National Technical University of Athens
School of Electrical and Computer Engineering
Division of Communication, Electronic and Information Engineering

Stochastic Computing Architectures for Information Processing Systems

by

Nikolaos Temenos

Supervisor
Paul-Peter Sotiriadis, Professor, NTUA

A dissertation submitted to the
Department of Electrical and Computer Engineering
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy

Athens, December 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Τομέας Επικοινωνιών, Ηλεκτρονικής & Συστημάτων Πληροφορικής

Stochastic Computing Architectures for Information Processing Systems

Διδακτορική Διατριβή

Νικόλαος Τέμενος

Υποβλήθηκε στο Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
σε μερική εκπλήρωση των απαιτήσεων για την απόκτηση διδακτορικού διπλώματος

Αθήνα, Δεκέμβριος 2022

National Technical University of Athens
School of Electrical and Computer Engineering
Division of Communication, Electronic and Information Engineering

Stochastic Computing Architectures for Information Processing Systems

PhD Dissertation

Nikolaos Temenos

Advisory Committee:

Paul - Peter Sotiriadis	Athanasios D. Panagopoulos	Kiamal Pekmestzi
Professor, NTUA	Professor, NTUA	Professor Emeritus, NTUA

Approved by the seven-member examination committee on 13/12/2022.

Paul - Peter Sotiriadis	Athanasios D. Panagopoulos	Kiamal Pekmestzi
Professor, NTUA	Professor, NTUA	Professor Emeritus, NTUA

Theodora Varvarigou	Panayiotis Psarrakos	Anastasios Doulamis
Professor	Professor	Associate Professor

Nikolaos Doulamis
Professor

Athens, December 2022

The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) under the HFRI PhD Fellowship grant (Fellowship Number:1216).

.....

Νικόλαος Τέμενος

Abstract

Arithmetic operations on stochastic sequences is the basis of the unconventional computational approach known as Stochastic Computing (SC). Deviating from the standard binary arithmetic, SC encodes and processes the value of binary numbers in the form of stochastic sequences, making arithmetic operations and highly-complex functions realizable using a few simple standard logic gates and memory elements, having inherent natural robustness in soft-errors. SC's properties and advantages have been exploited in a plethora of fields characterized by massive parallelism requirements like Neural Networks and Image Processing. Beyond its strong points, SC introduces an accuracy-latency trade-off impacting the energy efficiency. Therefore, achieving low latency along with increased computational accuracy is the primary design goal is SC systems.

This dissertation presents novel SC architectures realizing essential arithmetic operations and non-linear functions, as well as realistic Neural Networks and Image Processing applications based on them.

In the first part of the dissertation, the operating principles of the architectures are introduced and their behavior is modeled based on Stochastic Finite-State Machines (SFSM) and analyzed using Markov Chains (MC). This leads to a deeper understanding of their stochastic dynamics and the verification of their proper operation. The MC modeling is further extended to a general methodology enabling the analytical derivation of the SFSMs' first and second moment statistical properties. The methodology is accompanied by overflow/underflow MC modeling, allowing to balance the accuracy-latency trade-off according to performance requirements, and to set the guidelines for the selection of the register's size.

In the second part of the dissertation, the proposed architectures are compared to existing ones, in the SC literature, in computational accuracy and hardware resources, including area, power and energy consumption as well as in terms of their advantages in the overall design flow. The efficacy of the architectures is demonstrated by using them as building blocks in the realization of several Digital Signal Processing (DSP) operations, including convolution, noise reduction and image down-sampling filters as well as Neural Networks. Finally, the results of the introduced architectures' performance in computational accuracy and hardware resources are compared to those achieved using standard binary computing methods highlighting the advantages of the first ones.

Keywords: Stochastic Computing, Stochastic FSM, Markov Chain Modeling, Digital Circuits

Περίληψη

Οι αριθμητικές πράξεις με στοχαστικές ακολουθίες είναι η βάση της μη συμβατικής τεχνικής που είναι γνωστή ως Στοχαστικός Υπολογισμός (ΣΥ). Αποκλίνοντας από την τυπική δυαδική αριθμητική, ο ΣΥ κωδικοποιεί και επεξεργάζεται την τιμή των δυαδικών αριθμών με τη μορφή στοχαστικών ακολουθιών, καθιστώντας δυνατή την πραγματοποίηση αριθμητικών πράξεων και εξαιρετικά πολύπλοκων συναρτήσεων με τη χρήση λίγων τυπικών λογικών πυλών και στοιχείων μνήμης. Μαζί με την φυσική ευρωστία του ΣΥ σε σφάλματα, οι ιδιότητες και τα πλεονεκτήματά του έχουν αξιοποιηθεί σε πληθώρα πεδίων με ανάγκες μαζικού παραλληλισμού και μικρή ανοχή σε σφάλματα, συμπεριλαμβανομένων των νευρωνικών δικτύων και της επεξεργασίας εικόνας μεταξύ πολλών άλλων. Πέρα από τα ισχυρά του σημεία, ο ΣΥ εισάγει ένα συμβιβασμό ακρίβειας-καθυστέρησης που επηρεάζει την ενεργειακή απόδοση και, ως εκ τούτου, για να αξιοποιηθεί με τον καλύτερο δυνατό τρόπο, η επίτευξη χαμηλής καθυστέρησης σε συνδυασμό με αυξημένη υπολογιστική ακρίβεια είναι πρωταρχικό μέλημα.

Στην παρούσα διατριβή παρουσιάζονται νέες αρχιτεκτονικές που υλοποιούν βασικές αριθμητικές πράξεις και μη γραμμικές συναρτήσεις σε ΣΥ. Οι εσωτερικοί καταχωρητές που χρησιμοποιούν καθιστούν την επεξεργασία της ακολουθίας εισόδου τους αιτιοκρατική, βελτιώνοντας έτσι το συμβιβασμό ακρίβειας-καθυστέρησης του ΣΥ. Για να αναδειχθούν οι ιδιότητες και η αρχή λειτουργίας των αρχιτεκτονικών, αναλύονται διεξοδικά με τη χρήση στοχαστικών μηχανών πεπερασμένης κατάστασης (ΣΜΠΚ) και μοντελοποιούνται με τη χρήση αλυσίδων Markov (ΑΜ).

Στο πρώτο μέρος της διατριβής, η αρχή λειτουργίας των αρχιτεκτονικών αναλύεται με τη χρήση ΣΜΠΚ και μοντελοποιείται με τη χρήση ΑΜ, η οποία επιτρέπει την καλύτερη κατανόηση της μακροπρόθεσμης στοχαστικής δυναμικής τους και την επαλήθευση της ορθής λειτουργίας τους. Η μοντελοποίηση της ΑΜ επεκτείνεται περαιτέρω σε μια γενική μεθοδολογία που επιτρέπει την αναλυτική εξαγωγή των στατιστικών ιδιοτήτων της πρώτης και της δεύτερης ροπής των ΣΜΠΚ. Η μεθοδολογία συνοδεύεται από μοντελοποίηση ΑΜ υπερχείλισης/υποχείλισης, επιτρέποντας την εκτίμηση του αριθμού των καταστάσεων που μειώνουν τα σφάλματα ψηφίων που προέρχονται από την εμφάνιση υπερχείλισης/υποχείλισης, θέτοντας έτσι τις κατευθυντήριες γραμμές για την επιλογή του μεγέθους του καταχωρητή που χρησιμοποιούν.

Στο δεύτερο μέρος της διατριβής, οι αρχιτεκτονικές συγκρίνονται εκτενώς με τις υπάρχουσες στη βιβλιογραφία των ΣΥ όσον αφορά την υπολογιστική ακρίβεια και τους πόρους υλικού, συμπεριλαμβανομένου του χώρου που καταλαμβάνουν τα κυκλώματα, την κατανάλωση ισχύος και ενέργειας, καθώς και τα οφέλη που εισάγουν στη συνολική ροή σχεδίασης. Η αποτελεσματικότητα των αρχιτεκτονικών αναδεικνύεται με τη χρήση τους ως δομικά στοιχεία στην υλοποίηση διαφόρων επεξεργαστικών μονά-

δων, που περιλαμβάνουν συνέλιξη, φίλτρα μείωσης θορύβου και υποδειγματοληψίας εικόνας, καθώς και νευρωνικά δίκτυα. Τα αποτελέσματα των επιδόσεων των αρχιτεκτονικών όσον αφορά την υπολογιστική ακρίβεια και τους πόρους υλικού συγκρίνονται με εκείνα που επιτυγχάνονται με τη χρήση τυπικών δυαδικών μεθόδων υπολογισμού, προβάλλοντας τα πλεονεκτήματά του ΣΥ ως πολλά υποσχόμενη εναλλακτική μορφή επεξεργασίας σημάτων.

Λεξεις Κλειδια: Στοχαστικός Υπολογισμός, Στοχαστικά Αυτόματα, Αλυσίδα Markov, Ψηφιακά Κυκλώματα

Εκτεταμένη Περίληψη

Στην παρούσα διατριβή παρουσιάζονται καινοτόμες ψηφιακές αρχιτεκτονικές που υλοποιούν βασικές αριθμητικές πράξεις και μη γραμμικές συναρτήσεις στον στοχαστικό υπολογισμό (Stochastic Computing - SC). Η ανάλυσή τους γίνεται διεξοδικά με τη χρήση στοχαστικών μηχανών πεπερασμένων καταστάσεων (Stochastic Finite-State Machines - SFSMs), ενώ για την κατανόηση της στοχαστικής δυναμικής συμπεριφοράς τους, μοντελοποιούνται μέσω αλυσίδων Markov (Markov Chains - MC). Τα πλεονεκτήματα και η αποτελεσματικότητα τους, επιδεικνύονται με την αξιοποίησή τους στην υλοποίηση δομικών μονάδων ψηφιακής επεξεργασίας σήματος (Digital Signal Processing - DSP).

Ο στοχαστικός υπολογισμός ανήκει στην κατηγορία των μη συμβατικών μεθόδων υπολογισμού, καθώς κωδικοποιεί και επεξεργάζεται την τιμή δυαδικών αριθμών με τη μορφή στοχαστικών ακολουθιών του ενός ψηφίου (bit). Η σειριακή επεξεργασία σε επίπεδο ψηφίου, επιτρέπει την υλοποίηση θεμελιωδών αριθμητικών πράξεων με τη χρήση μεμονομένων λογικών πυλών (logic gates), ενώ ιδιαίτερα πολύπλοκες συναρτήσεις, όπως για παράδειγμα μη γραμμικές, υλοποιούνται απλά με τη χρήση μηχανών πεπερασμένων καταστάσεων. Επιπλέον, δεδομένης της πιθανοτικής φύσης του, ο στοχαστικός υπολογισμός είναι εγγενώς ανεκτικός σε σφάλματα (soft errors), που σημαίνει ότι η αντιστροφή μερικών ψηφίων δεν είναι επιζήμια για την πληροφορία του ίδιου του σήματος. Πέραν των πλεονεκτημάτων του, ο στοχαστικός υπολογισμός εισάγει μία αντιστάθμιση (trade-off) μεταξύ του μήκους των στοχαστικών ακολουθιών προς επεξεργασία και της ακρίβειας του χρονικού μέσου όρου του στοχαστικού αριθμού. Ως εκ τούτου, για να αξιοποιηθεί με τον καλύτερο δυνατό τρόπο, η επίτευξη χαμηλής καθυστέρησης (latency) σε συνδυασμό με αυξημένη υπολογιστική ακρίβεια αποτελεί πρωταρχικό μέλημα σχεδιασμού σε συστήματα βασισμένα στο στοχαστικό υπολογισμό, ώστε να αποφευχθεί η συνολική κατανάλωση ενέργειας.

Τα πλεονεκτήματα και οι ιδιότητες του στοχαστικού υπολογισμού, ευνοούν κατά κόρον εφαρμογές που η υλοποίησή τους και οι απαιτήσεις τους συνδυάζουν ταυτόχρονα ανάγκες για μαζικό παραλληλισμό, περιορισμό σε έκταση και ανοχή σε μικρές αποκλίσεις από τους ακριβείς υπολογισμούς. Οι εφαρμογές αυτές περιλαμβάνουν τα τεχνητά νευρωνικά δίκτυα εμπρόσθιας τροφοδότησης (Artificial Neural Networks - ANNs) με έμφαση στα πολυεπίπεδα perceptron (Multi-Layer Perceptrons - MLPs) και τα συνελικτικά νευρωνικά δίκτυα (Convolutional Neural Networks - CNN) στον τομέα της τεχνητής νοημοσύνης, τις μηχανές διανυσμάτων υποστήριξης (Support Vector Machines - SVMs) στον τομέα της μηχανικής μάθησης (Machine Learning - ML) και των φίλτρων χωρικής βελτίωσης συμπεριλαμβανομένων των φίλτρων μείωσης θορύβου, διάμεσης τιμής, ενίσχυσης ευκρίνειας εικόνας και άλλων στον τομέα της ψηφιακής επεξεργασίας εικόνας (Digital Image Processing -DIP). Ωστόσο, ο στοχαστικός υπολογισμός δεν περιορίζεται μόνο στα ανωτέρω πεδία, καθώς έχει εφαρμοστεί με επιτυχία στο ελαστικό φιλτράρι-

σμα (soft filtering), που συμπεριλαμβάνει φίλτρα πεπερασμένης κρουστικής απόκρισης (Finite Impulse Response - FIR) και άπειρης κρουστικής απόκρισης (Infinite Impulse Response - IIR), στην κωδικοποίηση/αποκωδικοποίηση διόρθωσης σφαλμάτων (error correcting codes), στην επίλυση πολυωνύμων (polynomial solving) και σε άλλα.

Μια βασική λειτουργία που εκτελείται στους πυρήνες ψηφιακής επεξεργασίας σήματος σε όλες τις παραπάνω εφαρμογές, είναι αυτή του πολλαπλασιασμού και της πρόσθεσης (multiply-and-add). Στο στοχαστικό υπολογισμό, η πράξη του πολλαπλασιασμού αποτελεί έναν από τους βασικότερους παράγοντες που τον καθιστούν ελκυστικό, καθώς υλοποιείται απλά με τη χρήση μίας πύλης AND ή XNOR, αναλόγως με την αναπαράσταση αριθμών που χρησιμοποιείται. Το κομμάτι της πρόσθεσης όμως, υλοποιείται τυπικά από έναν πολυπλέκτη (Multiplexer - MUX), ο οποίος απαιτεί μια επιπρόσθετη πηγή τυχαίων αριθμών για το σήμα επιλογής του, πέραν των δύο εισόδων του. Ωστόσο, η πηγή τυχαίων αριθμών αποτελεί από μόνη της ογκώδες δομικό κομμάτι, αφού σε σύγκριση με τις υπόλοιπες λογικές πύλες που χρησιμοποιούνται, καταλαμβάνει το μεγαλύτερο μέρος της επιφάνειας της σχεδίασης. Επιπλέον, η έξοδος του αθροιστή συνήθως κλιμακώνεται κατά το ήμισυ, που σημαίνει ότι για ένα δεδομένο μήκος ακολουθίας η ανάλυση μειώνεται στο μισό, ενώ η μείωση της ανάλυσης, εντείνεται περαιτέρω όταν υφίστανται αρκετοί κλιμακωτοί υπολογισμοί. Ως εκ τούτου, ο πολυπλέκτης είναι η λιγότερο ελκυστική επιλογή για άθροιση, αφού σε όλα τα παραπάνω προστίθεται και η αυξημένη κατανάλωση ενέργειας δεδομένης της ανάγκης για αύξηση της ανάλυσης της ακολουθίας εξόδου. Τα ίδια μειονεκτήματα παρουσιάζει και ο αφαιρέτης ο οποίος υλοποιείται με τη χρήση πολυπλέκτη, με τη μόνη διαφορά οτι περιορίζεται σε μόνο μία από τις δύο βασικές αναπαραστάσεις του στοχαστικού υπολογισμού.

Για την αντιμετώπιση των μειονεκτημάτων που εισάγει ο πολυπλέκτης, έχουν διερευνηθεί διάφοροι αθροιστές και αφαιρέτες, εστιάζοντας ταυτόχρονα στην υπολογιστική και σχεδιαστική αποδοτικότητα. Μία προσέγγιση βασισμένη στην αρχή κλιμάκωσης που εισάγει ο πολυπλέκτης, αποφεύγει την επιπλέον πηγή τυχαίων αριθμών στο σήμα επιλογής, αντικαθιστόντας τη με ένα στοιχείο μνήμης T Flip-Flop, αυξάνοντας παράλληλα την ακρίβεια στους υπολογισμούς. Μια παρόμοια, κλιμακωτή προσέγγιση, επεκτείνει τη χρήση του ενός στοιχείου μνήμης T Flip-Flop σε παραπάνω, εφαρμόζοντας μία μηχανή πεπερασμένων καταστάσεων για να αυξήσει περαιτέρω την ακρίβειά του. Αναφορικά με τους μη-κλιμακωτούς αθροιστές, μία προσέγγιση βασίζεται στην αναπαράσταση ενός στοχαστικού αριθμού που φέρει την πληροφορία του σε δύο ακολουθίες, μία για το πρόσημό του και μία για την τάξη μέγεθους του. Αν και είναι μια πολλά υποσχόμενη προσέγγιση σε επίπεδο εφαρμογής, η κωδικοποίηση στοχαστικού αριθμού μέσω δύο ακολουθιών επιβάλλει περιορισμούς στη συνολική σχεδίαση, καθώς απαιτεί από τις υπόλοιπες πράξεις, π.χ. πολλαπλασιαστές, να ακολουθούν επίσης αυτή την αρχή λειτουργίας. Ομοίως με τον προηγούμενο αθροιστή, άλλη προσέγγιση κωδικοποιεί έναν στοχαστικό αριθμό χρησιμοποιώντας τον λόγο των λογικών μονάδων και μηδενικών μεταξύ των ακολουθιών εισόδου του. Ωστόσο, η αναπαράσταση αυτή είναι ασύμβατη με τις τυπικές αναπαραστάσεις που χρησιμοποιούνται στο στοχαστικό υπολογισμό, ενώ η παραγωγή δύο ακολουθιών για έναν μόνο στοχαστικό αριθμό, επηρεάζει τη συνολική αξιοποίηση των πόρων και του υλικού.

Όσον αφορά τους στοχαστικούς αφαιρέτες, μία προσέγγιση συσχετίζει τις ακολουθίες εισόδου. Αυτό, ωστόσο, απαιτεί προσοχή, καθώς ο στοχαστικός υπολογισμός είναι επιρρεπής σε σφάλματα που προκαλούνται από συσχετισμένες εισόδους. Επιπλέον, εάν η αφαίρεση είναι μια ενδιάμεση αριθμητική πράξη,

δηλαδή υφίσταται μεταξύ δύο άλλων υπολογισμών, η αναγέννηση συσχετισμένων εισόδων είναι απαραίτητη, αυξάνοντας την αξιοποίηση των πόρων και του υλικού. Μια άλλη προσέγγιση εφαρμόζει επιπλέον λογικές μονάδες για να βελτιώσει την ακρίβεια μιας πύλης XNOR η οποία προσεγγίζει την αφαίρεση, ανταλλάσσοντας πόρους υλικού και καθυστέρηση για ακρίβεια υπολογισμών, και τα δύο βασισμένα στον αριθμό των επιπλέον λογικών μονάδων που χρησιμοποιούνται.

Συνοψίζοντας από τα παραπάνω, οι περισσότερες προσεγγίσεις ανταλλάσσουν το χρόνο εκτέλεσης ή/και την επιφάνεια του υλικού για την ακρίβεια υπολογισμών. Επιπλέον, ορισμένες από αυτές εισάγουν περιορισμούς που μειώνουν την ευελιξία στο χώρο σχεδιασμού του στοχαστικού υπολογισμού. Με κίνητρο τα προαναφερθέντα, στην παρούσα εργασία προτείνονται αρχιτεκτονικές μη κλιμακωτών αθροιστών και αφαιρετών. Τα πλεονεκτήματα που προσφέρουν είναι πολυάριθμα: δεν απαιτούν καμία πηγή τυχαίων αριθμών, δεν κλιμακώνουν το αποτέλεσμα εξόδου, λειτουργούν με ανεξάρτητες και πανομοιότυπα κατανεμημένες ακολουθίες εισόδου, δηλαδή δεν απαιτούνται ειδικά συσχετισμένες είσοδοι, είναι συμβατές με τις τυπικές αναπαραστάσεις αριθμών του στοχαστικού υπολογισμού και επιτυγχάνουν υψηλή υπολογιστική ακρίβεια χρησιμοποιώντας μικρά μήκη ακολουθιών εισόδου.

Σε περιπτώσεις όπου χρειάζεται η πράξη του πολλαπλασιασμού και της πρόσθεσης να γίνει μαζικά, η χρήση μεμονομένων αθροιστών σε δομή δένδρου εισάγει προκλήσεις σε επίπεδο αύξησης υλικού και ταχύτητας σχεδίασης. Για να αντιμετωπιστούν αυτές, στον στοχαστικό υπολογισμό εξετάζεται ο συσσωρευτικός παράλληλος μετρητής (Accumulative Parallel Counter - APC), ο οποίος αθροίζει αιτιοκρατικά όλες τις ακολουθίες εισόδου, παράγοντας το αποτέλεσμα σε δυαδική μορφή. Ωστόσο, σε αλυσιδωτούς υπολογισμούς η δυαδικό έξοδος του APC εισάγει τις ακόλουθες προκλήσεις σχεδιασμού: 1) περιορίζει την εφαρμοσιμότητα των υφιστάμενων στοχαστικών μηχανών πεπερασμένων καταστάσεων που υλοποιούν ιδιαίτερα πολύπλοκες συναρτήσεις, συμπεριλαμβανομένων μη γραμμικών συναρτήσεων και 2) στην περίπτωση που απαιτούνται και άλλες αριθμητικές πράξεις, για παράδειγμα όταν οι πολλαπλασιασμοί ακολουθούν την έξοδο των στοχαστικών μηχανών πεπερασμένων καταστάσεων, η δυαδική έξοδος πρέπει να αναγεννηθεί ως στοχαστική ακολουθία προκειμένου να να χρησιμοποιηθούν λογικές πύλες.

Με κίνητρο τους ανωτέρω περιορισμούς του APC, η παρούσα εργασία εισάγει μια αρχιτεκτονική αθροιστή που χρησιμοποιεί έναν διαμορφωτή σίγμα-δέλτα πρώτης τάξης (SDM). Ο προτεινόμενος αθροιστής στοχαστικού υπολογισμού σίγμα-δέλτα (SCSD) αθροίζει τα ψηφία των ακολουθιών εισόδου σε ένα δίαυλο δεδομένων και στη συνέχεια χρησιμοποιεί ένα εσωτερικό σχήμα μετατροπής εύρους δεδομένων ώστε να εκμεταλλευτεί την ιδιότητα του Σ-Δ να μετατρέπει ένα σήμα υψηλής ανάλυσης σε σήμα του ενός ψηφίου. Προσφέρει τα ακόλουθα πλεονεκτήματα: 1) λειτουργεί με ανεξάρτητες εισόδους, 2) η πρόσθεση γίνεται αιτιοκρατικά χωρίς επιπρόσθετες πηγές τυχαίων αριθμών, 3) επιτυγχάνει γρήγορη σύγκλιση με μικρά μήκη ακολουθίας εισόδου, 4) επιτρέπει να γίνουν αποτελεσματικά αλυσιδωτές πράξεις με τα υπάρχοντα αριθμητικά κυκλώματα και 5) επιτρέπει τη χρήση οποιασδήποτε στοχαστικής μηχανής πεπερασμένων καταστάσεων διευρύνοντας έτσι τη σχεδίαση νευρωνικών δικτύων και μη στο χώρο σχεδίασης του στοχαστικού υπολογισμού.

Όσο αναφορά τις μη γραμμικές συναρτήσεις που χρησιμοποιούνται στον στοχαστικό υπολογισμό, μεταξύ αρκετών όπως της υπερβολικής εφαπτομένης, του γραμμικού κέρδους, την εκθετική, του μεγίστου και του ελαχίστου, οι τελευταίες δύο είναι οι πιο δημοφιλείς δεδομένης της χρήσης τους στο στρώμα μέγιστης συγκέντρωσης (max pooling layers) στα νευρωνικά δίκτυα και στα φίλτρα μείωσης θορύβου.

Μία πρώτη προσέγγιση για την υλοποίηση του μεγίστου και ελαχίστου, χρησιμοποιεί πολυπλέκτες και τη συνάρτηση υπερβολικής εφαπτομένης υλοποιημένης ως μηχανή πεπερασμένων καταστάσεων. Ωστόσο, ένας από τους δύο πολυπλέκτες χρησιμοποιεί μία επιπλέον γεννήτρια παραγωγής στοχαστικών ακολουθιών για το το σήμα επιλογής του πολυπλέκτη, αυξάνοντας έτσι τις απαιτήσεις του υλικού. Έχοντας ως βάση την προηγούμενη αρχή λειτουργίας, άλλη προσέγγιση αντικαθιστά τον μετατροπέα δυαδικού σε στοχαστικό με μία λογική πύλη XOR για να μειώσει την επιβάρυνση υλικού, διατηρώντας την υπόλοιπη δομή επεξεργασίας. Μία παρόμοια προσέγγιση, αντικαθιστά τη μηχανή πεπερασμένων καταστάσεων με καταχωρητή μετατόπισης (shift register) για την αποθήκευση των λογικών μονάδων από τη μία εκ των δύο εισόδων του και το λιγότερο σημαντικό bit (Least-Significant Bit - LSB) παράγει λογική μονάδα μονο αν έχει κορεστεί μέχρι αυτό. Ένα βασικό μειονέκτημα όμως είναι το ακριβές μέγεθος του καταχωρητή μετατόπισης, το οποίο αν δεν επιλεγεί με σωστά, η υπολογιστική ακρίβεια της εξόδου μειώνεται δραματικά.

Παρακινούμενοι από τις σχεδιαστικές προκλήσεις των προηγούμενων μεθόδων σε συνδυασμό με την ανάγκη για υπολογισμούς με χαμηλή καθυστέρηση στο στοχαστικό υπολογισμό, στην παρούσα εργασία προτείνονται δύο διαφορετικές προσεγγίσεις για την υλοποίηση του μεγίστου/ελαχίστου. Σε αντίθεση με άλλες προσεγγίσεις, οι προτεινόμενες αρχιτεκτονικές χρησιμοποιούν έναν συσσωρευτή για την απευθείας αποθήκευση των προσημασμένων διαφορών των ψηφίων μεταξύ των δύο ακολουθιών εισόδου τους, χωρίς πρόσθετες πηγές παραγωγής τυχαίων αριθμών, καθιστώντας τη λειτουργία τους αιτιοκρατική. Αυτό έχει ως αποτέλεσμα τη μείωση της καθυστέρησης και ταυτόχρονα την επίτευξη υπολογισμών υψηλής ακρίβειας με τη χρήση μικρού μήκους ακολουθιών εισόδου.

Σχετικά με τη χρήση στοχαστικών μηχανών πεπερασμένης κατάστασης για την υλοποίηση μη γραμμικών συναρτήσεων, για να είναι εφικτή η προσέγγιση τους, θα πρέπει να ικανοποιούν ταυτόχρονα ορισμένες συνθήκες. Συγκεκριμένα, θα πρέπει να αποτελούνται από έναν πεπερασμένο αριθμό καταστάσεων με την πρώτη και την τελευταία να είναι κορεσμένες, δηλαδή να μην μπορούν να ξεπεραστούν, θα πρέπει οι μεταβάσεις εντός των καταστάσεών τους να οδηγούνται από ακολουθίες εισόδου, με στοχαστικές ιδιότητες και πεπερασμένο μήκος και τέλος όλες οι καταστάσεις να επικοινωνούν μεταξύ τους. Οι προηγούμενες ιδιότητες περιγραφούν τις μηχανές πεπερασμένων καταστάσεων ως εργοδικές αλυσίδες Markov, επιτρέποντας τη σύνθεση συναρτήσεων μέσω της εκτέλεσης λογικών πράξεων μεταξύ των πιθανοτήτων των καταστάσεων.

Παρά τα πολλαπλά πλεονεκτήματα τους, οι στοχαστικές μηχανές πεπερασμένων καταστάσεων έχουν και τις δικές τους αδυναμίες. Η κυριότερη από αυτές, είναι η εισαγωγή συσχετίσεων μεταξύ των ψηφίων της ακολουθίας εξόδου, γεγονός που είναι λογικό, δεδομένων των στοιχείων μνήμης που απαιτούνται για την υλοποίηση των μηχανών κατάστασης. Στην πρώτη προσέγγιση που έγινε για τη μοντελοποίησή τους, ο υπολογισμός της αυτοσυσχέτισης (autocorrelation) της εξόδου καθώς και της μέσς τιμής, επαληθεύτηκε με αριθμητικά πειράματα. Σε μία δεύτερη προσέγγιση, χρησιμοποίηθηκαν αλυσίδες Markov για να αποδείξουν την αρχή λειτουργίας αρκετών μη γραμμικών συναρτήσεων, χωρίς ωστόσο να διερευνώνται οι στατιστικές ιδιότητες της εξόδου. Στο γενικό πλαίσιο της συσχέτισης, αυτή προσεγγίζεται κυρίως από την οπτική γωνία της ακολουθίας εισόδου, στην οποία οι μετατροπείς δυαδικών σε στοχαστικούς αριθμούς μοιράζονται την πηγή τυχαίων αριθμών τους με σκοπό τη δημιουργία ακολουθιών εισόδου με μέγιστη επικάλυψη μεταξύ των θέσεων των λογικών τους μονάδων. Η τεχνική αυτή ενώ επιτρέπει

την αποδοτική υλοποίηση ορισμένων αριθμητικών πράξεων, όπως για παράδειγμα την αφαίρεση, είναι προσαρμοσμένη στην ίδια την πράξη.

Με αφορμή τις ανάγκες για βαθιά κατανόηση των στατιστικών ιδιοτήτων των εξόδων των στοχαστικών μηχανών πεπερασμένων καταστάσεων, στην παρούσα εργασία εισάγεται ένα μαθηματικό πλαίσιο για τη λεπτομερή ανάλυση και εξαγωγή τους, βασισμένο σε αλυσίδες Markov. Πρόκειται για μια γενική μεθοδολογία, υπό την έννοια ότι μπορεί να εφαρμοστεί σε οποιοδήποτε στοχαστική μηχανή πεπερασμένων καταστάσεων εκφρασμένη υπό τη μορφή Moore και μοντελοποιημένη ως αλυσίδα Markov. Η κύρια συνεισφορά της εργασίας είναι ο αναλυτικός υπολογισμός με τη χρήση κλειστού τύπου εκφράσεων των ακόλουθων στατιστικών ιδιοτήτων που περιλαμβάνουν: την αναμενόμενη τιμή και τη μέση τιμή της εξόδου, την αυτοσυσχέτιση και την αυτοσυνδιακύμανση της εξόδου, την ετεροσυσχέτιση και η ετεροσυνδιακύμανση της εξόδου με τις εισόδους, της διακύμανσης και της τυπική απόκλισης του μέσου όρου της εξόδου, του μέσου τετραγωνικού σφάλματος του μέσου όρου της εξόδου, την πιθανότητα υπερχείλισης και υποχείλισης στις καταστάσεις κορεσμού και τέλος τον αναμενόμενο αριθμό βημάτων πριν από τις υπερχειλίσεις και τις υποχειλίσεις, ο οποίος κατά συνέπεια θέτει τις κατευθυντήριες γραμμές για την επιλογή του αριθμού των καταστάσεων που μειώνουν τα λανθασμένα ψηφία που προέρχονται από τις υπερχειλίσεις και τις υποχειλίσεις.

Για την αξιολόγηση της επίδοσης των προτεινόμενων αρχιτεκτονικών, γίνεται η σύγκρισή τους με υπάρχουσες προσεγγίσεις στη βιβλιογραφία του στοχαστικού υπολογισμού. Ειδικότερα, οι προτεινόμενες αρχιτεκτονικές συγκρίνονται σε υπολογιστή ακρίβεια χρησιμοποιώντας μετρικές σφαλμάτων για διαφορετικά μήκη ακολουθίας εισόδου καθώς και σε αξιοποίηση πόρων υλικού, συμπεριλαμβανομένων του χώρου που καταλαμβάνουν τα κυκλώματα, κατανάλωση ενέργειας και ισχύος σύμφωνα με την μέγιστη δυνατή συχνότητα λειτουργίας. Τα αποτελέσματα έδειξαν πως δεδομένης της αξιοποίησης των εσωτερικών καταχωρητών και μετρητών, οι προτεινόμενες αρχιτεκτονικές επιτυγχάνουν μεγαλύτερη υπολογιστική ακρίβεια με μικρά μήκη ακολουθιών εισόδου, αυξάνοντας ελάχιστα τους συνολικούς πόρους. Αξίζει να σημειωθεί πως λαμβάνοντας υπόψιν την αντιστάθμιση καθυστέρησης - αύξησης υπολογιστικής ακρίβειας, στην πραγματικότητα η συνολική κατανάλωση ενέργειας για τις προτεινόμενες αρχιτεκτονικές είναι παρόμοια ή και μικρότερη από τις υπάρχουσες προσεγγίσεις, αφού είναι περιττή η χρήση μεγάλου μήκους ακολουθιών εισόδου.

Σε επίπεδο εφαρμογής, οι προτεινόμενες αρχιτεκτονικές αξιοποιήθηκαν για την υλοποίηση διαφόρων διεργασιών που εκτελούνται από ψηφιακούς επεξεργαστές. Σε αυτές συμπεριλαμβάνονται: δομική μονάδα συνέλιξης, φίλτρα χωρικής ενίσχυσης (spatial enhancement filters) και τέλος νευρωνικό δίκτυο MLP. Σε επίπεδο αξιοποίησης υλικού, οι δομικές μονάδες συγκρίθηκαν με τις συμβατές δυαδικές υλοποιήσεις σε χώρο που καταλαμβάνουν τα κυκλώματα, κατανάλωση ενέργειας και ισχύος σύμφωνα με την μέγιστη δυνατή συχνότητα λειτουργίας.

Ξεκινώντας με τον προτεινόμενο αθροιστή, για να γίνει αισθητή η αποτελεσματικότητα του σε αλυσιδωτούς υπολογισμούς δεδομένης της μη κλιμακωτής φύσης του, χρησιμοποιήθηκε παράλληλα με λογικές πύλες AND για την υλοποίηση δομικής μονάδας που εκτελεί την πράξη της συνέλιξης. Έπειτα, η δομική μονάδα χρησιμοποιήθηκε ως μάσκα για το φιλτράρισμα εικόνας με σκοπό την εξομαλύνση των εικονοστοιχείων (pixels) της έτσι ώστε να γίνει μείωση του θορύβου της εικόνας. Τα αποτελέσματα στην αξιολόγηση της ποιότητας εικόνας έδειξαν πως ο προτεινόμενος αθροιστής επιτυγχάνει αποδεκτές τιμές,

οι οποίες είναι εξαιρετικά βελτιωμένες συγκριτικά με τους κλιμακωτούς αθροιστές. Σε επίπεδο υλικού, ο χώρος που καταλαμβάνεται είναι εξαιρετικά μικρότερος από αυτόν του συμβατού δυαδικού, ωστόσο, η κατανάλωση ενέργειας ανέρχεται σε μέτριες τιμές, που είναι αναμενόμενο δεδομένης της φύσης του στοχαστικού υπολογισμού

Συνεχίζοντας με τον προτεινόμενο στοχαστικό αφαιρέτη, η αξιοποίηση του γίνεται αισθητή στην υλοποίηση φίλτρου ενίσχυσης ευκρίνειας εικόνας (image sharpening filter). Αναλυτικότερα, το φίλτρο ενίσχυσης ευκρίνειας εικόνας διαχωρίζεται σε τρεις υπολογισμούς που περιλαμβάνουν το φιλτράρισμα, την εξαγωγή των λεπτομερειών της εικόνας και τέλος την ενίσχυσή της. Από τα παραπάνω, το φιλτράρισμα γίνεται μέσω συνέλιξης χρησιμοποιώντας τη δομική μονάδα που υλοποιείται μέσω του στοχαστικού αθροιστή, ο οποίος χρησιμοποιείται και για την τελική ενίσχυση. Η εξαγωγή των λεπτομερειών απαιτεί την αφαίρεση της φιλτραρισμένης εικόνας από την αρχική, μία διαδικασία που είναι κατάλληλη για τον προτεινόμενο στοχαστικό αφαιρέτη καθώς οι ήδη υπάρχουσες προσεγγίσεις αδυνατούν στην υλοποίηση μη κλιμακωτής αφαίρεσης. Αναφορικά με την αξιολόγηση της ποιότητας εικόνας, τα αποτελέσματα ήταν σχεδόν βέλτιστα, ενώ σε επίπεδο αξιοποίηση πόρων η μείωση του χώρου που καταλαμβάνει το κύκλωμα συγκριτικά με τη συμβατή δυαδική υλοποίηση ήταν αισθητή, ωστόσο, η κατανάλωση ενέργειας ανήλθε σε μέτριες τιμές.

Οι αρχιτεκτονικές μεγίστου/ελαχίστου, ήταν κατάλληλες για την υλοποίηση φίλτρου διαμέσου (median filter), το οποίο χρησιμοποιείται για τη βελτίωση εικόνας που έχει υποστεί αλλοίωση στα εικονοστοιχεία της, όπως για παράδειγμα αλλοίωση λόγω θορύβου. Η δομή της διάταξης του φίλτρου διαμέσου, βασίζεται σε αλγόριθμο αποδοτικής ταξινόμησης, δηλαδή συγκρίσεις μεγέθους μεταξύ των εισόδων. Συγκριτικά με το φίλτρο εξομάλυνσης, το φίλτρο διαμέσου έχει την ιδιότητα να διατηρεί τις ακμές της εικόνας (edge preservation), καθιστόντας το κατάλληλο για στάδια προ-επεξεργασίας εικόνας πριν την ανίχνευση ακμών (edge detection). Τα αποτελέσματα σε επίπεδο υπολογιστικής ακρίβειας έδείξαν πως και οι δύο προτεινόμενες αρχιτεκτονικές μεγίστου/ελαχίστου ήταν ικανοποιητικές για φιλτράρισμα εικόνας με θόρυβο. Σχετικά με την αξιοποίηση των πόρων, η δεύτερη αρχιτεκτονική μεγίστου/ελαχίστου αξιοποιεί παραπάνω κυκλωματικό χώρο από την πρώτη, όμως, και οι δύο καταλάμβαναν σχεδόν το μισό από αυτό της συμβατής δυαδικής υλοποίησης. Όπως και με τις άλλες δύο εφαρμογές, η κατανάλωση ενέργειας ανήλθε σε μέτριες τιμές συγκριτικά με τη συμβατή δυαδική, δεδομένου του συνολικό μήκος ακουλουθιών που επεξεργάζονται.

Η δεύτερη προτεινόμενη αρχιτεκτονική του μεγίστου, χρησιμοποιήθηκε για την υλοποίηση φίλτρου μέγιστης συγκέντρωσης (max pooling). Η λειτουργία του βασίζεται στην υποδειγματοληψία (undersampling) εικόνας καθώς μειώνει τη διάσταση της, ενώ αποτελεί αναπόσπαστο κομμάτι στα σύγχρονα νευρωνικά δίκτυα δεδομένου ότι επιτρέπει την εξαγωγή των σημαντικότερων χαρακτηριστικών της εικόνας εισόδου. Τα αποτελέσματα σε επίπεδο υπολογιστικής ακρίβειας έδειξαν πως η υποδειγματοληψία της εικόνας πραγματοποιείται με το βέλτιστο δυνατό τρόπο, το οποίο υποστηρίζεται από τις μετρικές που λαμβάνονται υπόψιν. Όσο αναφορά την αξιοποίηση των πόρων του υλικού, συγκριτικά με την δυαδική υλοποίηση παρατηρείται πως ο χώρος που καταλαμβάνει το κύκλωμα μειώνεται επαρκώς.

Τέλος, ο αθροιστής SCSD χρησιμοποιήθηκε μαζί με την πρώτη αρχιτεκτονική του μεγίστου για την υλοποίηση στοχαστικού νευρώνα, ο οποίος αποτέλεσε τη βάση για την υλοποίηση ενός MLP. Τα αποτελέσματα σε επίπεδο κατηγοριοποίησης για δύο διαφορετικές αρχιτεκτονικές δικτύου MLP σε ρεαλιστικό

σύνολο δεδομένων, έδειξαν πως η ακρίβεια που επιτυγχάνεται υπερβαίνει αυτήν που επιτυγχάνεται με την τυπική δυαδική αναπαράσταση αριθμών των οκτώ και των δεκαέξι ψηφίων. Επιπλέον, συγκριτικά με αυτές τις αναπαραστάσεις, μειώνεται δραματικά χώρος που καταλαμβάνει ο εκάστοτε νευρώνας. Συγκρίσεις με ήδη υπάρχοντα MLP στο πεδίο του στοχαστικού υπολογισμού, ανέδειξαν τη δυνατότητα για επεξεργασία με μικρά μήκη ακολουθιών εισόδου καθώς και την ευελιξία στη συνολική σχεδίαση.

Acknowledgements

I am extremely grateful to my supervisor Professor Paul-Peter Sotiriadis for his invaluable help and support in making this dissertation possible. His teachings and immense scientific knowledge were of vital importance as they effectively guided me towards becoming a devoted research scientist. Having worked under his supervision is honorable for me to the greatest degree and for that I am deeply thankful.

I would like to express my appreciation to the honorable members of my advisory committee Professor Emeritus Kiamal Pekmestzi and Professor Athanasios Panagopoulos for their excellent feedback and support throughout the years of my PhD studies. My sincere thanks are extended to the honorable members of the examination committee Professor Panayiotis Psarrakos, Professor Theodora Varvarigou, Associate Professor Anastasios Doulamis and Professor Nikolaos Doulamis for their participation and their instructive comments. My special thanks to Professor Panayiotis Psarrakos for his important teachings in Linear Algebra.

I would like to thank from the bottom of my heart my fellow PhD students Baxevanakis Dimitrios, Konstantinos Papafotis, Konstantinos Touloupas, Konstantinos Asimakopoulos, Costas Oustoglou, Charis Basetas, Christos Dimas, Ioannis Georgakopoulos, Vassilis Alimisis, Neoclis Hadjigeorgiou as well as the rest NTUA Circuits & Systems Group members for the moments that we shared and the friendships we developed, hopefully lasting in the years to come. A special thanks goes to my friend Dr. Nikolaos Voudoukis for his life advices and to Konstantinos Touloupas and to Konstantinos Papafotis for their invaluable time they spent in carefully reviewing and improving my publications.

I am forever grateful to my teachers and mentors Professors Dimitrios Nikolopoulos and Panayiotis Yannakopoulos for introducing me to the scientific way of thinking, for their invaluable teachings and for supporting me throughout the years of my studies, in every possible way.

Many thanks towards the Hellenic Foundation for Research and Innovation (HFRI) for supporting financially this dissertation.

My wholehearted thanks go to my parents Eirini and Alekos Temenos, my aunt Despoina Driva, my brother Tasos and my friends for their continuous support, understanding, love and encouragement throughout the years of my studies. Finally yet importantly, my deepest gratitude goes to Mr. Panagiotis Sotiriadis, a generous and kind soul, for inspiring me and for his tremendous effort in helping me to further continue my studies to the advanced degrees. I will remember him forever.

Abbreviations

SC	Stochastic Computing
SNG	Stochastic Number Generator
LFSR	Linear-Feedback Shift Register
MC	Markov Chain
FSM	Finite-State Machine
SFSM	Stochastic Finite-State Machine
DSP	Digital Signal Processing/Processor
FPGA	Field-Programmable Gate Array
IC	Integrated Circuit
FxP	Fixed Point Arithmetic
FP	Floating Point Arithmetic
SCPB	Stochastic Computing Processing Block
MAE	Mean Absolute Error
MSE	Mean Squared Error
PSNR	Peak Signal-to-Noise Ratio
SSIM	Structural Similarity Index Measure

Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and Scope

Efficient realization of digital systems in Integrated Circuits (ICs) and Field Programmable Gate Arrays (FPGAs) is of utter importance given the accelerated growth of emerging applications [75, 28, 22, 29]. The typical binary arithmetic representations used for their implementation, namely the Fixed-Point (FxP) and Floating Point (FP), can be hardware-demanding for the modern Digital Signal Processors (DSPs), especially when massive parallelization is necessary [22, 50, 12]. This is further intensified when non-linear functions are required in the processing, for instance the exponential and the hyperbolic tangent [50, 80, 76, 43]. To this end, unconventional computing paradigms are under extensive exploration [64, 10, 29, 36, 67], with Stochastic Computing being an effective approach among many [40, 28, 66, 29].

Stochastic Computing (SC) deviates from the standard binary arithmetic and its processing, as it encodes the value of binary numbers in the form of finite-length stochastic sequences of logic 0s and 1s [23, 7, 9]. Therefore, its single-bit processing allows for the fundamental arithmetic operations and highly-complex functions to be realized using a few logic gates and standard cells [69, 9], thereby reducing dramatically the hardware area requirements compared to the traditional binary arithmetic [50]. An inherent property of SC is that of the robustness on soft-errors, meaning that occurring bit-flips are not detrimental (up to a certain degree) for the reliability of the signals' information [23]. Beyond its strong points, SC requires computational cycles to increase the calculations' accuracy, impacting on the energy being dissipated [20, 65, 46, 66]. Hence, to make the best of it, achieving low latency combined with increased computational accuracy, is of primary design concern in SC [88, 34, 66].

The properties and advantages of SC favour applications that combine massive parallelism needs, area constraints and tolerance to small deviations from the exact calculations. These applications include Multi-Layer Perceptrons (MLPs) [48, 49, 37], Convolutional Neural Networks (CNN) [12, 74, 91] and others [62, 51, 16, 35, 52] in the field of Deep Learning, Support Vector Machines (SVMs) [55, 56, 30] in the field of Machine Learning (ML) and noise reduction, averaging, smoothing, sharpening and other spatial enhancement filters [42, 43, 8, 79, 81] in the field of Image Processing. However, SC is not limited to the previous fields; it has been successfully applied in soft-filtering, i.e. Finite Impulse Response (FIR) [33, 2, 11, 82, 85, 1] and Infinite Impulse Response (IIR) [73, 33, 53] filters, error correcting coding &

decoding [26, 7, 9, 27], polynomial solving [54, 63, 6, 3, 71, 86] and others [26, 7, 9, 90, 45, 44].

Essential operations performed in the DSP cores utilized by the aforementioned applications, rely mostly in multiply-and-add operations and non-linear functions [80]. With respect to the fundamental arithmetic operations, multiplication is the simplest in SC. According to the SC number representation used, a single AND gate for positive-signed stochastic numbers or an XNOR gate for negative-signed stochastic numbers is used [23]. The addition and subtraction operations between two stochastic sequences should follow the probabilistic nature of SC, meaning that their result cannot exceed one or be less than minus one. For this reason, the operation of most adders [41, 84, 72, 18] and subtracters [4, 18, 54] is based upon the scaling of their result with a typical value of two. However, when multiple cascaded computations are required, scaled adders and subtracters do not favour them, especially when 1) other operations follow, for instance non-linear functions and 2) the number of adders and/or subtracters is not a multiple of two. On the other hand, existing non-scaling adders [72, 18, 90] impose design constraints as they do not follow the standard SC number representation formats.

Regarding the realization of non-linear functions, Stochastic FSMs (SFSMs) are employed for such purpose [15, 16]. They are known for their ability to approximate widely used functions such as the hyperbolic tangent (tanh) [15, 43], the exponential [15, 43], the linear gain [15, 43], the max & min [43, 89, 58, 39] and others [15, 43], with the max & min being the most popular ones due to their presence in max pooling operations and in median filtering [50, 81]. Despite their importance, SFSMs have only been used in complement with Markov Chains (MCs) to formally prove the non-linear functions' principle of operation [15, 43], without further investigating their statistical properties nor their impact on the calculations [78, 39, 4, 57, 17]. Increased correlation among the bits of the SFSMs' output sequences may result in calculation errors in the operations following, for instance a potential multiplication of the output with itself, thus degrading the overall accuracy[78, 13, 59, 14].

This dissertation presents novel architectures realizing essential arithmetic operations and non-linear functions in Stochastic Computing, including a non-scaling adder, non-scaling a subtracter, two different max and min architectures and a multi-input single-bit output adder. Their main advantage they offer, is the improvement on the SC's accuracy-latency trade-off, which stems from their ability to combine highly-accurate computations with short sequence lengths. The above properties are demonstrated with an in-depth analysis using SFSMs and MCs.

The operation principle of the architectures is analysed using SFSMs and MC modeling which allows for a better understanding of their long-term stochastic dynamics and the verification of their proper operation. The MC modeling is further extended to a general methodology for the analytical derivation of the SFSM' statistical properties, including their expected value, their variance and standard deviation, their correlation and covariance as well as the mean squared error. The methodology is accompanied by overflow/underflow MC modeling allowing to estimate the number of states that reduce bit-errors originating from overflow/underflow occurrence, setting the guidelines for the selection of the register's size.

For the evaluation of their performance, the architectures are compared extensively with existing ones in the SC literature in computational accuracy using standard error metrics and hardware resources, including area, power and energy consumption as well as in the benefits they introduce in the overall design

flow. The efficacy of the architectures is demonstrated with their use as building blocks in the realization of several DSP tasks, including convolution, noise reduction and image down-sampling filtering as well as Neural Networks. The results of the architectures' performance in computational accuracy and hardware resources are compared to those achieved using standard binary computing methods so as to highlight their advantages.

1.2 Thesis Outline

The present dissertation is divided in two parts, theoretical analysis and experimental results. In the first part including chapters 3 and 4, the operation principle of the proposed architectures using SFSMs and their modeling using MCs is introduced. In the second part including chapters 5 and 6, experimental results and applications realized using the proposed architectures are presented. Specifically, the rest of this dissertation is organized as follows.

In Chapter 2, the conversion of binary numbers into stochastic sequences, the two fundamental SC number representation formats, the sequences' properties and the notation used throughout the dissertation is provided. Moreover, the operation of the essential logic gates used in SC under different number representation formats is described.

In Chapter 3, the proposed architectures, namely the non-scaling adder, the non-scaling subtracter, the two max and min and the SCSD adder are introduced and their operation principle is described using SFSMs. Then, their modeling using MCs is shown and their expected value is derived analytically, used to prove their proper operation in the limiting case.

In Chapter 4, the MC modeling of Chapter 3 is extended to a general methodology for the analytical derivation of the SFSMs' statistical properties and the modeling of overflows/underflows. Two SFSMs selected from the SC literature are modeled using the proposed framework and their results are compared to those obtained from the numerical experiments.

In Chapter 5, the proposed architectures are compared extensively with existing ones in the SC literature in computational accuracy, hardware resources and their trade-offs in the overall SC design is discussed.

In Chapter 6, applications of standard DSP tasks realized using the proposed architectures are shown, while comparisons with the conventional binary computing methods in hardware resources and computational accuracy demonstrate their efficacy.

Finally, Chapter 7 concludes the present dissertation.

2 Stochastic Computing Principles

The Stochastic Number Generator (SNG), shown in Fig. 2.1 [9, 23], is the standard circuit converting a k-bit deterministic number into its stochastic $0, 1$ sequence representation, also referred as a ***stochastic number***. A pseudo-random number generator uniformly distributed in $\{0, 1, \ldots, 2^k - 1\}$ and typically implemented as a k-bit Linear-Feedback Shift Register (LFSR), generates on every clock cycle a k-bit random number which is compared with the deterministic number $B \in [0, 1]$. The bit generation is completed after $N = 2^k$ clock cycles and corresponds to the length of the sequence [9, 7, 20]. To convert the stochastic number back to its binary form, an up-counter of k-bits is used.

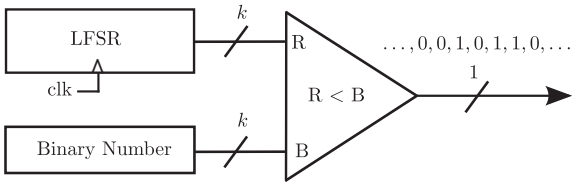


Figure 2.1: Stochastic Number Generator

The N-bit output sequence generated by the SNG, i.e. $\{X_n\}, n = 1, 2, ..., N$, with n being the current time index (or clock cycle), is independent and identically distributed (i.i.d.). It represents a non-negative number in $[0, 1]$ and is known as ***unipolar format*** in SC. The probability of the stochastic number is defined as $X \triangleq P_r(X_n = 1) = B/2^k$, which is the normalized value of B in k-bit representation and its mean is given as

$$\widetilde{X}_N = \frac{1}{N} \left(X_1 + X_2 + \cdots + X_N \right). \tag{2.1}$$

Negative numbers, known as ***bipolar format***, can also be represented using the transformation $X \mapsto 2X - 1$, expanding the range of the stochastic number to $[-1, 1]$ [23]. For both stochastic number formats, the length of the sequence N is directly associated with the accuracy of the representation, which increases at the cost of additional clock cycles and is considered as SC's essential design trade-off.

Fundamental mathematical operations are supported within the context of SC and can be realized simply by logic gates according to the format used [69, 82]. To proceed with the analysis of the most

important logic gates used in essential operations, we assume that inputs $\{X_n\}$, $\{Y_n\}$ are stochastic sequences generated by different SNGs and that $\{Z_n\}$ is the result of their operation.

- NOT Gate: The NOT gate in unipolar format, $Z_n = \text{NOT}(X_n)$, complements the probability of the input,

$$Z = P(Z_n = 1) = P(X_n = 0) = 1 - P(X_n = 1) = 1 - X, \tag{2.2}$$

  whereas in the bipolar format, it operates as a sign inverter.

$$Z = P(Z_n = 1) = P(X_n = 0) = 1 - P(X_n = 1) = -X. \tag{2.3}$$

- AND Gate: The AND gate in unipolar format, $Z_n = \text{AND}(X_n, Y_n)$, performs multiplication.

$$Z = P(Z_n = 1) = P(X_n = 1, Y_n = 1) = P(X_n = 1)P(Y_n = 1) = XY. \tag{2.4}$$

- XNOR Gate: The XNOR gate in bipolar format, $Z_n = \text{XNOR}(X_n, Y_n)$, performs multiplication.

$$\begin{aligned} Z &= P(Z_n = 1) = P(X_n = 1, Y_n = 1) + P(X_n = 0, Y_n = 0) \\ &= 2P(X_n = 1)P(Y_n = 1) - P(X_n = 1) - P(Y_n = 1) + 1 \\ &= XY. \end{aligned} \tag{2.5}$$

- Multiplexer: Assuming an an i.i.d. control sequence $\{C_n\}$, the multiplexer (MUX), $Z_n = \text{MUX}(X_n, Y_n; C_n)$, is the standard way to perform scaled addition between two stochastic numbers, regardless of the format used, and is given as

$$\begin{aligned} Z &= P(Z_n = 1) = P(X_n = 1, C_n = 1) + P(Y_n = 1, C_n = 0) \\ &= P(X_n = 1)P(C_n = 1) + P(Y_n = 1)P(C_n = 0) \\ &= XC + Y\overline{C}. \end{aligned} \tag{2.6}$$

  Furthermore, if (and only) $P(C_n = 1) = 1/2$, the MUX operates as a scaling adder, i.e.,

$$Z = P(Z_n = 1) = \frac{P(X_n = 1) + P(Y_n = 1)}{2} = \frac{X + Y}{2}. \tag{2.7}$$

  Stochastic subtraction, on the other hand, can only be realized in the bipolar format, using a NOT gate in one of the two inputs as

$$Z = P(Z_n = 1) = \frac{P(X_n = 1) + P(Y_n = 0)}{2} = \frac{X - Y}{2}. \tag{2.8}$$

It is important to mention here that the logic gates OR,NOR and XOR do not realize a specific operation in neither SC number representation formats [69] and are only used complementary with other logic gates.

Part I

Theoretical Analysis

3 Stochastic Computing Architectures

In this chapter, the following proposed SC architectures are presented: 1) a non-scaling adder [80], 2) a non-scaling subtracter [80], 3) a MAX [81], 4) a MIN [81], 5) a compact MAX & MIN [76], capable of realizing both functions and 6) a multi-input single-bit output adder. [1]. The architectures' principle operation is described using Stochastic Finite-State Machines, while their stochastic behavior is modeled using Markov Chains, allowing for the derivation of the expected value of their output and their verification of proper operation.

3.1 Non-Scaling Adder and Subtracter Architectures

An essential operation performed in the SC-based DSP cores, is that of the multiply-and-add. The multiplication part, is implemented using a single AND or XNOR gate according to the stochastic number representation used. Their addition part in SC though, is typically realized by the MUX which requires an additional random number source for its select signal, besides its inputs. However, the random number source by itself is a large block compared to the SC elements, occupying most of the design's area [65]. In addition, the adder's output is typically scaled by $1/2$ meaning that for a given sequence length the resolution has dropped by 2. This makes the MUX less attractive for cascaded computations and blocks that rely on exact calculations. The same apply for the MUX implementation of the subtracter.

To address the former issues focusing on computational and design efficiency, several adders [41, 84, 72, 18] and subtracters [4, 18, 54] have been published and explored [68] within the context of SC. The adder in [41], is based on the multiplexer's scaling principle, but avoids the extra random number source by using a single T Flip-Flop, increasing also its accuracy. A similar (scaling) approach is presented in [84], but instead of a T Flip-Flop, it employs a two-state FSM to further increase its accuracy.

The semi-stochastic approach explored in [83], uses the parallel input adder originally proposed in [68], which provides computations in binary format that do not always favor next stage SC-based computational blocks, for instance non-linear functions. The non-scaling adder in [72] is based on a two-line representation of a stochastic number carrying its information in two sequences; one for its sign and one for its magnitude. Although it is a promising approach in application level [90], the two-line encoding imposes system design constraints as it requires from other operations, e.g. multipliers, to follow this principle as well. Furthermore, the size of its counting unit is only estimated empirically [90]. Similarly to the previous adder, the adder (and subtracter with one inverted input) presented in [18], encodes a stochastic number by using the ratio of logic ones and zeros between its input sequences. Yet, its unique representation is incompatible with standard SC formats, while the generation of two sequences for a single stochastic number, influences the overall hardware utilization. Regarding stochastic subtracters, the method in [4, 39, 5] correlates the input sequences. This, however, requires caution since SC elements are prone to errors caused by correlated inputs [65]. Moreover, if the subtraction is an intermediate operation, regenerating correlated inputs is necessary. Another technique presented in [54] applies iterative logic units to enhance the accuracy of an XNOR gate with one of its inputs inverted. As expected, it trades hardware resources and latency for accuracy, both depending on the number of stages used.

All the above methods, trade circuit run-time and/or hardware area for accuracy. In addition, certain of them introduce constraints that reduce the flexibility on the SC design space. Motivated by the aforementioned and to achieve the best of both worlds, we propose non-scaling adder and subtracter architectures for SC. They offer the following advantages: 1) They do not require any random number source, 2) They do not scale the output result, 3) They operate with independent and identically distributed input sequences (i.e., no specially correlated inputs required), 4) They are compatible with standard SC formats and 5) They are fast-converging, achieving high accuracy with short sequences lengths.

3.1.1 Non-Scaling Adder Architecture

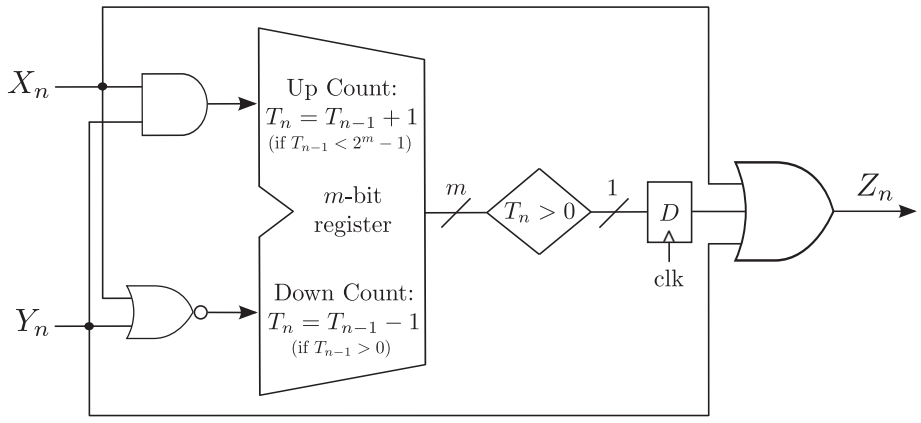


Figure 3.1: Proposed stochastic adder Architecture. T_n is the m-bit register's state, updated according to 3.1.

The proposed adder architecture is shown in Fig. 3.1. If $\mathrm{OR}(X_n, Y_n) = 1$ then the output is $Z_n = 1$.

In the case of $X_n = Y_n = 1$, a 1 is also stored and carried in the register (up-count by 1) in order to be outputted in the first future clock cycle n', i.e., $Z_{n'} = 1$, for which $X_{n'} = Y_{n'} = 0$. Moreover, when $X_n = Y_n = 0$, the register is down-counted by 1 if it had a positive prior value. The procedure of storing 1s, when $X_n = Y_n = 1$, and carrying them until they can be outputted compensates for the inability of the single-bit output to accommodate instantaneous value of more than 1.

The above are captured in the schematic of Fig. 3.1, where the register is m-bit with current state T_n in the set $\mathcal{T}_R \triangleq \{0, 1, 2, ..., M-1\}$, where $M = 2^m$. Note that T_n equals the number of accumulated logic 1s "owned" to the output, with initial value $T_0 = 0$ when the operation starts. From Fig. 3.1 one can conclude that the state of the adder evolves according to the following iteration

$$T_n = \min\Big\{T_{n-1} + X_n Y_n - \big(T_{n-1} > 0\big)\overline{X}_n\overline{Y}_n, M-1\Big\} \tag{3.1}$$

where $\overline{X}_n = \text{NOT}(X_n) = 1 - X_n$ and similarly for $\overline{Y}_n$.

Although the proposed adder is designed to process stochastic sequences, its behavior is deterministic. As seen in Fig. 3.1, the output is a deterministic function of the inputs without any additional randomization which could increase uncertainty and degrade precision. Specifically, the adder's output precision is determined by the length, N, of the input sequences, their stochastic properties and the register's size, m.

3.1.1.1 Markov Chain Modeling

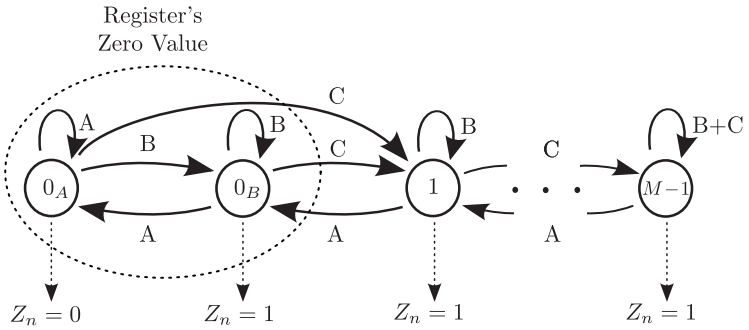


Figure 3.2: Markov Chain model of the proposed stochastic adder. The register's zero state, is represented by two states in the model, 0_A and 0_B. Transition probabilities A, B and C are given by (3.3)

The operation of the stochastic adder architecture is modeled by the Markov Chain (MC) in Fig. 3.2. To explain its derivation we note first that we assign two states 0_A and 0_B to the zero value of the register, whereas states 1 to $M-1$ represent the corresponding values of the register. Therefore, the MC state S_n can take the $M+1$ values in the set

$$\mathcal{S} \triangleq \{0_A, 0_B, 1, 2, ..., M-1\}. \tag{3.2}$$

Although using two zero states may appear confusing, it simplifies the analysis significantly because it allows us to relate the output value, Z_n, to the state only (i.e. the output is a function of the MC state and

not of the inputs X_n and Y_n).

Let the MC's state be S_{n-1}. Then the transition to the next state S_n and the output Z_n are determined according to the following transition probabilities

$$\begin{aligned} A =& P_r(X_n = 0)P_r(Y_n = 0) \\ B =& P_r(X_n = 1) + P_r(Y_n = 1) - 2P_r(X_n = 1)P_r(Y_n = 1) \\ C =& P_r(X_n = 1)P_r(Y_n = 1). \end{aligned} \tag{3.3}$$

As seen in Fig. 3.2 there are three kinds of states: A) The two zero states 0_A and 0_B corresponding to register's zero state and also embedding information of the predecessor input-state pair; B) States 1 to $M-2$ capturing a sequential increase/decrease of the register's value, and C) State $M-1$ corresponding to the maximum value of the register which is also the overflow state in the case of $X_n = Y_n = 1$ with probability C.

To analyze the behavior of the MC, which captures that of the proposed stochastic adder, we proceed with standard definitions. The $(M+1) \times (M+1)$ transition probability matrix, with state ordering $(0_A, 0_B, 1, 2, ..., M-1)$, is defined as

$$H = \Big[P_r(S_{n+1} = s_b | S_n = s_a)\Big]_{s_a, s_b \in \mathcal{S}}$$

where the (s_a, s_b) entry of the matrix is the probability to transition to state s_b from state s_a. Matrix H is written as

$$H = \begin{bmatrix} A & B & C & \dots & \dots & 0 \\ A & B & C & \dots & \dots & 0 \\ 0 & A & B & C & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & 0 & A & B & C \\ 0 & \dots & \dots & 0 & A & B+C \end{bmatrix}. \tag{3.4}$$

The probability distribution vector of state S_n, defined as

$$p_n^T \triangleq \begin{bmatrix} P_r(S_n = 0_A) \\ P_r(S_n = 0_B) \\ P_r(S_n = 1) \\ \vdots \\ P_r(S_n = M-1) \end{bmatrix} \in [0,1]^{M+1} \tag{3.5}$$

can be expressed as

$$p_n = p_0 H^n \in [0,1]^{M+1}, \tag{3.6}$$

where

$$p_0 = \Big[1, 0, 0, ...0\Big] \in [0, 1]^{M+1} \tag{3.7}$$

is the initial distribution vector and represents the starting state of the register $S_0 = 0_A$.

3.1.1.2 Expected Output Value and Verification of Operation

We use the MC model equations from the previous subsection to derive the expected value of the adder's output. To this end we calculate first the expected value of the instantaneous output Z_n. Note that since Z_n depends only on the state S_n, it is zero if and only if $S_n = 0_A$. Therefore it is

$$\mathbb{E}[Z_n] = P_r(Z_n = 1) = P_r\left(S_n \in \mathcal{S} - \{0_A\}\right) = 1 - p_0 H^n e_1^T, \tag{3.8}$$

where we used (3.6) and $e_i = \Big[0, ...0, 1, 0, .., 0\Big] \in \mathbb{R}^{M+1}$ is the i-th normal vector. Then, the average value of the output N-bit sequence,

$$\widetilde{Z}_N = \frac{1}{N}\Big(Z_1 + Z_2 + \cdots + Z_N\Big) \tag{3.9}$$

has expected value

$$\mathbb{E}[\widetilde{Z}_N] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[Z_n] = 1 - \frac{1}{N}p_0\left(\sum_{n=1}^{N} H^n\right)e_1^T. \tag{3.10}$$

Both the expected value of $\{Z_n\}$ and its mean are essential in quantifying the model's accuracy given its inputs $\{X_n\}, \{Y_n\}$ and will be used to verify the operation of the architecture.

The operation of the proposed architecture as an adder is proven here. As above, for the IID input sequences we use notation $X \triangleq P_r(X_n = 1)$ and $Y \triangleq P_r(Y_n = 1)$. In addition, we assume that $0 < X, Y < 1$ implying $A, B, C > 0$, as defined in (3.3). Therefore, the main, first upper and first lower diagonals of matrix H in (3.4) are positive implying the following Lemma whose proof is straightforward.

Lemma 1. *All entries of H^{M-1} are positive, i.e., $H^{M-1} > 0$.*

The result of Lemma 1 implies that $(I + |H|)^{M-1} > 0$ which along with Theorem 1 from [31] below proves that H is irreducible.

Theorem 1. *Matrix H is irreducible if and only if $(I + |H|)^{M-1} > 0$, where I is the identity matrix.*

Moreover, since H is a stochastic matrix it's spectral radius is $\rho(H) = 1$ having 1 as an eigenvalue. Now consider vector $v \in \mathbb{R}^{M+1}$ such that

$$v^T = \theta\left[1, \frac{1-A}{A}, \frac{\rho}{A}, \frac{\rho^2}{A}, \ldots, \frac{\rho^{M-1}}{A}\right] \tag{3.11}$$

where we have set

$$\rho \triangleq \frac{C}{A} = \frac{XY}{(1-X)(1-Y)}, \tag{3.12}$$

$$\theta \triangleq A\frac{\rho-1}{\rho^M-1} \tag{3.13}$$

and $\underline{1} = [1, 1, ..., 1]^T \in \mathbb{R}^{M+1}$ is the column vector of ones.

It can be verified that v^T and $\underline{1}$ are left and right eigenvectors of H corresponding to eigenvalue 1, i.e. $v^T H = v^T$ and $H\underline{1} = \underline{1}$. Moreover it is $v^T\underline{1} = 1$. From Theorem 8.6.1 in [31] we get that

$$\lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} H^n = \underline{1}v^T, \tag{3.14}$$

noting that $\underline{1}v^T$ is a $(M+1)\times(M+1)$ rank-one matrix. From (3.10) and (3.14) we get $\lim_{N\to\infty}\mathbb{E}[\widetilde{Z}_N] = 1 - p_0\underline{1}v^T e_1^T$. Since $p_0\underline{1} = 1$ and $v^T e_1^T = \theta$ we get $\lim_{N\to\infty}\mathbb{E}[\widetilde{Z}_N] = 1-\theta$ and by replacing θ we have

$$\lim_{N\to\infty}\mathbb{E}[\widetilde{Z}_N] = 1 - A\frac{\rho-1}{\rho^M-1}. \tag{3.15}$$

We assume in addition that $X + Y < 1$ which along with $0<X, Y<1$ imply that $0 < \rho < 1$ and so $\lim_{M\to\infty}\rho^M = 0$. Therefore since $1 - A(1-\rho) = 1 + C - A = X + Y$ we get

$$\lim_{M\to\infty}\left(\lim_{N\to\infty}\mathbb{E}[\widetilde{Z}_N]\right) = X + Y \tag{3.16}$$

which proves the correct operation in the limiting case.

The result of (3.16) is valid for stochastic addition in unipolar format and it is extended directly to bipolar format via the transformation $Z \mapsto 2(Z-1)$, where $Z{=}X{+}Y$ as before.

3.1.2 Non-Scaling Subtracter Architecture

The proposed subtracter architecture is shown in Fig. 3.3. It is comprised of the proposed stochastic adder with inverted one input and its output. Therefore, the subtracter operates like the adder with inputs $\overline{X}_n$ and Y_n having probabilities $P_r(\overline{X}_n = 1) = 1 - X$ and $P_r(Y_n = 1) = Y$ respectively. The addition operation implies that $\widetilde{Z}_N \approx 1 - X + Y$ and the output inversion gives $\widetilde{C}_N = 1 - \widetilde{Z}_N \approx X - Y$. For the subtracter to operate appropriately it must be $X \geq Y$.

Similarly to the stochastic adder, the counter's value T_n (with initial value $T_0 = 0$) belongs to $\mathcal{T}_R \triangleq \{0, 1, 2, ..., M-1\}$, where $= 2^m$ and m is the register's size. The state T_n indicates the number of logic 1s "owned" to the addition $(1 - X_n) + Y_n$ and evolves according to

$$T_n = \min\Big\{T_{n-1} + \overline{X}_n Y_n - \big(T_{n-1} > 0\big)X_n\overline{Y}_n, M-1\Big\}. \tag{3.17}$$

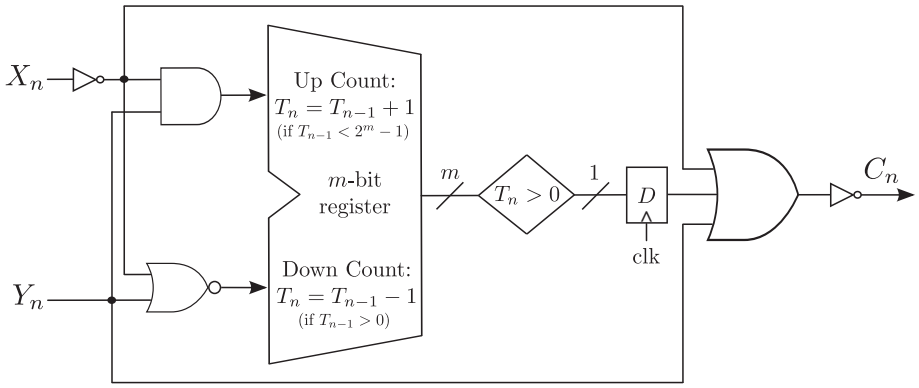


Figure 3.3: Proposed stochastic subtracter architecture. T_n is the m-bit register's current state, updated according to (3.17).

3.1.2.1 Markov Chain Modeling

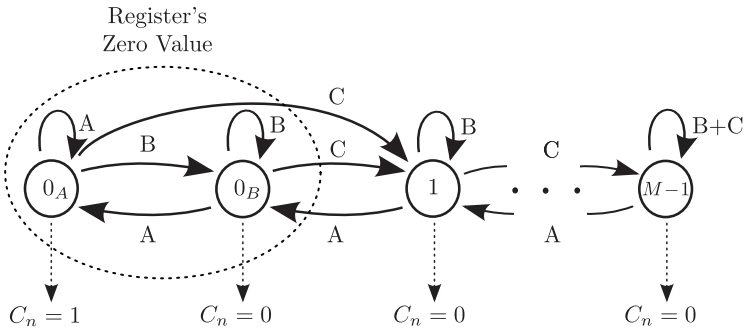


Figure 3.4: Markov Chain model of the proposed stochastic subtracter. The register's zero state, is represented by two states in the model, 0_A and 0_B. Transition probabilities A, B and C are given by (3.18)

The operation of the proposed subtracter architecture is modeled by the MC model shown in Fig. 3.4. Both zero states 0_A and 0_B represent the zero value of the register, whereas 1 to $M-1$ represent the corresponding non-zero values of the register. Moreover, the state of the MC model, S_n belongs to the set of $M+1$ elements given in (3.2), while the state S_n and its output C_n is determined by the following transition probabilities

$$\begin{aligned} A =& P_r(X_n = 1)P_r(Y_n = 0) \\ B =& P_r(X_n = 1)P_r(Y_n = 1) + P_r(Y_n = 0)P_r(X_n = 0) \\ C =& P_r(X_n = 0)P_r(Y_n = 1). \end{aligned} \tag{3.18}$$

The analysis of the MC's behavior can be obtained by using equations (3.4), (3.5) and (3.7), along with (3.18) to calculate the probability distribution vector of state S_n after $n = 1, 2, \ldots, N$ steps. Note that although matrix H is the same for both the adder and subtracter, transition probabilities A, B and C are different. Also, the MC models are the same, except the output values.

3.1.2.2 Expected Output Value and Verification of Operation

According to the MC model of Fig. 3.4 it is $C_n = 1$ if and only if $S_n = 0_A$. Therefore, the expected value of the instantaneous output is

$$\mathbb{E}[C_n] = P_r(C_n = 1) = P_r(S_n = 0_A) = p_0 H^n e_1^T. \tag{3.19}$$

The average value of the output N-bit sequence is

$$\widetilde{C}_N = \frac{1}{N}\left(C_1 + C_2 + \cdots + C_N\right), \tag{3.20}$$

with expected value given by

$$\mathbb{E}[\tilde{C}_N] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[C_n] = \frac{1}{N}p_0\Big(\sum_{n=1}^{N} H^n\Big)e_1^T. \tag{3.21}$$

The procedure to verify the operation of the proposed subtracter architecture is identical to that of the adder in Subsection 3.1.1.2. Following Lemma 1 and since matrix H is irreducible according to Theorem 1 and corresponding assumptions, we conclude that the operation at the limit case using (3.21) implies

$$\lim_{M\to\infty}\left(\lim_{N\to\infty}\mathbb{E}[\widetilde{C}_N]\right) = X - Y. \tag{3.22}$$

Also, it can be shown that bipolar representation $C = X - Y$ of the stochastic subtracter is achieved using $C \mapsto 2C$.

3.2 MAX and MIN Architectures

The MAX & MIN are very popular non-linear functions [50], especially in max pooling operations, and thus their efficient implementation is significant within SC's context. Current MAX & MIN architectures include the following ones.

The architecture by Lee et al.[39] realizes the stochastic MAX & MIN by correlating [4] the input sequences using a three state FSM and then a single gate to produce the output, depending on the desired function (MAX or MIN). The FSM's number of states limits the accuracy of the output since it can only store logic ones according to the FSM depth used.

Another architecture, by Li et al. in [43] uses MUXs and the FSM-based tanh function [15] to realize the MAX & MIN. One of the two MUXs though, uses an additional hardware-demanding binary-to-stochastic converter to generate the MUX's select signal (besides its inputs) thus increasing the hardware requirements [65]. Furthermore, the dependence of the FSM's number of states with the input sequence length requires numerical simulations beforehand to derive the register's size that yields the highest computational accuracy. Following Li et al. [43], the approach by Yu et al. in [89] replaces the binary-to-stochastic converter with an XOR to reduce the hardware overhead, keeping the rest of the processing

structure.

A recent method to realize the MAX & MIN is proposed by Lunglmayr et al. in [58]. Instead of tanh-based FSM as in Yu et al. [89], it uses a shift register to store the ones from one of its inputs, and its least significant bit (LSB) produces a logic 1 if it has saturated up to the LSB. Similarly to [89], the size of shift register that yields the highest computational accuracy is derived with numerical simulations according to the stochastic sequence length used. Moreover, if the shift register's size is not selected accurately, the output's accuracy is reduced as shown in [58].

Motivated by the design challenges of the former methods combined with the necessity for fast computations in SC, we propose a different approach for MAX & MIN. The proposed architectures utilize an accumulator to capture and store the signed bit-differences between their two input sequences, without additional random sources, making their operation deterministic. This results in fast convergence and at the same time highly-accurate computations using short input sequence lengths. The above properties are demonstrated by modeling the architectures using Markov Chains, allowing us to explain their operating principles in detail, derive the first moment statistics of their output and prove their proper operation at the limit.

3.2.1 Stochastic MAX Architecture

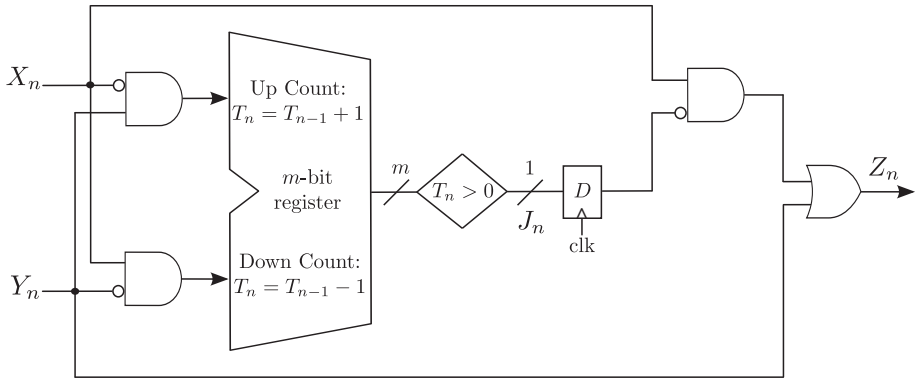


Figure 3.5: Proposed stochastic MAX architecture. T_n is the m-bit register's state, updated according to (3.23).

The proposed stochastic MAX architecture is shown in Fig. 3.5, where $\{X_n\}$, $\{Y_n\}$ are the stochastic input sequences and $\{Z_n\}$ is the output. Ideally, if for some n it is $Y_n > X_n$, then the m-bit register's value is increased by 1 (up count), whereas if $Y_n < X_n$, it is decreased by 1 (down count). If $Y_n = X_n$ the register's value remains unchanged. Also, we assume the initial value $T_0 = 0$. One could say that the m-bit register's purpose is to count the *signed* bit-differences between its two inputs.

It is important to note that the up & down counting of the m-bit register is *saturating*, meaning that states 0 and $M-1$ cannot be exceeded and it is always $T_n \in \mathcal{T}_R$ where $\mathcal{T}_R \triangleq \{0, 1, 2, ..., M-1\}$, with $M = 2^m$ being the total number of states. Hence, from the architecture of Fig. 3.5 we conclude that the

state T_n evolves according to

$$T_n = \max\Big\{ \min\big\{T_{n-1} + \overline{X}_n Y_n - X_n \overline{Y}_n, M-1\big\}, 0\Big\}, \tag{3.23}$$

where $\overline{X}_n = 1 - X_n$ and $\overline{Y}_n = 1 - Y_n$.

The architecture's output Z_n is determined as follows: if X_n and Y_n are both 0 or both 1, then Z_n is 0 or 1 respectively; if $Y_n > X_n$ then $Z_n = 1$; if $Y_n < X_n$, then $Z_n = 1$ if the register was zero in the previous cycle, i.e. if $T_{n-1} = 0$, and it is $Z_n = 0$ otherwise. Defining J_n to be 1 if $T_n > 0$ and zero otherwise, and by inspecting the architecture in Fig. 3.5, the output Z_n can be expressed as

$$Z_n = Y_n + X_n \overline{J_{n-1}}. \tag{3.24}$$

The deterministic behavior of the proposed architecture in Fig. 3.5 is captured by (3.23) and (3.24). Specifically, the output Z_n is a function of the inputs and the state T_n without any additional randomization from any source. As such, the resolution of $\{Z_n\}$ is only limited by the length of the N-bit stochastic input sequences and the register's size.

3.2.1.1 Markov Chain Modeling

To model the operation of the proposed architecture with the stochastic inputs, we consider two Markov Chain (MC) models. The first one is more simple and allows us to easily model the transitions of the state. However, it not convenient for modeling the output which is a function of the previous state and the current inputs. To simplify the derivation of output's statistics, we extend the first model by doubling the number of states, so that the output depends only on the current state. Both models are helpful in explaining different aspects of the architecture's behavior and are discussed in the following subsections.

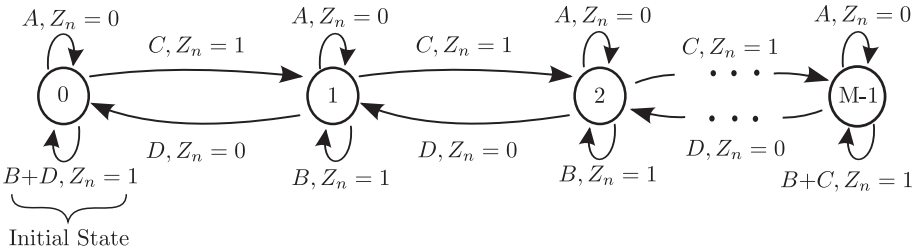


Figure 3.6: Markov Chain model of the proposed stochastic MAX architecture. Output Z_n is determined by the state's transition according to transition probabilities A, B, C, D given by (3.26).

The first MC model is shown in Fig. 3.6. It describes the MAX architecture's operation, corresponding to a Mealy FSM. The model's M states have the obvious one-to-one correspondence with the register's states. The MC state S_n at time index n, starting from $S_0 = 0$, transitions within the set

$$\mathcal{S} \triangleq \{0, 1, 2, \ldots, M-2, M-1\}. \tag{3.25}$$

If the MC's current state is S_{n-1} at time index $n-1$, then inputs X_n, Y_n along with S_{n-1} determine the output Z_n as well as the next state S_n. The transition probabilities A, B, C and D are

$$\begin{aligned} A =& P_r(X_n = 0)P_r(Y_n = 0) \\ B =& P_r(X_n = 1)P_r(Y_n = 1) \\ C =& P_r(X_n = 0)P_r(Y_n = 1) \\ D =& P_r(X_n = 1)P_r(Y_n = 0). \end{aligned} \tag{3.26}$$

To proceed with the analysis of the MC's behavior, we define the $M \times M$ transition probability matrix

$$H = \Big[P_r(S_{n+1} = s|S_n = \sigma)\Big]_{\sigma,s\in\mathcal{S}}$$

where $P_r(S_{n+1} = s|S_n = \sigma)$ is the transition probability from state σ to state s, at time index n, and $\sigma, s = 0, 1, \ldots, M-1$. From (3.26) it is

$$H = \begin{bmatrix} 1-C & C & 0 & \ldots & \ldots & 0 \\ D & A+B & C & 0 & \ldots & 0 \\ 0 & D & A+B & C & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & D & A+B & C \\ 0 & \ldots & \ldots & 0 & D & 1-D \end{bmatrix}. \tag{3.27}$$

The probability distribution vector of state S_n is defined as

$$p_n^T \triangleq \begin{bmatrix} P_r(S_n = 0) \\ P_r(S_n = 1) \\ P_r(S_n = 2) \\ \vdots \\ P_r(S_n = M-1) \end{bmatrix} \in [0,1]^M. \tag{3.28}$$

For $n = 1, 2, \ldots, N$ steps it is expressed as

$$p_n = p_0 H^n \in [0,1]^M, \tag{3.29}$$

where p_0 is the initial distribution vector representing the starting state of the register, i.e. $S_0 = 0$, i.e.,

$$p_0 = \Big[1, 0, 0, ...0\Big] \in [0,1]^M. \tag{3.30}$$

Despite its simplicity, the MC model of Fig. 3.6 is not convenient for the analysis of the statistics of the output. Instead, we can double the number of its states to get the MC model of Fig. 3.7. This extended MC model corresponds to a Moore FSM, relating the output value Z_n only to the state. Each

register state is represented by two states in the model of Fig. 3.7. The states of the model are classified into two sub-sets; the first one, $\mathcal{S}_a \triangleq \{0_a, 1_a, \ldots, (M-1)_a\}$ containing the states that output $Z_n = 0$, and the second one, $\mathcal{S}_b \triangleq \{0_b, 1_b, \ldots, (M-1)_b\}$ containing the states that output $Z_n = 1$. The MC's state $\tilde{S}_n$ transitions within the $2M$ states in

$$\tilde{\mathcal{S}} \triangleq \mathcal{S}_a \cup \mathcal{S}_b = \{0_a, 0_b, 1_a, 1_b, \ldots, (M-1)_a, (M-1)_b\}, \tag{3.31}$$

according to inputs X_n, Y_n and with initial state $\tilde{S}_0 = 0_a$. The transition probability matrix $\tilde{H} \in$

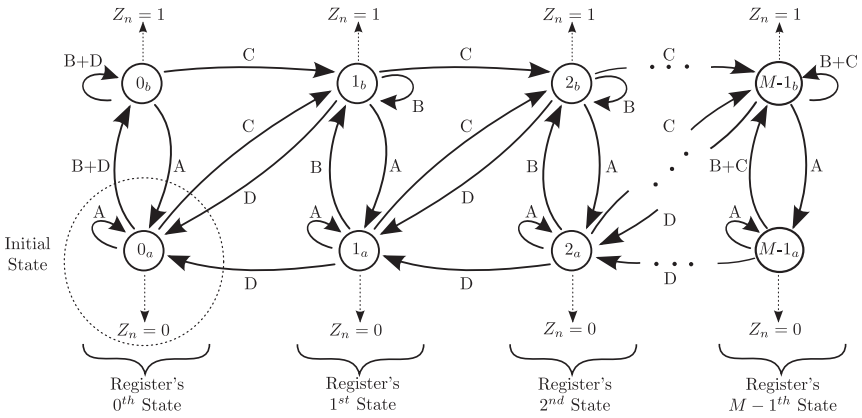


Figure 3.7: Extended Markov Chain model of the proposed stochastic MAX architecture with transition probabilities given by (3.26). Each register state is represented by two states in the model and is classified into two subsets of states; upper ones outputting $Z_n = 1$ and lower ones outputting $Z_n = 0$. Subscripts a, b denote in which subset $\tilde{S}_n$ is currently into. Transition probabilities A, B, C, D are given by (3.26).

$[0,1]^{2M \times 2M}$ of the model in Fig. 3.7 is expressed using A, B, C, D from (3.26), the definitions $F \triangleq B + D$ and $U \triangleq B + C$ and the state ordering $\big(0_a, 0_b, 1_a, 1_b, \ldots, (M-1)_a, (M-1)_b\big)$ as follows

$$\tilde{H} = \begin{bmatrix}
A & F & 0 & C & 0 & & & & \ldots & & & 0 \\
A & F & 0 & C & 0 & & & & \ldots & & & 0 \\
D & 0 & A & B & 0 & C & 0 & & \ldots & & & 0 \\
D & 0 & A & B & 0 & C & 0 & & \ldots & & & 0 \\
0 & 0 & D & 0 & A & B & 0 & C & 0 & \ldots & & 0 \\
0 & 0 & D & 0 & A & B & 0 & C & 0 & \ldots & & 0 \\
\vdots & & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & & \vdots \\
\vdots & & & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\
0 & & & \ldots & & 0 & D & 0 & A & B & 0 & C \\
0 & & & \ldots & & 0 & D & 0 & A & B & 0 & C \\
0 & & & \ldots & & & & 0 & D & 0 & A & U \\
0 & & & \ldots & & & & 0 & D & 0 & A & U
\end{bmatrix}. \tag{3.32}$$

The probability distribution vector of state $\tilde{S}_n$, is defined as

$$\tilde{p}_n^T \triangleq \begin{bmatrix} P_r(\tilde{S}_n = 0_a) \\ P_r(\tilde{S}_n = 0_b) \\ P_r(\tilde{S}_n = 1_a) \\ P_r(\tilde{S}_n = 1_b) \\ \vdots \\ P_r(\tilde{S}_n = (M-1)_a) \\ P_r(\tilde{S}_n = (M-1)_b) \end{bmatrix} \in [0,1]^{2M} \tag{3.33}$$

and it is expressed as

$$\tilde{p}_n = \tilde{p}_0 \tilde{H}^n \in [0,1]^{2M}, \tag{3.34}$$

where the initial state of the register $\tilde{S}_0 = 0_a$ is given by

$$\tilde{p}_0 = \Big[1, 0, 0, ...0\Big] \in [0,1]^{2M}. \tag{3.35}$$

3.2.1.2 Expected Output Value and Proof of Operation

To derive the first moment statistics of the MAX architecture, we use the MC model of Fig. 3.7 along with equations (3.32), (3.34) and (3.35). Based on the model, we use the fact that $Z_n = 1$ if and only if $\tilde{S}_n \in \mathcal{S}_b$. Therefore, the expected value of the output Z_n is

$$\mathbb{E}[Z_n] = P_r(Z_n = 1) = P_r\left(\tilde{S}_n \in \mathcal{S}_b\right) = \tilde{p}_0 \tilde{H}^n q_e^T, \tag{3.36}$$

with q_e (ones in the even-indexed positions) defined as

$$q_e \triangleq [0, 1, 0, 1, \ldots, 0, 1] \in [0,1]^{2M}. \tag{3.37}$$

The average of the N-bit output sequence is

$$\widetilde{Z}_N = \frac{1}{N}\Big(Z_1 + Z_2 + \cdots + Z_N\Big), \tag{3.38}$$

and using (3.36) its expected value is written as

$$\mathbb{E}[\widetilde{Z}_N] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[Z_n] = \frac{1}{N}\tilde{p}_0\left(\sum_{n=1}^{N}\tilde{H}^n\right)q_e^T. \tag{3.39}$$

To proceed with the proof of operation, we assume that $0 < X, Y < 1$ and $X \neq Y$, which imply that

$0<A, B, C, D<1$ and $\rho \neq 1$ where

$$\rho \triangleq \frac{C}{D} = \frac{(1-X)Y}{(1-Y)X}. \tag{3.40}$$

By inspecting the MC model of Fig. 3.7 one can observe that the chain is *irreducible*, since every state is accessible from every other one, and so the transition matrix $\tilde{H}$ is also irreducible.

Let $v^T = [v_1, v_2, \ldots, v_{2M}]^T \in \mathbb{R}^{2M}$ be the left eigenvector of $\tilde{H}$, i.e. $v^T\tilde{H} = v^T$, corresponding to eigenvalue 1 and be normalized such that $v^T\underline{1} = 1$, where $\underline{1} = [1, 1, \ldots, 1]^T \in \mathbb{R}^{2M}$ is a column vector of ones. Then, it can be verified that

$$\begin{aligned} v_1 &= Aw_1 + Dw_2 \\ v_2 &= Fw_1 \\ v_{2k-1} &= Aw_k + Dw_{k+1} \\ v_{2k} &= Cw_{k-1} + Bw_k \\ v_{2M-1} &= Aw_M \\ v_{2M} &= Cw_{M-1} + Uw_M \end{aligned} \tag{3.41}$$

where $k = 2, 3, \ldots, M-1$ and w_k is given by

$$w_k = \lambda\rho^{k-1}, k = 1, 2, \ldots, M \tag{3.42}$$

with

$$\lambda \triangleq \frac{\rho - 1}{\rho^M - 1}. \tag{3.43}$$

Since the transition matrix H is irreducible, from Theorem 8.6.1 in [31] it is $\lim_{N\to\infty} \frac{1}{N}\sum_{n=1}^{N} \tilde{H}^n = \underline{1}v^T$. Combining it with (3.39) we get

$$\lim_{N\to\infty} \mathbb{E}[\tilde{Z}_N] = \tilde{p}_0\underline{1}v^T q_e^T = v^T q_e^T = \sum_{k=1}^{M} v_{2k}. \tag{3.44}$$

From (3.41) and (3.42) we have

$$\begin{aligned} v_2 &= F\lambda \\ v_{2k} &= \lambda(C + B\rho)\rho^{k-2} \\ v_{2M} &= \lambda(C + U\rho)\rho^{M-2}, \end{aligned} \tag{3.45}$$

resulting in

$$\sum_{k=1}^{M} v_{2k} = \lambda \left\{ F + (C + B\rho)\frac{\rho^{M-2} - 1}{\rho - 1} + (C + U\rho)\rho^{M-2} \right\}. \tag{3.46}$$

Combining the above and taking the limit when $N, M \to \infty$ we get

$$\lim_{M\to\infty} \left(\lim_{N\to\infty} \mathbb{E}[\tilde{Z}_N] \right) = \lim_{M\to\infty} \left(\sum_{k=1}^{M} v_{2k} \right) = \begin{cases} X, & X > Y \\ & \\ Y, & Y > X \end{cases}, \tag{3.47}$$

which verifies that $\tilde{Z}_N$ converges to $\max\{X, Y\}$.

3.2.2 Stochastic MIN Architecture

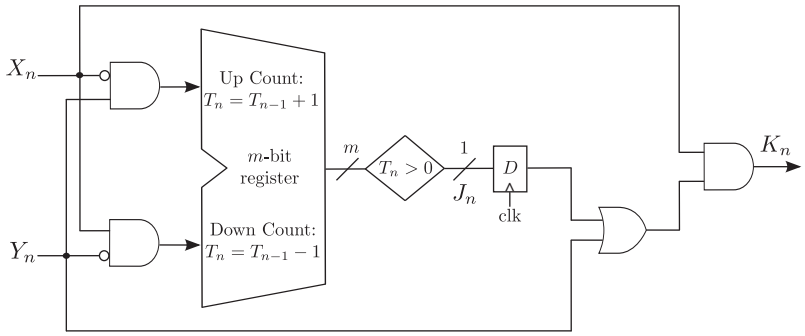


Figure 3.8: Proposed stochastic MIN architecture. T_n is the m-bit register's state, updated according to (3.23).

The proposed MIN architecture is shown in Fig. 3.8. Again, the m-bit register is used to count the number of cases $Y_n > X_n$ minus the number of cases $Y_n < X_n$. Therefore, the accumulator's current value T_n, starts from $T_0 = 0$, belongs in the set $\mathcal{T}_R \triangleq \{0, 1, 2, ..., M-1\}$ which has a total of $M = 2^m$ states and is updated according to (3.23). Similarly to the max architecture, states 0 and $M-1$ constrain the values' range of T_n.

In contrast to the MAX architecture, the output K_n here is determined as follows: if X_n and Y_n are both 0 or 1, then K_n has the same value 0 or 1 respectively; if $Y_n > X_n$, then K_n always outputs 0; and, if $Y_n < X_n$, then $K_n = 1$ if and only if the register's previous value was $T_{n-1} > 0$, and $K_n = 0$ otherwise. Summarizing the former cases and also considering the architecture in Fig. 3.8 as well as the definition $J_n = T_n > 0$, the instantaneous output K_n is expressed as

$$K_n = X_n\big(Y_n + J_{n-1}\big). \tag{3.48}$$

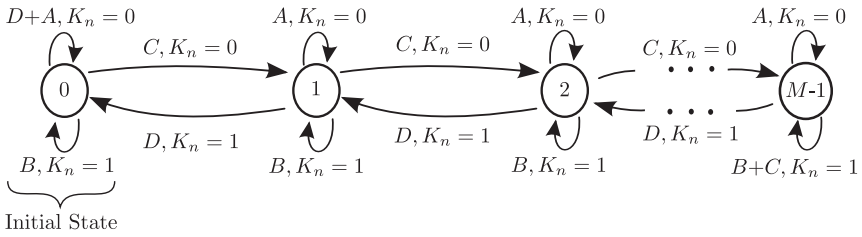


Figure 3.9: Markov Chain model of the proposed stochastic MIN architecture. Output K_n is determined by the state's transition according to transition probabilities A, B, C, D given by (3.26).

3.2.2.1 Markov Chain Modeling

The operation of the proposed MIN architecture is modeled using the MC model in Fig. 3.9. The MC's current state S_n transitions within its M states in the set $\mathcal{S}$ given by (3.25), while its probability distribution vector after N steps is calculated using equations (3.26), (3.27), (3.29) and (3.30).

The MC model in Fig. 3.9 is converted to that in Fig. 3.10 which allows to relate its current state $\tilde{S}_n$ to the output K_n by classifying the model's states into the sub-sets $\mathcal{S}_a, \mathcal{S}_b$ that always output $K_n = 1$ and $K_n = 0$ respectively. Furthermore, $\tilde{S}_n$ transitions within $2M$ states in the set given by (3.25), with initial value $\tilde{S}_0 = 0_a$. Assuming the states' ordering $\big(0_a, 0_b, \ldots, (M-1)_a, (M-1)_b\big)$, the transition probability

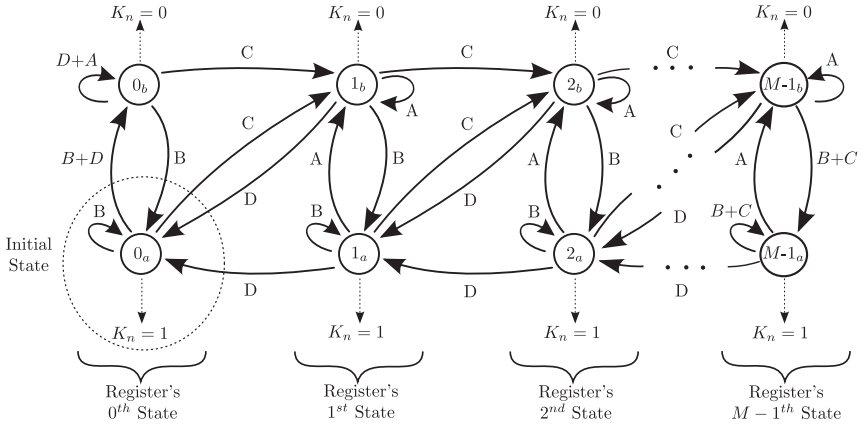


Figure 3.10: Extended Markov Chain model of the proposed stochastic MIN with transition probabilities given by (3.26). Each register state is represented by two states; upper one outputs $K_n = 0$ and lower one outputs $K_n = 1$. Subscripts a, b denote in which set $\tilde{S}_n$ is currently into.

matrix of the MC model is given by (3.49) where we have defined $U \triangleq B + C$ and $W \triangleq D + A$

$$\tilde{H} = \begin{bmatrix}
B & W & 0 & C & 0 & & & \dots & & 0 \\
B & W & 0 & C & 0 & & & \dots & & 0 \\
D & 0 & A & B & 0 & C & 0 & \dots & & 0 \\
D & 0 & A & B & 0 & C & 0 & \dots & & 0 \\
0 & 0 & D & 0 & A & B & 0 & C & 0 & \dots & 0 \\
0 & 0 & D & 0 & A & B & 0 & C & 0 & \dots & 0 \\
\vdots & & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\
\vdots & & & & \ddots & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\
0 & & & \dots & & 0 & D & 0 & A & B & 0 & C \\
0 & & & \dots & & 0 & D & 0 & A & B & 0 & C \\
0 & & & \dots & & & & 0 & D & 0 & A & U \\
0 & & & \dots & & & & 0 & D & 0 & A & U
\end{bmatrix}. \tag{3.49}$$

3.2.2.2 Expected Output Value and Proof of Operation

The expected value of the instantaneous output K_n is

$$\mathbb{E}[K_n] = P_r(K_n = 1) = P_r\left(\tilde{S}_n \in \mathcal{S}_a\right) = \tilde{p}_0 \tilde{H}^n q_o^T, \tag{3.50}$$

where we used (3.35) and (3.49), and, q_o is defined as

$$q_o \triangleq [1, 0, 1, 0, \dots, 1, 0] \in [0, 1]^{2M}. \tag{3.51}$$

The average of the output N-bit sequence is

$$\widetilde{K}_N = \frac{1}{N}\Big(K_1 + K_2 + \cdots + K_N\Big), \tag{3.52}$$

and its expected value, using (3.50), is given by

$$\mathbb{E}[\widetilde{K}_N] = \frac{1}{N}\sum_{n=1}^{N} \mathbb{E}[K_n] = \frac{1}{N}\tilde{p}_0 \left(\sum_{n=1}^{N} \tilde{H}^n\right) q_o^T. \tag{3.53}$$

Note that the procedure to prove the operation of the MIN architecture, follows closely that of the MAX one.

3.3 Compact MAX and MIN Architectures

In this section we present a different architecture for the realization of the MAX and the MIN. In contrast to the previous proposed ones, the architecture is compact in the sense that it can realize the MAX and/or the MIN without changing the logic gates constituting the architecture. Of major interest is

also the analysis for the first moment statistics and the proof of operation accompanying the architecture, which is conducted on a stochastic FSM expressing a Mealy behavior.

3.3.1 Compact MAX Architecture

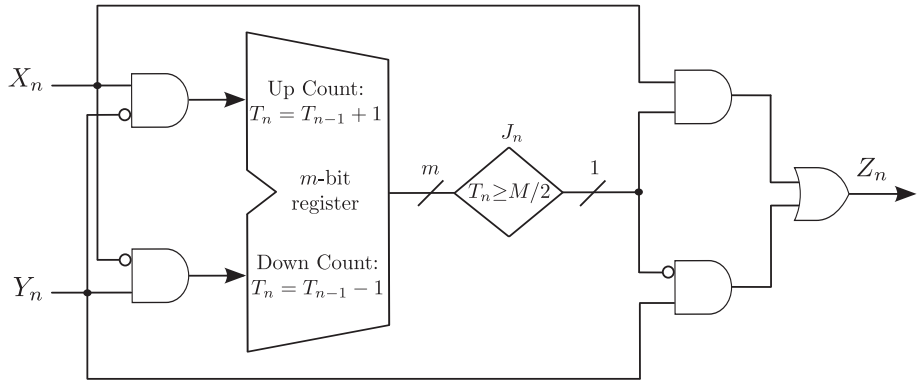


Figure 3.11: Proposed compact stochastic MAX architecture where $M = 2^m$. T_n is the register's current value, updated according to (3.54).

Fig. 3.11 shows the proposed stochastic MAX architecture where $\{X_n\}$ and $\{Y_n\}$ are the stochastic input sequences, assumed to be generated by SNGs, and $\{Z_n\}$ is the output. Its operation is based on increasing the m-bit register's current value T_n by 1 if $X_n > Y_n$ and decreasing it by 1 if $X_n < Y_n$, within the set $\mathcal{T}_R \triangleq \{0, 1, 2, \ldots, M-1\}$, starting from $T_0 = M/2$, where $M = 2^m$ is the number values. Essentially, the register counts the number of *signed* bit-wise differences between its two inputs, X_n and Y_n. We can express the update of the register's value as

$$T_n = \max\Big\{ \min\{T_{n-1} + X_n - Y_n, M-1\}, 0 \Big\}, \tag{3.54}$$

where the *min* and *max* functions imply the natural *saturating* behavior of the counter since values 0 and $M-1$ cannot be exceeded.

To derive the output Z_n, we define first the result of the comparison between the register's current value T_n and the reference value $M/2$ as

$$J_n = \begin{cases} 0, & \text{if } T_n < M/2 \\ 1, & \text{if } T_n \geq M/2 \end{cases}, \tag{3.55}$$

which, following Fig. 3.11, implies that

$$Z_n = J_n X_n + \overline{J}_n Y_n, \tag{3.56}$$

where $\overline{J}_n = 1 - J_n$ (considering 0 and 1 as Real numbers). Note that $J_n = 1$ means that input sequence X_n has had more 1s than Y_n had, within the storing range of the register. In this case, the output is $Z_n = X_n$ as expected, whereas if $J_n = 0$ it is $Z_n = Y_n$.

Although the input sequences are stochastic, the architecture's operation is deterministic, modeled by Eqs. (3.54) - (3.56), and the output Z_n is a function of T_n, X_n and Y_n. These imply that the accuracy of $\{Z_n\}$ depends on: 1) the size, m, of the register, and 2) the length, N, of the input sequences.

3.3.1.1 Markov Chain Modeling

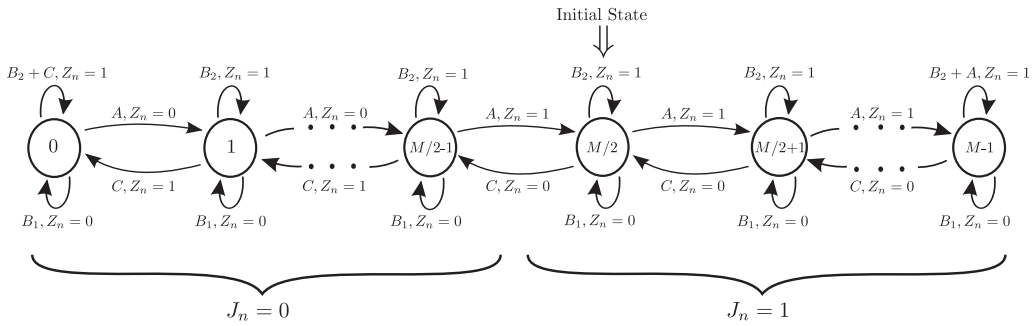


Figure 3.12: Markov Chain model of the proposed compact stochastic MAX architecture. Transition probabilities are given by (3.58). J_n denotes the result of the comparison between the register's current value with the initial one $M/2$.

To investigate the stochastic behavior of the proposed MAX architecture we model it as the Markov Chain (MC) shown in Fig. 3.12. The MC has the M states in the set given by (3.57)

$$\mathcal{S} \triangleq \{0, 1, \ldots, M-2, M-1\}, \tag{3.57}$$

and its current state is S_n, corresponding to the current value T_n of the register. The initial state is $S_0 = M/2$.

The transition from state S_{n-1} to state S_n is determined by S_{n-1}, X_n and Y_n. Using the probability distributions of inputs X_n and Y_n, the assumption that their sequences are IID, and the operation of the MAX architecture in Fig. 3.11, we derive the transition probabilities shown in the MC model in Fig. 3.12 as

$$\begin{aligned}
A &\triangleq P_r(X_n = 1)P_r(Y_n = 0) = X(1-Y) \\
B_1 &\triangleq P_r(X_n = 0)P_r(Y_n = 0) = (1-X)(1-Y) \\
B_2 &\triangleq P_r(X_n = 1)P_r(Y_n = 1) = XY \\
B &\triangleq B_1 + B_2 \\
C &\triangleq P_r(X_n = 0)P_r(Y_n = 1) = (1-X)Y,
\end{aligned} \tag{3.58}$$

where we have set $X = P_r(X_n = 1)$ and $Y = P_r(Y_n = 1)$. Assuming the state ordering $(0, 1, \ldots, M-1)$ in $\mathcal{S}$ and using (3.58), the $M \times M$ transition probability matrix $H = \left[P_r(S_{n+1} = s_j | S_n = s_i)\right]_{s_i, s_j \in \mathcal{S}}$

is written as

$$H = \begin{bmatrix} 1-A & A & 0 & \dots & \dots & 0 \\ C & B & A & 0 & \dots & 0 \\ 0 & C & B & A & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & C & B & A \\ 0 & \dots & \dots & 0 & C & 1-C \end{bmatrix}. \tag{3.59}$$

The probability distribution vector of state S_n, is defined as

$$p_n^T \triangleq \begin{bmatrix} P_r(S_n = 0) \\ P_r(S_n = 1) \\ P_r(S_n = 2) \\ \vdots \\ P_r(S_n = M-1) \end{bmatrix} \in [0,1]^M, \tag{3.60}$$

and it is expressed as,

$$p_n = p_0 H^n \in [0,1]^M. \tag{3.61}$$

Here, p_0 is the initial distribution vector representing the starting state of the register, $S_0 = M/2$. It is

$$p_0 = e_{M/2+1}, \tag{3.62}$$

where $e_i = [0, \dots, 0, 1, 0, \dots, 0] \in \mathbb{R}^M$ is the i-th standard vector, i.e., with all zeros except the i^{th} entry being one.

3.3.1.2 Expected Output Value and Proof of Operation

We use the MC model in Fig. 3.12 and (3.58)-(3.62) to derive the output's first moment statistics. The expected value of the output Z_n is expressed as

$$\begin{aligned} \mathbb{E}[Z_n] &= P_r(Z_n = 1) \\ &= \sum_{\substack{s \in \mathcal{S} \\ x,y \in \{0,1\}}} P_r(Z_n = 1, S_{n-1} = s, X_n = x, Y_n = y) \\ &= \sum_{\substack{s \in \mathcal{S} \\ x,y \in \{0,1\}}} P_r(Z_n = 1 | S_{n-1} = s, X_n = x, Y_n = y) P_r(S_{n-1} = s, X_n = x, Y_n = y) \end{aligned} \tag{3.63}$$

Regarding the conditional probability $P_r(Z_n = 1 | S_{n-1} = s, X_n = x, Y_n = y)$ we note that Z_n is a (deterministic) function of S_{n-1}, X_n and Y_n, as can be seen in the MC model in Fig. 3.12. Using the MC model and Eq. (3.56) we can distinguish between three possible cases, i.e.:

1. When $S_{n-1} \leq M/2-2$, then $Z_n = 1$ if and only $Y_n = 1$,

2. When $S_{n-1} = M/2-1$, then $Z_n = 1$ if and only at least one of X_n, Y_n is 1,

3. When $S_{n-1} \geq M/2$, then $Z_n = 1$ if and only $X_n = 1$.

Therefore we can decompose the summation in Eq. (3.63) as

$$\begin{aligned}\mathbb{E}[Z_n] = &\sum_{s=0}^{M/2-2} P_r(S_{n-1}=s, Y_n=1) + \sum_{(x,y)\neq(0,0)} P_r(S_{n-1}=M/2-1, X_n=x, Y_n=y)\\ &+ \sum_{s=M/2}^{M-1} P_r(S_{n-1}=s, X_n=1). \end{aligned} \tag{3.64}$$

Since X_n, Y_n and S_{n-1} are independent random variables , it is

$$P_r(S_{n-1}=s, X_n=x, Y_n=y) = P_r(S_{n-1}=s)P_r(X_n=x)P_r(Y_n=y)$$

simplifying (3.64) to

$$\begin{aligned}\mathbb{E}[Z_n] &= Y\sum_{s=0}^{M/2-2} P_r(S_{n-1}=s) + (X+Y-XY)P_r(S_{n-1}=M/2-1)\\ &\quad + X\sum_{s=M/2}^{M-1} P_r(S_{n-1}=s)\\ &= p_{n-1}\Big(Ye_L^T + (X+Y-XY)e_{M/2}^T + Xe_U^T\Big), \end{aligned} \tag{3.65}$$

where $e_L = \sum_{i=1}^{M/2-1} e_i$ and $e_U = \sum_{i=M/2+1}^{M} e_i$. The N-bit output sequence time-average,

$$\widetilde{Z}_N = \frac{1}{N}\Big(Z_1 + Z_2 + \cdots + Z_N\Big), \tag{3.66}$$

has the expected value below based on Eq. (3.65),

$$\mathbb{E}[\widetilde{Z}_N] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[Z_n] = \frac{1}{N}p_0\left(\sum_{n=0}^{N-1} H^n\right)\Big(Ye_L^T + (X+Y-XY)e_{M/2}^T + Xe_U^T\Big). \tag{3.67}$$

Eq. (3.67) is used to confirm the operation of the MAX architecture for large N and M values.

To verify the operation of the proposed MAX architecture we assume that $0 < X, Y < 1$ and $X \neq Y$. Then, from (3.58) it is $0 < A, B, C < 1$, as well as $\rho \neq 1$, where we have defined

$$\rho \triangleq \frac{A}{C} = \frac{X(1-Y)}{Y(1-X)}. \tag{3.68}$$

Moreover, note that $\rho > 1$ if and only if $X > Y$.

Observing the MC model in Fig. 3.12 one can conclude that the MC is irreducible, as each state s_j is accessible, with positive probability, from every other state s_i, implying irreducibility for the matrix H as well. Therefore, from Theorem 8.6.1 in [31] we have that

$$\lim_{N\to\infty}\frac{1}{N}\sum_{n=0}^{N-1}H^n=\underline{1}^T v, \tag{3.69}$$

where the row vector $v\in\mathbb{R}^M$ is the unique left eigenvector of H, $vH=v$, corresponding to eigenvalue 1 and being normalized, i.e. $v\underline{1}^T=1$, and, $\underline{1}=[1,1,\ldots,1]\in\mathbb{R}^M$ is the all ones vector. It can be verified directly that $v=\lambda_M[1,\rho,\rho^2,\ldots,\rho^{M-1}]$, where

$$\lambda_M\triangleq\frac{1-\rho}{1-\rho^M}. \tag{3.70}$$

Combining (3.67) and (3.69) and noting that $p_0\underline{1}^T=1$ we get

$$\begin{aligned}\lim_{N\to\infty}\mathbb{E}[\tilde{Z}_N]&=v\Big(Ye_L^T+(X+Y-XY)e_{M/2}^T+Xe_U^T\Big)\\&=Yve_L^T+(X+Y-XY)ve_{M/2}^T+Xve_U^T.\end{aligned} \tag{3.71}$$

Using the expressions of v, e_L and e_U we get

$$\begin{aligned}ve_L^T&=\lambda_M\left(1+\rho+\cdots+\rho^{M/2-2}\right)=\frac{1-\rho^{M/2-1}}{1-\rho^M}\\ve_{M/2}^T&=\lambda_M\rho^{M/2-1}=\frac{\rho^{M/2-1}-\rho^{M/2}}{1-\rho^M}\\ve_U^T&=\lambda_M\left(\rho^{M/2}+\rho^{M/2+1}+\cdots+\rho^{M-1}\right)=\frac{\rho^{M/2}-\rho^M}{1-\rho^M},\end{aligned} \tag{3.72}$$

directly implying from (3.72) that $\lim_{M\to\infty}\left(\lim_{N\to\infty}\mathbb{E}[\tilde{Z}_N]\right)=Y$ if $\rho<1$ and $\lim_{M\to\infty}\left(\lim_{N\to\infty}\mathbb{E}[\tilde{Z}_N]\right)=X$ if $\rho>1$, and so

$$\lim_{M\to\infty}\left(\lim_{N\to\infty}\mathbb{E}[\tilde{Z}_N]\right)=\begin{cases}X, & X>Y\\ Y, & Y>X\end{cases}, \tag{3.73}$$

which proves that the proposed architecture provides the correct expected result in the limiting case.

3.3.2 Compact MIN Architecture as a Variation of the MAX one

The MIN architecture can be obtained as a variation of the MAX one, as shown in Fig. 3.13. The counting of logic 1s is identical to that of the MAX architecture. The difference between the two architectures is the swap of the NOT gate between the two AND gates that along with the OR gate, determine the output. Therefore, the MIN architecture's analysis is similar to that of the MAX one's.

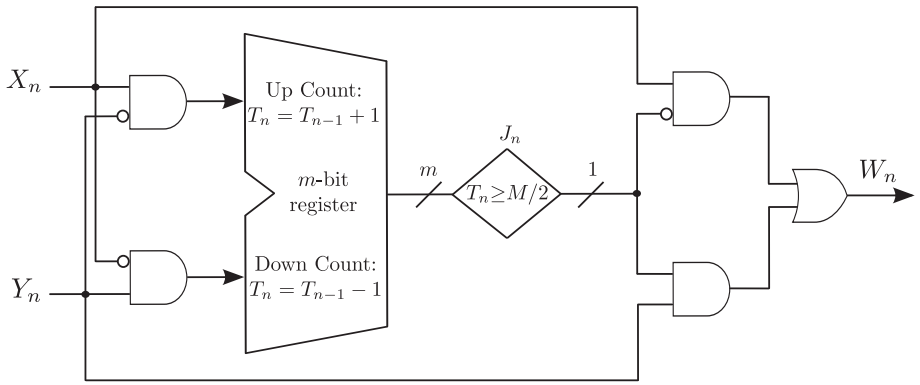


Figure 3.13: Proposed compact stochastic MIN architecture. T_n denotes the $M = 2^m$ register's current value and is updated according to (3.54).

3.4 Stochastic Computing Sigma-Delta Adder

Within the SC-based DSP cores, the multiply-and-add operation is the most important one. Each multiplication between two sequences is realized using an AND or XNOR gate according to the SC number representation format used. The addition part is mainly approached by two design strategies; two-input adder-tree structures and multi-input adders. Typical two-input SC adders scale the addition's result by a factor of two, reducing also the output sequence's resolution by the same factor [65]. This forces each subsequent layer within the adder-tree to increase the addition's scaling in increasing powers of two. To compensate for the resolution drop, the sequence length should be increased in powers of two according to the number of layers within the adder-tree, resulting in increased latency and total energy consumption[80]. Moreover, it is also expected for the adder-tree's output to constrain cascaded operations in handling cases where up-scaling is required, such as in non-linear functions [72].

To address the adder-tree's scaling challenges, multi-input adders were considered for use in SC, with the accumulative parallel counter (APC) [68] being the most popular one [21, 83, 50, 37]. The APC accumulates deterministically all input sequences in parallel producing the result in binary format. In SC-based cascaded computations, however, the APC's binary output introduces the following design challenges; 1) it limits the applicability of existing single-bit input/output Stochastic Finite-State Machines (SFSMs) realizing highly-complex functions including non-linear ones [15, 43] and 2) in case when other arithmetic operations are required, for instance when multiplications follow the output of SFSMs, the binary output has to be regenerated as a stochastic sequence in order for the SC logic gates to be used [52].

Both adder-tree and multi-input adder design strategies have been explored within the context of SC for the realization of Multi-Layer Perceptrons (MLP) [37, 49, 48], a class of NNs. In [37], each neuron forming the MLP is realized using an APC followed by a multi-bit input single-bit output FSM approximating the *tanh* (BTanh) non-linear activation function, implemented as a binary up/down counter. However, the BTanh's design is not systematic; on the one hand the FSM's number of states affecting the *tanh*'s approximation are derived using numerical experiments for fixed input sequence lengths, while

on the other the input sequence's bit-length driving the FSM's state update is not considered. In [49], a hybrid SC MLP is realized using an adder-tree structure composed of extended stochastic logic (ESL) adders [18] in the input layer and APCs in the rest layers. This hybrid format encoding enables the on-line update of weights. On the other hand, the ESL adders require an additional binary-to-stochastic number converted for the select signal, taxing on the hardware resources [18, 80], while the adder-tree requires a tripe modular redundant (TMR) binary search divider, resulting in large sequence lengths for its computation and stabilization phases [50, 49]. With respect to the activation functions, the same approach as in [37] was followed, but, a multi-input single-bit output FSM realizing the rectifier linear unit (ReLU) was used. Similar to [49], in [48] a gradient-based updating scheme was applied to a MLP, showing the effectiveness of the gradients' and the weights' on-line learning. Yet, in [49] no emphasis in the multiply-and-add operations is given.

Motivated by the limitations of existing SC adder design strategies, this work introduces a SC adder architecture that utilizes a first-order sigma-delta modulator (SDM). The proposed Stochastic Computing Sigma-Delta (SCSD) adder sums the bits of the input sequences into a single-data bus and then employs an internal data range conversion scheme so as to exploit the SDM's property of converting a high-resolution signal into a single-bit one. It offers the following advantages: 1) it operates on independent inputs, 2) the addition is done deterministically without additional random sources, 3) it is fast converging with small sequence lengths, 4) it enables cascaded operations to be made efficiently with existing SC arithmetic circuits and 5) it allows the use of *any* single-bit input/output SFSM, thereby opening the SC-based NN design space.

3.4.1 SCSD High-Level Architecture

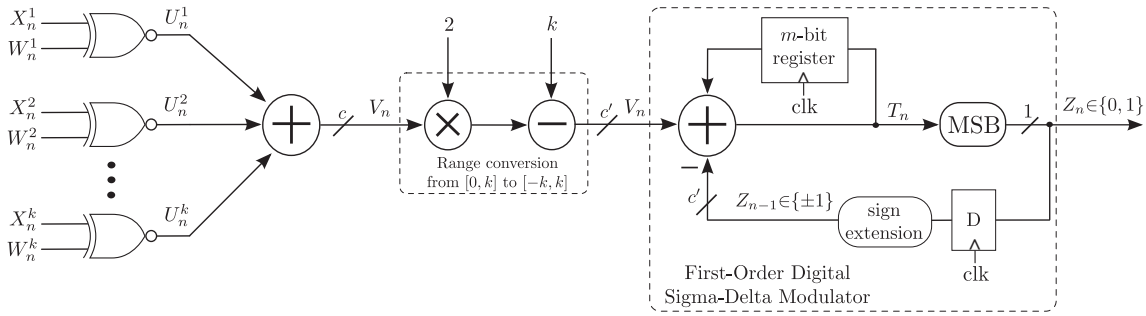


Figure 3.14: Architecture of the proposed Stochastic Computing Sigma-Delta (SCSD) adder. The XNOR gates between the input sequences $\{X_n^j\}_{n=1}^N$, $\{W_n^j\}_{n=1}^N$ are used to multiply numbers in bipolar format. The multiplication results are added to a single bus with the range of its represented value converted from $[0, k]$ to $[-k, k]$. The first-order digital SDM converts a higher resolution signal into a single-bit one, outputting the average of its input according to (3.80), realizing the sum-of-products.

The proposed multi-input single-bit output Stochastic Computing Sigma-Delta (SCSD) adder architecture is shown in Fig. 3.14. Its sequences $\{X_n^j\}_{n=1}^N$, $\{W_n^j\}_{n=1}^N$ with $j = 1, \ldots, k$ are assumed to be i.i.d., while $\{Z_n\}_{n=1}^N$ is the output sequence. The XNOR gates are used to multiply the input sequences in bipolar format [23], as $\{X_n^j\}_{n=1}^N$, $\{W_n^j\}_{n=1}^N$ may carry information of negative-signed numbers. This

results in k intermediate sequences $\{U_n^j\}_{n=1}^N$, whose time-average according to (2.1) is $\widetilde{U}_N^j = \widetilde{X}_N^j \widetilde{W}_N^j$, with probability $p_{U^j}{=}P_r(U_n^j = 1) = p_{X^j} p_{W^j}$.

The summation of the k bits of $\{U_n^j\}_{n=1}^N$ follows the multiplication operation, resulting in the sequence $\{V_n^j\}_{n=1}^N$, where $V_n = \sum_{j=1}^k U_n^j$. It is an integer-valued sequence, since it holds $V_n \in \mathcal{V}$, where $\mathcal{V} = \{0, 1, \ldots, k\}$, with probability $p_V(v) = P_r(V_n = v)$ and time-average value

$$\widetilde{V}_N = \frac{1}{N}\sum_{n=1}^{N} V_n = \frac{1}{N}\sum_{n=1}^{N}\sum_{j=1}^{k} U_n^j = \sum_{j=1}^{k}\left(\frac{1}{N}\sum_{n=1}^{N} U_n^j\right) = \sum_{j=1}^{k}\widetilde{U}_N^j = \sum_{j=1}^{k}\widetilde{X}_N^j \widetilde{W}_N^j. \tag{3.74}$$

It should be noted that the summation operation is strictly deterministic as it is implemented with conventional binary arithmetic as shown in Fig. 3.14, without additional randomizing sources. This implies that 1) there is no loss of information and 2) the precision of V_n is exclusively determined by the length, N, of the input sequences.

To exploit the first order SDM's property of converting a higher-resolution signal into a single-bit one, the value of V_n should have both positive and negative signed numbers. As such, since V_n is the sum of k inputs, the range of V_n $\{0, \ldots, k\}$ is extended to $\{-k, \ldots, k\}$ using the transformation $V_n \mapsto 2V_n - k$. Note that the multiplication operation existing in the range conversion process is realized using a left shift operation. The bit-width c of V_n is determined according to the number of inputs, k, and should be such that it can capture all incoming bits, namely $c = \lceil \log_2 k \rceil$, where $\lceil \cdot \rceil$ is the ceiling function. On the other hand, the bit-width after the range conversion should be $c' = c + 1$, accounting for the signed value of V_n.

The first-order digital SDM (DSDM) contained within the SCSD adder architecture of Fig. 3.14, consists of an adder and an m-bit register, followed by a most significant bit (MSB) selection block. The MSB block, replaces the quantizer existing in the system level model of a typical first order SDM as shown in Fig. 3.15, which is a simplification of the comparison between the register's current value and zero [19, 32]. Therefore, considering the signed representation of T_n, the MSB's operation implements a function $Q(\cdot)$ as

$$Q(T_n) = \begin{cases} 1, & \text{MSB}(T_n) = 0 \\ 0, & \text{MSB}(T_n) = 1 \end{cases}. \tag{3.75}$$

Since (3.75) describes the quantization process of a single-bit DSDM outputting $0, 1$, in the case when $\text{MSB}(T_n) = 1$, the quantizer's output, Z_n, is fed back as a -1 using sign extension, instead of a logic 0.

Assuming that the register's initial value T_0 can be any one within the set $\mathcal{T} = \{0, 1, \ldots, M-1\}$, where $M = 2^m$ is the number of states, the DSDM's current state T_n is updated as

$$T_n = \max\Big\{0, \min\big\{T_{n-1} + V_n - Z_{n-1}, M-1\big\}\Big\}. \tag{3.76}$$

The $\max(\cdot)$ and $\min(\cdot)$ functions are used here to denote the register's natural saturation to states 0 and $M-1$, given that they cannot be exceeded. Therefore, considering (3.75) and (3.76), the SDM's output

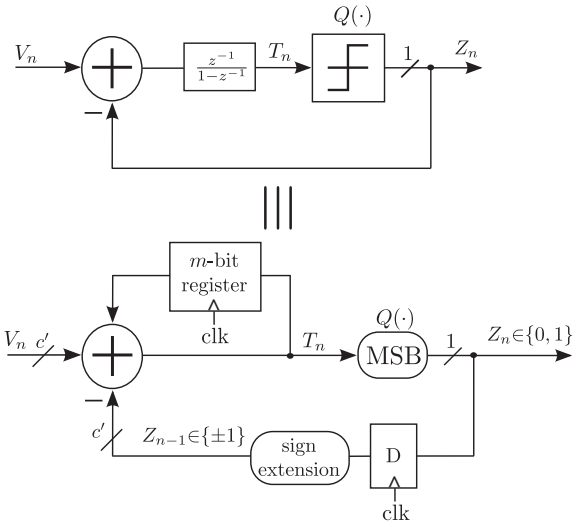


Figure 3.15: Top: system level model of a first-order Sigma-Delta Modulator. Bottom: realization of the first-order Digital Sigma-Delta Modulator. The quantizer block, is replaced by the selection of the most significant bit.

and consequently that of the SDSC adder is expressed as

$$Z_n = Q\left(\max\left\{0, \min\left\{T_{n-1}+V_n-Z_{n-1}, M-1\right\}\right\}\right). \tag{3.77}$$

The first order SDM is stable, i.e $\widetilde{Z}_N$ is bounded in $[-1,1]$ if and only if the input $\widetilde{V}_N$ is bounded in $[-1,1]$ [70]. Otherwise, if $\widetilde{V}_N < -1$ or $\widetilde{V}_N > 1$, a -1 or a 1 will be constantly fed back on every clock cycle respectively, resulting in a repeated decrease or increase of the register's current value T_n.

According to (3.76), the DSDM's state update describes a first-order difference equation. By simplifying the register's saturating behavior and taking the time-average of the sequences according to (2.1), the state update for $n = 0, 1, \ldots, N$ becomes

$$\frac{1}{N}\left(T_N - T_0\right) = \frac{1}{N}\sum_{n=0}^{N}\left(V_n - Z_{n-1}\right). \tag{3.78}$$

Taking the limit $N \to \infty$ in (3.78) and considering that $\widetilde{V}_N \in [-1,1]$, it holds that

$$\widetilde{Z}_N = \widetilde{V}_N, \tag{3.79}$$

which using (3.74) formulates the sum-of-product operation as

$$\widetilde{Z}_N = \sum_{j=1}^{k} \widetilde{X}_N^j \widetilde{W}_N^j. \tag{3.80}$$

Finally, by applying the transformation $\widetilde{Z}_N \mapsto 2\widetilde{Z}_N - 1$ in (3.80), the output's time-average in bipolar

format can be obtained.

3.4.2 Markov Chain Modeling

The proposed SCSD adder's long-term stochastic dynamics can be further explored by describing the operation of the first-order SDM as a Stochastic Finite-State Machine (SFSM) and consequently modeling it as a Markov Chain (MC) [78]. To proceed, it is important to explain first the derivation of 1) the MC's state space and 2) the MC's transition probabilities.

The quantizer's operation according to (3.75), expresses the behavior of the SFSM as a Moore one, given the relation of the current output Z_n to the state T_n. As such, the MC's current state, S_n, transitions within the set $\mathcal{S} = \{0, 1, \ldots, M-1\}$, which is a bijective mapping of the register's set $\mathcal{T}$.

The MC's transition probabilities are determined by $\{V_n\}_{n=1}^{N}$, and can be challenging to model since each r.v. V_n takes values within $\mathcal{V} = \{0, 1, \ldots, k\}$. To this end, we consider the probability generating function (P.G.F.) of V_n defined as $G_{V_n} \triangleq \mathbb{E}(s^{V_n})$, where $s \in \mathbb{R}$, calculated as

$$\begin{aligned} G_{V_n}(s) =& \mathbb{E}\left(s^{V_n}\right) = \mathbb{E}\left(s^{U_n^1+\cdots+U_n^k}\right) = \mathbb{E}\left(s^{U_n^1}\ldots s^{U_n^k}\right) = \mathbb{E}\left(s^{U_n^1}\right)\ldots\mathbb{E}\left(s^{U_n^k}\right) \\ =& \prod_{j=1}^{k} G_{U_n^j}(s) = \prod_{j=1}^{k}\Big((1-p_{U^j}) + p_{U^j}s\Big), \end{aligned} \tag{3.81}$$

where $G_{U_n^j}(s) = (1-p_{U^j}) + p_{U^j}s$, is the P.G.F. of the j-th r.v. U_n^j. Using (3.81), $P_r(V_n = v)$ is calculated as

$$P_r(V_n = v) = \left(\frac{1}{v!}\right)\frac{d^v}{ds^v}\left(G_{V_n}(s)\right)\Big|_{s=0}. \tag{3.82}$$

The MC's state update is similar to that of the register's one in (3.76), since it is determined by the previous state S_{n-1}, the current input V_n and the previous output Z_{n-1}. The relation with the previous output makes the analysis difficult, but, it can be eliminated by introducing a new r.v., V_n^*, as follows

$$S_n = S_{n-1} + V_n - Z_{n-1} = S_{n-1} + V_n^*. \tag{3.83}$$

Note that in (3.83), the state S_n is updated considering that $V_n \in \{-k, \ldots, k\}$ and $Z_n \in \{\pm 1\}$. Moreover, since Z_n is related to the state, it is convenient to partition $\mathcal{S}$ into two subsets $\mathcal{S}_a = \{0, 1, \ldots, M/2-1\}$ and $\mathcal{S}_b = \{M/2, \ldots, M-1\}$, such that $\mathcal{S} = \mathcal{S}_a \cup \mathcal{S}_b$, $\mathcal{S}_a \cap \mathcal{S}_b = \{\}$. Therefore, it holds that if $S_n \in \mathcal{S}_a \Rightarrow Z_n = -1$ and if $S_n \in \mathcal{S}_b \Rightarrow Z_n = 1$.

Once the state update within the state space is defined and the transition probabilities are derived, they can be used to define the $(M \times M)$ transition probability matrix as $H \triangleq [P_r(S_n = \sigma_j | S_{n-1} = \sigma_i)] = [p_{\sigma_i,\sigma_j}]$, where $\sigma_i, \sigma_j \in \mathcal{S}$.

To give a better intuition behind the transitions within the MC and the elements of H, we proceed with the following example. For $k = 3$ inputs, $V_n \in \{0, 1, 2, 3\}$ and after using the mapping $V_n \mapsto 2V_n - k$, $V_n \in \{-3, -1, 1, 3\}$. Moreover, considering that $V_n^* = V_n - Z_{n-1}$, then $V_n^* \in \{-4, -2, -, 2, 4\}$. Using (3.78) to calculate $P_r(V_n = v)$ and taking cases for the transition from S_{n-1} to S_n:

- If $S_{n-1} \in \mathcal{S}_a$
  - If $P_r(V_n = 0)$, then $S_n = S_{n-1} - 2$,
  - If $P_r(V_n = 1)$, then $S_n = S_{n-1}$,
  - If $P_r(V_n = 2)$, then $S_n = S_{n-1} + 2$,
  - If $P_r(V_n = 3)$, then $S_n = S_{n-1} + 4$.
- If $S_{n-1} \in \mathcal{S}_b$
  - If $P_r(V_n = 0)$, then $S_n = S_{n-1} - 4$,
  - If $P_r(V_n = 1)$, then $S_n = S_{n-1} - 2$,
  - If $P_r(V_n = 2)$, then $S_n = S_{n-1}$,
  - If $P_r(V_n = 3)$, then $S_n = S_{n-1} + 2$.

For simplicity we denote $p_V(v)$ as p_V^v and for $M = 8$ states we write matrix H using the state ordering $(0, 1, \ldots, 7)$ as

$$H = \begin{bmatrix} p_V^0+p_V^1 & 0 & p_V^2 & 0 & p_V^3 & 0 & 0 & 0 \\ p_V^0 & p_V^1 & 0 & p_V^2 & 0 & p_V^3 & 0 & 0 \\ p_V^0 & 0 & p_V^1 & 0 & p_V^2 & 0 & p_V^3 & 0 \\ 0 & p_V^0 & 0 & p_V^1 & 0 & p_V^2 & 0 & p_V^3 \\ p_V^0 & 0 & p_V^1 & 0 & p_V^2 & 0 & p_V^3 & 0 \\ 0 & p_V^0 & 0 & p_V^1 & 0 & p_V^2 & 0 & p_V^3 \\ 0 & 0 & p_V^0 & 0 & p_V^1 & 0 & p_V^2 & p_V^3 \\ 0 & 0 & 0 & p_V^0 & 0 & p_V^1 & 0 & p_V^2 + p_V^3 \end{bmatrix}. \tag{3.84}$$

Assuming that the MC's starting state S_0 can be any one within $\mathcal{S}$, then the initial distribution vector is defined as

$$\pi_0 = (1/M)\underline{1}^T \in [0, 1]^M \tag{3.85}$$

where $\underline{1}$ is the column vector of M ones. It can be used along with the transition probability matrix H from (3.84) to calculate the states' probability distribution vector as

$$\pi_n = \pi_0 H^n \in [0, 1]^M. \tag{3.86}$$

The states' probability distribution vector enables the derivation of the output's first-moment statistics. The expected value of Z_n is calculated as

$$\mathbb{E}[Z_n] = \sum_{z \in \{\pm 1\}} z P_r(Z_n = z) = (-1)\pi_n e_a^T + \pi_n e_b^T, \tag{3.87}$$

where $e_a = \sum_{i=1}^{M/2} e_i \in \mathbb{R}^M$ and $e_b = \sum_{i=M/2+1}^{M} e_i \in \mathbb{R}^M$, with $e_i = \Big[0, ...0, 1, 0, .., 0\Big] \in \mathbb{R}^M$ being the i-th standard vector. The result of (3.86) along with (3.87) can be used to derive the expected value of the time-average as

$$\mathbb{E}[\widetilde{Z}_N] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[Z_n] = \frac{1}{N}\sum_{n=1}^{N}\Big((-1)\pi_n e_a^T + \pi_n e_b^T\Big) = \frac{1}{N}\pi_0\left(\sum_{n=1}^{N} H^n\right)\Big((-1)e_a^T + e_b^T\Big). \quad (3.88)$$

In Fig. 3.16, the expected value of the time-average, $\mathbb{E}[\widetilde{Z}_N]$, is plotted using (3.88), for increasing values of the sequence length $N = 1, 2, \ldots, 1000$, number of states $M = 32$ and input probability values $p_U^1 = 0.1, p_U^2 = 0.2, p_U^3 = 0.3$. It can be observed that the proposed SCSD adder converges fast to the sum of the inputs, namely after $N = 100$ clock cycles.

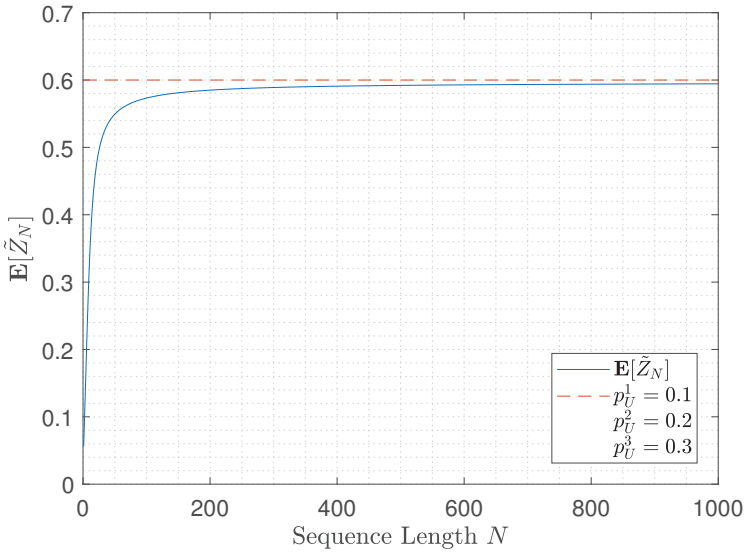


Figure 3.16: Expected value of the output's time-average, $\mathbb{E}[\widetilde{Z}_N]$, calculated using (3.88), estimating the sums of three inputs with probability values $p_U^1 = 0.1, p_U^2 = 0.2, p_U^3 = 0.3$, as the sequence length increases $N = 1, \ldots, 1000$.

4 Statistical Properties Of Stochastic Finite-State Machines

In this chapter [1] , a general methodology to derive analytically the statistical properties of Stochastic Computing Finite-State Machines (SFSM) is introduced. The SFSMs, expressed as Moore ones, are modeled using Markov Chains, enabling the derivation in closed form of their output sequences' statistical properties, including their expected value, their auto- & cross-correlation, their auto- & cross-covariance, their variance and standard deviation as well as their mean squared error. A MC overflow/underflow probability model accompanies the methodology, allowing to calculate analytically the expected number of steps before overflows/underflows, setting the guidelines to select the register's size that reduces erroneous bits originating from them. In the proposed methodology both the input sequence length and the number of the SFSMs' states are considered as parameters, accelerating the overall design procedure as the necessity for multiple time-consuming numerical simulations is eliminated. The proposed methodology's accurate modeling capabilities are demonstrated with its application in SFSMs selected from the SC literature, while comparisons with the numerical experiments justify its correctness.

4.1 Finite-State Machines in Stochastic Computing

The concept of using SFSMs to approximate non-linear functions such as the *tanh*, the *exponential* etc. was introduced in [15]. For the approximations to be feasible, the SFSMs should satisfy the following conditions according to [15]: 1) they have a finite number of ordered states with the first and last one being saturating, meaning that they cannot be exceeded; 2) the transitions within their states are driven by input sequences, with stochastic properties and finite length; and 3) each state communicates with the rest ones. These conditions allow for the operation of a SFSM to be described as an ergodic Markov Chain (MC), enabling the synthesis of functions based on simple logical operations between the states' probabilities[15].

Despite the SFSMs' multiple advantages, they also come with their own weaknesses [15]. In [15], it is mentioned that SFSMs introduce correlations among the bits of the output sequence, which is reasonable

given the memory elements required to implement the state machines [15]. However, the calculation of the output's auto-correlation is estimated with numerical experiments[15]. This is also the case for the SFSM's output that approximates the given function, in which, two important factors contribute as well: 1) the number of states and 2) the input sequence length.

The SFSM analysis of [15], is further extended in [43]. Specifically, in [43], MCs are used to formally prove the principle of operation of several SC-based non-linear functions, including the *exponential*, the *tanh* etc. [43]. A fault-tolerance analysis with respect to bit-flips is also considered in [43]. Nevertheless, the SFSM's statistical properties are not investigated.

Stochastic sequence correlation is often caused at the input as discussed in [9, 4, 39, 57]; the binary-to-stochastic number converters share a common random number source. This allows for certain arithmetic operations to be realized more efficiently as shown in [9, 4, 39, 57], at the cost of increased correlation between the input and the output sequences. The use of a de-correlator unit composed of D Flip-Flops to reduce correlations is mentioned in [39], but, the analysis is supported by numerical experiments.

With respect to the SFSM's output auto-correlation, it is only investigated in [13]. Its calculation, however, faces modeling difficulties when joint distributions are required and thus it is limited to approximations [13]. The variance in multi-stage SC circuits is analyzed in [59]. Yet, it is approached from a gate-level perspective, without further investigation in SFSMs.

Motivated by the needs for an in-depth understanding of the SFSMs' statistical properties, in this work we introduce a mathematical framework for their detailed analysis based on MCs. It is a general methodology, in the sense that it can be applied to any SFSM modeled as a MC. The major contribution of this manuscript is the analytical calculation using closed-formed expressions of the following quantities in a SFSM:

- The expected value of the output and the output's mean.
- The auto-correlation and auto-covariance of the output.
- The cross-correlation and cross-covariance of the output with the inputs.
- The variance & the standard deviation of the output's mean.
- The mean squared error of the output's mean.
- The probability of overflows and underflows in the saturating states.
- The expected number of steps before overflows and underflows, used to select the number of states of the SFSM balancing the computational accuracy hardware trade-off.

Once applied to a SFSM, the proposed framework can be an effective tool to: 1) evaluate the correctness of the SFSM's output when approximating a given function; 2) measure the correlation among the bits in the output sequence and to what extent it affects further operations (e.g. multiplication) of the output with itself and the inputs; 3) calculate the expected accuracy of the SFSM's output and to compare it with the experimental numerical results; and 4) select the register's size that balances computational accuracy compared to hardware resources. A further advantage of the proposed framework is that it considers as

parameters both the input sequence length and the number of states, which is of utter importance for the modeling of SFSMs; it eliminates the necessity for multiple time-consuming parametric simulations to derive the statistical properties and the register's size, thereby accelerating their design & modeling procedure.

4.2 Stochastic Finite State Machines & Markov Chain Modeling

In this section, the methodology to model SFSMs as Markov Chains is presented.

4.2.1 Stochastic Finite State Machines

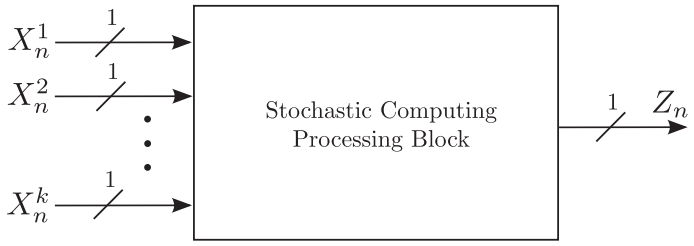


Figure 4.1: A multi-input single-output stochastic computing processing block.

From a system-level perspective, a SC processing block (SCPB) is represented by the abstract model of Fig. 4.1. Typically in SC, it can describe the operation of

1. a combinational logic expression,

2. a sequential logic expression,

3. a higher-level architecture, containing both of the previous processing elements.

Therefore, a SCPB can have many stochastic input sequences $\{X_n^j\}, j = 1, \ldots, k$, each one with probability $X^j = P_r(X_n^j = 1)$, while $\{Z_n\}$ is the output sequence.

The realization of sequential logic circuits and high-level architectures requires memory elements. This means that the SCPB must have a set of internal states $\mathcal{T}_R \triangleq \{0, 1, 2, ..., W-1\}$, where W is the number of states. When counters are used in SC, it is important to note that they may saturate. Assume for example that the states are linearly ordered, i.e. $0 < 1 < 2 <, \ldots, < W-1$ and the goal of the SCPB is to capture an operation of the form $T_n = T_{n-1} + f(X_n^1, \ldots, X_n^k)$, where T_n is the current state. State T_n is constrained in $\mathcal{T}_R$, i.e., within 0 and $W-1$ and what is (typically) realized by the SCPB is the state update process

$$T_n = \max\left\{\min\left\{T_{n-1} + f(X_n^1, \ldots, X_n^k), W-1\right\}, 0\right\}. \tag{4.1}$$

To provide with better insight on the state update, we proceed with the following example. Assume 3 i.i.d. input sequences $\{X_n^1\}, \{X_n^2\}, \{X_n^3\}$ and the function

$$f(X_n^1, X_n^2, X_n^3) = \text{AND}\left(X_n^1, X_n^2, X_n^3\right) - \text{AND}\left(\overline{X_n^1}, \overline{X_n^2}, \overline{X_n^3}\right)$$

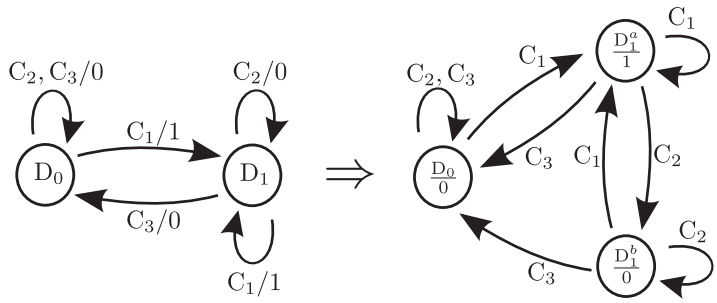
where $\overline{X_n^j} = 1 - X_n^j$. Considering the above, T_n increases its value by 1-bit when and only when all three inputs are simultaneously 1, i.e. $X_n^1 = X_n^2 = X_n^3 = 1$, decreases its value by 1-bit when $X_n^1 = X_n^2 = X_n^3 = 0$ and maintains its previous value otherwise, i.e. $T_n = T_{n-1}$.

The purpose of the counter's register in the previous example, is to remember the cases where all three inputs are 1, so as to "balance" them with the cases where all three inputs are 0.

With respect to the output, Z_n is determined according to the SCPB's operation. In the simplest case of combinational logic, Z_n is straightforward. However, in the case where the SCPB describes a sequential logic circuit or a higher-level architecture, then FSMs are utilized. Therefore, the SCPB's operation can be described using a stochastic FSM (SFSM) and consequently be modeled as a Markov Chain (MC), allowing for the exploration of its long-term stochastic dynamics and the calculation of its statistical properties.

4.2.2 Markov Chain Modeling of a Stochastic FSM

A SFSM expresses a behavior that falls into the category of either a Mealy or a Moore FSM. The former implies that the current output Z_n is a function of the inputs and the state, whereas the latter implies that Z_n is determined solely by the current state. Although the conversion from one FSM behavior to another is a feasible and standard task [60], as shown with the example in Fig. 4.2, here we consider only Moore-based FSMs. This is because relating the current state to the output only, makes the mathematical modeling, analysis and design of SFSMs using MCs more tractable.



$$\mathrm{C}_1 = P_r(X_n^1 = 1, X_n^2 = 1) \quad \mathrm{C}_2 = P_r(X_n^1 = 1 \oplus X_n^2 = 1) \quad \mathrm{C}_3 = P_r(X_n^1 = 0, X_n^2 = 0)$$

Figure 4.2: Conversion example of a stochastic Mealy (left) to Moore (right) FSM. State D_1 in the Mealy is separated into two states in the Moore D_1^a, D_1^b outputting 1 and 0 respectively. In this example, transition probabilities C_1, C_2, C_3, are arbitrary selected, but, determined by two stochastic input sequences $\{X_n^1\}$,$\{X_n^2\}$.

A SFSM can be described by a MC model, with an example shown in Fig. 4.3. The MC of Fig. 4.3 is used as reference to explain the modeling procedure of a SFSM, but, note that *any* MC can be used.

The MC of Fig. 4.3 has a total of M states and its current state, S_n, takes values within the set

$$\mathcal{S} \triangleq \{0, 1, 2, \ldots, M-2, M-1\}. \tag{4.2}$$

Therefore, the MC's current state S_n, is described as a function of the previous state and the inputs, i.e. $S_n = F(S_{n-1}, X_n^j)$ and thus the output is a function of the state, i.e. $Z_n = G(S_n)$.

Considering that a counter is used as memory element, there is a difference between the counter's total number of states, W, and the MC's number of states, M; the MC has *at least* as many states as the counter has, i.e. $M \geq W$, meaning that the mapping from the MC states to those of the counter is surjective, but, not necessarily injective. This can be based on many factors, such as the conversion from a Mealy SFSM behavior to a Moore one, the counter's register type, for instance a shift-register, the SCPB's number of inputs etc.

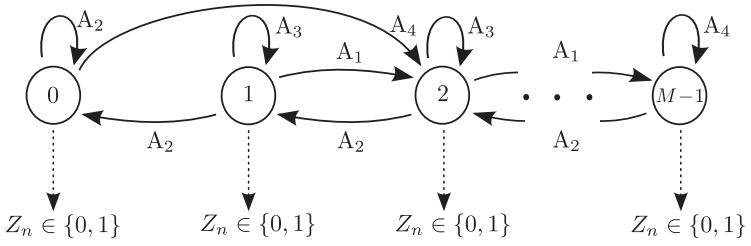


Figure 4.3: Example of a Markov Chain model describing the operation of a stochastic FSM. Transition probabilities A_j are defined by a boolean function and determine the state's transition (see example below). The output Z_n is related to the current state, expressing the FSM's behavior as a Moore one, outputting 0 or 1.

Proceeding to the MC's behavior and assuming that transitions occur from a state σ_i to any other one σ_j, with $\sigma_i, \sigma_j \in \mathcal{S}$, then the $(M \times M)$ transition probability matrix is defined as $H \triangleq [P_r(S_n = \sigma_j | S_{n-1} = \sigma_i)]$. Considering that the transitions from one state σ_i to another σ_j are determined by the inputs $X_n^1, X_n^2, \ldots, X_n^k$ of the SCPB, then the transition probabilities $A_j, j = 1, \ldots, l$ could be any boolean function, such as AND, OR, XOR etc, as

$$A_j = P_r\left(f_j(X_n^1, X_n^2, \ldots, X_n^k)\right). \tag{4.3}$$

To further explain how A_j are determined, consider the following example. Suppose that the MC's state S_{n-1} at time index $n-1$, transitions as follows

- If $X_n^1 = X_n^2 = 1$ and $S_{n-1} > 0$, then $S_n = S_{n-1} + 1$,
- If $X_n^1 = X_n^2 = 0$ and $S_{n-1} > 0$, then $S_n = S_{n-1} - 1$,
- If XOR$(X_n^1, X_n^2) = 1$ and $S_{n-1} > 0$, then $S_n = S_{n-1}$,
- If OR(X_n^1, X_n^2)=1 and $S_{n-1} = 0$, then $S_n = S_{n-1} + 2$,
- If OR(X_n^1, X_n^2)=1 and $S_{n-1} = M-1$, then $S_n = S_{n-1}$.

Based on the above, the transition probabilities can be described as $A_1 = P_r(\text{AND}(X_n^1, X_n^2) = 1)$, $A_2 = P_r(\text{NOR}(X_n^1, X_n^2) = 1)$ and $A_3 = P_r(\text{XOR}(X_n^1, X_n^2) = 1)$. They can be used along with the MC model of Fig. 4.3 and the state ordering $(0, 1, \ldots, M-1)$, to express H as in (4.4), where $A_4 = A_1 + A_3$.

$$H = \begin{bmatrix} A_2 & 0 & A_4 & \ldots & \ldots & 0 \\ A_2 & A_3 & A_1 & 0 & \ldots & 0 \\ 0 & A_2 & A_3 & A_1 & \ldots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ldots & 0 & A_2 & A_3 & A_1 \\ 0 & \ldots & \ldots & 0 & A_2 & A_4 \end{bmatrix}. \tag{4.4}$$

Note that since H is stochastic, it satisfies $\sum_{j=1}^{M} H_{i,j} = 1$, where (i, j) represents the i-th row and j-th column of the matrix H. The probability distribution vector of state S_n, is defined as

$$p_n^T \triangleq \begin{bmatrix} P_r(S_n = 0) \\ P_r(S_n = 1) \\ P_r(S_n = 2) \\ \vdots \\ P_r(S_n = M-1) \end{bmatrix} \in [0, 1]^M \tag{4.5}$$

and for $n = 1, 2, \ldots, N$ steps it is derived as

$$p_n = p_0 H^n \in [0, 1]^M. \tag{4.6}$$

Here, p_0 denotes the initial distribution vector representing the starting state of the MC, S_0, which can take any value within $\mathcal{S}$. It is expressed as

$$p_0 = e_i \in [0, 1]^M, \tag{4.7}$$

where $e_i = [0, \ldots, 1, \ldots, 0] \in \mathbb{R}^M$ is the i-th standard vector.

Before we proceed with the analysis in the next section, it is important to note that we consider only MCs that are *irreducible*; they have the property that starting from any state σ_i, it is possible to transition to any other one σ_j, regardless of the number of transition steps.

4.3 Statistical Modeling of Stochastic FSMs

In this section, we use the MC model and its principles to derive analytically the statistical properties of SFSMs.

4.3.1 Expected Value

To derive the first-moment statistics, one can observe first from the MC model of Fig. 4.3, that Z_n is related to the state only; each state outputs either 0 or 1, based on the SFSM's operation. It is convenient therefore, to partition $\mathcal{S}$ into two subsets $\mathcal{S}_1$ and $\mathcal{S}_0$, such that $\mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_0$, $\mathcal{S}_1 \cap \mathcal{S}_0 = \{\}$, where $S_n \in \mathcal{S}_1 \Rightarrow Z_n = 1$ and $S_n \in \mathcal{S}_0 \Rightarrow Z_n = 0$.

Considering the above and also the equations describing the MC (4.4), (4.6) and (4.7), the expected value of the instantaneous output Z_n is calculated as

$$\mathbb{E}[Z_n] = P_r(Z_n = 1) = P_r\left(S_n \in \mathcal{S}_1\right) = p_0 H^n q^T, \tag{4.8}$$

with $q \in \mathbb{R}^M$ defined as

$$q \triangleq \sum_{i \in \mathcal{S}_1} e_i, \tag{4.9}$$

where q represents the set of states outputting 1. To give a better intuition behind the calculation of (4.8) and the definition of q in (4.9), suppose that the states 0 and 1 are the only ones outputting 1. Then $\mathcal{S}$ is partitioned into $\mathcal{S}_1 = \{0, 1\}$ and $\mathcal{S}_0 = \{2, \ldots, M-1\}$ and thus $q = [1, 1, 0, \ldots, 0]$.

The average of the N-bit output sequence is

$$\widetilde{Z}_N = \frac{1}{N}\Big(Z_1 + Z_2 + \cdots + Z_N\Big), \tag{4.10}$$

and using (4.8) its expected value is calculated as

$$\mathbb{E}[\widetilde{Z}_N] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[Z_n] = \frac{1}{N} p_0 \left(\sum_{n=1}^{N} H^n\right) q^T. \tag{4.11}$$

Both $\mathbb{E}[Z_n]$ and $\mathbb{E}[\widetilde{Z}_N]$ are also essential in the calculation of the second-moment statistics in the following subsection.

4.3.2 Auto-Correlation & Covariance

The auto-correlation of the output $\{Z_n\}$ for time lag $r \geq 0$ is

$$\begin{aligned} R_Z(n+r, n) \triangleq & \mathbb{E}[Z_{n+r} Z_n] = P_r(Z_{n+r} = 1, Z_n = 1) = \sum_{j_1, j_2 \in \mathcal{S}_1} P_r(S_{n+r} = j_2, S_n = j_1) \\ = & \sum_{j_1, j_2 \in \mathcal{S}_1} P_r(S_n = j_1) P_r(S_{n+r} = j_2 | S_n = j_1) = \sum_{j_1, j_2 \in \mathcal{S}_1} (p_0 H^n e_{j_1}^T)(e_{j_1} H^r e_{j_2}^T) \\ = & p_0 H^n Q H^r q^T, \end{aligned} \tag{4.12}$$

where q is given by (4.9) and $Q \in \mathbb{R}^{M\times M}$ is

$$Q \triangleq \sum_{j_1\in\mathcal{S}_1} e_{j_1}^T e_{j_1} = \text{diag}(q). \tag{4.13}$$

The auto-covariance of the output $\{Z_n\}$ is calculated using (4.8) and (4.12) as

$$\begin{aligned} C_Z(n+r,n) &\triangleq \mathbb{E}\big[(Z_{n+r}-\mathbb{E}[Z_{n+r}])(Z_n-\mathbb{E}[Z_n])\big] \\ &= R_Z(n+r,n) - \mathbb{E}[Z_{n+r}]\mathbb{E}[Z_n] \\ &= p_0H^nQH^rq^T - p_0H^{n+r}q^Tp_0H^nq^T. \end{aligned} \tag{4.14}$$

4.3.3 Cross-Correlation & Covariance

We recall that Z_n and S_n depend only on $\{X_n^j\}$, $j = 1,2,\ldots,k$ and not on their future values. Moreover, the random variables of the input sequences $\{X_n^j\}$ are assumed to be independent to each other, since they originate from different random number sources. To this end, we derive the cross-correlation of the output $\{Z_n\}$ with the a single input $\{X_n\}$ as

$$R_{ZX}(n,n+r) \triangleq \mathbb{E}[Z_nX_{n+r}]=P_r(Z_n=1,X_{n+r}=1) \tag{4.15}$$

To proceed further, we distinguish cases for r,

- For $r=0$,

$$\begin{aligned} R_{ZX}(n,n) =& P_r(Z_n=1,X_n=1) = \sum_{\sigma\in\mathcal{S}} P_r(Z_n=1,X_n=1,S_{n-1}=\sigma) \\ &= \sum_{\substack{\sigma\in\mathcal{S},\\ \sigma_1\in\mathcal{S}_1}} P_r(S_n=\sigma_1,X_n=1,S_{n-1}=\sigma) \\ &= \sum_{\substack{\sigma\in\mathcal{S},\\ \sigma_1\in\mathcal{S}_1}} P_r(S_n=\sigma_1 \mid X_n=1,S_{n-1}=\sigma)P_r(X_n=1)P_r(S_{n-1}=\sigma) \\ &= p_0H^{n-1}(H\circ V)q^TP_r(X_n=1), \end{aligned} \tag{4.16}$$

  where matrix $H\circ V$ is the point-wise (Hadamard) product of H with V, where $V\in\{0,1\}^{M\times M}$ is such that $v_i,j=1$ if and only if the transition from the i-th to the j-th state is done with $X_n=1$.

- For $r\geq 1$, since Z_n and X_{n+r} are independent we have

$$R_{ZX}(n,n+r) = P_r(Z_n=1)P_r(X_{n+r}=1) = p_0H^nq^TP_r(X_{n+r}=1). \tag{4.17}$$

Summarizing,

$$R_{ZX}(n, n+r) = \begin{cases} p_0 H^{n-1}(H \circ V) q^T P_r(X_n = 1), & r = 0 \\ p_0 H^n q^T P_r(X_{n+r} = 1), & r > 0 \end{cases} \tag{4.18}$$

The cross-covariance between the output $\{Z_n\}$ and the input $\{X_n\}$ sequences

$$C_{ZX}(n, n+r) = R_{ZX}(n, n+r) - \mathbb{E}[Z_n]\mathbb{E}[X_{n+r}], \tag{4.19}$$

is derived directly from (4.8) and (4.18) and the definition $X = P_r(X_n = 1)$ giving

$$C_{ZX}(n, n+r) = \begin{cases} X p_0 H^{n-1}(H \circ V - H) q^T, & r = 0 \\ 0, & r > 0 \end{cases} \tag{4.20}$$

4.3.4 Variance and Standard Deviation

The variance of $\tilde{Z}_N$ from (4.10) is calculated using the expression (4.14) as follows

$$\begin{aligned} \text{Var}(\tilde{Z}_N) =& \mathbb{E}\big[(\tilde{Z}_N - \mathbb{E}[\tilde{Z}_N])^2\big] = \frac{1}{N^2} \sum_{i,j=1}^{N} \mathbb{E}\big[(Z_i - \mathbb{E}[Z_i])(Z_j - \mathbb{E}[Z_j])\big] \\ =& \frac{1}{N^2} \sum_{i,j=1}^{N} C_Z(i,j) = \frac{1}{N^2} \left[\sum_{i=1}^{N} C_Z(i,i) + 2 \sum_{i>j}^{N} C_Z(i,j) \right] \\ =& \frac{1}{N^2} \Bigg[p_0 \sum_{i=1}^{N} H^i Q q^T - \sum_{i=1}^{N} \left(p_0 H^i q^T\right)^2 + 2 \Bigg(\sum_{j=1}^{N-1} \sum_{i=j+1}^{N} p_0 H^j Q H^{(i-j)} q^T \\ & - \sum_{j=1}^{N-1} \sum_{i=j+1}^{N} \left(p_0 H^i q^T\right) \left(p_0 H^j q^T\right) \Bigg) \Bigg], \end{aligned} \tag{4.21}$$

while the standard deviation is obtained as $\sigma_{\tilde{Z}_N} = \sqrt{\text{Var}(\tilde{Z}_N)}$.

4.3.5 Mean Squared Error Analysis

To investigate the output accuracy of a SFSM, one can calculate analytically the Mean Squared Error (MSE) between the output's mean value $\tilde{Z}_N$ and the actual value of the computation $\bar{Z}$. It is calculated as

$$\begin{aligned} \text{MSE}(\tilde{Z}_N) &= \mathbb{E}\left[(\tilde{Z}_N - \bar{Z})^2\right] = \mathbb{E}\left[\tilde{Z}_N^2 - 2\tilde{Z}_N \bar{Z} + \bar{Z}^2\right] = \mathbb{E}\left[\tilde{Z}_N^2\right] - 2\mathbb{E}[\tilde{Z}_N]\bar{Z} + \bar{Z}^2 \\ &= \text{Var}(\tilde{Z}_N) + \mathbb{E}[\tilde{Z}_N]^2 - 2\mathbb{E}[\tilde{Z}_N]\bar{Z} + \bar{Z}^2, \end{aligned} \tag{4.22}$$

where the analytical expressions (4.11) and (4.21) are used.

4.4 Number of States Selection & Register Size Estimation

The registers utilized by the SCPBs are typically used to store and "remember" logic 1s based upon a counting process. Ideally, with finite input sequence length and infinite number of states, the counting is performed perfectly, i.e. without loss of 1s. In practice, however, this is not feasible given the register's finite number of states; if they are too few, the counting process results in overflows or underflows that may degrade the output's accuracy.

Consider the following scenario: starting from any initial state of the register, e.g. $T_0 \in \mathcal{T}_R$, the SCPB's inputs are such that they force T_n to perform a walk within states $0, \ldots, W-1$. Eventually, T_n will reach either of its saturating states $W-1$ or 0 and may visit them repeatedly. This can cause overflows or underflows given that states $W-1$ and 0 cannot be exceeded to allow for further counting and correctly storing of logic 1s. Therefore, it is important to investigate how the number of states W are related to overflows/underflows and when this impacts the accuracy of the output sequence.

4.4.1 Stochastic Finite-State Machine Overflow/Underflow Modeling

To explain the modeling procedure of overflows/underflows, consider the MC of Fig. 4.3 and suppose that its current state S_n is $M-1$ (or 0). The *overflows/underflows* occur when and only when the next combination of inputs at time index $n+1$, force the MC's state to return to itself, i.e. $S_{n+1} = S_n$, where it should transition to $S_{n+1} = S_n + 1$ or $S_{n+1} = S_n - 1$ instead. However, states M and -1 do not exist and as expected, the MC of Fig. 4.3 does not allow for overflows/underflows to be modeled. Therefore, we modify it to the one shown in Fig. 4.4 which contains two extra *absorbing* states M_a, M_b so as to capture the overflows/underflows. Note that both states M_a, M_b are used for modeling purposes only and do not imply any change of the register's states or size.

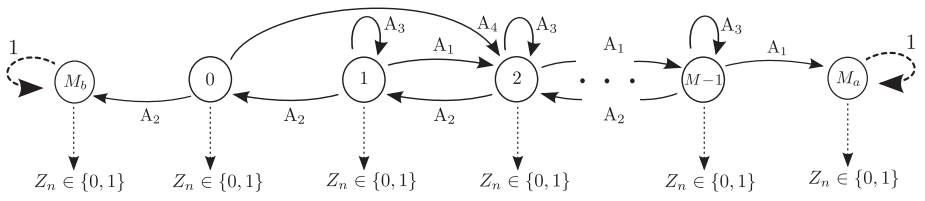


Figure 4.4: Example of the Markov Chain overflow/underflow model with absorbing states M_a, M_b corresponding to that of Fig. 4.3.

Based on the MC model of Fig. 4.4 one can calculate the *probability* of overflows/underflows in states M_a, M_b. First, the set of the MC's states is defined as $\hat{\mathcal{S}} \triangleq \{0, 1, 2, ..., M-1, M_a, M_b\}$ and assuming a state order $(0, 1, 2, ..., M-1, M_a, M_b)$ then the transition probability matrix $\hat{H} \in [0,1]^{(M+2)\times(M+2)}$ is

written as

$$\hat{H} = \begin{bmatrix} 0 & 0 & A_4 & \dots & \dots & 0 & 0 & A_2 \\ A_2 & A_3 & A_1 & 0 & \dots & 0 & 0 & A_4 \\ 0 & A_2 & A_3 & A_1 & \dots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & A_2 & A_3 & A_1 & 0 & 0 \\ 0 & \dots & \dots & 0 & A_2 & A_3 & A_1 & 0 \\ 0 & \dots & \dots & 0 & 0 & 0 & 1 & 0 \\ 0 & \dots & \dots & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.23}$$

The MC's current state $\hat{S}_n$ probability distribution vector is

$$\hat{p}_n^T \triangleq \begin{bmatrix} P_r(\hat{S}_n = 0) \\ P_r(\hat{S}_n = 1) \\ \vdots \\ P_r(\hat{S}_n = M-1) \\ P_r(\hat{S}_n = M_a) \\ P_r(\hat{S}_n = M_b) \end{bmatrix} \in [0,1]^{M+2} \tag{4.24}$$

and is calculated as

$$\hat{p}_n = \hat{p}_0 \hat{H}^n, \tag{4.25}$$

with initial distribution vector

$$\hat{p}_0 = e_i \in [0,1]^{M+2}. \tag{4.26}$$

Considering the above, the probability that the MC has overflowed/underflowed by clock cycle n in states M_a and M_b is $P_r(\hat{S}_n = M_a)$ and $P_r(\hat{S}_n = M_b)$ respectively and is calculated as

$$\Big[P_r(\hat{S}_n = M_a), P_r(\hat{S}_n = M_b)\Big] = \hat{p}_0 \hat{H}^n [e_{M+1}^T, e_{M+2}^T]. \tag{4.27}$$

4.4.2 Expected number of Steps before Overflows/Underflows

It is reasonable to further investigate the overflow/underflow process, especially when the operation of the SFSM and consequently that of the SCPB restrains their occurrence. For this reason, we calculate the expected number of transitions *before the first overflow/underflow*, i.e. before states M_a or M_b are reached. We write matrix $\hat{H}$ in its canonical form [25, 61] as

$$\hat{H} = \left[\begin{array}{c|c} \tilde{H} & R \\ \hline 0_{2,M} & I_2 \end{array}\right], \tag{4.28}$$

where $\tilde{H} \in [0,1]^{M\times M}$, $R \in [0,1]^{M\times 2}$, $I_2 \in [0,1]^{2\times 2}$ and $0_{2,M} \in [0,1]^{2\times M}$. Using $\tilde{H}$, the fundamental matrix of the absorbing MC [25, 61] is calculated as

$$F = (I_M - \tilde{H})^{-1} \in \mathbb{R}^{M\times M}. \tag{4.29}$$

Considering that the initial state, S_0, can be any within the states $\mathcal{S}$ of the MC of Fig. 4.3, then the expected number of transitions *before* the MC is absorbed is

$$N^* = p_0 F \underline{1}, \tag{4.30}$$

where $\underline{1} \in \mathbb{R}^M$ is the column vector of all ones and p_0 is given by (4.7). The potentially negative impact of overflows/underflows and the importance of N^* in the register's state selection, is discussed in the following subsection.

4.4.3 Guidelines to select the number of states

According to the SCPB's operation and the counting process itself, an overflow/underflow does not always result in an erroneous bit at the output. To give a better insight on this, we consider two cases for a MC with a finite number of states M:

- The MC's current state S_n starts from the initial state $S_0 = 0$, transitions within $\mathcal{S}$, and *is allowed* to transition to its saturating states, visiting them repeatedly as well. Typically in SC, such MC describes the operation of a SFSM that approximates an asymptotically bounded function and actually benefits from the overflows/underflows, for instance the stochastic *tanh* [15].

- The MC's current state S_n starts from the initial state $S_0 = 0$, transitions within $\mathcal{S}$, but, *is not allowed* to repeatedly visit the last state, $M-1$ which is a saturating one. Such MC describes the operation of a SFSM that captures the bit-differences from the input sequences and stores them cumulatively in a register, but, does not benefit from overflows as they may result in erroneous bits at the output sequence [80, 81].

From the above, it is reasonable for a SFSM to have the number of its states carefully selected so as to limit the use of registers taxing on the hardware resources. In this direction, one can use the expression of N^* in (4.30) as a guideline to select M and hence the register's size. First, one has to select the stochastic sequence length N, the number of states M and the input probabilities $X^j = P_r(X_n^j = 1), j = 1, \ldots, k$. Since N^* is a function of the inputs and the number of states $M = 2^w$, a reasonable register's size $\hat{w}$ can be selected

$$\hat{w} = \min\left\{ w \in \mathbb{N} \mid \min_{(X^1,\ldots,X^k)} N^*(X^1,\ldots,X^k, 2^w) \geq N \right\}. \tag{4.31}$$

4.5 Modeling Examples

In this section we show how the the proposed MC framework can be applied to model in detail the statistical properties of two SFSMs, selected from the SC literature. To demonstrate our framework's accurate modeling, we compare its results with those obtained from the numerical calculations for 10^4 runs with i.i.d. inputs, all conducted using Matlab.

4.5.1 Modeling Example 1: Stochastic Tanh

Architecture: The first modeling example we consider is the stochastic tanh function (STanh) introduced in [15]. Its architecture is shown in Fig. 4.5, where $\{X_n\}$ is the i.i.d. input sequence and $\{Z_n\}$ is the output. If $X_n = 1$, then the w-bit register's current value T_n is increased by 1-bit, whereas in the opposite case, i.e. $X_n = 0$, it is decreased by 1-bit.

The up & down counting of T_n occurs within $\mathcal{T}_R \triangleq \{0, 1, \ldots, W-1\}$, where W is the total number of states. Here, the up & down counting is realized using a ripple binary counter, able to count up to $W = 2^w$ states, where w is the register's size, but, note that it can also be realized by a shift-register. The first and last states, 0 and $W-1$, are saturating, which means that they cannot be exceeded. Therefore, considering (4.1), with initial state $T_0 = W/2$, T_n is updated as

$$T_n = \max\left\{\min\left\{T_{n-1} + X_n - \overline{X}_n, W-1\right\}, 0\right\}.$$

The instantaneous value of the output Z_n, is determined by the state's current value as $Z_n = T_n \geq W/2$. According to the analysis in [15] and considering the above, for an input X representing a stochastic number in bipolar format, the configuration shown in Fig. 4.5 approximates the Tanh$(\cdot)$ function as STanh$(W, X) \approx$ Tanh$(XW/2)$.

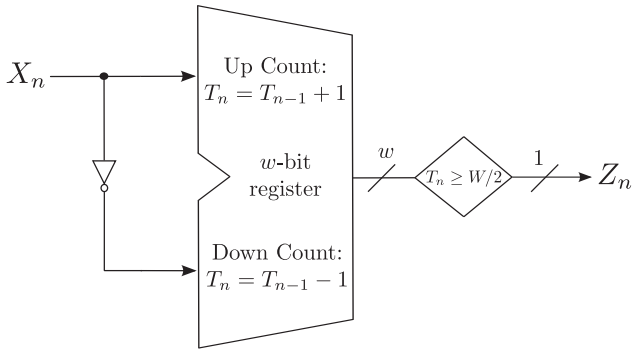


Figure 4.5: Architecture of the stochastic tanh function.

Markov Chain Modeling: The operation of the STanh architecture shown in Fig. 4.5 can be described by the MC model of Fig. 4.6. Its states have an one-to-one correspondence with the register's ones and therefore the MC's current value S_n transitions within $\mathcal{S} = \{0, 1, \ldots, M-1\}$. The transition probabilities

are

$$A_1 = P_r(X_n = 1)$$
$$A_2 = P_r(X_n = 0) = 1 - P_r(X_n = 1) \tag{4.32}$$

and can be used to describe the transition probability matrix $H \in \mathbb{R}^{M\times M}$ as

$$H = \begin{bmatrix} A_2 & A_1 & 0 & \dots & \dots & 0 \\ A_2 & 0 & A_1 & 0 & \dots & 0 \\ 0 & A_2 & 0 & A_1 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ 0 & \dots & 0 & A_2 & 0 & A_1 \\ 0 & \dots & \dots & 0 & A_2 & A_1 \end{bmatrix}. \tag{4.33}$$

Assuming an initial distribution vector $p_0 = e_{M/2} \in \mathbb{R}^M$, the MC's probability distribution vector p_n is calculated using (4.6).

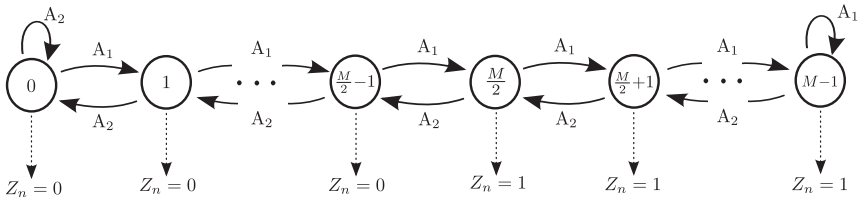


Figure 4.6: Markov Chain model describing the operation of the stochastic tanh function. Transition probabilities are given by (4.32).

First-Moment Statistics: Considering the MC model of Fig. 4.6, $\mathcal{S}$ can be separated into $\mathcal{S}_0 = \{0, \dots, M/2-1\}$ and $\mathcal{S}_1 = \{M/2, \dots, M-1\}$. Therefore, using (4.9), q is expressed as

$$q = \sum_{i=M/2+1}^{M} e_i, \tag{4.34}$$

allowing for $\mathbb{E}[Z_n]$ and $\mathbb{E}[\tilde{Z}_N]$ to be calculated using (4.8) and (4.11) respectively. A graphical representation of $\mathbb{E}[\tilde{Z}_N]$ approximating the Tanh function, is shown in Fig. 4.7 parameterized with $M = 4$ states and $N = 64$-bit sequence length.

Second-Moment Statistics: To derive the second-moment statistics, one can start from the calculation of the auto-correlation $R_Z(n+r, n)$ using (4.12). Note that $Q \in \mathbb{R}^{M\times M}$ is obtained from (4.13), where the vector q is used from (4.34). Once $R_Z(n+r, n)$ is calculated, it can be used to derive the auto-covarinace $C_Z(n+r, n)$ using (4.14). In Fig. 4.8, $C_Z(n+r, n)$ is plotted, for $M = 4$ states, input sequence length $N = 256$ and two time lags $r = 0, 1$. As one can observe, $C_Z(n+r, n)$ peaks when $X = 0$ (bipolar format) and is reduced when the delay is increased from 0 to 1 samples. The variance of the output's mean $\text{Var}(\tilde{Z}_N)$ is calculated using (4.21). In Fig. 4.9 we demonstrate this calculation using $M = 4$ states

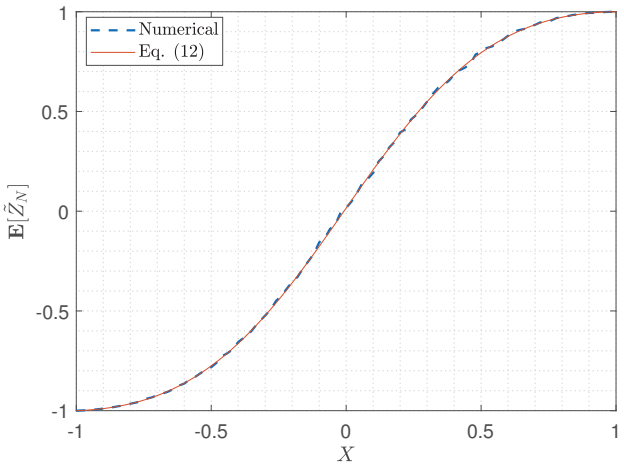


Figure 4.7: Expected value of the stochastic tanh's output mean $\mathbb{E}[\tilde{Z}_N]$ calculated using (4.11), parameterized with $M = 4$ states and sequence length $N = 64$. For the numerical calculations, 10^4 i.i.d. runs for each point are considered.

and input sequence length $N = 64$.

Mean Squared Error: The MSE is calculated using (4.22), in which $\bar{Z} = \text{Tanh}(XM/2)$. The results are shown in Fig. 4.10 for $M = 4$ states and input sequence length $N = 64$.

Overflow/Underflow Modeling: The modeling of overflows/underflows is achieved using the MC model of Fig. 4.11, which contains the two absorbing states M_a, M_b. With state ordering $(0, 1, 2, ..., M-1, M_a, M_b)$ and the transition probabilities from (4.32), the transition probability matrix $\hat{H} \in \mathbb{R}^{(M+2)\times(M+2)}$ becomes

$$\hat{H} = \begin{bmatrix} 0 & A_1 & 0 & \dots & \dots & 0 & 0 & A_2 \\ A_2 & 0 & A_1 & 0 & \dots & 0 & 0 & 0 \\ 0 & A_2 & 0 & A_1 & \dots & 0 & 0 & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots & \vdots & \vdots \\ 0 & \dots & 0 & A_2 & 0 & A_1 & 0 & 0 \\ 0 & \dots & \dots & 0 & A_2 & 0 & A_1 & 0 \\ 0 & \dots & \dots & 0 & 0 & 0 & 1 & 0 \\ 0 & \dots & \dots & 0 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{4.35}$$

Using (4.35), the probability distribution vector $\hat{p}_n$ is calculated from the expression given in (4.25), where the initial distribution vector is $\hat{p}_0 = e_{M/2}$. In addition, Fig. 4.12 shows the probability of overflow/underflow calculated using (4.27) for $X = 0.5$ (unipolar format), sequence length $N = 64$ and increasing number of states $M = 4, \dots, 32$.

Register's size selection: The matrix $\hat{H}$ from (4.35) can be used to derive the fundamental matrix F according to (4.29). Then, the expected number of steps before overflows N^* can be calculated using (4.30), considering that $p_0 = e_{M/2} \in \mathbb{R}^M$. In Fig. 4.13, N^* is plotted, parametrized with sequence

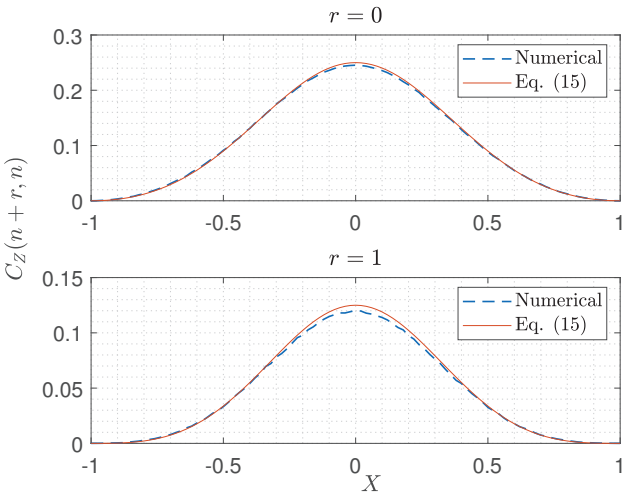


Figure 4.8: Auto-Covariance $C_Z(n+r,n)$ of the stochastic tanh's output calculated using (4.14), parameterized with $M = 4$ states, sequence length $N = 256$ and time lags $r = 0, 1$. For the numerical calculations, 10^4 i.i.d. runs for each point are considered.

length $N = 32$ and state sizes $M = 8, 16, 32$. It can be observed that the condition $N^* \geq N$ from (4.31), is satisfied only when $M = 16, 32$ states are used.

One can conclude that the advantage of modeling the expected number of steps before overflows N^* is twofold; on the one hand, it allows to accurately select the number of states that reduce the overflow/underflow occurrence, while on the other it prevents from selecting an unnecessarily large number of states, taxing on the hardware resources.

4.5.2 Modeling Example 2: Stochastic Adder

Architecture: The second modeling example we consider, is the non-scaling adder introduced in [80]. Its architecture is shown in Fig. 4.14, where $\{X_n^1\}, \{X_n^2\}$ are i.i.d. input sequences and $\{Z_n\}$ is the output. Its principle operation is based upon the storing of logic ones in a w-bit register when $X_n^1 = X_n^2 = 1$ so as to output them in a future clock cycle n' for which $X_{n'}^1 = X_{n'}^2 = 0$. The register's current value T_n, up and down counts within $\mathcal{T}_R = \{0, 1, \ldots, W-1\}$, where $W = 2^w$ is the total number of states. Therefore, T_n's accumulating behavior is expressed as [80]

$$T_n = \min\Big\{T_{n-1} + X_n^1 X_n^2 - \big(T_{n-1} > 0\big)\overline{X}_n^1 \overline{X}_n^2, W-1\Big\}.$$

From the architecture of Fig. 4.14, the instantaneous output can be described as $Z_n = X_n^1 + X_n^2 + T_{n-1} > 0$. Note that for the architecture's proper operation, it holds $0 \leq X^1 + X^2 \leq 1$.

Markov Chain Modeling: The MC model of Fig. 4.15 describes the operation of the adder's architecture. Here, the register's initial value 0 is represented by two states in the model, 0_A and 0_B, so as for its SFSM to be expressed as a Moore one. Hence, its current state S_n transitions within $M + 1$ values

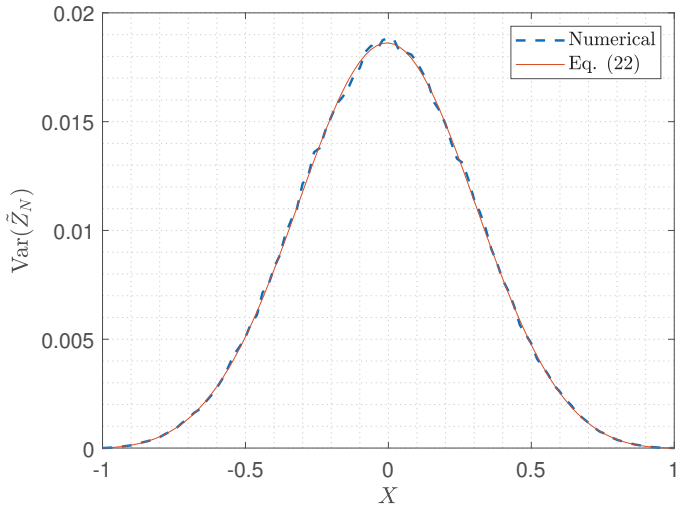


Figure 4.9: Variance $\mathrm{Var}(\tilde{Z}_N)$ of the stochastic tanh's output mean calculated using (4.21), parameterized with $M = 4$ states and sequence length $N = 64$. For the numerical calculations, 10^4 i.i.d. runs for each point are considered.

within $\mathcal{S} = \{0_A, 0_B, 1, 2, ..., M-1\}$, while its transition probabilities are

$$\begin{aligned} A_1 =& P_r(X_n^1 = 0)P_r(X_n^2 = 0) \\ A_2 =& P_r(X_n^1 = 1) + P_r(X_n^2 = 1) - 2P_r(X_n^1 = 1)P_r(X_n^2 = 1) \\ A_3 =& P_r(X_n^1 = 1)P_r(X_n^2 = 1). \end{aligned} \tag{4.36}$$

They can be used to write the transition matrix $H \in \mathbb{R}^{(M+1)\times(M+1)}$ as

$$H = \begin{bmatrix} A_1 & A_2 & A_3 & \dots & \dots & 0 \\ A_1 & A_2 & A_3 & \dots & \dots & 0 \\ 0 & A_1 & A_2 & A_3 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & 0 & A_1 & A_2 & A_3 \\ 0 & \dots & \dots & 0 & A_1 & A_2 + A_3 \end{bmatrix}. \tag{4.37}$$

Since the MC's initial state is $S_0 = 0_A$, here the initial distribution vector is $p_0 = e_1 \in \mathbb{R}^{M+1}$ and thus the states' probability distribution vector p_n is calculated using (4.6) [80].

First-Moment Statistics: Observing the MC model of Fig. 4.15, one can see that $S_n = 0_A \Rightarrow Z_n = 0$, separating $\mathcal{S}$ into $\mathcal{S}_0 = \{0_A\}$ and $\mathcal{S}_1 = \{1, 2, \dots, M-1\}$. Therefore, vector $q = [0, 1, \dots, 1] \in \mathbb{R}^{M+1}$ is expressed as

$$q = \sum_{i=2}^{M+1} e_i, \tag{4.38}$$

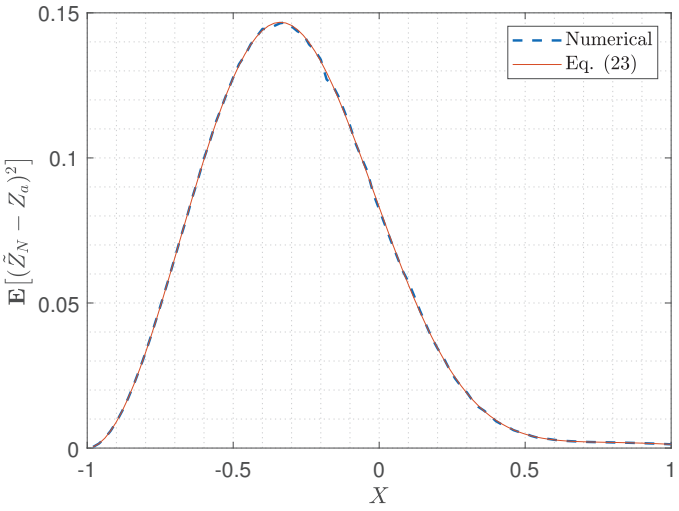


Figure 4.10: Mean Squared Error of the stochastic tanh's output mean calculated using (4.22) for $M = 4$ states and input sequence length $N = 64$. For the numerical calculations, 10^4 i.i.d. runs for each point are considered.

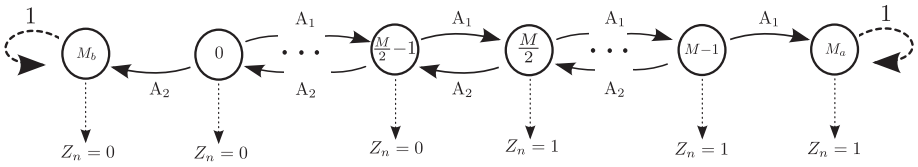


Figure 4.11: Markov Chain overflow/underflow model of the stochastic tanh function. Transition probabilities are given by (4.32).

and can be used to calculate $\mathbb{E}[Z_n]$ and $\mathbb{E}[\tilde{Z}_N]$ using (4.8) and (4.11) respectively [80]. For two inputs $X^1, X^2 \in [0, 1]$, the expected value of the output's mean is shown in Fig. 4.16, parametrized with $M = 8$ states and $N = 64$-bit sequence length. From Fig. 4.16, it can be seen that the distribution of the output's mean, $\mathbb{E}[\tilde{Z}_N]$, calculated using (4.11), matches the one obtained from the numerical experiments, verifying also the correctness of the additions for two inputs $X^1, X^2 \in [0, 1]$, such that $0 \leq X^1 + X^2 \leq 1$.

Second-Moment Statistics: The auto-correlation $R_Z(n + r, n)$ is calculated using (4.12), considering the vector q from (4.38) and can be used to calculate $C_Z(n + r, n)$ from the expression (4.14). The auto-covariance $C_Z(n + r, n)$ is illustrated in Fig. 4.17, for two inputs $X^1, X^2 \in [0, 1]$, parametrized with $M = 8$ states, input sequence length $N = 64$ and a delay $r = 1$. One can observe that the auto-covariance peaks when $X^1 = X^2 = 0.5$ with a negligible value of approximately 0.03 and gradually decreases when moving away from these values. Notice that the results obtained from the analytic calculation, follow the ones from the numerical experiments.

Considering $C_Z(n + r, n)$, the variance of $\tilde{Z}_N$, Var$(\tilde{Z}_N)$, can be calculated using the expression (4.21). In Fig. 4.18 it is demonstrated for $M = 8$ states and input sequence length $N = 64$. From Fig.

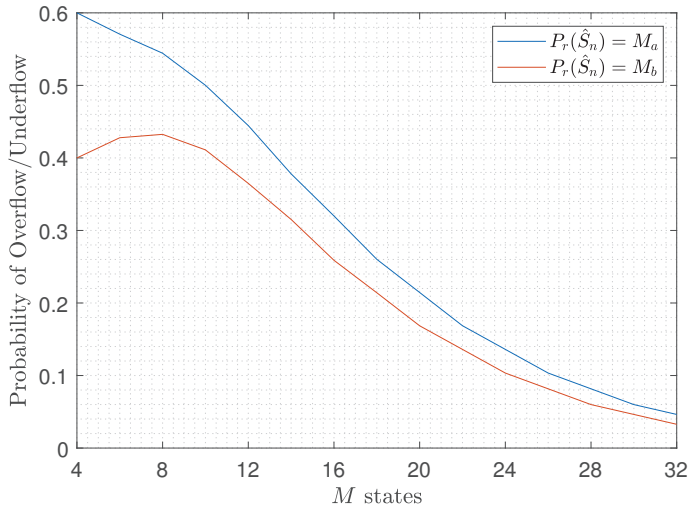


Figure 4.12: Probability of overflow/underflow of the stochastic tanh calculated using (4.27) for increasing number of states $M = 4, \ldots, 32$, input $X = 0.5$ and sequence length $N = 64$.

4.18, it is observed that the analytic calculation of $\text{Var}(\tilde{Z}_N)$ follows closely the one obtained from the numerical experiments, where the results have values with order of magnitude up to 10^{-3}.

Mean Squared Error: The MSE of the adder's output mean can be calculated using (4.22), where $\bar{Z} = X^1 + X^2$. In Fig. 4.19 the MSE is shown for $M = 8$ states, sequence length $N = 64$ and inputs $X^1, X^2 \in [0, 1]$. From Fig. 4.19, it is noticeable that the analytic calculation of the $\text{MSE}(\tilde{Z}_N)$ using (4.22) matches the one obtained from the numerical experiments.

Overflow Modeling: The procedure to model overflows deviates from the previous SFSM example. Here, we are interested in "how far" the MC's current value S_n can transition, corresponding to "how many" additional logic 1s are stored from the counting process. Therefore, the modeling of overflows becomes one-sided, in the sense that only one absorbing state is used, M_a. In Fig. 4.20, the adder's MC overflow model is shown.

With state ordering $(0_A, 0_B, 1, \ldots, M-1, M_a)$, the transition probability matrix $\hat{H} \in \mathbb{R}^{(M+2)\times(M+2)}$ is written using the transition probabilities given in (4.36) as

$$\hat{H} = \begin{bmatrix} A_1 & A_2 & A_3 & \ldots & \ldots & \ldots & 0 \\ A_1 & A_2 & A_3 & \ldots & \ldots & \ldots & 0 \\ 0 & A_1 & A_2 & A_3 & \ldots & \ldots & 0 \\ 0 & \ddots & \ddots & \ddots & \ddots & \ldots & 0 \\ \vdots & \ldots & 0 & A_1 & A_2 & A_3 & 0 \\ \vdots & \ldots & \ldots & 0 & A_1 & A_2 & A_3 \\ 0 & \ldots & \ldots & \ldots & \ldots & 0 & 1 \end{bmatrix}, \tag{4.39}$$

and using $\hat{p}_0 = e_1 \in \mathbb{R}^{M+2}$, the probability distribution vector after N steps is calculated with the

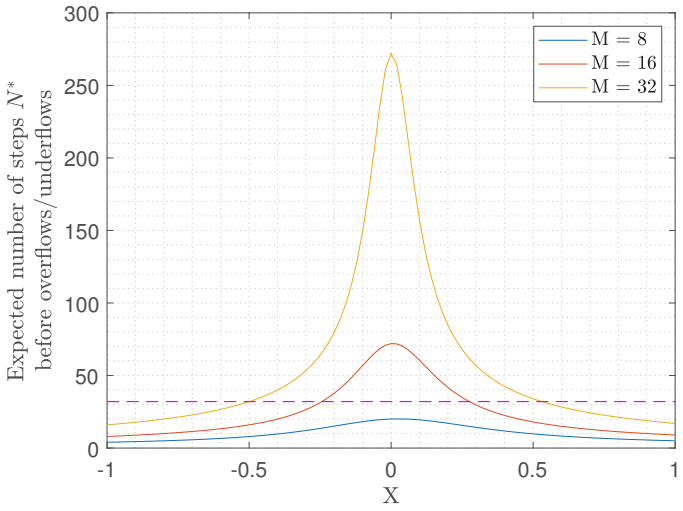


Figure 4.13: Expected number of steps before overflows/underflows N^* of the stochastic tanh calculated using (4.31), for $M = 8, 16, 32$ states and sequence length $N = 32$ (dashed line). The guideline $N^* \geq N$ allows for reduced overflow/underflow occurrence.

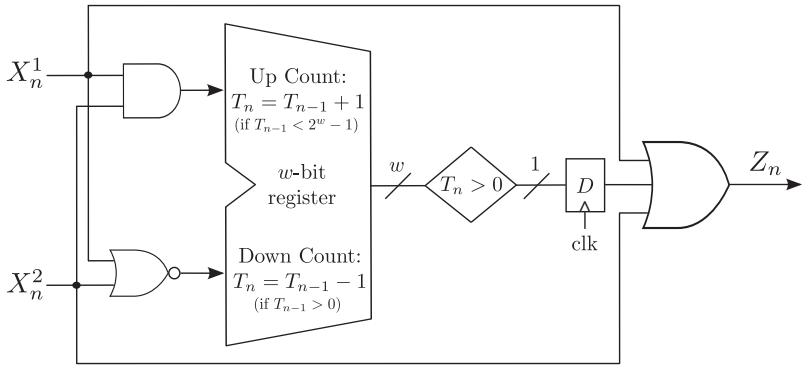


Figure 4.14: Architecture of the stochastic adder [80].

expression (4.25). Considering that only one absorbing state is used, then the probability of overflow is [80]

$$P_r(S_n = M_a) = \hat{p}_0 \hat{H}^n e_{M+2}^T \tag{4.40}$$

In Fig. (4.21), the adder's probability of overflow is graphically illustrated, for inputs with values $X^1 = X^2 = 0.5$, increasing number of states $M = 4, \ldots, 32$ and sequence lengths $N = 16, 32, 64, 256$. As expected, an increase on the number of states, reduces the probability of overflow.

Register's size selection: For the calculation of the expected number of steps before overflows N^*, the matrix $\hat{H}$ from (4.39) is used along with the initial distribution vector $p_0 = e_1 \in \mathbb{R}^{M+1}$. In Fig. 4.22, N^* is plotted for inputs $X^1 = X^2 = 0.5$, increasing number of states $M = 4, \ldots, 32$ and stochastic sequence lengths $N = 16, 32, 64, 128, 256$.

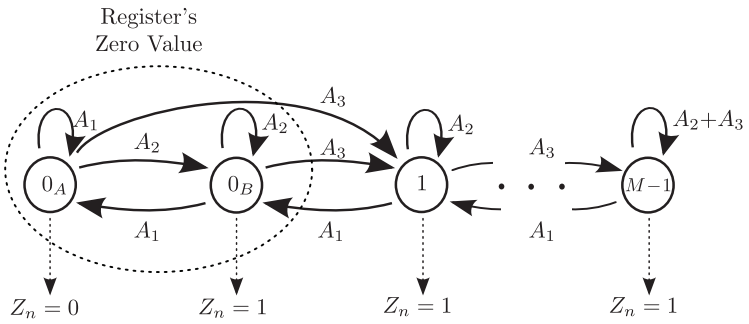


Figure 4.15: Markov Chain model describing the operation of the stochastic adder. Transition probabilities are given by (4.36).

It can be seen from Fig. 4.22 that for small values of N, namely $N = 16, 32$, a slight increase on the number of states has negligible difference on the condition to be satisfied, $N^* \geq N$. However, this is not the case for large values of N, for instance $N = 128$ and more, in which an increase of the number of states and hence the register's size, is necessary to satisfy $N^* \geq N$.

4.5.3 Execution Times Performance

To highlight the time efficiency of our proposed framework in the modeling of the SFSMs' statistical properties, we compare its execution times with those obtained from the numerical experiments. For the STanh, we use 10^2 input values uniformly distributed in $[0, 1]$ and for the Stochastic Adder we use 10^4 input values uniformly distributed in $[0, 1] \times [0, 1]$. Regarding the numerical experiments, we conduct 10^4 and 10^5 runs with i.i.d input sequences of length $N = 64$-bits for each input value we consider. To measure the relative performance we use the speedup metric, which is the ratio of the execution time of the numerical experiments, L_N, over that of the analytical modeling one, L_M, i.e. Speedup $= L_{\mathrm{N}}/L_{\mathrm{M}}$. The execution times are used to calculate the time saving metric as $(L_N - L_M)/L_N * 100\%$. We also cite the Mean Absolute Error (MAE), which is the absolute difference between the averaged output of the numerical experiments for 10^4 and 10^5 runs and the analytical modeling output, summed over all the uniformly distributed input values.

Table 4.1, presents the numerical simulation and analytical calculation execution times of the two SFSMs. When 10^5 runs are used, it is observed that the calculation of the expected value and the autocorrelation using the proposed framework, result in substantial time savings for both the STanh, 99.47% and 99.90% respectively, and the Stochastic Adder, 99.94% and 99.96% respectively. With respect to the calculation of the variance and the MSE, the analytical modeling of our proposed method yields significant time savings, corresponding to 95.07% and 94.93% respectively for the STanh and 92.21% and 91.91% respectively for the Stochastic Adder. Decreasing the number of runs to 10^4, is expected to increase the MAE and the execution time, at the cost however of reducing the numerical experiments' approximations.

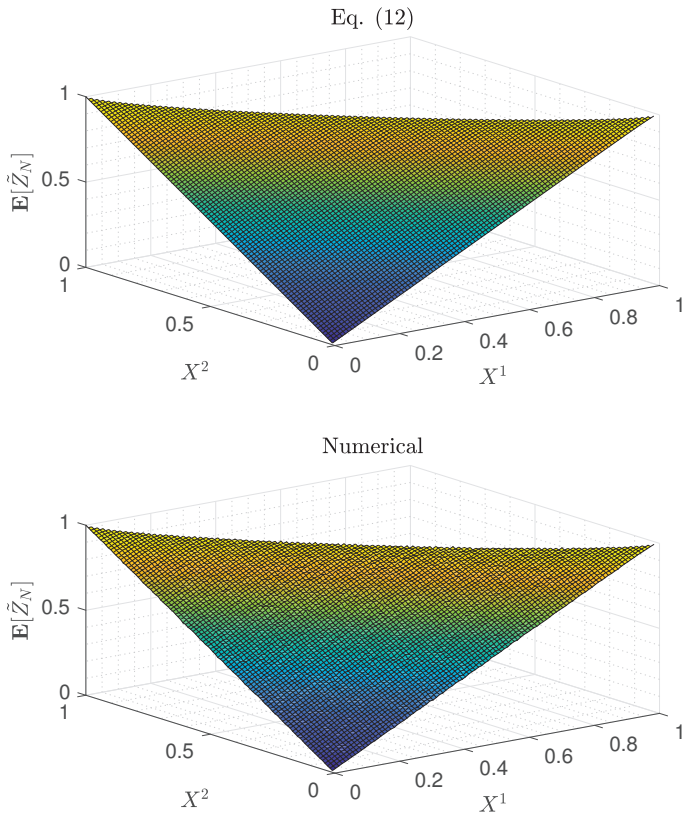


Figure 4.16: Expected value of the stochastic adder's output mean $\mathbb{E}[\tilde{Z}_N]$. Top: calculated using (4.11), parametrized with $M = 8$ states and sequence length $N = 64$. Bottom: Numerical calculations for 10^4 i.i.d. runs for each point.

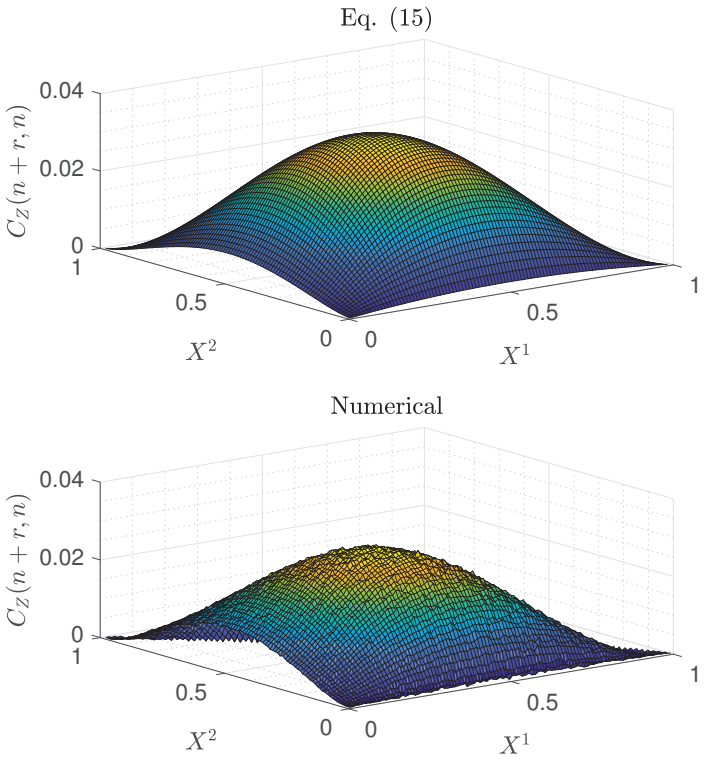


Figure 4.17: Auto-Covariance $C_Z(n+r,n)$ of the stochastic adder's output. Top: Calculated using (4.14), parametrized with $M = 8$ states, sequence length $N = 64$ and delay $r = 1$. Bottom: Numerical calculations for 10^4 i.i.d. runs for each point.

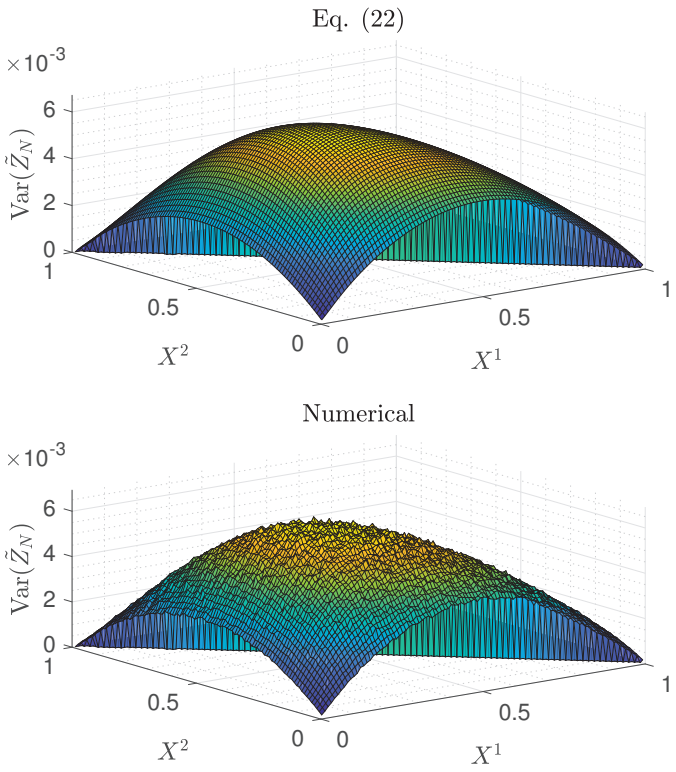


Figure 4.18: Variance of the stochastic adder's output mean $\mathrm{Var}(\tilde{Z}_N)$. Top: calculated using (4.21), parametrized with $M = 8$ states and sequence length $N = 64$. Bottom: Numerical calculations for 10^4 i.i.d. runs for each point.

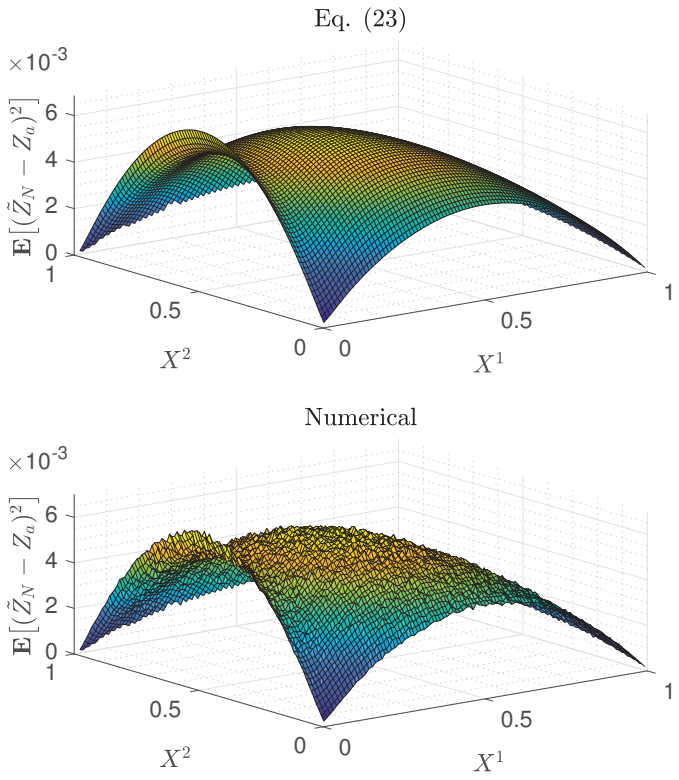


Figure 4.19: Mean Squared Error of the stochastic adder's output mean $\mathrm{MSE}(\tilde{Z}_N)$. Top: calculated using (4.22), parametrized with $M = 8$ states and input sequence length $N = 64$. Bottom: Numerical calculations for 10^4 i.i.d. runs for each point.

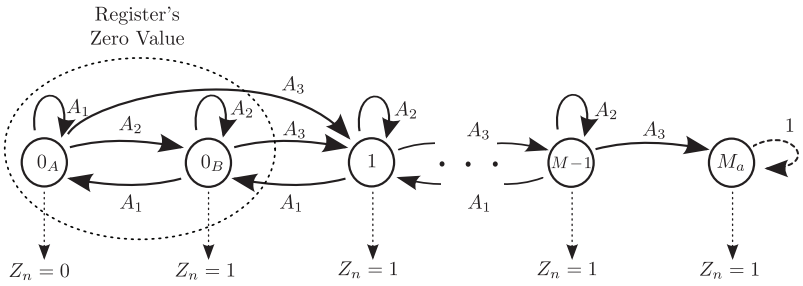


Figure 4.20: Markov Chain overflow model of the stochastic adder. Transition probabilities are given by (4.36).

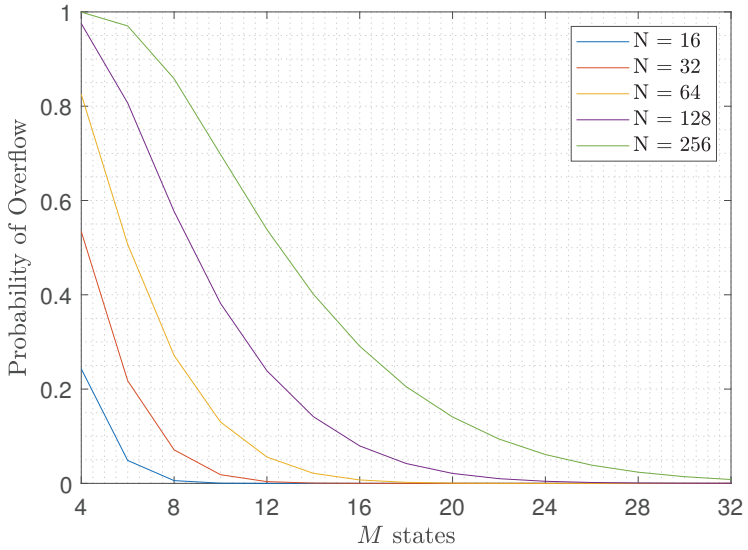


Figure 4.21: Probability of overflow of the stochastic adder calculated using (4.40), for inputs $X^1 = X^2 = 0.5$, increasing number of states $M = 4, \ldots, 32$ and increasing sequence lengths N.

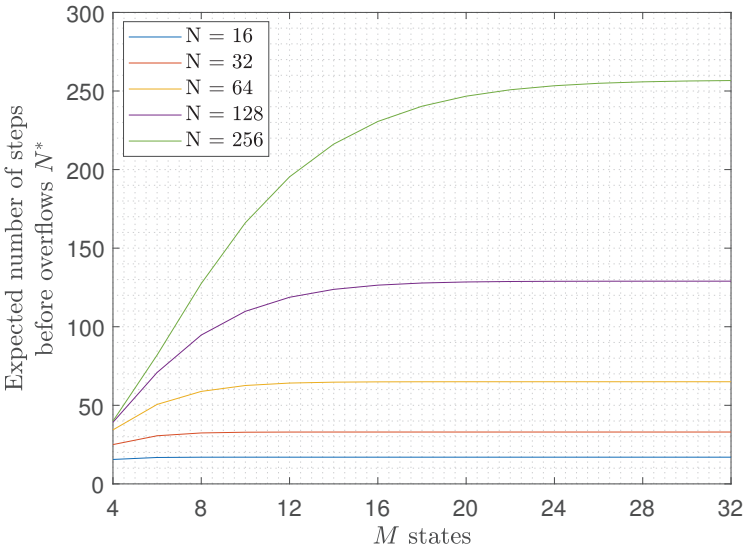


Figure 4.22: Expected number of steps before overflows N^* of the stochastic adder calculated using (4.31), for $M = 4, \ldots, 32$ states, inputs $X^1 = X^2 = 0.5$ and increasing sequences lengths N. The guideline $N^* \geq N$ allows for reduced overflow occurrence.

Table 4.1: Execution Times (s) for the Modeling of two SFSMs: the STanh and the Stochastic Adder

STanh 10^4 runs					
	Numerical Experiments(s)	Analytical Modeling(s)	Speedup	MAE	Time Savings (%)
Exp. Value	4.41	0.24	18.37	1.1×10^{-3}	94.55
Auto-Correlation	19.84	0.26	76.30	6.6×10^{-3}	98.58
Variance	6.76	3.35	2.01	1.7×10^{-4}	50.44
MSE	6.87	3.42	2.00	2.6×10^{-4}	50.12
Stochastic Adder 10^4 runs					
Exp. Value	303.80	1.62	187.5	5.1×10^{-4}	99.46
Auto-Correlation	302.89	2.28	132.49	3.8×10^{-3}	99.24
Variance	318.07	249.08	1.26	2.2×10^{-4}	21.69
MSE	320.19	251.15	1.27	1.1×10^{-4}	21.56
STanh 10^5 runs					
	Numerical Experiments(s)	Analytical Modeling(s)	Speedup	MAE	Time Savings (%)
Exp. Value	45.37	0.24	189.04	3.5×10^{-4}	99.47
Auto-Correlation	270.85	0.26	1041.73	9.8×10^{-4}	99.90
Variance	68.06	3.35	20.31	3.1×10^{-5}	95.07
MSE	67.56	3.42	19.75	8.2×10^{-5}	94.93
Stochastic Adder 10^5 runs					
Exp. Value	3.09×10^3	1.62	1.91×10^3	1.6×10^{-4}	99.94
Auto-Correlation	6.54×10^3	2.28	2.87×10^3	8.3×10^{-4}	99.96
Variance	3.20×10^3	249.08	1.28×10^3	4.9×10^{-5}	92.21
MSE	3.11×10^3	320.19	1.24×10^3	1.7×10^{-5}	91.91

Part II

Performance Results and Applications

5 Comparison with the Stochastic Computing Literature

In this chapter[1]. the proposed architectures are compared with popular SC adders [41, 72, 84, 23, 18], subtracters [4, 54, 18], MAX and MIN architectures [39, 43, 58, 89] existing in the literature, in both computational accuracy and hardware resources. With respect to the computational accuracy, the Mean Absolute Error (MAE) and/or the Mean Squared Error (MSE) are considered. For two inputs $X, Y \in [0, 1]$, the MAE and the MSE are defined respectively as

$$\text{MAE}(X, Y) = \mathbb{E}\left[\left|f(X, Y) - \tilde{Z}_N\right|\right] \tag{5.1}$$

and

$$\text{MSE}(X, Y) = \mathbb{E}\left[\left(f(X, Y) - \tilde{Z}_N\right)^2\right], \tag{5.2}$$

where $f(X, Y)$ is a function applied to the inputs such that $f : \mathbb{R}^2 \mapsto \mathbb{R}$ and $\tilde{Z}_N$ is a selected architecture's output mean. The MAE/MSE is estimated numerically in a grid of pair values (X, Y), assuming the unipolar SC format. For each grid point, 10^3 runs with pairs of i.i.d. sequences are conducted to derive the MAE/MSE. Then, all MAE/MSE values of each architecture are averaged, while the experiments are run for stochastic sequences with lengths $N = 2^k$, where $k = 4, \ldots, 10$.

Regarding the hardware resources, the operation of all architectures is described using Verilog HDL targeting the Xilinx Kintex-7 FPGA kit and then the designs are fed into the Synopsys Design Compiler using the FreePDK CMOS library at $45nm$ [77]. For the comparisons, the following estimates are provided: 1) the total area in μm^2, 2) the average power consumption for the max operating frequency in mW, 3) the critical path in ns and 4) the energy per operation (average power $\times$ the critical path) in pJ.

5.1 Comparison of Stochastic Adders

The SC adder architectures existing within the literature are shown in Fig. 5.1, where it is assumed that $\{X_n\}$ and $\{Y_n\}$ are the input sequences while $\{Z_n\}$ is the output sequence.

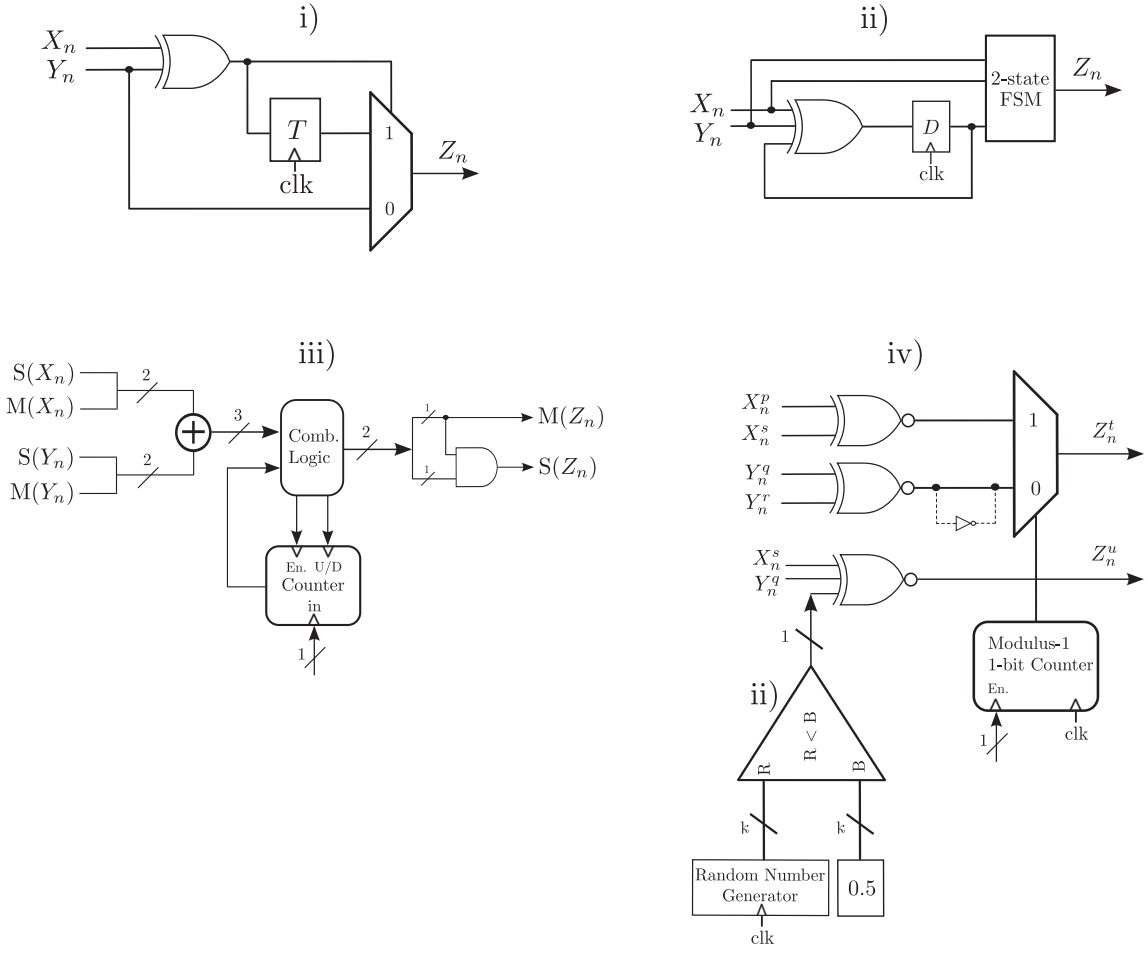


Figure 5.1: Stochastic Computing adders. From top left to bottom right: i) Scaling adder in [41], ii) Scaling adder in [84], iii) Non-scaling adder in [72] and iv) Scaling/Non-Scaling adder in [18].

The computational accuracy, the power $\times$ delay^2 and energy consumption of the adders considered are presented in Figs. 5.2 and 5.3, while their detailed hardware requirements, including the area, are cited in Table 5.1. Note that the hardware requirements for the input sequence generation are not included.

A) MUX: We consider the original circuit used for scaled addition (in unipolar format) and scaled subtraction (in bipolar) [23]. It requires large sequence lengths N to achieve acceptable accuracy compared to the other architectures, which reflects to the increased total energy consumption, according to Fig. 5.3. Moreover, the required SNG also impacts both power and energy consumption. This is why it is the least popular approach for both addition and subtraction.

B) Adders in [41] and [84]: The adder in [41] uses a T Flip-Flop to replace the SNG of the original

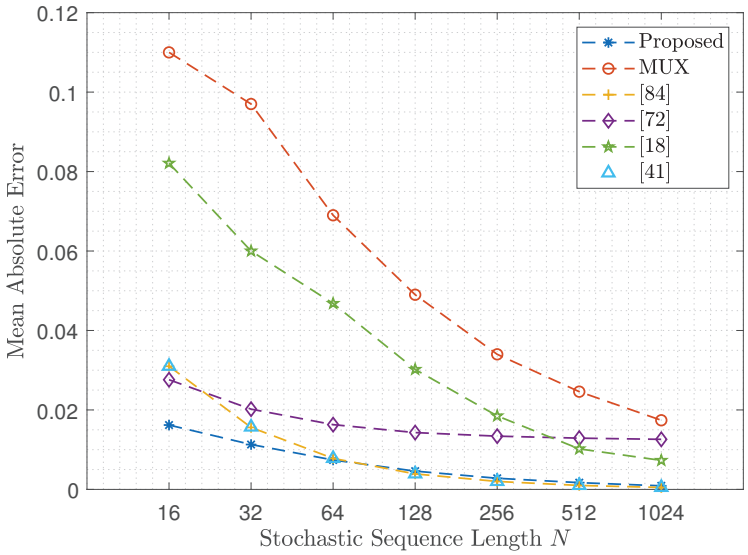


Figure 5.2: Comparison of accuracy in MAE of stochastic adders for typical stochastic sequence lengths N

MUX adder, while the adder in [84] employs a 1-bit register along with a two-state FSM to slightly improve on the accuracy. Compared to the adder in [84], the proposed one requires almost the same area for a register of $m = 2$-bits and slightly more power and energy consumption per operation according to Table 5.1. Compared to the adder in [41], the proposed one has higher power and energy consumption per operation. However, the proposed adder achieves better accuracy than both of them for short sequence lengths according to Fig. 5.2 and 5.4. Moreover, the non-scaling behavior of the proposed adder benefits cascaded computations since the resolution of the sequence is not reduced by 2 for every adder used.

C) Adder in [72]: The non-scaling adder in [72] assumes a two-line representation of a stochastic number, one to represent the magnitude and one the sign. Here we use the adder with unipolar format (plus fixed sign) and design parameter "threshold" 2, following the design methodology in [90], to compare it with the other adders. As shown in Fig. 5.2 it has lower accuracy than the proposed adder and has almost the same power consumption, with energy being its strong point according to Table 5.1. Moreover, the adder in [72] occupies more area compared to the proposed one for register sizes $m = 2, 3$.

D) Adder in [18]: The adder (and subtracter by using a NOT gate in one of its inputs) in [18] encodes stochastic numbers as the ratio of the switching activities of two sequences. The 4 inputs of the adder are pair-wise XNORed and then fed to a MUX that uses a modulus 1 counter as its select signal and it's output is the scaled result of the addition. To derive its non-scaling sum, two of the inputs and the output of a SNG are used as inputs to a 3 input XNOR. The XNOR's output as well as the MUX form the adder's outputs while their ratio yields the final (non-scaled) sum. According to Table 5.1, the overall hardware utilization is taxed due to the additional SNG, which impacts the power and energy consumption as well. Furthermore, to achieve comparable accuracy to that of the other adders, it requires large sequence lengths as shown in Fig. 5.2. Its main advantage is that it can be used for both scaling or non-scaling additions and the ratio encoding can be used to directly realize other standard operations as well, e.g. multipliers,

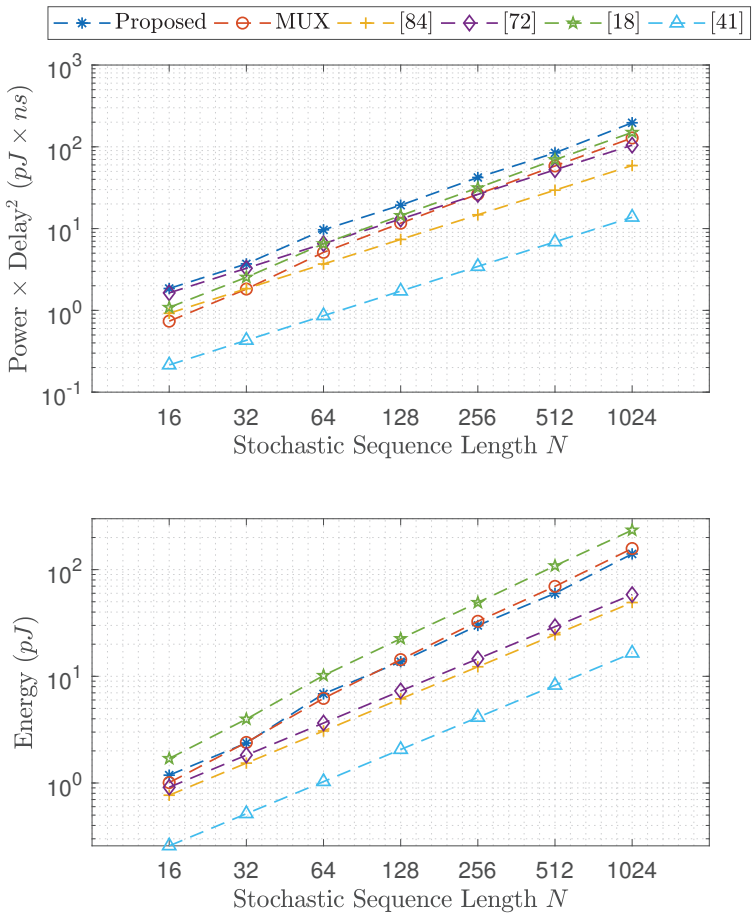


Figure 5.3: Comparison of Power $\times$ Delay^2 $(pJ \times ns)$ (top) and Energy (pJ) (bottom) consumption of stochastic adders for typical stochastic sequence lengths N

dividers etc. However, the incompatibility of the ratio encoding with other more popular SC encodings results in extra translation hardware complexity when is coupled with other traditional SC numerical architectures.

5.2 Comparison of Stochastic Subtracters

The SC subtracter architectures existing within the literature are shown in Fig. 5.5, where it is assumed that $\{X_n\}$ and $\{Y_n\}$ are the input sequences while $\{Z_n\}$ is the output sequence.

The accuracy, the power $\times$ delay^2 and energy comparison between the stochastic subtracters are shown in Figs. 5.6 and 5.7, while the detailed hardware utilization results are shown in Table 5.1. Since the NOT gate does not degrade the accuracy of the computations for the proposed subtracter as well as the MUX and [18], the results are identical that of the adder. Note that the area, power and energy consumption of the proposed subtracter is almost the same to those of the proposed adder since the additional

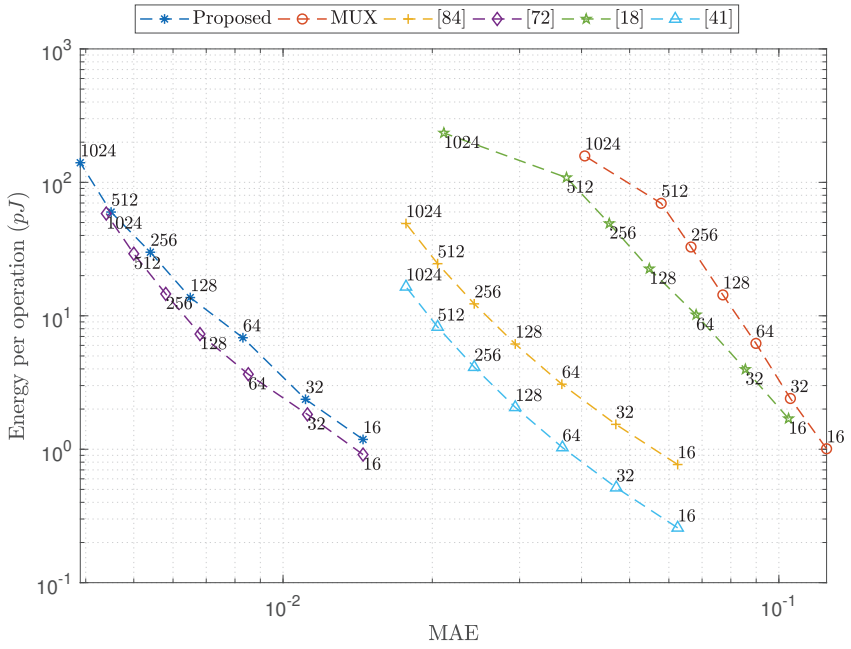


Figure 5.4: Comparison of Energy per operation ($pJ \times ns$) and MAE of stochastic adders for typical stochastic sequence lengths N. Sobol sequences are used.

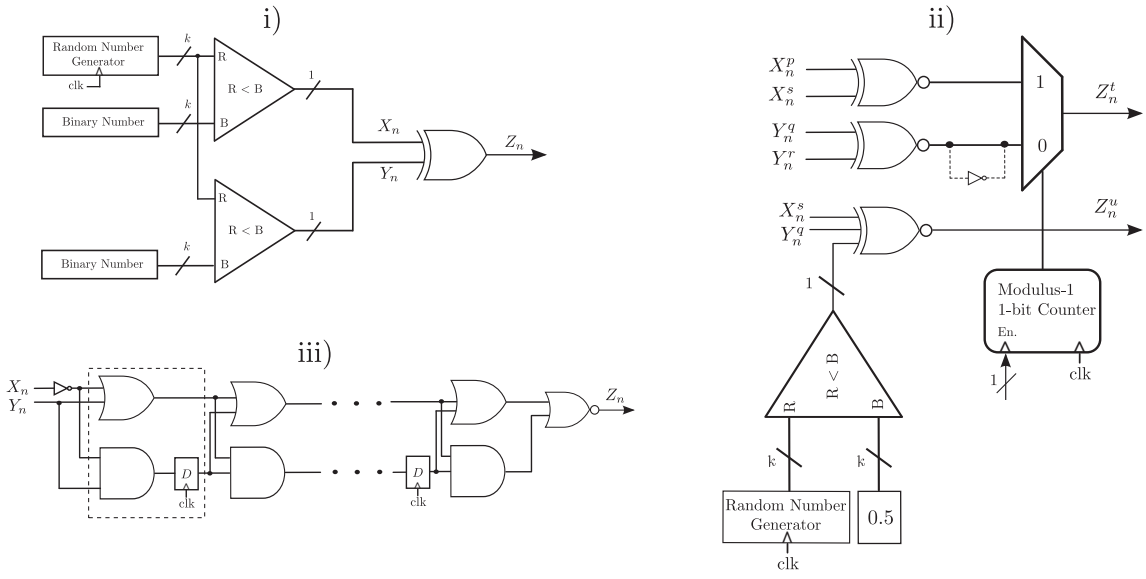


Figure 5.5: Stochastic Computing subtracters. From top left to bottom right: i) Absolute correlated input subtracter in [4], ii) Scaling/Non-scaling subtracter in [18] and iii) Subtracter in [54].

two NOT gates have minimal impact.

A) Subtracter in [4]: To realize the operation of subtraction hardware-efficiently, the method in [4] correlates two input sequences, using the same LFSR for two different comparators in the SNG stage,

Table 5.1: Hardware Resources Comparison between the Proposed Non-Scaling Adder and Subtracter and the State-of-the-Art in Area (μm^2), Critical Path (ns), Power Consumption (mW) and Energy (pJ) per operation

	Stochastic Adders				
	Register (bit)	Area (μm^2)	Power (mW)	Critical path (ns)	Energy (pJ)
Proposed* Adder/Subtracter	$m=2$	59.60	0.053	1.4	0.074
	$m=3$	83.49	0.077		0.108
	$m=4$	98.30	0.084		0.117
	$m=5$	112.61	0.098		0.137
[41]		22.41	0.021	0.8	0.016
[84]		54.39	0.040	1.2	0.048
[72]		92.49	0.071	1.2	0.057
MUX* Adder/Subtracter LFSR size k	$k=4$	74.34	0.079	0.8	0.063
	$k=5$	97.62	0.094		0.075
	$k=6$	122.09	0.122		0.097
	$k=7$	133.71	0.140		0.112
	$k=8$	168.21	0.160		0.128
	$k=9$	177.40	0.171		0.136
	$k=10$	192.20	0.193		0.154
[18]* Adder/Subtracter LFSR size k	$k=4$	83.49	0.106	0.8	0.084
	$k=5$	105.69	0.124		0.099
	$k=6$	126.24	0.159		0.127
	$k=7$	144.54	0.176		0.140
	$k=8$	168.65	0.192		0.153
	$k=9$	178.64	0.212		0.169
	$k=10$	194.75	0.229		0.183
	Stochastic Subtracters				
[54]		41.76	0.063	0.8	0.050
[4]** LFSR size k	$k=4$	105.12	0.14	0.8	0.112
	$k=5$	130.75	0.176		0.140
	$k=6$	166.13	0.229		0.183
	$k=7$	188.65	0.264		0.211
	$k=8$	207.01	0.299		0.239
	$k=9$	229.95	0.331		0.264
	$k=10$	264.54	0.350		0.280

* In these cases the subtracter is obtained with negligible additional hardware requirements (2 NOT gates for the proposed adder and 1 for the rest) and its impact is insignificant in the area, power and energy consumption

** Includes sequence correlation

and an XOR gate to provide the output sequence. This architecture has an important key point; consider the following two cases: I) If the first SC operation is the subtraction, one can shape the architecture to effectively generate two signals from SNGs with one LFSR [4, 39]. II) If, on the other hand, subtraction is an intermediate SC operation, to effectively use the XOR gate, the subtracter's input sequences must be re-generated in order to have high cross-correlation. Fig. 5.6 suggests that[4] achieves lower accuracy compared to the proposed stochastic subtracter, while it has the advantage of very low power and energy consumption when operating in case (I) above. When the subtraction is an intermediate operation, case (II), the proposed subtracter achieves better overall performance as shown in Fig. 5.7 and Table 5.1.

B) Subtracter in [54]: The subtracter in [54] uses an XNOR gate with inputs the two stochastic sequences, one of them inverted, to generate a rough estimate of the subtraction result, and cascaded logic stages to improve its accuracy. The number of additional logic stages considered here is 3, but can be fur-

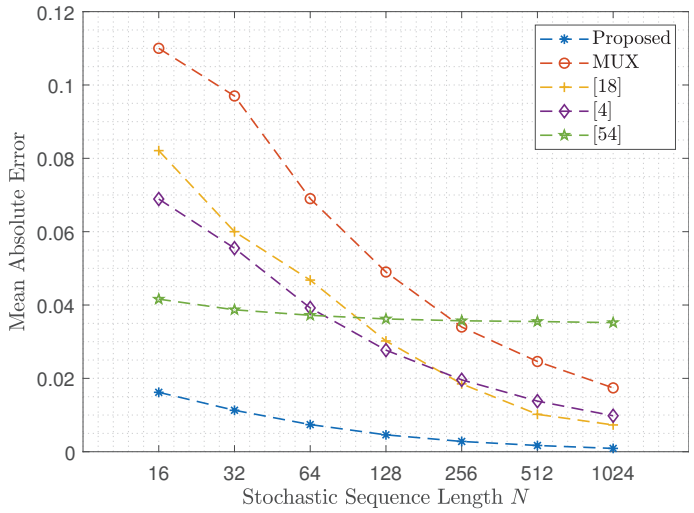


Figure 5.6: Comparison of accuracy in MAE of stochastic subtracters for typical stochastic sequence lengths N

ther expanded at the cost of additional hardware resources and delay. The proposed stochastic subtracter achieves better accuracy than the one in [54] as shown in Fig. 5.6, but, the latter one has lower power and energy consumption according to Table 5.1.

5.3 Comparison of Stochastic MAX and MIN

The SC MAX/MIN architectures existing within the literature are shown in Fig. 5.9, where it is assumed that $\{X_n\}$ and $\{Y_n\}$ are the input sequences while $\{Z_n\}$ (and $\{K_n\}$ if the MIN is also produced) is the output sequence.

The computational accuracy of the proposed MAX and MIN architectures is shown in Fig. 5.10, the power $\times$ delay^2 and energy consumption metrics are shown in Fig. 5.11, while the detailed hardware resources are cited in Table 5.3. We note that the MAX architectures in [39, 43, 58, 89] including the proposed one are able to output the MIN as well without affecting the total hardware resources, i.e. introducing additional logic units or registers. Therefore, the presented accuracy and hardware resource metrics for the MAX architectures apply to the MIN architectures as well. Moreover, in Table 5.2 the register sizes resulting in highest computational accuracy with respect to the sequence lengths N are cited.

A) MAX/MIN in [39]: The core of the architecture is a 3-state FSM that forces the overlap of logic ones between its two input i.i.d. sequences $\{X_n\}$ and $\{Y_n\}$, to produce two correlated outputs. These are used as inputs to an OR gate to produce the final output. If the OR gate is replaced by an AND in the architecture, then the MIN can be realized. According to Fig. 5.10 the proposed architecture has better accuracy regardless of the sequence length N used and this is intensified especially for smaller values of N. Hardware-wise, the proposed architecture occupies less area as well as consumes less power and energy for register sizes $m = 1, 2$, similar for $m = 3$, while for $m = 4, 5$ Lee et. al's method [39] is

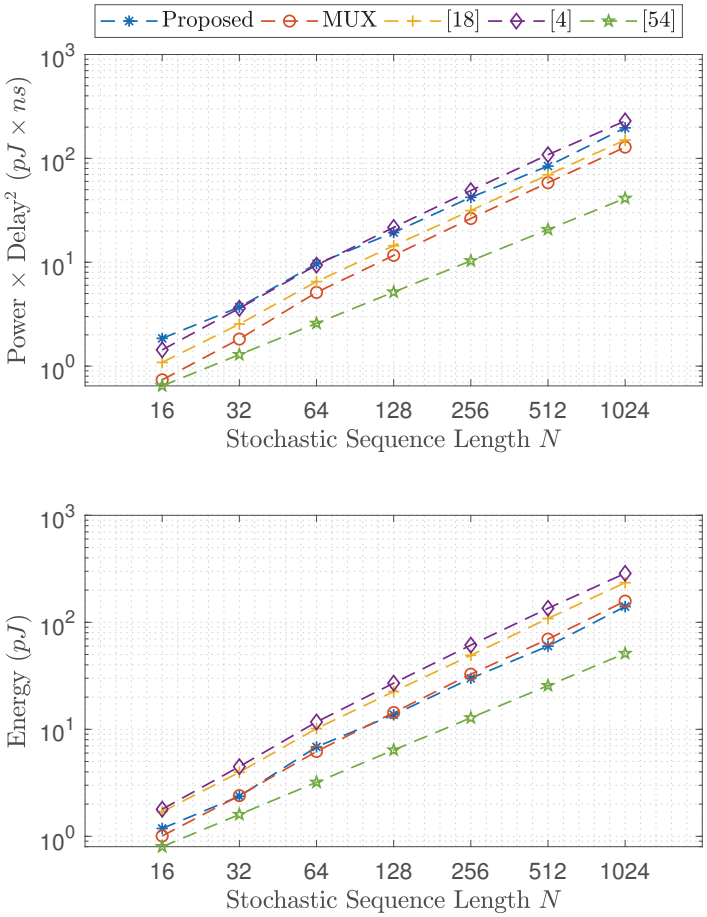


Figure 5.7: Comparison of Power $\times$ Delay^2 $(pJ \times ns)$ (top) and Energy (pJ) (bottom) consumption of stochastic subtracters for typical stochastic sequence lengths N

slightly better. However, the fact that Lee et. al's approach in [39] requires more clock cycles to achieve the same accuracy as the proposed architecture should not be neglected; the increased latency implies a further increase in dissipated energy, exceeding the proposed one's.

B) MAX/MIN in [43]: The inputs of this architecture, are fed to a MUX that uses a SNG as its select signal. The MUX's stochastic output is the input of the stochastic tanh function [15], implemented as a FSM of 2^m states (m-bits), while the FSM's output is determined by the current state; starting from the zero state, the first $2^m/2 - 1$ output 0, while the rest output 1. The FSM's output is also used as a select signal in a MUX that determines whether X_n or Y_n to be the architecture's current output. From Fig. 5.10, the proposed architecture results in better computational performance in terms of accuracy. The increased performance of the proposed architecture also applies to the hardware utilization as shown in Table 5.3, which is due to the additional SNG used, contributing negatively in the total area, power and energy consumption. Moreover, an important design aspect is the register's size. As stated in Li et

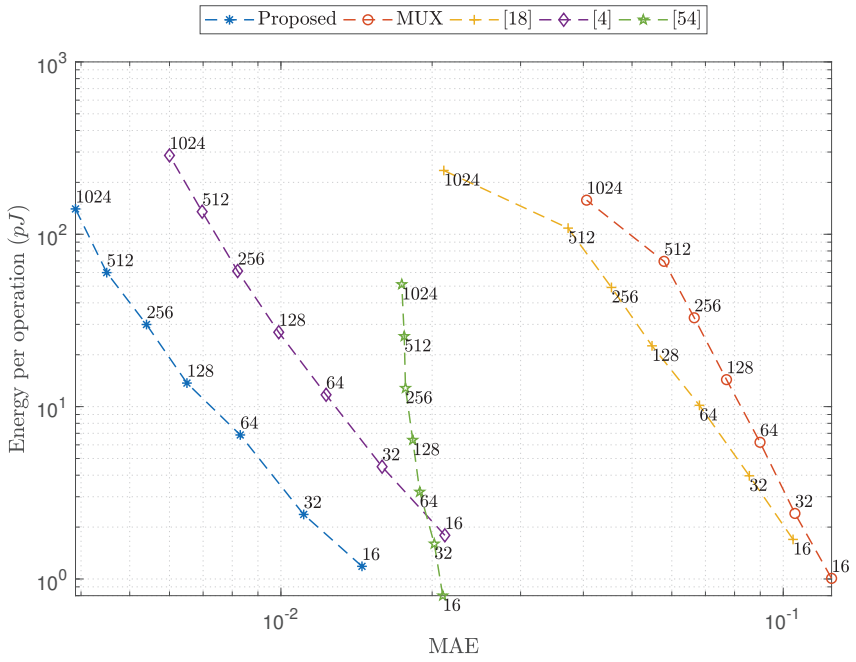


Figure 5.8: Comparison of Energy per operation ($pJ \times ns$) and MAE of stochastic subtracters for typical stochastic sequence lengths N. Sobol sequences are used.

al. in[43], increasing its size and hence the number of its states, the computational accuracy increases as well. Yet the selection of its size that yields the highest accuracy is estimated with numerical simulations. On the other hand, the guidelines to select the register's size in the proposed architecture eliminates the parametric simulation time completely.

C) MAX/MIN in [89]: To avoid the power and area hungry SNG from Li et al.'s method in [43], the architecture by Yu et al.[89] uses an XOR between the two inputs instead, that acts as an enable signal to up-count logic 1s coming from its input X_n. The counting is based on the stochastic tanh FSM, implemented in the same way shown by Li et al. in[43]. Consequently, the tanh's output is used as a select signal in a MUX that determines if X_n or Y_n is the output. Accuracy-wise, the proposed architecture results in better computational results as shown in Fig. 5.10. In terms of hardware resources, the proposed architecture occupies larger area, but, has reduced power and energy consumption when the same register size is used according to Table 5.3. From a designer's perspective, the register size that maximizes the accuracy of the Yu et al.'s architecture in [89] is derived with simulations. If not chosen carefully based on the sequence length N, it directly affects the output's accuracy; by reducing the number of its states it will increase the output's error. On the contrary, the analytic derivation of the proposed max's register size, provides insight on its design.

D) MAX/MIN in [58]: In this architecture, motivated by Yu et al.'s method in [89], an XOR between the inputs is used as an enable signal in a linear FSM, implemented as a shift register of m-bits (can also be implemented as a binary counter). The FSM performs a right shift of the most significant bit (MSB) if $X_n = 1$ whereas a left shift if $X_n = 0$. The FSM's output is determined by the least significant bit (LSB)

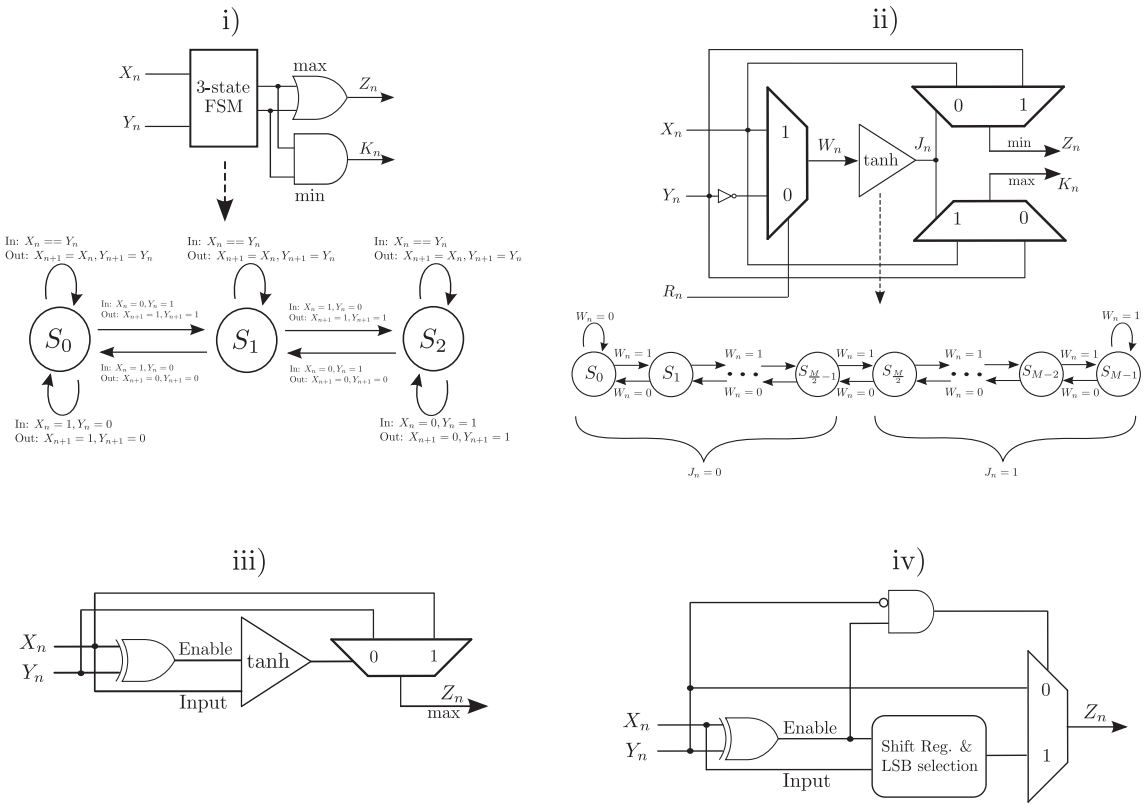


Figure 5.9: Stochastic computing max and min architectures. From top left to bottom right: 1) Correlated MAX/MIN in [39], ii) Tanh MAX/MIN in [43], iii) Tanh MAX/MIN w/o RNG [89] and iv) Shift Register MAX/MIN in [58].

of the register and produces 1 if it has saturated up to the LSB. Finally, a MUX selects either the FSM's output or Y_n along with additional logic gates. From the comparison with the proposed architecture shown in Fig. 5.10, the approach by Lunglmayr et al. in [58] results in lower computational accuracy. However, the power and energy consumption per clock cycle is its strong point which is due to the advantage of the shift register over the binary one as shown in Table 5.3. Yet, the architecture's output accuracy depends on the saturation of the shift register up to its LSB. If its size is not chosen accurately, for instance if it is less or more than a specific value, the output's accuracy can be greatly reduced and this is also shown in Lunglmayr et al. [58]. We note that the register size that results in highest accuracy possible is used in the simulations and is also cited in Table 5.2 , taken from [58].

5.4 Comparison of Stochastic Compact MAX and MIN

The computational accuracy and the energy consumption of the MAX and MIN are presented in Figs. 5.13 and 5.14. Furthermore, the hardware resources are cited in Table 5.5. We note the following: 1) for all architectures we selected for each N the register size m that results in the highest computational accuracy possible and we provide it in Table 5.4; 2) we assumed that the up & down counting in all

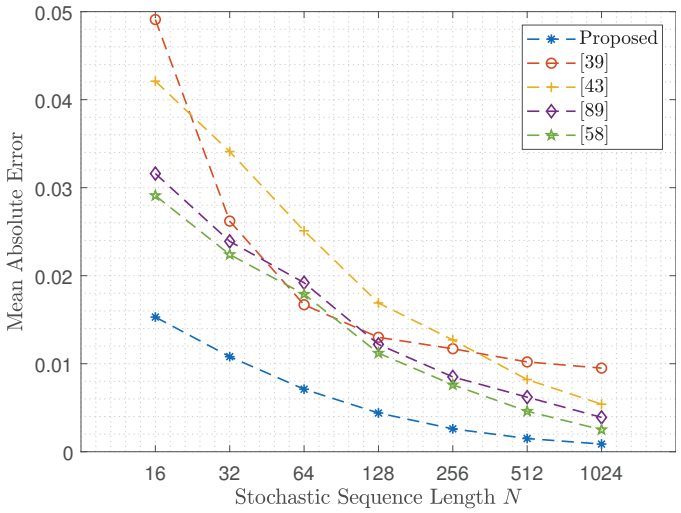


Figure 5.10: Accuracy comparison in MAE of stochastic MAX/MIN architectures for typical sequence lengths N. For each N, the architectures' number of states is selected to result in the highest computational accuracy. Corresponding register sizes are cited in 5.2.

MAX/MIN architectures is done using binary (ripple) counters able to count up to $M = 2^m$ states, where m is the register's size; and 3) the MAX architectures [39, 43, 58, 89] including the proposed one can output the MIN without additional hardware resources (only with modification). As such, the presented accuracy results in Fig. 5.13 and the hardware resources in Table 5.5 for the MAX architectures apply to the MIN ones as well.

A) MAX/MIN in [43]: in According to Fig. 5.13, the proposed MAX results in better computational accuracy and also occupies less resources according to Table 5.5, which is due to the SNG used in [43]. Considering the register's size, in the proposed MAX architecture it is derived according to the analysis shown in Section 4, whereas in the architecture in [43] numerical simulations are required.

B) MAX/MIN in [89]: Compared to [89], the proposed MAX results in better computational performance according to Fig. 5.13. In terms of hardware resources, the proposed MAX occupies slightly more area when the same register size is used, but, has less energy and power consumption; although it is expected that higher area will result in higher power and energy consumption, in fact, the synthesis tool optimizes further the design's mapping using high area, power and mapping effort. Therefore, even if one counts the gates used between the two architectures, the theoretical result will not reflect the one obtained from the synthesis tool. Moreover, the advantage of the proposed MAX architecture over the one in [89], is the design guidelines for the register's size selection according to N, which eliminates the simulation time completely.

C) MAX/MIN in [58]: From Fig. 5.13, the proposed architecture results in better accuracy, but, the architecture in [58] is more hardware-efficient according to Table 5.5, which is due to the shift-register used over the binary one. However, it is expected that if the shift-register's size is not chosen carefully based on the sequence length N, it will directly affect the output's accuracy; when the number of the FSM's states are reduced, it will result in reduced computational accuracy and this is shown in [58]. We

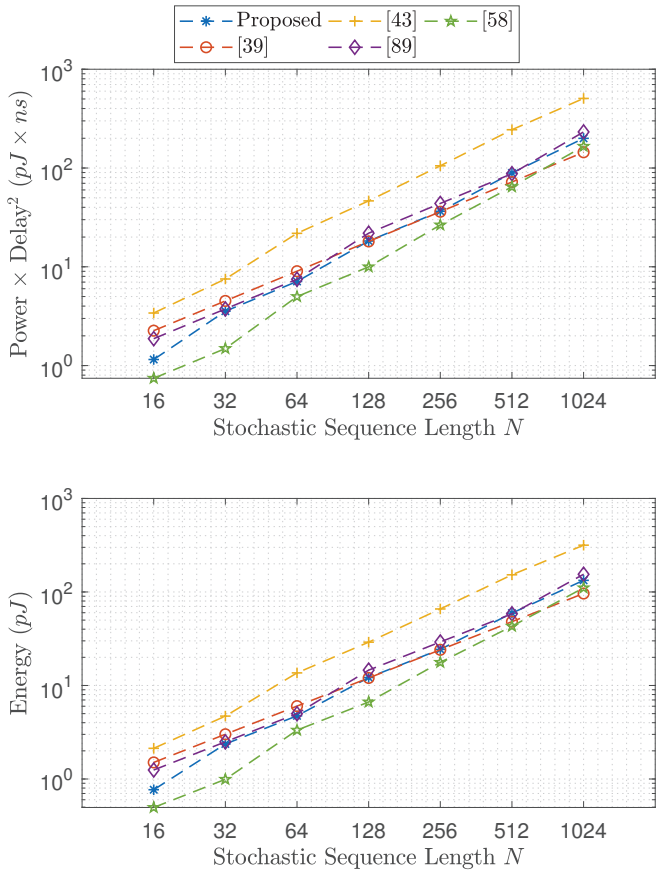


Figure 5.11: Comparison of Power $\times$ Delay^2 $(pJ \times ns)$ (top) and Energy pJ (bottom) consumption of stochastic MAX/MIN architectures. For each N, the architectures' number of states is selected to result in the highest computational accuracy. Corresponding register sizes are cited in 5.2.

note that the shift register size values shown in Table 5.4 are taken from [58].

D) MAX/MIN in [39]: From Fig. 5.13, it can be seen that the proposed MAX results in better accuracy, regardless of the stochastic sequence length N, when a 3-state FSM is used. To further investigate the impact of the FSM's number of states in accuracy, we increased their total number from 3 to 5. One can observe that the accuracy is increased for sequences with $N \geq 128$-bit length when compared to the 3-state FSM, but, it is lower than that of the proposed MAX architecture. In terms of hardware resources, the proposed MAX achieves similar performance with register sizes $m = 2, 3$-bits, while for more than 4-bits, the MAX in [39] is slightly better. However, one should not neglect the fact that to achieve the same accuracy as the proposed one, the MAX in [39] requires more computational cycles N, which reflects on the total energy consumed as shown in Fig. 5.12.

E) MAX/MIN in [81]: In the architecture in [81], the procedure to store the differences between the inputs follows that of the proposed MAX. However, in the proposed work the current value of the register's state is compared with $M/2$ instead of 0 in [81]. As such, the overflow/underflow modeling is

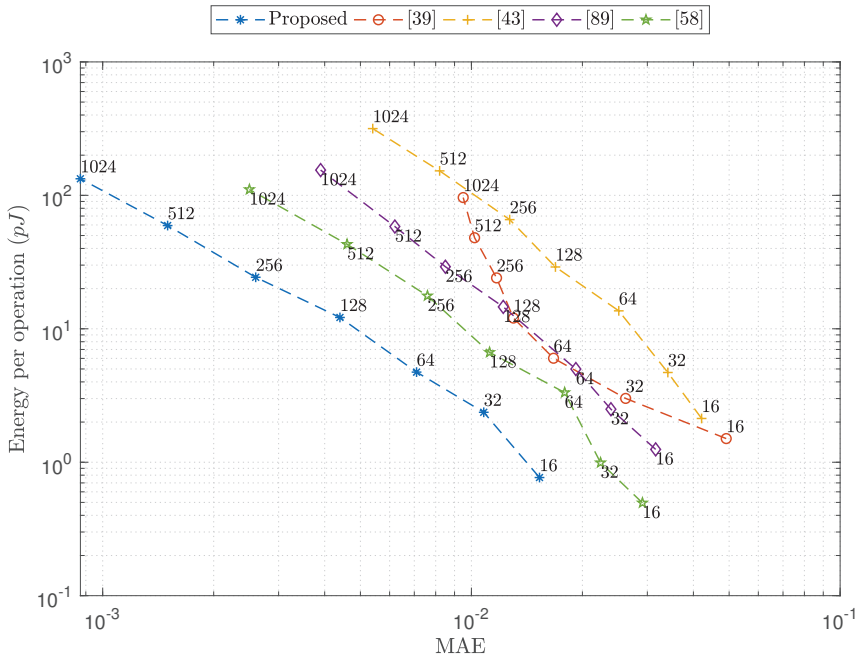


Figure 5.12: Comparison of Energy per operation $(pJ \times ns)$ and MAE of stochastic MAX/MIN for typical stochastic sequence lengths N.

improved and hence the understanding of the errors due to overflows. According to Fig. 5.13 it can be seen that the MSE has the same order of magnitude. However, with respect to the hardware resources, the proposed MAX requires slightly less area when the same register size is used.

Table 5.2: Comparison of computational accuracy and corresponding register sizes of MAX/MIN architectures for typical sequence lengths N

Mean Absolute Error (MAE) $\times 10^{-2}$							
$N = 2^k$	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Proposed	1.53	1.08	0.71	0.44	0.26	0.15	0.08
Register m-bit	1	2		3		4	5
[39]	4.91	2.62	1.67	1.30	1.11	1.02	0.95
Register m-bit	2						
[43]	4.21	3.41	2.51	1.69	1.27	0.82	0.54
Register m-bit	2		3			4	5
[89]	3.16	2.39	1.92	1.22	0.85	0.62	0.39
Register m-bit	2			3		4	5
[58]	2.91	2.24	1.79	1.11	0.76	0.46	0.25
Shift Register m-bit	2			3	4	5	6

Table 5.3: Hardware Resources Comparison between the Proposed MAX/MIN and the State-of-the-Art in Area (μm^2), Critical Path (ns), Power (mW) and Energy (pJ) Consumption per operation

	Register m-(bit)	Area (μm^2)	Power (mW)	Critical path (ns)	Energy (pJ)
Proposed	$m = 1$	37.54	0.032	1.5	0.048
	$m = 2$	60.86	0.049		0.074
	$m = 3$	88.69	0.063		0.095
	$m = 4$	106.24	0.077		0.116
	$m = 5$	119.21	0.086		0.130
[39]	$m = 2$	91.49	0.062	1.5	0.094
[43] MUX LFSR size k	$m = 2, k = 4$	109.81	0.083	1.6	0.133
	$m = 2, k = 5$	131.87	0.092		0.147
	$m = 3, k = 6$	176.92	0.133		0.213
	$m = 3, k = 7$	199.45	0.142		0.227
	$m = 3, k = 8$	236.32	0.161		0.257
	$m = 4, k = 9$	291.90	0.184		0.295
	$m = 5, k = 10$	311.61	0.193		0.309
[89]	$m = 2$	48.01	0.052	1.5	0.078
	$m = 3$	61.47	0.076		0.114
	$m = 4$	104.97	0.101		0.151
[58] Shift Register	$m = 2$	46.46	0.021	1.5	0.031
	$m = 3$	57.25	0.034		0.052
	$m = 4$	73.21	0.046		0.069
	$m = 5$	89.16	0.057		0.084
	$m = 6$	105.12	0.068		0.103

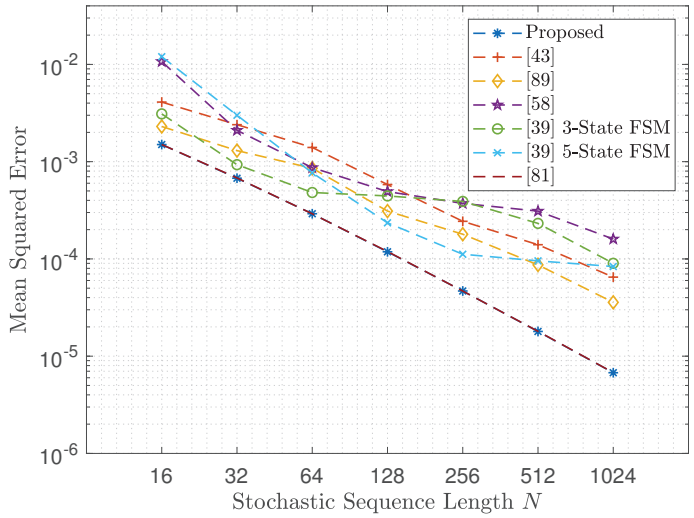


Figure 5.13: Accuracy comparison in MSE of stochastic MAX architectures for typical sequence lengths N. For each N, their register sizes are selected to result in the highest MSE and are cited in Table 5.4.

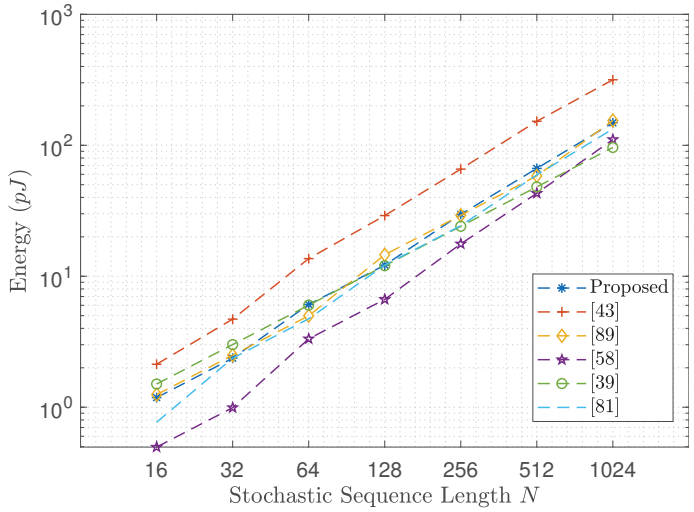


Figure 5.14: Energy comparison in pJ of stochastic MAX architectures for typical sequence lengths N. For each N, their register sizes are selected to result in the highest MSE and are cited in Table 5.4.

Table 5.4: Register sizes resulting in the highest MSE based on N

$N = 2^k$	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Proposed	2	3		4	5		6
[43]	2		3			4	5
[89]	2			3		4	5
[58]	2			3	4	5	6
[39]	2			3			
[81]	1	2		3		4	5

Table 5.5: Hardware Resources Comparison between the Proposed Compact MAX/MIN and the State-of-the-Art in Area (μm^2), Critical Path (ns), Power (mW) and Energy (pJ) Consumption

	Register m-(bit)	Area (μm^2)	Power (mW)	Critical path (ns)	Energy (pJ)
Proposed	$m = 2$	48.33	0.044	1.5	0.066
	$m = 3$	73.69	0.063		0.094
	$m = 4$	92.31	0.074		0.111
	$m = 5$	106.55	0.081		0.121
	$m = 6$	117.44	0.093		0.139
[43] MUX LFSR size k	$m = 2, k = 4$	109.81	0.083	1.6	0.133
	$m = 2, k = 5$	131.87	0.092		0.147
	$m = 3, k = 6$	176.92	0.133		0.213
	$m = 3, k = 7$	199.45	0.142		0.227
	$m = 3, k = 8$	236.32	0.161		0.257
	$m = 4, k = 9$	291.90	0.184		0.295
	$m = 5, k = 10$	311.61	0.193		0.309
[89]	$m = 2$	48.01	0.052	1.5	0.078
	$m = 3$	61.47	0.076		0.114
	$m = 4$	104.97	0.101		0.151
[58] Shift Register	$m = 2$	46.46	0.021	1.5	0.031
	$m = 3$	57.25	0.034		0.052
	$m = 4$	73.21	0.046		0.069
	$m = 5$	89.16	0.057		0.084
	$m = 6$	105.12	0.068		0.103
[39] 3-State FSM	$m = 2$	91.49	0.062	1.5	0.094
[81]	$m = 1$	37.54	0.032	1.5	0.048
	$m = 2$	60.86	0.049		0.074
	$m = 3$	88.69	0.063		0.095
	$m = 4$	106.24	0.077		0.116
	$m = 5$	119.21	0.088		0.130

6 Applications

In this chapter[1] we use the proposed architectures to implement several digital signal processing tasks [79, 80, 81]. We evaluate their performance with comparisons with the standard binary computing methods as well as with the SC literature in computational accuracy and hardware resources.

6.1 Image Blurring

To demonstrate the proposed adder's effectiveness in cascaded computations we use it as a building block, along with AND gates for multiplication, to realize a convolution kernel, as shown in Fig. 6.1. Specifically, the convolution is used in a Digital Image Processing task, which is the filtering of an entire image using a 3×3 Box blur kernel. By adjusting appropriately kernel's weight values and by including non-linear functions, this kernel can be used in Neural Networks.

For the application, we select a gray-scale image and represent each pixel with an 8-bit number as it is typically required in image processing. The pixels' and the kernel's values are normalized to range $[0, 1]$ in order to be processed in the SC domain. For the stochastic number representation, we consider typical stochastic sequence lengths, namely $N = 2^k$, with $k = 4, \ldots, 10$ and investigate their effect in the accuracy of the computations. Then, the proposed, as well as selected adders, are used to realize the convolution operation and their performance is reported.

Among the selected adders, we excluded the standard MUX from comparisons as it requires large N lengths to achieve acceptable accuracy as shown in Fig. 5.2 implying also an increased hardware overhead. The same applies to the adder in [18] due to the fact that 1) the two-sequence encoding of a stochastic number increases the design complexity in cascaded computations and 2) each non-scaling

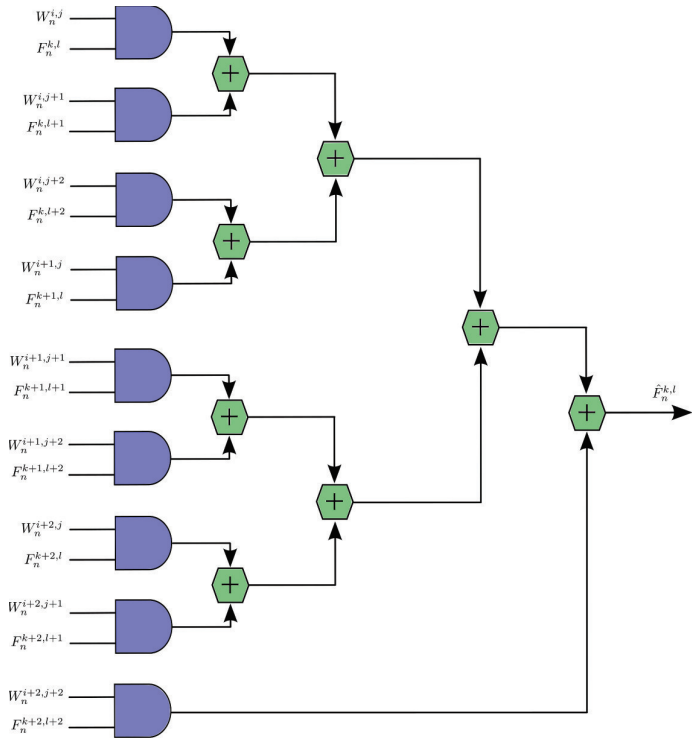


Figure 6.1: A 3×3 stochastic computing convolution kernel realized using 9 AND gates for multiplication and 8 proposed non-scaling stochastic adders. W_n and F_n denote the generated sequences of weights and the input image pixel values respectively.

adder is hardware demanding for moderate N lengths according to Table 5.1.

The accuracy comparison of the convolutions based on the selected adders is shown in Table 6.1 evaluated with two metrics: 1) the Peak Signal-to-Noise Ratio (PSNR) and 2) the Structural Similarity Index Measure (SSIM). The first measures the absolute accuracy of computation and is one of the standard metrics used for images, while the second one measures the perceived quality of an image with values in $[0, 1]$ (higher means better quality) [87]. In addition, a graphical representation of the computations using the proposed adder is shown in Fig. 6.2. Moreover, Table 6.2 presents the corresponding hardware resources to realize the convolution kernel.

a) b) c) d) e) f) g) h) i)

Figure 6.2: Image filtering using a 3×3 convolution kernel for various sequence lengths N. From left to right cases: a) Original image b) MATLAB's blur calculation c) $N = 16$ d) $N = 32$ e) $N = 64$ f) $N = 128$ g) $N = 256$ h) $N = 512$ i) $N = 1024$

We note first that the register size used for the proposed adder is $m = 2$, as it does not degrade the

Table 6.1: Accuracy and Image Quality Comparison in the Filtering with a 3×3 Convolution Kernel using the Proposed and State-of-the-Art Stochastic Adders

	Peak-Signal-to-Noise Ratio (PSNR)						
$N = 2^k$	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Proposed	16.61	19.25	22.06	25.08	28.01	30.90	33.94
[84]	<10	13.56	17.91	21.95	25.68	28.67	31.92
[41]	<10	13.01	17.58	21.52	25.71	28.57	31.84
[72]	16.45	19.04	22.08	25.15	28.01	30.96	33.63
[68]	15.75	18.76	21.57	24.80	27.83	30.81	33.84
	Structural Similarity Index Measure (SSIM)						
Proposed	0.210	0.292	0.391	0.512	0.628	0.734	0.831
[84]	<0.2	<0.2	0.231	0.346	0.490	0.620	0.748
[41]	<0.2	<0.2	0.250	0.367	0.504	0.628	0.747
[72]	0.218	0.297	0.391	0.498	0.612	0.725	0.828
[68]	0.209	0.289	0.381	0.487	0.598	0.713	0.811

Table 6.2: Comparison of Hardware Resources for Implementing the 3×3 Convolution Kernel using the Proposed and State-of-the-Art Stochastic Adders in Area (μm^2), Critical Path (ns), Power (mW) and Energy (pJ) per operation

	Area (μm^2)	Power (mW)	Critical Path (ns)	Energy (pJ)
Proposed	554.63	0.390	1.6	0.624
[68] Binary Output	443.95	0.157	1.9	0.298
[68] Stochastic Output	661.71	0.254	2	0.509
[41]	207.57	0.104	1.5	0.156
[84]	456.62	0.322	1.6	0.515
[72]	952.20	0.492	1.5	0.738
Conventional Binary	9,017.13	2.440	7.5	18.30

result of calculations in this specific application. Moreover, the kernel by itself, requires 9 multipliers (AND gates) and 8 stochastic adders, while its structure is adder tree based.

A) Convolution using adders [41] and [84]: According to Table 6.1, the two scaling adders [41] and [84] provide acceptable accuracy and SSIM results when using more than $N = 256$-bit sequence lengths, which is due to sequence resolution drop by 2 after each addition. On the contrary, the convolution using the proposed stochastic adder achieves the same accuracy using only half sequence length, e.g. $N = 128$. This leads to less energy per convolution for the proposed compared to the convolution using [84] due to different lengths required. The convolution using [41] however, despite the large N value, e.g. 256, it also achieves good power and energy efficiency. A further advantage of the proposed adder is that it benefits operations that require non-scaled computations after the convolution stage, whereas adders [41] and [84] face up-scaling followed by sequence regeneration design challenges.

B) Convolution using adder [72]: Due to its non-scaling nature, the convolution kernel using the adder in [72] achieves the same accuracy as the proposed approach according to Table 6.1. However,

it also has slightly increased energy consumption, making the proposed one more efficient for multiple non-scaling additions.

C) Convolution using adder [68]: The accumulative parallel counter (APC) is a popular adder capable of adding single-bit sequences in parallel producing binary output and it is used in the SC literature [50, 83] as it benefits multiply-and-accumulate stages. Compared to the proposed approach, it achieves almost the same performance in terms of accuracy. Note however that if the convolution is the final operation, i.e. no further operations are required, APC is effective terms of hardware. Otherwise it requires more area than the proposed approach due to the required binary-to-stochastic converter to re-randomize the output for further computations, e.g. non-linear functions.

D) Convolution with conventional binary: Compared to the conventional arithmetic architectures, the proposed approach requires negligible area, which is its strong point and is approximately $\times 16$ less according to Table 6.2. On the other hand, given the moderate number clock cycles to achieve acceptable results, for instance $N = 64$, its energy consumption is higher, namely $21pJ$.

6.2 Image Sharpening Filter

We demonstrate the efficiency of the proposed non-scaling adder and subtracter in cascaded computations with the realization of an image sharpening filter [24]. Its operation, is described as

$$g(k,l) = f(k,l) + c\big(f(k,l) - w * f(k,l)\big), \tag{6.1}$$

where $f(k,l)$ and $g(k,l)$ are the input and output images of size $k \times l$ respectively, w is a weight mask and c is a constant.

To further explain the image enhancement properties of each operation in (6.1), we start first with its second term. Convolving the input image $f(k,l)$ with a weight kernel w, outputs a filtered version of $f(k,l)$, determined by the kernel's weight values. Then, subtracting $w * f(k,l)$ from $f(k,l)$ allows to extract the "details" of an image. The multiplication with the constant value c, results in image sharpening for $c = 1$ and high-boost filtering for values $c > 1$. Here, we consider $c = 1$. Finally, the sharpened image $g(k,l)$ is obtained by adding the extracted details to the input image. Note that in the convolution process, AND gates are used for multiplication. The architecture realizing the image sharpening filter is shown in Fig. 6.3, where the convolution kernel of Fig. 6.1 is used.

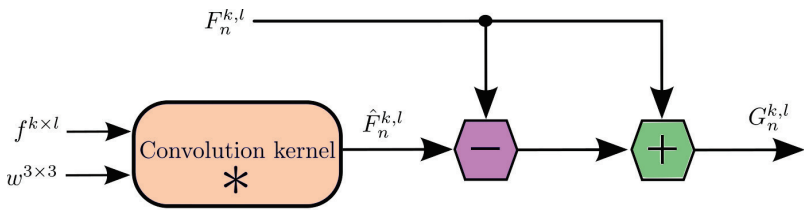


Figure 6.3: Image sharpening filter realized using the proposed non-scaling adder and subtracter. The convolution kernel is realized as shown in Fig. 6.1.

Proceeding to the experimental setup, we selected a gray-scale image assuming an 8-bit number

representation for each pixel and then we normalized their values to range $[0, 1]$. The normalization is based on stochastic numbers with sequence length $N = 2^k$, with $k = 6, \ldots, 10$. This also applies to the weight values of the 3×3 mask w, which are selected here to be 0.125. Note that all computations are conducted with simulations using MATLAB and Sobol sequences[46].

The computational accuracy of the proposed stochastic sharpening filter is evaluated with two metrics, the PSNR in dB and the SSIM. In Table 6.3, the results for typical values of N considered are shown, while in Fig. 6.4 a graphical illustration of the computations using the proposed architectures with stochastic sequence length $N = 256$ and a register of $m = 4$-bits is demonstrated.

Table 6.3: Computational Accuracy & Image Quality for the Image Sharpening Filter Realized using the Proposed Architectures

$N = 2^k$	2^6	2^7	2^8	2^9	2^{10}
PSNR (dB)	31.98	33.23	34.63	34.93	34.98
SSIM	0.950	0.960	0.966	0.967	0.967

Figure 6.4: Image Sharpening Filter. From left to right: a) MATLAB's Original Image, b) MATLAB's Image Sharpening calculation, c) Image Sharpening Filter realized with the proposed SC architectures. Sequence length $N = 256$ and register size $m = 4$-bit.

From a hardware perspective, the realization of the image sharpening filter requires the following computations: 1) a 3×3 convolution kernel, 2) a subtraction and 3) an addition. Among them, the convolution kernel is the largest computational block and its implementation using the proposed stochastic adder requires 8 adders and 9 AND gates for multiplication. For the 9 adders in total and the subtracter, a register size of $m = 4$-bits is used, as it does not degrade the accuracy of computations for stochastic sequences with length N up to 1024.

According to the results shown in Table 6.4, the advantage of the proposed approach is that of the area occupation, which is approximately 11% of the standard binary one's. On the other hand, the total energy consumed by the proposed approach is determined by N, which is selected according to the accuracy requirements. For instance, for $N = 2^7$ the total energy dissipated has moderate values equal to $142pJ$. Note that since N is determined by the sobol input sequence generators, they are not included in Table 6.4, but, they can be designed efficiently according to [46].

Table 6.4: Comparison of Hardware Resources for the Implementation of the Image Sharpening Filter

	Area (μm^2)	Power (mW)	Critical Path (ns)	Energy (pJ)
Proposed	1,093	0.62	1.8	1.11
Binary 8-bit	9,284	2.8	7.6	21.28

6.3 Median Filter

In this section we demonstrate the effectiveness of the proposed MAX and MIN and the Compact MAX and MIN architectures in a standard Digital Image Processing application. We use them as building blocks to implement a 3×3 median filter, which is typically used to reduce noise from images [24]. The kernel's structure is based on the sorting network presented in [43], shown in Fig. 6.5.

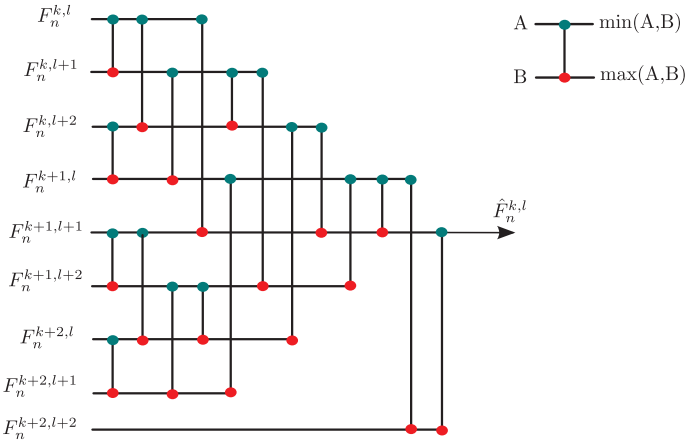


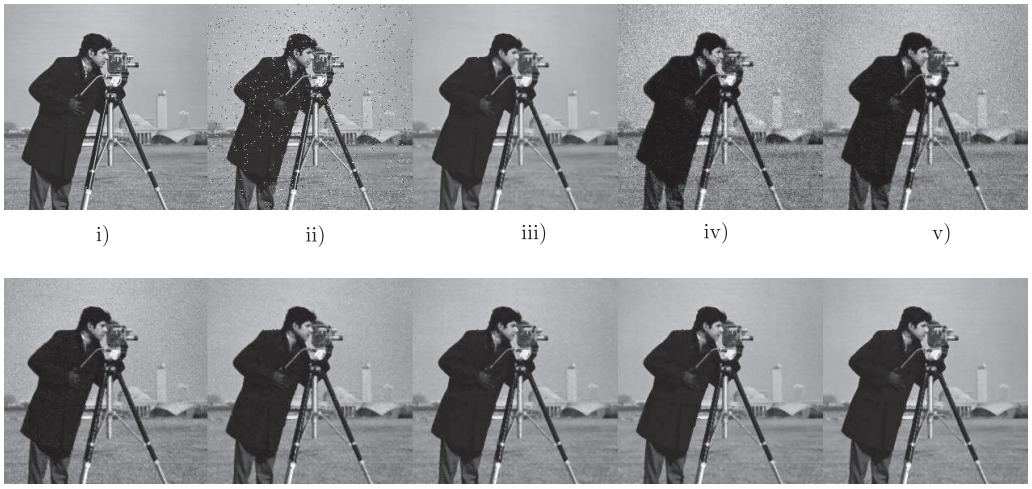
Figure 6.5: Sorting network realizing a SC median filter. Each node is realized using the proposed MAX and MIN architectures.

6.3.1 MAX and MIN

We first select a gray-scale image with 8-bit representation for each pixel and inject salt & pepper noise with a noise density of 0.02. Afterwards, we normalize the pixel values to range $[0, 1]$ so as to be processed in the SC domain. Given the fact that the accuracy of the architecture is based on the stochastic sequence length $N = 2^k$, we use typical values of $k = 4, 5, \ldots, 10$ and investigate their effect in computational accuracy with simulations using Matlab.

A graphical illustration of the computations using the proposed architectures to implement the median filter is shown in Fig. 6.6, while their respective accuracy results evaluated with the PSNR in dB are cited in Table 6.5. From sequence lengths $N = 64$ and forth, the proposed approach provides with a sufficient approximation of the median filter's computation, supported by the PSNR of 25.55 dB as well.

To proceed with the hardware resources, we note that the selected register size for each max and min we utilize is $m = 2$ as it does not degrade the computational accuracy and holds for all the lengths N



i) ii) iii) iv) v)

vi) vii) viii) ix) x)

Figure 6.6: Median Filtering with a 3×3 kernel, realized using the proposed MAX and MIN architectures for various sequence lengths N. From upper left to lower right: i) MATLAB's Original Image ii) MATLAB's Noisy Image with salt & pepper noise density 0.02 iii) MATLAB's filtered image iv) $N = 16$ v) $N = 32$ vi) $N = 64$ vii) $N = 128$ viii) $N = 256$ ix) $N = 512$ x) $N = 1024$. Register size used is $m = 2$ corresponding to $M = 4$ states

Table 6.5: Accuracy in PSNR of the realized 3×3 Median Filter using the Proposed Max and Min Architectures

Peak-Signal-to-Noise Ratio (PSNR) dB							
$N = 2^k$	2^4	2^5	2^6	2^7	2^8	2^9	2^{10}
Proposed	20.64	23.16	25.55	27.75	29.64	31.05	32.11

used in this specific application. In Table 6.6 the comparison between the traditional binary method using 8-bits is shown. It is important to note that since the SNG's LFSR size determines N, it also affects the overall hardware utilization. Moreover, given the fact that N is a parameter selected according to the accuracy requirements, the SNG's hardware resources are therefore not included in Table 6.6. Yet, their design can be optimized according to[88, 34].

According to the results, the proposed approach occupies almost half the binary method's area, which is its strong point. Depending on the required accuracy from the sequence length N, the hardware efficiency follows accordingly. For instance, for $N = 64$ which provides acceptable results, the total energy consumption has moderate values compared to the binary ones corresponding to $68.35\ pJ$.

Table 6.6: Hardware Resources for the Implementation of a 3×3 Median Filter using the Proposed MAX and MIN Architectures in Area (μm^2), Critical Path (ns), Power (mW) and Energy (pJ) per operation

	Area (μm^2)	Power (mW)	Critical Path (ns)	Energy (pJ)
Proposed	1,139	0.534	2.0	1.068
Binary 8-bit	2,470	2.295	2.2	5.049

6.3.2 Compact MAX and MIN

We first select a gray-scale image using 8-bit representation and then inject salt & pepper noise, with noise density of 0.05. The pixel values are afterwards normalized from range $[0, 255]$ to range $[0, 1]$ in order to be processed in the SC domain. To investigate the computational accuracy we consider typical stochastic sequence lengths $N = 2^k$, with $k = 5, \ldots, 10$ and calculate the PSNR in dB and the SSIM.

In Table 6.7 the calculated PSNR and SSIM results for typical values of N are shown. Moreover, in Fig. 6.7 for $N = 2^8$-bit sequences and a register size of $m = 3$-bits, the denoising with the 3×3 median filter using the proposed stochastic MAX and MIN is shown, compared to the computation using MATLAB. As one can observe, both the PSNR and the SSIM with values 33.81 and 0.90 respectively, demonstrate the increased computational efficiency of the proposed MAX & MIN architectures.

In Table 6.8, we present the hardware resources required to realize the 3×3 median filter using the proposed MAX & MIN architectures and the standard binary method, where we cite two different implementations. In the first one, we have not included the hardware resources for the generation of the input sequences as we want for our implementation to be flexible based on the designer's choice of inputs (pseudorandom, random etc). In the second one, we have included the hardware resources of an optimized SNG based on the sharing scheme in [34]. We note that we relaxed the register size requirements to $m = 3$-bits in the proposed MAX/MIN architectures for this specific task as our experiments showed that the accuracy of computations is not degraded.

From the results shown in Table 6.8, the advantage of the proposed method is the occupied area, which is reduced by approximately $40\%/28\%$ with/without SNGs when compared to the binary one. Moreover, with respect to the energy efficiency, it is expected that the stochastic sequence length N affects it directly; for example for $N = 64$-bit sequences the total energy consumed is $85.76/75.52pJ$ with/without SNGs resulting in moderate values compared to the binary method.

Figure 6.7: Denoising using a 3×3 median filter. From left to right: I) MATLAB's 8-bit noisy image with salt & pepper noise density 0.05, II) MATLAB's median filtered image, III) Proposed stochastic median filter with sequence length $N = 256$ and register size $m = 3$-bits.

Table 6.7: Computational Accuracy in PSNR and SSIM of the realized 3×3 Median Filter using the Proposed Max and Min Architectures

$N = 2^k$	2^5	2^6	2^7	2^8	2^9	2^{10}
PSNR (dB)	24.85	27.33	29.72	31.87	33.66	34.91
SSIM	0.59	0.73	0.83	0.90	0.94	0.96

Table 6.8: Hardware Resources for the Implementation of a 3×3 Median Filter using the Proposed Compact MAX & MIN Architectures in Area (μm^2), Critical Path (ns), Power (mW) and Energy (pJ)

	Area (μm^2)	Power (mW)	Critical Path (ns)	Energy (pJ)
Proposed	1,539	0.59	2.0	1.18
Proposed w/ SNG	1,812	0.67	2.0	1.34
Binary 8-bit	2,520	2.295	2.2	5.05

6.4 MAX Pooling

The down sampling is a standard process used in NNs as it reduces the dimensionality of the input image based on a max pooling kernel, allowing for the most important features to be preserved. Here, we consider a 2×2 max pooling kernel, realized using the proposed compact MAX architecture. Similar to the denoising task, we first select a grayscale image and normalize its pixel values to range $[0, 1]$. Then we select stochastic sequence lengths $N = 2^k$ with $k = 5, \ldots, 10$ and investigate the the kernel's performance considering the PSNR and SSIM metrics.

In Tables 6.9 and 6.10 the accuracy on computations and the required hardware resources to realize the 2×2 max pooling kernel are respectively cited. It is shown that using more than $N = 2^7$-bit sequence lengths, the downsampling of an image can be achieved accurately. This is also demonstrated in Fig. 6.8 for sequence length $N = 2^8$-bits and register size of $m = 4$-bits, where the max pooling is compared to the MATLAB's calculation.

For the reported hardware resources in Table 6.10, it is shown that the realization of the 2×2 kernel using the proposed stochastic MAX, occupies smaller area when compared to the binary one, approximately 40% less. This can benefit NN-based designs when 1) multiple copies of the kernel are required and 2) they have to operate in parallel.

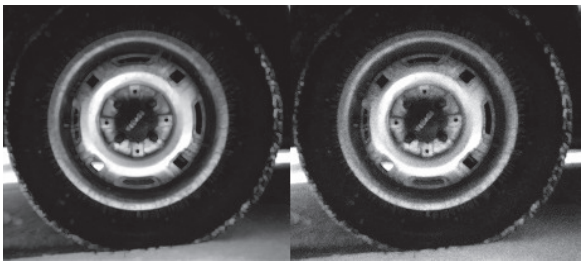


Figure 6.8: Down sampling using a 2×2 max-pooling kernel. Left: MATLAB's max pooling computation for 8-bit pixel representation, Right: max pooling kernel realized using the proposed compact MAX with sequence length $N = 2^8$ and register size $m = 4$-bits.

Table 6.9: Computational Accuracy in PSNR and SSIM of the realized 2×2 Max Pooling kernel using the Proposed Compact MAX Architecture

$N = 2^k$	2^5	2^6	2^7	2^8	2^9	2^{10}
PSNR (dB)	20.09	23.14	26.31	29.58	32.94	36.52
SSIM	0.58	0.72	0.82	0.90	0.94	0.97

Table 6.10: Hardware Resources for the Implementation of a 2×2 Max Pooling kernel using the Proposed Compact MAX Architecture in Area (μm^2), Critical Path (ns), Power (mW) and Energy (pJ)

	Area (μm^2)	Power (mW)	Critical Path (ns)	Energy (pJ)
Proposed	250.61	0.084	2.0	0.17
Binary 8-bit	432.71	0.058	2.2	0.12

6.5 Neural Network Design

Here we show how the proposed SCSD adder can be used to realize a SC artificial neuron along with the proposed MAX architecture.

6.5.1 SCSD Adder Artificial Neuron

The proposed SCSD adder's $0, 1$ output, enables further processing with the use of SC-based logic gates and/or SFSMs. The latter ones, are known to approximate non-linear functions within the context of SC, some of which belonging in the class of activation ones [15]. Therefore, placing a SFSM after the proposed SCSD adder, allows for the realization of an artificial neuron, as shown in Fig. 6.9.

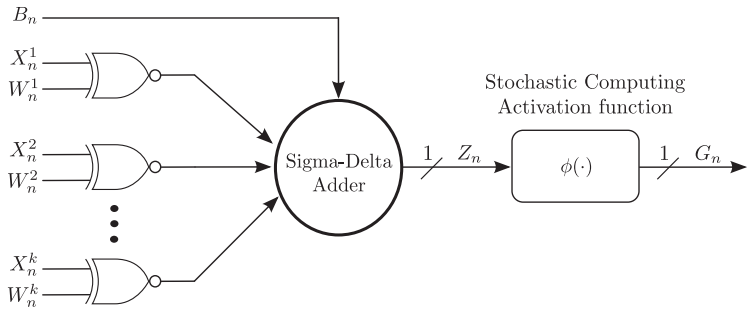


Figure 6.9: SC neuron realized using the proposed SCSD adder architecture shown in Fig. 3.14. The non-linear activation function is realized using any single-bit input/output Stochastic Finite-State Machine.

The SCSD neuron's current output value according to Fig. 6.9 is $G_n = \phi\left(Z_n\right)$, where $\phi(\cdot)$ represents the activation function, while using the time-average value of $\widetilde{Z}_n$ from (3.80), determines $\widetilde{G}_N$ as

$$\widetilde{G}_N = \phi\left(\widetilde{Z}_N\right) = \phi\left(\sum_{j=1}^{k} \widetilde{X}_N^j \widetilde{W}_N^j + \widetilde{B}_N\right). \tag{6.2}$$

Note that the time-average values $\widetilde{B}_N$, $\widetilde{W}_N$ of sequences $\{B_n\}_{n=1}^N$,$\{W_n\}_{n=1}^N$ refer to the bias and the weights of the neuron respectively.

Among the widely used activation functions in SC including the hyperbolic tangent (tanh), the exponential and the rectifier linear unit (ReLU) [15, 43], ReLU is the most popular one; it has the property of obtaining sparse representations by eliminating random fluctuations when used in NNs [50]. For an input sequence $\{X_n\}_{n=1}^N$ with time-average value $\widetilde{X}_N$, the non-linear activation function ReLU is defined as $\text{ReLU}(\widetilde{X}_N) \triangleq \max(0, \widetilde{X}_N)$. However, considering that the SC's range is constrained in $[-1, 1]$, in fact, a variation of the ReLU is realized in SC, namely the clipped ReLU, defined as

$$\text{Clipped ReLU}\left(\widetilde{X}_N\right) \triangleq \min\left(\max\left(0, \widetilde{X}_N\right), 1\right). \tag{6.3}$$

It is apparent that architectures realizing the max function are of high importance, with several architectures being explored in SC [39, 43, 89, 58, 81, 76]. Among them, we select the one proposed in [81], due to its property of combining short-sequence lengths with high computational accuracy.

To showcase the effectiveness of MAX architecture of Fig. 3.5 in approximating the clipped ReLU function of (6.3), we proceed as follows; we select $2 \cdot 10^2$ uniformly distributed input values $\hat{X} \in [-1, 1]$ and for 10^3 i.i.d. runs on each input value we calculate the mean of the output's time-average $\widetilde{A}_N$. For input sequence length $N = 256$-bits and a register size of $m = 4$-bits, the results are illustrated in Fig. 6.10. It is observed the MAX's output yields significant approximation results, with a slight deviation when the input value $\hat{X} \in [-0.05, 0.05]$.

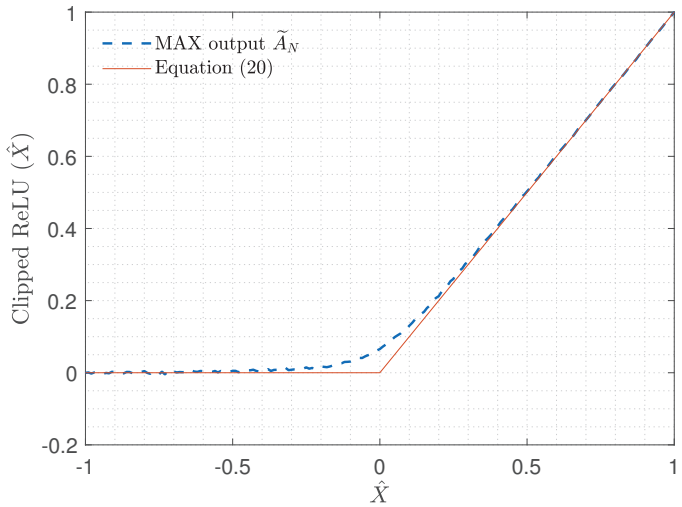


Figure 6.10: Approximating the clipped ReLU of (6.3) using the Stochastic MAX architecture of Fig. 3.5, for input values $\hat{X} \in [-1, 1]$, with 10^3 i.i.d. runs on each input value, sequence length $N = 256$ and register size $m = 4$-bits.

6.5.2 Forming a SC Multi-Layer Perceptron

The SCSD neuron in Fig. 6.9 can be used as a basic building block to form a SC-based MLP, which will be referred to as SCSD MLP. The MLP belongs to a class of feedforward NNs, with the network's architecture consisting mainly of three types of layers; input, hidden and output. In each layer, every unit is connected to every other one in the next layer, meaning that the MLP is fully connected. The layers within the MLP's architecture are shown in Fig. 6.12. Each layer within the MLP is explained as follows:

- *Input Layer:* The units of the input layer are the values of the input features, which in our case have been converted into sequences $\{X_n^j\}_{n=1}^N$, $j = 1, 2, \ldots, k$ by SNGs. The same applies to the values of the weights existing in all layers. For the sequence generation, Sobol number generators are considered as they require shorter sequence lengths when compared to the LFSR number generators when approximating a binary number as a stochastic one [47]. Moreover, they can be shared among the inputs and the weights respectively, as the neurons are independent to each other. An example of the sequence generation for a single neuron in the input layer is shown in Fig. 6.11.

- *Hidden Layers:* Each hidden layer is formed by stacking l_h neurons in parallel and the number of layers can be of any depth h. As such, considering the SCSD-based neuron of Fig. 6.9 the time-average output of each neuron in the MLP of Fig. 6.12 is described as

$$\widetilde{G}_N^{\lambda_\eta,\eta} = \phi\left(\widetilde{Z}_N^{\lambda_\eta,\eta}\right) = \min\left(\max\left(0, \widetilde{Z}_N^{\lambda_\eta,\eta}\right), 1\right), \tag{6.4}$$

  where $\phi(\cdot)$ is substituted with the clipped ReLU from (6.3), $\lambda_\eta = 1, 2, \ldots, l_\eta$ and $\eta = 1, 2, \ldots, h$ denote the current unit in each layer and the current hidden layer respectively.

- *Output Layer:* The MLP's output layer, obtains the desired predictions and is of length c, corresponding to the number of classes being learned. Assuming c classes to be learned, the time-average value of each output unit $c_o = 1, 2, \ldots, c$, is described as

$$\widetilde{O}_N^{c_o} = \sum_{\lambda_h=1}^{l_h} \widetilde{Z}_N^{\lambda_h,h} \widetilde{W}_N^{\lambda_h,h}. \tag{6.5}$$

  The output layer is the last processing stage within the MLP, so the result of (6.5) is realized in SC using a multiply-and-accumulate unit as shown in Fig. 6.13, with the result obtained after N clock cycles. In addition, the following should be noted: 1) the bit-width b' of the multiply-and-accumulate unit is determined by the number of the inputs to each unit in the output layer, l_h, and 2) the register length should be such that $N = 2^b$, where b refers to the length of the input binary number according to the definitions in Section 2 .

6.5.3 SCSD MLP Performance

The performance of the SCSD MLP is demonstrated with multiclass classification using the MNIST dataset [38]. It consists of $60,000$ training samples and $10,000$ testing samples of grayscale images

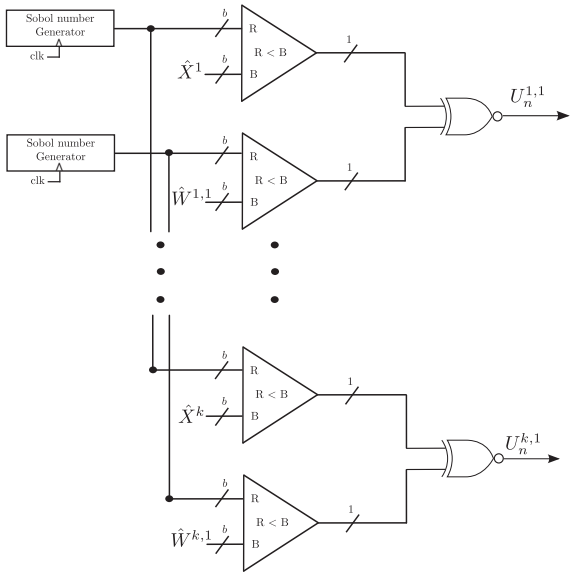


Figure 6.11: Example of sequence generation in the input layer, with $\hat{X}^j, \hat{W}^{j,1}, j = 1, \ldots, k$ corresponding to the values of the inputs and the weights of a single neuron respectively. The Sobol number generators are shared among the inputs and the weights respectively.

with pixel size 28×28, where each image represents handwritten digits with values ranging from 0 to 9, resulting in 10 classes in total. In the experiments, two network architectures are considered, set to $784 - 100/200 - 10$; a 784 input layer, a hidden layer of $100/200$ neurons and an output layer with 10 neurons corresponding to the dataset's classes.

With respect to the training procedure, in the hidden layers neurons employ the clipped ReLU defined in (6.3), whereas in the output layer the *softmax* activation function is used for the classification. The values of the inputs and the weights in all layers are constrained to range $[-1, 1]$ so as to be processed in the SC domain during the inference phase. Moreover, the use of bias in all layers is not considered as it would further tax on the hardware resources due to sequence generation. The training procedure for the weights' extraction is conducted using Python and the keras library.

Once the values of the weights are extracted from the training phase, they are used in the MLP for the inference (testing) procedure. For the evaluation of the MLP's inference along with the SCSD-based realization using different sequence lengths $N = 512, 1024$, fixed point (FxP) using $8, 16$ bits and Floating Point (FP) using 32 bits number representations are considered. The inference procedure of all MLPs is done using MATLAB.

6.5.3.1 Inference Accuracy

The classification performance of the MLP is evaluated using the accuracy metric, which is the ratio of the number of correct predictions to the total number of predictions. In the MNIST case, the total number of predictions corresponds to the number of testing samples, equal to $10,000$. To derive the accuracy of the SCSD MLP, the mean and standard deviation (std) of 10 independent runs over the testing samples

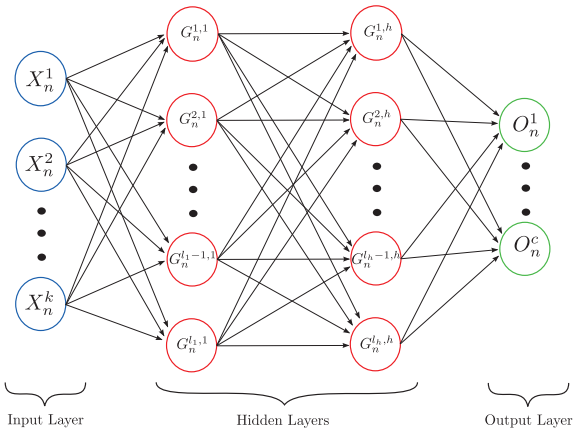


Figure 6.12: Multi-Layer Perceptron network architecture. Each hidden layer is realized using the proposed SC neuron of Fig. 6.9 containing the proposed SCSD adder architecture of Fig. 3.14.

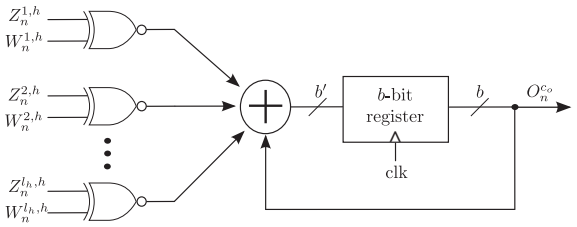


Figure 6.13: A multiply-and-accumulate processing block realizing each unit $O_n^{c_o}$ existing in the output layer. The result is obtained after N clock cycles.

are considered. This is due to the presence of the LFSR-based SNG existing in the MAX architecture where the input is compared to the value 0.5 (bipolar format of the value 0) so as to realize the clipped ReLU from (6.3). Sobol SNGs are reported to reduce the approximation accuracy of SFSMs over LFSR SNGs and therefore the latter ones are preferred [47]. The accuracy results of the proposed SCSD MLP accompanied by the FxP and FP realizations in percentages are cited in Table 6.11.

From the results shown in Table 6.11, it is observed that for the proposed SCSD MLP, when the sequence length N increases from 512-bit to 1024-bit length the std decreases, which is due to the improvement of the sequences' convergence in both network architectures. In addition, when the number of hidden layers is increased from 100 to 200, the percentage accuracy is improved by 0.82 and by 1.14 for sequence lengths $N = 512$ and $N = 1024$ respectively,

Compared to the FxP 8-bit realization it can be seen that the SCSD MLP with sequence length $N = 512$-bit achieves similar percentage accuracy results, increased by 0.49 and 0.33 for the networks $784 - 100 - 10$ and $784 - 200 - 10$ respectively, when the sequence length increases to $N = 1024$. Compared to the FxP 16-bit realization, the percentage accuracy of the SCSD MLP for the networks $784 - 100 - 10$ and $784 - 200 - 10$ is decreased by 0.26 and 0.12 using sequence length $N = 512$-bits respectively, but using $N = 1024$-bits, it is increased by 0.28 and 0.33 respectively. On the other hand, compared to the FP MLP realization, an expected reduction of approximately 1% in the percentage accuracy is observed

Table 6.11: Inference Accuracy in percentages $(\%)$ of the proposed SCSD, FxP and FP MLP realizations

	MLP 784-100-10	MLP 784-200-10
Proposed SCSD	(mean $\pm$ std)	(mean $\pm$ std)
Seq. Length $N = 512$-bit	95.41 ± 0.56	96.23 ± 0.68
Seq. Length $N = 1024$-bit	95.94 ± 0.41	96.68 ± 0.40
FxP 8-bit	95.45	96.24
FxP 16-bit	95.67	96.35
FP 32-bit	96.68	97.43

for the SCSD realizations, which is slightly higher for the FxP realizations.

Table 6.12: Hardware resources required for the realization of a 784-input neuron

	Area (μm^2)	Power (mW)	Delay (ns)	Energy (pJ)	ADP $(\times 10^3)$ $(\mu m^2 \times ns)$	EDP $(pJ \times ns)$
SCSD	44462.88	4.06	4	16.24	177.85	64.96
FxP 8-bit	703907	3.89	4.2	16.33	2956.40	68.61
FxP 16-bit	2368023.5	12.5	4.2	52.5	9945.70	220.5

6.5.3.2 Hardware Resources

The largest computational block within the MLP is the input layer considering its 784 inputs, making reasonable to investigate on its hardware resources. To this end, the SCSD and FxP neurons are described first using Verilog HDL and then they are fed into the Synopsys Design Compiler so as to extract their hardware resources using the FreePDK CMOS library at $45nm$ [77]. The following estimates are provided: 1) the total area in μm^2, 2) the average power consumption for the maximum operating frequency in mW, 3) the delay in ns, 4) the energy per operation in pJ, defined as the average power $\times$ delay product, 5) the area-delay product (ADP) in $\mu m^2 \times ns$ and 6) the energy-delay product (EDP) in $pJ \times ns$. The results for the hardware resources are cited in Table 6.12.

According to Table 6.12 it can be seen that the SCSD neuron reduces the area by 93.68% and by 98.12% of the 8-bit and 16-bit FxP ones respectively, resulting to $\times 15.83$ and $\times 53.25$ smaller area respectively. The ADP results follow the same direction as the area ones given that the delay between the SCSD and FxP neurons is similar, meaning that the SCSD neuron reduces by 93.98% and 98.21% the ADP of the 8-bit and 16-bit FxP neurons respectively. With respect to the energy per operation, the SCSD results in similar values compared to the 8-bit FxP, but, reduces the energy of the 16-bit FxP neuron by 69%. Similar results are also observed for the case of the EDP, where the SCSD neuron reduces the EDP of the 16-bit FxP neuron by approximately 70%.

It should be noted that the FP number representation introduces large hardware overhead when compared to the FxP number representation due to the presence of the FP multipliers [50, 49], making it less attractive for massive multiply-and-add operations. This is also the case for the SC multipliers when

the SNG sharing scheme is included [49]. Therefore, despite the FP MLP's accuracy improvement of approximately 1%, the hardware resources required to realize a FP neuron are not considered in Table 6.12.

6.5.3.3 Related Work

Table 6.13: Performance Comparison of SC-based MLPs in inference accuracy and hardware resources efficiency for the realization of the computational units

Work	Architecture	Seq. Length N	Relative error (ϵ) of Accuracy on MNIST	Hardware Efficiency (as a % of the FxP)	
				Area	Energy
Proposed SCSD	784-100-10 784-200-10	512/1024 512/1024	0.0131/0.0077 0.0123/0.0077	6.3% / 1.87% (FxP 8/16-bit)	94% / 30% (FxP 8/16-bit)
[37]	784-100-200-10	1024	0.0018*	50% (FxP 9-bit)	30% (FxP 9-bit)
[49]	784-200-100-10	4096	0.0133	40.7% (FxP 8-bit)	38% (FxP 8-bit)
[48]	784-128-128-10	1/gradient	0.0179**	7.3% (FxP 10-bit)	10% (FxP 10-bit)

* The highest accuracy score is used to calculate the relative error, taken from [37]
** The relative error of accuracy corresponds to the training procedure

For a fair comparison with the related work in SC in terms of classification performance, it is reasonable to consider the relative error of the inference accuracy given that the network architectures differ. It is defined as

$$\epsilon = \left| \frac{\text{Accuracy}_{FP} - \text{Accuracy}_{SC}}{\text{Accuracy}_{FP}} \right|, \tag{6.6}$$

where Accuracy_{FP} and Accuracy_{SC} are the classification accuracies using FP and SC number representations respectively. In terms of hardware efficiency, the area occupation and energy consumption in percentages over that of the FxP arithmetic are reported. The performance results are cited in Table 6.13.

In [37], a stochastic neuron is realized using an APC, followed by a FSM operating as the non-linear function *tanh* (BTanh) and implemented as binary up/down counter. The design of the BTanh is fixed, in the sense that the FSM's number of states affecting the *tanh*'s approximation are derived using numerical experiments for specific input sequence lengths. Moreover, the input sequence's bit-length driving the FSM affecting the number of the FSM's states is not considered.

According to Table 6.13, the relative accuracy error of the MLP in [37] is small, but, note that only the best score is reported, whereas in the proposed work and the rest ones, the average accuracy over independent runs is considered. For the hardware resources, in [37] a 200-input neuron is reported, utilizing 50% area and 30% energy of the FxP 9-bit realization On the other hand, the proposed 784-input neuron occupies only the $6.3\%/1.86\%$ of the $8/16$-bit FxP's area respectively, while the energy is $94\%/30\%$ of the $8/16$-bit FxP respectively. Moreover, the proposed SCSD adder opens the SC design space as it allows the use of single-bit output SFSMs, which consequently enables the realization of multiplications using AND/XNOR gates according to the SC number format used.

In [49], stochastic neurons in the input layer are realized by adopting the extended stochastic logic

(ESL) [18], whereas in the rest layers they are realized using APCs. Combining two different multiply-and-add processing methods allows the use of ESL-based backpropagation circuits, enabling online training. The ESL adder tree, however, requires a TMR binary search divider, resulting in large sequence lengths for its computation and stabilization phases[50, 49].

From the results in Table 6.13, the relative accuracy error of the MLP in [49] using $N = 4096$-bit sequence lengths is similar to that of the proposed SCSD one when $N = 512$-bit sequence lengths are used, resulting in a $\times 8$ faster convergence for the proposed approach. Comparing the area efficiencies, it can be seen that the proposed approach results in significant savings of the FxP $8/16$-bit realization, but, energy-wise the approach in [49] is better when 8-bits FxP are considered. From a design perspective, the proposed SCSD adder uses the standard SC encodings, whereas in [49] the use of ESL logic for the realization of the multiply-and-add units introduces design challenges [80].

Deviating from the previous approaches targeting multiply-and-add computational units, in [48], a signed SC gradient descent (SCGD) circuit capable of updating the value of the gradient and the weights was used in the training process of a MLP. From the computational accuracy results in [48], this method achieves significant training accuracy when compared to the FxP arithmetic with step size 2^{-10}. Note that in [48], only the FxP arithmetic is reported as an accuracy metric, hence for fair comparisons among the works cited, we consider FP arithmetic using a $784 - 128 - 128 - 10$ network, yielding 98.8% accuracy after 20 training epochs. Regarding the hardware resources, the SCGD circuits result in reduced area and energy compared to their FxP counterparts according to Table 6.13, but accuracy-wise the proposed approach results in smaller relative error.

7 Conclusion

In the first part of the dissertation, SC architectures for information processing systems were presented. Their operation principle was analysed in detail from a bit-level perspective, setting the foundations for the description of their behavior using SFSMs. To further explore their long-term stochastic dynamics, the architectures were modeled using MCs allowing for the derivation of their first-moment statistics, used to formally prove their proper operation at the limit. The MC modeling was extended to a general methodology that can be applied to any SFSM described as a Moore FSM, enabling the derivation of the second-moment statistics and the mean squared error, besides the first-moment ones. In addition, an extended overflow/underflow MC modeling procedure was introduced that can be used to estimate the number of states corresponding to the architectures' internal register size. From the above, it can be concluded that the contribution of the proposed architectures' theoretical analysis to the SC literature is the next step towards a better understanding of the behavior of SC elements relying on SFSMs, along with the acceleration of their design procedure.

In the second part of the dissertation, the performance of the proposed SC architectures is evaluated with comparisons with the existing approaches and realistic applications. With respect to the comparison with existing approaches, it was shown that the proposed architectures combine high accuracy outputs and small sequence lengths, which originates from the presence of the internal registers they use making their processing deterministic. When compared to existing approaches, their internal registers used by the proposed architectures impacts the power and energy consumption per clock cycle. Therefore, it would be reasonable to investigate to what extend a reduction of the number of states could balance the trade-off between the hardware resources and the high computational accuracy with small sequence lengths. Regarding the applications, it was shown that the proposed architectures are effective in executing several DSP operations, justified using standard computational performance metrics. Compared to the realizations using standard binary approaches, the proposed ones result in significant area savings with moderate total energy consumption values, due to the increased sequence lengths required for such applications. Hence, a further relaxation of the architectures' register size and its impact on the computational accuracy could be explored.

List of Publications

PhD Publications

International Peer-Reviewed Journals

1. **N. Temenos** and P. P. Sotiriadis, ”A Stochastic Computing Sigma-Delta Adder Architecture for Efficient Neural Network Design”, IEEE Journal on Emerging and Selected Topics in Circuits and Systems (JETCAS), *Accepted with minor revision*

2. **N. Temenos** and P. P. Sotiriadis, ”A Markov Chain Framework for Modeling the Statistical Properties of Stochastic Computing Finite-State Machines”, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Early Access 2022

3. P. P. Sotiriadis and **N. Temenos**, “Compact MAX and MIN Stochastic Computing Architectures”, Elsevier Integration, vol. 87, p.p. 194-204, 2022

4. **N. Temenos** and P. P. Sotiriadis, “Modeling a Stochastic Computing Non-Scaling Adder and its Application in Image Sharpening”, IEEE Trans. on Circuits and Systems II: Express Briefs, Early Access, vol. 69, no. 5, pp. 2543 - 2547, May 2022 **[Invited Journal]**

5. **N. Temenos**, A. Vlachos and P. P. Sotiriadis, “Efficient Stochastic Computing FIR Filtering using Sigma-Delta Modulated Signals”, MDPI Technologies, 10, 1, 2022 **[Invited Journal]**

6. **N. Temenos** and P. P. Sotiriadis, “Stochastic Computing MAX and MIN Architectures Using Markov Chains: Design, Analysis and Implementation”, IEEE Trans. on Very Large Scale Integration Systems, vol. 29, no. 11, pp. 1813 - 1823, Nov. 2021

7. **N. Temenos** and P. P. Sotiriadis, “Non-Scaling Adders and Subtracters for Stochastic Computing using Markov Chains”, IEEE Trans. on Very Large Scale Integration Systems, vol. 29, no. 9, pp. 1612-1623, Sept. 2021

International Peer-Reviewed Conferences

1. A. Vlachos, **N. Temenos**, P. P. Sotiriadis, ”Exploring the Effectiveness of Sigma-Delta Modulators in Stochastic Computing-Based FIR Filtering”, IEEE International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 2021 **[Nominated for Best Paper award]**

2. **N. Temenos** and P. P. Sotiriadis, ”Deterministic Finite State Machines for Stochastic Division in Unipolar Format”, IEEE International Symposium on Circuits and Systems (ISCAS), Virtual, 2020

3. **N. Temenos** and P. P. Sotiriadis, ”A New Technique for Stochastic Division in Unipolar Format”, IEEE International Conference on Modern Circuits and Systems Technologies (MOCAST), Thessaloniki, Greece, 2019

Additional publications

International Peer-Reviewed Journals

1. A. Temenos, **N. Temenos**, A. Doulamis and N. Doulamis, ”On the Exploration of Automatic Building Extraction from RGB Satellite Images Using Deep Learning Architectures Based on U-Net”, MDPI Technologies, 10, 1, 2022 **[Invited Journal]**

International Peer-Reviewed Conferences

1. A. Temenos, E. Protopapadakis, A. Doulamis and **N. Temenos**, ”Building Extraction from RGB Satellite Images using Deep Learning: A U-Net Approach.” In ACM The 14th PErvasive Technologies Related to Assistive Environments Conference (PETRA), June 2021.

2. C. Basetas, **N. Temenos**, and P. P. Sotiriadis, “Implementation of multi-step look-ahead sigma-delta modulators using ic technology”, In IEEE International Frequency Control Symposium (IFCS), Olympic Valley, USA, 2018 **[Nominated for Best Paper award]**

3. C. Basetas, **N. Temenos**, and P. P. Sotiriadis, “Comparison of two all-digital frequency synthesizers with a jitter removal circuit”, In IEEE International Frequency Control Symposium (IFCS), Olympic Valley, USA, 2018

4. **N. Temenos**, C. Basetas, and P. P. Sotiriadis. Hardware optimization methodology of multi-step look-ahead sigma-delta modulators. In IEEE International Conference on Modern Circuits and Systems Technologies on Electronics and Communications (MOCAST), Thessaloniki, Greece, 2018

5. C. Basetas, **N. Temenos**, and P. P. Sotiriadis, “An efficient hardware architecture for the implementation of multi-step look-ahead sigma-delta modulators”, In IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 2018

6. C. Basetas, **N. Temenos** and P. P. Sotiriadis, “Comparison of recently developed single-bit all-digital frequency synthesizers in terms of hardware complexity and performance”, In IEEE International Symposium on Circuits and Systems (ISCAS), Florence, Italy, 2018.

7. **N. Temenos**, C. Basetas, and P. P. Sotiriadis, “Noise shaping advantages of band-pass multi-step look-ahead sigma-delta modulators over conventional ones in signal synthesis”, In IEEE 4th Panhellenic Conference on Electronics and Telecommunications (PACET), Xanthi, Greece, 2017.

8. **N. Temenos**, C. Basetas, and P. P. Sotiriadis, “Efficient all-digital frequency synthesizer based on multi-step look-ahead sigma-delta modulation”, In IEEE 4th Panhellenic Conference on Electronics and Telecommunications (PACET), Xanthi, Greece, 2017.

9. C. Basetas, **N. Temenos**, and P. P. Sotiriadis, “Wide-band frequency synthesis using hardware-efficient band-pass single-bit multi-step look-ahead sigma-delta modulators”, In IEEE Int. Freq. Control Symp. & Europ. Freq. and Time Forum (IFCS-EFTF), Besançon, France, 2017.

10. C. Basetas, **N. Temenos**, and P. P. Sotiriadis, “Frequency synthesis using low-pass single-bit multi-step look-ahead sigma-delta modulators in quadrature upconversion scheme”, In IEEE Int. Freq. Control Symp. & Europ. Freq. and Time Forum (IFCS-EFTF), Besançon, France, 2017.

11. C. Basetas, P. P. Sotiriadis, and **N. Temenos**, “32-qam all-digital rf signal generator based on a homodyne sigma-delta modulation scheme”, In IEEE Int. Freq. Control Symp. & Europ. Freq. and Time Forum (IFCS- EFTF), Besançon, France, 2017.

References

[1] Hamdan Abdellatef et al. “Low-area and accurate inner product and digital filters based on stochastic computing”. In: *Signal Processing* 183 (2021), p. 108040.

[2] Kazi J Ahmed, Bo Yuan, and Myung J Lee. “High-accuracy stochastic computing-based fir filter design”. In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2018, pp. 1140–1144.

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

[4] Armin Alaghi and John P Hayes. “Exploiting correlation in stochastic circuit design”. In: *2013 IEEE 31st International Conference on Computer Design (ICCD)*. IEEE. 2013, pp. 39–46.

[5] Armin Alaghi and John P Hayes. “Fast and accurate computation using stochastic circuits”. In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2014, pp. 1–4.

[6] Armin Alaghi and John P Hayes. “STRAUSS: Spectral transform use in stochastic circuit synthesis”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.11 (2015), pp. 1770–1783.

[7] Armin Alaghi and John P Hayes. “Survey of stochastic computing”. In: *ACM Transactions on Embedded computing systems (TECS)* 12.2s (2013), pp. 1–19.

[8] Armin Alaghi, Cheng Li, and John P Hayes. “Stochastic circuits for real-time image-processing applications”. In: *Proceedings of the 50th Annual Design Automation Conference*. 2013, pp. 1–6.

[9] Armin Alaghi, Weikang Qian, and John P Hayes. “The promise and challenge of stochastic computing”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.8 (2017), pp. 1515–1531.

[10] Mohsen Riahi Alam, M Hassan Najafi, and Nima TaheriNejad. “Sorting in memristive memory”. In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* (2022).

[11] Mohammed Alawad and Mingjie Lin. “Fir filter based on stochastic computing with reconfigurable digital fabric”. In: *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines*. IEEE. 2015, pp. 92–95.

[12] Arash Ardakani et al. “VLSI implementation of deep neural network using integral stochastic computing”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.10 (2017), pp. 2688–2699.

[13] Timothy Baker and John Hayes. “Impact of autocorrelation on stochastic circuit accuracy”. In: *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE. 2019, pp. 271–277.

[14] Timothy J Baker and John P Hayes. “Bayesian accuracy analysis of stochastic circuits”. In: *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. 2020, pp. 1–9.

[15] Bradley D Brown and Howard C Card. “Stochastic neural computation. I. Computational elements”. In: *IEEE Transactions on computers* 50.9 (2001), pp. 891–905.

[16] Bradley D Brown and Howard C Card. “Stochastic neural computation. II. Soft competitive learning”. In: *IEEE Transactions on Computers* 50.9 (2001), pp. 906–920.

[17] Rahul Kumar Budhwani, Rengarajan Ragavan, and Olivier Sentieys. “Taking advantage of correlation in stochastic computing”. In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2017, pp. 1–4.

[18] Vincent Canals et al. “A new stochastic computing methodology for efficient neural network implementation”. In: *IEEE transactions on neural networks and learning systems* 27.3 (2015), pp. 551–564.

[19] Pong P Chu. *FPGA Prototyping by VHDL Examples: Xilinx MicroBlaze MCS SoC*. John Wiley & Sons, 2017.

[20] S Rasoul Faraji and Kia Bazargan. “Hybrid binary-unary hardware accelerator”. In: *IEEE Transactions on Computers* 69.9 (2020), pp. 1308–1319.

[21] S Rasoul Faraji et al. “Energy-efficient convolutional neural networks with deterministic bit-stream processing”. In: *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2019, pp. 1757–1762.

[22] Christiam F Frasser et al. “Fully Parallel Stochastic Computing Hardware Implementation of Convolutional Neural Networks for Edge Computing Applications”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2022).

[23] Brian R Gaines. “Stochastic computing systems”. In: *Advances in information systems science* (1969), pp. 37–172.

[24] R. C. Gonzalez, R. E. Woods, and S. L. Eddins. *Digital Image Processing Using Matlab*. Second. Gatesmark Publishing, 2009.

[25] Charles Miller Grinstead and James Laurie Snell. *Introduction to probability*. American Mathematical Soc., 1997.

[26] Warren J Gross and Vincent C Gaudet. *Stochastic Computing: Techniques and Applications*. Springer, 2019.

[27] Warren J Gross, Vincent C Gaudet, and Aaron Milner. "Stochastic implementation of LDPC decoders". In: *Conference Record of the Thirty-Ninth Asilomar Conference onSignals, Systems and Computers, 2005.* IEEE. 2005, pp. 713–717.

[28] Saransh Gupta et al. "COSMO: Computing with Stochastic Numbers in Memory". In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 18.2 (2022), pp. 1–25.

[29] Jie Han and Michael Orshansky. "Approximate computing: An emerging paradigm for energy-efficient design". In: *2013 18th IEEE European Test Symposium (ETS)*. IEEE. 2013, pp. 1–6.

[30] Kaining Han et al. "A Low Complexity SVM Classifier for EEG Based Gesture Recognition using Stochastic Computing". In: *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2020, pp. 1–5.

[31] Roger A Horn and Charles R Johnson. *Matrix analysis*. Cambridge university press, 2012.

[32] Kaveh Hosseini and Michael Peter Kennedy. *Minimizing Spurious Tones in Digital Delta-Sigma Modulators*. Springer Science & Business Media, 2011.

[33] Hideyuki Ichihara et al. "Compact and accurate digital filters based on stochastic computing". In: *IEEE Transactions on Emerging Topics in Computing* 7.1 (2016), pp. 31–43.

[34] Hideyuki Ichihara et al. "Compact and accurate stochastic circuits with shared random number sources". In: *2014 IEEE 32nd International Conference on Computer Design (ICCD)*. IEEE. 2014, pp. 361–366.

[35] Yuan Ji et al. "A hardware implementation of a radial basis function neural network using stochastic logic". In: *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2015, pp. 880–883.

[36] Honglan Jiang et al. "Approximate arithmetic circuits: A survey, characterization, and recent applications". In: *Proceedings of the IEEE* 108.12 (2020), pp. 2108–2135.

[37] Kyounghoon Kim et al. "Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks". In: *Proceedings of the 53rd Annual Design Automation Conference*. 2016, pp. 1–6.

[38] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[39] Vincent T Lee, Armin Alaghi, and Luis Ceze. "Correlation manipulating circuits for stochastic computing". In: *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2018, pp. 1417–1422.

[40] Vincent T Lee et al. "Architecture considerations for stochastic computing accelerators". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2277–2289.

[41] Vincent T Lee et al. "Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing". In: *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*. IEEE. 2017, pp. 13–18.

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

[43] Peng Li et al. “Computation on stochastic bit streams digital image processing case studies”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.3 (2013), pp. 449–462.

[44] Siting Liu, Warren J Gross, and Jie Han. “Introduction to dynamic stochastic computing”. In: *IEEE Circuits and Systems Magazine* 20.3 (2020), pp. 19–33.

[45] Siting Liu and Jie Han. “Hardware ODE solvers using stochastic circuits”. In: *2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC)*. IEEE. 2017, pp. 1–6.

[46] Siting Liu and Jie Han. “Toward energy-efficient stochastic circuits using parallel Sobol sequences”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.7 (2018), pp. 1326–1339.

[47] Siting Liu and Jie Han. “Toward energy-efficient stochastic circuits using parallel Sobol sequences”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.7 (2018), pp. 1326–1339.

[48] Siting Liu et al. “Gradient descent using stochastic circuits for efficient training of learning machines”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 37.11 (2018), pp. 2530–2541.

[49] Yidong Liu et al. “A stochastic computational multi-layer perceptron with backward propagation”. In: *IEEE Transactions on Computers* 67.9 (2018), pp. 1273–1286.

[50] Yidong Liu et al. “A survey of stochastic computing neural networks for machine learning applications”. In: *IEEE Transactions on Neural Networks and Learning Systems* 32.7 (2020), pp. 2809–2824.

[51] Yidong Liu et al. “An energy-efficient and noise-tolerant recurrent neural network using stochastic computing”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.9 (2019), pp. 2213–2221.

[52] Yidong Liu et al. “An energy-efficient online-learning stochastic computational deep belief network”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 8.3 (2018), pp. 454–465.

[53] Yin Liu and Keshab K Parhi. “Architectures for recursive digital filters using stochastic computing”. In: *IEEE Transactions on Signal Processing* 64.14 (2016), pp. 3705–3718.

[54] Yin Liu and Keshab K Parhi. “Computing polynomials using unipolar stochastic logic”. In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13.3 (2017), pp. 1–30.

[55] Yin Liu and Keshab K Parhi. “Computing RBF kernel for SVM classification using stochastic logic”. In: *2016 IEEE International Workshop on Signal Processing Systems (SiPS)*. IEEE. 2016, pp. 327–332.

[56] Yin Liu et al. “Machine learning classifiers using stochastic logic”. In: *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE. 2016, pp. 408–411.

[57] Yin Liu et al. “Synthesis of correlated bit streams for stochastic computing”. In: *2016 50th Asilomar Conference on Signals, Systems and Computers*. IEEE. 2016, pp. 167–174.

[58] Michael Lunglmayr, Daniel Wiesinger, and Werner Haselmayr. “Design and analysis of efficient maximum/minimum circuits for stochastic computing”. In: *IEEE Transactions on Computers* 69.3 (2019), pp. 402–409.

[59] Chengguang Ma, Shunan Zhong, and Hua Dang. “Understanding variance propagation in stochastic computing systems”. In: *2012 IEEE 30th International Conference on Computer Design (ICCD)*. IEEE. 2012, pp. 213–218.

[60] M Morris Manno and MD Ciletti. *Digital Design Global Edition*. 6th ed. 2018.

[61] Carl D Meyer. *Matrix analysis and applied linear algebra*. Vol. 71. Siam, 2000.

[62] Antoni Morro et al. “A stochastic spiking neural network for virtual screening”. In: *IEEE transactions on neural networks and learning systems* 29.4 (2017), pp. 1371–1375.

[63] M Hassan Najafi et al. “A reconfigurable architecture with sequential logic-based stochastic computing”. In: *ACM Journal on Emerging Technologies in Computing Systems (JETC)* 13.4 (2017), pp. 1–28.

[64] M Hassan Najafi et al. “Low-cost sorting network circuits using unary processing”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 26.8 (2018), pp. 1471–1480.

[65] M Hassan Najafi et al. “Performing stochastic computation deterministically”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27.12 (2019), pp. 2925–2938.

[66] M Hassan Najafi et al. “Time-encoded values for highly efficient stochastic circuits”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25.5 (2017), pp. 1644–1657.

[67] Vasileios Ntinas, Georgios Ch Sirakoulis, and Antonio Rubio. “Memristor-based Probabilistic Cellular Automata”. In: *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*. IEEE. 2021, pp. 792–795.

[68] Behraoz Parhami and Chi-Hsiang Yeh. “Accumulative parallel counters”. In: *Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*. Vol. 2. IEEE. 1995, pp. 966–970.

[69] Keshab K Parhi. “Analysis of stochastic logic circuits in unipolar, bipolar and hybrid formats”. In: *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE. 2017, pp. 1–4.

[70] Shanthi Pavan, Richard Schreier, and Gabor C Temes. *Understanding delta-sigma data converters*. John Wiley & Sons, 2017.

[71] Weikang Qian et al. “An architecture for fault-tolerant computation with stochastic logic”. In: *IEEE transactions on computers* 60.1 (2010), pp. 93–105.

[72] Ao Ren et al. “Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing”. In: *ACM SIGPLAN Notices* 52.4 (2017), pp. 405–418.

[73] Naman Saraf et al. “IIR filters using stochastic arithmetic”. In: *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE. 2014, pp. 1–6.

[74] Hyeonuk Sim, Saken Kenzhegulov, and Jongeun Lee. “DPS: Dynamic precision scaling for stochastic computing-based deep neural networks”. In: *Proceedings of the 55th Annual Design Automation Conference*. 2018, pp. 1–6.

[75] Yuhong Song et al. “BSC: Block-based Stochastic Computing to Enable Accurate and Efficient TinyML”. In: *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE. 2022, pp. 314–319.

[76] Paul P Sotiriadis and Nikos Temenos. “Compact MAX and MIN Stochastic Computing architectures”. In: *Integration* 87 (2022), pp. 194–204.

[77] James E Stine et al. “FreePDK: An open-source variation-aware design kit”. In: *2007 IEEE international conference on Microelectronic Systems Education (MSE’07)*. IEEE. 2007, pp. 173–174.

[78] Nikos Temenos and Paul P Sotiriadis. “A Markov Chain Framework for Modeling the Statistical Properties of Stochastic Computing Finite-State Machines”. In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2022).

[79] Nikos Temenos and Paul P Sotiriadis. “Modeling a Stochastic Computing Nonscaling Adder and its Application in Image Sharpening”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 69.5 (2022), pp. 2543–2547.

[80] Nikos Temenos and Paul P Sotiriadis. “Nonscaling adders and subtracters for stochastic computing using Markov chains”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29.9 (2021), pp. 1612–1623.

[81] Nikos Temenos and Paul P Sotiriadis. “Stochastic Computing Max & Min Architectures Using Markov Chains: Design, Analysis, and Implementation”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29.11 (2021), pp. 1813–1823.

[82] Nikos Temenos, Anastasis Vlachos, and Paul P Sotiriadis. “Efficient Stochastic Computing FIR Filtering Using Sigma-Delta Modulated Signals”. In: *Technologies* 10.1 (2022), p. 14.

[83] Pai-Shun Ting and John Patrick Hayes. “Stochastic logic realization of matrix operations”. In: *2014 17th Euromicro Conference on Digital System Design*. IEEE. 2014, pp. 356–364.

[84] Paishun Ting and John P Hayes. “Eliminating a hidden error source in stochastic circuits”. In: *2017 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*. IEEE. 2017, pp. 1–6.

[85] Anastasios Vlachos, Nikos Temenos, and Paul P Sotiriadis. “Exploring the Effectiveness of Sigma-Delta Modulators in Stochastic Computing-Based FIR Filtering”. In: *2021 10th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE. 2021, pp. 1–4.

[86] Shaowei Wang et al. “Weighted-Adder Based Polynomial Computation Using Correlated Unipolar Stochastic Bitstreams”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* (2022).

[87] Zhou Wang and Alan C Bovik. "Modern image quality assessment". In: *Synthesis Lectures on Image, Video, and Multimedia Processing* 2.1 (2006), pp. 1–156.

[88] Meng Yang et al. "Towards theoretical cost limit of stochastic number generators for stochastic computing". In: *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE. 2018, pp. 154–159.

[89] Joonsang Yu et al. "Accurate and efficient stochastic computing hardware for convolutional neural networks". In: *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE. 2017, pp. 105–112.

[90] Bo Yuan, Yanzhi Wang, and Zhongfeng Wang. "Area-efficient scaling-free DFT/FFT design using stochastic computing". In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 63.12 (2016), pp. 1131–1135.

[91] Aidyn Zhakatayev et al. "Sign-magnitude SC: Getting 10X accuracy for free in stochastic computing for deep neural networks". In: *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*. IEEE. 2018, pp. 1–6.