, *86*(11), 2278–2324.

[87] **Mu, N. and Gilmer, J.**, (2019), MNIST-C: A robustness benchmark for computer vision, https://arxiv.org/abs/1906.02337, arXiv.

[88] **Alaghi, A. and Hayes, J.P.** (2013). Survey of stochastic computing, *ACM Trans. Embed. Comput. Syst.*, *12*(2s), https://doi.org/10.1145/2465787.2465794.

[89] **Dehghan, B. and Parhizgar, N.** (2014). Survey the stochastic computing and probabilistic transfer matrix (PTM) in logic circuits, *International Journal of Advanced Research in Computer and Communication Engineering*, *3*(12), 8674–8678.

[90] **Alawad, M. and Lin, M.** (2019). Survey of stochastic-based computation paradigms, *IEEE Transactions on Emerging Topics in Computing*, *7*(1), 98–114.

[91] **Liu, Y.**, **Liu, S.**, **Wang, Y.**, **Lombardi, F. and Han, J.** (2021). A survey of stochastic computing neural networks for machine learning applications, *IEEE Transactions on Neural Networks and Learning Systems*, *32*(7), 2809–2824.

[92] **Lee, Y.Y. and Halim, Z.A.** (2020). Stochastic computing in convolutional neural network implementation: A review, *PeerJ Computer Science*, *6:e309*, 1–35.

[93] **Xiao, R. and Chen, C.** (2014). Gate-level circuit reliability analysis: A survey, *VLSI Design*, *2014*, 1–12.

[94] **Qian, W.** (2011). Digital yet deliberately random: Synthesizing logical computation on stochastic bit streams, *Ph.D. Dissertation*, University of Minnesota, Minneapolis, USA, http://cctbio.ece.umn.edu/wiki/images/3/33/Qian_Synthesizing_Logical_Computation_on_Stochastic_Bit_Streams.pdf.

[95] **Li, P.** (2013). Analysis, design, and logic synthesis of finite-state machine-based stochastic computing, *Ph.D. Dissertation*, University of Minnesota, Minneapolis, USA, https://hdl.handle.net/11299/155955.

[96] **Bai, Y.** (2016). Stochastic-based computing with emerging spin-based device technologies, *Ph.D. Dissertation*, University of Central Florida, Orlando, USA, https://stars.library.ucf.edu/cgi/viewcontent.cgi?article=6424&context=etd.

[97] **Liu, Y.** (2017). Digital signal processing and machine learning system design using stochastic logic, *Ph.D. Dissertation*, University of Minnesota, Minneapolis, USA, https://hdl.handle.net/11299/190534.

[98] **Barceló, M.L.A.** (2017). Methodologies for hardware implementation of reservoir computing systems, *Ph.D. Dissertation*, Universitat de les Illes Balears, Palma (Illes Balears), Spain, http://hdl.handle.net/10803/565422.

[99] **Li, B.** (2018). Distributed edge computing infrastructure with low hardware cost, performance evaluation and reliability, *Ph.D. Dissertation*, University of Minnesota, Minneapolis, USA, https://hdl.handle.net/11299/201151.

[100] **Najafi, M.H.** (2018). New views for stochastic computing: From time-encoding to deterministic processing, *Ph.D. Dissertation*, University of Minnesota, Minneapolis, USA, http://hdl.handle.net/11299/200291.

[101] **Liu, Y.** (2019). Design and evaluation of stochastic computing neural networks for machine learning applications, *Ph.D. Dissertation*, University of Alberta, Alberta, Canada, https://doi.org/10.7939/r3-sg1h-mf48.

[102] **Ardakani, A.** (2019). Complexity reduction of deep neural networks for efficient hardware implementations, *Ph.D. Dissertation*, McGill University, Montreal, Canada, https://escholarship.mcgill.ca/concern/theses/d791sm62n.

[103] **Lee, V.T.** (2019). Towards practical stochastic computing architectures for emerging applications, *Ph.D. Dissertation*, University of Washington, Seattle, USA, http://hdl.handle.net/1773/43658.

[104] **Ting, P.** (2019). Design of sequential stochastic computing systems, *Ph.D. Dissertation*, University of Michigan, Ann Arbor, USA, https://deepblue.lib.umich.edu/bitstream/handle/2027.42/151530/paishun_1.pdf?sequence=1.

[105] **Liu, S.** (2019). Toward energy-efficient, dynamic stochastic computing, *Ph.D. Dissertation*, University of Alberta, Alberta, Canada, https://doi.org/10.7939/r3-p2vw-yy63.

[106] **Prado, F.G.** (2020). Development of neuromorphic hardware and non-iterative learning designs for edge computing applications, *Ph.D. Dissertation*, Universitat de les Illes Balears, Palma (Illes Balears), Spain, http://hdl.handle.net/11201/155448.

[107] **El-Derhalli, H.** (2021). Design of stochastic computing architectures using integrated optics, *Ph.D. Dissertation*, Concordia University, Montreal, Canada, http://hvg.ece.concordia.ca/Publications/Thesis/Hassnaa-PhD-Thesis.pdf.

[108] **Najafi, A.** (2021). Systematic design of low-power processing elements using stochastic and approximate computing techniques, *Ph.D. Dissertation*, University of Bremen, Bremen, Germany, https://media.suub.uni-bremen.de/handle/elib/4663.

[109] **Brown, B. and Card, H.** (2001). Stochastic neural computation. I. Computational elements, *IEEE Transactions on Computers*, *50*(9), 891–905.

[110] **Brown, B. and Card, H.** (2001). Stochastic neural computation. II. Soft competitive learning, *IEEE Transactions on Computers*, *50*(9), 906–920.

[111] **Najafi, M.H. and Lilja, D.J.** (2017). High-speed stochastic circuits using synchronous analog pulses, *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Chiba, Japan, pp.481–487.

[112] **Ranjbar, M.**, **Salehi, M.E. and Najafi, M.H.** (2015). Using stochastic architectures for edge detection algorithms, *2015 23rd Iranian Conference on Electrical Engineering*, Tehran, Iran, pp.723–728.

[113] **Wang, R.**, **Han, J.**, **Cockburn, B.F. and Elliott, D.** (2016). Stochastic circuit design and performance evaluation of vector quantization for different error measures, *IEEE Transactions on VLSI Sys.*, *24*(10), 3169–3183.

[114] **Aygun, S.**, **Altun, M. and Gunes, E.O.** (2017). Sobel filter operation in image processing via stochastic arithmetic-logic unit design, *2017 25th Signal Processing and Communications Applications Conference (SIU)*, Antalya, Turkey, pp.1–4.

[115] **Li, P.**, **Lilja, D.J.**, **Qian, W.**, **Bazargan, K. and Riedel, M.D.** (2014). Computation on stochastic bit streams digital image processing case studies, *IEEE Transactions on VLSI Sys.*, *22*(3), 449–462.

[116] **Onizawa, N.**, **Katagiri, D.**, **Matsumiya, K.**, **Gross, W.J. and Hanyu, T.** (2018). An accuracy/energy-flexible configurable Gabor-filter chip based on stochastic computation with dynamic voltage–frequency–length scaling, *IEEE J. Emerg. Sel. Top. Circuits Syst.*, *8*(3), 444–453.

[117] **Boga, K.**, **Leduc-Primeau, F.**, **Onizawa, N.**, **Matsumiya, K.**, **Hanyu, T. and Gross, W.J.** (2018). A generalized stochastic implementation of the disparity energy model for depth perception, *Journal of Signal Processing Systems*, *90*(5), 709–725, https://doi.org/10.1007/s11265-016-1197-3.

[118] **Abdellatef, H.**, **Khalil-hani, M. and Shaikh-husin, N.** (2019). Accurate and compact stochastic computations by exploiting correlation, *Turkish Journal of Electrical Engineering and Computer Science*, *27*, 547 – 564.

[119] **Najafi, M.H. and Salehi, M.E.** (2016). A fast fault-tolerant architecture for Sauvola local image thresholding algorithm using stochastic computing, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *24*(2), 808–812.

[120] **Alaghi, A. and Hayes, J.P.** (2015). On the functions realized by stochastic computing circuits, *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, GLSVLSI'15, Pittsburgh, Pennsylvania, USA, p.331–336, https://doi.org/10.1145/2742060.2743758.

[121] **Seva, R.**, **Metku, P.**, **Kim, K.K.**, **Kim, Y.B. and Choi, M.** (2017). Variable bit truncation technique for approximate stochastic computing (ASC), *2017 International SoC Design Conference (ISOCC)*, pp.73–74, http://doi.org/10.1109/ISOCC.2017.8368832.

[122] **Zhang, Z.**, **Wang, R.**, **Zhang, Z.**, **Zhang, Y.**, **Guo, S. and Huang, R.** (2020). Circuit reliability comparison between stochastic computing and binary computing, *IEEE Transactions on Circuits and Systems II: Express Briefs*, *67*(12), 3342–3346.

[123] **Miao, L. and Chakrabarti, C.** (2013). A parallel stochastic computing system with improved accuracy, *SiPS 2013 Proceedings*, Taipei, Taiwan, pp.195–200.

[124] **Parhi, K.K. and Liu, Y.** (2014). Architectures for IIR digital filters using stochastic computing, *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*, Melbourne, VIC, Australia, pp.373–376.

[125] **Ichihara, H.**, **Sugino, T.**, **Ishii, S.**, **Iwagaki, T. and Inoue, T.** (2019). Compact and accurate digital filters based on stochastic computing, *IEEE Transactions on Emerging Topics in Computing*, *7*(1), 31–43.

[126] **Gaines, B.** (2019). Origins of stochastic computing, *Stochastic Computing: Techniques and Applications*, Springer International Publishing, pp.13–37.

[127] **Morro, A.**, **Canals, V.**, **Oliver, A.**, **Alomar, M.L. and Rossello, J.L.** (2015). Ultra-fast data-mining hardware architecture based on stochastic computing, *PLOS ONE*, *10*(5), 1–20, https://doi.org/10.1371/journal.pone.0124176.

[128] **Li, B.**, **Najafi, M.H. and Lilja, D.J.** (2015). An FPGA implementation of a restricted Boltzmann machine classifier using stochastic bit streams, *2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, Toronto, ON, Canada, pp.68–69.

[129] **Li, B.**, **Najafi, M.H. and Lilja, D.J.** (2016). Using stochastic computing to reduce the hardware requirements for a restricted Boltzmann machine classifier, *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA'16, p.36–41, https://doi.org/10.1145/2847263.2847340.

[130] **Liu, Y.**, **Venkataraman, H.**, **Zhang, Z. and Parhi, K.K.** (2016). Machine learning classifiers using stochastic logic, *2016 IEEE 34th International Conference on Computer Design (ICCD)*, Scottsdale, AZ, USA, pp.408–411.

[131] **Li, Z.**, **Li, J.**, **Ren, A.**, **Cai, R.**, **Ding, C.**, **Qian, X.**, **Draper, J.**, **Yuan, B.**, **Tang, J.**, **Qiu, Q. and Wang, Y.** (2019). HEIF: Highly efficient stochastic computing-based inference framework for deep neural networks, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *38*(8), 1543–1556.

[132] **Zhang, Y.**, **Zhang, X.**, **Song, J.**, **Wang, Y.**, **Huang, R. and Wang, R.** (2019). Parallel convolutional neural network (CNN) accelerators based on stochastic computing, *2019 IEEE International Workshop on Signal Processing Systems (SiPS)*, Nanjing, China, pp.19–24.

[133] **Smithson, S.C.**, **Boga, K.**, **Ardakani, A.**, **Meyer, B.H. and Gross, W.J.** (2016). Stochastic computing can improve upon digital spiking neural networks, *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*, Dallas, TX, USA, pp.309–314.

[134] **Li, Z.**, **Ren, A.**, **Li, J.**, **Qiu, Q.**, **Wang, Y. and Yuan, B.** (2016). DSCNN: Hardware-oriented optimization for stochastic computing based deep convolutional neural networks, *2016 IEEE 34th International Conference on Computer Design (ICCD)*, Scottsdale, AZ, USA, pp.678–681.

[135] **Rosselló, J.**, **Canals, V.**, **Oliver, A. and Morro, A.** (2014). Stochastic spiking neural networks at the EDGE of CHAOS, *2014 International Joint Conference on Neural Networks (IJCNN)*, Beijing, China, pp.2399–2406.

[136] **Rosselló, J.L.**, **Canals, V.**, **Morro, A. and Oliver, A.** (2012). Hardware implementation of stochastic spiking neural networks, *International Journal of Neural Systems*, *22*(04).

[137] **Canals, V.**, **Alomar, M.L.**, **Morro, A.**, **Oliver, A. and Rossello, J.L.** (2015). Noise-robust hardware implementation of neural networks, *2015 International Joint Conference on Neural Networks (IJCNN)*, Killarney, Ireland, pp.1–8.

[138] **Gross, W.**, **Onizawa, N.**, **Matsumiya, K. and Hanyu, T.** (2018). Application of stochastic computing in brainware, *Nonlinear Theory and Its Applications, IEICE*, *9*(4), 406–422.

[139] **Liu, Y.**, **Wang, Y.**, **Lombardi, F. and Han, J.** (2018). An energy-efficient stochastic computational deep belief network, *2018 Design, Automation and Test in Europe Conference Exhibition (DATE)*, Dresden, Germany, pp.1175–1178.

[140] **Sanni, K.**, **Garreau, G.**, **Molin, J.L. and Andreou, A.G.** (2015). FPGA implementation of a deep belief network architecture for character recognition using stochastic computation, *2015 49th Annual Conference on Information Sciences and Systems (CISS)*, Baltimore, MD, USA.

[141] **Sotiriadis, P.P. and Temenos, N.** (2022). Compact MAX and MIN stochastic computing architectures, *Integration*, *87*, 194–204, `https://www.sciencedirect.com/science/article/pii/S0167926022000736`.

[142] **Ghosh, S.K.**, **Raha, A. and Raghunathan, V.** (2020). Approximate inference systems (AxIS): End-to-end approximations for energy-efficient inference at the edge, *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED'20, Boston, Massachusetts, p.7–12, `https://doi.org/10.1145/3370748.3406575`.

[143] **Raha, A.**, **Ghosh, S.**, **Mohapatra, D.**, **Mathaikutty, D.A.**, **Sung, R.**, **Brick, C. and Raghunathan, V.** (2021). Special session: Approximate TinyML systems: Full system approximations for extreme energy-efficiency in intelligent edge devices, *2021 IEEE 39th International Conference on Computer Design (ICCD)*, Storrs, CT, USA, pp.13–16.

[144] **Sim, H. and Lee, J.** (2017). A new stochastic computing multiplier with application to deep convolutional neural networks, *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*, Austin, TX, USA, pp.1–6.

[145] **Sim, H. and Lee, J.** (2019). Cost-effective stochastic MAC circuits for deep neural networks, *Neural Networks*, *117*, 152–162, `https://www.sciencedirect.com/science/article/pii/S0893608019301236`.

[146] **Lee, J.**, **Seo, H.**, **Kim, Y. and Kim, Y.** (2020). Approximate adder design with simplified lower-part approximation, *IEICE Electronics Express*, *17*(15).

[147] **Onizawa, N.**, **Koshita, S.**, **Sakamoto, S.**, **Kawamata, M. and Hanyu, T.** (2018). An area/power-aware 32-channel compressive gammachirp filterbank chip based on hybrid stochastic/binary computation, *Nonlinear Theory and Its Applications, IEICE*, *9*(4), 423–435.

[148] **Kito, N.**, **Koketsu, Y. and Takagi, K.** (2018). Designs of component circuits for stochastic computing using rapid single flux quantum circuits, *The 21st Workshop on Synthesis and System Integration of Mixed Information Technologies, SASIMI*, Matsue, Japan.

[149] **Liu, Y. and Parhi, K.K.** (2016). Computing hyperbolic tangent and sigmoid functions using stochastic logic, *2016 50th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, CA, USA, pp.1580–1585.

[150] **Li, J.**, **Yuan, Z.**, **Li, Z.**, **Ding, C.**, **Ren, A.**, **Qiu, Q.**, **Draper, J. and Wang, Y.** (2017). Hardware-driven nonlinear activation for stochastic computing based deep convolutional neural networks, *2017 International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, USA, pp.1230–1236.

[151] **Ji, Y.**, **Ran, F.**, **Ma, C. and Lilja, D.J.** (2015). A hardware implementation of a radial basis function neural network using stochastic logic, *2015 Design, Automation and Test in Europe Conference Exhibition (DATE)*, Grenoble, France, pp.880–883.

[152] **Li, J.**, **Yuan, Z.**, **Li, Z.**, **Ren, A.**, **Ding, C.**, **Draper, J.**, **Nazarian, S.**, **Qiu, Q.**, **Yuan, B. and Wang, Y.** (2019). Normalization and dropout for stochastic computing-based deep convolutional neural networks, *Integration*, *65*, 395–403, `https://www.sciencedirect.com/science/article/pii/S0167926017302328`.

[153] **Yuan, Z.**, **Li, J.**, **Li, Z.**, **Ding, C.**, **Ren, A.**, **Yuan, B.**, **Qiu, Q.**, **Draper, J. and Wang, Y.** (2017). Softmax regression design for stochastic computing based deep convolutional neural networks, *Proceedings of the on Great Lakes Symposium on VLSI 2017*, GLSVLSI'17, Banff, Alberta, Canada, p.467–470, `https://doi.org/10.1145/3060403.3060467`.

[154] **Ardakani, A.**, **Leduc-Primeau, F.**, **Onizawa, N.**, **Hanyu, T. and Gross, W.J.** (2017). VLSI implementation of deep neural network using integral stochastic computing, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *25*(10), 2688–2699.

[155] **Choi, S.S.**, **Cha, S. and Tappert, C.** (2010). A survey of binary similarity and distance measures, *J. on Sys., Cyb. Inf.*, *8*, 43–48.

[156] **Lesot, M.**, **Rifqi, M. and Benhadda, H.** (2009). Similarity measures for binary and numerical data: A survey, *Int. J. Knowl. Eng. Soft Data Paradigm*, *1*(1).

[157] **Lande, T.**, **Constandinou, T.**, **Burdett, A. and Toumazou, C.** (2007). Running cross-correlation using bitstream processing, *Electronics Letters*, *43*, 1181–1183(2).

[158] **Hsiao, H.**, **Miguel, J.S.**, **Hara-Azumi, Y. and Anderson, J.** (2021). Zero correlation error: A metric for finite-length bitstream independence in stochastic computing, *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, Tokyo, Japan, pp.260–265.

[159] **Karl, P.** (1896). Mathematical contributions to the theory of evolution.–III. Regression, heredity and panmixia, *Philos. Trans. Royal Soc. London*, *187*, 253–318.

[160] **Jaccard, P.** (1908). Sur la distribution florale, *Bull. la Société Vaudoise des Sci. Nat.*, (163), 223–270.

[161] **Okyay, S. and Aygun, S.** (2021). Experimental interpretation of adequate weight-metric combination for dynamic user-based collaborative filtering, *PeerJ Computer Science*, 7:e784, https://doi.org/10.7717/peerj-cs.784.

[162] **Dice, L.R.** (1945). Measures of the amount of ecologic association between species, *Ecology*, *26*, 297–302.

[163] **Kulczynski, S.** (1927). Die pflanzenassoziationen der pieninen, *Bull. Int. l'Acad. Pol. des Sci. Lettr., Cl. des Sci. Math. Nat.*, 57–203.

[164] **Ochiai, A.** (1957). Zoogeographic studies on the soleoid fishes found in Japan and its neighbouring regions, *Bulletin of the Japanese Society for Science and Fisheries*, *22*, 526–530.

[165] **Najafi, M.H.**, **Lilja, D.J.**, **Riedel, M.D. and Bazargan, K.** (2018). Low-cost sorting network circuits using unary processing, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *26*(8), 1471–1480.

[166] **Najafi, M.H.**, **Jenson, D.**, **Lilja, D.J. and Riedel, M.D.** (2019). Performing stochastic computation deterministically, *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, *27*(12), 2925–2938.

[167] **Jalilvand, A.H.**, **Estiri, S.N.**, **Naderi, S.**, **Najafi, M.H. and Imani, M.** (2022). A fast and low-cost comparison-free sorting engine with unary computing: late breaking results, *Proceedings of the 59th ACM/IEEE Design Automation Conference*, DAC'22, San Francisco, California, USA, p.1390–1391, https://doi.org/10.1145/3489517.3530615.

[168] **Meyer, P.L.**, (1970). *Introductory Probability and Statistical Applications*, Addison-Wesley Publishing Company, p. 30.

[169] **Aygun, S. and Gunes, E.O.** (2022). Utilization of contingency tables in stochastic computing, *IEEE Transactions on Circuits and Systems II: Express Briefs*, *69*(6), 2942–2946.

[170] **Lee, V.**, **Alaghi, A.**, **Ceze, L. and Oskin, M.**, (2018), Stochastic synthesis for stochastic computing, https://arxiv.org/abs/1810.04756, arXiv.

[171] **Baker, T.J. and Hayes, J.P.** (2020). The hypergeometric distribution as a more accurate model for stochastic computing, *2020 Design, Automation and Test in Europe Conference Exhibition (DATE)*, Grenoble, France, pp.592–597.

[172] **Ma, C.**, **Zhong, S. and Dang, H.** (2012). Understanding variance propagation in stochastic computing systems, *2012 IEEE 30th International Conference on Computer Design (ICCD)*, Montreal, QC, Canada, pp.213–218.

[173] **Li, X.**, **Qian, W.**, **Riedel, M.**, **Bazargan, K. and Lilja, D.** (2009). A reconfigurable stochastic architecture for highly reliable computing, *the 19th ACM Great Lakes Symposium on VLSI*, Boston Area, MA, USA, p.315–320.

[174] **Asadi, S. and Najafi, M.H.** (2020). Late breaking results: LDFSM: A low-cost bit-stream generator for low-discrepancy stochastic computing, *2020 57th ACM/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, pp.1–2.

[175] **Asadi, S.**, **Najafi, M.H. and Imani, M.** (2021). A low-cost FSM-based bit-stream generator for low-discrepancy stochastic computing, *2021 Design, Automation and Test in Europe Conference Exhibition (DATE)*, Grenoble, France, pp.908–913.

[176] **Wang, Z.**, **Bovik, A.**, **Sheikh, H. and Simoncelli, E.** (2004). Image quality assessment: from error visibility to structural similarity, *IEEE Transactions on Image Processing*, *13*(4), 600–612.

[177] **Canny, J.** (1986). A computational approach to edge detection, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-8*(6), 679–698.

[178] **Deriche, R.** (1987). Using Canny's criteria to derive a recursively implemented optimal edge detector, *International Journal of Computer Vision*, *1*(2), 167–187.

[179] **Sobel, I. and Feldman, G.** (2015). A 3×3 isotropic gradient operator for image processing, *History and definition of the so-called "Sobel Operator"*, `http://dx.doi.org/10.13140/RG.2.1.1912.4965`.

[180] **Robert, L.G.** (1963). Machine perception of three-dimensional solids, *Ph.D. Dissertation*, Massachusetts Institute of Technology, Cambridge, MA, USA, `https://dspace.mit.edu/handle/1721.1/11589`.

[181] **Prewitt, J.M.S.** (1970). *Object Enhancement and Extraction*, Picture Processing and Psychopictorics, Academic Press, New York, NY.

[182] **Wong, M.M.**, **Wong, D.M.L.D.**, **Zhang, C. and Hijazin, I.** (2017). A new lightweight and high fault tolerance Sobel edge detection using stochastic computing, *International Journal of Computer and Electrical Engineering*, *9*(2), 403–420.

[183] **Joe, H. and Kim, Y.** (2019). Novel stochastic computing for energy-efficient image processors, *Electronics*, *8*(6), `https://www.mdpi.com/2079-9292/8/6/720`.

[184] **Li, P.**, **Qian, W.**, **Lilja, D.J.**, **Bazargan, K. and Riedel, M.D.** (2013). Case studies of logical computation on stochastic bit streams, *International Workshop on Power and Timing Modeling, Optimization and Simulation*, Springer Berlin Heidelberg, Berlin, Heidelberg, pp.235–244.

[185] **Najafi, M.H. and Lilja, D.J.** (2021). High quality down-sampling for deterministic approaches to stochastic computing, *IEEE Transactions on Emerging Topics in Computing*, *9*(1), 7–14.

[186] **Kho, D.**, **Fauzi, M.F.A. and Lim, S.L.** (2019). Hardware-based Sobel gradient computations for sharpness enhancement, *International Journal of Technology*, *10*(7), 1315–1325.

[187] **Mehra, R. and Verma, R.** (2012). Area efficient FPGA implementation of Sobel edge detector for image processing applications, *International Journal of Computer Applications*, *56*(16), 7–11.

[188] **Nosrat, A. and Kavian, Y.S.** (2012). Hardware description of multi-directional fast Sobel edge detection processor by VHDL for implementing on FPGA, *International Journal of Computer Applications*, *47*(25), 1–7.

[189] **Singh, S.**, **Saini, A.K. and Saini, R.** (2012). Real-time FPGA based implementation of color image edge detection, *International Journal of Image, Graphics and Signal Processing*, *4*(12), 19–25.

[190] **Vahapoğlu, E. and Altun, M.** (2016). Accurate synthesis of arithmetic operations with stochastic logic, *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, Pittsburgh, PA, USA, pp.415–420.

[191] **Lee, V.T.**, **Alaghi, A.**, **Pamula, R.**, **Sathe, V.S.**, **Ceze, L. and Oskin, M.** (2018). Architecture considerations for stochastic computing accelerators, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(11), 2277–2289.

[192] **Hsieh, T.Y.**, **Peng, Y.H. and Ku, C.C.** (2013). An efficient test methodology for image processing applications based on error-tolerance, *2013 22nd Asian Test Symposium*, Yilan, Taiwan, pp.289–294.

[193] **Alaghi, A.**, **Qian, W. and Hayes, J.P.** (2018). The promise and challenge of stochastic computing, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(8), 1515–1531.

[194] **Atallah, M.** (2001). Faster image template matching in the sum of the absolute value of differences measure, *IEEE Transactions on Image Processing*, *10*(4), 659–663.

[195] **Lewis, J.** (1995). Fast template matching, *Vision Interface 95, Canadian Image Processing and Pattern Recognition Society*, Canada, pp.120–123.

[196] **Hisham, M.**, **Yaakob, S.N.**, **Raof, R.**, **Nazren, A.A. and Wafi, N.** (2015). Template matching using sum of squared difference and normalized cross correlation, *2015 IEEE Student Conference on Research and Development (SCOReD)*, Kuala Lumpur, Malaysia, pp.100–104.

[197] **Di Stefano, L.**, **Mattoccia, S. and Mola, M.** (2003). An efficient algorithm for exhaustive template matching based on normalized cross correlation, *12th International Conference on Image Analysis and Processing*, Mantova, Italy, pp.322–327.

[198] **Xiong, L.**, **Zhong, X.**, **Xiong, N.N. and Liu, R.W.** (2021). QR-3S: A high payload QR code secret sharing system for industrial internet of things in 6G networks, *IEEE Trans. on Industrial Informatics*, *17*(10).

[199] **Ali, A.M. and Farhan, A.K.** (2020). Enhancement of QR code capacity by encrypted lossless compression technology for verification of secure e-document, *IEEE Access*, *8*, 27448–27458.

[200] **Lin, S.S.**, **Hu, M.C.**, **Lee, C.H. and Lee, T.Y.** (2015). Efficient QR code beautification with high quality visual content, *IEEE Transactions on Multimedia*, *17*(9), 1515–1524.

[201] **Alajmi, M.**, **Elashry, I.**, **El-Sayed, H.S. and Farag Allah, O.S.** (2020). Steganography of encrypted messages inside valid QR codes, *IEEE Access*, *8*, 27861–27873.

[202] **Press, W.H.**, **Teukolsky, S.A.**, **Vetterling, W.T. and Flannery, B.P.** (1995). *Numerical Recipes in C: The Art of Scientific Computing*, Camb. Univ. Press, New York, NY, USA.

[203] **Kim, K.H.**, **Shim, P.S. and Shin, S.** (2019). An alternative bilinear interpolation method between spherical grids, *Atmosphere*, *10*(3), `https://www.mdpi.com/2073-4433/10/3/123`.

[204] **Cao, C.**, **Wang, B.**, **Zhang, W.**, **Zeng, X.**, **Yan, X.**, **Feng, Z.**, **Liu, Y. and Wu, Z.** (2019). An improved faster R-CNN for small object detection, *IEEE Access*, *7*, 106838–106846.

[205] **Gribbon, K. and Bailey, D.** (2004). A novel approach to real-time bilinear interpolation, *IEEE International Workshop on Electronic Design, Test and Applications*, Perth, WA, Australia, pp.126–131.

[206] **Szeliski, R.** (2011). *Computer Vision Algorithms and Applications*, Texts in Computer Science. Springer, London., `http://dx.doi.org/10.1007/978-1-84882-935-0`.

[207] **Wang, Z. and Bovik, A.C.** (2009). Mean squared error: Love it or leave it? A new look at signal fidelity measures, *IEEE Signal Processing Magazine*, *26*(1), 98–117.

[208] **Bishop, C.M.** (2006). *Pattern Recognition and Machine Learning*, Information Science and Statistics, Springer New York, NY.

[209] **Nwankpa, C.**, **Ijomah, W.**, **Gachagan, A. and Marshall, S.**, (2018), Activation functions: Comparison of trends in practice and research for deep learning, `https://arxiv.org/abs/1811.03378`, arXiv.

[210] **Jurafsky, D. and Martin, J.H.** (2021). *Speech and Language Processing*, 3rd ed. draft.

[211] **Manwani, N.**, **Desai, K.**, **Sasidharan, S. and Sundararajan, R.** (2015). Double ramp loss based reject option classifier, *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp.151–163.

[212] **Liu, S.**, **Jiang, H.**, **Liu, L. and Han, J.** (2018). Gradient descent using stochastic circuits for efficient training of learning machines, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *37*(11), 2530–2541.

[213] **Courbariaux, M.**, **Hubara, I.**, **Soudry, D.**, **El-Yaniv, R. and Bengio, Y.**, (2016), Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1, `https://arxiv.org/abs/1602.02830`, arXiv.

[214] **Courbariaux, M.**, **Bengio, Y. and David, J.P.**, (2016), BinaryConnect: Training deep neural networks with binary weights during propagations, `https://arxiv.org/abs/1511.00363`, arXiv.

[215] **Hubara, I.**, **Courbariaux, M.**, **Soudry, D.**, **El-Yaniv, R. and Bengio, Y.** (2016). Binarized neural networks, *Advances in Neural Information Processing Systems (NIPS) 29*, Barcelona, Spain.

[216] **Simons, T. and Lee, D.J.** (2019). A review of binarized neural networks, *Electronics*, *8*(6), `https://www.mdpi.com/2079-9292/8/6/661`.

[217] **Glorot, X. and Bengio, Y.** (2010). Understanding the difficulty of training deep feedforward neural networks, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, Sardinia, Italy, pp.249–256.

[218] **Ren, A.**, **Li, Z.**, **Ding, C.**, **Qiu, Q.**, **Wang, Y.**, **Li, J.**, **Qian, X. and Yuan, B.** (2017). SC-DCNN: Highly-scalable deep convolutional neural network using stochastic computing, *ACM SIGPLAN Notices*, *52*(4), 405–418, `https://doi.org/10.1145/3093336.3037746`.

[219] **Hirtzlin, T.**, **Penkovsky, B.**, **Bocquet, M.**, **Klein, J.O.**, **Portal, J.M. and Querlioz, D.** (2019). Stochastic computing for hardware implementation of binarized neural networks, *IEEE Access*, *7*, 76394–76403.

[220] **Xiao, H.**, **Rasul, K. and Vollgraf, R.**, (2017), Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms, `https://arxiv.org/abs/1708.07747`, arXiv.

[221] **Umuroglu, Y.**, **Fraser, N.J.**, **Gambardella, G.**, **Blott, M.**, **Leong, P.**, **Jahre, M. and Vissers, K.** (2017). FINN: A framework for fast, scalable binarized neural network inference, *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, California, USA, p.65–74, `https://doi.org/10.1145/3020078.3021744`.

[222] **Agrawal, A.**, **Jaiswal, A.**, **Roy, D.**, **Han, B.**, **Srinivasan, G.**, **Ankit, A. and Roy, K.** (2019). Xcel-RAM: Accelerating binary neural networks in high-throughput SRAM compute arrays, *IEEE Transactions on Circuits and Systems I: Regular Papers*, *66*(8), 3064–3076.

[223] **Gupta, S.**, **Imani, M.**, **Zhao, H.**, **Wu, F.**, **Zhao, J. and Rosing, T.** (2020). Implementing binary neural networks in memory with approximate accumulation, *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, ISLPED'20, Boston, Massachusetts, p.247–252.

[224] **Jiang, L.**, **Kim, M.**, **Wen, W. and Wang, D.** (2017). XNOR-POP: A processing-in-memory architecture for binary convolutional neural networks in wide-IO2 DRAMs, *2017 IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, Taipei, Taiwan, pp.1–6.

[225] **Godse, A. and Bakshi, U.** (2009). *Digital and Linear Integrated Circuits*, Technical Publications Pune.

[226] **Yayla, M.**, **Buschjager, S.**, **Gupta, A.**, **Chen, J.**, **Henkel, J.**, **Morik, K.**, **Chen, K. and Amrouch, H.** (2022). FeFET-based binarized neural networks under temperature-dependent bit errors, *IEEE Transactions on Computers*, *71*(07), 1681–1695.

[227] **Yayla, M.**, **Thomann, S.**, **Buschjager, S.**, **Morik, K.**, **Chen, J.J. and Amrouch, H.** (2022). Reliable binarized neural networks on unreliable beyond Von-Neumann architecture, *IEEE Transactions on Circuits and Systems I: Regular Papers*, *69*(6), 2516–2528.

[228] **Riahi Alam, M.**, **Najafi, M.H.**, **TaheriNejad, N.**, **Imani, M. and Gottumukkala, R.** (2022). Stochastic computing in beyond von-Neumann era: Processing bit-streams in memristive memory, *IEEE Transactions on Circuits and Systems II: Express Briefs*, *69*(5), 2423–2427.

[229] **Aygun, S. and Gunes, E.O.** (2020). On the simulation of software-driven stochastic computing for emerging applications, *Design, Automation and Test in Europe Conference (DATE'20) - Workshop on SCONA: Stochastic COmputing for Neuromorphic Architectures*, Grenoble, France.

[230] **Clanuwat, T.**, **Bober-Irizar, M.**, **Kitamoto, A.**, **Lamb, A.**, **Yamamoto, K. and Ha, D.**, (2018), Deep learning for classical Japanese literature, `https://arxiv.org/abs/1812.01718`, arXiv.

[231] **CuPy**, (2021), "cupy.random.shuffle", `https://github.com/cupy/cupy/tree/v9.1.0/cupy/random`, (Date Retrieved: 22.06.2021).

[232] **Montobbio, N.**, **Bonnasse-Gahot, L.**, **Citti, G. and Sarti, A.** (2019). KerCNNs: biologically inspired lateral connections for classification of corrupted images, `http://arxiv.org/abs/1910.08336`, arXiv.

CURRICULUM VITAE

Name Surname : Sercan AYGÜN

EDUCATION :

- **B.Sc.** : 2013, Eskisehir Osmangazi University, Faculty of Architecture-Engineering, Department of Electrical-Electronics Engineering
- **B.Sc.** : 2013, Eskisehir Osmangazi University, Faculty of Architecture-Engineering, Department of Computer Engineering
- **M.Sc.** : 2015, Istanbul Technical University, Graduate School of Science Engineering and Technology, Electronics and Communication Engineering Department, Electronics Engineering Graduate Program
- **M.Sc.** : 2016, Anadolu University, Graduate School of Sciences, Department of Computer Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2013-2022 Istanbul Technical University, Turkey, Researcher at the System Modeling, Research, and Design Laboratory
- 2014-2022 Yildiz Technical University, Turkey, Research Assistant at the Department of Computer Engineering
- 2018-2019 Université Catholique de Louvain, Belgium, Exchange Ph.D. Student at the Institute of Information and Communication Technologies, Electronics and Applied Mathematics
- 2021-2022 University of Louisiana at Lafayette, LA, USA, Research Scholar at the School of Computing and Informatics

- **Reviewing activities:**
  Journals:
  -IEEE Transactions on Very Large Scale Integration (VLSI) Systems,
  -IEEE Journal on Emerging and Selected Topics in Circuits and Systems,
  -IEEE Access,
  -IEEE Sensors Journal,
  -Elsevier Applied Soft Computing (ASOC),
  -IET Wiley Electronics Letters,
  -PeerJ Computer Science,
  -The Journal Tehnički Vjesnik – Technical Gazette (TV-TG),
  -TÜBİTAK - Journal of Engineering Sciences and Design,
  -Journal of Agricultural Science and Technology,
  -Hindawi Wireless Communications and Mobile Computing,
  -Jordanian Journal of Computers and Information Technology,
  -Sigma Journal of Engineering and Natural Sciences.

  Conferences:
  -Int. Conference on VLSI Design (VLSID),
  -IEEE/ACM Int. Conference on Computer-Aided Design (ICCAD),
  -ACM/IEEE Int. Symposium on Low Power Electronics and Design (ISLPED),
  -IEEE Int. Power and Renewable Energy Conference (IPRECON),
  -IEEE Signal Processing and Communications Applications Conference (SIU).

PUBLICATIONS AND PRESENTATIONS ON THE THESIS:

- **Aygun, S.** and Gunes, E. O. 2022. Stochastic Bit-Stream Processing Learning Machines: Agile Simulation and Lightweight Network Design, *ACM Student Research Competition (SRC)*, Presented in *ESWEEK Conference* (SIGBED) and obtained *Best Scientific Research Award*, and accepted in *ICCAD Conference* (SIGDA) with a grant, October, 2022.

- **Aygun, S.** and Gunes, E. O. 2022. Stochastic Bit-Stream Processing Systems with New Perspectives Towards Vision and Learning Machines, *Ph.D. Forum at Embedded Systems Week (ESWEEK)*, October, 2022, Shanghai & Phoenix.

- **Aygun, S.**, Najafi, M. H., Imani, M., and Gunes, E. O. Agile Simulation of Stochastic Computing Image Processing with Contingency Tables, *Submitted*.

- **Aygun, S.** and Gunes, E. O. 2022. Design and Simulation of Stochastic Bitstream Processing Systems with New Perspectives Towards Vision and Learning Machines, *Ph.D. Forum at the Design Automation Conference (DAC)*, Student Travel Grant by ACM, July, 2022, San Francisco, California, USA.

- **Aygun, S.** and Gunes, E. O. 2022. Utilization of Contingency Tables in Stochastic Computing, *IEEE Transactions on Circuits and Systems II: Express Briefs*, 69(6), 2942-2946.

- **Aygun, S.** and Gunes, E. O. 2021. On the Limit of Multiplexers in Stochastic Computing, *International Journal of Multidisciplinary Studies and Innovative Technologies*, 5(1), 94-97.

- **Aygun, S.**, Gunes, E. O., and De Vleeschouwer, C. 2021. Efficient and Robust Bitstream Processing in Binarised Neural Networks, *Electronics Letters*, 57(5), 219-222.

- **Aygun, S.** and Gunes, E. O. 2020. On the Simulation of Software-Driven Stochastic Computing for Emerging Applications, *Design, Automation and Test in Europe Conference (DATE'20) - Workshop on SCONA: Stochastic COmputing for Neuromorphic Architectures*, March, 2020, Grenoble, France.

- **Aygun, S.**, Altun, M., and Gunes, E. O. 2017. Sobel Filter Operation in Image Processing via Stochastic Arithmetic-Logic Unit Design, *IEEE 25th Signal Processing and Communications Applications Conference (SIU)*, May, 2017, Antalya, Turkey.

OTHER PUBLICATIONS AND PRESENTATIONS:

- Schober, P., Estiri, S. N., **Aygun, S.**, TaheriNejad, N., Najafi, M. H. 2022. Sound Source Localization using Stochastic Computing, *ICCAD 2022, The 41st IEEE/ACM International Conference on Computer-Aided Design*, November, 2022, San Diego, California, USA.

- **Aygun, S.**, Katipoglu, M. Time-Bin-Based Neighbourhood Algorithm for Temporal Effects in Recommendation Systems, *Tehnički Vjesnik - Technical Gazette*, Accepted, in Press.

- Okyay, S. and **Aygun, S.** 2021. Experimental Interpretation of Adequate Weight-Metric Combination for Dynamic User-Based Collaborative Filtering, *PeerJ Computer Science*, 7:e784.

- Okyay, S. and **Aygun, S.** 2021. A Significant Fallacy of Built-in Correlation Functions in Recommender Systems, *3rd International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA)*, June, 2021, Ankara, Turkey.

- Balayev, K., Guluzade, N., **Aygun, S.**, and Ilhan, H. O. 2021. The Implementation of DCGAN in the Data Augmentation for the Sperm Morphology Datasets, *European Journal of Science and Technology*, Special Issue 26, 307-314.

- **Aygun, S.**, Kouhalvandi, L., Gunes, E. O., and Ozoguz, S. 2020. Inspecting Distortion in the Power Amplifiers with the aid of Neural Networks, *22nd International Conference on Advanced Communication Technology*, February, 2020, Phoenix Park, South Korea.

- Okyay, S. and **Aygun, S.** 2020. A Study of Static and Dynamic Significance Weighting Multipliers on the Pearson Correlation for Collaborative Filtering, *European Journal of Science and Technology*, Special Issue, 270-275.

- **Aygun, S.**, Gunes, E. O., Subasi, M. A., and Alkan, S. 2019. Sensor Fusion for IoT-Based Intelligent Agriculture System, *IEEE 8th International Conference on Agro-Geoinformatics*, July, 2019, Istanbul, Turkey.

- **Aygun, S.**, Cavus, G., and Gunes, E. O. 2018. Less Data for Face and Iris Biometric Traits: An Approximation for Feature Extraction, *IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, November, 2018, Vilnius, Lithuania.

- **Aygun, S.**, Kouhalvandi, L., and Gunes, E. O. 2018. 1.5 Bit/Stage, 12-Bit Pipeline ADC Design with Foreground Calibration, *International Journal of Advancements in Electronics & Electrical Engineering*, 6(2), 164-168.

- **Aygun, S.**, Kouhalvandi, L., Ors Yalcin, S.B., Gunes, E. O. 2017. Karatsuba Ofman Multiplication Implementation on SystemC for Diffie-Hellman Key Exchange Algorithm, *IEEE 4th International Conference on Knowledge-Based Engineering and Innovation (KBEI)*, December, 2017, Tehran, Iran.

- Kouhalvandi, L., **Aygun, S.**, Gunes, E. O., Kirci, M. 2017. Design of a High Gain Telescopic-Cascode Operational Amplifier Based on the ZTC Operation Condition, *IEEE 24th International Conference on Electronics, Circuits and Systems (ICECS)*, December, 2017, Batumi, Georgia.

- Kouhalvandi, L., **Aygun, S.**, Gunes, E. O., Kirci, M. 2017. An Improved 2 Stage Opamp with Rail-to-Rail Gain-Boosted Folded Cascode Input Stage and Monticelli Rail-to-Rail Class AB Output Stage, *IEEE 24th International Conference on Electronics, Circuits and Systems (ICECS)*, December, 2017, Batumi, Georgia.

- Kouhalvandi, L., **Aygun, S.**, Ozdemir, G. G., and Gunes, E. O. 2017. 10-bit High-Speed CMOS Comparator with Offset Cancellation Technique, *IEEE 5th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, November, 2017, Riga, Latvia.

- Kouhalvandi, L., **Aygun, S.**, Gunes, E. O., and Kirci, M. 2017. Design of a Fully-Differential Double Folded Cascode Class AB Opamp with Continuous Time Common Mode Feedback Network for 12-bit Pipeline ADC Applications, *IEEE International Conference on Computer Science and Engineering (UBMK)*, October, 2017, Antalya, Turkey.

- **Aygun, S.** and Gunes, E. O. 2017. A Benchmarking: Feature Extraction and Classification of Agricultural Textures using LBP, GLCM, RBO, Neural Networks, k-NN, and Random Forest, *IEEE 6th International Conference on Agro-Geoinformatics*, August, 2017, Fairfax, VA, USA.

- Gunes, E. O. and **Aygun, S.** 2017. Growth Monitoring of Plants Using Active Contour Technique, *IEEE 6th International Conference on Agro-Geoinformatics*, August, 2017, Fairfax, VA, USA.

- **Aygun, S.**, Yalcin, H., and Gunes, E. O. 2017. Seed Texture Classification by Random Forest and Neural Networks, *IEEE 25th Signal Processing and Communications Applications Conference (SIU)*, May, 2017, Antalya, Turkey.

- **Aygun, S.**, Gunes, E. O., and Kouhalvandi, L. 2016. Python Based Parallel Application of Knuth-Morris-Pratt Algorithm, *IEEE 4th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, November, 2016, Vilnius, Lithuania.

- Kouhalvandi, L., Cilasun, M. H., and **Aygun, S.** 2016. Recognition of Apple and Orange Fruit Pictures Based on Boundary Estimation, *3rd International Conference on Recent Innovations in Electrical and Computer Engineering*, September, 2016, Tehran, Iran.

- **Aygun, S.**, Yazgac, B. G., Kirci, M., and Gunes, E. O. 2016. Design of an Embedded System for Automatically Determination of the Varieties of Wheat Seeds Grown in Turkey Based on Image Analysis, *IEEE 5th International Conference on Agro-Geoinformatics*, July, 2016, Tianjin, China.

- **Aygun, S.** and Gunes, E. O. 2016. Computer Vision Techniques for Automatic Determination of Yield Effective Bad Condition Storage Effects on Various Agricultural Seed Types, *IEEE 5th International Conference on Agro-Geoinformatics*, July, 2016, Tianjin, China.

- **Aygun, S.**, Akcay, M., and Gunes, E. O. 2016. Bulut Sistemleri için Biyometrik Tabanlı Güvenlik Sistemleri, *Uluslararası Bilgi Güvenliği Mühendisliği Dergisi*, 2(1), 15-22.

- **Aygun, S.** and Akcay, M. 2016. Multibiometrics Approach on Biometric Passport Pictures by Using Fingerprint Minutiae Points, *IEEE 4th International Symposium on Digital Forensic and Security (ISDFS)* April, 2016, Little Rock, AR, USA (Best Student Paper Award).

- **Aygun, S.** and Okyay, S. 2015. Improving the Pearson Similarity Equation for Recommender Systems by Age Parameter, *IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*, November, 2015, Riga, Latvia.

- **Aygun, S.** and Akcay, M. 2015. Securing Biometric Face Images via Steganography for QR Code, *8th International Conference on Information Security and Cryptology, ISCTurkey*, October, 2015, Ankara, Turkey (Best Poster Award).

- **Aygun, S.** and Akcay, M. 2015. Yüz Tanıma Teknolojilerinde Yüksek Başarım için Paralel Hesaplama, *4th National High Performance Computing Conference, BAŞARIM*, October, 2015, Ankara, Turkey.

- **Aygun, S.** and Akcay, M. 2015. Matlab Paralel Hesaplama Aracı ile A* Algoritmasının Rota Planlama için Analizi, *Genç Mühendisler Sempozyumu*, May, 2015, Istanbul, Turkey.

- Gunes, E. O., **Aygun, S.**, Kirci, M., Kalateh, A., and Cakir, Y. 2014. Determination of the Varieties and Characteristics of Wheat Seeds Grown in Turkey Using Image Processing Techniques, *IEEE 3rd International Conference on Agro-Geoinformatics*, August, 2014, Beijing, China.