Stochastic Neural Networks: Components, Analysis, Limitations

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Florian Neugebauer

aus Traunstein, Deutschland

Hauptberichter:	Prof. Dr. Ilia Polian
Mitberichter:	Prof. Dr. John P. Hayes
	Prof. Dr. Weikang Qian

Tag der mündlichen Prüfung: 19. September 2022.

Institut für Technische Informatik der Universität Stuttgart

2022

Contents

Acknowledgments

I would like to thank Ilia Polian for giving me the opportunity to work on this thesis, and his great support and supervision over the years in Passau and Stuttgart. I would also like to thank my advisers John Hayes and Weikang Qian for reviewing this work. My thanks also go to Mirjam, Lothar and Helmut as well as all my colleagues at the institute for all of their support. A special thank you to Maël for being the best friend and officemate throughout the years, both inside and outside of work. I further want to thank Xiaoqing Wen for hosting me at the Kyushu Institute of Technology for one semester, which was a great experience. For his help and support during that time as well as his contribution to this work, I want to thank Stefan Holst. I would also like to thank Ponnanna Kelettira Muthappa, Junseok Oh and Vivek Vrujlal Vekariya, who worked on various SC-related projects with me during and after their studies at Stuttgart.

My biggest thanks go to my family and especially my parents. I never had to worry because I always knew I could count on your help and support in any situation.

Stuttgart, July 2022

Florian Neugebauer

Abstract

Stochastic computing (SC) promises an area and power-efficient alternative to conventional binary implementations of many important arithmetic functions. SC achieves this by employing a stream-based number format called Stochastic numbers (SNs), which enables bit-sequential computations, in contrast to conventional binary computations that are performed on entire words at once. An SN encodes a value probabilistically with equal weight for every bit in the stream. This encoding results in approximate computations, causing a trade-off between power consumption, area and computation accuracy. The prime example for efficient computation in SC is multiplication, which can be performed with only a single gate. SC is therefore an attractive alternative to conventional binary implementations in applications that contain a large number of basic arithmetic operations and are able to tolerate the approximate nature of SC. The most widely considered class of applications in this regard is neural networks (NNs), with convolutional neural networks (CNNs) as the prime target for SC. In recent years, steady advances have been made in the implementation of SC-based CNNs (SCNNs). At the same time however, a number of challenges have been identified as well: SCNNs need to handle large amounts of data, which has to be converted from conventional binary format into SNs. This conversion is hardware intensive and takes up a significant portion of a stochastic circuit's area, especially if the SNs have to be generated independently of each other. Furthermore, some commonly used functions in CNNs, such as max-pooling, have no exact corresponding SC implementation, which reduces the accuracy of SCNNs.

The first part of this work proposes solutions to these challenges by introducing new stochastic components: A new stochastic number generator (SNG) that is able to generate a large number of SNs at the same time and a stochastic maximum circuit that enables an accurate implementation of max-pooling operations in SCNNs. In addition, the

first part of this work presents a detailed investigation of the behaviour of an SCNN and its components under timing errors. The error tolerance of SC is often quoted as one of its advantages, stemming from the fact that any single bit of an SN contributes only very little to its value. In contrast, bits in conventional binary formats have different weights and can contribute as much as 50% of a number's value. SC is therefore a candidate for extreme low-power systems, as it could potentially tolerate timing errors that appear in such environments. While the error tolerance of SC image processing systems has been demonstrated before, a detailed investigation into SCNNs in this regard has been missing so far. It will be shown that SC is not error tolerant in general, but rather that SC components behave differently even if they implement the same function, and that error tolerance of an SC system further depends on the error model.

In the second part of this work, a theoretical analysis into the accuracy and limitations of SC systems is presented. An existing framework to analyse and manage the accuracy of combinational stochastic circuits is extended to cover sequential circuits. This framework enables a designer to predict the effect of small design changes on the accuracy of a circuit and determine important parameters such as SN length without extensive simulations. It will further be shown that the functions that are possible to implement in SC are limited. Due to the probabilistic nature of SC, some arithmetic functions suffer from a small bias when implemented as a stochastic circuit, including the max-pooling function in SCNNs.

Zusammenfassung

Stochastic computing (SC) verspricht eine flächen- und energieeffiziente Alternative zu konventionellen binären Implementierungen vieler arithmetischer Funktionen. SC erreicht dies durch Verwenden eines stream-basierten Zahlenformats, genannt Stochastic numbers (SNs), das bit-sequentielle Berechnungen ermöglicht, im Gegensatz zu Berechnungen im konventionellen Binärformat, die auf vollständigen Wörtern ausgeführt werden. Eine SN codiert einen Wert probabilistisch, wobei jedes Bit im Stream gleich gewichtet wird. Diese Codierung verursacht approximative Berechnungen, was zu einem Kompromiss zwischen Energieaufnahme, Fläche und Berechnungsgenauigkeit führt. Das Musterbeispiel für effiziente Berechnung in SC ist die Multiplikation, die mit nur einem einzelnen Gatter durchgeführt werden kann. SC ist deshalb eine attraktive Alternative zu konventionellen Binärimplementierungen für Anwendungen, die eine große Zahl an einfachen arithmetischen Operationen besizten und dabei die approximativen Berechnungen durch SC tolerieren können. Die diesbezüglich am häufigsten betrachtete Klasse von Anwendungen sind neuronale Netzwerke (NNs), mit convolutionellen neuronalen Netzwerken (CNNs) als Hauptziel für SC. In jüngerer Vergangenheit wurde dabei stetiger Fortschritt bei der Implementierung von SC-basierten CNNs (SCNNs) gemacht. Gleichzeitig wurden jedoch auch einige Herausforderungen identifiziert: SCNNs müssen große Mengen an Daten verarbeiten, die von konventionellem Binärformat zu SNs konvertiert werden müssen. Diese Konvertierung ist hardware-intensiv und verbraucht einen signifikanten Anteil der Fläche einer stochastischen Schaltung, insbesondere wenn die SNs unabhängig voneinander generiert werden müssen. Desweiteren besitzen einige häufig verwendete Funktionen in CNNs, wie beispielsweise die max-pooling Funktion, keine exakte SC Implementierung, was die Genauigkeit von SCNNs reduziert.

Der erste Teil dieser Arbeit präsentiert Lösungen für diese Herausforderungen in Form

neuer stochastischer Komponenten: Ein neuer stochastic number generator (SNG), welcher eine große Zahl an SNs gleichzeitig generieren kann, sowie eine Schaltung für die stochastische Maximumsfunktion, die eine genaue Implementierung von max-pooling Operationen in SCNNs ermöglicht. Darüber hinaus wird im ersten Teil dieser Arbeit eine detailierte Untersuchung des Verhaltens eines SCNNs und dessen Komponenten bei Timingfehlern präsentiert. Die Fehlertoleranz von SC is ein vielzitierter Vorteil, der durch die Tatsache zustande kommt, dass ein einzelnes Bit einer SN nur einen sehr geringen Beitrag zu ihrem Wert hat. Im Gegensatz dazu besitzen Bits in konventionellen Binärformaten unterschiedliche Gewichte und können bis zu 50% des Werts einer Zahl beitragen. SC ist daher ein Kandidat für extreme low-power Systeme, da es potentiell die in diesen Umgebungen auftretenden Timingfehler tolerieren kann. Während die Fehlertoleranz von SC Systemen in der Bildverarbeitung bereits gezeigt wurde, fehlte eine detailierte Untersuchung von SCNNs in dieser Hinsicht bis jetzt. Es wird gezeigt, dass SC nicht im Allgemeinen fehlertolerant ist, sondern dass sich SC-Komponenten unterschiedlich verhalten, sogar when sie die gleiche Funktion implementieren, und dass die Fehlertoleranz eines SC-Systems abhängig vom Fehlermodell ist.

Im zweiten Teil dieser Arbeit wird eine theoretische Analyse der Genauigkeit und Einschränkungen von SC-Systemen präsentiert. Ein existierendes Framework zur Analyse und Steuerung der Genauigkeit von kombinatorischen stochastischen Schaltungen wird auf sequentielle Schaltungen erweitert. Dieses Framework ermöglicht es einem Designer, die Effekte von kleinen Designänderungen auf die Genauigkeit eine Schaltung vorherzusagen, und wichtige Parameter, wie beispielsweise SN-Länge, ohne umfangreiche Simulationen zu bestimmen. Darüber hinaus wird gezeigt, dass die Funktionen, die in SC implementiert werden können, beschränkt sind. Durch die probabilistische Funktionsweise von SC leiden einige arithmetische Funktionen unter einem geringen Bias, wenn sie als stochastische Schaltung implementiert werden, darunter auch die max-pooling Funktion in SCNNs.

List of Figures

List of Tables

List of Abbreviations

CNN	Convolutional Neural Network
FSM	Finite State Machine
HOMC	Hardware-Oriented Max-pooling Circuit
iid	independent identically distributed
LFSR	Linear Feedback Shift Register
MAC	Multiply-accumulate
MC	Markov-Chain
MSE	Mean Squared Error
MUX	Multiplexer
NN	Neural Network
PCC	Probability Conversion Circuit
PRNG	Pseudo-random Number Generator
ReLU	Rectified Linear Unit
RNG	Random Number Generator
RNN	Recurrent Neural Network
SBoNG	S-Box based Number Generator
SC	Stochastic Computing
SCC	Stochastic Computing Correlation
SCNN	Stochastic Computing-based Neural Network
SN	Stochastic Number
SNG	Stochastic Number Generator
TRNG	True Random Number Generator
WBG	Weighted Binary Generator

Chapter 1

Introduction

The steady advance of the Internet of Things (IoT) has led to a significant growth in specialized hardware and software architectures. The large number of distributed devices in the IoT has made traditional central server based computing less attractive and less desirable. Centralized approaches struggle with the ever increasing number of devices and the massive increase in available data that has to be processed, causing a shift towards local data processing, called edge computing. By processing data close to its point of creation, network traffic and central processing capabilities can be reduced. Ultimately, this leads to energy savings due to reduced communication overhead and can also make results available faster. An example for the importance of the latter point is hazard detection in autonomous vehicles, where the sensor data has to be processed quickly in order to detect risks as early as possible.

On the other hand, local data processing requires corresponding processing capabilities and power on the device. However, the size and energy supply of the device might be heavily restricted, for example in environmental sensors that are not permanently connected to a power grid. Such sensors might operate with either a non-permanent internal power supply in the form of batteries or a small local power source such as solar cells. Small, low-power hardware is therefore necessary to enable local processing without sacrificing the longevity of such devices. Manufacturing in a very small technology node to improve power and area efficiency is theoretically possible, however it increases costs and may impact device performance, as smaller hardware nodes are more susceptible to environmental noise and fluctuations in supply voltage.

A different approach is taken in approximate computing. Not all applications require high precision computations to perform well. Many image processing applications such as edge detection can for example tolerate slightly inaccurate results, as long as the deviations are too small to be perceived by humans. Other applications such as Neural Networks (NNs) for sound and image classification have an inherent tolerance for approximate results. Such NNs usually only output a class label instead of a numerical value and as long as this label is correct, internal inaccuracies are not noticeable from an outside perspective.
By replacing area expensive hardware for exact high precision arithmetic with smaller, approximate arithmetic circuits, significant area and power savings are possible. Common practices are truncation, where lower significant bits are ignored during a computation, and function approximation, where a hard-to-compute function is approximated by one or more simpler functions. The most popular version of the latter is piecewise linear approximation, which replaces a non-linear function by a number of simple linear functions that are strung together.

Stochastic Computing (SC) is a type of approximate computing that replaces the conventional binary number format based on weighted bits with a probabilistic, stream based number format called Stochastic Number (SN). This allows certain arithmetic operations to be computed as sequential single-bit operations instead of simultaneous multi-bit operations in binary format. All bits of an SN have the same weight, which removes the need for complex circuitry to manage the varying bit weights in binary format. However, the probabilistic nature of SC causes operations to be generally inexact and bit stream based arithmetic can lead to longer computation times. Nevertheless, SC has shown promising results in a variety of the above mentioned applications, often due to its extremely small multiplication implementation, which is commonly cited as the prime example of SC's efficiency: In its simplest form, SC multiplication can be implemented using a single AND gate. In contrast to that, binary multipliers can comprise hundreds of gates and even approximate binary multipliers are often larger by orders of magnitude. Further examples are addition, which needs only a single 2-to-1 Multiplexer (MUX) and a number of commonly used activation functions in NNs, such as sigmoid, which are implemented with small finite state machines (FSMs).
The stream-based data format of SC has further advantages: The small significance of

each single bit makes an SN more resilient against transient faults than conventional binary numbers. A bit flip changes an SN's value only by the smallest possible amount, furthermore multiple bit flips in opposite directions cancel each other out. This makes SC potentially suitable for applications in noisy or hazardous environments. Moreover, it is generally possible to control the accuracy of a stochastic circuit by changing the SN length, therefore trading higher computation time for accuracy. In many other types of approximate computing this would require a modification of the circuit design, in SC it can be achieved by simply running the circuit for more clock cycles.

Opposed to these attractive benefits stand several challenges that SC faces: In general, inputs of an application, for example pixel values of images, are not given in SN format. Instead, they have to be converted from their original, commonly binary, format to SNs. This conversion procedure requires a source of entropy to introduce a probabilistic factor into the SNs, which is commonly achieved through the use of a (pseudo) random number generator ((P)RNG). The most commonly used RNGs in the context of SC are Linear Feedback Shift Registers (LFSRs) and low-discrepancy sequences, as they provide good, well controllable randomness for a relatively small area investment. However, compared to the previously mentioned efficient arithmetic blocks in SC, they make up a large portion of a stochastic circuit. In some cases, input conversion has even been shown to consume more than 50% of total area.

The second major limitation of SC is its restricted value range. SNs are based on probabilities and can therefore only represent values between 0 and 1 in their most basic format. A simple modification can be made to the SN format to cover negative numbers as well, thus increasing the range to -1 to 1. However, absolute values larger than 1 require major changes that result in much more complicated arithmetic circuits that strongly diminish the efficiency of SC overall.

Besides these fundamental limitations, further challenges for SC exist, both general and application specific. With the shift to NNs as the main target application of SC in recent years, new SC components for previously unsupported functions are needed. These encompass among others many non-linear functions that have to be implemented as sequential rather than combinational circuits, which puts an additional requirement on SNGs to reduce correlation of generated SNs as much as possible. Such SC-based NNs (SCNNs)

are often proposed for use in resource constrained environments, including low-power systems, due to the inherent error tolerance of SNs. While this error tolerance has been demonstrated for some SC systems such as edge detection applications before, the prevalence of new SC components with vastly different structures in SCNNs limits the applicability of previous findings in this regard. Furthermore, design of new SC components is for the most part done manually on a case-by-case basis and sometimes multiple different circuit implementations for the same function are possible. With the lack of automated tools to test and analyse correctness and accuracy of designs, this task is also usually left to the designer. A theoretical foundation for this analysis is necessary to facilitate the test procedure and reduce the reliance on extensive simulation.
This work presents solutions to several of the above mentioned challenges. Its main contributions are:

- A new PRNG specifically designed for SC is presented that decreases the cost of input conversion the more inputs a circuits has and reduces correlation of SNs.
- A circuit design for an SC maximum function is presented. It has improved accuracy compared to previously existing stochastic maximum functions and enables for the first time the accurate implementation of the widely used rectified linear unit (ReLU) activation function and the max-pooling operation in stochastic NNs.
- The timing error tolerance of SCNNs is extensively investigated. While error tolerance is often mentioned as an advantage of SC over binary computing, an in-depth investigation of this aspect in SCNNs has been missing so far, especially with regard to timing errors.
- It is shown that SCNNs show a resilience against adversarial attacks, which try to cause deliberate misclassifications in an NN.
- Previous work of the author's master thesis provided a theoretical framework to estimate the accuracy of a given combinational stochastic circuit. This work extends the framework to sequential stochastic circuits.
- Theoretical limitations of SC are investigated and it is proven that not all functions can be implemented exactly in SC, including all non-linear convex functions.

Chapter 2

SC Background

2.1 History of SCNNs and related works

Stochastic Computing was first introduced in the 1960s by Brian R. Gaines [Gai69] as an approach to machine learning and pattern recognition. He remarked that "the data input [in these applications] is, in some sense, redundant, and the machine does not have to make very fine discriminations based on small differences in the incoming signal" ([Gai69], p. 40). This property is also inherent to SNs, suggesting a potentially fitting application for SC. Gaines described basic SC components such as adders and multipliers and developed a counter based element called ADDIE that is able to implement division and square root among other functions. The initial interest in SC during this time period subsided quickly however, as the need for smaller and more efficient computing devices was solved by technological progress in transistor manufacturing. The benefits of SC were not considered big enough to outweigh its limitations. It took until the early 2000s for SC to become a field of active interest again, as Neural Networks quickly gained popularity and technological limitations were foreseeable in hindering the implementation of ever larger NNs.

In 2001, Bradley D. Brown and Howard C. Card presented the first major SC implementation of a small two layer neural network for the task of optical Magnetic Ink Character Recognition (MICR) under noisy conditions [BC01a] [BC01b]. While this network was much smaller than currently used NNs, it introduced basic SCNN building blocks that are still in use today. Shortly after, one of the few physically manufactured SCNN chips was

presented in [SNA+03], implementing a Boltzmann machine. After several years without significant advances in SCNNs, a shift to stochastic convolutional NNs took place, with several works in quick succession. These works introduced new concepts such as hybrid SC-binary addition and modification to Brown's SC activation function [KKY+16]. Together with further advances such as a stochastic max-pooling circuit [RLD+17] and an approximate SC ReLU activation function [LLR+18], these concepts enabled the implementation of larger SCNNs, from the standard LeNet5 architecture to AlexNet and ImageNet. Several further works combined and modified these components to build various types of SCNNs and NN accelerators, e.g. [LWLH18], [HGT+19] and [SL17]. Due to their low-power arithmetic operations, SCNNs are a promising candidate for near-sensor and resource constrained environments, and a number of publications in this area have been made in recent years, including [LAH+17], [HPB+19] and [FNL+19]. The above mentioned works focus mostly on the design aspect of SCNNs, but several works focusing on FPGA implementations of these networks also exist, among others [SGMA15], [KMM+17] and [MNPH20]. The body of work on SCNNs continues to grow, expanding also to RNNs in recent years [LLLH19] [MZWH19], which are however not the focus of the present work. While the implementation and design details in these works vary widely, the basic concepts of SNs and probabilistic computation is common to all of them.

2.2 Basic components and operations

2.2.1 Number representation and basic operations

The basic operating principle of SC is to convert the bit-parallel arithmetic operations of conventional binary number formats into bit-sequential operations of SNs, thereby generally trading higher computing time for lower circuit area. SNs are bit streams of variable length n that encode a given probability value. Similar to different binary number representations such as sign-magnitude and Twos-complement, SNs come in different formats. The most commonly used SN formats are unipolar and bipolar.

Definition 2.1. A Stochastic Number is a bit stream of length n with n_1 1s and n_0 0s and has the value $\frac{n_1}{n}$ in unipolar format and the value $\frac{n_1-n_0}{n}$ in bipolar format.

It follows directly from definition 2.1 that a unipolar SN covers values in the interval $[0, 1]$ with a precision of $\frac{1}{n}$ and that its value is equal to the probability of any single one of its bits being one. Bipolar SNs extend the range of representable values to $[-1, 1]$ and in turn have a reduced precision of $\frac{2}{n}$. For a bipolar SN B the relation between its value b and the probability p_B of any one of its bits being 1 is

$$p_B = \frac{b+1}{2}. \tag{2.1}$$

Besides these two main SN formats, several others have been proposed but are not widely used for practical purposes. One example is the Extended Stochastic Logic (ESL) format introduced in [CMO+15], which consists of a fraction of two bipolar SNs. ESL removes the value range restriction of unipolar and bipolar SNs and was supposed to facilitate the implementation of SCNNs, as it would remove the need to downscale NN weights. However it complicates addition significantly, as it requires computation of a common denominator and was therefore not adopted as an alternative. Another notable mention is the inverse bipolar format, which defines the value of an SN as $\frac{n_0-n_1}{n}$ and is used in stochastic computing synthesis through spectral transformation [AH15]. It does not differ significantly from the regular bipolar format with regards to implementation of arithmetic functions and is therefore not relevant outside of synthesis. For the remainder of this work, only unipolar and bipolar formats are used.

An important property of SNs are equally weighted bits throughout their entire length. In contrast to conventional binary formats, no high or low significance bit positions exist, which has several major implications: Firstly, all SNs with the same number of 1s and 0s have the same value, as the order of bits has no influence. However, they should not be treated as being equivalent. The order of bits can have influence on the behaviour of certain sequential stochastic circuits, as will be shown in later sections of this work. Secondly, SNs have high tolerance towards bit flips, as each flip changes the value of an SN by exactly $1/n$ in unipolar, respectively $2/n$ in bipolar format. Moreover, multiple bit flips in opposite directions will cancel out. Finally, equal weight of each bit enables the computation of some basic arithmetic operations such as multiplication with single bit input combinational circuits, as there is no need to take care of carry bits.

SC Multiplication is commonly cited as vastly more effective than its binary counterpart. In unipolar format it is implemented by a single AND gate, as the output of an AND gate is 1 if and only if both inputs A and B are 1. The probability p_{AND} of the output being 1 is therefore $p_A \cdot p_B$, if A and B are independent random variables. In bipolar format an XNOR gate is used instead:

$$\begin{aligned} p_{XNOR} &= P(\{A=1\} \wedge \{B=1\}) + P(\{A=0\} \wedge \{B=0\}) \\ &= P(\{A=1\}) \cdot P(\{B=1\}) + P(\{A=0\}) \cdot P(\{B=0\}) \\ &= \frac{a+1}{2} \cdot \frac{b+1}{2} + \left(1 - \frac{a+1}{2}\right) \cdot \left(1 - \frac{b+1}{2}\right) \\ &= \frac{ab+1}{2} \end{aligned} \tag{2.2}$$

which, according to (2.1), corresponds to a bipolar SN with value ab.

The second basic SC operation is scaled addition, which can be implemented with a 2-to-1 multiplexer (MUX) for both SN formats: Assume without loss of generality that the select input S of a MUX has a probability p_S of being 1 and that the MUX routes input SN A to the output if $S = 1$ and B otherwise. The output value o_{MUX} of the MUX is then $o_{MUX} = a \cdot p_S + b \cdot (1 - p_S)$. Basic SC addition is therefore inherently a downscaled, weighted addition. As SN values are restricted within $[-1, 1]$, a non-scaled addition with SN output is generally not possible. This limitation poses problems in some applications, as will be described in later chapters. On the other hand, it provides an efficient way to implement inner product computation, which has been exploited in the design of stochastic digital filters [WHCE16] [ISI+16].

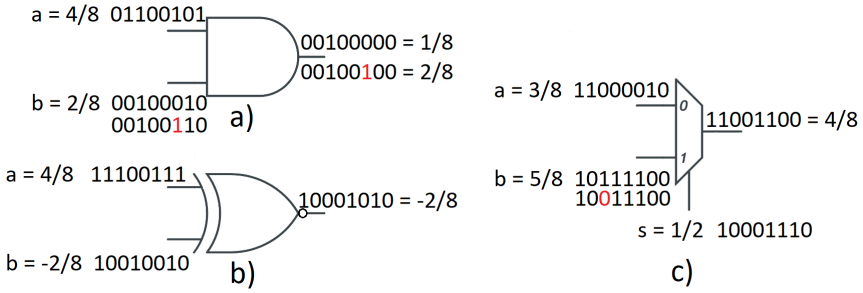

Figure 2.1: Basic SC operations: a) unipolar multiplication with example for error affected computation, b) bipolar multiplication, c) unipolar addition with example for error affected computation.

Figure 2.1 shows examples for basic multiplication and addition in both SN formats. In figure 2.1a) unipolar multiplication in an error free case and an error affected case is shown. The error only changes the result by $\frac{1}{n}$, i.e. the smallest possible amount. In the bipolar multiplication example b) one of the sources of inaccuracy in SC is demonstrated, as the correct result of $-\frac{1}{8}$ cannot be represented with bipolar numbers of length 8. Subfigure c) shows that in some cases errors in SC operations do not affect the result at all.

The inherent scaling of MUX-based addition poses a problem in applications with large sums, as the resulting absolute values decrease and relative errors therefore increase. As a solution to this issue, the use of SC-binary hybrid adders, mostly in the form of parallel counters, has been proposed [TH14]. These adders receive SNs as inputs and produce a stream of integers in conventional binary format as outputs. More specifically, an adder of this form with k input SNs $I_1, \cdots I_k$ of length n will output the sum $s_j = \sum_{i=1}^{k} I_{i,j}$ of all input bits in clock cycle $1 \leq j \leq n$. These hybrid adders avoid the scaling problem but in turn move the computation from the stochastic into the binary domain. They are mostly used in SCNNs, which will be described in more detail in subsection 2.2.5.

2.2.2 Correlation and SN generation

Both addition and multiplication in SC only work as described if inputs are independent of each other. In the context of SC, dependency between SNs is called correlation and is an important property that can be both beneficial and detrimental. Undesired correlation can change the intended functionality of a stochastic circuit. For example, assume that the input SNs A and B of a unipolar multiplier are correlated in a way that for each bit A_i and B_i it holds that $A_i = 1 \Rightarrow B_i = 1$ and overall $p_A \leq p_B$. In other words, all 1s in A overlap with 1s in B. Instead of multiplying A and B as intended, the multiplier will output an SN that is identical to $\min\{A, B\} = A$.

On the other hand, deliberate correlation can be exploited for the implementation of various functions. Said unipolar multiplier for example turns into a minimum function, if its inputs are always correlated. Further correlation based SC designs include the CORDIV divider [CH16] and an SC implementation of the Robert's Cross edge detection algorithm [ALH13], where the use of an XOR gate with correlated inputs is used to compute $|A - B|$.

A commonly used metric to measure correlation between two SNs is the SC Correlation (SCC) defined in [AH13]:

$$SCC(X,Y) = \begin{cases} \frac{ad-bc}{n \cdot \min(a+b,a+c)-(a+b)(a+c)} & \text{if } ad > bc \\ \frac{ad-bc}{(a+b)(a+c)-n \cdot \max(a-d,0)} & \text{otherwise} \end{cases} \tag{2.3}$$

where X and Y are two n-bit SNs and a is the number of overlapping 1s, b is the number of overlapping 1s in X with 0s in Y, c is the number of overlapping 0s in X with 1s in Y and d is the number of overlapping 0s. SCC ranges from -1 to 1, whereby an SCC of 1 means that all 1s overlap, and an SCC of -1 means that no 1s overlap. For example, the XOR gate from [ALH13] mentioned above assumes an input SCC of 1. Not all SCC values may be possible for any given pair of SNs. If the sum of 1s in both SNs is larger than n, an SCC of -1 is not possible, as $a > 0$ holds for all permutations of bits in X and Y. In general, two inputs of a circuit are considered uncorrelated if the average SCC of their SNs is 0, positively correlated if the average SCC of their SNs is larger than 0, and negatively correlated otherwise.
It has been shown that a smaller absolute SCC does not always mean higher computational accuracy, especially when SN pairs of one very small and one very large absolute value are involved [HMHAA21]. Comparing SCCs of specific SN pairs does therefore not in general allow conclusions about the correctness or accuracy of a stochastic circuit. However, averaging SCC over a large number of SN pairs is still a useful method to analyse a stochastic circuit, for example when comparing SNG sharing methods [IIS+14], as it identifies systematic correlation and therefore potential deviations from the intended functionality.

By definition, SCC and its alternative zero correlation error (ZCE) metric [HMHAA21] are always a relation between two SNs. So far, no generalized concepts of correlation between three or more SNs have been developed. However, stochastic circuits with more than two inputs that also rely on specific correlation between more than two inputs have not been designed so far, a metric for pairwise correlation has therefore been sufficient. Stochastic addition via MUX for example does not require uncorrelated data inputs, only the select input has to be uncorrelated to each of the data inputs. Correlation between two or more SNs is sometimes also referred to as cross-correlation in order to distinguish it from the

concept of autocorrelation, which refers to the dependency of subsequent bits within a single SN.

Autocorrelation of SNs is commonly measured using general, non SC-specific metrics such as the autocorrelation function from [Bro06] or the Box-Jenkins function [BJRL15]. Autocorrelation is an important factor in sequential stochastic circuits, e.g. FSM-based or with feedback loops, as they usually operate under the assumption that subsequent bits of their input SNs are independent of each other, i.e. the input is not autocorrelated. [Bro06] defines the auto-covariance of an SN $X = X_1, ..., X_n$ as

$$r_X(k) = \mathbb{E}(X_t X_{t+k}) - \mathbb{E}(X_t)^2 \tag{2.4}$$

for a distance k. [Bro06] also provides a way to estimate r_X that does not require knowledge of expected values:

$$r_X(k) \approx \frac{1}{n}\sum_{t=1}^{n-k} X_t X_{t+k} - \left(\frac{1}{n}\sum_{t=1}^{n} X_t\right)^2 \tag{2.5}$$

After normalization, the autocorrelation function is given by

$$\rho_X(k) = \frac{r_X(k)}{\mathbb{E}(X_t^2) - \mathbb{E}(X_t)^2} \tag{2.6}$$

which allows comparison of autocorrelations of SNs (although it requires knowledge of their expected values again).

The Box-Jenkins function defines the autocorrelation factor A_k similarly:

$$A_k = \frac{\sum_{i=1}^{n-k} (X_i - \mathbb{E}(X))(X_{i+k} - \mathbb{E}(X))}{\sum_{i=1}^{n} (X_i - \mathbb{E}(X))^2} \tag{2.7}$$

An autocorrelation factor of $A_k = 0$ signifies that groups of bits with distance k are statistically independent. Commonly $k = 1$ is the most important distance in SC due to particular decorrelation mechanisms explained further below.

Both cross-correlation and autocorrelation are important to consider when it comes to generating SNs, the task of stochastic number generators (SNGs). The basic design of an SNG is shown in figure 2.2. It consists of a k-bit random number generator (RNG) whose output R is compared to a k-bit constant input $B \in [0, 1]$ in unsigned binary format every

clock cycle. If R is uniformly distributed in $[0, 1]$, the output bit of the comparator is 1 with probability B. The comparator is therefore also referred to as a probability conversion circuit (PCC) in this context. After n clock cycles, a unipolar SN with value B will be generated (for generating bipolar SNs, B has to be adjusted according to eq. 2.1). In early works, e.g. by Gaines [Gai69] [Gai67], it was proposed to use fast toggling (several times higher than the circuit's operating frequency) flip-flops in order to achieve a signal probability of $\frac{1}{2}$ when sampled. They could then be combined to k-bit numbers for comparison, or transformed directly into SNs of constant value, e.g. to generate constant parameters for polynomials. However, achieving a signal probability of exactly $\frac{1}{2}$ in this manner is difficult and unreliable. SNGs therefore shifted to simpler pseudo RNGs (PRNGs), most prominently linear feedback shift registers (LFSRs). Besides LFSRs, other (P)RNGs have been proposed for use in SNGs, among others analogue RNGs [KA17] and more recently low-discrepancy sequence generators [NJLR19] [LH17] and emerging devices like memristors [KLZ14]. The use of different PCCs other than the comparator of the standard design has also been investigated, for example MUX chains [BC01a] and weighted binary generators [GK88], but their impact on SN generation is smaller than that of different random number sources.

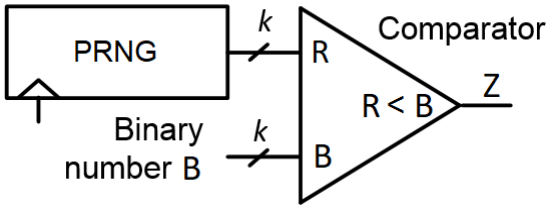

Figure 2.2: Standard SNG design consisting of a pseudo random number generator and a comparator.

SNGs can make up a significant portion of stochastic circuits area, in some cases more than 50% [ISI+16]. A major goal in SC design is therefore to reduce this overhead without sacrificing accuracy due to increased SN correlation. A simple approach is SNG sharing, whereby the RNG component is connected to two or more PCCs of different circuit inputs, if those inputs are not part of a common operation, or are inputs of correlation insensitive components such as NMax [NPH19], which will be covered in more detail later. Another

popular method is isolation, whereby SNs are generated using the same RNG and then decorrelated by delaying some of the SNs. This can be done by simply passing it through a flip-flop. Under the assumption that subsequent bits of an SN are independent of each other (i.e. the SN is not autocorrelated for distance 1), this will effectively make the SNs independent of each other. Figure 2.3 shows this process at the example of a squaring operation with shared SNGs. The initial state probability of the flip-flop does in general not match the value of the input SN and will introduce a small error in the calculation. Granting the circuit a number of warm-up cycles equal to the maximum number of isolators on any one circuit path can solve this issue. Figures 2.3c) and d) demonstrate that isolators behave differently depending on their position in the circuit. Cascading two copies of a circuit that computes $f(x) = x^2$ does not lead to a circuit with the function $g(x) = x^4$ as one could assume.

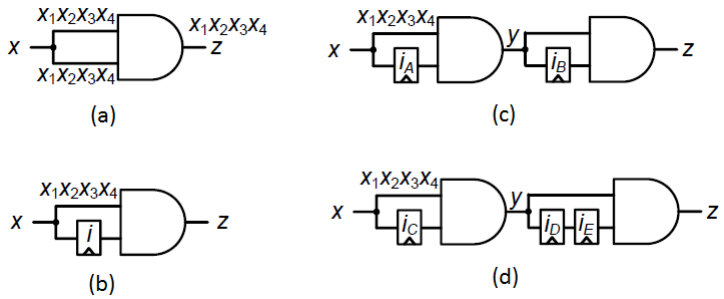

Figure 2.3: Examples for decorrelation via isolators: a) incorrect squaring operation due to SNG sharing. b) correctly decorrelated squaring operation. c) Two subsequent squaring circuits build a circuit for $f(x) = x^3$. d) Adding one more isolator leads to $g(x) = x^4$.

Unfortunately, the necessity of using small, simple PRNGs to avoid unacceptable area overhead means that the assumption of independence between bits that is in theory necessary for isolation generally does not hold. For example, subsequent states of an LFSR are obviously not independent of each other. This central problem of SNG design is covered in more detail in section 3.2.

2.2.3 Random number sources in SC

LFSRs have been considered as a standard source of random numbers in SC due to their low cost and fixed sequence length, which depends on an LFSR's characteristic feedback

polynomial. A k-bit LFSR with a primitive feedback polynomial has a sequence length of $2^k - 1$. If it is used to generate an SN of the same length, it will do so with an error of at most $\frac{1}{2^k-1}$, because it will generate every k-bit number in the half open interval $(0, 1]$ exactly once. The all-0 state can be added to any LFSR with little additional hardware to remove this error as well. However, this is commonly not done, as this initial error during SN generation is small compared to the errors commonly introduced by other sources of inaccuracy in a stochastic circuit and removing it is not worth the additional hardware overhead. While SN generation with LFSRs is highly accurate, operations using these SNs are generally not.

To guarantee accuracy of some operations, especially multiplication, PRNGs based on deterministic sequences have been proposed for use in SC. In an early work, pseudo noise (PN) sequences were considered [GK88]. To generate an SN of length n, several basic PN sequences are combined. In the first step, PN sequences with values $p_1 = 0.5$, $p_2 = 0.25$, $p_3 = 0.125$,... are constructed according to the following procedure: Sequence $A_1 = a_{1,1}, ..., a_{1,n}$ with value p_1 is generated using the output bit of an LFSR with corresponding sequence length. Further basic PN sequences A_2, A_3,... are constructed from A_1 in such a way that no 1s overlap in any position of the sequences, i.e.

$$\sum_{i=1}^{m} a_{i,k} \leq 1 \quad \forall k \in \{1, ..., n\} \tag{2.8}$$

for m basic sequences, which are required to generate SNs with m-bit precision. This is achieved by setting $a_{2,k} = \overline{a_{1,k}} a_{1,k+1}$, $a_{3,k} = \overline{a_{1,k} a_{1,k+1}} a_{1,k+2}$,... These basic sequences can then be combined through bitwise OR operations to generate sequences with a value equal to the sum of the basic sequences, as shown in table 2.1.

Instead of a comparator like in the standard SNG design, the PN-based SNG requires some circuitry to select the proper basic sequences for a given input value. Due to the non-overlapping 1s in the basic sequences, PN sequences generate SNs with the same accuracy as the LFSR+comparator design.

A more recent approach to SN generation similar to PN sequences are low discrepancy sequences such as van der Corput, Sobol and Halton sequences. Low discrepancy sequences are also called quasirandom sequences, as they are sometimes used for generating

Table 2.1: Generation of an SN with value 0.625 using PN sequences.

k	A_1	A_2	A_3	$X = 0.625(A_1 + A_3)$
1	0	1	0	0
2	1	0	0	1
3	0	1	0	0
4	1	0	0	1
5	1	0	0	1
6	1	0	0	1
7	0	0	1	1
1	0	1	0	0

uniformly distributed numbers. Despite this terminology and their use, these sequences are not random or even pseudo random, but follow a strictly deterministic pattern. An advantage over pseudorandom sequences are their more evenly distributed sequences, which lead to a highly uniform coverage of the $[0, 1]$ interval even for a small sequence length. Figure 2.4 shows a comparison of the coverage of $[0, 1]^2$ with points from Halton sequences, LFSRs and the Mersenne Twister [MN98], a software-based PRNG.

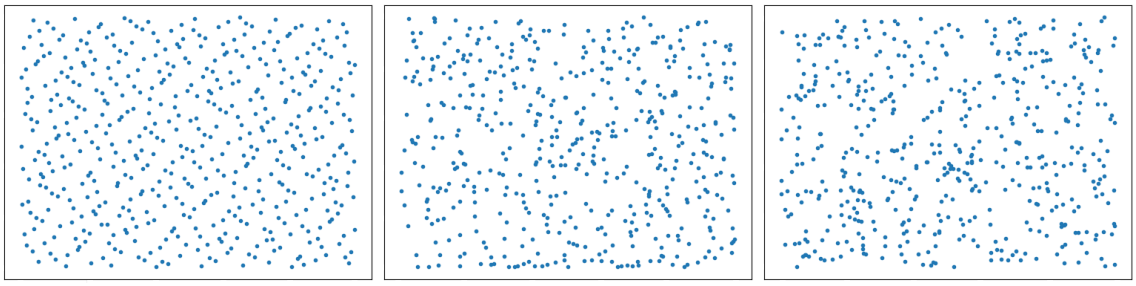

Figure 2.4: Distribution of the first 500 points in a 2-dimensional Halton sequence (left), LFSR (middle) and Mersenne Twister (right).

The advantage of deterministic sequences in SNGs is their regularity and the similarly regular patterns they cause in the generated SNs. It is clearly visible in figure 2.4 that the points in the low-discrepancy sequence are spread more uniformly over the area, whereas the pseudo randomly generated points cluster or leave larger open spaces between themselves. Points in the Halton sequence are also generated in a fixed, uniformly distributed order, which ensures high progressive precision of an SN (i.e. convergence towards its expected value), as its value does not fluctuate much. This leads to very small initial random

fluctuation errors during SN generation, or even none at all, if the length of the SN is chosen accordingly to the generator sequence. In addition, different generator sequences can be used in some cases to ensure exact operations, multiplication being the most prominent example. Several methods are known to achieve this. Using two Halton sequences with relatively prime bases, e.g. a base-2 Halton sequence H_2 and a base-3 Halton sequence H_3 is one possibility, as shown in table 2.2. If this method is used, then SN $A = 1$ if $H_2 \leq a$ and SN $B = 1$ if $H_3 < b$. Without this asymmetry, the result would be off by exactly one bit.

Table 2.2: Multiplication of values $a = \frac{1}{2}$ and $b = \frac{2}{3}$ using Halton sequences with relatively prime bases.

H_2	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{4}$	$\frac{1}{8}$	$\frac{5}{8}$	$\frac{3}{8}$	$\frac{7}{8}$	$\frac{1}{16}$	$\frac{9}{16}$	$\frac{5}{16}$	$\frac{13}{16}$	$\frac{3}{16}$	$\frac{11}{16}$	$\frac{7}{16}$	$\frac{15}{16}$	$p_{unipolar}$
H_3	$\frac{1}{3}$	$\frac{2}{3}$	$\frac{1}{9}$	$\frac{4}{9}$	$\frac{7}{9}$	$\frac{2}{9}$	$\frac{5}{9}$	$\frac{8}{9}$	$\frac{1}{27}$	$\frac{10}{27}$	$\frac{19}{27}$	$\frac{4}{27}$	$\frac{13}{27}$	$\frac{22}{27}$	$\frac{7}{27}$	
A	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	$\frac{8}{15}$
B	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	$\frac{10}{15}$
C	1	0	0	1	0	1	0	0	0	1	0	1	0	0	0	$\frac{5}{15}$

Table 2.2 shows clearly how the regular patterns in the generator sequence (see left part of figure 2.4) are transformed into similar patterns in the generated SNs. A has a regular *10* pattern with the exception of its first bit, which is due to $a = H_2$. Similarly, B has a *101* pattern. Relatively prime bases for the generator sequences cause the length of these patterns to be relatively prime in turn. As a consequence, the pattern of A is "shifted" over the pattern of B by one bit every occurrence, leading to an identical number of overlaps of every bit in A's pattern with every bit in B's pattern.

Deterministically generated SNs enable accurate combinational computations, but can interfere with sequential designs. Many of these designs depend on the statistical independence of bits, for example the concept of isolation and FSM-based SC components. This can limit the practical use of deterministic sequences in certain cases. For example, squaring the SN A from table 2.2 with an isolator-based multiplier (figure 2.3) leads to a highly inaccurate result as shown in table 2.3.

Accurate computation with deterministically generated SNs has strict requirements, which are violated by isolation. In this specific case, the SNs A and $\hat{A}$ have been generated with

Table 2.3: Interference of deterministically generated SNs with isolation-based squarer circuit.

A	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	$\frac{8}{15}$
$\hat{A}$	0	1	1	0	1	0	1	0	1	0	1	0	1	0	1	$\frac{8}{15}$
$A \cdot \hat{A}$	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	$\frac{1}{15}$

the same deterministic sequence H_2 and do therefore not consist of patterns with relatively prime lengths. It is clear that additional isolators cannot solve this issue, as the two SNs would either be identical (with an even number of isolators) or non-overlapping (with an odd number of isolators). Computing a^2 accurately in this case would therefore need two independent SNs generated by different Halton generators, increasing the SNG cost significantly. Other deterministic methods such as the clock division method [NJLR19] can reduce this overhead as there is no need for two separate sequence generators, however they are still significantly more expensive than isolation, which only requires a single flip-flop.

Further complications arise in FSM-based SC components when deterministic sequences are employed, including the stochastic hyperbolic tangent (stanh) (see figure 2.5) and its modification Btanh from [KKY+16] as the most prominently used examples. These components assume subsequent input bits or integers to be independent, as they compute their implemented arithmetic function essentially through a restricted one dimensional random walk. Their output values are therefore severely affected by dependencies between inputs, as has been shown for LFSR-based SN generation in [NPH17] and [NPH18a]. The effect that deterministically generated input SNs have on them can be even more severe, as a simple example with the SN B from example 2.2 and a 4-state stanh FSM in table 2.4 shows.

Table 2.4: Interference of deterministically generated SNs FSM-based SC components on the example of stanh (4 states) starting in state S_2.

SN B	1	0	1	1	0	1	1	0	1	1	0	1	*101* repeats
State	S_2	S_3	S_2	S_3	S_3	S_2	S_3	S_3	S_2	S_3	S_3	S_2	$S_3\ S_3\ S_2$ repeats
Output	1	1	1	1	1	1	1	1	1	1	1	1	1 repeats

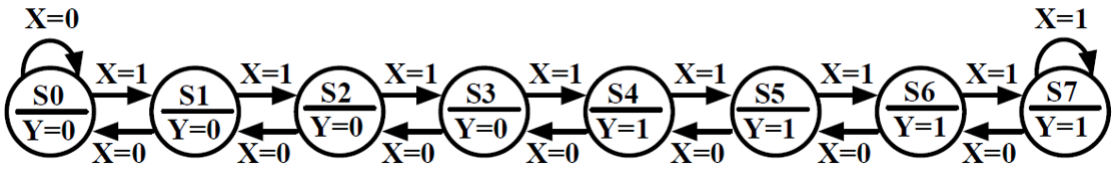

Figure 2.5: FSM for the computation of the stochastic hyperbolic tangent [BC01a]. X is a bit of the input SN, Y the output bit.

The correct (expected) value for the implemented function is $\tanh(\frac{2}{3}) = 0.583$, however the value computed by the FSM is 1. Due to the regular *101* bit pattern in B, the FSM never reaches states S_0 and S_1, and therefore never outputs any 0 bit. A different starting state (e.g. S_1) would not solve this issue, as it would only cause a change during the first three cycles, before the FSM starts to alternate between S_2 and S_3 again.

Besides the SNG implementations mentioned above, all of which are conventional CMOS-based components, there are proposed SNG designs based on emerging technologies, most prominently memristors [KLZ14] and spintronic devices [VVF+15]. There is a synergy and mutual benefit between these types of emerging devices on the one side and stochastic computing on the other. The devices present a cheap and efficient way of generating SNs due to their inherently probabilistic behaviour. Memristors are resistive devices whose resistive values can be switched between high resistance and low resistance states by applying a voltage pulse for a specific amount of time. This switching procedure is probabilistic, the shorter the voltage pulse is, the smaller the probability of the memristor changing its state. Experiments have shown that this switching probability follows a Poisson distribution. In theory, the probability of measuring a high or low resistance, i.e. a 0 or a 1, can therefore be controlled by the length of the applied voltage pulse, and an SNG could be implemented using only a single memristor. In practice however, the exact distributions of the switching probability differ slightly between devices due to manufacturing variations, and switching times cannot be controlled arbitrarily accurate either. SC provides an architecture in which these deficits are partially mitigated by the use of an error tolerant number format and inherently probabilistic operations, which are not expected to be accurate in general. An analysis of noisy emerging technologies in SNGs and techniques to reduce the influence of a device's manufacturing deficiencies can be

found in [YHFQ17]. In this work, SNGs based on emerging devices will not be considered further, as they are still not readily available and the vast majority of published works on SC uses conventional CMOS-based SNG designs.
SNG design is a central part of SC, and the choice of a specific design has a major impact on an SC system's viability. Some designs provide guarantees on the accuracy of select operations but interfere with common component designs and isolation, while other designs are universally usable, but lead to less accurate computations. Again others can be shared more easily among circuit inputs, but require more area for a single instance. An SNG of the latter type is covered in more detail in section 3.2 where the S-Box based Number Generator is presented.

2.2.4 Sources of inaccuracy and accuracy estimation

In general, SC produces inexact results due to the involvement of PRNGs and randomized SNs. Contrary to other forms of approximate computing however, inaccuracy in SC can vary each time a stochastic computation is performed, even if the input values are identical. In contrast, an approximate adder such as the design presented in [GMRR12] produces fixed errors for specific input combinations. On the other hand, the errors of two SC computations with identical input values are only the same, if identical PRNGs with identical starting states are used. For this reason, accuracy analysis in SC is done almost exclusively using methods from statistics and probability theory.
Errors in SC are commonly split up into three different sources [QLR+10] [AH13]:

Quantization errors are caused by insufficient SN length. For example, the value 0.41 needs an SN length of 100 in unipolar format or 200 in bipolar format (or integer multiples of these lengths) to be represented exactly. Any other SN length will lead to an approximated value.

Random fluctuation errors are caused by the randomized order of 1s in SNs. When two or more SNs are involved in an arithmetic operation, the output SN varies depending on the bit positions of the inputs and so does its value.

Correlation errors are caused by systematic non-0 cross-correlation of two or more SNs involved in a correlation sensitive stochastic circuit and by non-0 autocorrelation of SNs in sequential circuits.

An extreme example of correlation errors is figure 2.3a) where the inputs have an SCC of 1. It is obvious that any systematic non-0 correlation in this circuit will lead to an inaccurate result. An example for the other two error sources is given in figure 2.6.
In this work, correlation errors only encompass errors caused by systematic correlation. This systematic correlation is often unintentionally caused by design errors such as using the circuit from figure 2.3c) for implementing the function $f(x) = x^4$ or is an effect of SNG sharing as examined for example in [IIS+14]. Any circuit without systematic correlation (usually due to independently generated SNs) such as in figure 2.6 will almost always have inputs with non-0 correlation for any single computation, which however averages out to 0 over a large number of computations. Ultimately these temporary varying correlations are the cause of random fluctuation errors.

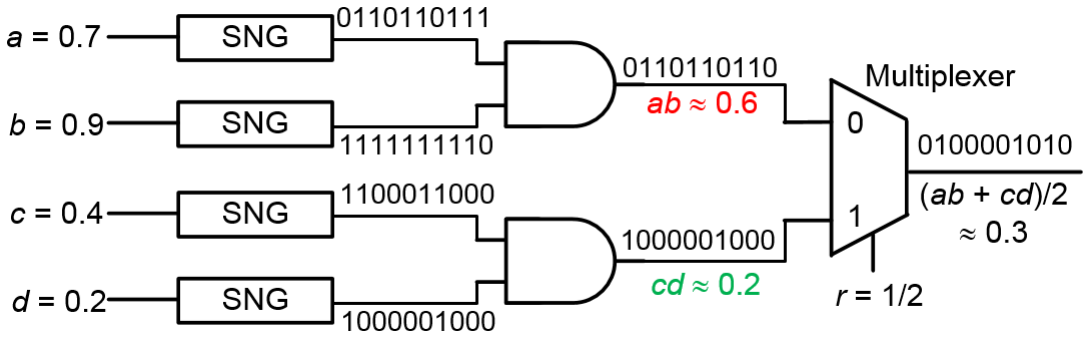

Figure 2.6: SC error sources in an exemplary computation. The value ab suffers from a quantization error, the value of cd from a random fluctuation error.

Recently, the use of previously mentioned low-discrepancy sequences in SNGs has been proposed to reduce the occurrence of random fluctuation errors in basic SC operations[NJLR19] [LH17]. However, it is challenging and potentially hardware intensive to eliminate these errors over several circuit levels in this manner, as sequences have to be combined carefully, simple decorrelation methods such as isolation cannot be applied, and generating multiple sequences requires increasingly larger circuits. Quantization errors on the other

hand are virtually unavoidable in SC. Even if the primary inputs of a stochastic circuit can all be represented exactly with the given SN length, any multiplication or scaled addition will most likely increase the required precision beyond this point. The only possible solution would be to determine the worst case required precision beforehand and design the circuit using the corresponding SN length. However, as SN length increases exponentially with precision, this would cause infeasibly long computation times. For example, input values with 8 bit binary precision require an SN length of $2^8 = 256$ bits to avoid quantization errors, but the result of a multiplication would require $2^{16} = 65536$ bits. It is therefore desirable to have estimates or even guarantees for the (average) accuracy of a given stochastic circuit, considering that an exact computation can almost never be guaranteed and errors are (pseudo) randomly distributed.

Basic observations regarding the variance of SNs were already included in Gaines' original work [Gai69]. He noted that the variance of a unipolar SN A with (expected) value a and length n is given by

$$\text{Var}(A_{up}) = \frac{a(1-a)}{n} \tag{2.9}$$

under the assumption that the individual bits of A are independent, identically distributed (iid) Bernoulli variables (i.e. generated using a TRNG). Correspondingly, the variance of A in bipolar format is

$$\text{Var}(A_{bp}) = \frac{1-a^2}{n}. \tag{2.10}$$

It was later shown in [Neu16] that the iid Bernoulli assumption enables much more detailed estimates of a combinational stochastic circuit's accuracy. Under this assumption a computation of a stochastic circuit using SNs of length n can be mathematically modelled as a direct simulation, a subtype of Monte-Carlo simulation. In the following, the definitions from [MGNR12] will be used:

Definition 2.2. Base experiment: A real-valued and integrable random variable Y with expected value $\mathbb{E}(Y) = f(x)$ for a target function $f(x)$ and a specific input value x is a base experiment for the computation of $f(x)$.

In the context of SC, running a stochastic circuit for a single clock cycle (i.e. an SN of length 1) computes a base experiment for the function f that the given circuit implements. A computation over n clock cycles forms a direct simulation:

Definition 2.3. Direct simulation: Let $Y_1, \cdots, Y_n$ be iid random variables with probability distribution $P_{Y_1} = P_Y$. The random variable

$$D_n = \frac{1}{n} \sum_{i=1}^{n} Y_i \tag{2.11}$$

is a direct simulation for the computation of $f(x)$ with n repetitions.

For example, the circuit from figure 2.6 can be expressed as a base experiment by modelling all data inputs and the MUX select input as (Bernoulli) random variables: $Y = rAB + (1-r)CD$. The direct simulation model provides a simple way to compute basic accuracy metrics such as mean square error (MSE) Δ^2 using the standard deviation of the base experiment (proof can be found in [MGNR12]):

$$\Delta^2(D_n) = \frac{1}{n} \cdot \sigma^2(Y) \tag{2.12}$$

It was further shown in [Neu16] that the variances of single SNs (according to equations 2.9 and 2.10) are identical for outputs of any given combinational stochastic circuit (replacing A with the output SN of the circuit), meaning that all functions that can be implemented as a combinational stochastic circuit are computed with the same accuracy. Furthermore, it was shown that the central limit theorem (CLT) applies to SC under the iid assumption, which results in a connection of confidence intervals and minimum required SN lengths n^*:

$$n^* \approx \left(\Phi^{-1}\left(1 - \frac{\gamma}{2}\right) \cdot \frac{\sigma(Y)}{\epsilon} \right)^2 \tag{2.13}$$

where $1-\gamma$ is the confidence level for a confidence interval of size 2ϵ, Φ is the cumulative normal distribution function and Y is the circuit's implemented function expressed as a base experiment. This function estimates the minimum length n^* that is needed such that a circuit's output value differs from the correct value by no more than ϵ with probability $1-\gamma$. Equation 2.13 does not factor in quantization error. However, as the quantization

error by itself can at most have a value of $\frac{1}{n}$ respectively $\frac{2}{n}$, it is usually dominated by the random fluctuation error.

The standard deviation of a circuit is key to the accuracy estimate but can be difficult to obtain in practice, as it depends on the type of SNG used. Modelling bits of SNs as iid Bernoulli variables can simplify this step greatly, but does generally not actually correspond to the properties of practical SNGs. It has recently been shown for example that LFSR-based SNGs are better modelled through hypergeometric distributions [BH20]. In most cases however, equation 2.13 can still serve as a lower bound on accuracy estimates, as LFSR-based SNGs can generate SNs with minimum error as previously shown and are therefore more accurate than the Bernoulli assumption suggests. The extension of the work from [Neu16] covers sequential stochastic circuits [NPH18b] and will be presented in chapter 5.

2.2.5 Convolutional neural networks and stochastic neural networks

Neural Networks are the state-of-the-art method for recognition, classification and prediction of various kinds of data, from images [HZRS16] over speech [PZJ+20] to biochemical data, e.g analysing and predicting protein folding structures [AlQ21]. Different types of NNs are employed depending on the task and the type of data, most prominently convolutional NNs (mostly used for image classification tasks) and recurrent NNs (predominantly used in speech recognition), although the latter is increasingly replaced by transformer networks. In the context of SC, CNNs are the most commonly considered NN type, with a few works targeting recurrent networks and only sporadic works focusing on other types such as binarized networks [HPB+19]. CNNs lend themselves to SC mostly due to their large potential for parallelism and their tolerance towards small output errors. The SC components presented in this work are primarily targeted towards CNNs as well, although some are also usable in other NN types. The background information given in this section focuses on CNNs, however, some concepts apply to other NN types as well.

The basic building block of any neural network is a neuron, which can have varying structures depending on the type of network it is used in. In CNNs, a neuron performs a basic multiply-accumulate (MAC) function in which the values of incoming connections, called *activations*, are multiplied with *weights* and summed together (fig. 2.7).

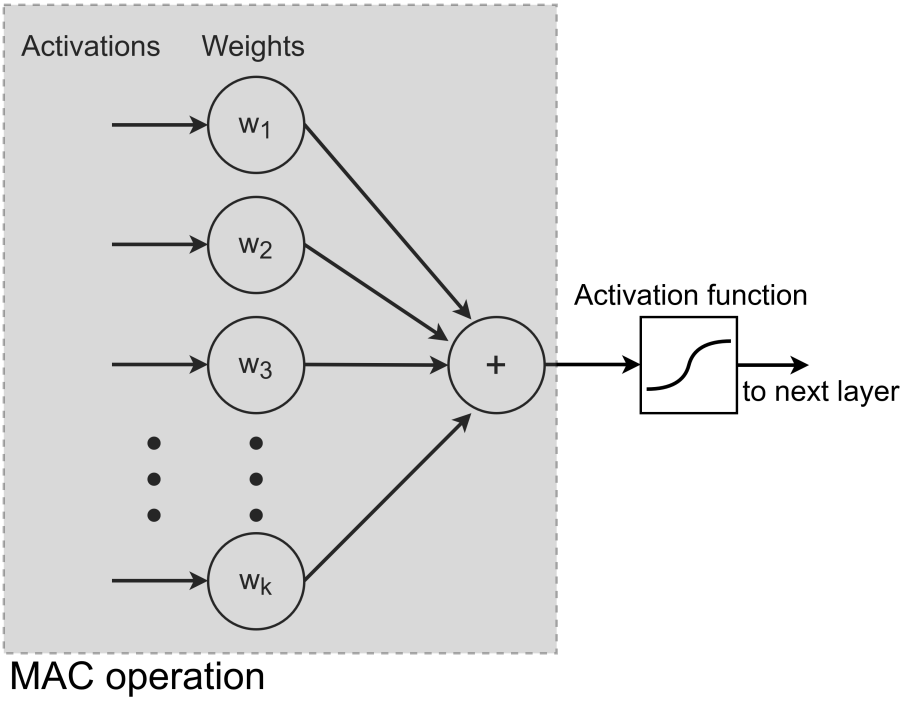

Figure 2.7: Basic schematic of a neuron in a CNN.

Depending on the input connections, a neuron can perform slightly different functions. In a CNN, convolutional and fully connected neurons are used. Inputs of convolutional neurons are a (usually relatively small) subset of the input space, which is selected by the convolution window. This window convolves over the entire input space to complete one full computation of a convolutional layer. These layers are used to extract certain features from the input data. In the context of image classification, these features can for example be colors and shapes of the input image in the first layer, and combinations of such simple features in deeper layers [ZF14]. The outputs of convolutional neurons are called feature maps and the set of weights assigned to a specific neuron is called filter or kernel.
A fully connected neuron can be seen as a special case of a convolutional neuron in which the convolutional window has the same size as the input data. These neurons usually form the final layers of a CNN to combine information from neurons in previous layers and determine the network's final decision.

The MAC operation of a neuron is commonly followed by an activation function. The most common activation functions in CNNs are the Rectified Linear Unit (ReLU) and sigmoidal

functions. Among the latter, the logistic function, which is often simply referred to as sigmoid function in the context of NNs, and the hyperbolic tangent are most relevant in the context of SC. The functions are given by:

$$\text{ReLU}(x) = \max(x, 0) \tag{2.14}$$

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{2.15}$$

and

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.16}$$

Activation functions introduce a non-linear component into the network. Because the neurons described above only consist of linear functions, activation layers are vital for an NN to model complex relationships in its inputs.

The third major component in CNNs is a pooling element that takes outputs from several neurons or activation functions and combines them into one output. Traditionally, the two most relevant pooling functions are average-pooling, where outputs of neurons are simply averaged, and max-pooling, where only the neuron output with the highest value is retained and other neuron's outputs are discarded. Max-pooling has been shown to lead to overall better performances in CNNs and is therefore the predominant pooling component. The purpose of the pooling layer is twofold. Its first goal is to reduce the amount of information present in the network. Convolutional layers generate huge amounts of data in order to recognize and extract various features from their inputs. Pooling layers are necessary to avoid an explosive increase in data that several consecutive convolutional layers can cause. Their second goal is to filter for important data and discard redundant and less relevant data. A commonly used pooling layer in image classification networks is 2×2-sized non-overlapping max-pooling.

These four types of layer can in theory be arranged freely. However, in practice usually several blocks consisting of one or more pairs of convolutional and activation layers followed by a pooling layer are connected in sequence, followed by one or more fully

connected layers that produce the network's final output. Large NNs for very difficult tasks such as ResNet [HZRS16] often have different configurations. ResNet for example includes short cut connections that skip several layers. Such large networks are however not common targets of SC implementations, as they are too large and resource intensive to be considered for edge and near-sensor computing.

In all networks of the described type, the vast majority of operations is concentrated in convolutional and fully connected layers. They consist of a large number of MAC operations that are individually very simple, but nevertheless require lots of computational power due to their sheer number. For example, a small CNN for MNIST classification, one of the simplest tasks in image classification, with 20 5×5 filters in its first convolutional layer requires $288,000$ multiplications and additions each for processing a single input image. These operations can be parallelized to an almost arbitrary degree. All filters are independent of each other and even within a single filter, computations of all pixels in a feature map are independent. A dedicated hardware implementation for CNNs such as Google's Tensor Processing Unit can take advantage of this parallelism and provide a large number of hardware neurons to speed up computation of a layer significantly. With conventional multipliers, this entails a significant cost in area and power consumption. With SC on the other hand, this cost is reduced drastically, as a multiplier only consists of a single gate. Moreover, independence of operations means that cross-correlation between SNs is not an issue, as long as they are not directly multiplied with or added to each other. SNGs can thus be easily shared between different neurons without worrying about correlation errors. This potential for highly parallelized computations in combination with reduced overhead from SNGs, combined with the inherent tolerance of CNNs for approximate computations make them a primary target application for SC.

In order to obtain a set of weights that leads to high classification accuracy, an NN is trained on known data. Back-propagation is the commonly used training mechanism for NNs and can be formally described as follows:
An NN with output O is a composition of functions with the network's weights as parameters:

$$O(x) = f^N(W^N f^{N-1}(W^{N-1}...f^1(W^1 x)...)) \tag{2.17}$$

In this equation, x is the network's input (e.g. an image as a matrix of pixel values), N is the number of layers and f^i is the activation function of layer i. $W^i = (w^i_{jk})$ is the weight matrix of layer i, whereby a weight w_{jk} connects an output node k of layer $i-1$, e.g. a value in a feature map, to a neuron j in layer i. For example, w^1_{12} is the weight that connects the node 2 of layer 0 with the first neuron in layer 1. In context of an image classification network, it can be viewed as the weight of the connection from the second input pixel to the neuron that produces the first feature map in layer 1.
During back-propagation, an NN computes its output $O(x_m)$ for an input x_m with a previously known class y_m. O is a vector of output values, one per class (typically but not necessarily probabilities). With the network's computed output and the known output y, a loss function L, also called cost function is computed. A typical choice is the standard Euclidean distance:

$$L(y_m, O(x_m)) = \frac{1}{2}\|y_m - O(x_m)\|^2 \tag{2.18}$$

Usually, several training pairs (x_m, y_m) are processed before the loss function is calculated in order to reduce the training effort. Such a set of b training pairs is called a batch and the loss function is computed for the entire batch B:

$$L(B) = \frac{1}{2b}\sum_{i=1}^{b}\|y_i - O(x_i)\|^2 \tag{2.19}$$

Loosely speaking, the loss function is a measure of the difference between the network's prediction and the correct classification. The higher the value of L, the less accurate the networks is. The goal of the training algorithm therefore is to minimize the loss function by adjusting the network's weight parameters in small steps. After processing a batch, the algorithm determines the contribution of each weight w_{jk} to the loss $L(B)$ by computing the partial derivative $\frac{\partial L}{\partial w_{jk}}$ starting from the output layer and moving backwards through the network to the input layer. Each weight is then adjusted by an amount Δw_{jk}, usually multiplied with some learning rate $\lambda > 0$ such that the weight's contribution is reduced:

$$\Delta w_{jk} = -\lambda\frac{\partial L}{\partial w_{jk}} \tag{2.20}$$

The learning rate is a hyper-parameter that does not affect the network structure itself, but it's performance. A higher learning rate can lead to faster training, but also oscillation around the minimum of L due to overshooting in the weight adjustments. On the other hand, a smaller learning rate can hit the minimum more accurately, but can lead to the algorithm getting stuck in a local minimum rather than finding the global minimum.
The training of an SCNN is commonly performed in software on an equivalent fully binary model. A few exceptions to this procedure exist, such as the SCNN implemented by Brown and Card [BC01a], which was small enough to enable network training in the stochastic domain. In [LWLH18], an SCNN with pre-trained weights that are refined by online learning in SC is presented. Most of the time however, weights of a fully trained binary NN are transferred to the SC model. SNs are not suitable to accurately represent the very small weight adjustments Δw_{jk} and a hardware implementation of the training algorithm including computation of derivatives of all weights is infeasible. The training procedure further needs to be slightly adjusted for SCNNs, as the weights of all layers implemented in SC have to lie within the range $[-1, 1]$. This is achieved by a second weight adjustment step after each batch. All weights in the affected layers are scaled by the maximum weight value after the correction by Δw:

$$w_{jk} = \frac{w_{jk} + \Delta w_{jk}}{\max(w_{jk} + \Delta w_{jk})} \tag{2.21}$$

The scaling ensures that the maximum absolute weight value in the SC layers is 1, thereby using as much of the available range as possible. At the same time, the ratio of weights within a layer and thus the relative influence of connections on the network's output is preserved.

The first major work on SC-based NNs (SCNNs) was published in 2001 [BC01a] [BC01b] and introduced SC elements that are still in use, most importantly the FSM-based implementations of activation functions often used in NNs such as the stochastic hyperbolic tangent (stanh, figure 2.5). The target NN of this work was a (for today's standards) relatively small network encompassing only 184 weighted connections in total and was used for recognition of noisy characters from the E-13B MICR (Magnetic Ink Character Recognition) font.

A large number of works on SCNNs was published starting around the year 2015, owing to the increased prevalence of embedded and distributed devices employing NNs in resource constrained environments. The main developments in many of these more recent works compared to Brown and Card's early work was the change in implementation of addition and the drastic increase in size of target NNs. While [BC01a] uses basic scaled addition via MUX, more recent works have replaced this with unscaled SC-binary hybrid adders (see 2.2.1), for example in the form of parallel counters in [KKY+16], or in [LRL+17] in the form of approximated versions of accumulative parallel counters (APCs) from [PY95]. Even relatively small CNNs frequently include additions with tens to several hundreds of summands that would require extremely long SNs to compute with reasonable accuracy via MUX due to the equally large downscaling factors. As these hybrid adders move the computation from the stochastic into the binary domain, they are often combined with a corresponding binary-SC hybrid activation function that returns the computation back into the stochastic domain. An example for such a component is Btanh [KKY+16], a modification of stanh. The tanh function's output is bounded by $[-1, 1]$ and therefore inherently converts all of its input values back to the standard range for bipolar SNs. SCNNs and SC-based NN accelerators have shown good performance for various image recognitions tasks, from simple greyscale datasets like MNIST [LAH+17] and fashion-MNIST [HPB+19] to the state-of-the-art dataset ImageNet [HGT+19] [LLR+18].

Conventional NNs have shifted towards using the Rectified Linear Unit (ReLU) as their activation function more often, as it simplifies and speeds up the training process. ReLU is defined as $\text{ReLU}(x) = \max(0, x)$ and is therefore identical to the max-pooling layer in terms of circuit implementation. In previous works, an accurate SC implementation of the maximum function that does not incur a large delay was considered impossible [RLD+17]. It was thought that a set of SNs would have to be processed completely first to compute their value in order to determine their maximum and only afterwards could the SN with the maximum value be processed further. Both max-pooling and ReLU operations were therefore only approximated [RLD+17] [LLR+18]. However, an accurate SC implementation of the maximum function with no delay is indeed possible as presented in section 3.1.

A prevalent design scheme for convolutional SC neurons and SCNN layers can be seen in most of the recent works. An SC neuron commonly consists of a column of parallel SC multipliers, followed by an SC-binary hybrid adder of varying design, and a binary-SC activation function, either implementing tanh or ReLU. Many times, four such neurons are put in parallel, followed by an SC max-pooling component. Such MAC-activation-pooling elements form the basic building blocks of SCNNs, a schematic is shown in figure 2.8.

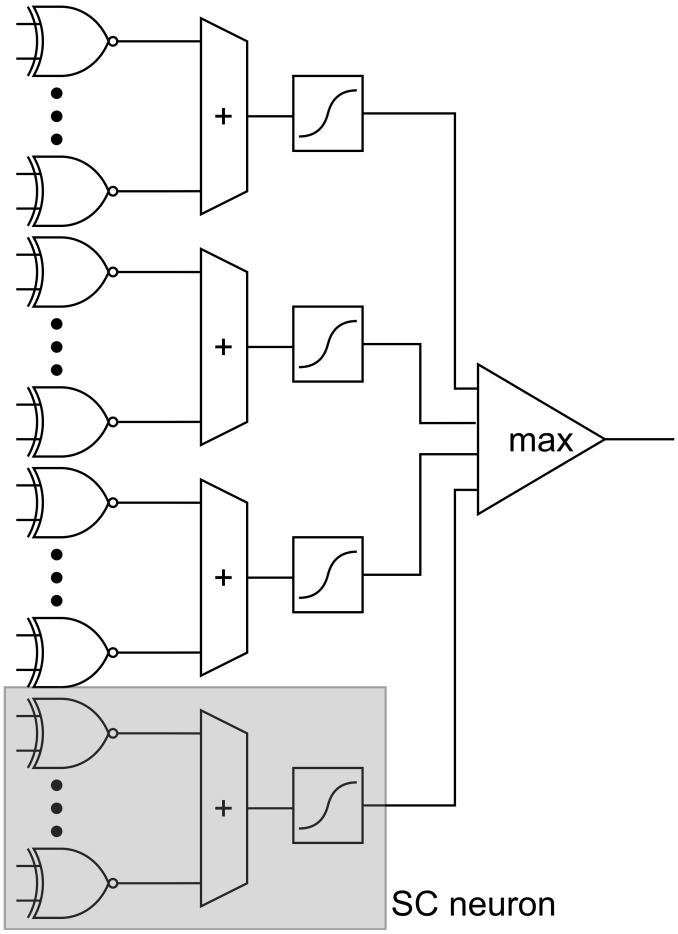

Figure 2.8: Schematic of a bipolar SC MAC-activation-pooling element consisting of four SC neurons.

In fully connected layers, SC neurons consist of only one MAC and activation function component, as there are no pooling layers after a fully connected layer. One SC neuron instance in hardware is in theory sufficient to compute a full convolutional layer, as all neurons within a layer have the same structure. One instance can thus be used several

times with changing weight inputs to compute all kernels of a convolutional layer. In practice, multiple neurons are usually employed in parallel to speed up computation. Inputs and outputs of the depicted SC neuron 2.8 are SNs, making it possible to implement several subsequent SC layers without additional intermediate components.

Lately, recurrent NNs (RNNs) have also been implemented in SC [MZWH19] [LLLH19]. These implementations make use of the same components as SC versions of CNNs, e.g. stanh. While RNNs are generally smaller than CNNs, SC versions of RNNs are faced with the additional challenge of feedback loops. In RNNs, outputs of neurons are used as inputs of the same neuron in subsequent computations. SC RNNs therefore also have to take care of potentially self perpetuating, increasing errors. In addition, RNN neurons have internal cell states that need to be stored and updated regularly. This also poses a problem for SC, as storing intermediate values interferes with its stream-based computing paradigm. These values have to be converted from stochastic to binary domain in order to avoid exponential memory requirements of SNs. Values are then updated and reconverted to SNs for the next computation phase. This procedure entails a computational delay equal to the SN length due to the conversions.

SC RNNs have so far only been shown to work for relatively simple tasks in symbol generation and basic speech recognition. Under high levels of noise however, SC RNNs can even outperform their binary counterparts, as shown in [LLLH19]. The main body of the work on SCNNs is concentrated on covolutional networks, specifically image classification. This work therefore focuses on convolutional networks for examples and explanations. However, many of the presented concepts are also applicable in RNNs.

Part I

Components and Error Resilience

Chapter 3

SCNN Components

3.1 NMax: An accurate stochastic maximum function

3.1.1 The maximum function in SC

With increasingly complex convolutional NNs becoming one of the main focus points of SC applications, the maximum function equally gained in importance, as it is used for sub-sampling (max-pooling) and activation (ReLU) functions. While early CNNs such as the well-known LeNet5 [LBBH98] used general weighted additions for sub-sampling, CNNs nowadays implement sub-sampling through max-pooling layers most of the time (see also section 2.2.5). These layers are placed between convolutional layers to filter unimportant data, avoid overfitting and reduce the overall amount of data in the network. Average pooling can still be found in some CNNs as well, but is generally considered to perform worse overall, although it may produce better results for particular datasets. In general though, max-pooling tends to preserve important features for classification, such as edges in an image, whereas average pooling tends to blur information from multiple datapoints together. State-of-the-art CNNs such as AlexNet [KSH12] and ResNet [HZRS16] therefore employ max-pooling layers.
The second main task of the maximum function is the rectified linear unit (ReLU) activation function, which is defined as

$$\text{ReLU}(x) = \max\{x, 0\}. \tag{3.1}$$

Due to its easily computable gradient, ReLU speeds up the training process significantly without sacrificing classification accuracy. It is therefore the preferred activation function in most large scale CNNs. An SC implementation of the maximum function is therefore essential for accurate and well-trainable SCNNs.

The simplest possible implementation of an SC maximum function is an OR gate with maximally correlated inputs. For two input SNs X_1 and X_2, the result of $X_1 \vee X_2$ will then be equal to the SN with more 1s. However, maximum correlation is in most cases too big of a requirement for this theoretically small implementation to be useful. If the maximum function is not computed on primary inputs, but follows other stochastic components, its inputs are generally not correlated in a specific way. This is the case in SCNNs, where the max-pooling layer follows a convolutional and an activation layer. Introducing correlation at this point requires a process called *regeneration*, which is a conversion from SNs to binary numbers and back to SNs. During the second part of regeneration, correlation can be introduced by sharing RNGs. As previously described though, SNGs are very expensive components, making the OR gate solution to the maximum function infeasible in most cases.

The maximum function for arbitrarily correlated inputs poses a challenge for SC, as it requires knowledge of the overall value of an SN, which is generally only known after processing the whole SN. In contrast to basic functions such as multiplication, it cannot be implemented by processing bits in each clock cycle independently, as each bit by itself carries almost no information about an SN's value. Previously, SC implementations of the maximum function were approximative. Implementations in [YKLC17] (proposed for use in SCNNs) and [LL11] (proposed for use in digital image processing, specifically median filters) were based on the stanh function in combination with multiplexers, where the tanh function was used to track the difference of both input values. As stanh produces its highest inaccuracies for an input value of 0, the accuracy of these maximum circuits is lowest when input values are close to each other. Both circuits were initially designed with two inputs, but can be readily arranged in a tree structure to compute the maximum of more inputs.

The hardware-oriented max-pooling circuit proposed in [RLD+17] shown in figure 3.1 exploits the property of progressive precision in SNs. Counting the number of 1s in a

c-bit block of each input SN gives an approximation of the overall value of each SN. The maximum of these block values therefore is expected to also indicate the SN with maximum overall value, which is used to route this SN through a MUX to the circuit's output for the next c clock cycles. In other words, the circuit updates its routing every c clock cycles.

This approximation works well if the difference between the maximum and the second largest input value is large, as differences in individual block values are also large in that case. If this is not the case however, small fluctuations in the c-bit blocks can cause the MUX to select an input other than the maximum SN. Furthermore, the first c-bit block has to be chosen randomly or according to a fixed choice (e.g. always choosing the top MUX input), otherwise the circuit would cause a delay of c clock cycles. For these reasons, the expected output value of the hardware-oriented max-pooling circuit is slightly smaller than the exact value. This circuit will be also referred to as approximate maximum (AMax) in the following.

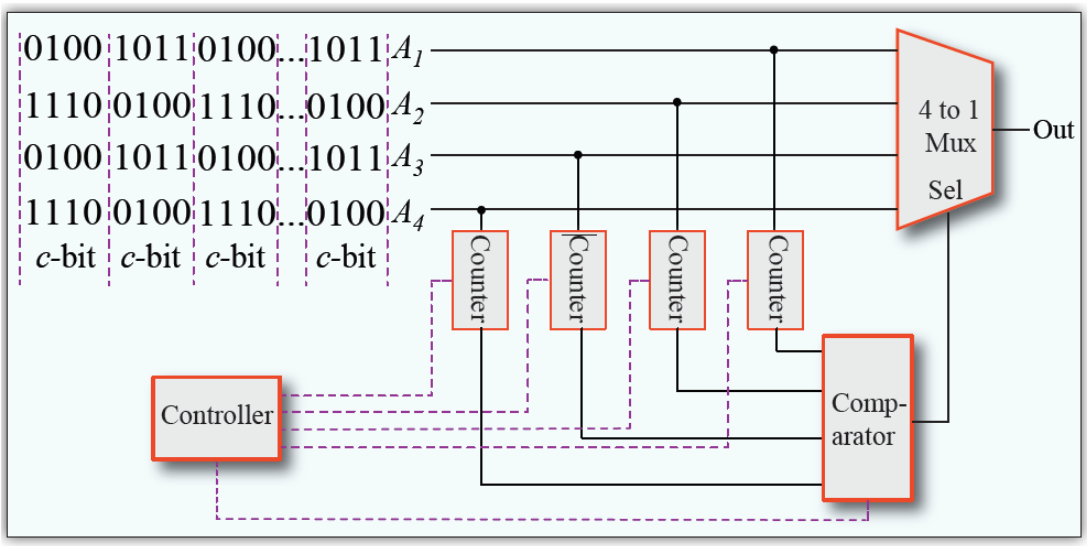

Figure 3.1: Hardware-oriented max-pooling circuit from [RLD+17].

SC implementation of ReLU in [LLR+18] is performed using an FSM similar to Btanh with a saturating up/down counter. This SC-based ReLU was designed to interface directly with a parallel counter at its input. Due to its (in theory) unlimited input value range but an

output value range that is limited to $[-1, 1]$, it cannot implement ReLU in its basic form, but rather computes a clipped version:

$$\text{ReLU}_{clipped} = \min\{\max\{0, x\}, 1\} \tag{3.2}$$

Simulations in [LLR+18] show that the circuit is mostly accurate for inputs $x \in [-5, 5]$ with $|x| > 2$, but less so in the center of the range, where most values in a CNN commonly concentrate (cf. section 6.2). Furthermore, as previously mentioned, FSM-based stochastic circuits are susceptible to cross-correlation and autocorrelation of its inputs, which appear often in SCNNs due to their massive number of inputs that share SNGs in order to reduce SNG area. An SC maximum function circuit should therefore be accurate over the whole value range and correlation insensitive, to both cross- and autocorrelation.

3.1.2 NMax design details

The SC maximum circuit "NMax" posessing these properties has been presented in [NPH19]. NMax is a counter-based circuit with a variable number of inputs. The circuit diagram for NMax with three inputs is shown in figure 3.2, further inputs can be added with identical blocks of counters with increment (Inc)/decrement (Dec) logic and cross-connections to the OR gates of other counter blocks and the final OR gate before the output. It is noted in [RLD+17] that a stochastic circuit that accurately determines a maximum input by comparing the values of its input SNs incurs an unacceptable delay, as the value of an SN can only be accurately determined after looking at all of its bits. A stochastic circuit operating according to this principle would therefore have to process all n bits of its input first before outputting another n bits, causing an n-cycle delay in the system. NMax solves this problem by comparing the differences between inputs at every clock cycle instead of differences in overall values. Each input SN $X_i \in \{X_1, X_2, \cdots, X_m\}$ has an associated counter $C_1, C_2, \cdots, C_m$ which tracks the value

$$c_i = \max\{C_1^l, C_2^l, \cdots, C_m^l\} - C_i^l \tag{3.3}$$

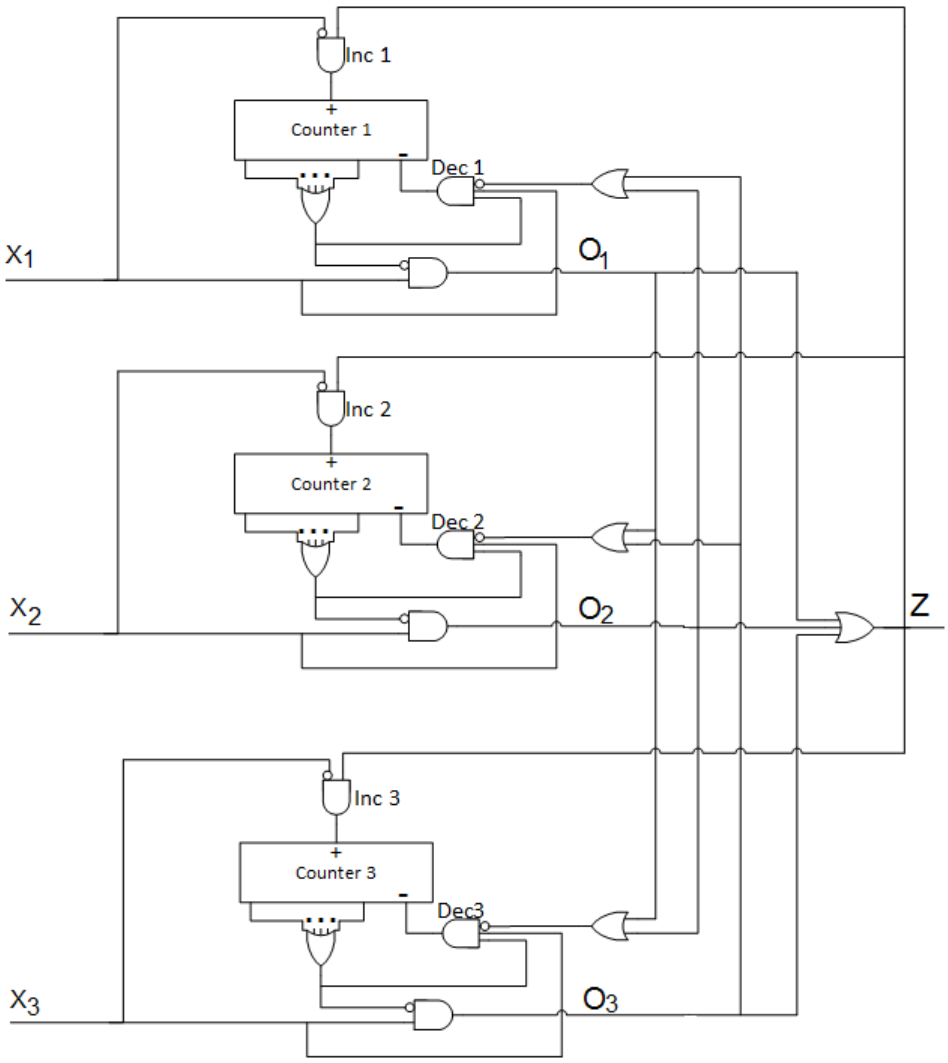

Figure 3.2: Stochastic maximum circuit NMax with three inputs from [NPH19].

where C_j^l is the number of 1s in SN X_j until clock cycle l, i.e. the unipolar value of SN X_j multiplied by l. NMax outputs a 1 when any SN X_j with $C_j^l = 0$ and $X_j^l = 1$ exists and a 0 otherwise. The algorithm is formally described below.

The values c_i can never be negative. It is also certain that at least one c_i equals 0, belonging to the counter(s) that are assigned to those SNs that contained the most 1s up to the current clock cycle l. The condition $c_i = 0$ for a 1 at the output of block i ensures that only SNs with the current maximum value can contribute a 1 to the circuit's output. Blocks with $c_i \neq 0$ will instead decrease their counter values and are irrelevant for the circuit's overall output. Therefore, NMax's output value only increases when the maximum input value

Set all $c_j = 0$;
while $l \leq n$ **do**
  for *each input SN* X_j **do**
    if $c_j^l = 0$ **then**
      Set $\text{Cout}_j = 0$;
    else
      Set $\text{Cout}_j = 1$;
    end
  end
  Set $O_j = X_j^l \wedge \overline{\text{Cout}_j}$;
  Set $Z = \bigvee_{i=1}^{m} O_i$;
  for *each input SN* X_j *in clock cycle* l **do**
    Set $\text{Inc}_j = \overline{X_j^l} \wedge Z$;
    Set $\text{Dec}_j = \text{Cout}_j \wedge X_j^l \wedge (\overline{\bigvee_{i \neq j} O_i})$;
  end
end

does so as well. This property holds in every clock cycle, as formally proven in appendix A.1. Table 3.1 shows an example of an NMax computation over eight clock cycles.

Table 3.1: Example of an NMax computation over 8 clock cycles with bipolar input SNs $X_1 = 0.5$, $X_2 = 0$, $X_3 = -0.5$ and $X_4 = -0.5$.

Clock cycle:	0	1	2	3	4	5	6	7
X_1	0	1	0	1	1	1	1	1
X_2	0	1	0	0	1	1	1	0
X_3	1	0	1	0	0	0	0	0
X_4	0	0	0	0	0	0	1	1
c_1	0	1	0	1	0	0	0	0
c_2	0	1	0	1	1	1	1	1
c_3	0	0	0	0	0	1	2	3
c_4	0	1	1	2	2	3	4	4
Z	1	0	1	0	1	1	1	1

It can be seen in table 3.1 that the output value is equal to the maximum input value, but the distribution of 1s is not identical, as it is determined by X_3 in clock cycles 0 to 3, and X_1 afterwards. Moreover, the counter values c_3 and c_4 are different in clock cycle 7, even though the values of X_3 and X_4 are identical. This is due to the table showing the counter

values at the beginning of each clock cycle (at the beginning of cycle 8, c_3 and c_4 would be equal).

3.1.3 Analysis and evaluation

NMax was primarily designed to be used in SCNNs. The first simulation set-up therefore evaluates the performance of NMax in a max-pooling layer of a small SCNN (configuration shown in table 3.2). Table 3.3a shows that the SCNN with NMax as its max-pooling component performs consistently better than the version using AMax. In addition, the accuracy of the AMax-based SCNNs vary slightly with block size c, which can only be determined through trial and error and is dependent on the SN length.

Table 3.2: Network structure used in simulations for NMax evaluation.

Layer ID	Layer type	Kernel parameters	Activation function
1	2D Convolution	$4 \times 4 \times 20$	tanh
2	Max-pooling	2×2	-
3	2D Convolution	$3 \times 3 \times 60$	ReLU
4	2D Convolution	$3 \times 3 \times 60$	ReLU
5	Max-pooling	2×2	-
7	2D Convolution	$3 \times 3 \times 120$	ReLU
8	Fully connected	100	-
9	Fully connected	10	softmax

As previously mentioned, NMax works regardless of any correlation within or between its inputs. Formally, this is also shown by the proof in the appendix, which does not assume any presence or absence of correlation. However, correlation within NMax's output and between the output and its inputs has to be investigated as well. As established in the circuit design and visible in table 3.1, NMax output tracks the maximum input SN bit for bit. If several input values have similar values, the tracking can temporarily switch between them. It is therefore expected that the cross-correlation between the output Z and these inputs is high, while cross-correlation with other inputs is similar to the cross-correlation between inputs. Autocorrelation is also expected to be similar to the inputs due to the bit-for-bit tracking. The correlation properties of NMax have been examined in the second simulation set-up: Inputs are generated independently of each other and have no

deliberate autocorrelation. SN length has been set to $n = 1024$. As both correlation metrics are independent of SN length, this choice does not limit the validity of the simulation.

Table 3.3: NMax simulations for NN accuracy and correlation.

(a) Accuracy of an SCNN classifying fashion-MNIST images with NMax and AMax based max-pooling layers.

Component:	NMax	AMax $c = 32$	AMax $c = 64$	AMax $c = 128$	AMax $c = 256$
NN Accuracy:	91.3%	90.7%	90.7%	90.5%	90.4%

(b) Correlation analysis of 4-input NMax. Values of the simulated input sets: Randomized Set 1 = $(0.80, 0.54, 0.14, -0.22)$, Randomized Set 2 = $(0.55, -0.42, 0.04, -0.88)$, Fixed Set 1 = $(0, 0, 0, 0)$, Fixed Set 2 = $(0.75, 0.75, 0.23, -0.52)$, Fixed Set 3 = $(0.80, 0.78, -0.20, -0.67)$.

Input values	SCC(X_1, Z)	SCC(X_2, Z)	SCC(X_3, Z)	SCC(X_4, Z)	$AC_1(Z)$
Randomized Set 1	0.994	0.007	0.006	0.021	0.002
Randomized Set 2	0.957	0.015	-0.017	0.007	-0.003
Fixed Set 1	0.252	0.249	0.249	0.248	-0.020
Fixed Set 2	0.496	0.500	-0.012	0.028	0.009
Fixed Set 3	0.765	0.221	-0.001	0.033	-0.006

Table 3.3b confirms the expectations regarding NMax's influence on correlation. For ease of writing, the inputs have been reordered such that X_1 always has the maximum input of an input set, in the following X_1 therefore refers to the SN with maximum value. If the value of X_1 is significantly distinct from the other input values (Random Sets 1+2), SCC(X_1, Z) is almost 1, as NMax outputs X_1 with almost no changes, which almost always appear in the first few output bits. Furthermore, the SCC between Z and the remaining input SNs is close to 0 and independent of their values, which is also expected when inputs are generated independently. Fixed Set 1 shows that NMax does not prioritize any of its inputs when input values are identical. The SCC values of this set show that NMax tracked almost exactly 25% of each input to generate Z. Similarly, almost exactly 50% of X_1 and X_2 have contributed to Z in fixed set 2, while the remaining two inputs are virtually uncorrelated to Z. Fixed set 3 shows that even a small difference between X_1 and any other value (approximately 20 out of 1024 bits in this example) leads to significantly different SCC values, showing that NMax converges fast towards the correct output value, i.e has good progressive precision.

Autocorrelation of distance 1 (see equation 2.7) is close to 0 in all cases, which is ex-

pected, as input SNs were generated without autocorrelation in this simulation and NMax does not reorder input bits in a predetermined way. The most important conclusion from this result is that NMax is compatible with decorrelation by isolation and can be used in sequence with FSM-based SC components such as stanh, which are commonly susceptible to autocorrelated inputs.

In section 6.1 it will be shown that some functions cannot have exact corresponding SC implementations but instead exhibit a bias in SC. The maximum function belongs to this group of functions and under specific circumstances it is possible to compute its bias using equation 6.6. As NMax is a non-approximate SC maximum implementation, it therefore also exhibits this bias, while approximate implementations such as in [RLD+17] and [YKLC17] might not be subject to it. As a detailed theoretical evaluation, e.g. in the form of a Markov chain, of these circuits is infeasible due to their complexity, a simulation-based approach was used to evaluate these circuits' biases.

Two different input situations are considered:

1. Input values are iid. This situation can be found for example in SCNNs for image classification when input images contain areas with consistent colour (e.g. a black background).

2. Input values are non-iid. This situation can be found for example in sorting networks.

In theory, iid inputs should lead to a larger bias than non-iid inputs in most cases. To illustrate this, take a case with n normal distributed, real-valued iid input values $X_1, ..., X_n$ with expected value μ and standard deviation σ. The probability $P_{>\mu}$ that at least one of the input values is larger than μ is

$$\begin{aligned} P_{>\mu} &= P\left(\{X_1 > \mu\} \vee \{X_2 > \mu\} \vee ... \vee \{X_n > \mu\}\right) \\ &= 1 - P\left(\{X_1 < \mu\} \wedge \{X_2 < \mu\} \wedge ... \wedge \{X_n < \mu\}\right) \\ &= 1 - P\left(\{X_1 < \mu\})\right) \cdot P\left(\{X_2 < \mu\})\right) \cdot ... \cdot P\left(\{X_n < \mu\})\right) \\ &= 1 - \frac{1}{2^n} = \frac{2^n - 1}{2^n}. \end{aligned} \tag{3.4}$$

On the other hand, take the case of n normal distributed non-iid input values $X_1, ..., X_n$ with different expected values and standard deviations $\mu_1, ..., \mu_n$ and $\sigma_1, ..., \sigma_n$. Assume without loss of generality that $\mu_1 > \mu_2, ..., \mu_n$. Then, $P_{>\mu_1}$ can again be expressed as

$$\begin{aligned} P_{>\mu_1} &= P\left(\{X_1 > \mu_1\} \vee \{X_2 > \mu_1\} \vee ... \vee \{X_n > \mu_1\}\right) \\ &= 1 - P\left(\{X_1 < \mu_1\} \wedge \{X_2 < \mu_1\} \wedge ... \wedge \{X_n < \mu_1\}\right). \end{aligned} \tag{3.5}$$

Because the normal distribution is symmetric around its mean, and $\mu_1 > \mu_2, ..., \mu_n$, it follows that

$$P\left(\{X_2 > \mu_1\}\right) < P\left(\{X_2 > \mu_2\}\right) \tag{3.6}$$

and correspondingly for all other X. Therefore, it follows that

$$P_{>\mu_1} \leq 1 - P\left(\{X_1 < \mu_1\} \wedge \{X_2 < \mu_2\} \wedge ... \wedge \{X_n < \mu_n\}\right) = P_{>\mu}. \tag{3.7}$$

This does of course not prove that the bias in the iid case is always larger than in a non-iid case. A simple counter example is an iid case with $\mu = 1$, which equates to a variance and therefore bias of 0 in SC. However, it provides a reasoning why the bias in an iid case is oftentimes larger when the input values are not very close to the boundaries of SN value range.

A simulation with randomized values supports the theoretical reasoning and provides a comparison to the bias of other stochastic maximum circuits previously mentioned. Table 3.4 shows biases of NMax, AMax (HOMC) [RLD+17] and the stanh-based maximum circuit by Yu et al. [YKLC17] using randomly generated input values. The results reported in the table are obtained from $10,000$ independent simulations with SNs of length $1,024$ bit. When inputs have identical values, NMax shows a bias that gets bigger as the values get closer to 0, as indicated by the theoretical analysis above. For an input value of 0, the measured bias of NMax matches the theoretical prediction from equation 6.5 up to three decimal places. AMax deals well with identical inputs, as the circuit essentially computes the average of its inputs under these circumstances. On the other hand, in the more general situation of randomized (i.e. non-identical input values), NMax achieves its goal of computing an exact maximum, while AMax suffers from a significant negative bias,

Table 3.4: Bias analysis of three stochastic maximum circuits.

Input values	NMax	AMax with block size				Yu et al.
		32	64	128	256	
-0.67	0.023	0	0	0	0	-0.003
-0.56	0.027	0	0	0	0	-0.002
0	0.032	0	0	0	0	-0.004
0.58	0.026	0	0	0	0	0
0.80	0.019	0	0	0	0	-0.003
randomized	0	-0.036	-0.072	-0.144	-0.291	-0.003
randomized	0	-0.021	-0.033	-0.065	-0.130	-0.072

which increases with the chosen block size. The first output block in AMax has to be chosen without having any information about input values to avoid introducing a delay and does therefore not correspond to the maximum in most cases. As the block size increases, the impact of the first block on the final output value increases accordingly. Yu et al.'s circuit lies between NMax and AMax regarding its bias, but its major disadvantage is that it is based on an stanh component, which is highly correlation sensitive. AMax can in theory also be affected by correlation between blocks, which is however less likely in practice, as it would require very specific and thus improbable input patterns. It follows that NMax provides the highest accuracy of available stochastic maximum circuits in the general case, only suffering from a small bias under very specific circumstances, and is also the only circuit that cannot be affected by input correlation. It therefore provides the most reliable and accurate implementation of max-pooling and ReLU functions in SCNNs.

3.2 SBoNG: An S-box based number generator

3.2.1 Challenges of stochastic number generation

SNGs are generally the most expensive component of a stochastic circuit. Not only do they have to include a register to store the (pseudo) random numbers that the SNG needs, they also include a circuit part to compare and transform their binary form inputs into a probability signal, the so-called probability conversion circuit (PCC). The conceptually simplest PCC is a comparator, which is often used as the default PCC in SC, but there are alternatives such as the MUX-chain [BC01a] and weighted binary generator (WBG)

[GK88] that can have advantages over the comparator, such as smaller area overhead. Lowering the cost of SNGs is an important issue in SC, as the overhead from number generation affects all SC systems regardless of application. Two approaches can be made to achieve this goal: Either the cost of the SNG parts, i.e. the size of the RNG and/or the size of the PCC can be reduced, or parts are shared between several SNGs.

The first option can only provide very limited improvements. The authors of [YLL+18] make the conjecture that the minimal cost of the non-shareable part of a PCC that converts k-bit binary inputs is $2k - 1$ (where cost is the number of 2-input gates in the implementation). The authors do not provide a strict proof of this conjecture, but support it with the following argument: A PCC receives two k-bit binary inputs: one k-bit input with the desired SN value v and one k-bit random input r. For each of the k bits of v to contribute to the PCC output with its correct weight, there has to be an operation (i.e. probability multiplication via AND gate) between each pair of bits of v and r. These operations require at least one 2-input gate per bit and therefore k gates in total. Afterwards, the resulting k bits have to be somehow combined into the single PCC output bit. Regardless of methodology, this requires at least another $k - 1$ 2-input gates, leading to a total cost of $2k - 1$ for this part of the SNG. A version of the WBG that achieves this minimal cost is presented in [YLL+18], a further cost reduction of this SNG part is therefore not possible. Furthermore, the cost of a k-bit LFSR can be considered close to the minimal cost of the RNG part, as it only consists of a k-bit register and a small number of XOR gates to generate its pseudo random sequence. The cost of an SNG consisting of an LFSR and a WBG can therefore not be improved by reducing the cost of its components.
This leaves the second option as the only possibly successful approach to cost reduction: Sharing parts between SNGs. [YLL+18] splits the WBG into a non-shareable and a shareable part with minimum cost. The remaining room for optimization therefore has to come from sharing one RNG among several SNGs. SNs that have been generated using the same random number source will be maximally correlated. In some instances, this correlation is desired, for example in the CORDIV divider (3.3) and the edge detection circuit from [ALH13]. Apart from these special instances though, correlation needs to be avoided in order to minimize correlation errors. The two main strategies to achieve this goal are:

1. Decorrelation by isolation of affected SNs.

2. Introducing an intermediate "scrambling" component between shared RNG and non-shared part of the PCC.

The general process of option 1 has been described in section 2.2.2, although in the case of RNG sharing it is not used to create uncorrelated duplicates of SNs, but rather to reduce cross-correlation between jointly generated SNs with generally different values. Isolation requires low autocorrelation of SNs, i.e. subsequent SN bits should be independent of each other in the optimal case. As SC commonly employs PRNGs, this is not a given and the extent of the correlation depends on the specific output sequence of the PRNG. Option 2 on the other hand potentially introduces additional hardware overhead in the SNG and can interfere with isolation. Consider for example an LFSR with maximum sequence length defined by the feedback polynomial $x^8 + x^6 + x^5 + x^4 + 1$ and initial state 00111010. Table 3.5 shows the states of this LFSR when implemented with external feedback function (also called Fibonacci LFSR) and internal feedback function (also called Galois LFSR) and the corresponding binary values that are used by the SNG.

Table 3.5: Sequence of LFSR states for Fibonacci and Galois LFSRs with feedback polynomial $x^8 + x^6 + x^5 + x^4 + 1$ and corresponding unsigned binary value.

Fib. LFSR state	Binary value	1 Bit cyc. shift	Gal. LFSR state	Binary value
00111010	0.2265625	0.11328125	00111010	0.2265625
00011101	0.11328125	0.5546875	00011101	0.11328125
00001110	0.0546875	0.02734375	10110110	0.7109375
00000111	0.02734375	0.51171875	01011011	0.35546875
00000011	0.01171875	0.50390625	10010101	0.58203125
10000001	0.50390625	0.75	11110010	0.9453125
11000000	0.75	0.375	01111001	0.47265625
01100000	0.375	0.1875	10000100	0.515625
00110000	0.1875	0.09375	01000010	0.2578125
10011000	0.59375	0.296875	00100001	0.12890625

It is clear that subsequent binary values in table 3.5 are not independent of each other. In fact, in many cases they are simply roughly half of the previous value (whenever a 0 is shifted in from the left), or the previous value increases by roughly a factor of 2^f where f is the most significant position with a 0 in the current state (whenever a 1 is shifted in

from the left). Changing the order of the shift, i.e. shifting in from the LSB on the right only reverses these patterns (values double instead of halve) but does not eliminate them. This high dependency between subsequent LFSR states leads to a high autocorrelation in the generated SN, which in turn can cause systematic correlation errors in a circuit.
A simple example can be found in the squarer circuit in figure 2.3b. With an input value of $x = 0.75$ the circuit would be expected to compute $y = x^2 = 0.5625$ on average. However, when a Fibonacci LFSR is used to generate the SN X, the circuit consistently outputs an SN Y with the value $y = 0.625$ (with minor fluctuations due to quantization errors). This error is much more serious than quantization errors or random fluctuation errors of the same order, as it affects the circuit systematically. In other words, it changes the implemented function of the circuit for this specific input value. When the input X is analysed in more detail, the reason for the error becomes clear (in the following one specific example of X is chosen for illustration):

$$X = 111011011110111110111110000110110001111 0...$$

No occurrence of a single 1 (also referred to as a run of length 1) appears in X, which would be highly unlikely if X where a Bernoulli sequence, which decorrelation by isolation is based on. For example, for an SN of length 128, the probability of having no run of length 1 is 0.289% [MP11] when bits are iid. The base assumption of isolation is apparently not met in this case and therefore the isolator in circuit 2.3b does not work as intended, leading to the systematic output error.

Circuits based on specific input correlations are affected similarly. The CORDIV divider (fig. 3.3) is based on input SNs X_1 and X_2 that are generated using a shared PRNG in order to achieve $SCC(X_1, X_2) = 1$. CORDIV outputs X_1^i in clock cycle i if $X_2^i = 1$ or repeats its output of the previous cycle $i - 1$ if $X_2^i = 0$. While the circuit generally works well together with LFSR-based SNGs, specific input value combinations can be found for which CORDIV fails similarly to the example above. One such combination is $x_1 = 0.5$, $x_2 = 0.75$. The expected result is $\hat{z} = \frac{0.5}{0.75} = 0.67$, however the observed output value is $z = 0.5$ when a Fibonacci LFSR is used in the SNG. Closer examination of X_1, X_2 and the

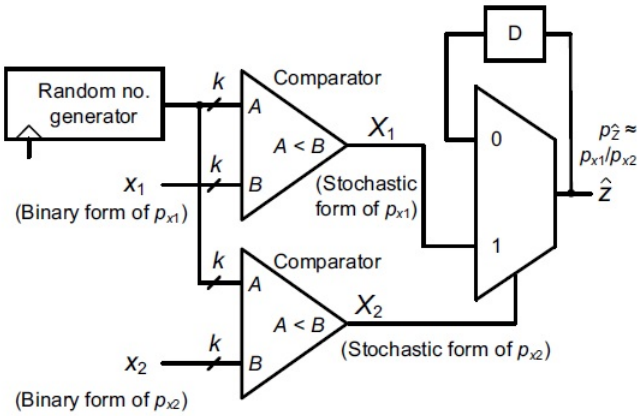

Figure 3.3: CORDIV divider circuit from [CH16].

output SN Z again reveals significant autocorrelation as the cause (as above, the following example is used for illustration):

$$X_1 = 100111001100101001100...$$

$$X_2 = 110111101110111101110...$$

$$Z = 100111001100101001100...$$

All 1s in the SNs overlap as required by the circuit, however the additional 1s in X_2 are distributed in a very specific manner: Whenever a run of 1s starts in X_1, a run of ones starts in X_2 at the same position, but exactly one bit longer. If SNs were Bernoulli sequences, the additional 1s in X_2 should be distributed with no such discernible pattern. In CORDIV, this pattern causes the output Z to be an exact bit-for-bit copy of X_1.

The most important type of SC components that can be heavily affected by autocorrelation is given by FSMs. Their use as activation functions in SCNNs makes them a critical component, yet their simple design cannot generally tolerate correlations. Figure 3.4 shows a comparison of the output function of stanh (figure 2.5) with eight states using input SNs generated by different RNGs: PN sequence, LFSR and Mersenne Twister (MT) [MN98]. MT is a complex software-based PRNG which is used as standard PRNG in several programming languages and tools, e.g. python and MATLAB. In the context of SC, its high

complexity and long period makes MT essentially behave like a true RNG and therefore a good reference to test for effects of correlation. Each data point in the figure is the average result of $1,000$ independent simulations with the corresponding RNG.

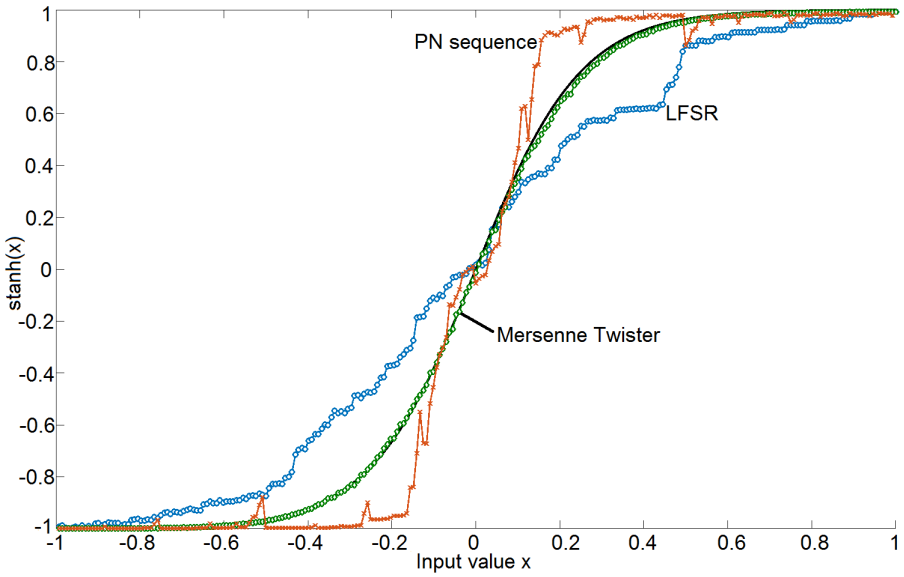

Figure 3.4: Simulation of stanh(4x), comparing different RNGs. The black line shows the expected output function. SN length 256.

PN sequence and LFSR show a clear deviation from the target function, while MT matches the expected result very well. The design of stanh does not take into account possible dependencies between subsequent input bits and is therefore not able to tolerate the autocorrelation in SNs generated by PN sequences and LFSRs. Low discrepancy sequences cause similar problems, as illustrated by the following example:
An SN X with value $\frac{2}{3}$ and length n generated using a Halton sequence has a regular pattern, e.g.

$$X = 011011011011011...$$

and the stanh FSM will traverse through its states according to table 3.6, assuming eight states in total and starting state S4. The resulting output SN Y will have a value of $\frac{n-1}{n}$ instead of the expected 0.583, as the FSM will never reach the left half of the FSM after clock cycle 2, because of the regular pattern of X with only single occurrences of 0s.

Table 3.6: Example for stanh state traversal with a Halton sequence-based input SN X.

X	State	Output
0	S4	1
1	S3	0
1	S4	1
0	S5	1
1	S4	1
1	S5	1
0	S6	1
1	S5	1
1	S6	1
...	...	1

Closely related to the issue of autocorrelation and isolation is the property of an RNG to produce different pseudo random numbers from the same state by shifting or scrambling of bits. If several random numbers can be produced in this way such that they lead to SNs with little correlation, isolation may be unnecessary. In [IIS+14] the authors explored circular shifting of LFSR states as a method to share an RNG without the need for isolation. A circular shift by $L/2$ where L is the LFSR size produced the best results (i.e. the lowest SCC between generated SNs), while small shifts by one or two bits cause significant cross-correlation. The third column in table 3.5 provides some intuitive reasoning for this effect, as the value of the right shifted state is often identical to the value of the original state one clock cycle later. Cyclic state shifting by one bit therefore often behaves not much different from isolation and can in fact interfere significantly with it. An SN generated using the original LFSR state in table 3.5 would be significantly correlated with an isolated (i.e delayed by one clock cycle) SN that was generated using the shifted state. Cyclical shifting by larger amounts however leads to fewer options for sharing. With a shifting amount of $L/2$, only two SNGs can share the same LFSR.

Some SC systems, especially digital filters and SCNNs can have a very large number of inputs. As filter systems are generally much smaller than SCNNs, the percentage of total circuit area occupied by RNGs in them is significant. In a 32-tap digital filter analysed in [ISI+16], LFSRs consume up to 70% of the total circuit area when no sharing is used. Through a carefully considered sharing strategy, the authors were able to reduce this area to only 3.6% at the cost of an increase in absolute error from $3.31 \cdot 10^{-2}$ to $3.94 \cdot 10^{-2}$

on average. This considerable improvement is possible in filters due to their mostly cross-correlation insensitive MUX-based structure and their purely combinational design, which makes them insensitive to autocorrelation as well.
However, SCNNs do not have these beneficial properties; they contain sequential components that are affected by autocorrelation and components that are potentially sensitive to crosscorrelation such as AMax. For example, an NN for the classification of the MNIST datasets has 784 input pixels and several hundreds of weight and bias values, out of which commonly 20 to 50 values (depending on the size of the convolution kernel) are part of one computation set. Within such sets, SNs are interacting directly with each other in MAC operations and activation functions, and outputs of different sets can be involved in the same max-pooling operation. In an SCNN with a 4×4 convolutional kernel, 16 weights and 16 input values are part of one computation set, as they are involved in the same MAC operation and subsequent activation function. Three other computation sets are independent up to this operation, but are part of the same max-pooling function (cf. figure 2.8). Using the sharing method with the best results from [IIS+14] would still result in a total of at least 16 required LFSRs, even if max-pooling is not considered.

3.2.2 SBoNG design and evaluation

A PRNG for SC that is designed to be shareable should satisfy two requirements: Produce sequences with low autocorrelation and include a scrambling component with low hardware cost. The S-Box based Number Generator (SBoNG) first presented in [NPH17] and [NPH18a] was designed with these requirements in mind.
In its basic form, SBoNG consists of an LFSR for area efficient generation of pseudo random numbers and a substitution box (S-box) used in a small scale version of the AES encryption scheme [GBH+16]. An S-box is a non-linear component used in cryptography that replaces fixed-size blocks of its input with different blocks of the same size according to a specifically determined mapping. The purpose of an S-box is to introduce a high-entropy step in an encryption scheme in order to make algebraic attacks on the cypher more difficult. Cyphers that use an S-box commonly consist of two different types of layers, a confusion layer and a diffusion layer. The diffusion layer is a combination of linear operations, such as shifting rows and mixing columns of the bit matrix that is currently

being encrypted. The S-box comprises the confusion layer and is used to break the linearity of the diffusion layer. As it is in essence just a look up table, it does not require much hardware area to be implemented directly, thus making it a good option for the scrambling component in SBoNG. The design schematic of SBoNG including connections to subsequent PCCs is shown in figure 3.5.

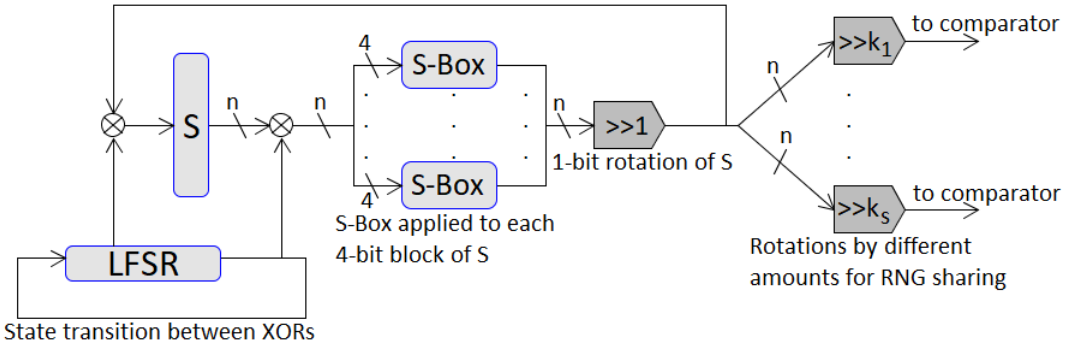

Figure 3.5: Design schematic of SBoNG with connections to PCCs.

SBoNG has an n bit internal state, which is randomly initialized. To generate a pseudo random number with SBoNG, the following steps are taken:

1. The internal state is XORed with the state of an n bit LFSR.
2. The resulting n bit state is split into blocks of 4 bit each, which are passed through (identical) S-boxes.
3. The S-box outputs are concatenated and the resulting n bit state is rotated by one bit to the right. This rotated value is considered the output of SBoNG.
4. The internal state is XORed with the LFSR state once more.

The XOR operation in step one ensures that every bit of the internal state has a roughly 50% chance of flipping compared to the previous cycle, which prevents sequences of highly dependent output values such as shown in table 3.5. However, subsequent states would still be linearly dependent on each other. The S-box in the second step breaks this linearity for each 4 bit block of the internal state. The rotation in step three causes an interaction between the otherwise largely independent blocks and ensures that the entropy introduced by the S-boxes is spread over the entire n bit state.

One downside coming with the use of S-boxes is a restriction in state size. As the internal state has to be split into blocks of size 4 bit, n can only be a multiple of 4. The S-box from [GBH+16] was chosen specifically because of its 4 bits block size, in contrast to the 8 bit size in regular AES. A small block size ensures a particularly small hardware implementation and enables a wider range of internal state sizes. The specific substitution mapping of SBoNG's S-box is given in table 3.7 and the corresponding hardware implementation in figure 3.6.

Table 3.7: Substitution mapping of the 4 bit S-box in SBoNG (in hexadecimal).

Input:	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output:	6	B	5	4	2	E	7	A	9	D	F	C	3	1	0	8

The impact of the S-box as a non-linear component can be seen clearly when autocorrelation of generated SNs is measured and compared to SNs generated by LFSR-based SNGs. SBoNG generates SNs with an autocorrelation factor A_1 according to equation (2.7) that is one to two orders of magnitude better on average. Table 3.8 shows the average A_1 and A_2 factors of $1,000$ independently generated SNs of length 256 bit for selected randomly chosen input values.

Table 3.8: Autocorrelation values of SNs generated with SBoNGs and LFSRs.

SN value	A_1		A_2	
	SBoNG	LFSR	SBoNG	LFSR
0.15	-0.002	0.40	-0.02	0.12
0.23	0.002	0.35	0	0.02
0.30	-0.003	0.28	-0.03	0.17
0.41	0.003	0.15	-0.03	0.15
0.55	-0.003	0.09	-0.03	0.09
0.63	0.001	0.28	-0.08	0.20
0.70	0.001	0.27	-0.06	0.16
0.83	0	0.39	-0.03	0.10

Across all input values, autocorrelation of SNs generated by SBoNG are much lower than their LFSR-generated counterparts. An interesting observation is that A_2 factors are larger than A_1 factors for SBoNG. This is likely caused by the XORing of subsequent LFSR states, which does not get entirely de-linearised by the S-box. In cryptography, cyphers with S-

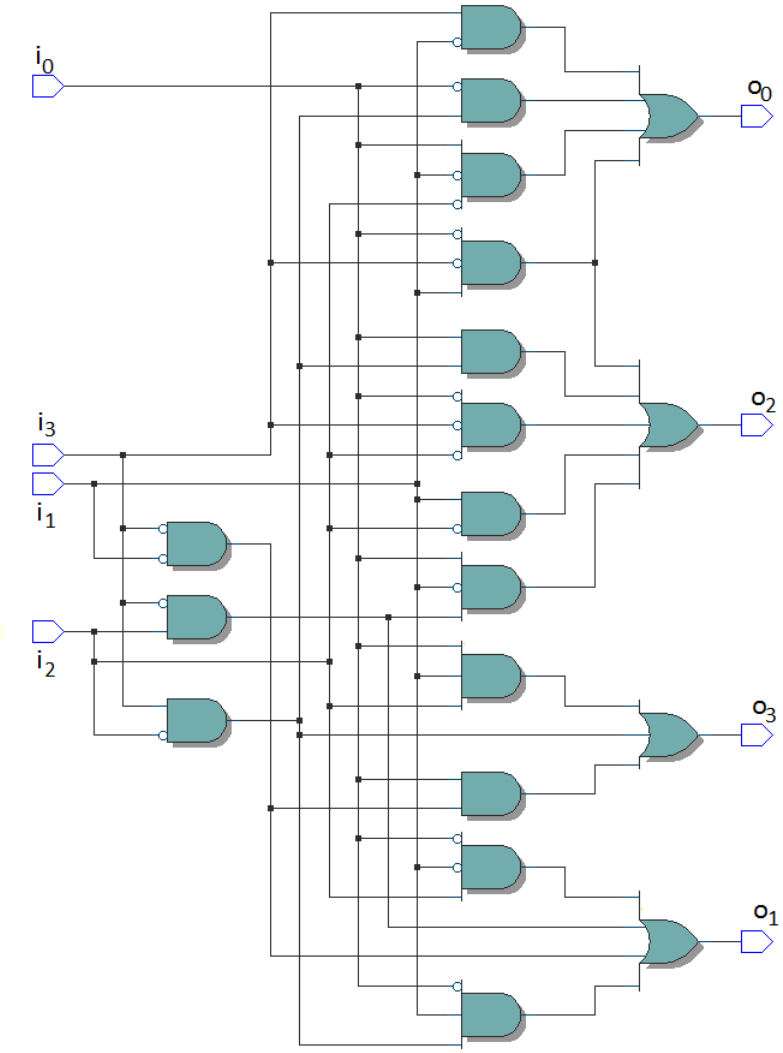

Figure 3.6: Hardware implementation of the S-box from table 3.7 with input bits $i_0, \cdots, i_3$ and output bits $o_0, \cdots, o_3$.

boxes commonly consist of several applications of the previously mentioned confusion and diffusion layers to prevent such residual linear dependencies. This would however cause a significant area overhead in SC, where even a single LFSR is already considered to be a relatively large component. Even so, SBoNG's A_2 factors are still significantly smaller than those of the LFSR, and are highly unlikely to impact SC operations to a noticeable degree. This is further corroborated by the performance of FSM-based SC components in combination with SBoNG. In figure 3.4, the deficiencies of LFSR and PN sequences have been

illustrated for stanh. The same simulation repeated for SBoNG is shown in figure 3.7. SBoNG almost perfectly matches the expected output function and is virtually indistinguishable from MT for most input values. Only very minor differences between the two can be seen for input values in $[-0.4, -0.2]$ and $[0.2, 0.5]$.

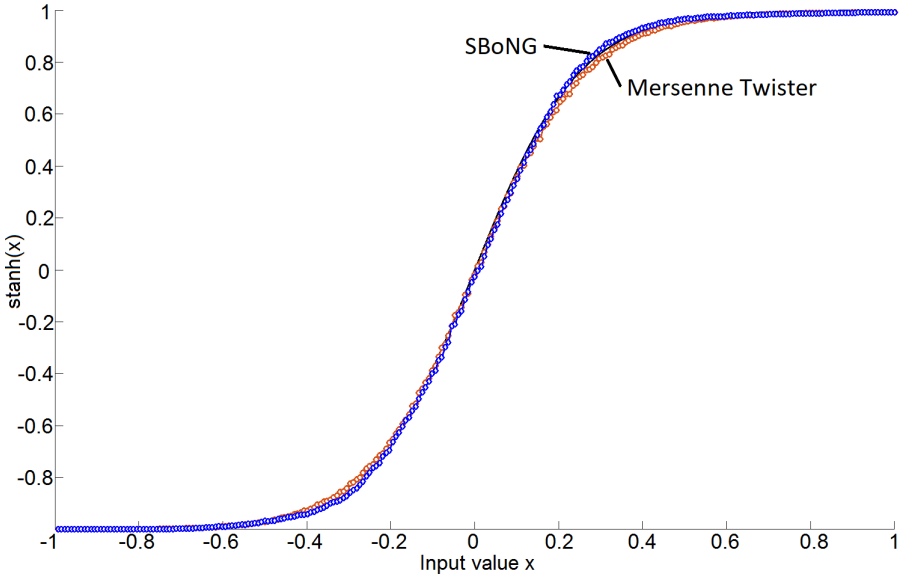

Figure 3.7: Simulation of stanh(4x) with SBoNG (blue) and MT (red) used as RNGs. The black line shows the expected output function. SN length 256.

A further important point to investigate is SBoNG's compatibility with isolation and cross-correlation. Copying an SN through a fan-out with isolation on the fan-out lines is a common practice that is of great importance in polynomial circuits. Basic building blocks shown in figure 2.3 allow simple computation of monomials with only a single input SN. The isolated SN should have low cross-correlation with the input SN for the multiplication to be accurate. As shown in table 3.9, SBoNG clearly meets this target and reaches cross-correlation values that are only slightly higher than those obtained with MT.
Values in the table were obtained after $1,000$ independent simulations for both SBoNG and MT. For comparison, the SCC of two SNs obtained by LFSR rotation method [IIS+14] is included as well. The SCC values of SBoNG are almost as low as those of MT, meaning that SBoNG performs almost as well as a significantly more complex software-based RNG in the context of SC. Higher SCC values in the table, e.g. in the last line are due to the manner in which SCC is computed, which causes SNs with values towards the edges of

Table 3.9: Cross-correlation (SCC) value examples of SNs generated with SBoNG, MT and LFSR rotation.

SN value	SBoNG		MT		LFSR
	min	max	min	max	
0.18	0.074	0.109	0.073	0.113	0.401
0.39	0.006	0.021	0.005	0.023	0.180
0.58	0.002	0.016	0.001	0.012	0.149
0.77	0.051	0.086	0.043	0.079	0.354
0.90	0.183	0.275	0.177	0.246	0.468

the $[0, 1]$ range to have higher SCC (see section 2.2.2). In conclusion, SBoNG is fully compatible with isolation and FSM-based sequential SC components.

3.2.3 Statistical analysis of SBoNG

Statistical properties of RNGs are often evaluated with specifically designed test suites such as dieharder [Bro] and the NIST test suite [RSN+]. These test suites are aimed at RNGs for cryptographic applications, which require a much higher complexity that LFSR and SBoNG provide. It is therefore expected that both of them do not pass the entire tests suites. Nevertheless, several of the simpler tests in the NIST suite were performed to investigate if the different behaviours of these RNGs in SC can be traced to specific properties of their generated sequences. For comparison, MT was also tested. MT uses significantly more hardware area than LFSR and SBoNG and its inclusion here should only provide context, it should not be seen as a viable alternative in SC.

The following tests from the NIST test suite are performed: Frequency, normal runs, longest runs in a block, discrete Fourier transform (DFT), cumulative sums and serial test. All tests are performed with input sizes (i.e. length of tested sequences) recommended in the NIST documentation. Each test consists of 100 runs, an RNG passes a test when at least 96 runs are successful. A P-value is used to evaluate each test run. The P-value is the probability for observing the specific test outcome under the condition that the input sequence is indeed random. If the P-value is larger than 0.01, the run is considered successful.

The specifics of each test vary and an extended description of each test would be out of

place here. Therefore, only the frequency test will be explained in some detail to illustrate the general procedure, and a short description of the remaining tests will be given.

Frequency test procedure: The frequency test focuses on the proportion of 1s and 0s in the test sequence. In a truly random bit stream, the ratio of 1s and 0s should be one, the test focuses on the fraction of 1s, which should be $\frac{1}{2}$ in this case. A bit stream $s = s_1, s_2, ..., s_n$, $n \geq 100$ is generated as input for each test run. This bit stream is the output of an RNG, and is not to be confused with an SN, which is the output of an SNG that is supplied by the RNG. A parameter s_{obs} is computed for the sequence according to

$$s_{obs} = \frac{\sum_{i=1}^{n}(2s_i - 1)}{\sqrt{n}} \tag{3.8}$$

from which the P-value is further determined by

$$\text{P-value} = erfc\left(\frac{s_{obs}}{\sqrt{2}}\right) \tag{3.9}$$

where $erfc$ is the error function:

$$erfc(z) = \frac{2}{\sqrt{\pi}}\int_{z}^{\infty} e^{-u^2}du \tag{3.10}$$

For example, let $s = 1001011110$ with $n = 1$, then $s_{obs} \approx 0.632$ and P-value ≈ 0.527. s is therefore considered to be random and this test run is considered successful.

Normal runs test: The number of runs (consecutive bits of the same value) of 1s and 0s is determined and compared to the expected numbers from a true random sequence.

Longest runs in a block test: The longest run of 1s in an M bit block is compared to the length expected from a true random sequence.

Discrete Fourier transform (DFT) test: A DFT is applied to the sequence and the peaks of the resulting transform are examined. This test is used to detect periodic features, i.e. repeating patterns in the sequence that are not expected from truly random sequences.

Cumulative sums test: A random walk on the sequence is performed and the sum $\sum(2s_i - 1)$ is computed after every step i of the walk. The maximal absolute value of the series of sums is determined, higher values suggest insufficient randomness. This test consists of two parts: a forwards and a backwards pass.

Serial test: This test determines the frequency of all possible overlapping m-bit and $m-1$-bit patterns in the sequence. In a truly random sequence, every one of the possible patterns is expected to appear with the same frequency.

The results of these test are shown in table 3.10:

Table 3.10: Results of performed NIST tests for LFSR, SBoNG and MT.

Test	**LFSR**	**SBoNG**	**MT**
Frequency	100	100	100
Normal runs	98	99	100
Longest runs in block	97	99	100
DFT	97	100	100
Cumulative sums part 1	100	100	100
Cumulative sums part 2	100	100	100
Serial part 1	98	100	100
Serial part 2	98	99	100

Interestingly, differences between LFSR and SBoNG test results are almost non-existent. The results from these tests with the recommended test parameters do not seem to relate to the performance of RNGs in SN generators. This is especially noticeable in the serial test, which is of particular interest for SN generation. Subsequent bits of an SN are generated using subsequent states of an RNG. If the output of the RNG is seen as a bit stream, as it is in order to generate the input sequences for the test, this is equivalent to using blocks (states) of length k with an overlap of $m = k - 1$ between consecutive blocks. The recommended overlap in [RSN$^+$] is $m_{rec} < \lfloor \log_2 N \rfloor - 2$ for sequences of length N. However, in a stochastic circuit with SN length n, the sequence length of the RNG is commonly chosen such that $N \geq 2^{\lceil \log_2 n \rceil}$, with equality being preferred to keep hardware cost of RNGs as low as possible. The overlap between consecutive states used for SN generation is therefore $m_{SC} = \lceil \log_2 n \rceil - 1$, while the recommended parameter for

the test is $m_{rec} < \lceil \log_2 n \rceil - 2$.

Consider for example a $k = 8$ bit LFSR with a maximum sequence length of 255. For the purpose of the serial test, the full output sequence is viewed as a single bit stream $b_1 b_2 ... b_{255}$ with 8-bit blocks and an overlap of $m_{rec} < \lceil \log_2 255 \rceil - 2 = 5$, e.g. $B_1 = b_1 ... b_8$ and $B_2 = b_5 ... b_{12}$. In an SNG, the situation is different: in the first clock cycle, the state of the LFSR, which is used to generate the first SN bit, is $S_1 = b_1 ... b_8$. The next SN bit in the second clock cycle is generated using the state corresponding to $S_2 = b_2 ... b_9$, an overlap of 7 bits. The serial test with the recommended settings is therefore not capable to detect correlations caused by the additional 3 overlapping bits.

Additionally, different significance of bits is not considered in these tests, as the test input sequences are seen as a constant stream of equally significant bits. In an SNG however, the states of the RNG are interpreted as unsigned binary numbers, e.g. in the above example $B_1 = b_1 ... b_8$, the bit b_1 has significance 2^{-1} while the bit b_8 has significance 2^{-8}. High significance bits of course have more influence on the generated SN bits and correlations between them therefore have a larger effect. This could account for the differences between LFSR and SBoNG regarding autocorrelation and cross-correlation even though both RNGs perform almost identically in NIST tests. It is thus vital to evaluate PRNGs in the context of SC using SC-specific metrics such as SCC or, if possible, through simulation of specific SC components directly.

To illustrate the practical impact of SBoNG, a digital signal filter, a popular application for SC (e.g. [ISI+16] [WHCE16] [YW16]), is simulated for different SNG configurations. In particular, a 31-tap finite impulse response (FIR) filter is implemented as a MUX tree as described in [ISI+16], which is an equivalent, though slightly modified, version of the architecture in [WHCE16]. Digital filters have several properties that make them profit from shareable RNGs: Firstly, they have a large number of input values, split between varying data inputs and constant filter coefficients. Secondly, the MUX-based core component of the filter has multiple correlation sensitive select inputs. Thirdly, SNGs make up a large part of such a filter's hardware area due to the small arithmetic core component and the large number of inputs, thus the filter benefits greatly from reduced SNG costs. Another major advantage of SBoNG comes into play here: As SBoNG includes a basic LFSR, correlation insensitive inputs such as the data inputs can be generated using only this LFSR,

which allows a quantization error free SN generation, while correlation sensitive inputs such as the MUX select inputs can be generated using the full SBoNG architecture to ensure fewer correlation errors.

For this application, a 31-tap low pass filter with filter coefficients generated in MATLAB was implemented. The number of taps was chosen such that the SC filter can be implemented by a balanced MUX tree with five levels. The SN length used is 2^{16} which is quite long compared to SN length in other SC applications such as image processing, but is necessary to achieve the higher accuracy requirements of digital filters according to [WHCE16]. A 16 bit internal state is therefore needed for SBoNG. A five level MUX tree has five select inputs that are generated from a single SBoNG instance by rotating the SBoNG output by multiples of three bit for each of these inputs (corresponding to $k_1, ..., k_S$ in figure 3.5). Data inputs are all generated from this SBoNG's internal LFSR. No decorrelation is necessary, as the data input SNs are insensitive to crosscorrelation. The output of the stochastic filter on an electrocardiogram signal from the PhysioBank database [GAG+00] (original ECG data from [Lug05]) is depicted in figure 3.8. For comparison, results of an implementation using a single LFSR with the optimal rotation method found in [IIS+14], and an exact floating point computation for reference are also included in the figure.

The output of the SBoNG-based stochastic circuit matches the reference result almost exactly, while the LFSR-based circuit is not working as intended and does not properly function as a low pass filter. In order to achieve similarly accurate results, the five select signals of the MUX tree would have to be implemented using multiple independent LFSRs. The SBoNG-based implementation achieves an MSE of $7.00 \cdot 10^{-6}$, which lies slightly below the predicted upper bound of $1.53 \cdot 10^{-5}$ according to equation 2.12. On the other hand, the MSE of the LFSR-based circuit is 0.0027 which is not even significantly better than the MSE of the unfiltered input signal at 0.0164. These numbers can be interpreted as the noise of the respective signals compared to the reference result. Note that stochastic circuits technically remove all original input noise and instead introduce new noise sources in the form of quantization and random fluctuation errors. This is true for both SBoNG and LFSR, the difference between these two implementations thus stems from correlation errors, which SBoNG successfully eliminates.

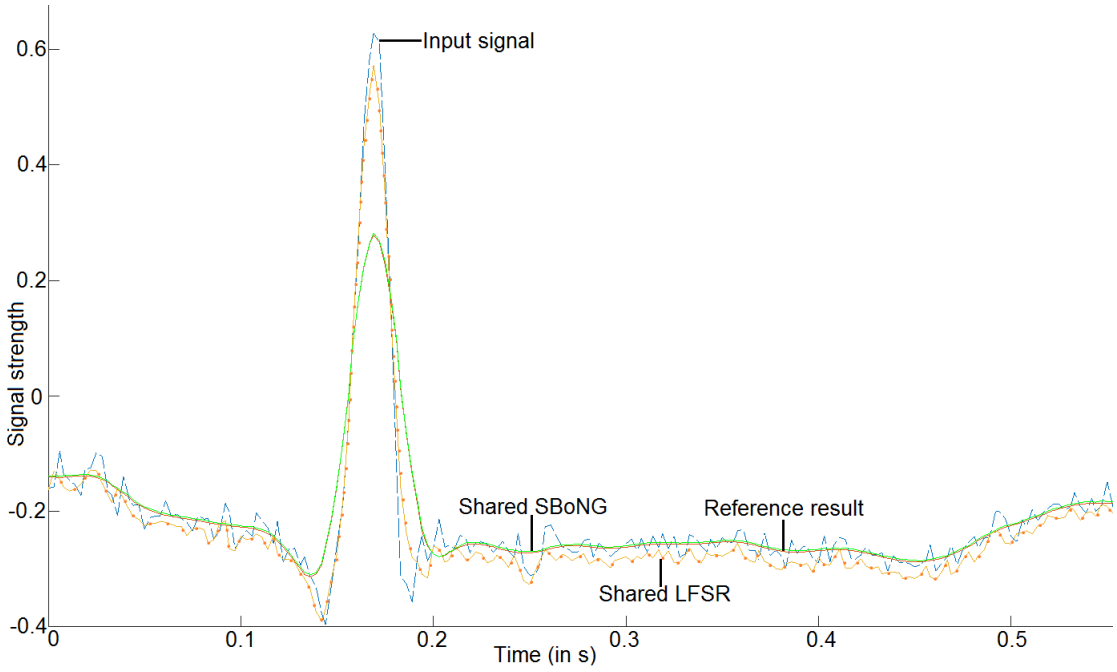

Figure 3.8: Results of SC low pass filter using SBoNG (green line), LFSR with rotation (orange line) and floating point reference (red line). Blue line shows the input signal.

Besides eliminating correlation errors, reducing the hardware overhead of SNGs was another design goal of SBoNG. As explained at the beginning of this section, the PCC part of an SNG is independent of the RNG and cannot be shared or reduced in size beyond a specific point. The benefit of an SBoNG based SNG therefore have to come from its capacity to be shared among multiple SNGs without introducing correlation errors. Table 3.11 shows the cost (in 2-input gates and flip-flops) of individual SNGs as well as a full SCNN layer (following the design from section 4.1.2) for SBoNG and LFSR-based implementations.

A single SBoNG instance is of course much larger than a single LFSR, but SBoNG's ability to supply multiple SNGs without causing issues due to cross-correlation quickly lets it pull ahead of LFSR-based SNGs in terms of sequential cost. The majority of the combinational cost comes from the comparators and not from the PRNGs. As there are less than 100 primitive polynomials of degree 8, the numbers for LFSR-based SNGs are extrapolated in this case. Note that SBoNG's internal state length l has to be a multiple of four due to the 4-bit S-Box inputs. This means that for an SN length other than 2^l, SBoNGs internal state has to be larger than that of a single LFSR. For example, for an SN length of 1024, a 10 bit

Table 3.11: Sequential and combinational cost of SNGs and SCNN layer implemented with SBoNG and LFSR.

	SN length	LFSR		SBoNG	
		seq.	comb.	seq.	comb.
1 SNG	256	8	51	16	114
	4,096	12	83	24	178
5 SNGs	256	40	254	16	302
	4,096	60	413	24	490
10 SNGs	256	80	505	24	570
	4,096	120	826	24	880
100 SNGs	256	(800)	(5,100)	208	5,571
	4,096	1,200	8,251	216	8,698
SCNN layer	4,096	2,169	62,776	1,817	62,860

LFSR is sufficient, however SBoNG needs a 12 bit internal state. Even so, efficient sharing enables significantly lower overall sequential hardware cost for circuits requiring three or more uncorrelated inputs.

Chapter 4

Error Resilience

4.1 Robustness of stochastic circuits

Two often mentioned advantages of SC are its supposedly high tolerance of soft errors due to the structure of SNs, and its low power requirement due to its smaller circuit sizes. SC has therefore been proposed for use in noisy and power-constrained environments, for example in battery-powered sensor nodes. Indeed, the tolerance of any single SN towards bit flips is significant and much larger than in conventional binary number formats. Any bit flip in an n-bit unipolar SN changes its value only by $\frac{1}{n}$, while it can possibly change the value of a binary number by $\frac{1}{2}$. Furthermore, bit flips in opposite directions even at different positions cancel each other out, as the total number of 1s in the SN stays the same. These beneficial properties have been confirmed to carry over to whole SC systems in image processing, as shown in [ALH13] and [LL11] among others. More recently however, as SC shifted towards NNs as its main application, stochastic components have become more complex and a larger proportion of sequential stochastic circuits has been introduced.

The behaviour of these newer components differs substantially from the mostly combinational image processing systems. Specifically, soft errors in sequential circuits have the potential to affect subsequent clock cycles, while the effects are isolated to single clock cycles in combinational circuits. The more substantial impact of errors in sequential circuits have been demonstrated on the example of FSM-based stochastic components, for example on the stanh function in [IMII19]. The authors showed that the binary state encoding

in an FSM drastically reduces the robustness of this component. In [ACH+17] a detailed investigation of voltage and frequency scaling of stochastic circuits, mostly focused on (combinational) image processing SC systems, has been performed. The authors conclude that these circuits are extremely tolerant of timing errors, making it an attractive technique for aggressive voltage and frequency scaling. The significantly different behaviour of sequential and combinational stochastic circuits in these works suggests that the error tolerance from individual SNs does in general not carry over to entire SC systems, but is instead highly dependent on specific implementations.

One possibility to rigorously analyse the reliability of a circuit with probabilistic behaviour is to construct a Markov-chain (MC) model for it. In both works mentioned above an MC model has been used to complement the experimental data. MC models describe a system as a set of states similar to an FSM. Transitions between states in an MC model are however probabilistic instead of an FSM's deterministic transitions. This enables the theoretical analysis of a system in the presence of errors that appear with given probabilities. For example, a correctly functioning 3-bit counter that is unaffected by errors has a probability of 1 to transition from the state 000 to the state 001, while all other transitions have probability 0. Assuming an independent bit flip probability of 0.01 for each bit, the corresponding MC model for such a counter is given in table 4.1.

Table 4.1: Markov-chain model for a 3-bit counter with bit flip probability 0.01. Rows are counter states in clock cycle i, columns are counter states in cycle $i+1$. Due to rounding errors, probabilities add up to slightly more than 1.

State	000	001	010	011	100	101	110	111
000	0.010	0.970	$9.9 \cdot 10^{-5}$	0.010	$9.9 \cdot 10^{-5}$	0.010	$1.0 \cdot 10^{-6}$	$9.9 \cdot 10^{-5}$
001	0.010	$9.9 \cdot 10^{-5}$	0.970	0.010	$9.9 \cdot 10^{-5}$	$1.0 \cdot 10^{-6}$	0.010	$9.9 \cdot 10^{-5}$
010	$9.9 \cdot 10^{-5}$	0.010	0.010	0.970	$1.0 \cdot 10^{-6}$	$9.9 \cdot 10^{-5}$	$9.9 \cdot 10^{-5}$	0.010
011	0.010	$9.9 \cdot 10^{-5}$	$9.9 \cdot 10^{-5}$	$1.0 \cdot 10^{-6}$	0.970	0.010	0.010	$9.9 \cdot 10^{-5}$
100	$9.9 \cdot 10^{-5}$	0.010	$1.0 \cdot 10^{-6}$	$9.9 \cdot 10^{-5}$	0.010	0.970	$9.9 \cdot 10^{-5}$	0.010
101	$9.9 \cdot 10^{-5}$	$1.0 \cdot 10^{-6}$	0.010	$9.9 \cdot 10^{-5}$	0.010	$9.9 \cdot 10^{-5}$	0.970	0.010
110	$1.0 \cdot 10^{-6}$	$9.9 \cdot 10^{-5}$	$9.9 \cdot 10^{-5}$	0.010	$9.9 \cdot 10^{-5}$	0.010	0.010	0.970
111	0.970	0.010	0.010	$9.9 \cdot 10^{-5}$	0.010	$9.9 \cdot 10^{-5}$	$9.9 \cdot 10^{-5}$	$1.0 \cdot 10^{-6}$

The state transition probability $P(\{S \to \hat{S}\})$ for an n-bit counter can be obtained with the formula

$$P\left(\{S \to \hat{S}\}\right) = \prod_{k=1}^{n} \left(|(S+1)_k - \hat{S}_k| \cdot p + ||(S+1)_k - \hat{S}_k| - 1| \cdot (1-p)\right) \tag{4.1}$$

where S_k is the k-th bit of state S and p is the bit flip probability. Markov Chain models are a useful tool for the analysis of error affected circuits, as they can accurately (assuming the error probability is known) predict the circuit behaviour over an arbitrary number of clock cycles. Unfortunately however, such analysis methods don't scale well. The number of state transitions grows exponentially with the number of sequential elements, i.e. flip-flops in the circuit. Generally, the state transition table will have $(2^n)^2$ entries for a circuit with n flip-flops. This poses a problem for the general error analysis using Markov chains in SC, mainly for two reasons:

Firstly, SNGs should be part of this analysis. A faulty SNG can have a major effect on a circuit's accuracy. For instance, an LFSR that is put into the all-0 state due to a fault cannot function as a PRNG any more. Similarly, deterministic SC depends on specific correlations and overlapping patterns in the generated SNs, such as obtained by rotation and clock division methods [NJLR19], or matching low-discrepancy sequences [FNL+19]. If these correlations and patterns are disturbed, computations can potentially become highly inaccurate even after a single error. SNGs always consist of at least one register, and in most SC systems, several SNGs are needed to avoid undesired cross-correlation. They thus increase the state space of the SC system significantly.

Secondly, SCNNs have many independent computations that are commonly implemented in parallel to reduce computation time. This includes sequential components, which therefore also increases the state space of the overall circuit massively. For example, a single 8-state stanh FSM has a state transition table with 64 entries, while two of them in parallel have a combined table with 4096 entries, as there are 64 possible state combinations.

The use of Markov Chain models to analyse large-scale SC systems, especially SCNNs, is therefore infeasible. A simulation-based approach is preferable, although it does not come with the mathematical certainties that a Markov Chain model provides. Previous works that employed MC models for the analysis of SC were restricted to relatively small systems such as a single FSM [IMII19] or combinational circuits [ACH+17]. Furthermore, probabilistic models make it hard to accurately model errors that appear in non-random

fashion, for example timing errors that may only appear at certain operating frequencies and on certain paths.

The effect of such timing errors is of special interest to SC, as it is often proposed for use in low-power systems. In such systems, the supply voltage is lowered from the nominal value down to a value slightly larger than the threshold voltage of the transistors in order to save energy (e.g. Razor [EKD+03]). This has significant effects on a circuit's susceptibility towards errors: A smaller margin between operating and threshold voltage makes transistors more vulnerable to physical variations (i.e. process variations during manufacturing) as well as outside effects such as noise on the power supply network or effects of radiation. Moreover, lower supply voltages increase the rising and falling signal delays of gates, effectively slowing down the circuit. If this slow down becomes too large, set-up and hold times of flip-flops may not be met any more and incorrect values are captured by them. These incorrect values corrupt the state of a sequential component and disturb its functionality. In contrast to single event upsets due to radiation, timing errors do not appear randomly with certain probabilities over the whole circuit area. They are instead clustered at specific points of the circuit, for example the flip-flop at the end of the circuit's critical path. As rising and falling delays of gates are generally not identical, they can also be biased towards a transition direction, i.e. favour $1 \rightarrow 0$ or $0 \rightarrow 1$ errors.

While SC is often proposed for use in low-power systems due to its assumed robustness, a detailed investigation, especially of potentially more vulnerable hybrid SC-binary circuits, has been lacking so far. The aforementioned works are restricted to simplified error models and/or relatively small and mostly combinational SC systems. If SC is supposed to be deployed in noisy environments, its robustness has to be properly investigated and its potential risks should be known.

In [NHP22], a detailed investigation of several state-of-the-art SC components has been performed, including a large SC system in the form of a full layer of a convolutional SCNN including SNG sharing. Both errors due to timing violations as well as single event upsets on gate level are considered as potential causes for errors. The results of this investigation are presented in this section.

4.1.1 Capture errors and simulation procedure

Any signal that transitions between logical values does so with a delay that depends on the length of the path, i.e. the number of gates it has to pass through. Circuits are designed with a timing margin, which ensures that all signals stabilize before they are captured by flip-flops. Each gate on a path has its own transition delay that depends on the type of the gate and the direction of the transition and adds to the path's overall delay. Gate delays increase when the supply voltage V_{DD} of the circuit decreases. A smaller supply voltage therefore leads to an increased signal propagation delay, in essence reducing the timing margin of the circuit. If the voltage is reduced too far, signals may not stabilize in time and wrong values can be captured (in the following called a *capture error*). In contrast to random bit flips, capture errors do not affect all gates and paths equally, as short paths have a larger timing margin than long paths. Figure 4.1 shows an illustration where a long path causes a capture error due to increased propagation delay, while the signal on a short path still stabilizes in time.

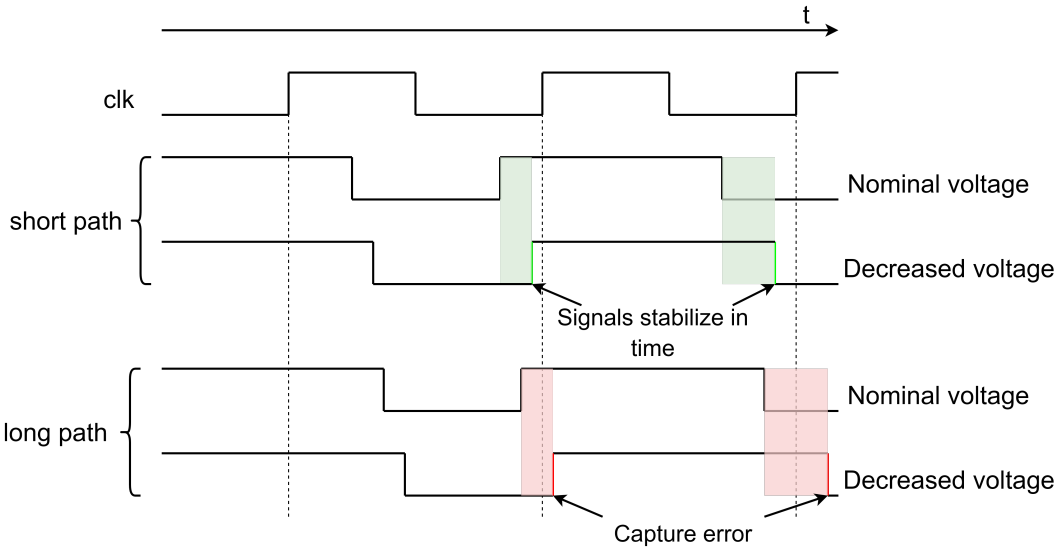

Figure 4.1: Illustration of capture errors in longer paths under decreased supply voltage.

In extreme low-power systems, V_{DD} is decreased to be only slightly higher than the threshold voltage of a gate's transistors (so-called *near-threshold operation*). When operated in this range, small manufacturing differences in the transistors also make such designs vulnerable to voltage variations due to external noise (e.g. from the power delivery network)

or faults caused by radiation, adding additional error sources. In contrast to conventional binary computing, the natural error tolerance of SNs may enable smaller timing margins and thus higher operating frequencies, as occasional capture errors only have a small effect on an SN's value. SC is therefore a promising candidate for extreme low-power systems, however the extent to which more recent FSM-based or hybrid SC-binary circuits can tolerate these errors has to be analysed in detail.

Investigating the overall effect that near-threshold operation has on an SC system requires extensive simulation including variations in individual gate delays. In contrast to basic logic simulation, these timing simulations have to be performed with signal wave forms, including timing information of each logic gate. Stefan Holst from the Kyushu Institute of Technology has developed a GPU-based simulator for such detailed timing simulations. The simulator is not part of this work, for details on its design and functionality, see [HIW15]. The simulator does not offer the functionality of changing voltages directly, however it can model the effects of voltage changes by modifying delays of individual gates. For the simulations that were performed to obtain the results presented in this section, this modelling was done in the following way: The Verilog files of the base circuits are synthesized with Synopsis Design Compiler, resulting in gate-level net-lists including nominal delay information for gates and interconnects. For any delay d in this net-list, the simulator adds a random variation according to a normal distribution $\mathcal{N}(\mu, \sigma)$, with mean $\mu = d$ and and standard deviation $\sigma = s \cdot d$ with a configuration parameter $0 < s \leq 1$. These individual delays are assigned independently before a particular simulation is started to every gate and interconnect to model manufacturing differences in the components. The circuit-wide increase in delay due to lower voltage is modelled by multiplying all delays d with a factor $d_f > 1$ called *delay factor* before simulation start.
A GPU-based simulator is desired in this case due to the enormous number of individually simulated values, even in small networks. The LeNet5 architecture [LBBH98] as the most well-known example of a (for today's standards) small convolutional network outputs $150,528$ bits in its first layer (assuming 32-bit fix-point format). A timing simulation of the layer requires individual simulation of each of these bits, each time taking into account all delay and waveform information on the entire path through the circuit. GPUs make it possible to distribute these simulations, which consist of many individually simple steps, over

many independent simulation threads. The structure of the underlying task lends itself well to parallelization, as it is composed of operations that are independent of each other on several levels: On the network level, all input images are processed independently by an SCNN. A GPU can therefore simulate the same layer with two different inputs at the same time without running into dependency problems. On the layer level, each kernel in a layer operates independently of all other kernels. Simulation of kernels can thus be performed in parallel as well. On the level of individual kernels, each convolution operation consists of multiple independent MAC operations on different areas of the input data, as the kernel moves over this data. This enables the computation of a full feature map in a single step by computing all involved MAC operations in parallel.

4.1.2 Design of simulated circuits

While there effectively exists a standard SCNN neuron design, the individual components can vary. SNGs may be implemented with different types of PRNGs, adders may be implemented as accurate or approximate parallel counters and max-pooling may be performed using different versions of stochastic maximum circuits. To take these differences into account, several different versions of the stochastic design were implemented using varying components described in the following.

SNGs: Differences in the number generator of an SNG have two main consequences: Firstly, an LFSR as a sequence-based generator will have a short portion of its natural sequence repeated when its state changes due to an error. This can cause an increased quantization error in the resulting SN, as the sequence length and SN length do not match any more. In the worst case, an LFSR is put into the all-0 state from which it cannot recover, causing the SNG to output only 1s for the remaining runtime.
Secondly, different number generators have varying potential for SNG sharing (cf. section 3.2). As an RNG provides its output to multiple SNGs, an error also spreads to more SNGs and its effects potentially cover a larger portion of the circuit and influence more outputs. For the simulations in this section, LFSR and SBoNG were considered as PRNGs.
MAC and activation function: This component follows the standard design using parallel XNOR gates for multiplication with an SC-binary hybrid approximate parallel counter

for addition. The subsequent hyperbolic tangent activation function is implemented using the FSM-based binary-SC hybrid Btanh circuit [KKY+16]. As a sequential circuit, Btanh is potentially very susceptible to timing errors that corrupt the FSM's state.

Max-pooling: Max-pooling is implemented in two different ways to investigate the influence of sequential depth on the robustness of SC components: The AMax component from [RLD+17] and the NMax circuit presented in section 3.1 are considered. Both circuits function according to the same principle: A counter tracks which input SN has the (approximate) maximum value and routes this SN to the circuit's output. The main difference lies in the periodic reset of counters in AMax, which is absent in NMax. The counter reset in AMax is used to periodically update its estimate of the maximum input value. This causes the circuit to generally output inaccurate values slightly smaller than the maximum in an error free case. However, it may make AMax more robust towards errors, as the periodic reset acts as a recovery mechanism. In NMax on the other hand, any error can potentially affect all subsequent output bits. The reset period in AMax is not fixed in [RLD+17]; for the simulation in this section a reset period of 32 clock cycles was chosen, as it provided a good middle point between accuracy in error-free cases and robustness in experiments (see also table 3.4).

These components are assembled into a full convolutional SCNN layer with 20 convolution filters (kernels) of size 4×4 using a tanh activation function, and a 2×2 max-pooling operation. Fig. 4.2 shows the RTL schematic of the resulting circuit.

The design is highly parallelized. On the highest level, all 20 kernels are computed in parallel, as they operate independently of each other. For each kernel, MAC-operation, activation function and max-pooling are likewise computed in parallel. While they are connected in sequence in the schematic, they effectively form a pipeline with each stage processing one SN bit in every clock cycle and thus operate on the same SN simultaneously. In addition to the functional components, the full SC circuit contains an intermediate register stage between SNGs and MAC-operation. This register stage splits the long combinational path between PRNG registers and state registers of the Btanh FSM, which increases the maximum possible operating frequency of the circuit significantly. In order to provide all four required inputs to the max-pooling operations at the same time, four parallel copies of the MAC-activation function block are used per kernel. These copies cover

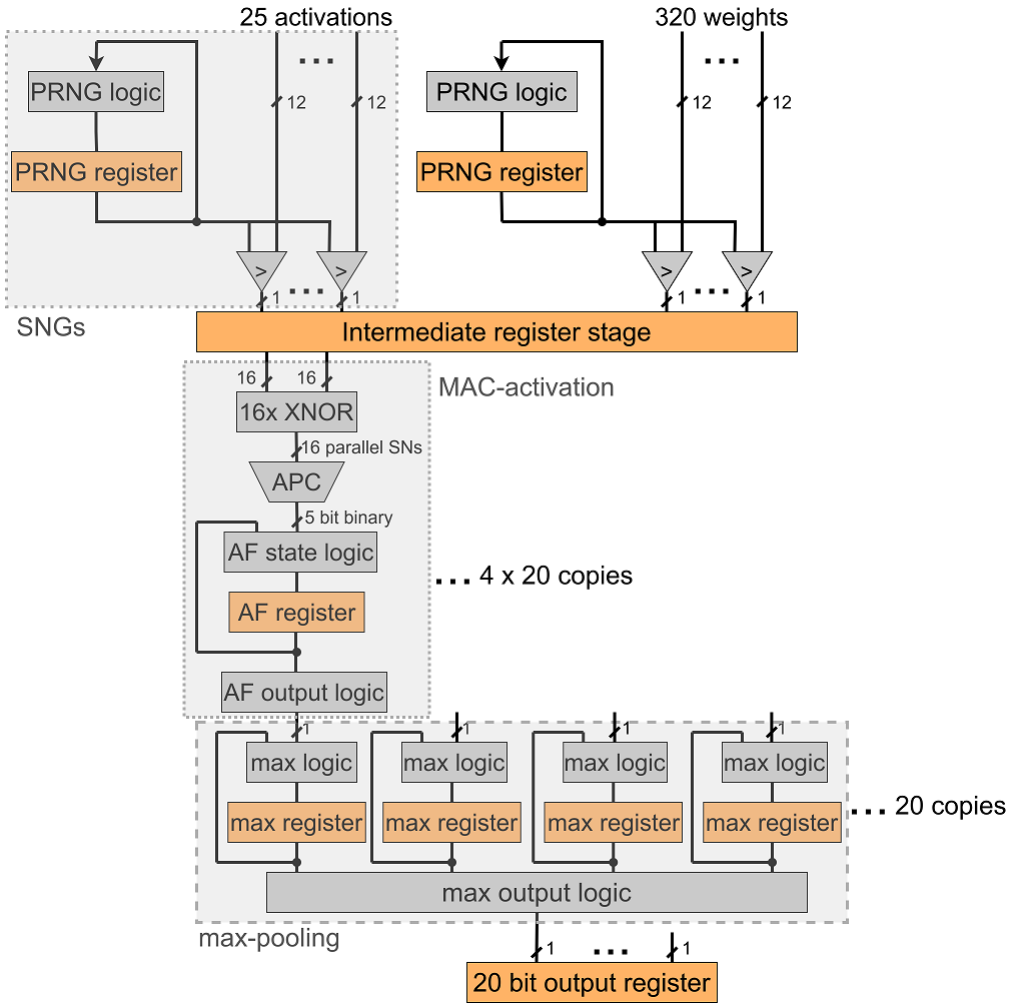

Figure 4.2: RTL schematic of the SCNN layer. Combinational elements in grey, sequential elements in orange.

a 5×5 pixel sized area of the input that corresponds to the 2×2 pooling window. Figure 4.3 illustrates the resulting parallelism within each kernel with parallel MAC-operations shown in different colors, with MAC_1 using inputs A_{11} to A_{33}, MAC_2 using inputs A_{12} to A_{34}, MAC_3 using inputs A_{21} to A_{43} and MAC_4 using inputs A_{22} to A_{44}.

All kernels operate on the same section of the input image simultaneously, leading to the total number of 25 input pixel values and 320 weights for the entire implemented SCNN layer. Overall, the circuit therefore contains 345 SNGs with shared PRNGs. The degree to which they can be shared differs between SBoNG and LFSRs. Two 16-bit SBoNG instances, one for input pixels and one for weights, are sufficient for the NN to reach

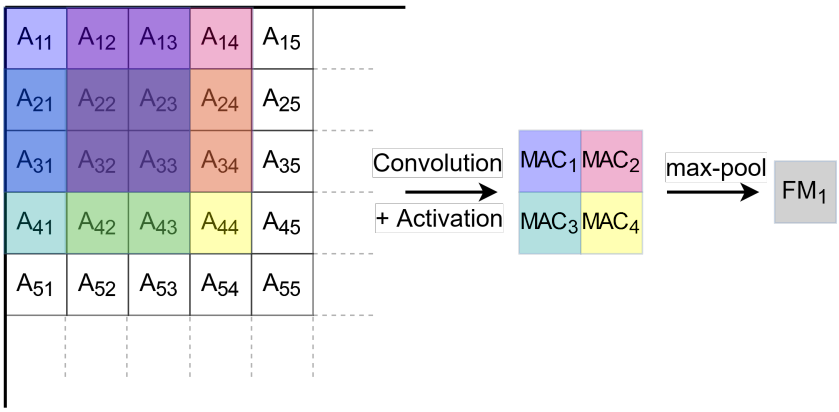

Figure 4.3: Example of parallel operations within an SC component for computation of one kernel. In this example, kernel size is 3×3, max-pooling size is 2×2.

a classification accuracy comparable to the one achieved with 32 LFSRs with different feedback polynomials. 16 LFSRs are used for generating SNs for input pixel values, the remaining ones are used to generate the 16 weights within each kernel. As all kernels operate independently of each other, sharing between kernels does not lead to conflicts for any PRNG. MAC operations in the circuit are computed with 80 parallel MAC-Btanh components, which provide the inputs for the 20 max-pooling circuits.

The design of the circuit computing the binary NN layer follows a different approach. It consists of only a single multiplier and accumulator, which are used sequentially to compute the entire layer. Fig. 4.4 shows the circuit's RTL design.

Two large multiplexers control which input values and weights are routed to the multiplier. The resulting products are accumulated for 16 cycles (to cover one 4×4 sized MAC-operation) and routed through a custom, piecewise linear approximation of the tanh function. The tanh function as given in equation 2.16 requires computation of several exponential functions and a division. These operations are very costly to implement in hardware and would in fact require significantly more area than the entire remaining binary layer. For this reason it is common practice in hardware NN design to use piecewise linear approximations that can be computed using only adders and multipliers [YWT+18]. The approximation pw in the present design can even be computed without a multiplier, as it only involves multiplication with powers of two, which are implemented as shifts:

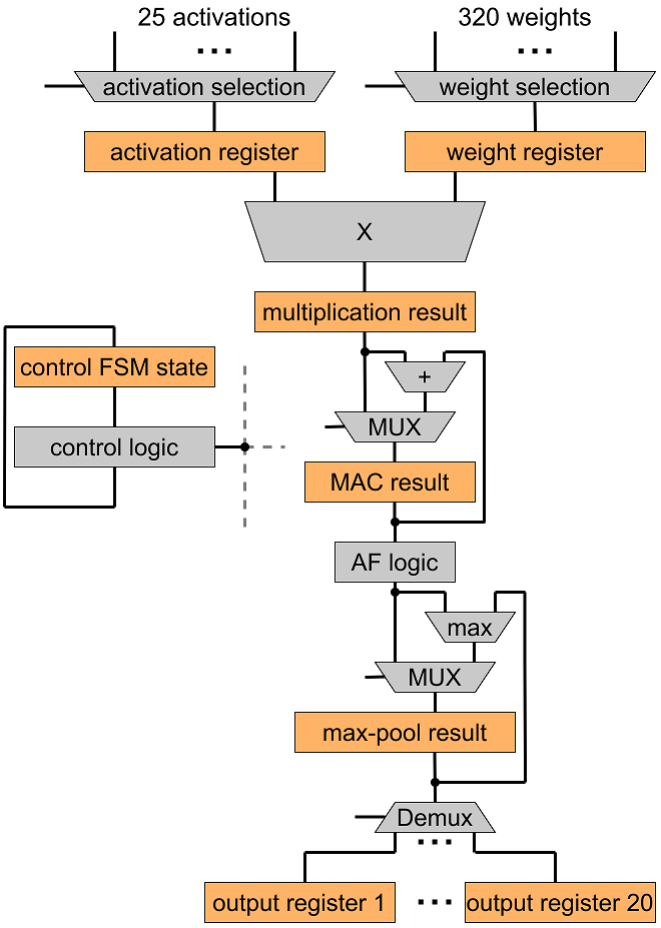

Figure 4.4: RTL schematic of the binary NN layer. Combinational elements in grey, sequential elements in orange.

$$pw(x) = \begin{cases} -1, & \text{if } x < -2 \\ \frac{1}{4} \cdot (x-2), & \text{if } x \geq -2 \text{ and } x < -\frac{11}{16} \\ x, & \text{if } x \geq -\frac{11}{16} \text{ and } x < \frac{11}{16} \\ \frac{1}{4} \cdot (x+2), & \text{if } x \geq \frac{11}{16} \text{ and } x < 2 \\ 1, & \text{if } x \geq 2 \end{cases} \tag{4.2}$$

Boundaries between cases are multiples of $\frac{1}{16}$ and can therefore be distinguished with a comparison of only the four most significant decimal bits, further reducing the complexity of this component. The whole binary circuit is controlled by an FSM that is responsible

for routing the correct weights and input values, and for storing the output of each kernel in its respective register. This FSM is part of the circuit and is therefore also considered in the fault simulation.

Table 4.2 lists the costs of each individual component and the overall cost of the resulting SCNN layer, with the binary layer's cost for comparison. Combinational cost is given as the number of 2-input gates, sequential component cost is the number of flip-flops.

Table 4.2: Component costs (in 2-input gates and flip-flops) of individual components and the resulting SCNN layer.

Component	**Combinational cost**	**Sequential cost**	**Total cost**
SNG (LFSR)	81	12	93
SNG (SBoNG)	211	16	227
MAC-Btanh	335	6	341
NMax	811	48	859
AMax	307	28	335
SCNN layer (SBoNG)	62,860	1,817	64,677
SCNN layer (LFSR)	62,776	2,169	64,945
Binary layer	17,363	103	18,466

Due to the sequential design, the binary circuit has a much lower gate count than the SC version. Roughly 30% of the stochastic circuit's area is consumed by the comparators of the SNGs. Much of this overhead could be saved by opting for a design that likewise computes kernels sequentially, at the cost of significantly longer computation times. For fault simulation, these differences are however irrelevant, as faults in any specific kernel do not affect other kernels. Gate counts in table 4.2 are therefore not a generally applicable comparison between the sizes of SCNNs and binary NNs, as the implementation focused on obtaining comparable delays for the binary and stochastic circuits: The critical path delay is 21.8ns for the binary circuit and 26ns for the stochastic circuit, which is mainly due to the simple approximation pw of the activation function. 12-bit precision is used in both cases, leading to a corresponding SN length of 4096. The results presented in the following were obtained using the fashion MNIST dataset [XRV17] but are not restricted to this dataset (except for specific examples). The NN structure used for evaluation of network accuracies is given in table 3.2.

4.1.3 Gate-level bit flips

In an initial simulation, the robustness of the described circuits against generic gate-level bit flip errors is tested. For this purpose, bit flips are simulated on each line of a circuit with a fixed probability. This is the current standard method to analyse robustness of stochastic circuits and provides a reference to compare against the results of the timing fault simulations. Bit flip probabilities between 10^{-6} and 10^{-2} were simulated; fig. 4.5 shows the resulting degradation in classification accuracy for the binary network (blue), and two versions of the SCNN: an implementation with an NMax-based max-pooling component (orange) and an implementation with AMax-based max-pooling (green). As LFSR and SBoNG-based implementations showed virtually the same behaviour, only graphs for SBoNG are shown for NN accuracy for better visibility.

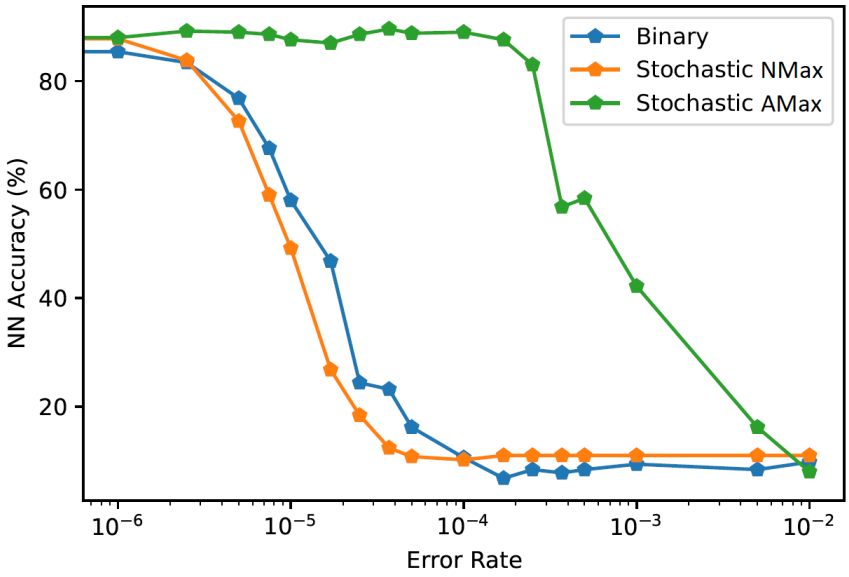

Figure 4.5: Classification accuracy as function of bit error rate for binary (blue), and SC implementations.

The classification accuracy of the binary network starts to drop noticeably even for small error rates of around $2 \cdot 10^{-6}$ and falls of sharply after that point. Such a behaviour is expected, as even a single bit flip can have a high impact on the binary circuit, if it affects a high significance bit.

Surprisingly, the behaviour of SC implementations is highly inconsistent. While the AMax-

based circuit supports the assumption that SC provides an inherent robustness against bit-flips, the NMax-based implementation shows an entirely different picture. Its overall classification accuracy degrades quickly, partially even quicker than the accuracy of the binary implementation. As the maximum circuit is the only component that differs between these implementations, the reason for the significant difference in robustness must lie within these circuits. Individual simulation of these circuits provides more detailed information about their behaviour. In order to visualize the effect of bit flips on the output value distribution of the analysed circuits, scatter plots are used in many of the following figures. In these plots, expected output values are shown on the x-axis and error-affected output values are shown on the y-axis. In an error-free case, the plots would show a narrow distribution of points around a diagonal through the origin, as the output values of an SC component are distributed around the expected values. Point clouds that cover a wider area signify higher output variance, while clouds that are not centred around the origin signify a biased output distribution. Figs. 4.6 and 4.7 show such plots for AMax and NMax for error probabilities of 0, 10^{-5}, 10^{-4} and 10^{-3}.

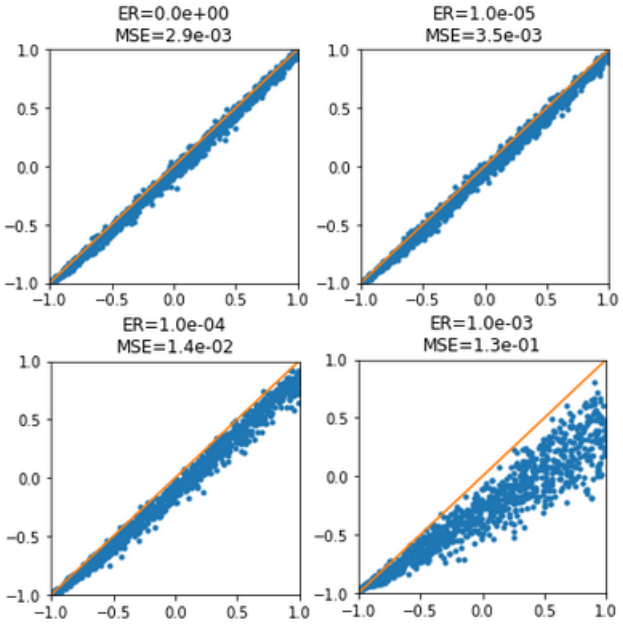

Figure 4.6: AMax output bias under gate-level errors.

It is apparent that AMax is a significantly more robust component with regard to gate-level errors. While it does show a small output bias in the error-free case (see section 3.1 for an

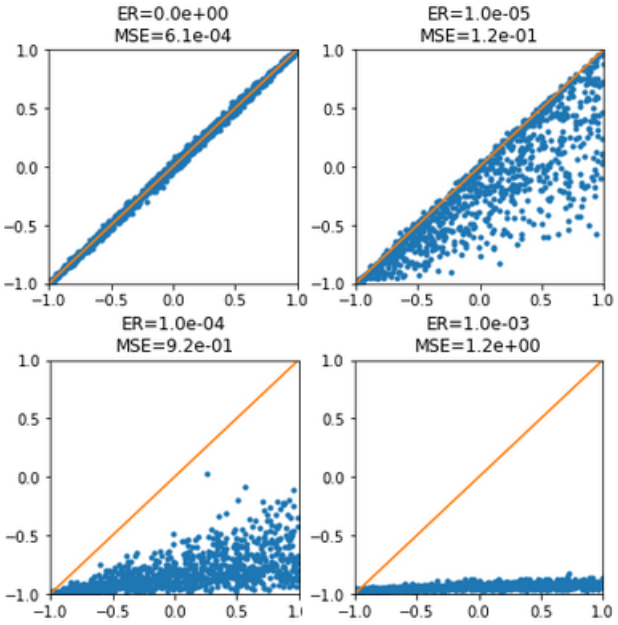

Figure 4.7: NMax output bias under gate-level errors.

explanation), it tolerates error rates up to 10^{-4} with only a small increase in output bias and variance. NMax on the other hand is strongly affected already at error rates of 10^{-5}. It shows a massive increase in output variance and bias, such that it cannot reliably perform its intended function of max-pooling any more. This effect increases with the error rate, such that it outputs almost a constant -1 at an error rate of 10^{-3}. The reason for these different behaviours lies in the circuit structures. Both are counter-based circuits that are essentially multiplexers. In any clock cycle i, the input with the maximum value up to cycle i is routed through to the circuit's output. The main difference between the circuits lies in the counter logic: AMax periodically resets its counters to update the routing after every c clock cycles (in the simulation above $c = 32$). NMax on the other hand relies on accurately tracking the differences between its inputs for correct routing and can therefore not reset its counters during computation. Any bit flip in AMax can thus only affect at most c output bits, before the circuit recovers from the error due to the counter reset. In NMax on the other hand, a bit flip can potentially affect all remaining cycles of the computation, as illustrated by the example in table 4.3.

As explained in section 3.1, at least one counter of NMax is 0 in any clock cycle by design. When this circuit state is violated due to an error, the circuit outputs constant 0s, until a

Table 4.3: Example of an NMax computation without and with error (in cycle 5) over 8 clock cycles with bipolar input SNs $X_1 = 0.5$, $X_2 = -0.75$ and $X_3 = -0.5$.

Clock cycle:	0	1	2	3	4	5	6	7
X_1	0	1	0	1	1	1	1	1
X_2	0	1	0	0	0	0	0	0
X_3	1	0	1	0	0	0	0	0
c_1	0	1	0	1	0	0/4	0/3	0/2
c_2	0	1	0	1	1	2	3	4
c_3	0	0	0	0	0	1	2	3
Z	1	0	1	0	1	1/0	1/0	1/0

valid circuit state is reached again. The strong negative bias of NMax in fig. 4.7 is caused by this behaviour. Depending on the timing and position of the error, and the specific input SNs, recovery can take any number between one and n clock cycles, leading to the large variance of output values in fig. 4.7.

While this simulation is just one example with specific circuits, it already suggests that in general, the error tolerance of their underlying number format does not always transfer to stochastic circuits. It is therefore not justified to assume that SC as a whole is error tolerant, rather any specific implementation should be evaluated on its own. This example does however demonstrate that a main characteristic of a stochastic circuit that can influence a given design's error tolerance is its sequential depth. A circuit that is guaranteed to recover from an error within a short time frame, e.g. AMax, will in general be less affected by an error than a circuit that carries the error's effects of this error through the whole remaining computations (e.g. NMax).

However, a large sequential depth does not necessarily lead to low error tolerance in the case of SC. Consider for example an LFSR-based SNG generating generating an SN A with expected value a with length n. Its sequential depth is n, as an error that changes the LFSR's state in the first clock cycle causes this state to differ from the error-free state for the entire remaining computation. The SNG however will in most cases still generate A with value a, as the sequence of pseudo-random numbers that the LFSR produces will be the same as in the error-free case, only shifted. The only exception is an error that puts the SNG into the all-0 state, from which it cannot recover. Such a case is unlikely, as it requires possibly multiple simultaneous bit flips at very specific positions. Fig. 4.8 shows

simulation results for an LFSR-based SNG under gate-level errors. Error rates in all following figures in this subsection are colour coded with the following probabilities: green for 0.01%, blue for 0.1% and red for 1%.

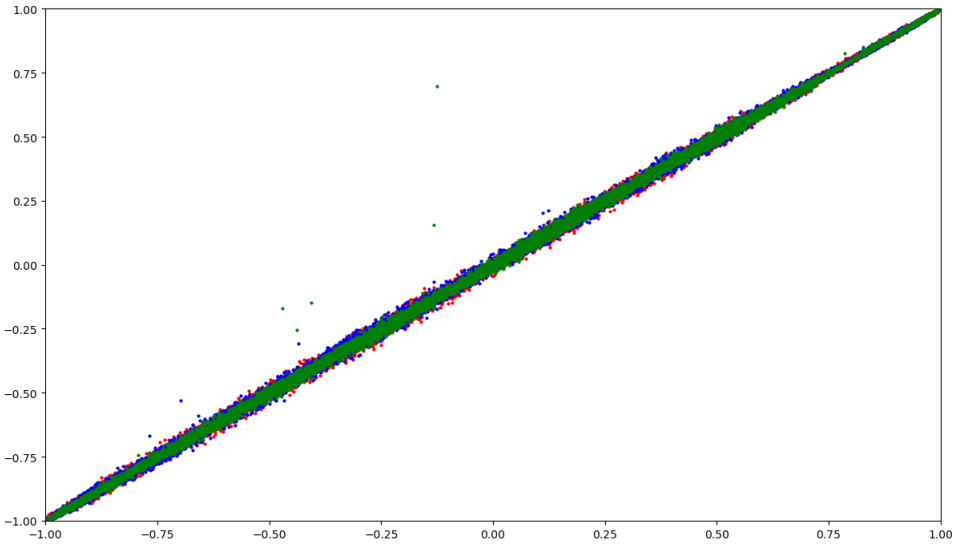

Figure 4.8: SNG output bias under gate-level errors.

The graph shows clearly that for all error rates almost all points are distributed narrowly around the line through the origin, i.e. the component produces the expected results with low variance. A few outliers are visible above this line, which are caused by errors that put the LFSR into the all-0 state. In this state, the SNG outputs only 1s, causing the output values to be larger than expected. Depending on the timing of the error and if the circuit recovers from this state due to another error, the outliers distance to the line varies. For the simulation with 1% error rate (red data points) no such outliers are visible, as the error rate is high enough such that the LFSR does not remain in the all-0 state long enough to affect the output value significantly. The key observation of Fig. 4.8 is that sequential depth alone is not sufficient to gauge the robustness of a stochastic circuit. The characteristic error distribution of the circuit, i.e. variance and bias of error-affected outputs, have to be considered as well and, more importantly, evaluated on a case-by-case basis.

The remaining major component of the SCNN layer, the MAC-activation block, exhibits a mix of characteristics from both components examined above. On the one hand, it has

sequential depth n, as the state of the FSM of the activation function does not reset during the processing of a single SN, and can stay in an erroneous state for the whole computation. On the other hand, its FSM is implemented as a saturating up-down counter, which can cause it to recover from errors in cases where the maximum or minimum counter value is reached and held for several clock cycles. For example, a counter that saturates at a value of 8 will recover from any error after counting "up" for eight consecutive clock cycles, as it will reach the maximum value from any erroneous or error-free starting state. Simulation results for gate-level errors of the MAC-activation component are shown in fig. 4.9.

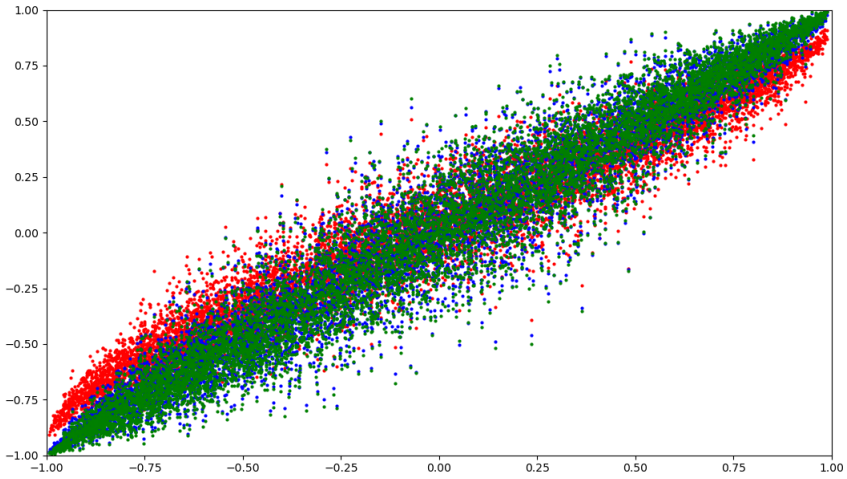

Figure 4.9: MAC-activation block output bias under gate-level errors.

A small bias under higher error rates (red data points) is clearly visible in fig. 4.9. It increases with the absolute of the expected output value, and always has the opposite sign, i.e. the bias is positive if the expected output value is negative and vice-versa. The FSMs of the various stochastic hyperbolic tangent implementations are split in a lower half with states S_0 to $S_{\frac{N}{2}-1}$ and an upper half with states $S_{\frac{N}{2}}$ to S_{N-1}, with N being the total number of states (cf. fig. 2.5). When the FSM is in a state of the lower half, it will output a 0, when it is in a state of the upper half, it will output a 1. An expected output with a large absolute value means that the FSM is either in the lower half or the upper half for the vast majority of the computation. An error that corrupts the state such that it enters the other half of the FSM will therefore cause a bias with the opposite sign of

the expected output value. This bias acts effectively as a downscaling factor for the whole MAC-activation component as shown in fig. 4.10.

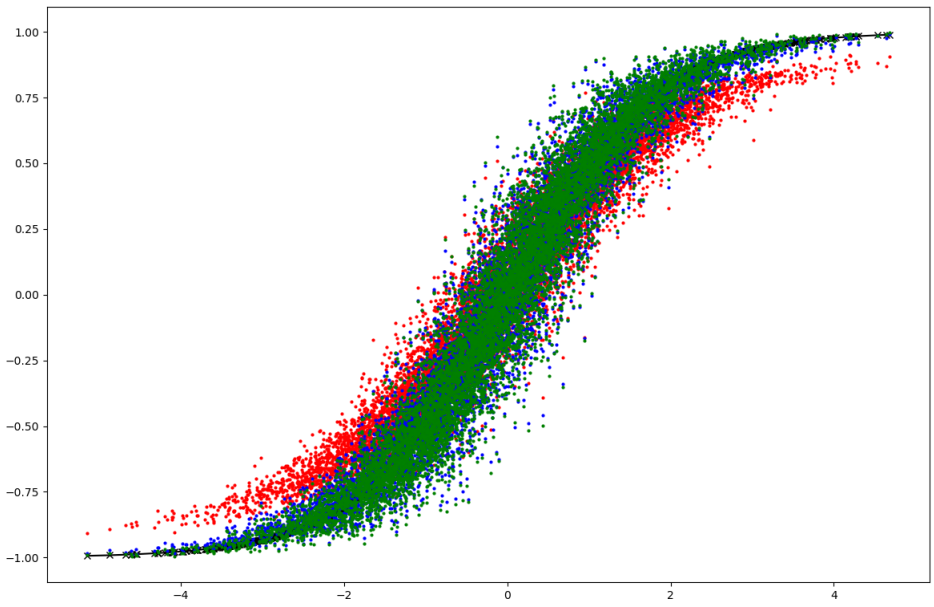

Figure 4.10: MAC-activation block downscaling under gate-level errors.

It follows that gate-level errors change the functionality of the MAC-activation component systematically, and in effect lead to a modification of the activation layer compared to the initial NN model that the SCNN is based on.
In summary, the data from gate-level error simulations shows several important properties of stochastic circuits:

- Not all stochastic circuits retain the high error tolerance of SNs.
- Different stochastic circuits have vastly different behaviours under errors.
- Stochastic circuits often show biased error distributions.
- Some stochastic circuits can lose their functionality entirely even for low error rates.

The assumption that SC provides inherent error tolerance is therefore not generally valid for gate-level errors. Rather, every specific stochastic circuit has its own degree of error tolerance and affects the overall system differently.

4.1.4 Timing error analysis

Timing error analysis is carried out according to the procedure laid out in section 4.1.1. The classification accuracy of the SCNNs and the binary network under timing errors is given in figure 4.11 as a function of the simulated delay factor, starting at 100% (i.e. nominal delay).

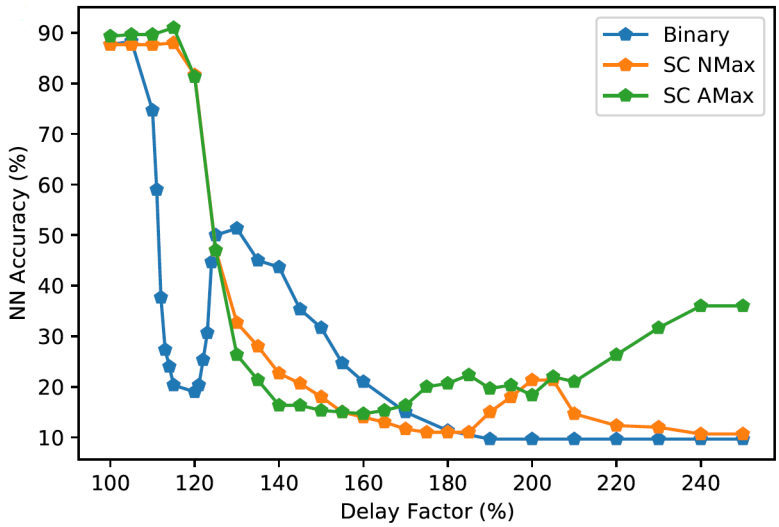

Figure 4.11: Classification accuracy as function of delay factor for binary (blue) and SC implementations.

Several observations can be made in the graph: Firstly, the binary circuit shows an immediate sharp drop in classification accuracy for delays of 110% of the nominal delay and recovers again starting at 120%. This local minimum is due to a glitch in the multiplier stage of the design (cf. figure 4.4), which is heavily biased towards outputting 1s as the result of an error for these delays. For other delays, faulty outputs of this component are roughly equally distributed between 1s and 0s. Secondly, both stochastic circuits perform very similar for delays up to 160%, a significant difference to the behaviour of the AMax-based SCNN under gate-level errors (cf. figure 4.5). Thirdly and most importantly, none of the SCNN implementations shows any significant tolerance towards timing errors above a delay factor of 1.2, but they show no drop in accuracy up to this factor.

Similarly to the results of the gate-level error simulation, both versions of the SCNN show different behaviour, though in the case of timing errors, these differences only appear at higher delays above 160%. Therefore, two questions have to be answered:

- What is the reason for the equally low error tolerance of both SCNN version even though they differ in components and have previously been shown to behave differently under a different error model?
- What is the reason for the fluctuating classification accuracies of the SCNNs for high ($\geq 160\%$) delays?

As a first step towards answering these questions, the dependency between MSE and classification accuracy of the simulated circuits is analysed. Figure 4.12 shows the resulting graph. Each data point in the graph corresponds to a simulated delay in figure 4.11, with each data set approximated by an exponential curve.

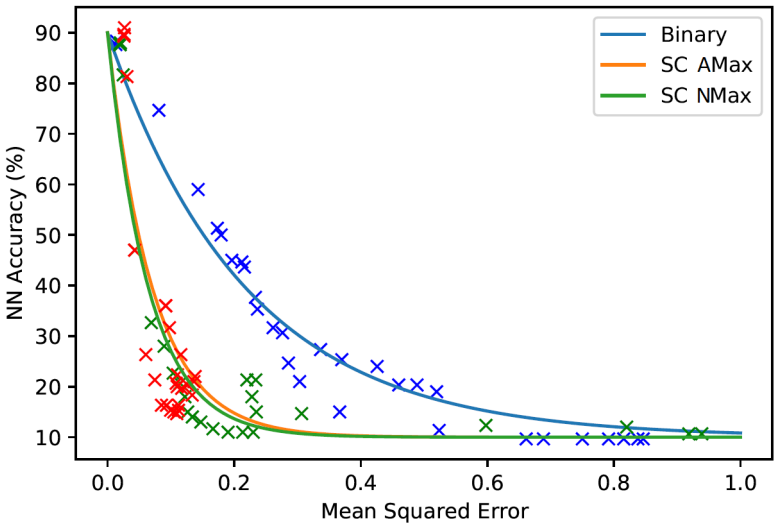

Figure 4.12: Classification accuracy as function of delay factor for binary (blue) and SC implementations.

The data in figure 4.12 shows that even though the MSE of the SCNNs is lower than that of the binary NN for most delays, their classification accuracy is lower as well. This suggests that the cause of this low accuracy is due to biased error distributions. In this regard, timing errors in stochastic circuits would behave similarly to gate-level errors in that they

mainly cause small, but biased absolute errors. Furthermore, one difference between the NMax-based and the AMax-based SCNNs becomes apparent: The AMax-based implementations have a much lower MSE than the NMax-based implementation for most delays above 160%, which leads to the differences in accuracy between these two implementations visible in 4.11.

In order to further determine the reasons for these observed phenomena, SC components are analysed individually, as previously done in section 4.1.3. To achieve this, data from individual stages is extracted from the overall SCNN simulation and visualized in individual plots for each simulated delay. As before, the expected output value of a component is plotted on the x-axis, the simulated output value on the y-axis. If both match, the graph is a diagonal through the origin. Deviations from this diagonal signify an increased MSE, which increases with a point's distance from the diagonal in y-direction, and bias, which increases with the number of points on one side of the diagonal.

Max-pooling:

Analysis of the stochastic maximum circuits reveals that AMax shows consistently good behaviour for all simulated delays, with small MSE and virtually no bias (figure 4.13). The main reason for its good performance is the small length of its critical path, which is not enough to cause capture errors. Furthermore, its periodic reset of all memory elements allows AMax to recover from errors frequently, as described in section 4.1.3. Due to its approximate nature, the graph for AMax does not match the diagonal exactly.

On the other hand, the graph for NMax (figure 4.14) is a perfectly straight line for delays up to 115% and almost exact for delays up to 150%. This behaviour is expected, as NMax implements the maximum function accurately (cf. section 3.1) and also has a short critical path length. When delay increases however, its performance starts to deteriorate and a clear failure point can be seen for delays of 205% and 210%. The difference to AMax can be attributed to two points: NMax's critical path is longer than AMax's, mostly due to the large OR gate combining all bits of each counter, which was synthesised as a chain of 2-input or gates (cf. figure 3.2). In addition, NMax accumulates errors as it misses any reset mechanism.

MAC-activation: The MAC-activation component turns out to be the most problematic part of the SCNN (figure 4.15). It is affected only slightly by timing errors for delays up to

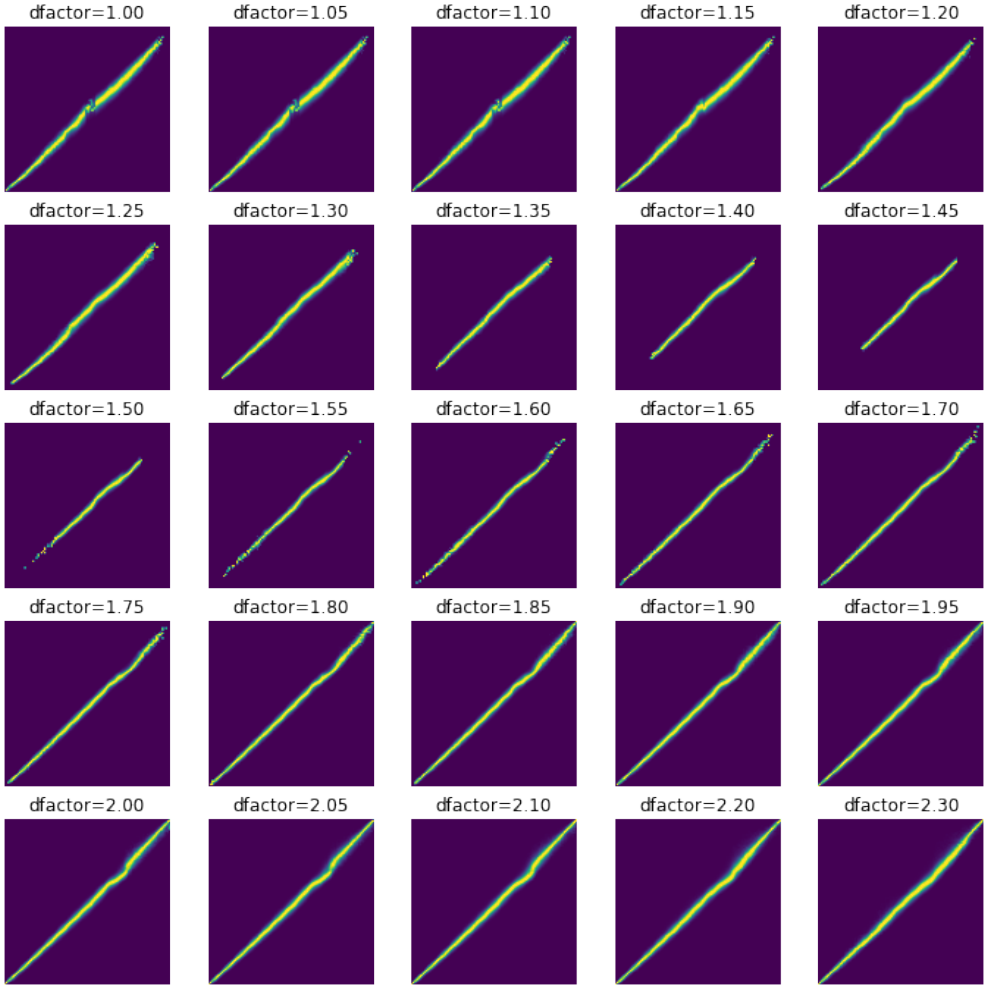

Figure 4.13: Bias and MSE of AMax for varying delays.

120%, but shows quickly deteriorating functionality for longer delays. This deterioration is however not monotonic, as the circuit reaches a maximum MSE and bias at delays around 170% and starts to recover slightly at even higher delays.

In general, the error distribution of this component shows the same trend already observed under gate-level errors, however with an even stronger bias. The distribution also shows a clear prevalence towards a positive bias, which can be inferred from most output values lying above the diagonal. In contrast, figure 4.9 is roughly point symmetric to the origin, having approximately equal halves with positive and negative bias respectively. This suggests that timing errors are not uniformly distributed across all flip-flops and/or lead to asymmetric bit flips, i.e. there are more $0 \rightarrow 1$ than $1 \rightarrow 0$ flips.

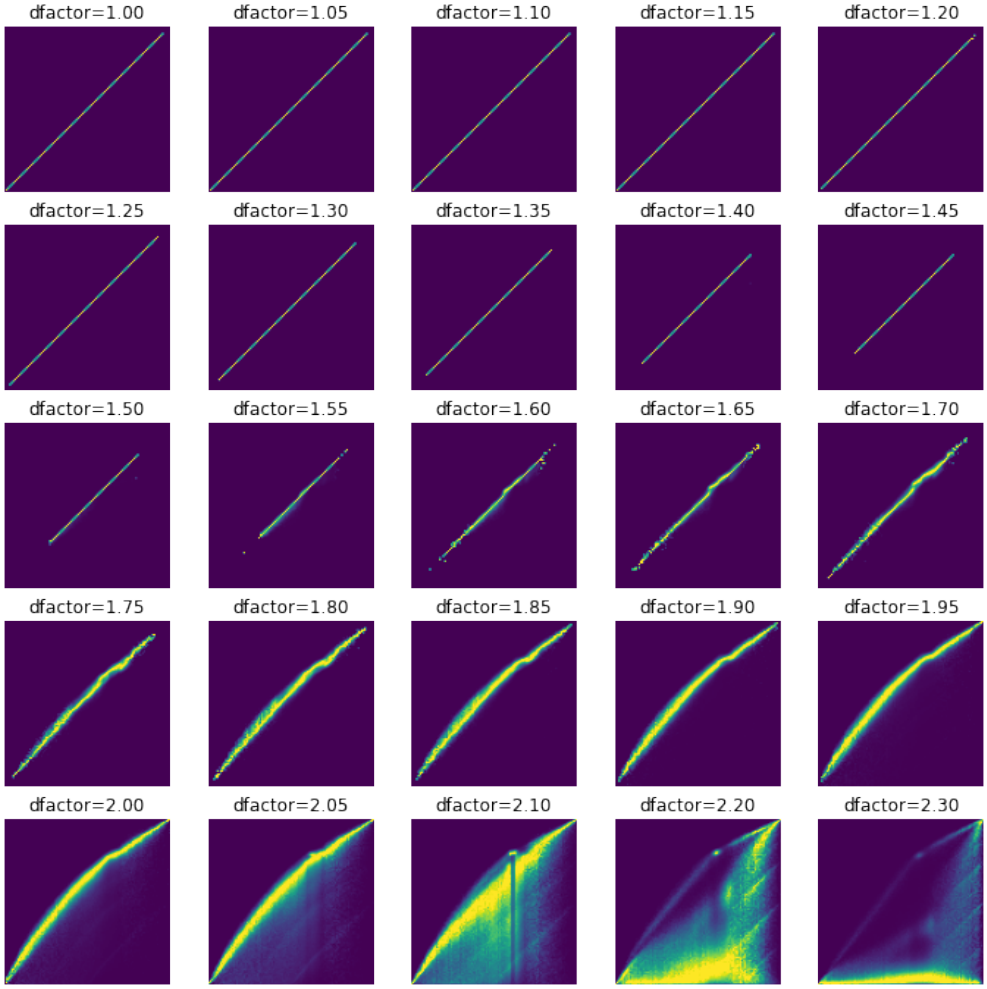

Figure 4.14: Bias and MSE of NMax for varying delays.

SNG: The final individual component to be analysed is the SNG. Timing errors in an SNG can affect both its internal state (i.e. the LFSR/SBoNG register) and the final output after the comparator. As previously mentioned, errors in its internal state have on average no effect on the final output if the PRNG still produces uniformly distributed numbers over the entire $[0, 1]$ interval. On the other hand, errors in the comparator portion will in general change the value of the output SN. Figure 4.16 shows the simulation results of the SBoNG-based SNG under timing errors.

The SNG is almost unaffected by delays up to 160% with only a very minor anomaly for large values (upper right corner of each graph). At delays between 160% and 205%, the SNG's output values concentrate around three values: -1, 0 and 1. The erroneous output

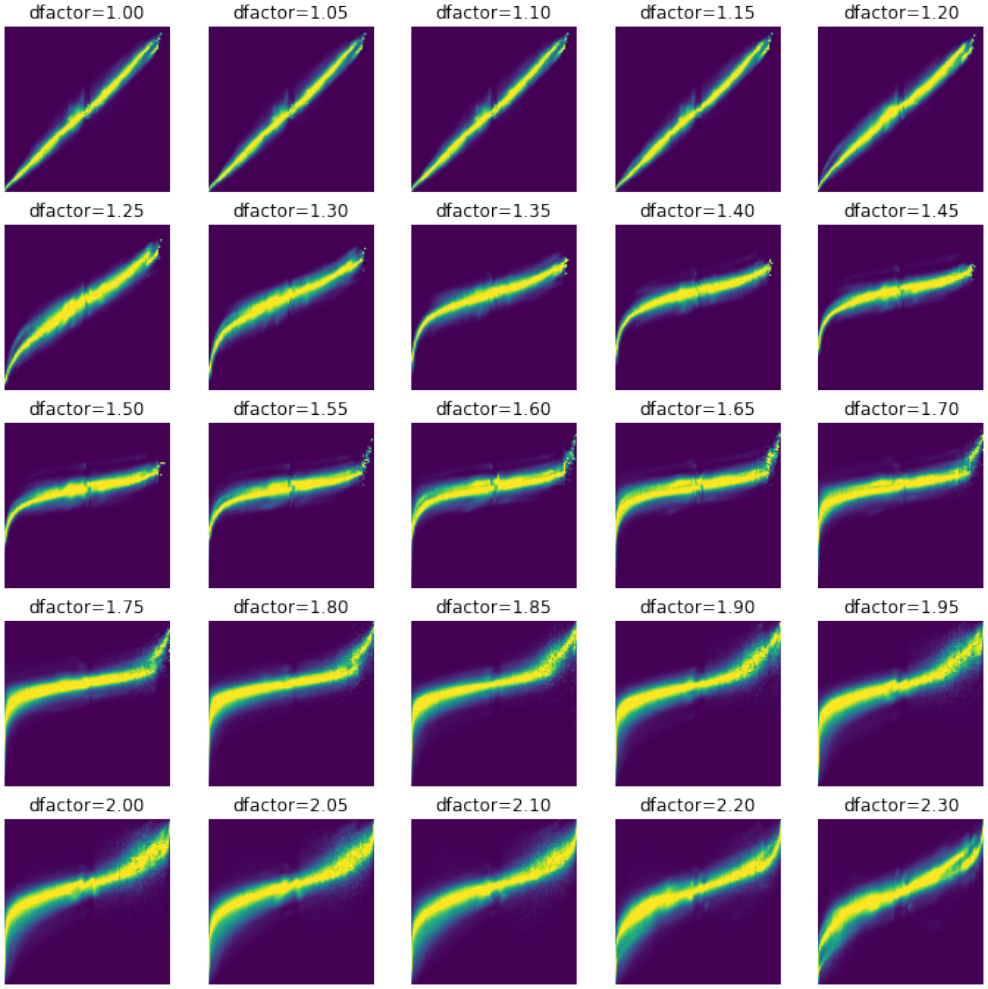

Figure 4.15: Bias and MSE of the MAC-activation component for varying delays.

values mainly tend towards -1 if the expected value was negative, and are mostly split between 0 and 1 for expected positive values. This distinct behaviour must mostly stem from erroneous behaviour of the comparator section, as the PRNG is independent of the SNG's input value, which is equal to its expected output value. An erroneous PRNG would therefore have to show consistent behaviour regardless of this value.

In the simulations of the entire SCNN layer (figures 4.17 and 4.18), the combined influence of all components described above on the layer's output is clearly visible.
Starting at a delay of 125%, the bias of the MAC-activation component becomes the dominating influence on the circuit and is therefore the main reason for the SCNNs' significantly

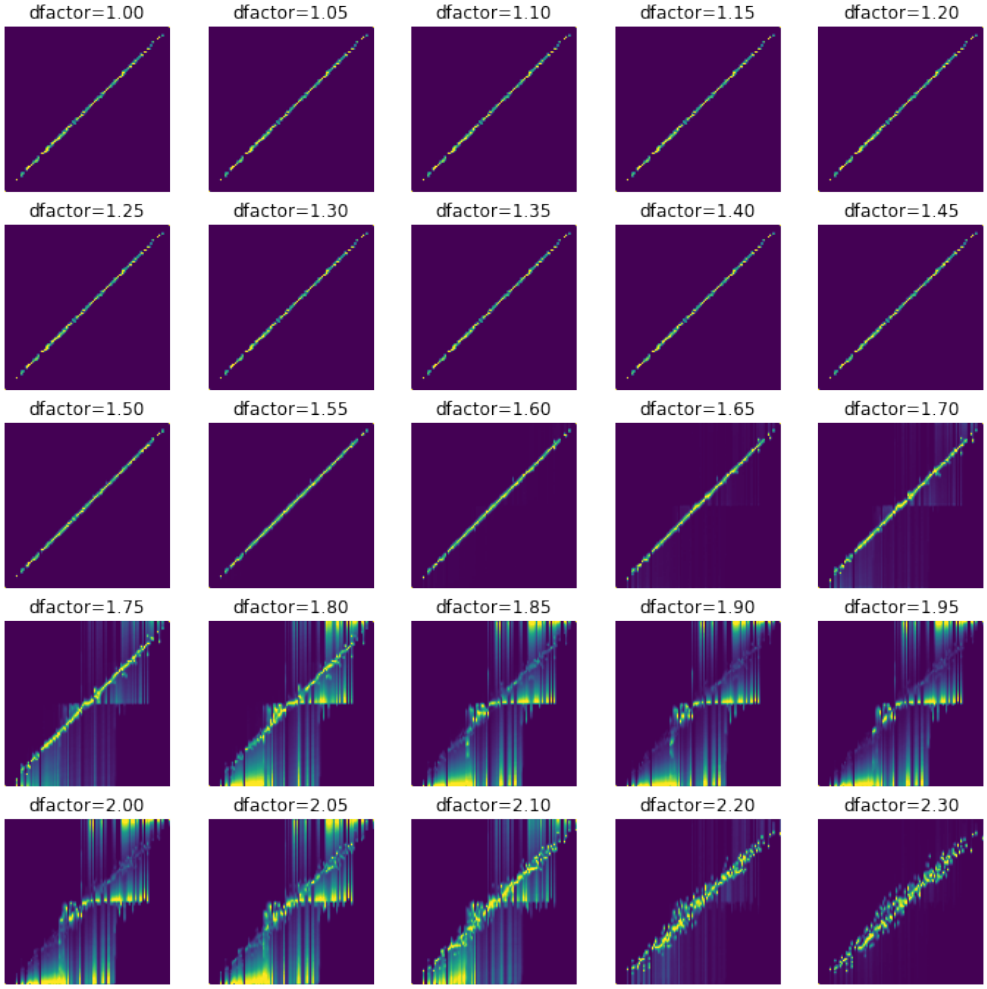

Figure 4.16: Bias and MSE of the SBoNG-based SNG for varying delays.

decreasing accuracy. Both SC implementations have almost identical graphs for delays up to about 170% due to the similar performance of AMax and NMax in this range. As seen in figure 4.11, the implementations diverge after this point. In figures 4.17 and 4.18 this divergence becomes clearly visible at a delay of 175%, where the output of the NMax-based SCNN layer shows a noticeably larger bias (as the graph is shifted upwards compared to the graph of the AMax-based layer). At the same point, timing errors in the SNG start to have an effect on the layer's output, noticeable due to the characteristic vertical streams originating from the main branch of the graph. In the AMax-based implementation, the effects of these two components partially cancel each other out, leading to a reduced bias. This is the reason for the recovering accuracy of the AMax-based SCNN at delays higher

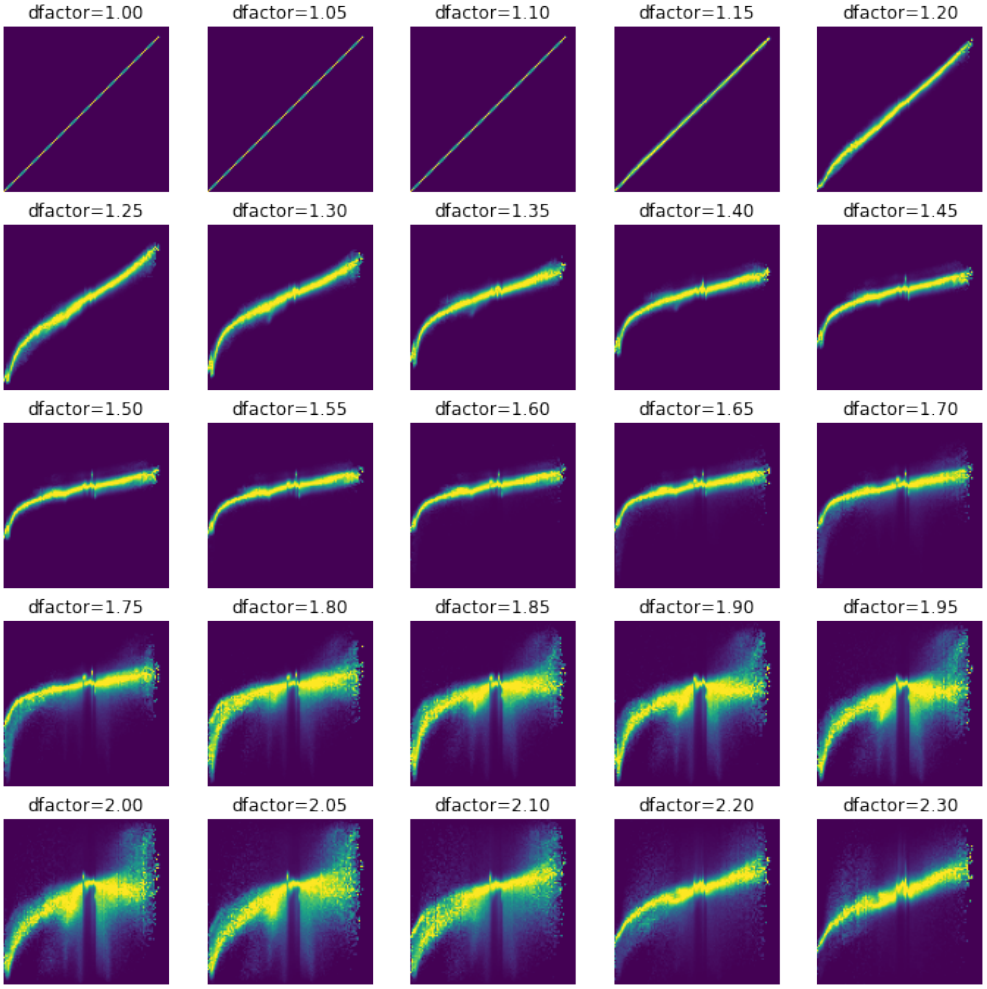

Figure 4.17: Bias and MSE of the AMax-based SCNN layer for varying delays.

than 170% in 4.11. For a small range of delay values, the same behaviour sets in in the NMax-based layer at delays between 190% and 205%. However, at this point the NMax circuit starts to dominate the error characteristic with its strong negative bias, causing a final drop in accuracy for this version of the SCNN.

For comparison, the same graphs have been produced for the binary NN layer (figure 4.19), allowing a direct comparison of the error characteristics of the various implementations.

In general, a difference in the error characteristic of the binary layer compared to the SCNN layers is immediately obvious. The binary implementation has a relatively symmetrical error distribution with one major exception. Starting with a delay of 110%, a thin

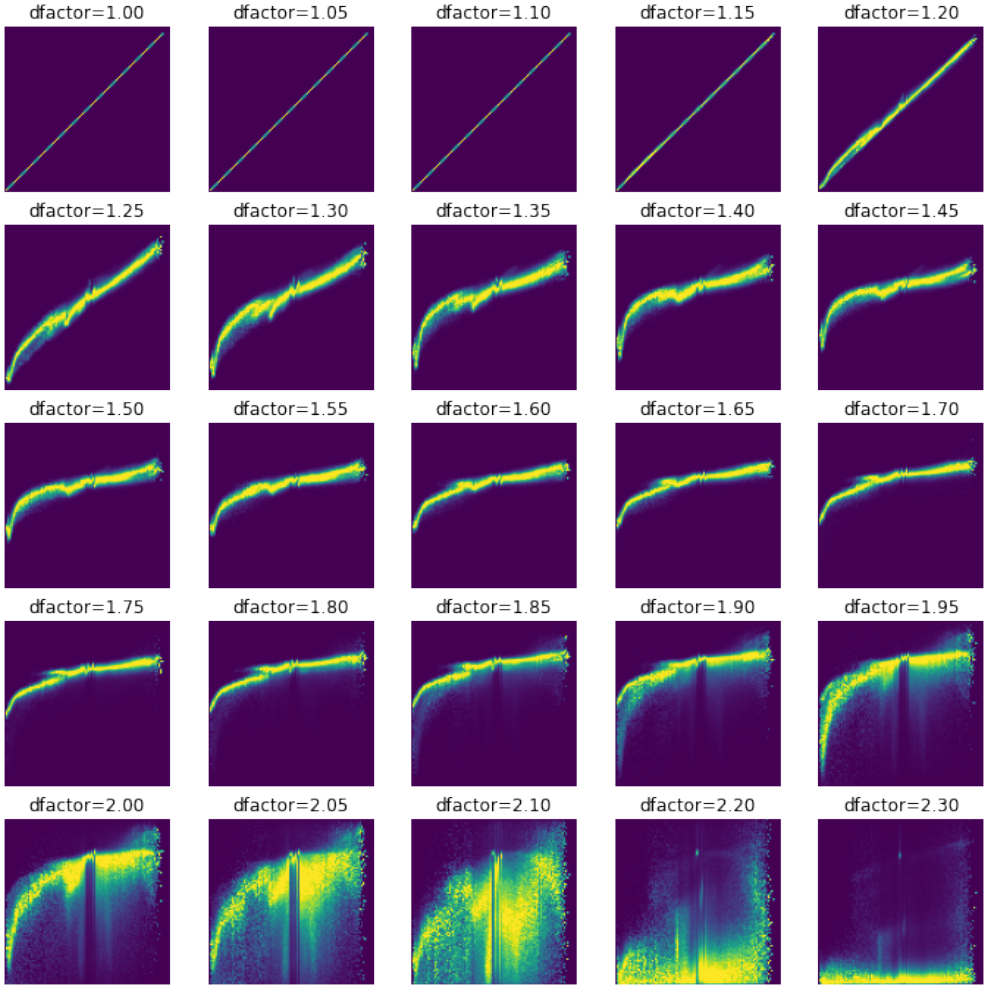

Figure 4.18: Bias and MSE of the NMax-based SCNN layer for varying delays.

line appears at the upper edge of each graph. Initially, these errors are caused by the aforementioned glitch in the multiplier and cause the local minimum in accuracy in figure 4.11. While this specific glitch in the multiplier disappears at a delay of around 125%, the large effects of errors in high significance bits cause a slow but steady concentration of values at the extreme upper and lower edges of the graphs, leading to the similarly steady decline of the binary NN's accuracy.

The very distinct behaviour of stochastic and binary NN implementations also manifests itself clearly in the output feature maps of the layer. Comparing feature maps of both circuits for delays that lead to comparable classification accuracy gives an impression of the characteristics of the error distributions. Figure 4.22 shows feature maps of the same

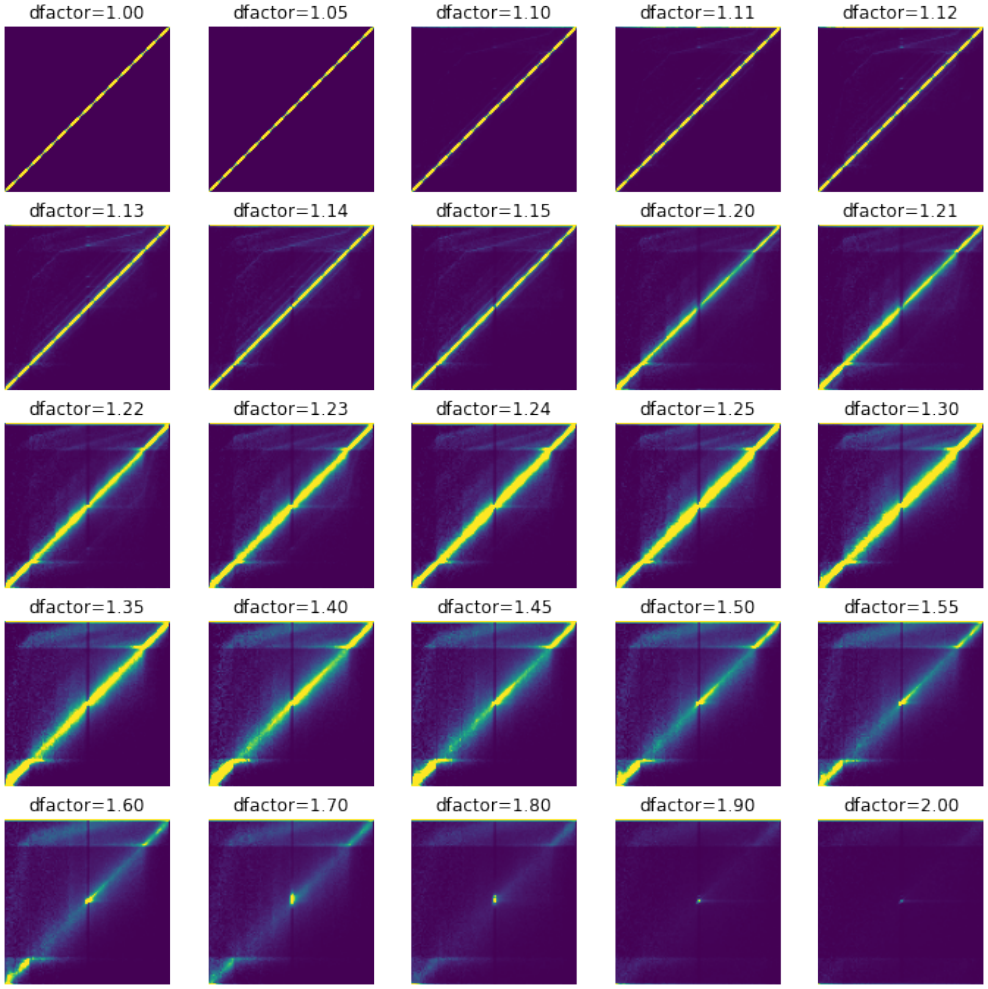

Figure 4.19: Bias and MSE of the NMax-based SCNN layer for varying delays.

kernel produced by the binary NN (a) and the AMax-based SCNN (b) for delays of 150% and 125% respectively (error distribution sub-graphs are extracted from previous figures) and the fault free reference output (c). The classification accuracies of both networks lie between 46% and 48% for these delays (see figure 4.11).

The feature map of the binary network shows a large MSE, with widely fluctuating pixel values. The object's outline is however still clearly visible. The stochastic circuit on the other hand blurs image and background together, as the biased error distribution has a downscaling effect. Even though both networks produce unreliable results to the same degree at these delays, the reasons for their failure are different. High MSE is the main cause for the failure of the binary network (except for the glitch in the multiplier), while

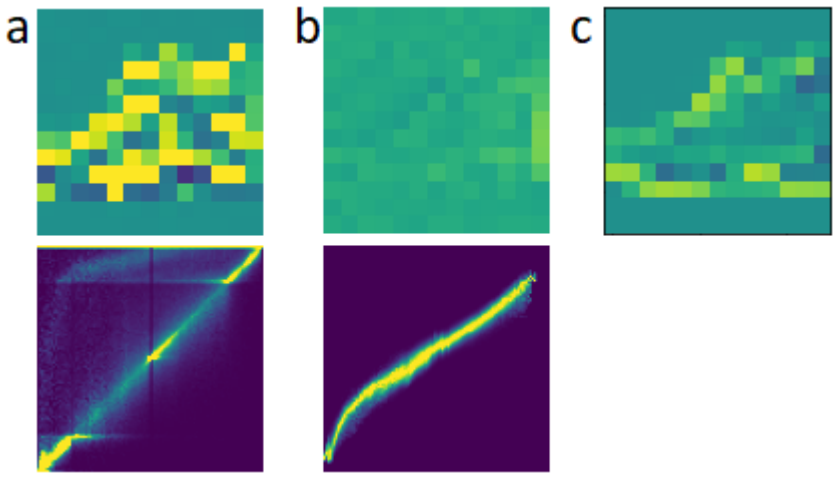

Figure 4.20: Output feature maps of binary circuit (a) with delay factor 150%, and stochastic circuit (b) with delay factor 125%. Brightness of pixels in the feature map corresponds to pixel values (yellow = 1, black = −1). (c) shows the error free feature map.

the stochastic networks fail due to high and consistent bias. This is quite different from the behaviour that would naively be expected from SC, as individual SNs do not have an inherent tendency towards biased errors.

4.1.5 Summary of SC's error resilience

In summary, the assumption that stochastic circuits have a higher error tolerance than binary circuits could not be confirmed for SCNNs in general. While errors in stochastic circuits do indeed result in a smaller MSE due to the error tolerance of individual SNs, this does not always translate to a better error tolerance of the overall SC system. For fully combinational stochastic circuits (with exception of the SNG), high error tolerance has been demonstrated before on the example of SC image processing systems [ALH13] [LL11]. However, the simulation results given here show that sequential stochastic circuits behave quite differently. Their sequential elements remove the independence between subsequent bits of SNs and thus allow errors to accumulate and spread throughout the computation. Error distributions of sequential stochastic circuits differ significantly from those of binary circuits, as errors tend to be smaller in magnitude, but are often biased instead. Biased error distributions not only change a circuit's output accuracy, but also

its functionality and can therefore have an equal or even larger detrimental effect than uniformly distributed errors of higher magnitude. According to the simulation results, an SC system's behaviour under errors depends mainly on the following properties of its components:

- Characteristic error distribution
- Sequential depth
- Error model

The first property depends on a component's implemented function and its implementation details. For example, the activation function implements a point symmetrical function using an FSM with a symmetrical structure and its characteristic error distribution therefore shows a point symmetrical bias. The second property describes loosely the maximum recovery time of a component. AMax and NMax implement the same functionality, but due to its much shorter sequential depth, AMax recovers from errors much faster and shows less bias in its error distribution. The first two properties are linked as the example of the maximum circuits shows, but do not have a clear correlation. As the simulation of gate-level errors in SNGs (figure 4.8) shows, it is possible for components with maximum (i.e. SN length) sequential depth to have unbiased error distributions with low MSE, while other such components, e.g. NMax, have highly biased error distributions. Lastly, SC systems can show highly different behaviour depending on the error model that the simulation is based on. Most notably in the presented simulation results is the performance of the AMax-based SCNN layer, which greatly outperforms both the NMax-based SCNN layer and the binary layer under gate-level errors, but has almost identical performance under timing errors.

It is vital in the design of SC systems for error-prone environments in general and low-power environments in particular to consider these properties. All components of the system should be simulated individually and in combination using accurate error models for the intended environment. Depending on the outcomes of these test simulations, a designer can then adapt both the specific hardware designs as well as application details to improve error tolerance if necessary. Possible design improvements include a reset mechanism, which reduces dependency between subsequent output bits and enables periodic

recovery, and PRNGs without zero-state such as SBoNG. A further improvement for FSM-based SC components was presented in [IMII19] and is based on encoding FSM-states as SNs instead of binary numbers.

On the application side, a designer should consider avoiding functions that result in SC implementations with heavily biased error distributions if the application allows for alternatives. Neural networks are flexible with regard to several functions and parameters such as layer and kernel sizes and activation functions. For example, the hyperbolic tangent function that is commonly used in SCNNs today can be replaced by a clipped ReLU function, which can be implemented using the relatively error tolerant AMax component. While such modifications to an underlying application can lower its accuracy in an error-free environment, it can improve its performance under errors significantly. The findings made in this section enable a more systematic and thorough design of stochastic circuits for low-power, error-prone environments, which are central target domains for SC systems.

4.2 Adversarial attacks on SCNNs

The increasing prevalence of NNs in safety critical systems such as self-driving cars has made them an attractive target for attacks. Such attacks aim to cause networks to misclassify a given input by making carefully crafted, small changes called perturbations to it that are imperceptible to humans. These so-called adversarial attacks were first demonstrated in 2013 [SZS+13] and have since diversified into a large number of different attack algorithms. As a consequence, a variety of defence mechanisms have been developed in turn, covering an equally wide range of protection algorithms. Among the proposed defences are training algorithms that incorporate previously generated adversarial examples [GSS14] [NKM17], input transformations such as rescaling and compression [GRCVDM17] and defences based on randomness such as random neuron dropout [DAL+18] and the addition of randomization layers into the network [XWZ+17]. The inherent randomness in SC and its successful implementation of NNs make it a possible candidate for low-cost protection against adversarial attacks, and early results have indeed demonstrated that SCNNs provide a natural robustness against such attacks [TH19]. The

authors replace the last fully connected layer of their network with an SC implementation and generate adversarial examples using the zeroth-order optimization attack [CZS+17]. Their results show that the attacks success rate drops from 76% without SC layer to 59% with an SC layer present. Their experiments demonstrate that SC can increase the robustness of NNs against adversarial attacks, however only a black box attack scenario is considered. A white box attack scenario is not investigated, even though white box attacks are generally considered to be more powerful due to their detailed knowledge of an NN's structure and parameters. Furthermore, replacing the last layer of the NN with an SC implementation possibly limits its defensive capabilities, as undefended layers prior to it may have already magnified the perturbations to an unrecoverable degree. In this section, adversarial attacks on SCNNs are investigated further by including white box attacks in the form of the C&W attack [CW17] and it will be demonstrated that a first layer SC implementation provides a higher robustness against adversarial attacks. It will further be shown that such an SCNN can even prevent the generation of adversarial examples for some black-box attacks entirely using the example of boundary attack [BRB17]. This section is partially based on [Vek20], where the C&W attack has been performed on an SCNN.

4.2.1 Attack algorithms and attack scenarios

The general goal of any adversarial attack can formally be described as follows: For an input x, a classifier function c of the target NN and a distance metric d, an adversarial attack tries to find a modified input x' such that $c(x') \neq c(x)$ and $d(x', x)$ is small. If $c(x')$ can be any class other than $c(x)$, the attack is called untargeted, if $c(x')$ has to evaluate to a specific target class, the attack is called targeted. The latter are generally harder to perform, as they restrict the solution space for generating x'. The distance metric d can vary, but almost all attack algorithms use one of three different L_p norms, specifically the L_0, L_2 and L_∞ norms. The L_p norm is defined as

$$\|x' - x\|_p = \left(\sum_{i=1}^{m} |x'_i - x_i|^p \right)^{\frac{1}{p}} \tag{4.3}$$

for an m-dimensional input, e.g. a grey scale image with m pixels.
With regard to an image classification task, the L_0 norm corresponds to the number of differing pixels between x' and x. Specific pixel values are of no importance to this metric, a change by 1% or 100% results in the same L_0 norm. An example for an attack using this norm is the one pixel attack [SVS19], which places the restriction $d(x', x) = \|x' - x\|_0 = 1$ on its generated adversarial examples. The L_2 norm corresponds to the standard Euclidean norm in an m-dimensional space. With regard to images, it measures the average change in all modified pixels. The L_∞ norm measures the maximum change in any pair of pixels x'_i and x_i and can be interpreted as the opposite of the L_0 norm. The number of modified pixels has no influence on the L_∞ norm, only the pixel that was modified the most matters. Some attacks, like the one pixel attack, are tailored to a specific norm, while others, like the C&W attack, have variants for all norms.
Attacks are commonly split into categories depending on the knowledge about the target NN that the attack algorithm requires: white-box, grey-box and black-box attacks.

- White-box attacks have complete information about the NN architecture, its implementation details and parameters such as weights and biases. Many gradient-based attacks fall into this category.

- Grey-box attacks have partial information about the target NN. A grey-box attack might for example know the architecture of an NN, but lack information about the exact parameter values or implementation details.

- Black-box attacks have no information about the NN. They can only send queries to the target network and receive answers in form of a class label or the output of the final network layer.

White-box attacks are in general the most powerful type of attack, as they have access to all information about the target NN. In many cases they are gradient-based, i.e. they use a back-propagation algorithm as described in section 2.2.5 to identify how neurons are affected by the perturbations to the input image. This information is then used to compute the minimum perturbations to the input that cause the desired misclassification. With regard to SCNNs, this type of attack is however very hard or even impossible to

perform. An SCNN is often a close approximation of an underlying binary model, but not an exact match. Components such as AMax and stanh implement specific arithmetic functions very well, but not exactly. The deviations to their binary counterparts have to be incorporated into the back-propagation algorithm of a white box attack, which leads to major problems:

Firstly, the output values of a sequential stochastic component may not only depend on its input values, but also on the specific order of bits in the input SNs. It is possible for two different SNs X_1 and X_2 with $x_1 = x_2$ to lead to different output values of such a stochastic circuit. For example, $X_1 = 00001111$ and $X_2 = 10101010$ are SNs with identical values, but lead to significantly different output values in stanh (figure 2.5, starting state S4), with stanh$(X_1) = 10000000$ and stanh$(X_2) = 11111111$. It is therefore not possible to perform back-propagation only based on SN values, instead every single bit operation has to be simulated, thus massively increasing the required computation effort.
Secondly, even in the case that an attacker invests this effort, the bit order in any SN in the SCNN is ultimately dependent on the PRNG output of the SNGs, either directly (for all SNs that are generated from binary inputs) or indirectly (for SNs that are output of stochastic components in the NN) and thus on the PRNG's starting state. However, not only can this state easily be changed during runtime of the application, it may also depend on data that was processed before. For example, SBoNG's period is not matched specifically to an SN length. When multiple subsequent SNs are generated, SBoNG's starting state therefore changes from the view of any individual SN. With regard to SCNNs, this would mean that an attacker would have to know how many inputs were processed before the targeted input since the last circuit reset.
Thirdly, while the parameter data of an NN may be publicly available and therefore enable white-box attacks, as it is the case for popular networks such as ResNet [HZRS16], details of any specific hardware implementation may not. A white-box attack on an SCNN needs this information as well, as the same functionality can be implemented using different components, e.g. a max-pooling layer with AMax or NMax, and an SNG with varying PRNGs.

Due to these major hindrances for white-box attacks, a grey-box scenario is much more likely for SCNNs. This allows an attacker to use the underlying binary model as a basis on

which perturbations are computed, and then rely on the nearly identical functionality of the SCNN for successful misclassification. For black-box attacks the structure and specific implementation of the internal components do not matter by definition. Any black-box attack can therefore readily be used on SCNNs. Furthermore, SCNNs are intended for specialized, low-cost hardware implementations with limited communication capabilities like sensor nodes, which may have no functionality to retrieve the parameter values needed for a white or grey-box attack. As previously mentioned, the C&W attack will be used to evaluate a white-box, respectively grey-box scenario. As [TH19] used a black-box attack scenario, the performance of boundary attack will also be evaluated in order to compare the effect of a first layer SC implementation to the final layer SC approach of [TH19].

C&W attack: The C&W attack is a targeted attack that uses a gradient-based algorithm to solve the optimization problem described above, i.e. minimize the distance between the adversarial example x' and the original image x, such that x' evaluated to the target class t. In [CW17] the formulation is slightly different with $x' = x + \delta$ and is written as:

$$\begin{aligned} \text{minimize} \quad & d(x, x+\delta) \\ \text{such that} \quad & c(x+\delta) = t \\ \text{and} \quad & x + \delta \in [0,1]^m \end{aligned} \tag{4.4}$$

This is a minor rephrasing that includes the constraint that the adversarial example must only consist of valid pixel values (here scaled to $[0, 1]$). The main problem with this initial formulation is the constraint $c(x+\delta) = t$, as the function c is non-linear. The authors solve this problem by rephrasing the optimization problem once more by defining an objective function f such that $c(x + \delta) = t$ if and only if $f(x + \delta) \leq 0$. They list several candidate functions and determine empirically which of them performs best. As a second step, a change of variables is used to make the last constraint compatible with more existing optimization algorithms:

$$\delta_i = \frac{1}{2}(\tanh(w_i) + 1) - x_i \tag{4.5}$$

With these changes made, the authors define their attack algorithms for the three distance metrics described above. As the simulations in this chapter will be based on the L_2 dis-

tance, only the corresponding C&W attack will be given here:
Given an input x and a target class t other than the original class of x, find w that solves the following optimization problem:

$$\text{minimize} \quad \|\frac{1}{2}(\tanh(w)+1) - x\|_2^2 + h \cdot f\left(\frac{1}{2}(\tanh(w)+1)\right) \tag{4.6}$$

where f is the objective function chosen empirically by the authors of C&W attack as

$$f(y) = \max\{\max\{Z(y)_i : i \neq t\} - Z(y)_t, -\kappa\}. \tag{4.7}$$

In equation 4.7, Z is the logits function of the NN, i.e. the output function of the NN excluding the final softmax normalization. h is an adjustable hyperparameter and κ controls the distance of an adversarial example from the NN's decision boundary, i.e. its classification confidence level. The code that the authors provide for their attack [CW] was used to generate adversarial examples for the SCNN.

Boundary attack: Boundary attack was chosen for the black-box scenario because it requires only very little communication with the target network. Only the final classification result, i.e. the class label is required for the attack. This information is always available to an attacker, even if the SCNN is employed in an environment with heavily restricted communication capabilities. The attack algorithm starts with an initial adversarial sample $\tilde{x}^0$ than can be obtained for example by adding random noise to the original input x. Through an iterative process, boundary attack then tries to minimize the distance $d(\tilde{x}^i, x)$ while keeping the samples $\tilde{x}^i$ in the adversarial region during each step. The authors achieve this by following three steps in each iteration:

1. Generate the candidate for the next adversarial sample $\tilde{x}^{i+1} = \tilde{x}^i + \eta^{i+1}$ using an iid Gaussian distribution for η, followed by rescaling and clipping to ensure that the sample consists only of valid pixel values and lies within a given maximum distance from the original image.

2. Project η^{i+1} onto a sphere around x to ensure that $d(x, \tilde{x}^i + \eta^{i+1}) = d(x, \tilde{x}^i)$.

3. Move the sample towards the original image.

In step 2, only the direction in which the new sample moves from the previous one is determined. This allows the algorithm to move a sample along the network's decision boundary between the original class of x and any adversarial class. In step 3, the distance between the new sample and the original image is then reduced. The attack terminates after a set number of iterations have been performed, and the distance of the adversarial sample to the original image is below a set threshold. Boundary attack simulations in this section were performed using the code provided by the authors [BRB].

4.2.2 Evaluation of adversarial attacks on SCNN

For the evaluation of C&W and boundary attack, an NN with a first layer in SC following the design principles laid out in section 2.2.5 was implemented. Remaining layers were implemented in conventional binary format, specific layer types and parameters are given in table 4.4. It has been noted that the first layer in CNNs is more susceptible to attacks than later layers [RHF19], a defensive mechanism in the first layer therefore promises the highest success rate.

Table 4.4: Network structure used in simulations. Kernel sizes vary with the input type (greyscale or RGB).

Layer ID	Layer type	Kernel parameters	Activation function
1	2D Convolution	$3 \times 3 \times 32/3 \times 3 \times 3 \times 64$	tanh
2	2D Convolution	$3 \times 3 \times 32/3 \times 3 \times 3 \times 64$	ReLU
3	Max-pooling	2×2	-
4	2D Convolution	$3 \times 3 \times 64/3 \times 3 \times 3 \times 92$	ReLU
5	2D Convolution	$3 \times 3 \times 64/3 \times 3 \times 3 \times 92$	ReLU
6	Max-pooling	2×2	-
7	2D Convolution	$3 \times 3 \times 128$	ReLU
8	2D Convolution	$3 \times 3 \times 128$	ReLU
9	Max-pooling	2×2	-
10	Fully connected	$20/100$	-
11	Fully connected	10	softmax

The network consists of three blocks of two convolutional layers and one max-pooling layer each, following the NN design from [CW17], where two such blocks were used. Depending on the input data, kernel numbers can vary. For the grey-scale fashion-MNIST data set, less kernels are required than for the coloured CIFAR10 data set. In addition, the

structure of the first layer is slightly different, as the network needs three input channels to handle RGB data and only one for grey-scale. With an SN length of 4096 bits, the SCNN reaches a classification accuracy of 91% for fashion-MNIST and 82% for CIFAR10. The binary version achieves 93% and 88% respectively.

C&W attack results: To evaluate the performance of the attack, 900 adversaries were generated for fashion-MNIST and 450 adversaries for CIFAR10 using the L_2 norm version of the attack. Only classes that were initially classified correctly by the network were considered for the attack and each class was represented equally, with 10% of all initial images and target misclassifications per class. An attack is considered successful, if the network classifies the adversarial example as the target class, and unsuccessful otherwise. Unsuccessful attacks are further split into adversarial examples that were misclassified as some class other than the correct or target class, and examples that were classified correctly. The success rates of the attack on both the binary NN and the SCNN are shown in table 4.5.

Table 4.5: Classification results of CW attack on fashion MNIST and CIFAR10 datasets.

	Binary NN		**SCNN**	
	Inputs	Ratio	Inputs	Ratio
Fashion-MNIST				
Target	734	81.6%	46	5.1%
Correct	42	4.7%	497	55.2%
Other	124	13.8%	357	39.7%
CIFAR10				
Target	441	98.0%	41	9.1%
Correct	1	0.2%	94	20.9%
Other	8	1.8%	315	70.0%

Success rates are very high in case of the binary network with 81.6% for fashion-MNIST and 98% for CIFAR10. The attack is not 100% successful, as sometimes an adversarial example could not be found within the specified L_2 limit and the given number of iterations. Success rate is higher for CIFAR10, as the attack can use a larger search space for RGB images than for grey-scale. The SCNN on the other hand shows a remarkable resilience against the attack. Only 5.1% of attacks on the fashion-MNIST dataset are successful,

while more than half of all adversarial examples are classified correctly. As expected, many inputs end up being misclassified due to the combination of the attack's perturbations and SC's randomness. These cases can be considered a partially successful defence, because the attacker's goal, i.e. a specific target class, was not achieved. Similar results can be observed for the CIFAR10 data set, where only 9.1% of attacks lead to the desired misclassification. On the other hand, most inputs are incorrectly classified overall in this case, making the network very unreliable. However, if the goal is to prevent an attacker's targeted output class, the SCNN can be considered a very good defence. In summary, SC reduces the success rate of the C&W attack by a factor of 16 for the fashion-MNIST data set and a factor of 10.8 for CIFAR10.

Boundary attack results: The execution of Boundary attack takes significantly longer than C&W attack, as it is performed directly on the SCNN and thus has to simulate all bitwise operations during each of its iterations. Initially, an attempt to generate adversarial examples for 100 images of fashion-MNIST was made. After 10,000 iterations however, only 22 of those examples had been found. In the remaining 78 cases, the L_2 distances did not decrease notably at all from the distance of the attack's starting point. For example, the average L_2 distances of iterations 1053 and 1828 were 2.90 and 2.86, but the eventually successful adversarial samples had reached distances in the order of 10^{-3} by that point. This observation fits to an example given by the attack's authors in [BRB17], where the L_2 distances of one sample are $8.0 \cdot 10^{-3}$ at iteration 1053 and $5.6 \cdot 10^{-4}$ at iteration 1828. The distances of the unsuccessful samples on the other hand fluctuated in further iterations, but did not show a gradual decrease.

The reasons for the low success rate of Boundary attack on the SCNN lies in the lack of clearly defined decision boundaries. In a conventional binary NN, boundaries between classes are sharp and can only be moved by changing the network's weights. Boundary attack uses this boundary as a guideline along which it moves its adversarial samples as described above. In an SCNN, this boundary line becomes a boundary space as illustrated in figure 4.21. Within this space, the SCNN does not always place the same input in the same class, but instead only assigns classes around this boundary space with certain probabilities. An input close to the centre of this space has an almost identical probability to be classified as any one of the surrounding classes.

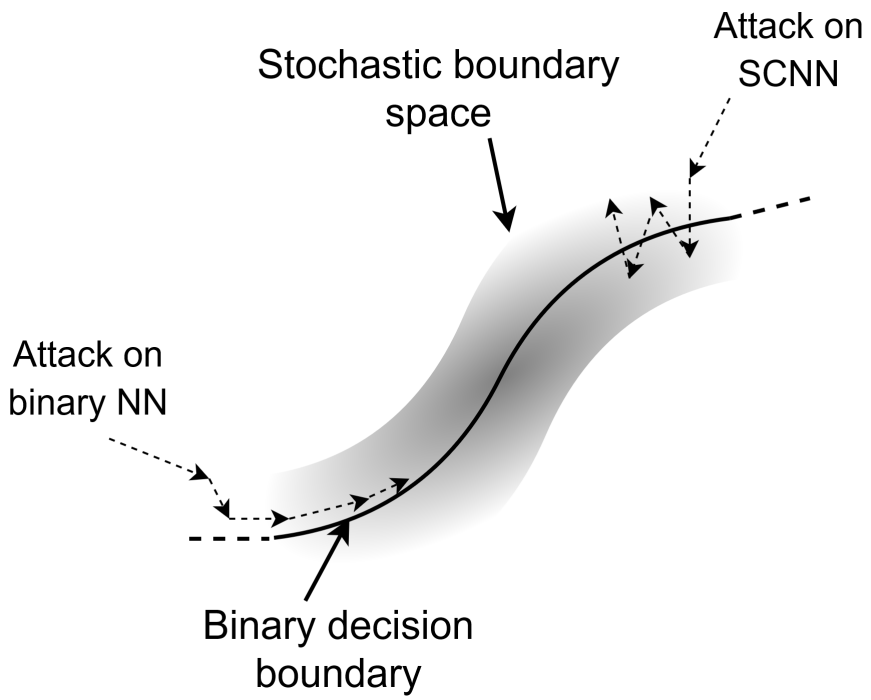

Figure 4.21: Illustration of decision boundary in binary NNs versus decision space in SCNNs.

For example, in a two-class problem there is a boundary next to which a conventional binary classifier would assign an output vector such as $(A = 0.501, B = 0.499)$ and thus assign class A to the input. A very small perturbation in the input, i.e. an adversarial attack, can change the output vector just enough to change the classification consistently to B. An equivalent SC-based classifier would assign class A with a probability of slightly more than 50% and class B with slightly less. The same perturbation in the input would change this probability only a little in favour of class B, but would not lead to a consistent misclassification. Boundary attack relies on this consistency near the decision boundary to generate adversarial examples with low distance to the original input. In SCNNs, this consistency is often only given further away from this boundary, which leads to the large L_2 distances observed in the simulations.

4.2.3 Interference of randomness with adversarial attacks

Inherent randomness in SCNNs interferes with both attacks described in the previous section. The main defensive property of SCNNs is that even two computations with identical inputs and network parameters can lead to different outcomes, if the PRNG starting states don't match. Figure 4.22 shows feature maps of the same kernel of an SCNN for identical inputs and weights, but with different PRNG starting states. The variations caused by these different states are clearly visible, most prominently in the background pixels. Regular patterns in the background caused by RNG sharing can also be seen. It is also notable that the shape of the object is not effected much by the variations. As mentioned in section 2.2.4 equation 2.9, variance in SNs is highest at the centre of their value range. In the case of figure 4.22, background pixels have an expected value of 0, exactly at the centre of the range for bipolar SNs. A convenient side effect of this fact is that SCNNs naturally concentrate variations more in areas of the image that are less useful for classification, while keeping the important features for classification mostly intact.

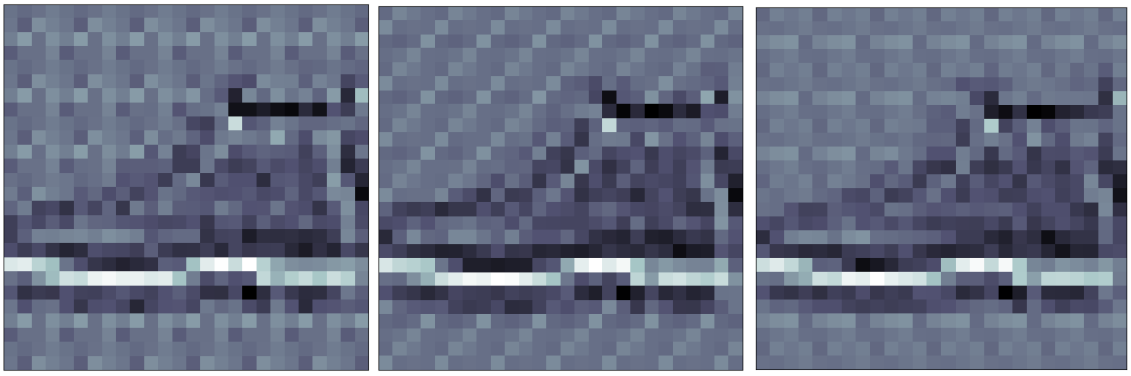

Figure 4.22: Feature maps from the same kernel using identical inputs and weights but different PRNG starting states (fashion-MNIST dataset).

The core process of the described attacks is an iterative algorithm, with small added perturbations in each step, determined by the outcome of the previous iteration. A single iteration can only make small changes, as it may otherwise not find a solution to the underlying optimization problem. However, such small changes in an input can lead to unpredictable changes in the output of SCNNs, as the order of bits in SNs has a much higher influence on the computation result than changes in value of small magnitude. If

an input value changes by 1%, it can affect at most 1% of bits in an SN. A change in the PRNG starting state can potentially affect all bits in an SN and can therefore also have a much higher impact. As the attacks can not always reliably infer the required input perturbations from the result of the previous iteration in this condition, they can fail to converge. In the case C&W attack, the specific reasons are as follows:

C&W attack: White-box attacks chose their perturbations according to the result of the back-propagation step, in which the contribution of each input value to the classification result is computed. They are therefore deliberately chosen in a way to minimize the distance from the original values and still provide the desired misclassification. Due to their small magnitude, the inherent random fluctuations in the SCNN "overwrite" these perturbations however, and the perturbations can not spread throughout the network as determined by the attack. Figure 4.23 illustrates this effect using an image from the CIFAR10 dataset. The original image depicts a cat, but the binary NN classifies it as an airplane after the attack. The SCNN classifies the image correctly.

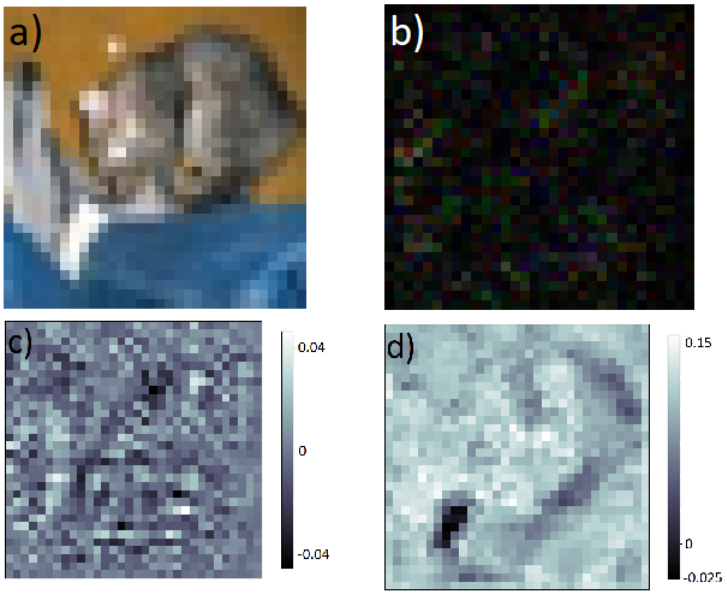

Figure 4.23: Original image (a) and difference to adversarial example (b) (enhanced by factor 10). Differences in feature map 0 between classification of original image and adversary for Binary network (c) and SCNN (d).

While the perturbations in sub-figure 4.23b that cause the misclassification in the binary network might look like random noise, they are specifically crafted by the attack algorithm after multiple forward and backwards passes through the network. The resulting variations in the feature maps (see sub-figure c for one example) are thus also deliberately generated to magnify throughout the network's layers. In the corresponding feature map of the SCNN (sub-figure d) however, these variations were entirely overwritten by the random fluctuations of the SNs and SC operations. As the figure depicts the differences between the feature map of the original image and the adversarial example, it effectively creates a map that is a mixture of variations caused by the attack's perturbations and SC's random effects. The magnitude of the latter is greater and dominates the map. Because SC's variations do indeed have characteristics of random noise, they cancel each other out frequently throughout the subsequent layers, leading to a correct classification most of the time. Interestingly, the shape of the input image can still be vaguely recognized, as the magnitude of the variations depends on the absolute pixel values in the feature maps.

Expanding on early results by Ting et al. in [TH19], the results above further demonstrate that SCNNs possess inherent defensive properties against adversarial attacks. Small random fluctuations overwrite the effect of specifically crafted input perturbations that C&W attack and other white-box attacks are based on. Boundary attack's iterative optimization algorithm fails to reliably find good adversarial examples due to SCNNs' probabilistic output behaviour, which removes clear decision boundaries that the algorithm needs for orientation. In contrast to probabilistic defensive mechanisms introduced in binary networks, SCNNs achieve these feats without any additional network layers, operations or input transformations and therefore no hardware or computational overhead. It has to be noted that the above simulations only cover two specific attacks and do therefore not prove that SCNNs are resilient towards all attacks. It is for example conceivable that a white-box attack with complete information about the network including the detailed hardware design as well as initial states of all registers and flip-flops in the network could be able to craft SCNN-specific perturbations that persist through the random fluctuations (although such an attack would be computationally significantly more expensive than an attack on a binary network). However, SC's defensive properties do not rely on a specific network type, structure or attack algorithm, they are emerging from elementary proper-

ties of basic arithmetic components. It is therefore not unlikely that these properties carry over in part or even entirely to other attacks, making SC a promising candidate for the implementation of secure small-scale NNs.

Part II

Analysis and Limitations

Chapter 5

Extended Accuracy Management Framework

Accuracy management is an important part of SC design. SN length is directly proportional to computation time and logarithmically proportional to circuit size due to the required size of SNGs and counters in various SC components. It is therefore desirable to keep SNs as short as possible while still reaching the computational accuracy that is required for the application to work as intended. On the other hand, it is beneficial to use SN lengths that are equal to the maximum sequence length of the employed PRNG, as this ensures SN generation with minimum quantization error. In that case knowing the resulting accuracy of the circuit is important for adapting the application correspondingly. The accuracy estimation method for combinational stochastic circuits from [Neu16] has been covered in section 2.2.4. Purely combinational circuits are however only a small sub-class of SC, as they are only able to implement multilinear polynomials [AH15]. In a multilinear polynomial every term can only contain products of variables of degree of at most one. Polynomials with variables of degree two or higher can only be implemented by isolation (or an equivalent decorrelation method), which requires sequential circuit elements.
Due to this limited capability of combinational stochastic circuits, many more complex functions require sequential elements in SC. Among them are for example division [CH16], several types of activation functions for NNs [BC01a] and the maximum function [RLD+17] [NPH19]. Moreover, isolation is a widely used method for decorrelating SNs and requires the use of flip-flops. It is therefore beneficial to have a method to estimate and manage accuracy of sequential stochastic circuits as well. The theoretical accuracy estimation framework for combinational circuits from [Neu16] provides a basis, but is not readily ap-

plicable to sequential circuits due to two main reasons: Firstly, a base experiment covers exactly one clock cycle in a combinational circuit, but can span over an arbitrary number of clock cycles in a sequential circuit. Secondly, a basic assumption of the framework is that inputs are iid random variables, which might not be the case if the circuit includes delay elements and feedback loops. These limitations were overcome in [NPH18b], which successfully extended the original framework to sequential stochastic circuits.

5.1 Sequential circuits without feedback

Sequential circuits without feedback encompass circuits with memory elements (i.e. flip-flops) but no feedback of gate outputs. In SC, this class is formed by circuits with isolators for decorrelation due to shared PRNGs or fanout in the circuit. The core principle of [NPH18b] is to transform a target sequential circuit into a combinational circuit with pseudo primary inputs through a time frame expansion, and then extend the modelling of the circuit's base experiment over multiple clock cycles to express dependencies between random variables. In a time frame expansion, the combinational part of a circuit is copied several times and sequential elements are replaced by connections between these copies and pseudo primary inputs. Figure 5.1 shows an example for a time frame expansion $C_{tf,3}$ of the circuit from figure 2.3d for three clock cycles, which is the necessary amount to remove the three isolators. In the following, this circuit will be used as an example for the general procedure in the case of sequential circuits without feedback.

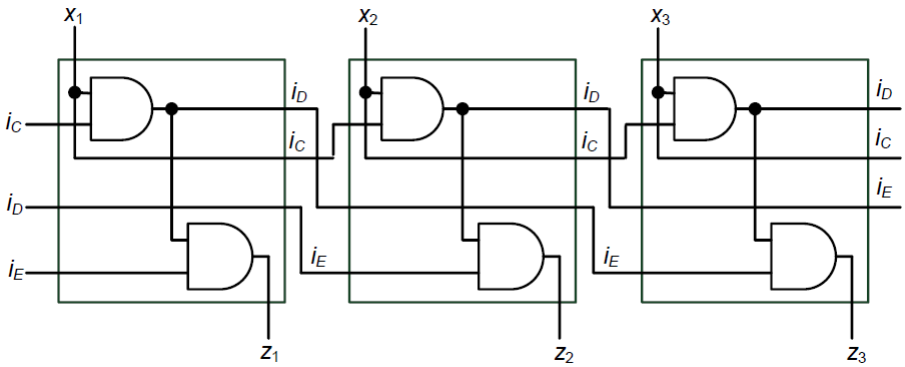

Figure 5.1: Time frame expansion $C_{tf,3}$ of the circuit from figure 2.3d for three clock cycles.

Time frame expansion enables a remodelling of the base experiment, as $C_{tf,3}$ covers three clock cycles of the original circuit in one cycle. However, $C_{tf,3}$ does not address the second problem raised above: The inputs of the subsequent circuit block that computes z_4 are not independent of the inputs of $C_{tf,3}$, for example $x_3 = i_C$ and $x_1x_2 = i_E$. It is therefore not sufficient to formulate a base experiment for $C_{tf,3}$ alone, it has to cover all clock cycles, i.e. the full length of the output SN. A base experiment covering $k \geq 4$ clock cycles is thus modelled as

$$Y = \frac{z_1 + z_2 + \cdots + z_k}{k} \tag{5.1}$$

In the case of $C_{tf,3}$ $z_1 = x_1 i_C i_E$, $z_2 = x_1 x_2 i_D$, $z_3 = x_1 x_2 x_3 i_C$ and $z_j = x_{j-3} x_{j-2} x_{j-1} x_j$ with $j \in \{4, \cdots, k\}$. Again all inputs are modelled as Bernoulli random variables where $\mathbb{E}(x_1) = \mathbb{E}(x_2) = \cdots = \mathbb{E}(x_k) = x$. For the initial i_C, i_D and i_E several different cases are possible. If it is assumed that the circuit is run for at least three warm up cycles, the initial flip-flop states are not influencing the circuit any more and therefore $\mathbb{E}(i_C) = x$ and $\mathbb{E}(i_D) = \mathbb{E}(i_E) = x^2$. Alternatively, other assumptions can be modelled as needed, for example all flip-flops are initially set to 0 or 1 (with corresponding expected values) or are random (e.g. with expected value $\frac{1}{2}$). To keep equations shorter, it will be assumed in the following that the circuit has a warm up period. The variation and standard deviation of the base experiment are then given by

$$\sigma^2(Y) = \mathbb{E}(Y^2) - \mathbb{E}^2(Y) = \frac{\mathbb{E}((z_1 + z_2 + \cdots + z_k)^2)}{k^2} - \mathbb{E}^2(Y) \tag{5.2}$$

which can be rewritten as

$$\sigma^2(Y) = \frac{\mathbb{E}(\hat{z}_1 + \hat{z}_2 + \cdots + \hat{z}_{k^2})}{k^2} - x^8 = \frac{\mathbb{E}(\hat{z}_1) + \mathbb{E}(\hat{z}_2) + \cdots + \mathbb{E}(\hat{z}_{k^2})}{k^2} - x^8 \tag{5.3}$$

where $\hat{z}_{(i-1)k+j} = z_i z_j$ with $i, j \in 1, \cdots, k$. For example, $\hat{z}_1 = z_1^2$, $\hat{z}_2 = z_1 z_2$ and $\hat{z}_{k+3} = z_2 z_3$. The numerator includes k^2 terms, but in circuits without feedback, these terms follow simple patterns. In the example of $C_{tf,3}$, $\mathbb{E}(\hat{z})$ can only take on one of five different values:

- $\mathbb{E}(\hat{z}) = x^4$ if $i = j$.
- $\mathbb{E}(\hat{z}) = x^5$ if $|i - j| = 1$.
- $\mathbb{E}(\hat{z}) = x^6$ if $|i - j| = 2$.
- $\mathbb{E}(\hat{z}) = x^7$ if $|i - j| = 3$.
- $\mathbb{E}(\hat{z}) = x^8$ otherwise.

In general, any sequential circuit without feedback with sequential depth d will have $d+2$ different product terms in the numerator. In this particular example the resulting variance is thus given by

$$\sigma^2(C_{tf,3}) = \frac{kx^4 + 2(k-3)x^5 + 2(k-1)x^6 + 2(k-2)x^7 + (12-7k)x^8}{k^2} \tag{5.4}$$

This base experiment variance now allows accuracy estimation with the formulas given in section 2.2.4. The variable n in equations 2.11 and 2.13 is set to 1, as there is now only a single base experiment that in itself encompasses all n clock cycles. In general, the concrete input values of a circuit are unknown. The presented modelling allows for a worst case estimate by maximizing a circuit's variance within the input bounds of any given application.

Beyond quantifying accuracy of given circuits, the technique allows for a design space exploration of sequential circuits, especially regarding the trade off between accuracy and RNG overhead. RNG sharing leads to lower hardware costs, but on the other hand reduces accuracy, as isolation cannot fully decorrelate SNs from shared RNGs and is furthermore susceptible to autocorrelation. For example, the circuit from figure 2.3d has a range of alternative implementations. On one side of the spectrum is the depicted implementation with one SNG and three isolators, on the other side would be a combinational implementation with four independent SNGs and no isolators. During design phase, these alternatives can now be compared easily by modelling their respective base experiments and using the provided formulas. Continuing the ongoing example, four implementations of $f(x) = x^4$ as shown in figure 5.2 will be compared.

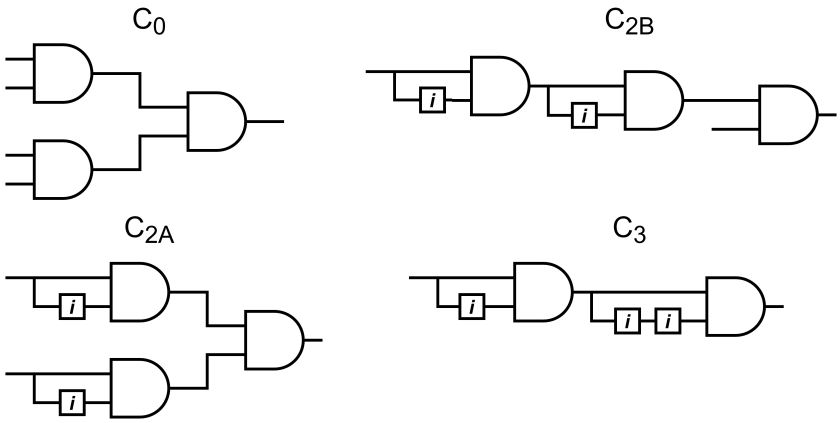

Figure 5.2: Four alternative implementations of $f(x) = x^4$ with differing numbers of SNGs and isolators.

The variance of C_3 has already been computed in equation 5.4. Using the same procedure of time frame expansion and base experiment modelling, the respective variances for the other sequential circuits are:

$$\sigma^2(C_{2A}) = \frac{kx^4 + 2(k-1)x^6 + (2-3k)x^8}{k^2} \tag{5.5}$$

$$\sigma^2(C_{2B}) = \frac{kx^4 + 2(k-1)x^6 + 2(k-2)x^7 + (6-5k)x^8}{k^2} \tag{5.6}$$

and C_0 is a combinational circuit with variance

$$\sigma^2(C_0) = \frac{x^4 - x^8}{k} \tag{5.7}$$

according to equation 2.9. Using the formulas given in 2.2.4, accuracy differences between the circuit variants can now be quantified. Figure 5.3 shows a comparison between estimated MSEs, figure 5.4 the convergence behaviour of the circuits (squares are simulation results, lines are theoretical estimates). Convergence behaviour refers to the confidence $p_n(\epsilon)$ to reach a certain precision ϵ in any single computation. In other words, it specifies the probability that the result of a single computation of a given stochastic circuit differs from the expected result by more than ϵ. This probability can be determined by solving equation 2.13 for γ with the standard deviation of the circuit as given above.

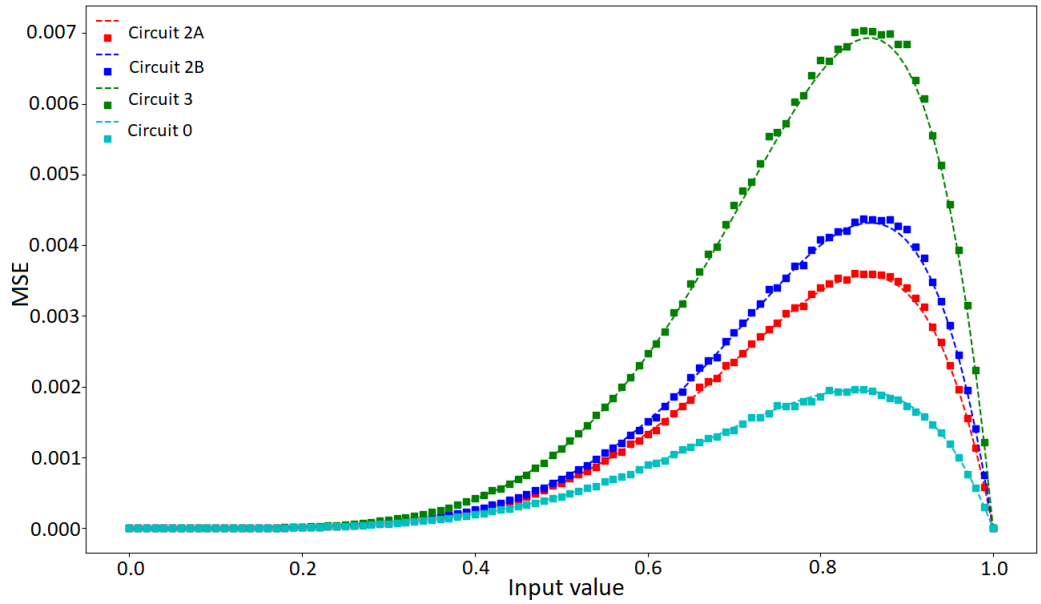

Figure 5.3: MSE of circuit variants from figure 5.2.

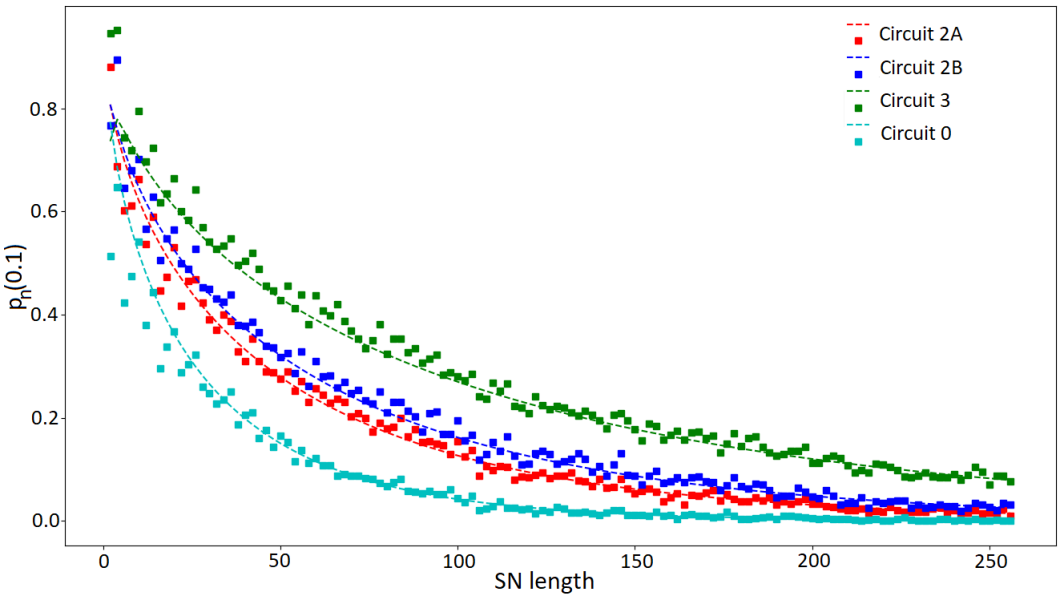

Figure 5.4: Convergence behaviour of circuit variants from figure 5.2 for $\epsilon = 0.1$.

Circuit C_0 shows the lowest MSE and the fastest convergence behaviour, while C_3 has the highest MSE and slowest convergence. This is expected, as C_0 uses four independent SNGs, while C_3 only requires a single SNG, which is decorrelated multiple times, thus cre-

ating dependencies between SNs in the circuit. Much more interesting are the behaviours of circuits C_{2A} and C_{2B}, as both use two SNGs and two isolators. Even so, their MSE and convergence behaviours differ notably. For example, C_{2A} requires an SN length of 116 to reach a confidence level of 10% for a precision $\epsilon = 0.1$, while C_{2B} requires an SN length of 138. Even though the two circuits consume exactly the same hardware area and have the same number of components, their accuracies differ, which the presented framework is able to quantify exactly.

5.2 Sequential circuits with feedback

Several important functions in SC require a circuit implementation with a feedback loop. This includes elementary functions like division with the CORDIV divider (figure 3.3) [CH16], but also more complex functions such as hyperbolic tangent [BC01a] and various implementations of the maximum function [RLD+17] [NPH19] that are based on saturating counters. The general procedure for this type of circuits is the same as presented in the previous section:

1. Construct a time frame expansion over k clock cycles.

2. Extend the modelling of the base experiment over the whole time frame expansion according to equation 5.1.

3. Compute the variance of the base experiment as shown in equation 5.2.

The main difference compared to circuits without feedback lies in the much longer and more complex equations to determine the variance. Any bit in a circuit with feedback can potentially influence all subsequent bits until the end of the computation. The $\mathbb{E}(\hat{z})$ terms in equation 5.3 are therefore all different in general, leading to a fraction with k^2 unique product terms in the numerator. In some cases, it is however possible to find a concise recursive formula to compute the variance. For example the variance of CORDIV is given by:

$s_0 = \frac{x^2}{y} + \frac{x}{y} - x, \quad \sigma_0^2 = k \cdot \frac{x}{y}$ with $1 \leq i \leq k$:

$$\sigma_i^2 = \sigma_{i-1}^2 + 2(k-i) \cdot s_{i-1}$$
$$s_i = \frac{x^2}{y} + (1-y) \cdot s_{i-1}$$

The overall variance σ^2 is then given by:

$$\sigma^2 = \frac{\sigma_k^2 - k^2 \cdot \frac{x^2}{y^2}}{k^2} \tag{5.8}$$

Design space exploration is possible for this circuit type as well. In the case of CORDIV, additional flip-flops can be inserted to increase the accuracy of the circuit at additional hardware cost. In the original design with one flip-flop, every output bit depends on all previous output bits. By adding a second flip-flop, output bits are split in two groups of independent bits (even and odd bit positions of the outputs SN). Adding more flip-flops increases the number of groups. During design phase the impact that adding a flip-flop has on the accuracy can be quantified with the presented formulas. The variance of CORDIV with two flip flops is given by equation 5.8 with

$s_{-1} = \frac{x^2}{y} + \frac{x}{y} - x, \quad t_0 = \frac{x^2}{y^2}, \quad \sigma_0^2 = k \cdot \frac{x}{y} + (2k-2) \cdot \frac{x^2}{y^2}$ with $1 \leq i \leq k-1$

$$\sigma_i^2 = \begin{cases} \sigma_{i-1}^2 + 2(k-1-i) \cdot s_{i-2} & \text{if} \quad i \bmod 2 = 1 \\ \sigma_{i-1}^2 + 2(k-1-i) \cdot t_{i-2} & \text{if} \quad i \bmod 2 = 0 \end{cases}$$
$$s_i = \frac{x^2}{y} + (1-y) \cdot s_{i-2} \quad \text{if} \quad i \bmod 2 = 1$$
$$t_i = \frac{x^2}{y} + (1-y) \cdot t_{i-2} \quad \text{if} \quad i \bmod 2 = 0$$

and for CORDIV with three flip flops by equation 5.8 with

$$s_{-2} = \frac{x^2}{y} + \frac{x}{y} - x, \quad t_{-1} = \frac{x^2}{y^2}, \quad u_0 = \frac{x^2}{y^2}, \quad \sigma_0^2 = k \cdot \frac{x}{y} + (4k-6) \cdot \frac{x^2}{y^2} \quad \text{with } 1 \leq i \leq k-2$$

$$\sigma_i^2 = \begin{cases} \sigma_{i-1}^2 + 2(k-2-i) \cdot s_{i-3} & \text{if} \quad i \bmod 3 = 1 \\ \sigma_{i-1}^2 + 2(k-2-i) \cdot t_{i-3} & \text{if} \quad i \bmod 3 = 2 \\ \sigma_{i-1}^2 + 2(k-2-i) \cdot u_{i-3} & \text{if} \quad i \bmod 3 = 0 \end{cases}$$

$$s_i = \frac{x^2}{y} + (1-y) \cdot s_{i-3} \quad \text{if} \quad i \bmod 3 = 1$$

$$t_i = \frac{x^2}{y} + (1-y) \cdot t_{i-3} \quad \text{if} \quad i \bmod 3 = 2$$

$$u_i = \frac{x^2}{y} + (1-y) \cdot u_{i-3} \quad \text{if} \quad i \bmod 3 = 0$$

These equations for the variances result in terms that include k-th powers of x and y, multiplied by comparably large integers. For larger k it is therefore important to ensure that an adequate number format is used for evaluation to avoid numerical errors. For very large k, e.g. $k = 500$ even double precision floating point numbers might not be sufficient any more to evaluate these equations correctly without rearrangements. This affects all circuits with feedback, as the first bit potentially influences all subsequent bits.

Figure 5.5 shows an analysis for the MSE of CORDIV with one, two and three isolators computing $\frac{x}{0.8}$ for 100 evenly distributed values of $x \in [0, 0.8]$ and an SN length of 128. The MSE for each data point was computed over $10,000$ simulations and a small warm-up period of one, two or three clock cycles (depending on the number of isolators) to flush out the initial unknown flip-flop states is assumed. Differences between the MSEs of the three circuit versions are so small that it is difficult to determine from simulation alone (squares, triangles and circles) if the circuits have different accuracies at all. Using the theoretically obtained variances above reveals that accuracy indeed differs between circuit versions, although the difference is so small that it will most likely not justify the increase in circuit size and can only be properly seen in a zoomed-in section of the graph (figure 5.6). This example demonstrates that the presented accuracy analysis framework enables a designer to compare different circuits precisely without the uncertainties that naturally come with all simulations in SC, and pinpoint even tiny changes in accuracy that the addition or removal of a single isolator can cause.

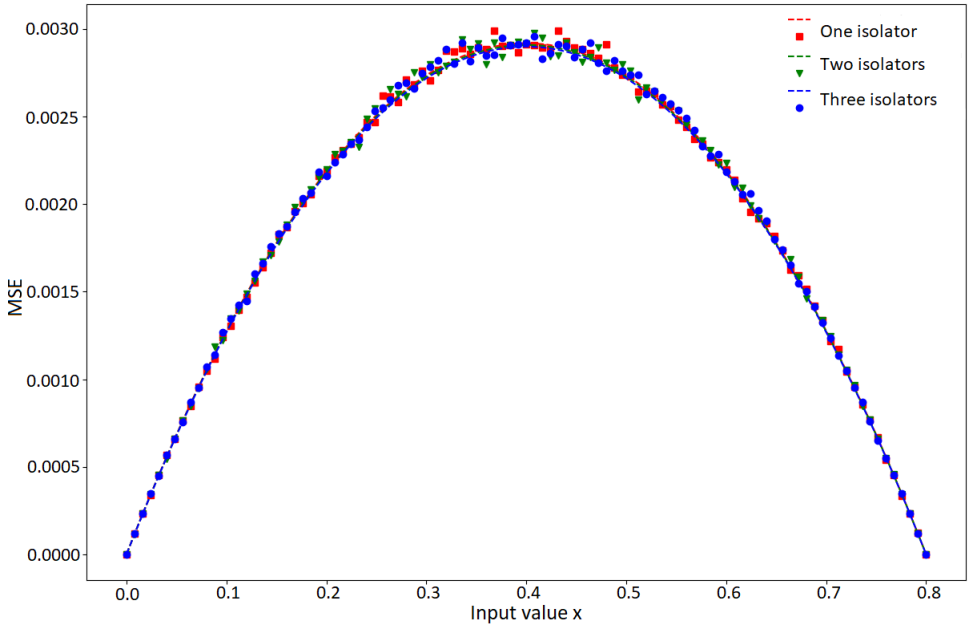

Figure 5.5: Analysis of MSE for CORDIV with one, two and three isolators.

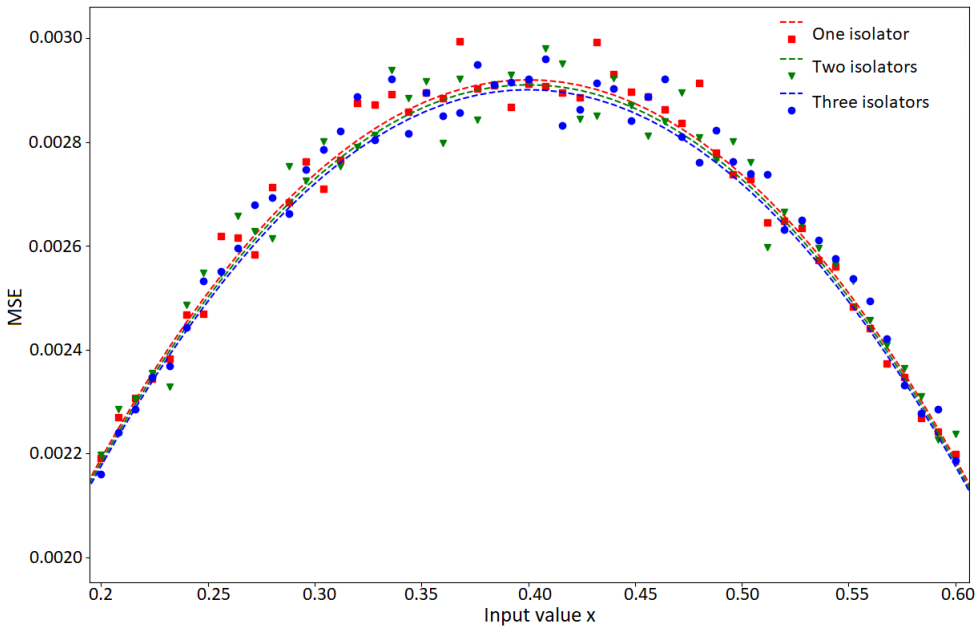

Figure 5.6: Zoomed-in section of subfigure a) that shows the differences of theoretical MSE estimates.

Chapter 6

On the Limitations of SC

6.1 On implementable functions in SC

One important question to be answered is: What are the limits of SC, i.e. which functions are implementable as a stochastic circuit and which are not. In order to address this question it is necessary to first define the concept of an "exact" SC implementation:

Definition 6.1. Exact SC implementation: An SC implementation of a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ with input SNs $X_1, \cdots, X_n$ and $\mathbb{E}(X_1) = x_1, \cdots, \mathbb{E}(X_n) = x_n$ is called an exact SC implementation if the mean $\mathbb{E}(f(X_1, \cdots, X_n))$ exists and

$$\mathbb{E}(f(X_1, \cdots, X_n)) = f(\mathbb{E}(X_1), \cdots, \mathbb{E}(X_n)) = f(x_1, \cdots, x_n) \tag{6.1}$$

Any function f that does not meet the requirement of equation 6.1 cannot have a corresponding exact stochastic circuit. The consequences of this restriction have been investigated in [NPH19] and [NPH19] and are presented in this section.

For some functions it is possible to determine if they fit the above definition without evaluating equation 6.1. This includes all convex functions, as they are satisfying Jensen's inequality:

$$\mathbb{E}\left(f(X)\right) \geq f\left(\mathbb{E}(X)\right) \tag{6.2}$$

with the opposite relation holding for concave functions. Any concave function can be turned into a convex function by a multiplication with -1, therefore only convex functions will be mentioned from here on. Equality in 6.2 only holds if f is linear, or if X is constant almost surely. The latter condition cannot be met in SC, as that would mean that a circuit's inputs never change, which would make it obsolete. Therefore, it follows from definition 6.1 that an exact SC implementation for a convex function f can only exist, if f is also linear. Linear functions have to satisfy additivity and homogeneity of degree 1:

Additivity:

$$f(a+b) = f(a) + f(b) \tag{6.3}$$

Homogeneity of degree 1:

$$f(\alpha \cdot a) = \alpha \cdot f(a) \tag{6.4}$$

For example, scaled addition via MUX $g(a,b) = \frac{a+b}{2}$ is a linear function as it is additive

$$g\left((a,b)+(c,d)\right) = g(a+c, b+d) = \frac{a+c+b+d}{2} = \frac{a+b}{2} + \frac{c+d}{2} = g(a,b) + g(c,d)$$

and homogeneous

$$g\left(\alpha \cdot (a,b)\right) = g(\alpha \cdot a, \alpha \cdot b) = \frac{\alpha \cdot a + \alpha \cdot b}{2} = \alpha \cdot g(a,b)$$

and therefore can be implemented exactly in SC.
On the other hand, the function $h(a) = a^2$ is not linear, as it is not additive, and is homogeneous of degree 2:

$$h(a+b) = (a+b)^2 \neq a^2 + b^2$$

$$h(\alpha \cdot a) = \alpha^2 \cdot a^2 \neq \alpha \cdot h(a)$$

A more visual description of a linear function is a function that maps linear subspaces onto linear subspaces of equal or lower dimensionality, e.g a straight line is mapped onto

a straight line or a point. In the case of h, a straight line (e.g. the real line in the interval $[0, 1]$) is mapped onto a curved line, signifying that h is not linear.

It has to be noted that Jensen's inequality only applies to convex functions and does not allow any conclusion otherwise. It is possible for non-linear functions to meet the requirements in definition 6.1. An important example in the context of SC is the function $j(x, y) = xy$, which is neither additive nor homogeneous of degree 1, but satisfies equation 6.1 if the respective random variables of x and y are independent. This case is not covered by Jensen's inequality because j is neither convex nor concave. The function is of special importance for SC, as Jensen's inequality would otherwise suggest that polynomials cannot be implemented exactly in SC (see function h above). However there have been several works that show the opposite, for example works on synthesis and design of polynomial stochastic circuits [AH15] [QR08] [QLR$^+$10]. This apparent contradiction stems from a simplified way of notation in SC. [AH15] already notes that SC actually implements multi-linear polynomials, where every variable appears with degree of at most 1. Different random variables with the same expected value are however commonly considered to be the same real variable in the context of SC. For example, the squarer circuit 2.3b) is said to implement x^2. However, due to decorrelation it would be more accurate to say that it is implementing xy with $x = y$, as the AND gate receives two independent bits (i.e. random variables) in each clock cycle under the assumption that subsequent bits of X are independent of each other.

Definition 6.1 also does not imply that a function without exact SC implementation is not usable in SC, but that one should be aware of the possible effects. Such a function is subject to an approximation error which causes a systematic deviation from the target function similar to correlation errors. This is specifically an issue in applications where these biases can accumulate in further operations, such as SCNNs.
As a consequence, the most affected function in recent SC applications is the maximum function. It is used in SCNNs for both activation functions (ReLU) and max-pooling subsampling layers, and is often followed by multiply-accumulate operations. The maximum function is a convex function (proof in appendix A.2) that is non-linear, as it is not addi-

tive:

$$\max((1,3)+(2,1)) = \max(3,4) = 4 \neq \max(1,3) + \max(2,1) = 3 + 2 = 5$$

Strict inequality therefore holds in Jensen's inequality (6.2) and definition 6.1 is not met. The knowledge that the maximum function is biased in SC is valuable by itself, however a quantification of the effect is desirable, i.e. a way to compute

$$\text{bias} = \mathbb{E}(\max(X_1, \cdots, X_n)) - \max(\mathbb{E}(X_1), \cdots, \mathbb{E}(X_n)). \tag{6.5}$$

A formula for the general case of the maximum of non-identically distributed random variables is not known at this time, but an expression can be found for the special case of iid random variables. While this case may seem very restrictive, it is still practically relevant in SCNNs, as many of them deal with the classification of images. Maximum operations in those systems are typically performed over small 2×2-sized windows of neighbouring pixels that often have very similar or even identical pixel values (e.g. the black background pixels in figure 6.1). It has been explained in section 2.2.4 that the value of an SN can be approximated as a normal distributed random variable, if the iid assumption holds for individual bits of this SN. Under those circumstances, it is possible to find a closed form expression for equation 6.5, as shown in [NPH19] (proof in appendix A.3):

$$\mathbb{E}\left(\max_i X_i\right) = \mu + \sigma \int_{-\infty}^{\infty} t \frac{d}{dt} \Phi(t)^n \, dt \tag{6.6}$$

However, even in this special case, the value can only be computed numerically for $n < 5$. Experimental results for the bias have been presented in table 3.4 and match the theoretically predicted values well, for example the measured bias of 0.032 for $\mathbb{E}(X) = 0$ corresponds up to three decimal places to the result of equation 6.6. Even though the extent of the bias seems small compared to the effects of random fluctuations, it should be considered carefully in SC design, as it affects not only the output of each individual computation, but changes the expected output value and therefore the function implemented by the circuit itself.

6.2 On types of stochastic circuits and practical limitations

Contrary to the theoretical limitations of SC regarding implementable functions, practical limitations are much harder to judge, as they depend on factors that vary on a case by case basis. An SN length of 512 might be perfectly acceptable in a sensor node that only transmits information once per minute, but unacceptable in an image processing system with real-time requirements. Especially the value range restriction of unipolar and bipolar SNs often leads to complications that necessitate work-arounds in many SC systems. In image processing, pixel values (either grey scale or RGB) are commonly given in the range of 0 to 255; NN parameters are commonly also not restricted to specific values except for quantization to reduce required memory for storing. In the case of pixel values, scaling down all values by a factor of 256 is often sufficient and does not even require any additional hardware. However, input parameters that have unbounded values cannot simply be scaled down, as the required scaling factor is unknown. In that case, the algorithm itself has to be adapted, e.g. by restricting NN weights to $[-1, 1]$ during training, which in turn restricts the search space of the training algorithm.

Beyond the basic building blocks of multiplication and scaled addition, many new SC components have been introduced that include conventional binary counters or perform parts of their computations in binary arithmetic. Examples can regularly be found in works on SCNN, where parallel counters and approximate versions thereof are used as unscaled adders, replacing the basic scaled addition by MUX [KKY$^+$16][RLD$^+$17]. Lately, complete systems based on integral SC have been proposed [ALPO$^+$17] that function according to the same basic principle as SC, but are based on streams of small Integers instead of single bits. The idea of stochastic-binary hybrid components is not new and has in fact been an important part of Gaines' original work, where he introduces the ADDIE component that computes functions such as square root of an SN using an integrator. These hybrid components overcome some limitations of basic combinational SC components, as they are generally not restricted in the number range and can have an internal memory, but the costs can be significant. SC-binary hybrid components can often not be used in sequence with each other due to their mix of binary and stochastic input and output formats, and might therefore require additional SNGs, which are among the most expensive compo-

nents in an SC system. This was the main reason for the relatively small role that the ADDIE played in subsequent research despite its versatility, as each individual ADDIE requires an SNG or at least a comparator in order to produce an SN at its output. In the case of unscaled addition, it is clear that such a component can not be readily followed by another SC-binary hybrid component, as the computation has been moved from the stochastic into the binary domain.

Furthermore, hybrid components potentially lose one of basic SC's strongest benefits, namely its error tolerance (see section 4.1). The use of elements with conventional binary number formats within an SC component reintroduces weighted bit positions. While any bit in an SN contributes to the overall SN value only by $\frac{1}{n}$ respectively $\frac{2}{n}$ (depending on the SN format), and can therefore also only cause the SN value to deviate by this amount in case of a bit flip, any binary element can potentially change a value by 50% of its maximum value when the MSB flips. SCNN implementations rely especially on SC-binary hybrid components for addition, because the downscaling of MUX-based addition poses a significant problem as shown later in this section. Due to these numerous significant differences between purely stochastic and hybrid components, a discussion of SC's limitations needs to take the implementation type into account. A clear classification of stochastic circuits is therefore desirable. On the basis of the discussion in [NPH19], three classes are proposed:

1. **Strongly stochastic circuit type 1:** *Basic* strongly stochastic circuits of type 1 receive only SNs as inputs, compute a single basic arithmetic operation with only SNs as outputs and do not include elements that use a non-SN number format. A strongly stochastic circuit of type 1 consists exclusively of *basic* strongly stochastic circuits of type 1.

2. **Strongly stochastic circuit type 2:** *Basic* strongly stochastic circuits of type 2 receive only SNs as inputs and compute a single basic arithmetic operation with only SNs as outputs. A strongly stochastic circuit of type 2 consists exclusively of *basic* strongly stochastic circuits of type 2.

3. **Weakly stochastic circuit:** *Basic* weakly stochastic circuits receive only SNs as inputs and compute a single basic arithmetic operation with at least one output in a

conventional binary number format, or vice-versa. A weakly stochastic circuit includes at least one basic weakly stochastic circuit.

Type 1 strongly stochastic circuits include many combinational circuits such as the functional components of SC image processing implementations [ALH13] and digital filters [WHCE16] [ISI+16] that mainly consist of multiplication and MUX-based addition, but also sequential circuits such as the CORDIV divider [CH16] and otherwise combinational circuits with isolators. They form the most basic and "classical" type of stochastic circuits, with a strict limitation of all input, output and internal values to the range of $[-1, 1]$. They are therefore the most restrictive type in terms of implementable functions, but on the other hand offer the highest error tolerance, as demonstrated in investigations of strongly stochastic image processing circuits in [ALH13] [LL11].

Counter and FSM-based circuits often fall into the class of type 2 strongly stochastic circuits, including stanh [BC01a] and different implementations of the maximum function [RLD+17] [NPH19] among others. Gaines' ADDIE component is part of this class as well. The internal use of non-SC number formats allows this type of stochastic circuit to track dependencies between or within SNs over several clock cycles more easily, which is for example a requirement for the efficient and delay free implementation of the maximum function. The trade off for this increase in flexibility is a reduced tolerance towards errors and sometimes also towards autocorrelation. Usually, FSM-based stochastic circuits like stanh are designed assuming no systematic patterns in input SNs and can therefore perform poorly when used in combination with autocorrelated inputs, as shown in 3.2. Strongly stochastic circuits of both types can be combined easily without intermediate conversions or normalizations, as their input and output formats match.

The most important examples of weakly stochastic circuits are the classical SNG with binary inputs, stochastic outputs and a PCC, and the final SN-to-binary conversion usually implemented with a counter as the final component of an SC system. Having an unrestricted range for either input or output values makes them the most flexible class regarding their functionality, but also comes with several drawbacks. Besides their reduced error tolerance, non matching value ranges prevent them from being readily used in sequence with strongly stochastic circuits and often even with other weakly stochastic circuits. Mod-

ification for interfacing is generally required, such as the Btanh variant [KKY+16], which allows stanh to be used in combination with unscaled addition.

With SNGs in the class of weakly stochastic circuits, almost all SC systems are considered weakly stochastic circuits as a whole. The only exceptions are systems with unconventional SNGs, for example direct analogue to stochastic conversions [LAH+17]. While this seems counter-intuitive at first, consider that any stochastic circuit produces incorrect results if the input values themselves (which are in binary format before being transformed by the SNG) are faulty. Similarly, if the SC-to-binary conversion is faulty due to errors in the counter, the output of the SC system will be incorrect, independently of the properties and accuracy of its arithmetic portion. The main difference between the vast majority of SC systems therefore lies in the classification of their arithmetic circuit portion.

6.2.1 Practical limitations of scaled addition

A major practical limitation in SC is scaled strongly stochastic addition. Some SC applications such as digital filters and NNs include sums of several hundred or even thousands of values. The corresponding large downscaling factors often cause a significant reduction in absolute values in the systems. This leads to problems with loss of information and longer computation times due to higher required precision. For example, the different variants of the well known MNIST datasets consist of grey scale images with an object on black background. Background pixels have a value of 0, while pixels belonging to the object have values up to 1 (white). As pixel values are added together in an NN that classifies these objects, values gradually get downscaled and concentrate in a smaller interval around 0. Figure 6.1 shows an example of an image from the MNIST dataset, with the corresponding histogram of pixel values in figure 6.2.

During the convolution operation in the first layer of a CNN, these pixel values are first multiplied with weights that lie in the range $[-1, 1]$ and these products are added. With scaled stochastic addition, the results are downscaled by a factor of k^2 for a $k \times k$ convolution kernel, causing most post-convolution values to concentrate around the center of the value range. Figure 6.3 shows the resulting distribution of values in the network after the layer. Almost all values fall in the interval $[-\frac{1}{16}, \frac{1}{16}]$.

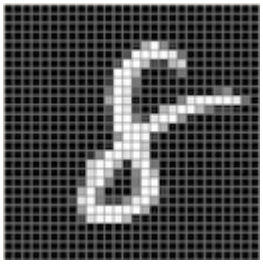

Figure 6.1: Example for an input image in the MNIST dataset.

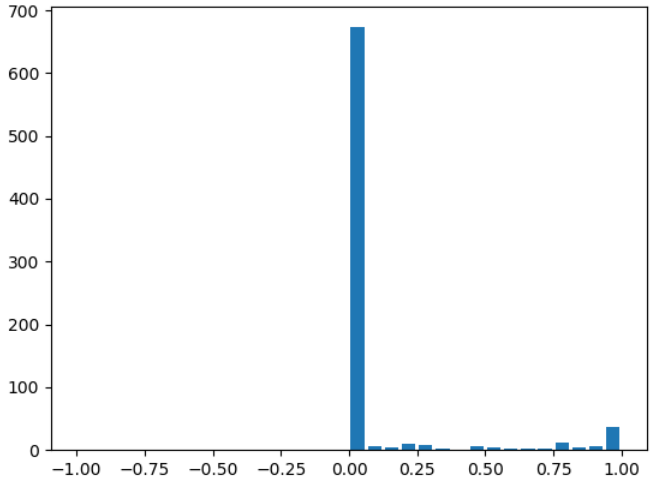

Figure 6.2: Distribution of pixel values in image 6.1.

Situations such as the example above lead to two significant problems. Firstly, the signal to noise ratio (SNR) is reduced, as the random noise in SC does not scale down accordingly. In fact, according to equation 2.10 the noise in bipolar SNs is highest when the SN's value is 0. As the SNR decreases and the relative errors in the SNs increase, information is lost and cannot be recovered any more. Chaining such operations together will eventually lead to a loss of all useful information in the system, a problem that affects SCNNs to a large extent due to their sequential, layer-based structure. There are two main possibilities to address this issue in a strongly stochastic system: Increasing the SN length proportionally to the scaling factor reduces the loss of information, as longer SNs have higher precision, but does not prevent the gradual decrease in absolute values. Upscaling of values on the other hand can keep values distributed over a broader range, but does not help against loss of information.

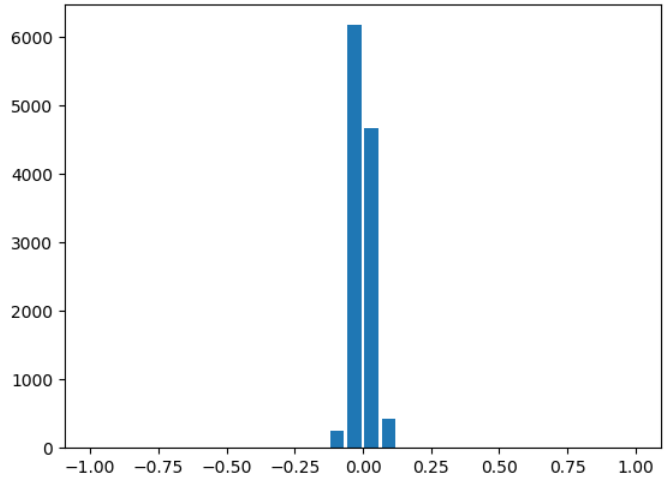

Figure 6.3: Distribution of pixel values after the first convolutional layer of an SCNN classifying the image 6.1.

Increasing the SN length naturally comes with an increase in computation time and/or circuit area and is therefore not the preferred solution. As SN length increases linearly with precision, downscaling by a factor of d requires an increase in SN length by the same factor to counteract any possible information loss due to this scaling. Even in small NNs with few layers, combined scaling factors exceed factors of several hundreds or thousands, making this possibility infeasible in practical applications. For example, the NN structure used in the simulations of section 4.1 would have a scaling factor of 16 in the first layer and 180 in the second layer, which would combine to a factor of 2880. Therefore, upscaling is commonly used in combination with strongly stochastic addition when possible. From a mathematical view, upscaling is a tough challenge in SC, as it requires a multiplication with a value larger than 1, i.e. outside of the typical range of SNs, and also requires a clipping mechanism to ensure that the resulting value stays inside this range as well. Performing these operations in binary is not an option, as conversion between the SC and binary domain is the most hardware intensive part of SC, as laid out in section 3.2. On the other hand, circuits that duplicate 1s in an SN suffer from the problem that SN values are generally not known during computation. Upscaling in this manner would therefore lead to significant saturation errors, e.g. an initial SN value of 0.6 that is upscaled by a factor of 2 would suffer a saturation error of $1.2 - 1 = 0.2$.

Some stochastic circuits posess intrinsic upscaling functionality. The stanh component (figure 2.5) approximates the function $f(x) = \tanh\left(\frac{N}{2}x\right)$. When used after a strongly stochastic addition, e.g. in a previous convolutional layer, it serves as a countermeasure to downscaling during addition and a functional component at the same time. However, this does not prevent information loss during the downscaling and may in fact even decrease accuracy, as small relative errors are magnified and lead to large absolute errors.

Due to the issues of strongly stochastic addition, most SCNN applications have moved entirely to weakly stochastic addition, i.e. hybrid SC-binary adders in different forms of (approximate) parallel counters. The use of strongly stochastic addition lies mostly in relatively small functional circuits, such as the Robert's cross algorithm for edge detection [ALH13], the implementation of polynomials [QR08] [QLR$^+$10], and SC filters [ISI$^+$16] [WHCE16]. Although there have been recent advances in the design of more accurate strongly stochastic adders [BH22], the large additions in SCNNs will continue to pose a limitation to MUX-based SC adders.

Chapter 7

Conclusion

Neural networks have become the main target applications for SC in recent years for several reasons. Firstly, their large reliance on multiplications makes them an attractive target, as SC provides a very efficient multiplier implementation. Secondly, they are able to tolerate small computational inaccuracies and can therefore handle SC's random fluctuations well. Thirdly, the increasing importance of edge devices and low-power computing increases the need for implementations that work reliably in resource constrained environments. With its error tolerant number format, SC is a promising candidate for this task. On the other hand, each of these benefits is connected with its own challenges and open questions: SNs can be multiplied efficiently, but their generation is more expensive. Sharing of PRNGs can reduce the resulting hardware overhead, but can interfere strongly with sequential SC components. SC's random fluctuations can be tolerated up to some degree by an NN, but generally lead to lower classification accuracies. Finally, SC's error tolerance has been investigated in the past, but prior analysis was primarily focused on combinational SC systems, e.g. image processing applications.

The first part of this work presented several contributions regarding these challenges and questions: The S-Box based random number generator is introduced to enable PRNG-sharing without introducing correlations that interfere with sequential SC components. SBoNG achieves this by combining an LFSR with a cryptographic S-box to shuffle the LFSR's state, which mostly eliminates the linear dependencies between subsequent LFSR states that lead to cross-correlation and autocorrelation of SNs. The cross-correlation values of multiple SNs generated by one SBoNG instance are comparable to those achieved

with complex and expensive software-based PRNGs like Mersenne Twister. This enables SBoNG to be shared between many more SNGs than LFSRs, reducing the hardware overhead of SNGs in systems that require many uncorrelated SNs and/or include many sequential components that are sensitive to autocorrelation within SNs. Simulations of a digital filter and an SCNN show that SBoNG provides accurate results while keeping the hardware cost low.

The NMax circuit was proposed as the first non-approximate SC implementation of the maximum function and reduces the random fluctuations in max-pooling layers of SCNNs. Previous stochastic circuits for this function were based on approximations of the maximum value to avoid delays resulting from the accurate comparison of input values. NMax avoids these delays entirely by only tracking the differences between input values, not their actual values. It does not rely on specific input correlations and can therefore readily be interfaced with any other SC components. Outside of improving max-pooling layers, NMax also enables a non-approximate implementation of the clipped ReLU activation function and can in general be used for any SC application that requires computation of maximums.

The third major practical contribution of this work is a detailed analysis of the effect of timing errors on an SCNN and its components. Existing works focused on bit flip models and/or combinational SC components and could therefore not capture the behaviour of SCNNs with their various sequential components accurately under low-power conditions. The analysis presented in this work expands significantly on these prior investigations by employing an error model that combines circuit wide slow-downs to model low-power environments with individual per-gate delay variations to model manufacturing differences. The results show that the investigated SCNN is able to compensate small increases in delay very well, as it shows no decrease in classification accuracy when the circuit delay is increased up to 20% over nominal delay. Larger delays however lead to a significant drop in accuracy, proving that some SCNN components are heavily influenced by timing errors, most strongly among them the combined convolution-activation circuit. The reason was found to be a strong bias in the error distributions of these components. These biases lead to systematic deviations in the implemented arithmetic functions and can cause the SCNN to have a lower accuracy than the binary NN, even though its MSE is lower. It was further

demonstrated on the example of AMax and NMax that different SC components that implement the same functionality can have vastly different characteristic error distributions, and show very different behaviour depending on the error model. While an AMax-based SCNN outperforms its NMax-based counterpart under a bit flip error model, both show almost identical behaviour under a timing error model. These observations are crucial for the design of SC for error-prone environments in general and extreme low-power environments specifically. While SC systems do indeed often show higher resilience than corresponding binary circuits, error tolerance is by no means guaranteed, and should therefore be an important part of the SC design process. SC components should be designed with a particular error model in mind and tested for biased error distributions under this model.

The second main part of this work addresses theoretical observations on and insights into SC. The author's previously existing accuracy management framework for combinational circuits was extended to sequential circuits. Through time frame expansion, a sequential circuit is modelled as a combinational circuit with multiple inputs. Intermediate signals that include identical bits of the input SNs are modelled as dependent random variables. With this extension, it is now possible to estimate the accuracy of sequential stochastic circuits without simulation. The effect of isolators on a circuit's accuracy can be determined as well, which provides a means to find an optimal balance between RNG sharing and additional isolators to trade-off circuit area and output accuracy.
Finally, a contribution towards better understanding the practical and theoretical limitations of SC was made. It was shown that some classes of functions cannot be implemented exactly in SC. This includes all non-linear convex and concave functions. SC implementations of such functions exhibit systematic, biased deviations from their target functions. Furthermore, the categorization of strongly and weakly stochastic circuits was introduced to distinguish between types of stochastic circuits that have differing practical limitations. Strongly stochastic circuits are limited by their restricted input and output values range and are prone to loss of information in applications that include many additions, e.g. digital filters and NNs. Weakly stochastic circuits circumvent this specific limitation with either binary inputs or outputs, but generally require more expensive building blocks such as full adders and accumulators, and potentially additional number conversions. Each type of stochastic circuit comes with its own advantages and limitations, and large SC systems

such as SCNNs often combine all types to find a balance between efficient and accurate implementation.

SCNNs have become the primary application for SC in recent years. Their tolerance for approximate computations and large number of individually simple operations has made them an attractive target for SC implementations and advent of distributed, resource constrained devices further incentivizes the development of these systems. New SC components, insights into SC's robustness under such resource constrained conditions, a clearer understanding of its limitations, and an extended framework to analyse the accuracy of SC systems presented in this work help in the design, evaluation and improvement of these future SC systems.

Bibliography

[ACH+17] Armin Alaghi, Wei-Ting J. Chan, John P. Hayes, Andrew B. Kahng, and Jiajia Li. Trading accuracy for energy in stochastic circuit design. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):1–30, 2017.

[AH13] Armin Alaghi and John P. Hayes. Exploiting correlation in stochastic circuit design. In *2013 IEEE 31st International Conference on Computer Design (ICCD)*, pages 39–46. IEEE, 2013.

[AH15] Armin Alaghi and John P. Hayes. Strauss: Spectral transform use in stochastic circuit synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1770–1783, 2015.

[ALH13] 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*, pages 1–6, 2013.

[ALPO+17] Arash Ardakani, François Leduc-Primeau, Naoya Onizawa, Takahiro Hanyu, and Warren J. Gross. Vlsi implementation of deep neural network using integral stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10):2688–2699, 2017.

[AlQ21] Mohammed AlQuraishi. Machine learning in protein structure prediction. *Current opinion in chemical biology*, 65:1–8, 2021.

[BC01a] Bradley D. Brown and Howard C. Card. Stochastic neural computation. i. computational elements. *IEEE Transactions on computers*, 50(9):891–905, 2001.

[BC01b] Bradley D. Brown and Howard C. Card. Stochastic neural computation. ii. soft competitive learning. *IEEE Transactions on Computers*, 50(9):906–920, 2001.

[BH20] Timothy J. Baker and John P. Hayes. The hypergeometric distribution as a more accurate model for stochastic computing. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 592–597. IEEE, 2020.

[BH22] Timothy J Baker and John P Hayes. Cemux: Maximizing the accuracy of stochastic mux adders and an application to filter design. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 27(3):1–26, 2022.

[BJRL15] George E.P. Box, Gwilym M. Jenkins, Gregory C. Reinsel, and Greta M. Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015.

[BRB] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Boundary attack code. `https://github.com/greentfrapp/boundary-attack`. Accessed: 2022-02-10.

[BRB17] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *arXiv preprint arXiv:1712.04248*, 2017.

[Bro] Robert G. Brown. Dieharder: A random number test suite. `https://webhome.phy.duke.edu/~rgb/General/dieharder.php`. Accessed: 2021-07-28.

[Bro06] Petrus M.T. Broersen. *Automatic autocorrelation and spectral analysis*. Springer Science & Business Media, 2006.

[CH16] Te-Hsuan Chen and John P. Hayes. Design of division circuits for stochastic computing. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 116–121. IEEE, 2016.

[CMO+15] Vincent Canals, Antoni Morro, Antoni Oliver, Miquel L. Alomar, and Josep L. Rosselló. A new stochastic computing methodology for efficient neural network implementation. *IEEE transactions on neural networks and learning systems*, 27(3):551–564, 2015.

[CW] Nicholas Carlini and David Wagner. Cw attack code. `https://github.com/carlini/nn_robust_attacks`. Accessed: 2022-02-09.

[CW17] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *2017 ieee symposium on security and privacy (sp)*, pages 39–57. IEEE, 2017.

[CZS+17] Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models. In *Proceedings of the 10th ACM workshop on artificial intelligence and security*, pages 15–26, 2017.

[DAL+18] Guneet S. Dhillon, Kamyar Azizzadenesheli, Zachary C. Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Anima Anandkumar. Stochastic activation pruning for robust adversarial defense. *arXiv preprint arXiv:1803.01442*, 2018.

[EKD+03] Dan Ernst, Nam Sung Kim, Shidhartha Das, Sanjay Pant, Rajeev Rao, Toan Pham, Conrad Ziesler, David Blaauw, Todd Austin, Krisztian Flautner, et al. Razor: A low-power pipeline based on circuit-level timing speculation. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pages 7–7. Citeseer, 2003.

[FNL+19] S. Rasoul Faraji, M. Hassan Najafi, Bingzhe Li, David J. Lilja, and Kia Bazargan. Energy-efficient convolutional neural networks with deterministic bit-stream processing. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1757–1762. IEEE, 2019.

[GAG+00] Ary L. Goldberger, Luis A.N. Amaral, Leon Glass, Jeffrey M. Hausdorff, Plamen Ch. Ivanov, Roger G. Mark, Joseph E. Mietus, George B. Moody, Chung-Kang Peng, and H. Eugene Stanley. Physiobank, physiotoolkit, and physionet: components of a new research resource for complex physiologic signals. *circulation*, 101(23):e215–e220, 2000.

[Gai67] Brian R. Gaines. Stochastic computing. In *Proceedings of the April 18-20, 1967, spring joint computer conference*, pages 149–156, 1967.

[Gai69] Brian R. Gaines. Stochastic computing systems. In *Advances in information systems science*, pages 37–172. Springer, 1969.

[GBH+16] Maël Gay, Jan Burchard, Jan Horácek, Ange-Salomé Messeng Ekossono, Tobias Schubert, Bernd Becker, Martin Kreuzer, and Ilia Polian. Small scale aes toolbox: algebraic and propositional formulas, circuit-implementations and fault equations. 2016.

[GK88] Prabhat Kumar Gupta and Ramdas Kumaresan. Binary multiplication with pn sequences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(4):603–606, 1988.

[GMRR12] Vaibhav Gupta, Debabrata Mohapatra, Anand Raghunathan, and Kaushik Roy. Low-power digital signal processing using approximate adders. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(1):124–137, 2012.

[GRCVDM17] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens Van Der Maaten. Countering adversarial images using input transformations. *arXiv preprint arXiv:1711.00117*, 2017.

[GSS14] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[HGT+19] Reza Hojabr, Kamyar Givaki, SM Reza Tayaranian, Parsa Esfahanian, Ahmad Khonsari, Dara Rahmati, and M. Hassan Najafi. Skippynn: An embedded stochastic-computing accelerator for convolutional neural networks. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2019.

[HIW15] Stefan Holst, Michael E. Imhof, and Hans-Joachim Wunderlich. High-throughput logic timing simulation on gpgpus. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 20(3):1–22, 2015.

[HMHAA21] Hsuan Hsiao, Joshua San Miguel, Yuko Hara-Azumi, and Jason Anderson. Zero correlation error: A metric for finite-length bitstream independence in stochastic computing. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pages 260–265, 2021.

[HPB+19] Tifenn Hirtzlin, Bogdan Penkovsky, Marc Bocquet, Jacques-Olivier Klein, Jean-Michel Portal, and Damien Querlioz. Stochastic computing for hardware implementation of binarized neural networks. *IEEE Access*, 7:76394–76403, 2019.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[IIS+14] Hideyuki Ichihara, Shota Ishii, Daiki Sunamori, Tsuyoshi Iwagaki, and Tomoo Inoue. Compact and accurate stochastic circuits with shared random number sources. In *2014 IEEE 32nd International Conference on Computer Design (ICCD)*, pages 361–366. IEEE, 2014.

[IMII19] Hideyuki Ichihara, Yuki Maeda, Tsuyoshi Iwagaki, and Tomoo Inoue. State encoding with stochastic numbers for transient fault tolerant linear finite state machines. In *2019 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT)*, pages 1–6. IEEE, 2019.

[ISI+16] Hideyuki Ichihara, Tatsuyoshi Sugino, Shota Ishii, Tsuyoshi Iwagaki, and Tomoo Inoue. Compact and accurate digital filters based on stochastic computing. *IEEE Transactions on Emerging Topics in Computing*, 7(1):31–43, 2016.

[KA17] M. Burak Karadeniz and Mustafa Altun. Sampling based random number generator for stochastic computing. In *2017 24th IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, pages 227–230. IEEE, 2017.

[KKY+16] Kyounghoon Kim, Jungki Kim, Joonsang Yu, Jungwoo Seo, Jongeun Lee, and Kiyoung Choi. Dynamic energy-accuracy trade-off using stochastic computing in deep neural networks. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6, 2016.

[KLZ14] Phil Knag, Wei Lu, and Zhengya Zhang. A native stochastic computing architecture enabled by memristors. *IEEE Transactions on Nanotechnology*, 13(2):283–293, 2014.

[KMM+17] Daewoo Kim, Mansureh S Moghaddam, Hossein Moradian, Hyeonuk Sim, Jongeun Lee, and Kiyoung Choi. Fpga implementation of convolutional neural network based on stochastic computing. In *2017 International Conference on Field Programmable Technology (ICFPT)*, pages 287–290. IEEE, 2017.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[LAH+17] Vincent T. Lee, Armin Alaghi, John P. Hayes, Visvesh Sathe, and Luis Ceze. Energy-efficient hybrid stochastic-binary neural networks for near-sensor computing. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 13–18. IEEE, 2017.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[LH17] Siting Liu and Jie Han. Energy efficient stochastic computing with sobol sequences. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 650–653. IEEE, 2017.

[LL11] 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)*, pages 154–161. IEEE, 2011.

[LLLH19] Yidong Liu, Leibo Liu, Fabrizio Lombardi, and Jie Han. An energy-efficient and noise-tolerant recurrent neural network using stochastic computing. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(9):2213–2221, 2019.

[LLR+18] Zhe Li, Ji Li, Ao Ren, Ruizhe Cai, Caiwen Ding, Xuehai Qian, Jeffrey Draper, Bo Yuan, Jian Tang, Qinru Qiu, et al. Heif: Highly efficient stochastic computing-based inference framework for deep neural networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(8):1543–1556, 2018.

[LRL+17] Ji Li, Ao Ren, Zhe Li, Caiwen Ding, Bo Yuan, Qinru Qiu, and Yanzhi Wang. Towards acceleration of deep convolutional neural networks using stochastic computing. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 115–120. IEEE, 2017.

[Lug05] Tatiana S. Lugovaya. Biometric human identification based on electrocardiogram. *Master's thesis, Faculty of Computing Technologies and Informatics, Electrotechnical University 'LETI', Saint-Petersburg, Russian Federation*, 2005.

[LWLH18] Yidong Liu, Yanzhi Wang, Fabrizio Lombardi, and Jie Han. An energy-efficient online-learning stochastic computational deep belief network. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 8(3):454–465, 2018.

[MGNR12] Thomas Müller-Gronbach, Erich Novak, and Klaus Ritter. *Monte Carlo-Algorithmen*. Springer-Verlag, 2012.

[MN98] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

[MP11] Frosso S. Makri and Zaharias M. Psillakis. On success runs of a fixed length in bernoulli sequences: Exact and asymptotic results. *Computers & Mathematics with Applications*, 61(4):761–772, 2011.

[MZWH19] Guy Maor, Xiaoming Zeng, Zhendong Wang, and Yang Hu. An fpga implementation of stochastic computing-based lstm. In *2019 IEEE 37th International Conference on Computer Design (ICCD)*, pages 38–46. IEEE, 2019.

[NJLR19] M. Hassan Najafi, Devon Jenson, David J. Lilja, and Marc D. Riedel. Performing stochastic computation deterministically. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2925–2938, 2019.

[NKM17] Taesik Na, Jong Hwan Ko, and Saibal Mukhopadhyay. Cascade adversarial machine learning regularized with a unified embedding. *arXiv preprint arXiv:1708.02582*, 2017.

[PY95] Behraoz Parhami and Chi-Hsiang Yeh. Accumulative parallel counters. In *Conference Record of The Twenty-Ninth Asilomar Conference on Signals, Systems and Computers*, volume 2, pages 966–970. IEEE, 1995.

[PZJ+20] Daniel S Park, Yu Zhang, Ye Jia, Wei Han, Chung-Cheng Chiu, Bo Li, Yonghui Wu, and Quoc V Le. Improved noisy student training for automatic speech recognition. *arXiv preprint arXiv:2005.09629*, 2020.

[QLR+10] Weikang Qian, Xin Li, Marc D. Riedel, Kia Bazargan, and David J. Lilja. An architecture for fault-tolerant computation with stochastic logic. *IEEE transactions on computers*, 60(1):93–105, 2010.

[QR08] Weikang Qian and Marc D. Riedel. The synthesis of robust polynomial arithmetic with stochastic logic. In *2008 45th ACM/IEEE Design Automation Conference*, pages 648–653. IEEE, 2008.

[RHF19] Adnan Siraj Rakin, Zhezhi He, and Deliang Fan. Bit-flip attack: Crushing neural network with progressive bit search. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1211–1220, 2019.

[RLD+17] Ao Ren, Zhe Li, Caiwen Ding, Qinru Qiu, Yanzhi Wang, Ji Li, Xuehai Qian, and Bo Yuan. Sc-dcnn: Highly-scalable deep convolutional neural network using stochastic computing. *ACM SIGPLAN Notices*, 52(4):405–418, 2017.

[RSN+] Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, Alan Heckert, James Dray, and San Vo. A statistical test suite for random and pseudorandom number generators for cryptographic applications. `https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf`. Accessed: 2021-07-28.

[SGMA15] Kayode Sanni, Guillaume Garreau, Jamal Lottier Molin, and Andreas G Andreou. Fpga implementation of a deep belief network architecture for character recognition using stochastic computation. In *2015 49th Annual Conference on Information Sciences and Systems (CISS)*, pages 1–5. IEEE, 2015.

[SL17] Hyeonuk Sim and Jongeun Lee. A new stochastic computing multiplier with application to deep convolutional neural networks. In *Proceedings of the 54th Annual Design Automation Conference 2017*, pages 1–6, 2017.

[SNA+03] Shigeo Sato, Ken Nemoto, Shunsuke Akimoto, Mitsunaga Kinjo, and Koji Nakajima. Implementation of a new neurochip using stochastic logic. *IEEE Transactions on Neural Networks*, 14(5):1122–1127, 2003.

[SVS19] Jiawei Su, Danilo Vasconcellos Vargas, and Kouichi Sakurai. One pixel attack for fooling deep neural networks. *IEEE Transactions on Evolutionary Computation*, 23(5):828–841, 2019.

[SZS+13] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[TH14] Pai-Shun Ting and John P. Hayes. Stochastic logic realization of matrix operations. In *2014 17th Euromicro Conference on Digital System Design*, pages 356–364. IEEE, 2014.

[TH19] Paishun Ting and John P. Hayes. Exploiting randomness in stochastic computing. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6. IEEE, 2019.

[Vek20] Vivek Vrujlal Vekariya. Evaluating robustness of stochastic neural networks against adversarial learning attacks. Master's thesis, University of Stuttgart, 2020.

[VVF+15] Rangharajan Venkatesan, Swagath Venkataramani, Xuanyao Fong, Kaushik Roy, and Anand Raghunathan. Spintastic: Spin-based stochastic logic for energy-efficient computing. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1575–1578. IEEE, 2015.

[WHCE16] Ran Wang, Jie Han, Bruce F. Cockburn, and Duncan G. Elliott. Design, evaluation and fault-tolerance analysis of stochastic fir filters. *Microelectronics Reliability*, 57:111–127, 2016.

[XRV17] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *arXiv preprint arXiv:1708.07747*, 2017.

[XWZ+17] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. *arXiv preprint arXiv:1711.01991*, 2017.

[YHFQ17] Meng Yang, John P. Hayes, Deliang Fan, and Weikang Qian. Design of accurate stochastic number generators with noisy emerging devices for stochastic computing. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 638–644. IEEE, 2017.

[YKLC17] Joonsang Yu, Kyounghoon Kim, Jongeun Lee, and Kiyoung Choi. Accurate and efficient stochastic computing hardware for convolutional neural networks. In *2017 IEEE International Conference on Computer Design (ICCD)*, pages 105–112. IEEE, 2017.

[YLL+18] Meng Yang, Bingzhe Li, David J Lilja, Bo Yuan, and Weikang Qian. Towards theoretical cost limit of stochastic number generators for stochastic computing. In *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 154–159. IEEE, 2018.

[YW16] Bo Yuan and Yanzhi Wang. High-accuracy fir filter design using stochastic computing. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 128–133. IEEE, 2016.

[YWT+18] Tao Yang, Yadong Wei, Zhijun Tu, Haolun Zeng, Michel A. Kinsy, Nanning Zheng, and Pengju Ren. Design space exploration of neural network activation function circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(10):1974–1978, 2018.

[ZF14] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Appendix A

Appendix

A.1 Proof of NMax correctness

Proof of NMax correctness. Proving the correctness of NMax is done by induction over clock cycle $l \in \{1, \cdots, n\}$:

Induction hypothesis: At clock cycle l, NMax has output an SN MAX with the same value p_{MAX} as its maximum input value up to this cycle.

Induction basis: At clock cycle $l = 1$ all counters have been initialized to 0 ($c_j^1 = 0 \; \forall j$) and thus there can be two cases:

Case 1 (all inputs are 0):

$$
\begin{aligned}
& X_j^1 = 0 \; \forall j \\
\Rightarrow & O_j = 0 \; \forall j \\
\Rightarrow & Z = 0 \\
\Rightarrow & p_{MAX^1} = 0 = \max(X_1^1, \cdots, X_m^1) \text{ and } \mathrm{Inc}_j = 0 \text{ and } \mathrm{Dec}_j = 0 \; \forall j \\
\Rightarrow & c_j^2 = 0 \; \forall j
\end{aligned}
$$

Case 2 (at least one input is 1):

$$
\begin{aligned}
&\exists k \in 1, \cdots, j : X_k^1 = 1 \\
\Rightarrow &O_k = 1 \\
\Rightarrow &Z = 1 \\
\Rightarrow &p_{MAX^1} = 1 = p_{X_k^1} = \max(X_1^1, \cdots, X_m^1) \text{ and } \mathrm{Inc}_j = 1 \, \forall j \neq k \text{ and } \mathrm{Inc}_k = 0 \\
\Rightarrow &c_k^2 = 0 \text{ and } c_j^2 = 1 \, \forall j \neq k
\end{aligned}
$$

Induction step $l \rightarrow l+1$**:**

Circuit blocks are split into two cases: case 1 covers blocks with $c_j^{l+1} = 0$ and case 2 all blocks with $c_j^{l+1} > 0$.

Case 1 (all counter blocks with current count 0):

If $\exists X_k, k \in 1, \cdots, j$ with $X_k^{l+1} = 1$:

$$
\begin{aligned}
&O_k = 1 \\
\Rightarrow &Z = 1 \\
\Rightarrow &p_{MAX^{1,\cdots,l+1}} = \frac{p_{MAX^{1,\cdots,l}} \cdot l + 1}{l+1} \overset{\text{IC}}{=} \frac{p_{X_k^{1,\cdots,l}} \cdot l + 1}{l+1} \\
&= \frac{\max(X_1^{1,\cdots,l}, \cdots, X_m^{1,\cdots,l}) \cdot l + 1}{l+1} \\
&= \max(X_1^{1,\cdots,l+1}, \cdots, X_m^{1,\cdots,l+1}) \\
&\text{Furthermore } \mathrm{Inc}_k = 0 \wedge \mathrm{Dec}_k = 0 \Rightarrow c_k^{l+2} = 0
\end{aligned}
$$

Otherwise:

$$
\begin{aligned}
&O_j = 0 \, \forall j \\
\Rightarrow &Z = 0 \\
\Rightarrow &p_{MAX^{1,\cdots,l+1}} = \frac{p_{MAX^{1,\cdots,l}} \cdot l}{l+1} \overset{\text{IC}}{=} \frac{p_{X_j^{1,\cdots,l}} \cdot l}{l+1} \\
&= \frac{\max(X_1^{1,\cdots,l}, \cdots, X_m^{1,\cdots,l}) \cdot l}{l+1} \\
&= \max(X_1^{1,\cdots,l+1}, \cdots, X_m^{1,\cdots,l+1}) \text{ because all } X_j^{l+1} = 0 \\
&\text{Furthermore } \mathrm{Inc}_k = 0 \wedge \mathrm{Dec}_k = 0 \Rightarrow c_k^{l+2} = 0
\end{aligned}
$$

Case 2 (all counter blocks with current count > 0):

$\forall j$ with $X_j^{l+1} = 0$:

$O_j = 0$

$\Rightarrow Z$ is independent of X_j and therefore p_{MAX} is only determined by case 1.

If $\nexists k \neq j$ with $X_k^{l+1} = 1 \ \wedge \ c_k^{l+1} = 0$:

$Z = 0$ according to case 1

$\Rightarrow \text{Inc}_j = 0 \ \wedge \ \text{Dec}_j = 0$

$\Rightarrow c_j^{l+2} > 0$

Otherwise:

$Z = 1$ according to case 1

$\Rightarrow \text{Inc}_j = 1 \ \wedge \ \text{Dec}_j = 0$

$\Rightarrow c_j^{l+2} > 0$

$\forall j$ with $X_j^{l+1} = 1$:

$O_j = 0$

$\Rightarrow Z$ is independent of X_j and therefore p_{MAX} is only determined by case 1.

If $\nexists k \neq j$ with $X_k^{l+1} = 1 \ \wedge \ c_k^{l+1} = 0$:

$Z = 0$ according to case 1

$\Rightarrow \text{Inc}_j = 0 \ \wedge \ \text{Dec}_j = 1$

$\Rightarrow c_j^{l+2} \geq 0$

Otherwise:

$Z = 1$ according to case 1

$\Rightarrow \text{Inc}_j = 0 \ \wedge \ \text{Dec}_j = 0$

$\Rightarrow c_j^{l+2} > 0$

In conclusion:

- Only input SNs with associated counter values of 0 can affect the output of NMax and the output value p_{MAX} is always equal to the maximum of the input values.

- A counter will have a value of 0 if and only if its corresponding input SN has the current maximum value. Otherwise, a counter will have a value larger than 0.

□

A.2 Proof that the maximum function is convex

Proof that maximum is convex. The maximum function $\max(x, y)$, $x, y \in \mathbb{R}$ can be written as:

$$\max(x, y) = \frac{x + y}{2} + |\frac{x - y}{2}|$$

The function $\frac{x+y}{2}$ is linear and therefore convex (and concave). The absolute value function $|\cdot|$ is convex:
Let $a, b \in \mathbb{R}$ and $\alpha, \beta \in \mathbb{R}_{\geq 0}$ with $\alpha + \beta = 1$ and f the absolute value function, then:

$$\begin{aligned} f(\alpha \cdot a + \beta \cdot b) &= |\alpha \cdot a + \beta \cdot b| \\ &\leq |\alpha \cdot a| + |\beta \cdot b| \quad \text{(triangle inequality)} \\ &= |\alpha| \cdot |a| + |\beta| \cdot |b| \\ &= \alpha \cdot f(a) + \beta \cdot f(b) \end{aligned}$$

The maximum function is therefore a composition of convex functions and likewise convex.

□

A.3 Proof of equation 6.6

Proof of equation 6.6. Let X_i, $1 \leq i \leq n$ be independent, normally distributed random variables with mean μ and standard deviation σ, and let Z_i, $1 \leq i \leq n$ be independent, normally distributed random variables with mean 0 and standard deviation 1, such that $X_i = \mu + \sigma Z_i$.

Then $\max_i X_i = \mu + \sigma \max_i Z_i$. Therefore:

$$\mathrm{P}\left(\max_i X_i \leq \mu + t\sigma\right) = \mathrm{P}\left(\max_i Z_i \leq t\right) = \prod_i \mathrm{P}\left(Z_i \leq t\right) = \Phi(t)^n$$

With the substitution $u = \mu + t\sigma$, it follows that

$$\mathrm{P}\left(\max_i X_i \leq u\right) = \Phi\left(\frac{u-\mu}{\sigma}\right)^n.$$

The expected value of $\max_i X_i$ is defined as

$$\mathbb{E}\left(\max_i X_i\right) = \int_{-\infty}^{\infty} u \frac{d}{du} \Phi\left(\frac{u-\mu}{\sigma}\right)^n du$$

where $\frac{d}{du}\Phi\left(\frac{u-\mu}{\sigma}\right)^n$ is the probability density function (pdf) of $\max_i X_i$ (as the pdf is the derivative of the cumulative distribution function). With re-substitution of u and $du = \sigma \cdot dt$, and because the integral of the pdf over all $\mathbb{R}$ equals 1, this leads to equation 6.6:

$$\begin{aligned}
\mathbb{E}\left(\max_i X_i\right) &= \int_{-\infty}^{\infty} (\mu + t\sigma) \frac{d}{\sigma \cdot dt} \Phi(t)^n \sigma \cdot dt \\
&= \int_{-\infty}^{\infty} \left(\mu \frac{d}{dt}\Phi(t)^n + t\sigma \frac{d}{dt}\Phi(t)^n\right) dt \\
&= \mu \int_{-\infty}^{\infty} \frac{d}{dt}\Phi(t)^n dt + \sigma \int_{-\infty}^{\infty} t \frac{d}{dt}\Phi(t)^n dt \\
&= \mu + \sigma \int_{-\infty}^{\infty} t \frac{d}{dt}\Phi(t)^n dt
\end{aligned}$$

□

Publications of the Author

In the following, all former publications of the author are listed.

Journal Publications

[NPH18a] Florian Neugebauer, Ilia Polian, and John P. Hayes. S-box-based random number generation for stochastic computing. *Microprocessors and Microsystems*, 61:316–326, 2018.

[NPH18b] Florian Neugebauer, Ilia Polian, and John P Hayes. Framework for quantifying and managing accuracy in stochastic circuit design. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 14(2):1–21, 2018.

Conference Proceedings

[MNPH20] Ponnanna Kelettira Muthappa, Florian Neugebauer, Ilia Polian, and John P. Hayes. Hardware-based fast real-time image classification with stochastic computing. In *2020 IEEE 38th International Conference on Computer Design (ICCD)*, pages 340–347. IEEE, 2020.

[NPH19] Florian Neugebauer, Ilia Polian, and John P. Hayes. On the maximum function in stochastic computing. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, pages 59–66, 2019.

[NPH17] Florian Neugebauer, Ilia Polian, and John P. Hayes. Building a better random number generator for stochastic computing. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 1–8. IEEE, 2017.

[NHP22] Florian Neugebauer, Stefan Holst, and Ilia Polian. On the impact of hardware timing errors on stochastic computing based neural networks. In *2022 IEEE European Test Symposium (ETS)*, pages 1–6. IEEE, 2022.

[NPH19] Florian Neugebauer, Ilia Polian, and John P. Hayes. On the limits of stochastic computing. In *2019 IEEE International Conference on Rebooting Computing (ICRC)*, pages 1–8. IEEE, 2019.

[NPH17] Florian Neugebauer, Ilia Polian, and John P. Hayes. Framework for quantifying and managing accuracy in stochastic circuit design. In *Design, Automation & Test in Europe (DATE), 2017*, pages 1–6, 2017.

[ONPH20] Junseok Oh, Florian Neugebauer, Ilia Polian, and John P. Hayes. Retraining and regularization to optimize neural networks for stochastic computing. In *2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 246–251. IEEE, 2020.

Other Publications

[Neu16] Florian Neugebauer. Quantifying accuracy in stochastic circuits. Master's thesis, University of Passau, 2016.

Declaration

All the work contained within this thesis,
except where otherwise acknowledged, was
solely the effort of the author.
At no stage was any collaboration entered into
with any other party.

Florian Neugebauer