Fault Tolerance of Stochastic Decoders for Error Correcting Codes

by

Assem Shoukry Mohamed Hussein

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2017

Abstract

Low-Density Parity-Check (LDPC) codes are very powerful linear error-correcting codes, first introduced by Gallager in 1963. They are now used in many communication standards due to their ability to achieve near Shannon-capacity performance. Stochastic decoding is a hardware-efficient method of iterative decoding of LDPC codes. In this work, we investigate the capability of stochastic decoding to tolerate circuit soft errors while maintaining good bit error rate performance and low error floor. Soft errors can be intended faults as a result of either supply voltage scaling to reduce power consumption or overclocking the system to achieve a higher throughput. They can also be unintended faults as a result of temperature or process variations.

We develop two models to emulate these circuit errors at the system level. We apply our models to two standardized LDPC codes (10GBASE-T and WiMAX). Simulation results show that stochastic decoding is very tolerant to faults and errors, where it can tolerate a probability of setup time violation of 0.1 in the wires of the decoder. Hence, stochastic decoding can be very useful in systems with very low power or high performance requirements where we can push the limits of power or speed by lowering the supply voltage or highly overclocking the system while maintaining good performance. In addition, a chip has been designed and sent to fabrication to do post-silicon validation and verify our models.

Acknowledgements

I would like to express my sincere gratitude and appreciation to my supervisors, Professor Vincent Gaudet and Professor Mohamed Elmasry for their continuous support and encouragement during my research and academic work. This work wouldn't have been successful without your advice and guidance.

I would also like to thank Professor Mark Aagaard and Professor Liang-Liang Xie for reviewing my thesis and providing valuable feedback and notable comments. In addition, I would like to thank Professor Adel Sedra for his invaluable discussions and for his help and guidance.

I would like to express my gratitude to everyone who helped with the chip implementation and the CAD tools especially Professor Mark Aagaard, Professor Manoj Sachdev, Govindakrishnan Radhakrishnan, Sakib Imtiaz, Yahya Beltagy and Mohamed Hassan. Thanks so much for your valuable discussions during the chip Tape-Out. In addition, I would like to express my thanks to my research group and lab mates, Rachna Srivastava and Maurice Yang. It has been a great pleasure to work with you. Moreover, I would like to thank my professors and teaching assistants at Cairo University for their efforts during my undergraduate study, especially, Dr. Hassan Mostafa for his continuous advice and guidance.

My deepest gratitude and appreciation goes to my mother, Asmaa El Hakim. No words are enough to even describe how important and special you are in my life. I am forever grateful for your endless support and guidance throughout my life. I would also like to thank my father, Shoukry Hussein. Thanks for all the sacrifices you have made, for everything you have given me and for always believing in my capabilities. Moreover, a special thank you to my brother, Mohamed Hussein, for his continuous support, advice and guidance. I couldn't wish for a better family.

In addition, I would like to acknowledge the Natural Sciences and Engineering Research Council (NSERC) as this research is supported in part by an NSERC grant to my co-supervisor, Professor M. I. Elmasry. I would also like to thank the Canadian Microelectronics Corporation (CMC) for their help and support with the chip tape-out.

Finally, I would like to take this opportunity to thank everyone who ever taught me in my life. I would have never reached anything without you.

Dedication

This is dedicated to my family with love.

Table of Contents

List of Figures

Chapter 1

Introduction

In 2003, it was shown that stochastic computing [1] is a promising technique for iterative decoding of capacity-approaching error-control codes [2], which were conventionally decoded using algorithms such as the sum-product algorithm (SPA) [3]. Subsequently, there was a lot of work into getting stochastic decoding to work for codes such as low-density parity-check (LDPC) codes [4,5], requiring novel approaches such as noise-dependent scaling (NDS) [6], edge memories (EMs) [6], and tracking forecast memories (TFMs) [7–9]. Since then, there has been a lot of work done to design hardware-efficient stochastic LDPC decoders (see, e.g., [9–19]). More recently, this has re-invigorated interest in stochastic computing in general.

One of the notable features of stochastic computing is its robustness to errors and noise. Stochastic computing systems perform calculations on streams of randomly generated bits (Bernoulli sequences), and we are more interested in the statistical behaviour of the output rather than on computing a certain specific value with arbitrarily large precision. Flipping a few bits in the input stochastic stream should not affect the result in a significant way. This can lead to compelling advantages when applied to error-control codes.

In this thesis, we explore the error tolerance of stochastic decoding. We develop two models where we insert errors by flipping bits in messages exchanged between variable nodes and check nodes in LDPC decoders. These models emulate soft errors that might occur in hardware decoders (e.g., setup time violations).

Two standardized codes are used in our studies. The first is the (2048,1723) LDPC code from the 10GBASE-T standard [20]. The second is the (1056,704) LDPC code from the IEEE 802.16e WiMAX standard [21]. We pick these two different LDPC codes as they have different properties so that we can have a more general overview of the fault tolerance of stochastic decoding. The (2048,1723) code is regular while the (1056,704) is irregular.

Simulation results show that stochastic decoding has a strong tolerance to soft errors compared to other decoding algorithms. This is because messages are single bits for stochastic decoders while they are n-bit likelihoods in other decoding algorithms; consequently, a "most-significant bit" error in a conventional decoder can have a strong impact. In addition, in stochastic decoding, probabilities converge over time with decoding cycles, so flipping a single bit results in only a very small deviation in the probability at the worst case. In other decoding algorithms, where the messages transferred between the nodes are probabilities with n number of bits, flipping one bit is highly dependent on the magnitude of the flipped bit. The worst case happens when flipping the most significant bit in a propagating probability.

Our simulation results show that we can decrease the design constraints on stochastic decoders and push the limits of power or speed by lowering V_{DD} or by highly overclocking the system while maintaining a good BER. They also show that stochastic decoders are good solutions for applications with noisy environments (e.g., nanoscale technologies) or low power requirements.

The rest of the thesis is organized as follows. Chapter 2 gives a brief review about stochastic computing. Chapter 3 discusses the stochastic decoding of LDPC codes and describes the general stochastic LDPC decoder architecture. Chapter 4 demonstrates the model used to emulate the circuit faults. Then, Chapter 5 presents the simulation results of the performance of the decoders for different case scenarios. Next, Chapter 6 shows the detailed architecture of the implemented chip. Finally, Chapter 7 concludes our work.

Chapter 2

Stochastic Computing

2.1 Historical Background

Stochastic computing was first introduced in the 1950s by Von Neumann [22]. During the 1960s and the 1970s, stochastic computing gained a lot of interest and much work has been done in that area. In 1969, Esch introduced a programmable analog computer based stochastic computing techniques [23]. During this time, research was directed towards developing stochastic computing processors [24]. An International Symposium on Stochastic Computing was held in France in 1978. It was the first symposium on stochastic computing and unfortunately, the last. Over the next years, conventional digital logic circuits overpowered stochastic computing techniques for general computing purposes because of the stochastic computing weaknesses that we will mention, later on.

Recently, stochastic computing has showed great promising results in specific applications. These applications include image processing applications [25], neural decoding [26] and stochastic decoding of error correcting codes [27].

2.2 Stochastic Representation

In stochastic computing, numbers are represented as stochastic streams of bits of length N. A value can be represented by the number of ones in the stochastic stream. This is called the unipolar format. It represents a number in the [0,1] interval which can be interpreted as a probability. There are many other formats proposed in the literature like the bipolar, inverted bipolar and the numbers represented by the ratio of ones to zeros [28, 29]. In this thesis, we will be working with the unipolar format as the LDPC decoder deals mainly with probabilities as inputs to it. This will be explained in Chapter 3. Now, we will introduce the notations that will be used later on:

- P is an L-bit binary number
- P_{max} is an all-one L-bit binary number
- S is an N-bit stochastic stream, where $S = \{S_0, S_1, \ldots, S_i, \ldots, S_{N-1}\}$, $0 \leq i < N - 1$.
- N is the length of stochastic stream

S is a Bernoulli sequence generated from a Bernoulli process. The probability of observing a 1 at any bit of the stochastic stream is equal to:

$$E[S] = P(S_i = 1) = \frac{P}{P_{max}} = p = \sum_{i=0}^{N-1} S_i \times p_i \tag{2.1}$$

We use a big N, so that according to the law of large numbers (LLN), the average (mean) value of the sequence S approaches p:

$$\bar{S} = p \tag{2.2}$$

$$\because \bar{S} \equiv \frac{1}{N} \times \sum_{i=0}^{N-1} S_i \tag{2.3}$$

$$\therefore p = \frac{1}{N} \times \sum_{i=0}^{N-1} S_i \tag{2.4}$$

From equation 2.4, we conclude that the number of ones in a stochastic stream divided by the total number of bits in this stream represent the number P.

Because sequences are generated using a stochastic process, different sequences S can be generated for a given P. This is not a problem because we are more interested in the statistics of the stochastic stream rather than the exact order of ones and zeros. The accuracy of the stochastic representation S of a value P is dependent on the length N of the stochastic stream. This is discussed in the example below.

Example 1: In this example, we show the impact of the length stochastic stream, N, on the accuracy of the stochastic stream representation, where:

- $L = 4$,
- $P = (8)_{10} = (1000)_2$,
- $P_{max} = (15)_1 0 = (1111)_2$.
- N is tested for 3 different values = 100, 1000 and 10000.

For every value of N we generate 100,000 stochastic streams. For each stochastic stream, we are interested in the average value of the sequence S which is calculated from equation 2.1. Next, we collect these values for each N value and plot the probability density function (PDF) shown in Figure 2.2. Since P="1000", then the expected value according to equation 2.1 is equal to 0.5333.

Figure 2.1 shows that the PDF will be a normal (Gaussian) distribution with mean (μ) equal to the expected value of 0.5333. As we increase the length of the stochastic stream N, the standard deviation (σ) gets smaller and the stochastic representation becomes more accurate. If N approaches infinity, the resultant PDF will approach the Dirac Delta function.

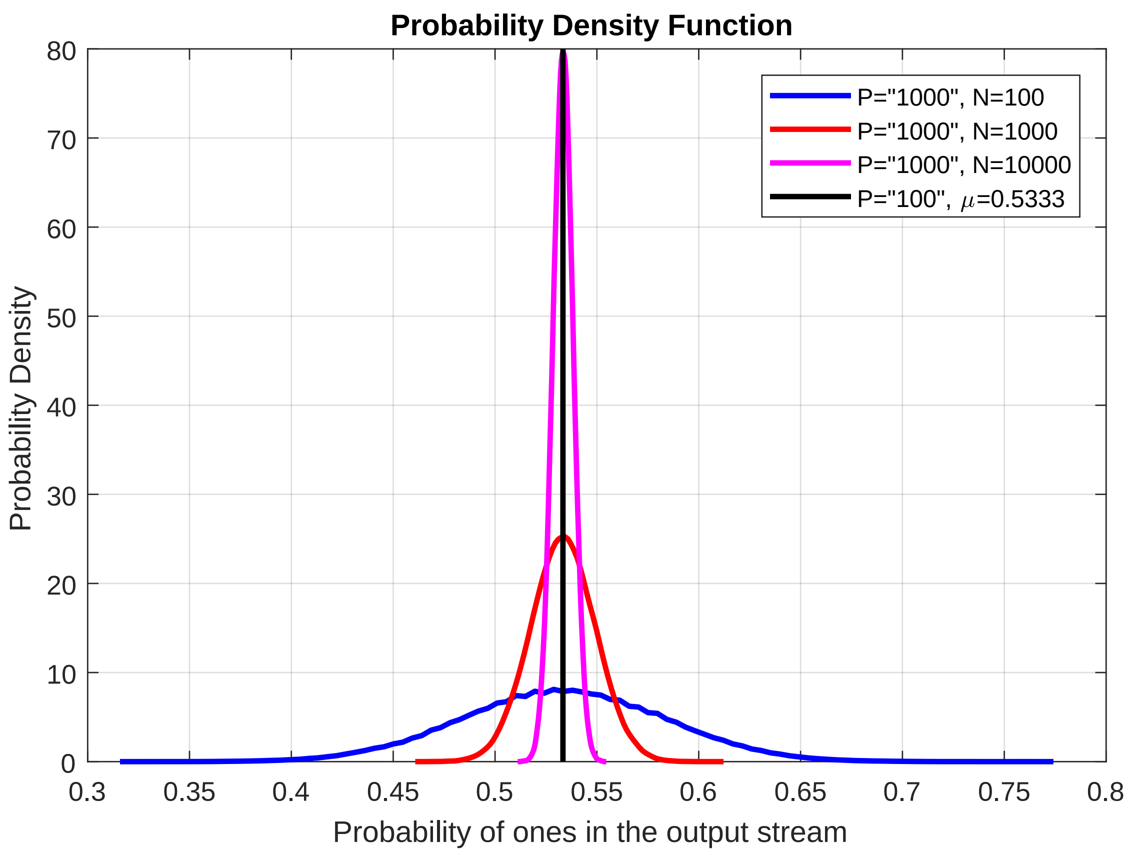

Figure 2.1: Probability density function of the average probability of 100,000 stochastic streams for each of the three values of N= 100, 1000 and 10000 at P= “1000”.

Next, we want to investigate the possibility that the stochastic stream misrepresents the underlying binary number. Consequently, we generate 100,000 stochastic streams of the same length $N = 100$ for two consecutive binary numbers, $P = (0111)_2$ and $P = (1000)_2$. Figure 2.2 shows the PDF of the mean values of the resultant stochastic stream for each of the two binary numbers.

Figure 2.2 shows that 2 numbers, once represented as stochastic streams with $N = 100$, likely cannot be resolved. The probabilities lying under the common area between the blue curve and the red curve cannot be distinguished. This is because of the large value

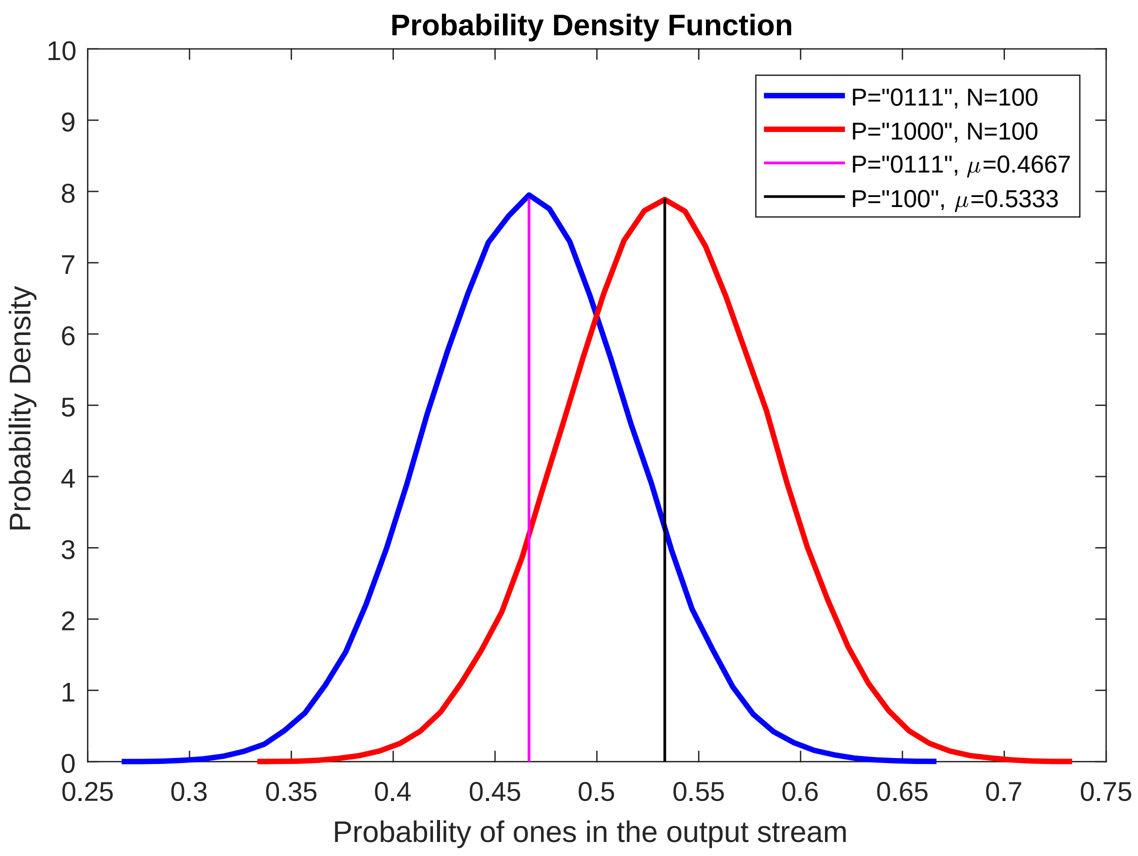

Figure 2.2: Probability density function of the mean values of 100,000 stochastic streams for N= 100 at P= "0111" and P= "1000".

of the standard deviation of both of the two distributions. We don't know whether they represent $P = (0111)_2$ or $P = (1000)_2$. When increasing the length of the stochastic stream to $N = 1000$, this common shared area under both curves significantly decreases and standard deviation of each distribution becomes much smaller as in Figure 2.3. Hence, we the stochastic streams become more accurate in representing the underlying binary number. We conclude that the longer the stochastic stream, the more accurate it is representing the binary number.

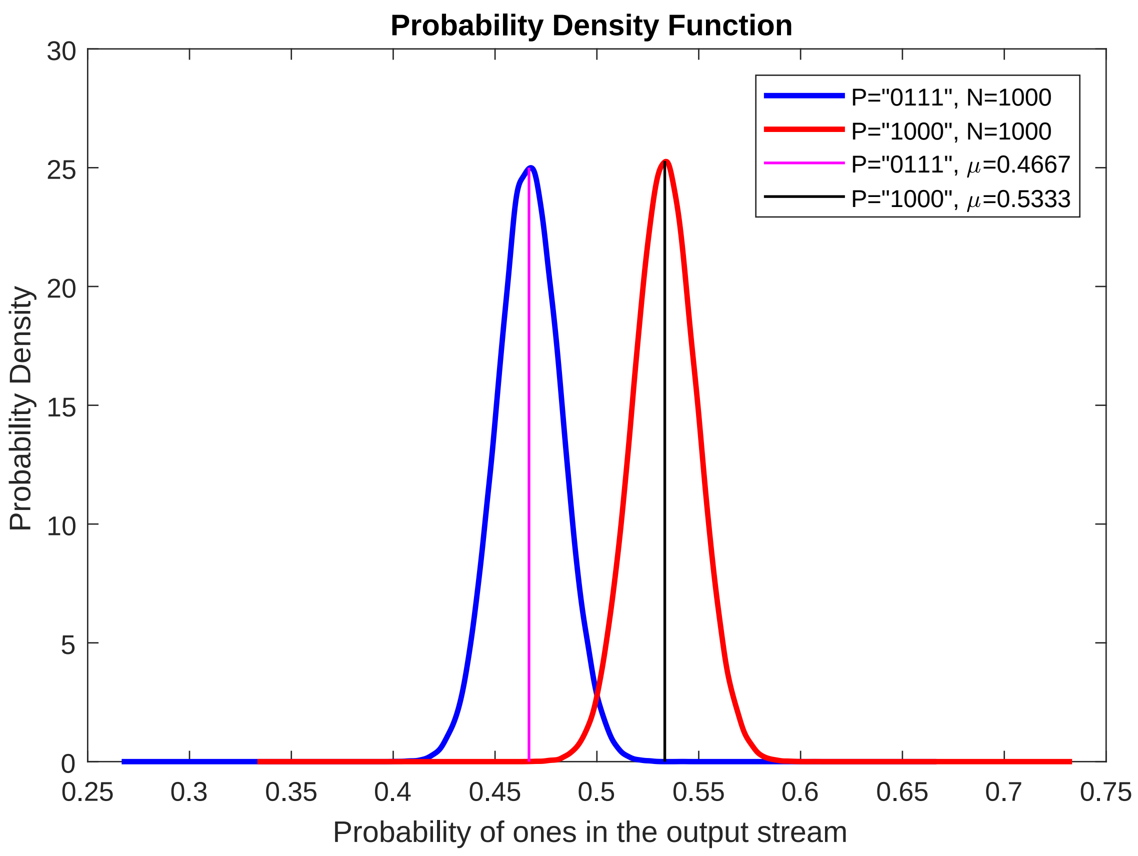

Figure 2.3: : Probability density function of the average probability of 100,000 stochastic streams for N= 1000 for P= "0111" and P= "1000".

2.3 Stochastic Computing Circuits

In this section, we will introduce some of the stochastic computing circuits. We will start with the binary to stochastic converter.

2.3.1 Binary-to-Stochastic Converter

Figure 2.4 shows a circuit that is often used to convert an L-bit binary number to a stochastic stream S of length N bits, where R is an L-bit Pseudo Random Number. R_i is

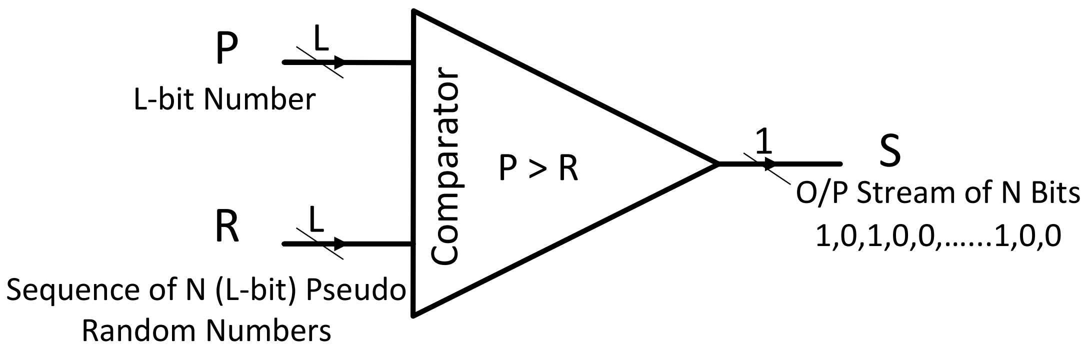

Figure 2.4: L-bit Binary-to-Stochastic Converter.

changed each iteration using a uniform pseudo random number generator. In the circuit, P is compared to R_i, where R_i is changed each iteration using a uniform pseudo random number generator. The output is equal to 1 if P is greater than R_i.

$$S_i = \begin{cases} 0, & \text{if } P \leq R_i \\ 1, & \text{if } P > R_i \end{cases} \tag{2.5}$$

This results in an L-bit stream of ones and zeros. This conversion is a Bernoulli process. Hence, S is Bernoulli sequence.

2.3.2 Arithmetic Operations

Researchers developed stochastic circuits that perform arithmetic operations such as addition, multiplication, division, etc [30]. We will only show the multiplication circuit as an example that we will use in the next section to compare between stochastic computing and conventional computing techniques. Figure 2.5 shows the circuit used to perform the stochastic multiplication. The two input stochastic streams are X and Y. The output stochastic stream is Z, where:

$$P_X = P(X_i = 1) = \frac{4}{5} \tag{2.6}$$

$$P_Y = P(Y_i = 1) = \frac{1}{2} \tag{2.7}$$

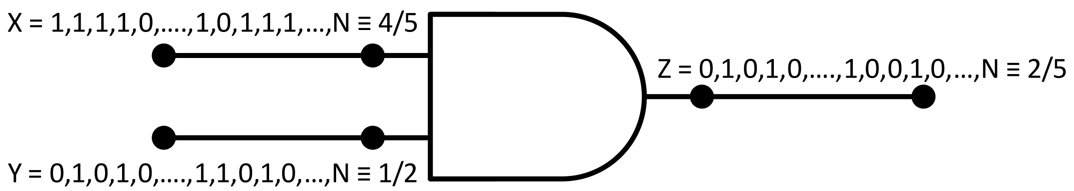
X = 1,1,1,1,0,....,1,0,1,1,1,...,N ≡ 4/5
Y = 0,1,0,1,0,....,1,1,0,1,0,...,N ≡ 1/2
Z = 0,1,0,1,0,....,1,0,0,1,0,...,N ≡ 2/5

Figure 2.5: Stochastic Multiplier.

From Figure 2.5:

$$\because Z_i = 1 \text{ if } \big((X_i = 1) \text{ AND } (Y_i = 1)\big) \tag{2.8}$$

$$\therefore P_Z = P(Z_i = 1) = P_X \times P_Y = \frac{2}{5} \tag{2.9}$$

Consequently, an AND gate can be used to implement a stochastic multiplier after converting the binary numbers to stochastic streams. However, this is assuming that X and Y are two independent stochastic streams.

2.4 Stochastic Computing Versus Traditional Digital Computing Methods

Stochastic computing is an excellent option compared to traditional digital computing methods in many applications due to its features and advantages. However, it dramatically fails in other applications due to its critical drawbacks for these applications. In this section, we will discuss the advantages and disadvantages of stochastic computing compared to traditional digital circuits. This will then introduce us to the main topic of the thesis which is stochastic decoding.

2.4.1 Circuit Complexity

The first advantage of stochastic computing is the low complexity of its underlying circuits in many arithmetic calculations. Low circuit complexity means lower number of gates. This results in lower number of transistors and consequently, smaller die area and lower cost. One example on this is the multiplication operation. Binary multiplication in hardware usually requires using big circuits of full adders, etc [31]. In addition, the more we increase the number of bits per input, the more complex the circuit becomes. However, in stochastic computing we only need an AND gate to perform the multiplication operation, as shown in Figure 2.5. Moreover, increasing the number of bits per input does not increase the complexity of the circuit. So, it is considered low-cost and area-efficient solution.

2.4.2 Fault Tolerance

One more important advantage to stochastic computing is its robustness to noise. The stochastic input is fed into the underlying stochastic circuit as a long stochastic stream of bits. The value of the output is determined by the statistics of the output stream of bits rather than the exact value of the bits. Consequently, having very few bits flipped in the input should not significantly affect the output compared to conventional exact calculation techniques. Taking the example in Figure 2.5 of the stochastic multiplier, assume that the input stochastic streams are 1000 bits long (i.e. $N = 1000$). Consequently, flipping one bit in one of the input streams, (X,Y) might not significantly affect the output probability. On the other hand, in conventional arithmetic multipliers, 1-bit flip is highly dependent on the order of the bit. If the flipped bit is the most significant bit, the amount of error in the output is large. However, in stochastic computing, the error is minimal.

2.4.3 Progressive Precision

In addition, stochastic computing provides a rapid rough estimate to the output. This estimate then keeps getting accurate with more samples being exploited in the calculation. Using the example in Figure 2.5 of a stochastic multiplier, we can get a fast estimate of the output with the first incoming bits. On the other hand, conventional multipliers usually output the least significant bits before the most significant bits. As a result, we cannot know an estimate of the output before getting the most significant bit in the last step as this is most important bit in determining the estimate. This early rough estimate in stochastic computing is known as progressive precision [27].

2.4.4 Tolerance to Skew in Input Streams

Finally, stochastic computing is tolerant to possible skew in the arrival times of the input streams of bits to the stochastic circuit. This is because we are more concerned about the statistics of the input stream of bits rather than the exact index or location of the bit. This can decrease the complexity of the system as we don't need to spend a lot of resources on expensive clock synchronization techniques.

On the other hand, Stochastic computing has some weaknesses:

2.4.5 Accuracy

The first is the precision. For a confined stochastic stream of length N, where N is not large enough, stochastic computing is not able to provide the same level of accuracy as the exact conventional computational methods as shown in Figure 2.2. The accuracy of the stochastic computations is highly dependent on the length of the generated stochastic streams. The longer the stochastic streams, the better accuracy you will get. Increasing accuracy requires an exponential increase in the length of the stochastic stream of bits. This implies an exponential increase in the delay [27].

2.4.6 Possible Dependency Between Stochastic Streams

In addition, one disadvantage is the possible dependency between the stochastic streams [27]. Ideally, we assumed that the input stochastic streams to the stochastic circuit are independent. However, if they are dependent, stochastic computing can totally fail in giving the correct outputs. Figure 2.6 shows a stochastic multiplier with dependent input stochastic streams to clarify this idea. Assume we are trying to get the square of a number, if we used the same random number generator once to generate the same stochastic stream, then the two input stochastic streams are identical and the AND gate will work as if its two inputs are shorted. The output in this case will be equal to the input not its square. This example is just to clarify the problem of the dependency between the stochastic streams. The solution here is simple which is generating the stochastic streams by two independent random number generators. However, in big complex systems with feedback structures, the system becomes vulnerable to the correlation between the stochastic streams. Hence, much effort has to be put to decorrelate the stochastic streams and ensure an acceptable level of independence.

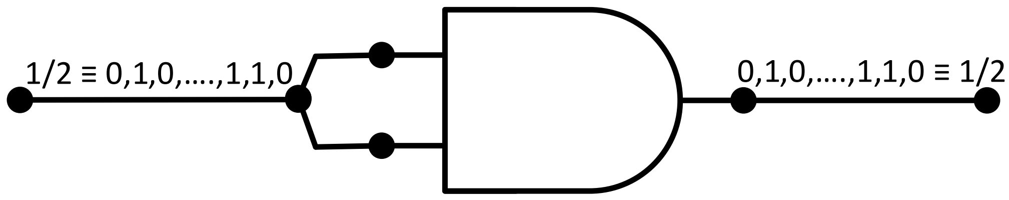

Figure 2.6: Stochastic Multiplier with Dependant Inputs.

2.4.7 Latency

One disadvantage that might show up is the latency of the stochastic computing circuits. However, that is dependent on the application and the complexity of the circuit, as well. The reason behind the possible long latency of the stochastic computations is the length of the stochastic streams. As discussed in the Stochastic "Representation" section, the accuracy of the stochastic computations is dependent on the length of the stochastic stream N. Here comes the tradeoff between accuracy, circuit complexity and latency. However, in some specific applications, where very high precision is not required, and due to the much-improved circuit complexity, the latency in stochastic computing might still be better than traditional digital computing circuits. An L-bit multiplier can be an example on that in case of a large L.

2.4.8 Power

Stochastic computing circuits in general exhibit high switching activity in its circuits. This is due to the long stochastic streams that might have many bit-flips. However, that is also dependent on the application where one is using stochastic computing. An example on that is applications without a high-precision specification, and as a result, short stochastic streams. Moreover, some applications exploiting stochastic computing might have much lower gate count than traditional digital circuits and as a result, consume less power than them. The multiplier circuit is an example on that, specially, if it is a 2 L-bit multiplier where L is large. In addition, some applications like stochastic decoding, have low switching activity and don't have many bit flips during the computations. As a result, the power consumption might be lower than traditional digital logic circuits.

2.4.9 Random Number Generators

Finally, we have to consider the cost of the random number generators (RNGs) used to generate the stochastic streams. There are many ways to implement RNGs. One of the famous simple methods of implementing a pseudo random number generator is the linear feedback shift register.

Depending on the number of inputs and the size of the conventional non-stochastic circuit, stochastic computing may lose the merit of being a low-cost solution because of the overhead cost of the pseudo random number generators. However, the size of the RNGs in many stochastic computing applications is negligible compared to the circuit size. One example on that is the chip fabricated in [15]. The RNGs consume 1% of the total gate count of the circuit and only 0.3% of the power consumption.

2.5 Applications of Stochastic Computing

Stochastic Computing is being exploited in many applications. The most important of these are:

2.5.1 Stochastic LDPC Decoding

In 2003, stochastic computing arose as a promising technique to implement the iterative decoding algorithm of the LDPC codes [2]. A lot of work has been done, since then, to implement high throughput, low power and high performance LDPC decoders [8,9,9–19]. The stochastic LDPC decoder will be described in detail through the next chapters in the thesis.

2.5.2 Neural Networks

Recently stochastic computing has been exploited in the efficient implementation of Neural networks and deep learning systems [26,32–34].

2.5.3 Image Processing

Image processing is a promising field for stochastic computation. Many applications in image processing require performing operations on every input pixel of an image. For example, the functional transformation on the image input pixels [27]. These operations are usually simple, but because they are performed on a large number of pixels, the amount of computations becomes significantly high. Consequently, if we can perform these calculations using a low-cost computing method like stochastic computing, we will be able to build efficient highly parallel circuits that can perform the image processing tasks [25,35].

2.5.4 General Arithmetic Functions

The research community was interested in exploiting stochastic computing to implement general purpose circuits since stochastic computing first appeared in the 1960s till late 1970s. One example on that is in [23]. However, later on, conventional digital logic overpowered stochastic computing for the reasons outlined in the disadvantages. Recently stochastic computing was revived to compute some general arithmetic functions and polynomials in [36–38]. Some of this work was later used in image processing applications.

Chapter 3

Stochastic LDPC Decoding

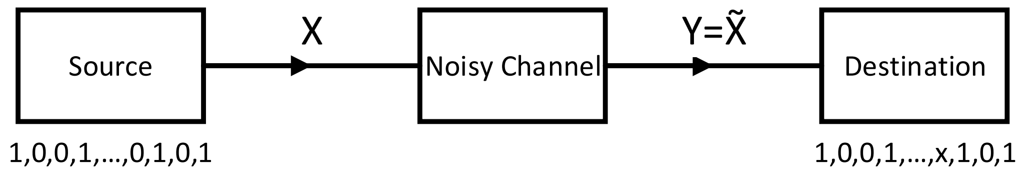

Figure 3.1: Basic Communication System.

In communication systems, the main objective is to transfer data from a source to a destination over an imperfect channel with minimal loss of data. The theoretical maximum capacity of a channel is described by the ShannonHartley theorem [39]. If data is directly transmitted over the channel without encoding and decoding as shown in Figure 3.1, any data lost during the transmission cannot be recovered. As a result, we encode the data before sending it over the channel and decode them after being received. This is done by adding redundant bits so that we can recover bits lost due to the noise. The full system is shown in Figure 3.2. The modulator and demodulator blocks convert the data bits into symbols that can be transferred over the channel.

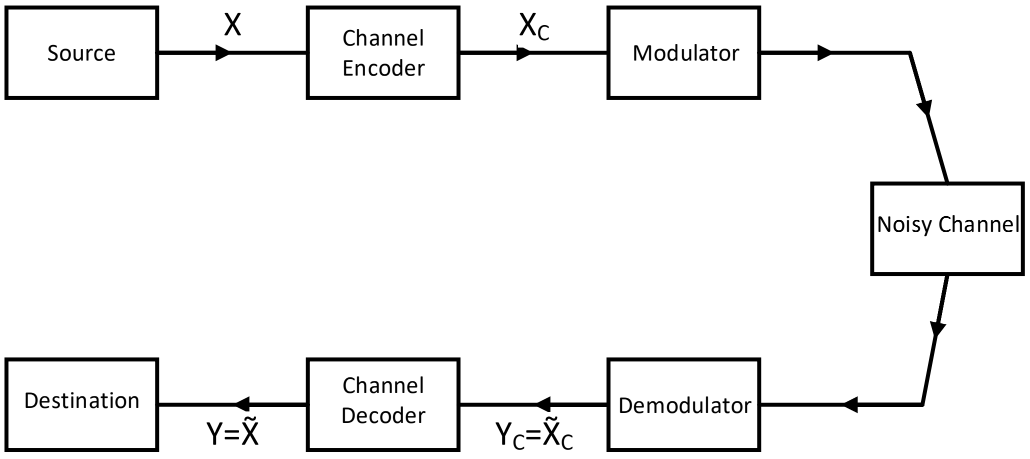

Figure 3.2: Communication System with Modulation and Channel Coding.

3.1 Channel Coding

Channel encoding and decoding has two inter-related tasks:

1. Error detection: to detect if the received message is correct or some bits are corrupted by noise.
2. Error correction: to correct any corrupted bits received in the message.

The idea behind channel coding is to add redundant bits to the message that can be used to detect and possibly correct errors.

An example is shown in Figure 3.1 and Figure 3.3. In the system shown in Figure 3.1, suppose we want to transmit 8 data bits $X = \{x_1, x_2, \ldots, x_8\} = \{1, 0, 0, 1, 0, 1, 0, 1\}$. If the data bits are transmitted without encoding and the 5th bit x_5 was corrupted and received as $y_5 = \tilde{x}_5 = 1$ rather than a zero, $Y_c = \tilde{X}_c = \{1, 0, 0, 1, 1, 1, 0, 1\}$, we will never be able to detect that there is an error in one of the bits. However, in Figure 3.3, we use one simple form of encoding by generating an even parity check bit to the data bits,

$X = \{1,0,0,1,0,1,0,1\}$ and adding it to the message. So that $X_c = \{1,0,0,1,0,1,0,1,0\}$. Next, the message is modulated and transmitted over the channel to the receiver side, then demodulated. Assume that the noise in the channel had a significant effect on the 5th bit of the message, x_5. If we applied the even parity check equation on the message bits, we can then deduce that there is one corrupted bit. With this simple encoding method, we certainly know that there is at least one corrupted bit but we can not detect which bit is wrong. Moreover, of course, we can not correct it. Now, assume we know the order of the bit $\tilde{x}_{c5}$ that has a possible error. If we again apply the parity check equation, we can infer that the original correct bit was 0 not 1.

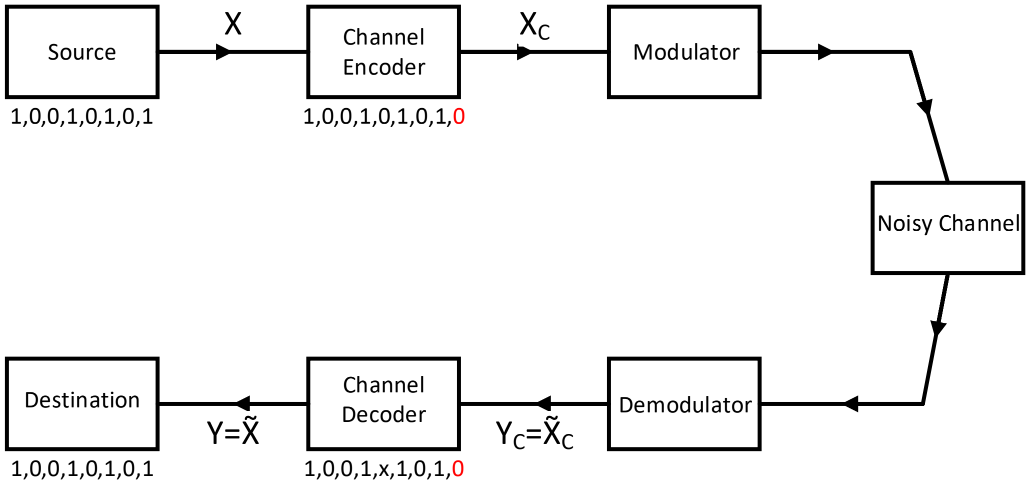

Figure 3.3: Communication System with Modulation and Channel Coding Capable of Correcting Errors.

3.2 Channel Noise Model

For simplicity, the model we use in this thesis for the noise in the channel is the Additive White Gaussian Noise. The input message, X is affected by the noise signals Z and the received message is Y. Then as shown in Figure 3.4:

$$Y_i = X_i + Z_i, \text{ where } Z \sim \mathcal{N}(0, \sigma_N^2) \tag{3.1}$$

Z_i represents a noise sample drawn from a normal Gaussian distribution with zero mean and variance σ_N^2, dependent on the noise power. The model also assumes that the noise samples are independent.

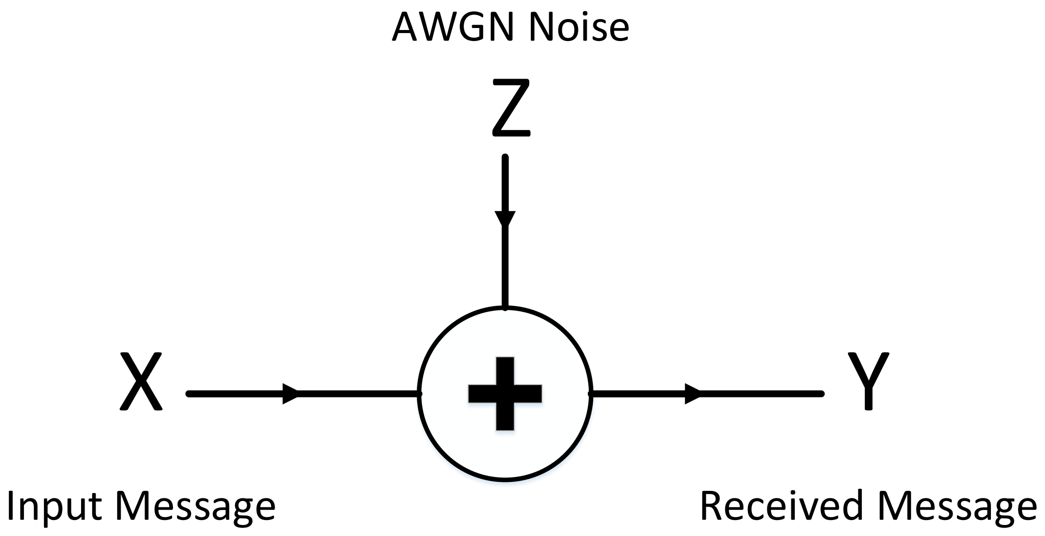

Figure 3.4: AWGN Noise Model.

3.3 Error-Correcting Codes

Generally, in Error-Correcting Codes (ECC), we add redundant bits to the original message, X, so that we can detect and correct possible corrupted bits. Two of the major types of error correcting codes are:

1. Convolutional Codes

   In convolutional codes, data are processed bit by bit. Convolutional codes are usually soft decoded using the Viterbi decoder.

2. Block Codes

   Block codes are processed block by block, where each block has a fixed size.

Error-correcting codes are capable of correcting errors up to a certain limit. This limit is known as Shannon-capacity limit. Shannon's theorem proposes a maximum efficiency for a code by describing a maximum data rate for the code at a certain Signal-to-Noise ratio (SNR) such that the message is transmitted with a significant low bit-error rate (BER). Turbo codes and LDPC codes are examples on ECC. They are very efficient codes capable of approaching the theoretical Shannon-capacity limit.

3.4 LDPC Codes

Low-Density Parity-Check (LDPC) codes are some types of linear error correcting codes. LDPC codes are very efficient codes that can approach the theoretical Shannon-capacity limit. They are currently being exploited in many applications and communication standards due to their very good performance compared to other codes.

Generally, a (n, k) LDPC code can be represented by bi-partite graph of n variable nodes (VN) and (n-k) parity check nodes (CN) [3] as shown in Figure 3.5. Connections between variable nodes and check nodes are described by a parity check matrix H like the one shown in Figure 3.5 where:

$$HX_c^\top = 0 \tag{3.2}$$

A degree of a VN, d_v, is the number of connections form this VN to different CNs in the factor graph. Same for the degree of a CN, d_c, which is equal to the number of connections form this CN to different VNs.

The H-matrix shown in equation 3.3 describes the connections between the VNs and the CNs. Each column represents a VN and each row represents a CN. For a certain element h_{ij} in the H-matrix, if $h_{ij} = 1$, this means that the VN, j is connected to the CN, i. If

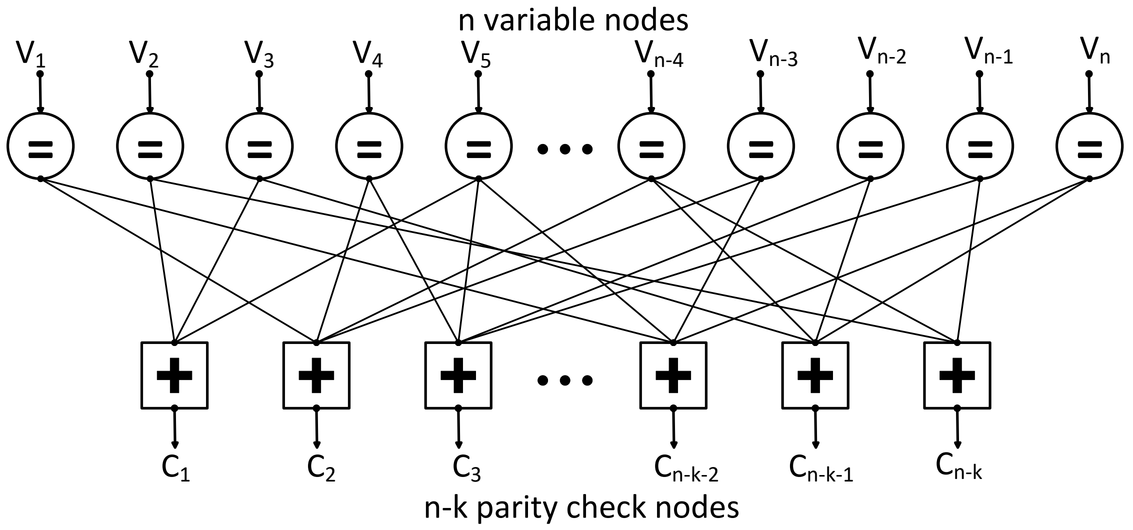

Figure 3.5: A Factor Graph from an (n,k) LDPC Code with n Variable Nodes and n-k Parity Check Nodes.

$h_{ij} = 0$ then there is no connection between VN, j and CN, i. The number of ones in a row is equal to d_c and the number of ones in a column is equal to d_v.

$$H = \begin{pmatrix} h_{1,1} & h_{1,2} & \cdots & h_{1,n} \\ h_{2,1} & h_{2,2} & \cdots & h_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ h_{n-k,1} & h_{n-k,2} & \cdots & h_{n-k,n} \end{pmatrix} \tag{3.3}$$

For regular codes, all the variable nodes have the same degree, d_v and all the check nodes have the same degree, d_c. While for irregular codes, VNs can have different degrees and CNs can also have different degrees.

3.5 Modulation Scheme

The modulation scheme used throughout the thesis is the Binary Phase-shift keying (BPSK). The zero bit is modulated as 1 and the one bit is modulated as -1. Other modulation schemes can be used. The only difference will only be in the method of calculation of the probabilities fed into the decoder.

3.6 LDPC Decoding Algorithms

Many iterative decoding algorithms are capable of decoding LDPC codes based on the iterative belief propagation of messages like the Sum-Product Algorithm (SPA) [3], the Min-Sum Algorithm (MSA) [40] and the stochastic decoding algorithm. These decoding algorithms use the probabilities of the received messages to start decoding. Hence, getting the probabilities of the received bits is a common step among the decoding algorithms. It is introduced in the next section. Next, we will discuss the SPA. The MSA is just an approximation to the SPA but with a less-complex implementation. After that, we will discuss the stochastic decoding algorithm.

3.6.1 Channel Probability

Once a signal is received from a communication channel, log-likelihood ratio (LLR) values are calculated for each symbol. Next, LLR values are converted into probabilities. Suppose for a code of length n, X is a set of the transmitted data, Y represents the received data, LLR is the log-likelihood ratios and V is the probability of the received bits where:

$$X = \{x_i\}, \quad \forall \ 0 \leq i < n-1 \tag{3.4}$$

$$Y = \{y_i\}, \quad \forall \ 0 \leq i < n-1 \tag{3.5}$$

$$LLR = \{LLR_i\}, \quad \forall \ 0 \leq i < n-1 \tag{3.6}$$

$$V = \{V_i\}, \quad \forall \ 0 \leq i < n-1 \tag{3.7}$$

LLRs are computed as follows:

For every $0 \leq i < n-1$:

$$LLR_i = \ln \frac{P(x_i = 0 \,|\, y_i)}{P(x_i = 1 \,|\, y_i)} \tag{3.8}$$

$$LLR_i = \ln \frac{P(y_i \,|\, x_i = 0) \times P(x_i = 0) \times P(y_i)}{P(y_i \,|\, x_i = 1) \times P(x_i = 1) \times P(y_i)} \tag{3.9}$$

Assume that:

$$P(x_i = 0) = P(x_i = 1) \tag{3.10}$$

$$\therefore LLR_i = \ln \frac{P(y_i \,|\, x_i = 0)}{P(y_i \,|\, x_i = 1)} \tag{3.11}$$

$$LLR_i = \ln \frac{e^{-\frac{(y_{ci}-1)^2}{N_0}}}{e^{-\frac{(y_{ci}+1)^2}{N_0}}} \tag{3.12}$$

$$LLR_i = \ln e^{-\frac{(y_{ci}-1)^2-(y_{ci}+1)^2}{N_0}} \tag{3.13}$$

$$LLR_i = \ln e^{\frac{4y_{ci}}{N_0}} \tag{3.14}$$

$$LLR_i = \frac{4y_{ci}}{N_0} \tag{3.15}$$

The probability of a bit received from the channel, V_i is:

$$\text{probability } = V_i = P(x_i = 1 \,|\, y_i) \tag{3.16}$$

$$V_i = \frac{P(y_i \,|\, x_i = 1)}{P(y_i \,|\, x_i = 1) + P(y_i \,|\, x_i = 0)} \tag{3.17}$$

$$V_i = \frac{e^{-\frac{(y_{ci}+1)^2}{N_0}}}{e^{-\frac{(y_{ci}+1)^2}{N_0}} + e^{-\frac{(y_{ci}-1)^2}{N_0}}} \tag{3.18}$$

$$V_i = \frac{1}{e^{\frac{(y_{ci}+1)^2}{N_0} - \frac{(y_{ci}-1)^2}{N_0}} + 1} \tag{3.19}$$

$$V_i = \frac{1}{e^{\frac{4y_{ci}}{N_0}} + 1} \tag{3.20}$$

$$V_i = \frac{1}{e^{LLR_i} + 1} \tag{3.21}$$

3.6.2 The Sum-Product Algorithm

Sum-Product Algorithm (SPA) is a soft iterative decoding algorithm based on the iterative belief propagation of messages [3]. SPA uses the probabilities of the received bits calculated in the previous section to start decoding on the factor graph on the LDPC code by propagating the message between the VN and the CN. Finally, a hard decision is made on the final probabilities when decoding ends. In this section, we will show how the SPA works, but first, we will introduce some mathematical notations to be used later on in the equations:

- Let D_d is the d-th decoding cycles out of D decoding cycles.
  Decoding cycles D = $\{D_d\}$, where $0 \leq D_d < D$
- Let VN_{ij} be the probability message transmitted from the VN, i to the CN, j.
- Let CN_{cm} be the probability message from CN, c to VN, m in the d-th decoding cycle.

Figures 3.6 and 3.7 show the message propagation between the Variable nodes and the check nodes.

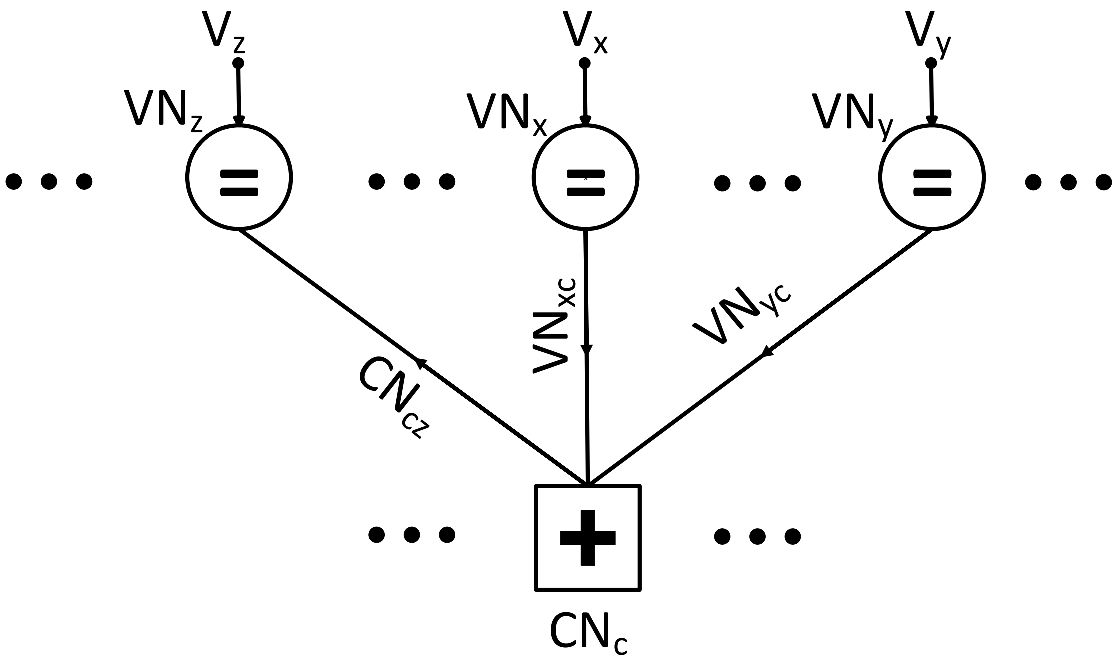

Figure 3.6: Message Propagation from VNs to CN.

The following equations describe the operation of the SPA: For every CN, c of the (n-k) CNs and for every CN edge m of the d_c edges of the CN

$$\mathrm{CN}_{cm} = \frac{1}{2} - \Big(\frac{1}{2} \times \prod_{q \in E_c \setminus m} (1 - 2\mathrm{VN}_{qc})\Big) \tag{3.22}$$

Where E_c is a set of all the VNs connected to the CN, c.

For every edge attached to a CN, the CN performs the above equation to calculate the message that will be passed to the corresponding VN based of the factor graph and the H-matrix. When the messages are received at the VNs, the VNs perform the equations below for every VN, i of the n VNs and for every VN edge, j of the d_v edges of the VN.

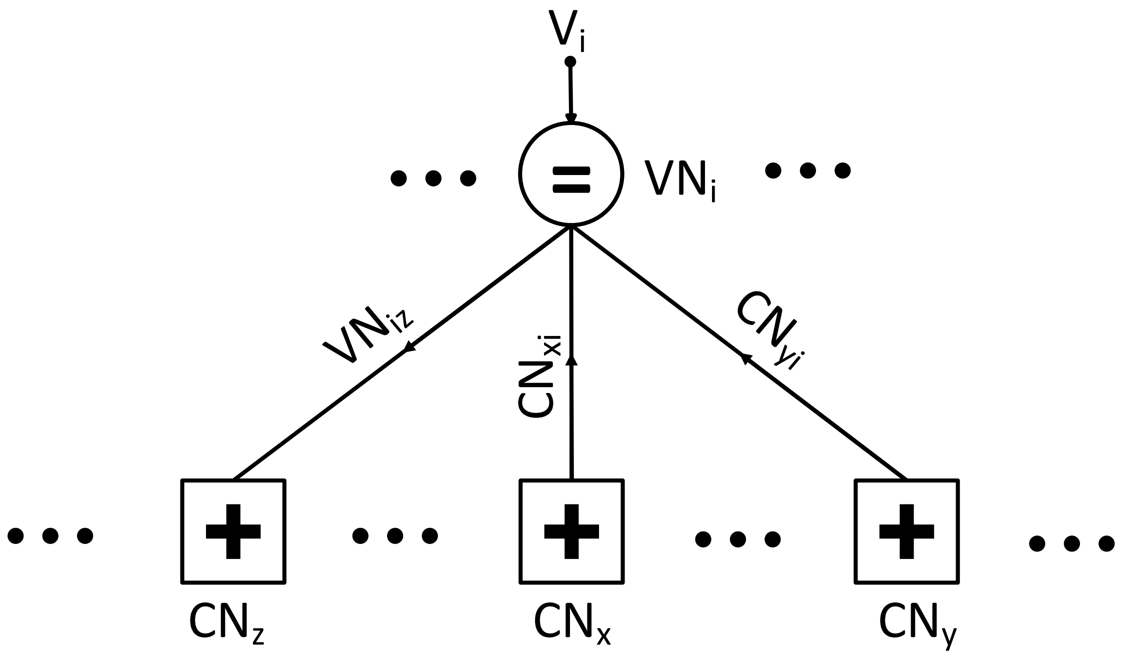

Figure 3.7: Message Passing from CNs to VN.

$$\mathrm{VN}_{ij} = \frac{\left(V_i \times \prod_{q \in E_v \setminus j} \mathrm{CN}_{qi}\right)}{\left(V_i \times \prod_{q \in E_v \setminus j} \mathrm{CN}_{qi}\right) + \left((1 - V_i) \times \prod_{q \in E_v \setminus j} (1 - \mathrm{CN}_{qi})\right)} \tag{3.23}$$

Where Ev is a set of all CNs connected to the VN, i.

The probability of the received bit, computed after every decoding cycle is:

$$\mathrm{VN}_{i} = \frac{\left(V_i \times \prod_{q \in E_v} \mathrm{CN}_{qi}\right)}{\left(V_i \times \prod_{q \in E_v} \mathrm{CN}_{qi}\right) + \left((1 - V_i) \times \prod_{q \in E_v} (1 - \mathrm{CN}_{qi})\right)} \tag{3.24}$$

A hard decision is then made on these probabilities (VN_i) to give the estimated bits:

$$Y_c = \tilde{X} = \{\tilde{x_i}\}, \quad \text{where } 0 \leq i < n \tag{3.25}$$

If:

$$H\tilde{X}^{\top} = 0 \tag{3.26}$$

Then decoding ends and the estimated information bits $\tilde{X}$ can then be extracted. Otherwise, decoding continues and the messages keep propagating between the CNs and the VNs until $H\tilde{X}^{\top} = 0$ is satisfied (early termination) or until the algorithm reaches a pre-defined maximum value for the decoding cycles.

3.6.3 Stochastic LDPC Decoding Algorithm

In this section, we will discuss the stochastic decoding algorithm. As mentioned in the channel probability section, we calculate the channel probabilities based on the LLRs. Each probability is then used to generate a stochastic stream of bits (Bernoulli sequence). The fraction of '1's in the Bernoulli sequence represents the probability that the received bit equals to one. This process is often done by feeding the input probability to a comparator along with numbers from a pseudo random number generator (RNG) as shown in Figure 2.4. After that, the stochastic stream of bits is fed into the variable nodes which are connected to the check nodes based on the parity check matrix.

Assume that D is the length of the stochastic stream which is also equal to the maximum number of decoding cycles. Moreover, The stochastic stream representing the probability of bit i is denoted by $V_{stch\,i}$, where:

$$V_{stch\,i} = \{V_{stch\,i\,j}\}, \quad \text{where } 0 \leq j < D \tag{3.27}$$

Decoding Operation

Similar to the SPA, in stochastic decoding algorithm the CNs and the VNs exchange the messages based on the message belief propagation technique. However, in stochastic decoding the messages are single bits rather than several bits representing the probabilities in the SPA.

Parity-check nodes in stochastic decoding represent the parity check equations of the H-matrix of the LDPC code. CNs can be implemented using networks of XOR gates. Figure 3.8 shows a degree-3 CN as an example. CN_{in0}, CN_{in1} and CN_{in2} are the outputs of the corresponding VNs connected to the CN based on the H-matrix and the factor graph.

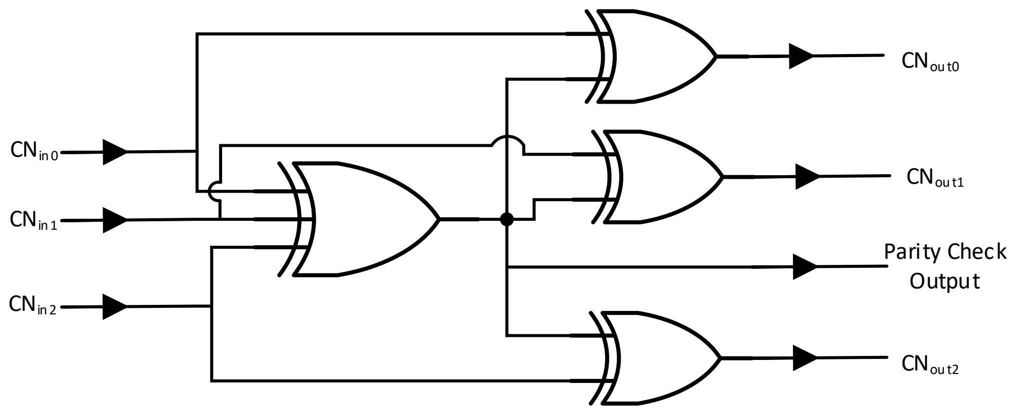

Figure 3.8: Architecture of a Degree-3 Check Node.

The VNs calculate a value for each outgoing edge based the values of the other corresponding incoming edges from the CNs. The first primitive degree degree-2 VN was introduced in [2]. Figure 3.9 shows this circuit for one of the two edges of the VN, i where:

- $V_{in1} = V_{stchij}$.
- $V_{in2} = CN_{mi}$, where CN_{mi} represents the output of the corresponding CN, m connected to the VN, i based on the H-matrix and the factor graph.
- r = VN$_{iq}$, where VN$_{iq}$ represents the output of the VN, i connected to the corresponding CN, q based on the H-matrix and the factor graph.

$$\mathrm{VN}_{iq} = \begin{cases} V_{stch\,i\,j} = \mathrm{CN}_{mi}, & \text{if } V_{stch\,i\,j} = \mathrm{CN}_{mi} \\ \text{hold state}, & \text{if } V_{stch\,i\,j} \neq \mathrm{CN}_{mi} \end{cases} \tag{3.28}$$

The circuit in Figure 3.9 shows that if the incoming edge along with the input bit from the stochastic stream are equal, then the output edge has a regenerative bit of the same value as the other input edge and the stochastic stream. If they are not equal, then the edge is in a "hold state". The output in this case will hold its previous value.

Decoding stops when all the parity check equations of the H-matrix are satisfied (early termination) at one decoding cycles. Otherwise, the VNs and the CNs keep exchanging the bits until the decoder reaches a pre-defined maximum number of cycles.

The final error corrected bits, $\tilde{X}$, are calculated based on a majority function of the incoming messages from the check nodes into a certain VN.

$$\tilde{x_i} = \begin{cases} 0, & \text{if } \sum\limits_{q \in E_v} \mathrm{CN}_{qi} \leq \frac{d_v}{2} \\ 1, & \text{if } \sum\limits_{q \in E_v} \mathrm{CN}_{qi} > \frac{d_v}{2} \end{cases} \tag{3.29}$$

Latching Problem

This primitive VN circuit is not very efficient for practical long codes with cycles. In LDPC codes with cycles in the factor graph, a latching problem occurs with stochastic decoding [41, 42].

The latching problem is caused by a cycle in the factor graph where some CNs and VNs lock into a "hold state" of the propagating messages between the nodes. However, this cycle can be broken by switching of the inner messages of the nodes. This can be done by the random switching activity of the stochastic streams. At high Signal-to-Noise Ratios (SNR), this problem can be worse as the probabilities of the bits are converging towards a strong 1 or 0 making the random switching activity of the stochastic stream rare. Several

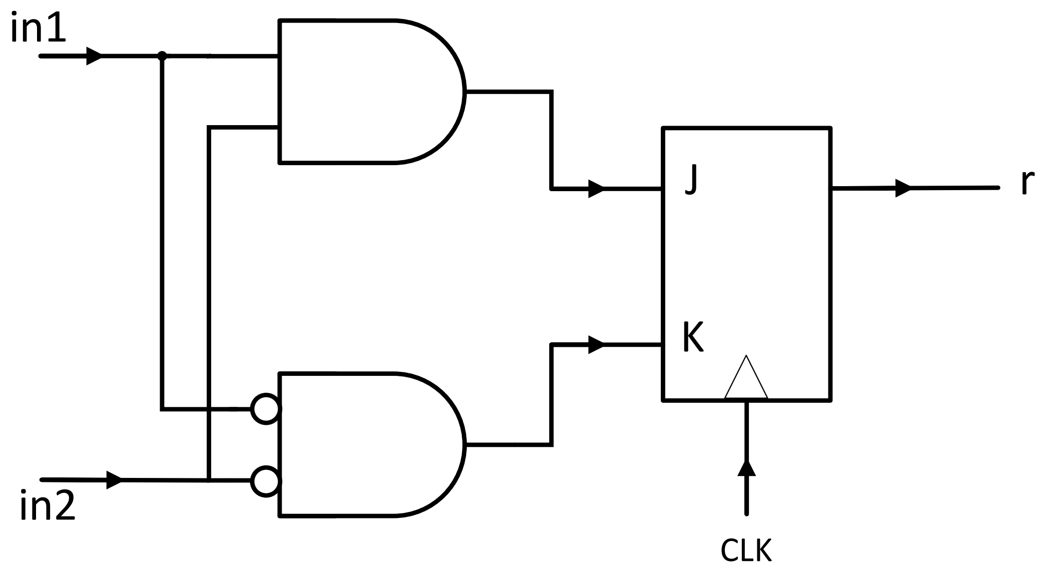

Figure 3.9: First Introduced Degree-2 VN.

solutions have been introduced in [6, 41, 42]. The most important and practical of those are the noise dependent scaling (NDS) and the edge memories (EM).

Noise Dependent Scaling (NDS)

NDS is scaling down the LLR values by a factor dependent on the noise spectral density N_0 to ensure a near-same level of switching activity in the stochastic stream at different SNRs. This decreases the effect of the switching activity at high SNRs.

Assume LLR_{NDS} are the noise-dependent scaled LLRs.

$$\text{LLR}_{NDS} = (\alpha \times N_0) \times \text{LLR} \tag{3.30}$$

α is a constant factor chosen by the designer for a good performance. We picked a value of half in our simulations later on, which is also reported in the literature [6].

Edge Memories

One other solution reported in [6] is instead of using the flip-flop in Figure 3.9, we use a big register attached to each VN edge, for example, a 32 or a 64-bit shift register. In case of a regenerative bit, the output is the regenerative bit and the edge memory is updated with the value of the regenerative bit. In case of a hold state, the output bit is randomly chosen from a memory built up using only the regenerative bits of the previous decoding cycles. That solution gives a good performance as reported in [6,7]. However, it is not area efficient. One example on that is the 10GBASE-T LDPC code which has 2048 variable nodes each of $d_v = 6$. If we are using a 64-bit memory attached to each edge, the total number of flip-flops becomes $2048 \times 6 \times 64 = 786,432$ flip-flops. To solve this problem, tracking forecast memory and majority tracking forecast memory have been introduced.

Tracking Forecast Memory

There are different types of that memory reported in the literature [6–9]. TFM can be interpreted as a state machine where the states are only updated in case of a regenerative bit. Otherwise, the states are not updated. In this case the output, is chosen based on the current state of the state machine. Figure 3.10 shows a circuit for the TFM used instead of the flip-flop in Figure 3.9 Figure 3.9 where:

- $P(t)$ represents the probability of the underlying bit while decoding. It is initialized to the probability received from the channel.
- $R(t)$ is a pseudo random number.
- $r(t)$ is the output bit from the AND gate in Figure 3.9.
- β is a constant predefined value by the designer.

Then:

$$P(t+1) = \begin{cases} P(t) - \beta\, P(t), & \text{if } r(t) = 0 \\ P(t) + \beta\, \overline{P(t)}, & \text{if } r(t) = 1 \end{cases} \tag{3.31}$$

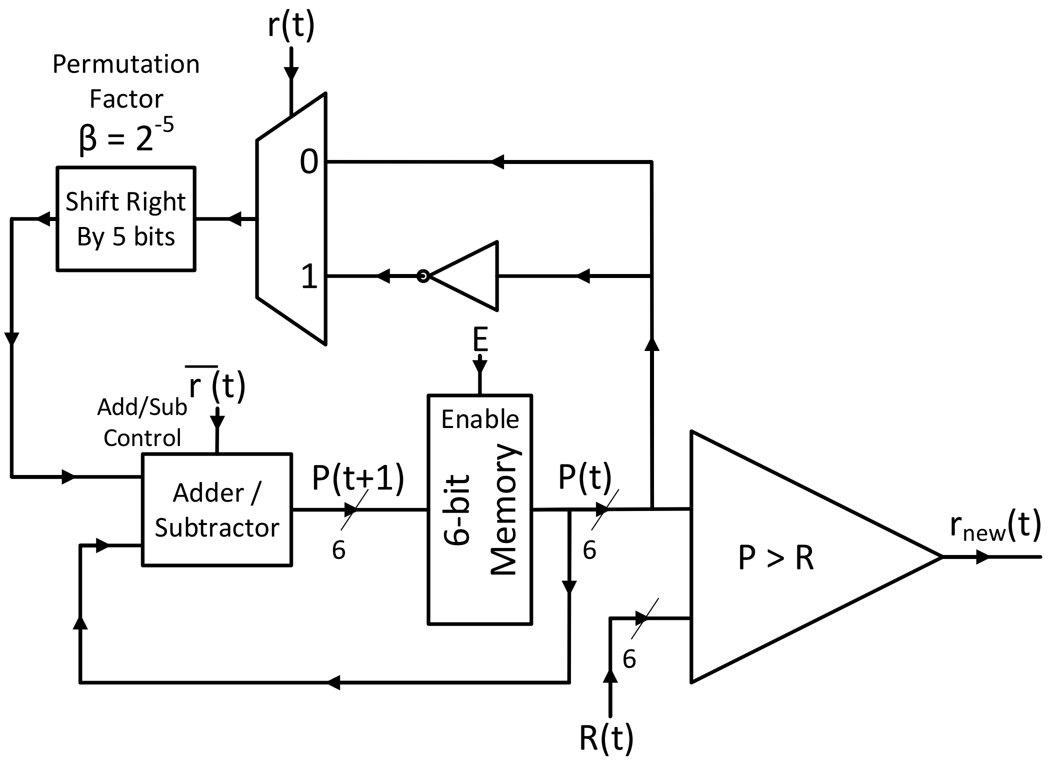

Figure 3.10: Tracking Forecast Memory.

The TFM achieves a good performance but we can still do better. Instead of attaching this TFM to each VN edge, we can use one majority TFM for the whole VN while still getting a good decoding performance.

Majority Tracking Forecast Memory (MTFM)

The memory used in our simulations is the majority tracking forecast memory (MTFM) [9,10]. A full degree-3 VN structure with the MTFM is shown in Figures 3.11, 3.12 and 3.13 as an example. Figure 3.11 shows a degree-3 sub-node. Figure 3.12 shows the MTFM block which is composed of the TFM in Figure 3.10. Figure 3.13 shows the complete corresponding degree-3 variable node.

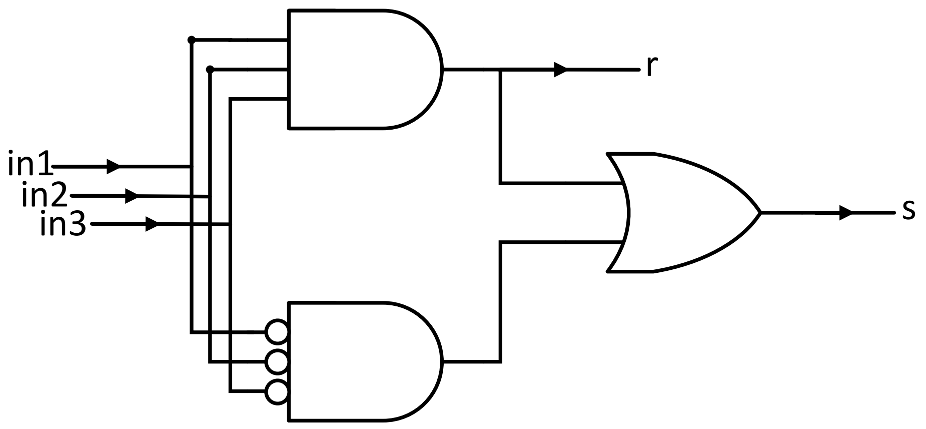

Figure 3.11: Degree-3 Sub-node of the Degree-3 Variable Node. (s) is the state bit and (r) is the output bit.

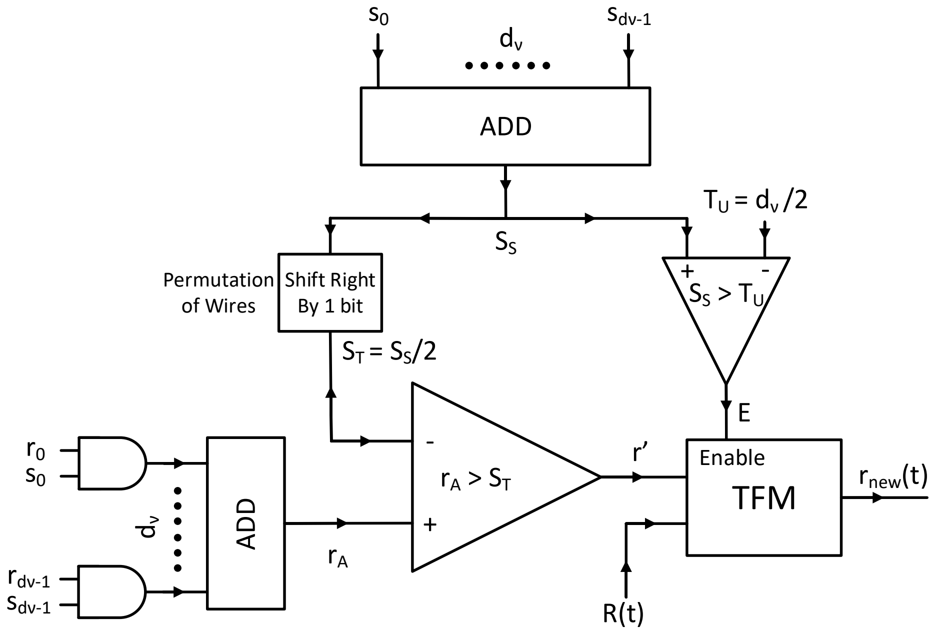

Figure 3.12: Majority Tracking Forecast Memory.

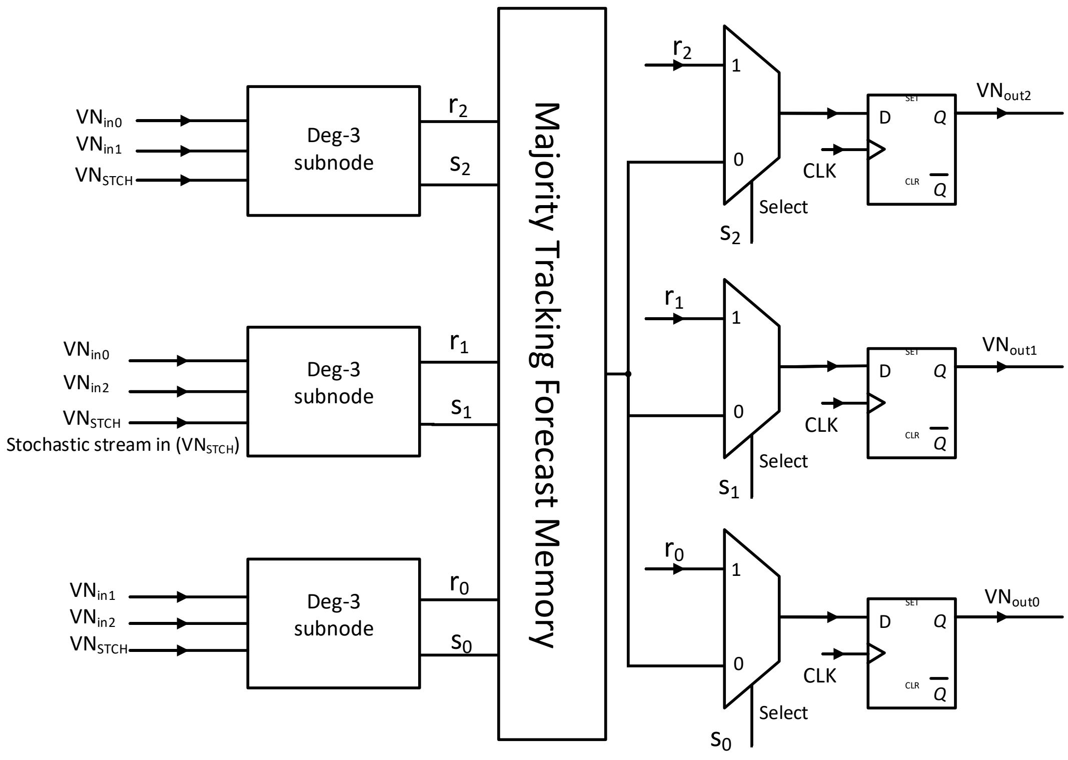

Figure 3.13: Variable Node of degree 3.

3.7 Simulated Codes

Two standardized LDPC codes were chosen to be simulated, the (1056,704) LDPC code from the WiMAX standard and the (2048,1723) LDPC code from the 10GBASE-T standard. We implemented both the SPA algorithm and the stochastic decoding algorithm and simulated them using Monte-Carlo simulation. To test the performance of a decoder, we simulate an all-zero codeword and plot the Frame-Error Rate (FER) and the Bit-Error Rate (BER) of the error-corrected bits versus the SNR per bit, (E_b/N_0). (E_b/N_0) is the ratio between the energy per bit to the noise power spectral density. It is the normalized SNR. The FER represents the probability of error for certain transmitted block, calculated by simulating the algorithm for a large number of blocks at a certain SNR and dividing the number of wrongly decoded blocks by the total number of blocks. The BER on the other hand is the probability of error in the received bit, calculated by simulating the algorithm for a large number of blocks at a certain SNR and dividing the number of wrongly decoded bits by the total number of bits. Figure 3.14 and Figure 3.15 show the BER and the FER curves of the (1056,704) WiMAX LDPC code. Figure 3.16 and Figure 3.17 show the BER and the FER curves of the 10GBASE-T LDPC code We are simulating the sum product algorithm to compare its results with the stochastic decoding algorithm. The sum product algorithm runs for 6 decoding cycles (DCs) for both the WiMAX LDPC code and the 10GBASE-T LDPC code. For stochastic decoding algorithm, the number of decoding cycles is 400 for the 10GBASE-T LDPC code, while it is 700 for the WiMAX LDPC code. The number of decoding cycles is chosen by the designer for a good decoding performance. The more the decoding cycles used, the better the decoding performance gets. In addition, we are using the early termination technique described earlier. Hence, in most of the cases, decoding ends much earlier than the maximum number of decoding cycles. At each simulated SNR value, we stop decoding after reaching 10 block errors for both of the two codes.

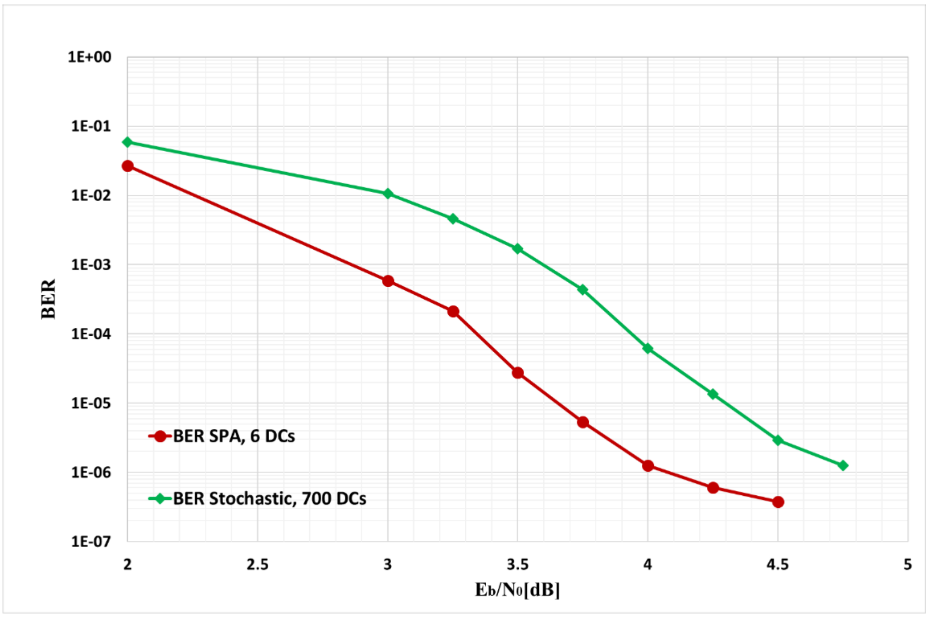

Figure 3.14: BER of the (1056,704) WiMAX LDPC Code for the SPA and the Stochastic LDPC Decoding Algorithms.

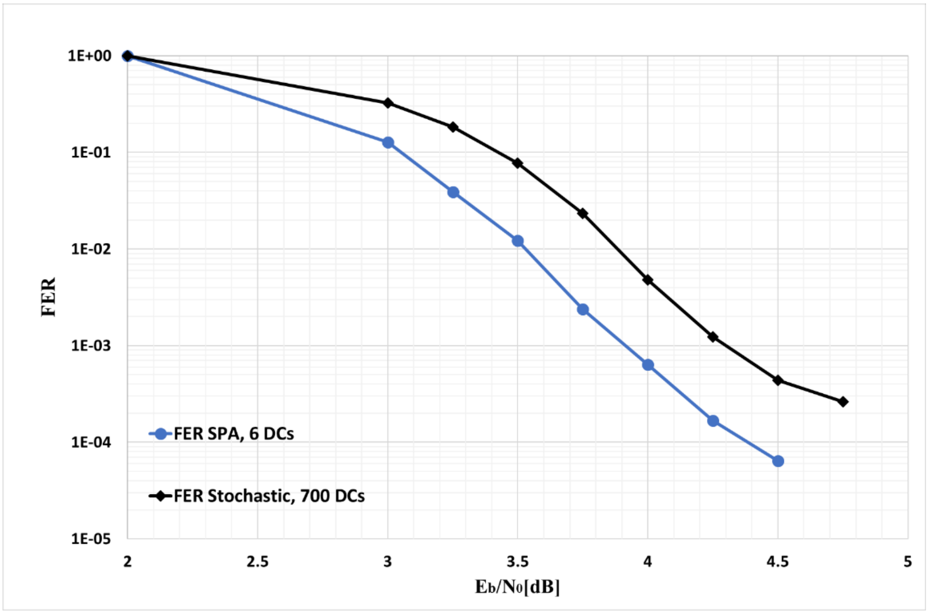

Figure 3.15: FER of the (1056,704) WiMAX LDPC Code for the SPA and the Stochastic LDPC Decoding Algorithms.

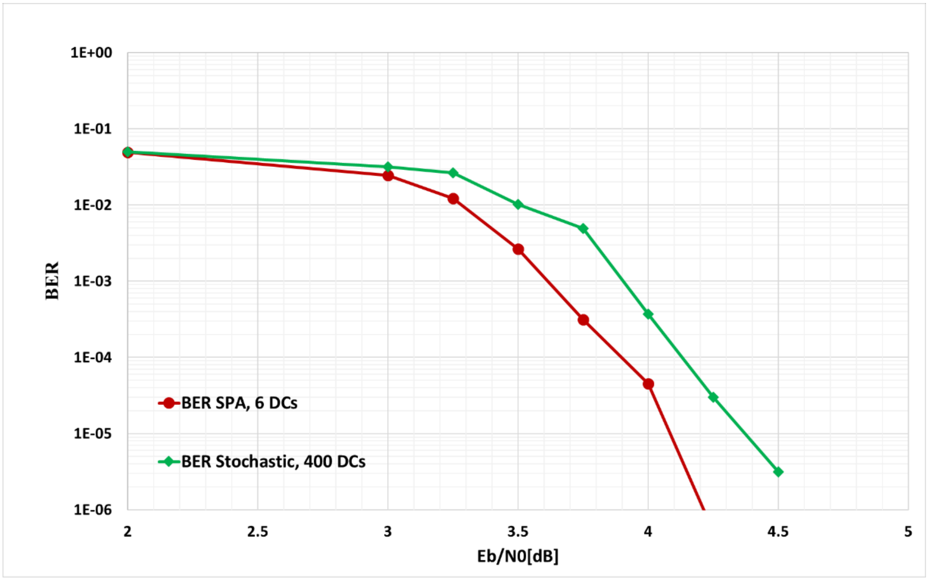

Figure 3.16: BER of the (2048,1723) 10GBASE-T LDPC Code for the SPA and the Stochastic LDPC Decoding algorithms.

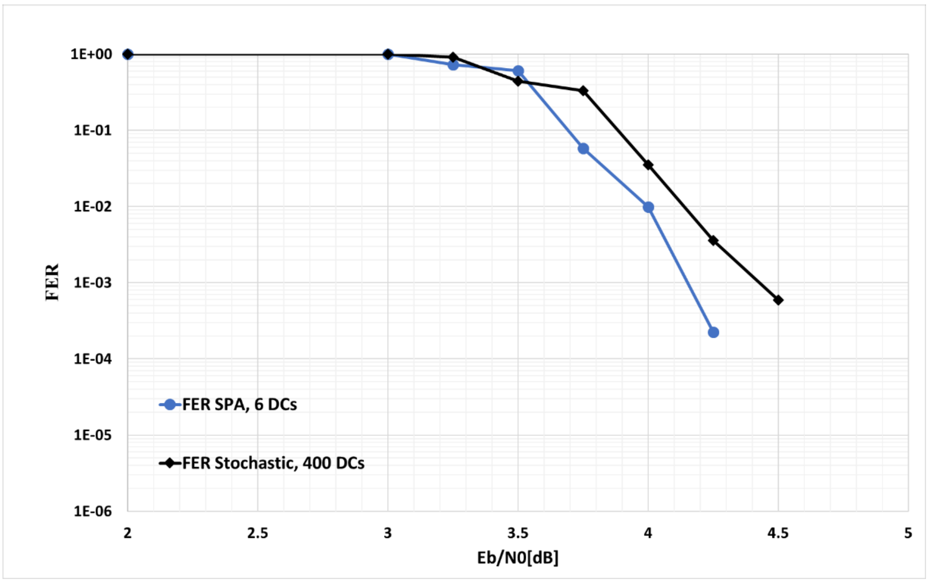

Figure 3.17: FER of the (2048,1723) 10GBASE-T LDPC Code for the SPA and the Stochastic LDPC Decoding algorithms.

Chapter 4

Modeling of Soft Errors

In this Chapter, we intend to demonstrate two models that we developed to emulate the effects of soft errors on the performance of stochastic LDPC decoders.

As shown in Chapter 3, different logic structures can be used to build stochastic LDPC decoders. This may have direct impact to the source of errors and how we model them. When lowering V_{DD} or increasing the clock speed beyond a certain limit, errors may occur. This means that we have a violation in the critical path and that we are consequently, violating the setup time of the flip-flop placed at the incoming edge of a VN node. We are thus interested in the critical path (and near-critical paths) in a decoder. To develop an accurate model, we should know how the critical (and near-critical) paths are distributed in a system. Based on the implementation mentioned in Chapter 3, the longest paths of a stochastic LDPC decoder are the paths from the output of flip-flops of the VN nodes back to the input of flip-flops of other VN nodes as shown in Figure 4.1, assuming that the system has no pipelining stages inside the VN nodes or the CN nodes. It is important to know how the longest paths are distributed in the decoder. We may have a few very long critical paths while the other paths are much shorter. On the other hand, the longest paths might be distributed evenly among the nodes, meaning that, the path delays are close to each other. There are many factors that may affect the number of the critical paths and how they are distributed in the system. The most important of these are the type of the LDPC code (regular or irregular), the structure of the code, the wiring circuitry, and the structure of the sub-blocks. The type of LDPC code is very important because having regular codes means that all the sub-blocks have the same circuitry and hence, the same inner propagation delays. Thus, the wiring will have an important impact on the delay based on the code structure specially in the deep submicron technologies, where wiring capacitance has an important effect. Nodes connected using long wires will exhibit higher capacitance and longer delays. The 10GBASE-T code is an example on that. The 10GBASE-T code is a regular code with 2048 degree-6 variable nodes and 384 degree-32 parity check nodes. This is considered a highly-congested code as we need $2048 \times 6 \times 2 = 24,576$ connections from variable nodes to check nodes and from check nodes back to variable nodes. Consequently, the structure of the code and the wiring, here, has a direct impact on the critical path, assuming no buffers were used to support long wires. On the other hand, the (1056,704) WiMAX code has variable nodes with degrees of 2, 3 and 4. The check nodes are of degrees 10 and 11. The longest paths here occur in the variable nodes of degree 4 because the degree-4 variable nodes themselves exhibit longer propagation delays than the degree-3 and the degree-2 nodes. This is because of the combinational delay inside the block. Next, the wiring capacitance and the code structure come into effect on the propagation delay among the degree-4 variable nodes.

Moreover, the error model depends on the placement and routing (P&R) of the circuit. In other words, we can have a certain few long critical paths than other paths in a regular code as a result of the P&R. In addition, violating the longest path doesn't always imply the occurrence of errors. The reason is that error only occurs when we have a flip-flop changing its state from 1 to 0 or vice versa depending on the switching activity [43]. However, if the flip-flop holds its old state, we do not expect an error. Moreover, we may have a flipped bit while the setup time is not violated. This happens because the propagation delay depends on several previous states of the internal memories (IMs), TFMs, random numbers generated from the RNG, etc. Consequently, lowering V_{DD} or overclocking the system to violate a certain path delay doesn't always mean that we will have an error, especially at high SNRs where flipped bits are less likely to happen.

As a result, two different models are used in our paper to represent the violation of the critical paths. In both models, errors are being added intentionally at the outputs of the flip-flops of the variable nodes. These errors model the setup time violation due to V_{DD}

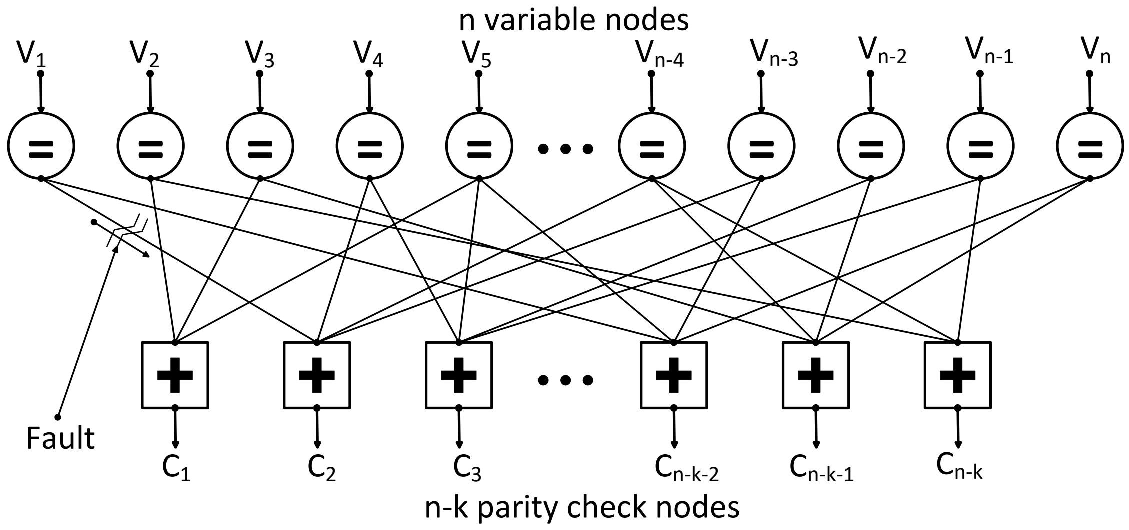

Figure 4.1: A Factor Graph from an (n,k) LDPC Code with n Variable Nodes and n-k Parity Check Nodes Showing the Faults.

scaling or system overclocking.

4.1 First Error Model

For the first model, we assume that all variable nodes are vulnerable to setup time violations. Hence, errors are inserted at each variable node edge with a certain probability. In other words, every decoding cycle, each variable node has a probability of setup time violation. Consequently, errors are inserted in 100% of the variable nodes with a certain probability in case of a flipping bit. The tested probabilities are 0.001, 0.005, 0.05 and 0.1.

4.2 Second Error Model

The second model is similar to the first model, but we only insert errors in a certain percentage of the variable node flip-flops. We assume that there is a certain percentage of the variable nodes that has much longer propagation delay than other variable nodes. Hence, scaling down V_{DD} or overclocking the system will result in errors in these nodes only. Each of these variable nodes has a probability of setup time violation in case there is a flipping bit. The model we test is that we assume 20% of the variable nodes have the longest paths and are vulnerable to errors. We then test four possible probabilities for each variable node: 0.005, 0.025, 0.25 and 0.5.

4.3 Simulation Results

We carried out several experiments to validate and prove the tolerance of stochastic decoding to errors for both the (2048,1723) LDPC code of the 10GBASE-T standard and the (1056,704) of the WiMAX standard. The first is running the decoding algorithms with no errors inserted. Next, we test the decoder with the first error model where errors are inserted at all variable node edges. We try four different probabilities of error to occur at each variable node edge: 0.001, 0.005, 0.05 and 0.1. Then we test the second error model where errors are inserted at only 20% of the variable node edges with four different probabilities to be inserted at the chosen 20% of variable node edges: 0.005, 0.025, 0.25 and 0.5. One important note to consider is that for the second model, all the 20% affected nodes are random chosen only within the high degree nodes (degree= 4) of the WiMAX LDPC code. While for the 10GBASE-T LDPC code they were chosen randomly from all the nodes as all of them have the same degree. These results have been published in [44].

4.3.1 First Error Model

Figure 4.2 and Figure 4.3 show the BER and the FER of the WiMAX LDPC code in the ideal case where no errors are inserted and in the case of the 1$^{\text{st}}$ model errors. The figures show that the decoder achieves a near ideal performance when applying error model 1 on the WiMAX LDPC code.

Figure 4.4 and Figure 4.5 show the BER and the FER of the 10GBASE-T LDPC code in the ideal case where no errors are inserted and in the case of the 1$^{\text{st}}$ model errors. The

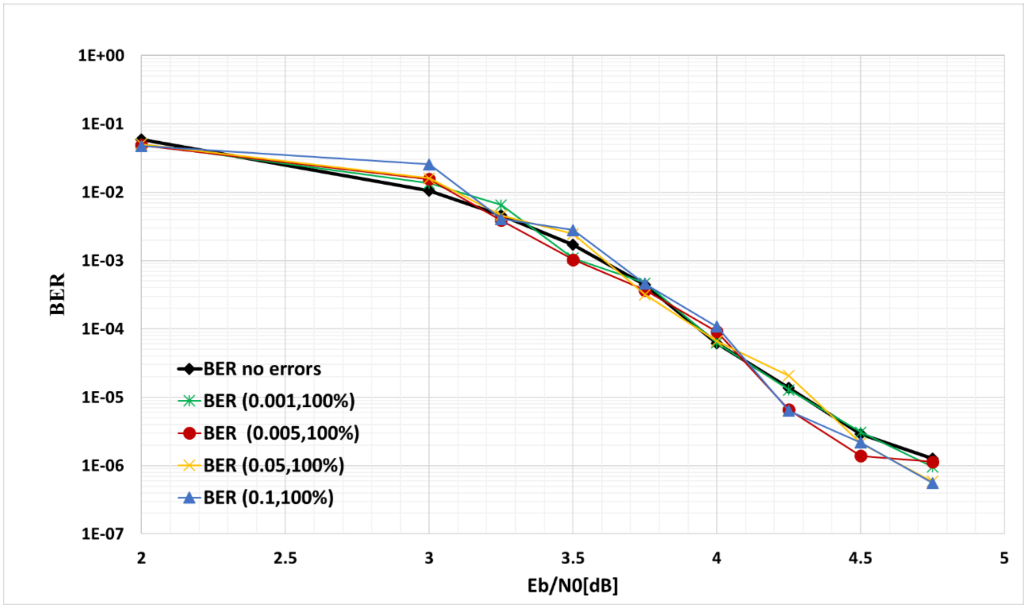

Figure 4.2: BER of the (1056,704) WiMAX LDPC code in Case No Errors Are Inserted and in Case of Model 1 Errors (100% of VN edges).

figures show that the decoder achieves a near ideal performance when applying error model 1 on the 10GBASE-T LDPC code, as well.

4.3.2 Second Error Model

Figure 4.6 and Figure 4.7 show the BER and the FER of the WiMAX LDPC code in the ideal case where no errors are inserted and in the case of the 2nd model errors. The figures show that the decoder achieves a near ideal performance when applying error model 2 on the WiMAX LDPC code.

Figure 4.8 and Figure 4.9 show the BER and the FER of the 10GBASE-T LDPC code in the ideal case where no errors are inserted and in the case of the 2nd model errors. The figures show that the decoder achieves a near ideal performance when applying error model 2 on the 10GBASE-T LDPC code, as well.

The graphs clearly show the strong fault tolerance of stochastic LDPC decoders. Ac-

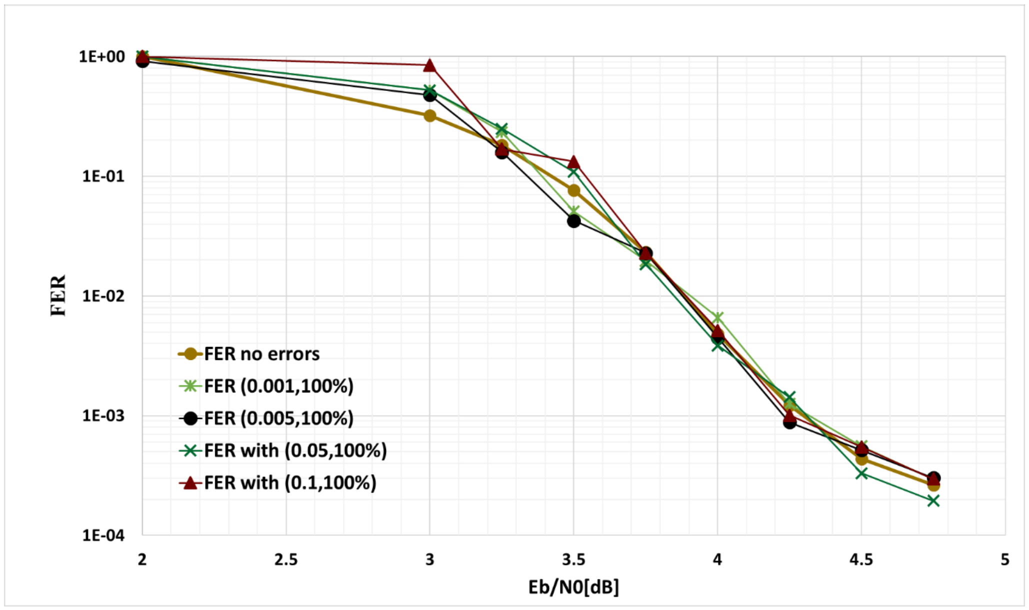

Figure 4.3: FER of the (1056,704) WiMAX LDPC code in Case No Errors Are Inserted and in Case of Model 1 Errors (100% of VN edges).

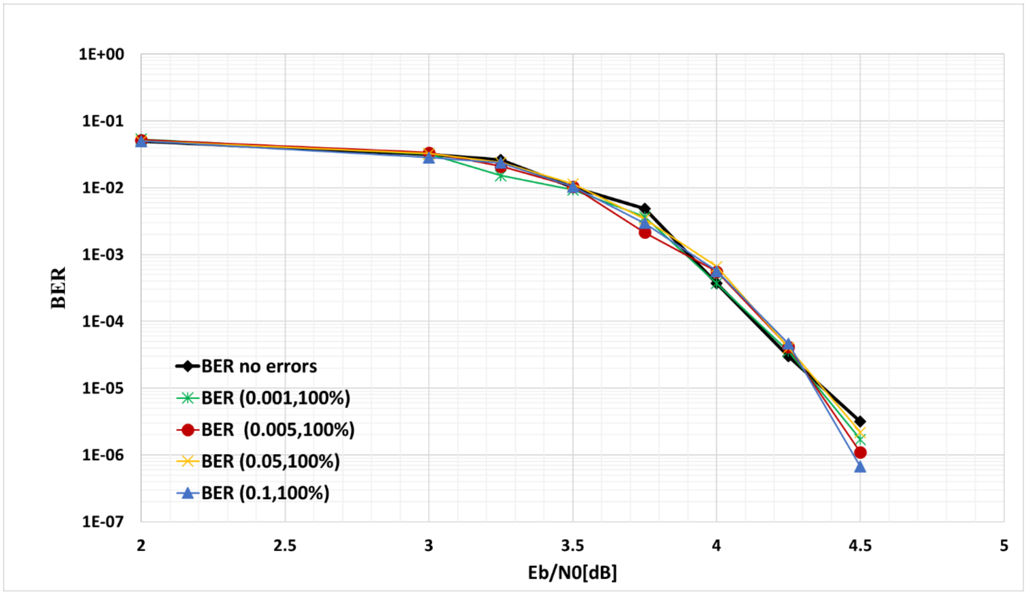

Figure 4.4: BER of the (2048,1723) 10GBASE-T LDPC code in Case No Errors Are Inserted and in Case of Model 1 Errors (100% of VN edges).

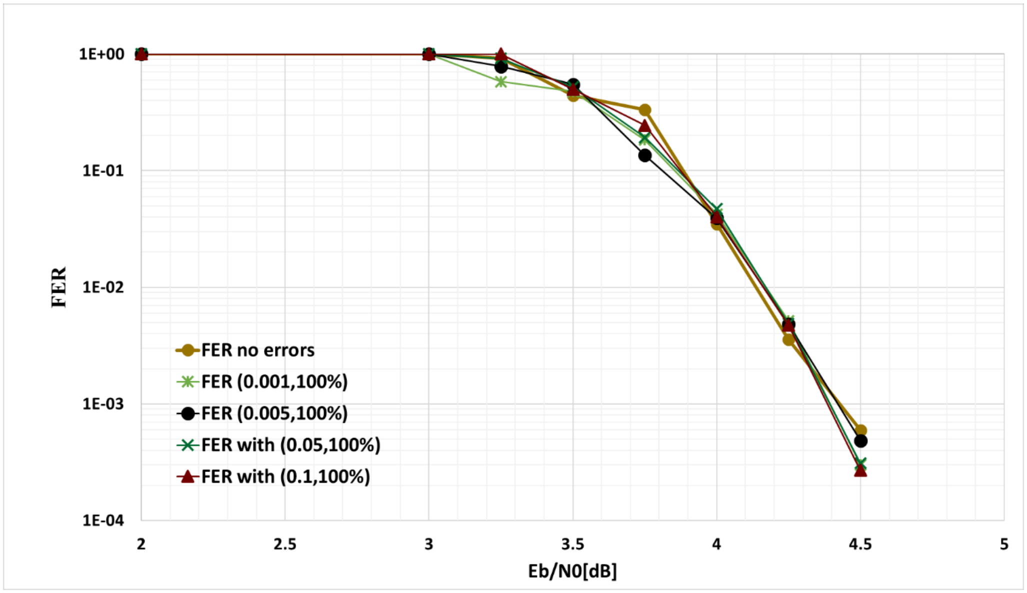

Figure 4.5: FER of the (2048,1723) 10GBASE-T LDPC code in Case No Errors Are Inserted and in Case of Model 1 Errors (100% of VN edges).

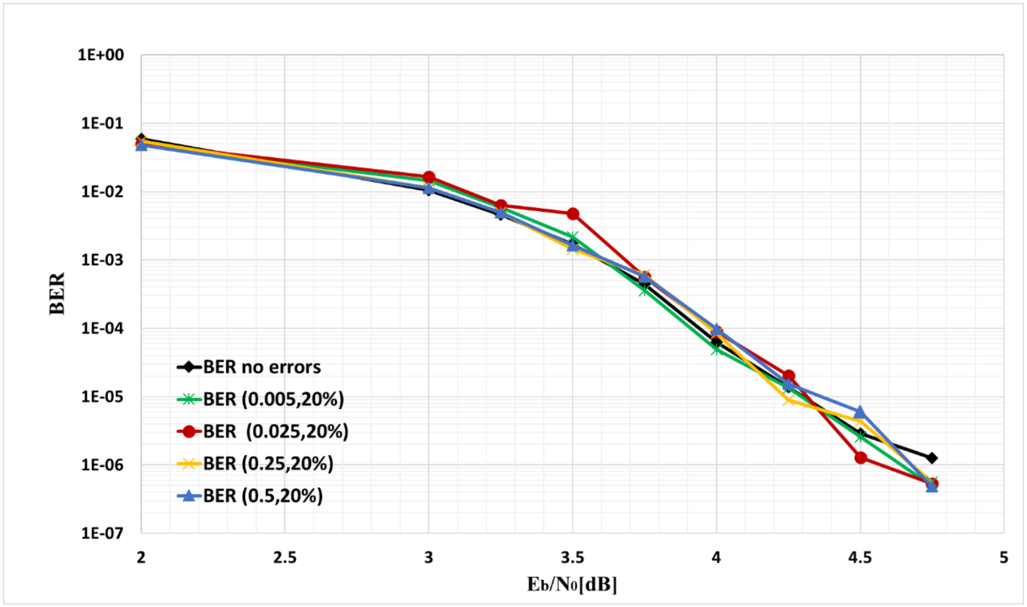

Figure 4.6: BER of the (1056,704) WiMAX LDPC code in Case No Errors Are Inserted and in Case of Model 2 Errors (20% of VN Edges).

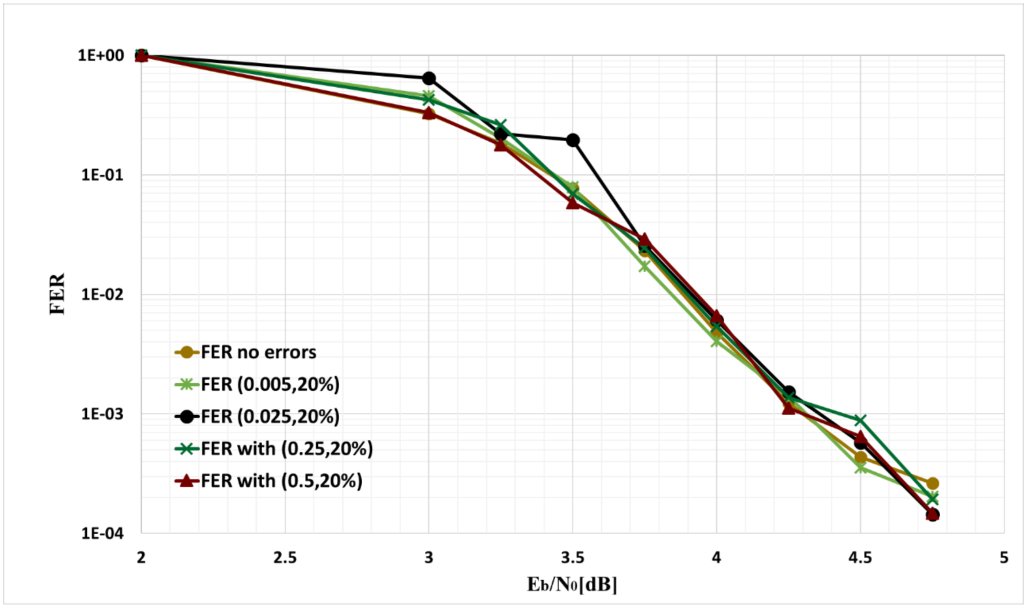

Figure 4.7: FER of the (1056,704) WiMAX LDPC code in Case No Errors Are Inserted and in Case of Model 2 Errors (20% of VN edges).

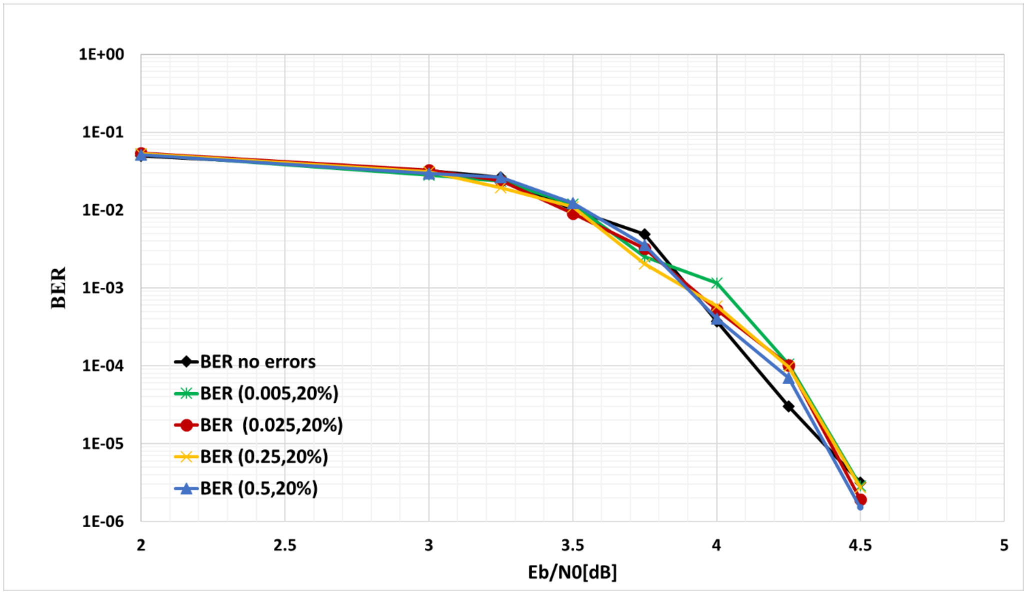

Figure 4.8: BER of the (2048,1723) 10GBASE-T LDPC code in Case No Errors Are Inserted and in Case of Model 2 Errors (20% of VN edges).

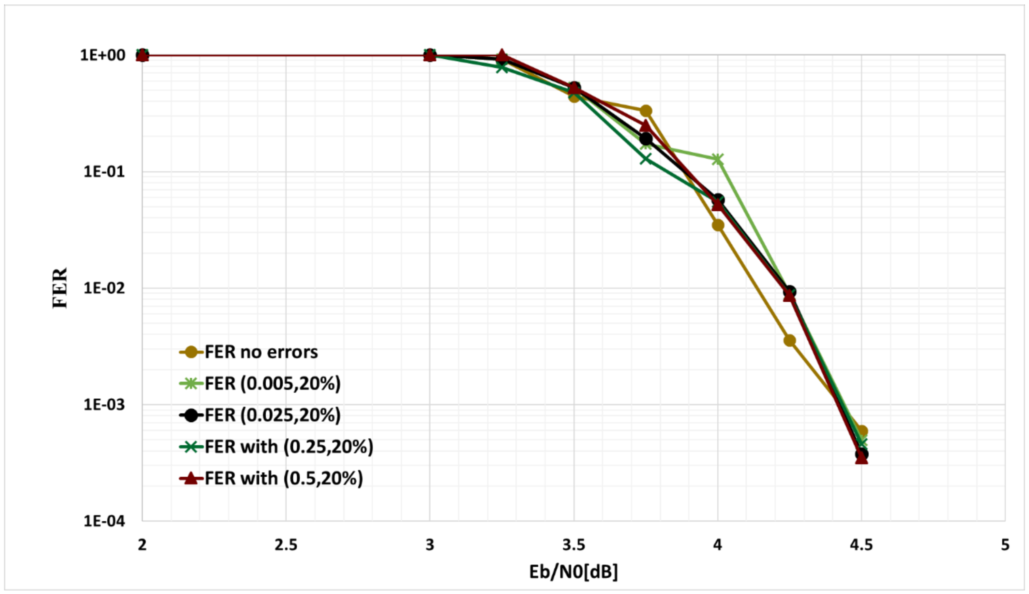

Figure 4.9: FER of the (2048,1723) 10GBASE-T LDPC code in Case No Errors Are Inserted and in Case of Model 2 Errors (20% of VN edges).

cording to the simulation results, the performance under both error models nearly achieve the same performance as the ideal case. This proves that stochastic decoding can be very useful in systems with high performance or very low power requirements where we can speed up the clock or lower V_{DD} while maintaining good bit error rate performance.

Chapter 5

Chip Implementation

5.1 5.1 General Overview

In this chapter, we will discuss the implementation details of the chip we designed to test the tolerance of stochastic LDPC decoding algorithm to soft errors and verify the results presented in Chapter 5. The implemented LDPC decoder decodes the (1056,704) LDPC code from the WiMAX standard. The chip is still in fabrication so we don't have the post-layout results yet. Here are the code characteristics:

- Irregular Code.
- Block code length = number of variable nodes (VNs) = 1056.
- Number of Parity Checks (CNs) = 352.
- Degrees of Variable nodes = 2,3,4
- Degrees of parity check nodes = 10,11

In Chapter 3, we discussed the general architecture of stochastic LDPC decoders. In this chapter, we will present the specific architecture that we implemented and the structure of each block in more detail. Figure 5.1 shows the general structure of the chip. It is composed of the input buffers, the main decoding structure and the output buffers. The decoding part is composed of several blocks. These blocks include the variable nodes, check nodes, random number generators and input output buffers.

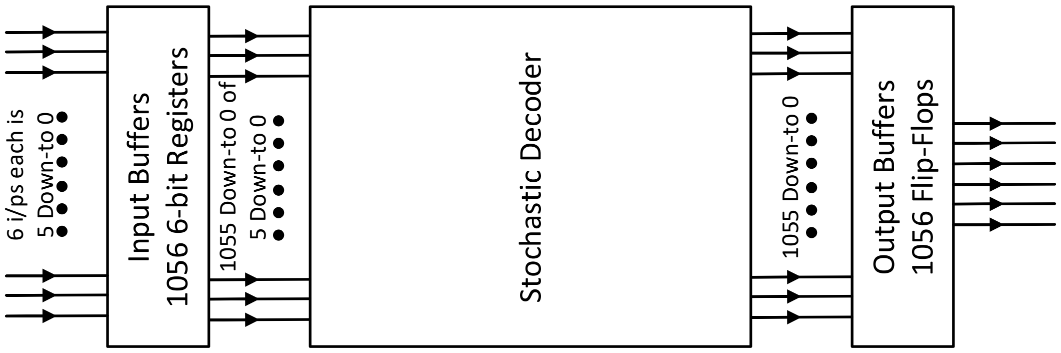

Figure 5.1: Chip General Structure

The inputs to the chip are the probabilities of the bits in a received codeword from the channel, where each probability is represented in 6 bits. These probabilities are fed into the input buffers in series until the full codeword is saved in the buffers. Next the probabilities are passed to the decoder. Using comparators, stochastic streams are generated for each probability. These stochastic streams are fed into the variable nodes. After that, decoding starts, where messages are exchanged between the variable nodes and the check nodes as discussed in Chapter 3. Decoding ends when a maximum number of decoding cycles is met or when all the parity check equations are satisfied. Finally, the error corrected codeword is saved in the output buffers and outputted in series for processing off-chip.

As shown in Figure 5.1, we have 3 main pipelined stages, the input buffers, the decoder and the output buffers. A state machine controls the data manipulation between the stages. The input buffers need T_{load} clock cycles to load one block of message probabilities. The output buffers need T_{out} to output the decoded bits off-chip. The decoder needs T_{dec} to finish decoding, where:

$$T_{load} = \frac{1056 \times 6}{6 \times 6} = 176 \text{ clock cycles} \tag{5.1}$$

$$T_{out} = \frac{1056}{6} = 176 \text{ clock cycles} \tag{5.2}$$

$$0 < T_{dec} \leq 700 \text{ clock cycles} \tag{5.3}$$

Since we are using the early termination technique, explained in Chapter 3, T_out can take any value from zero to the maximum pre-defined value of 700 clock cycles. Hence, the decoder can finish decoding much earlier than the time it takes the buffers to load and output the data bits, especially at high SNRs. On the other hand, the decoder might continue decoding till the 700th clock cycle when it can't converge to a correct codeword or generally at low SNRs. In these two cases, the state machine is designed to halt the operation of the blocks which finished their job earlier than the others. If the decoder finishes decoding in less than 176 clock cycles, it waits till the 176th clock cycle to start decoding the new data. On the other hand, if the decoder continues decoding after the 176 clock cycles, the input buffers stop taking new input data bits or outputting any error corrected bits. The chip rises a flag to indicate this situation to other chips communicating with it.

This is not the optimal setting of the chip to achieve the highest possible output. However, we are designing the chip to test the capability of stochastic decoders to tolerate soft errors. We are not trying to design the fastest possible chip. In addition, this is a test chip, not a practical chip. In a practical chip, to achieve a high throughput, inputs are usually fed on-chip from other preceding blocks. In our chip, we feed the probabilities directly to the input pins to have an off-chip control on other blocks preceding the decoder that don't affect the main objective of the chip which is testing the fault tolerance of the stochastic decoder to soft errors.

Figure 5.2 shows the decoder block with more details. We will discuss the inside blocks of the decoder for this specific (1056,704) LDPC code from the WiMAX standard in the next sections.

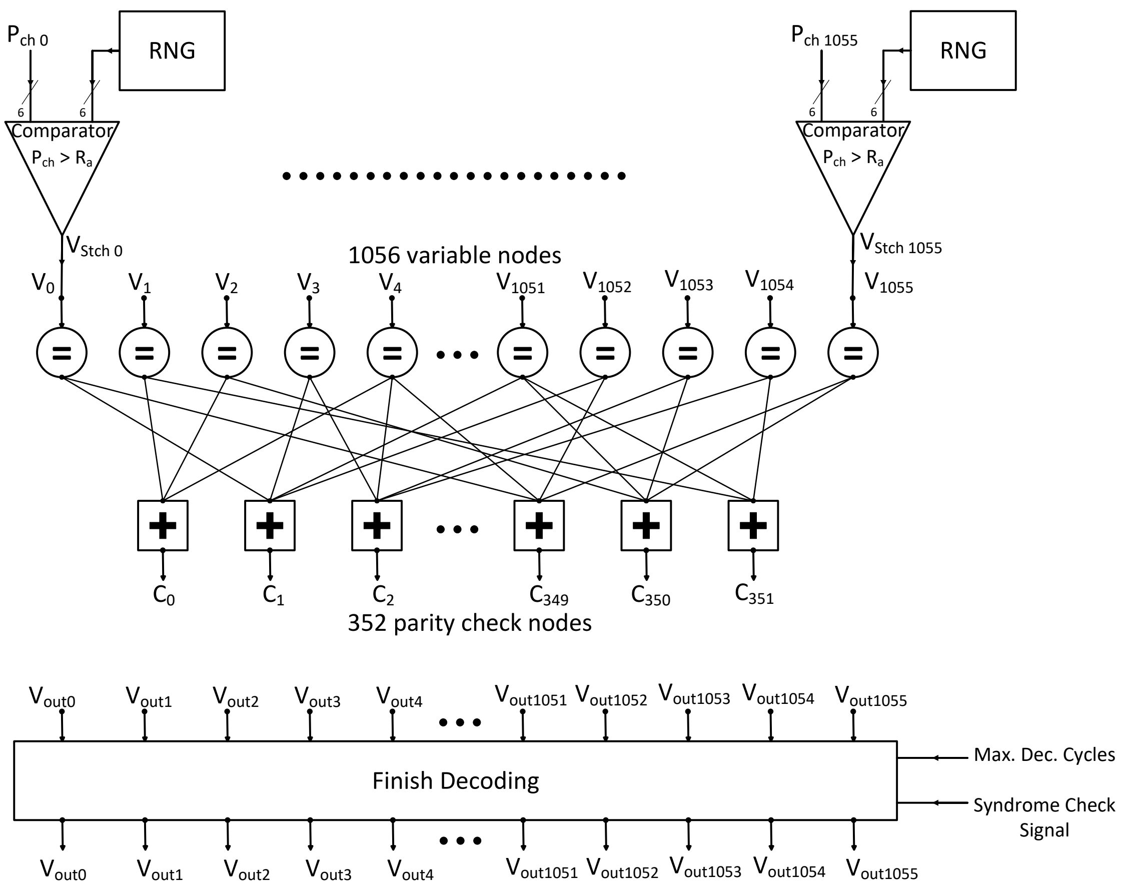

Figure 5.2: Chip Implementation of the Stochastic LDPC Decoder to the (1056,704) LDPC Code from the WiMAX Standards.

5.2 Check Nodes

As discussed in Chapter 3, the check nodes are implemented using XOR gates. The (1056,704) LDPC code from the WiMAX standard has check nodes of degrees 10 and 11. Figure 5.3 shows the general circuit used to implement the CN for each of the two degrees.

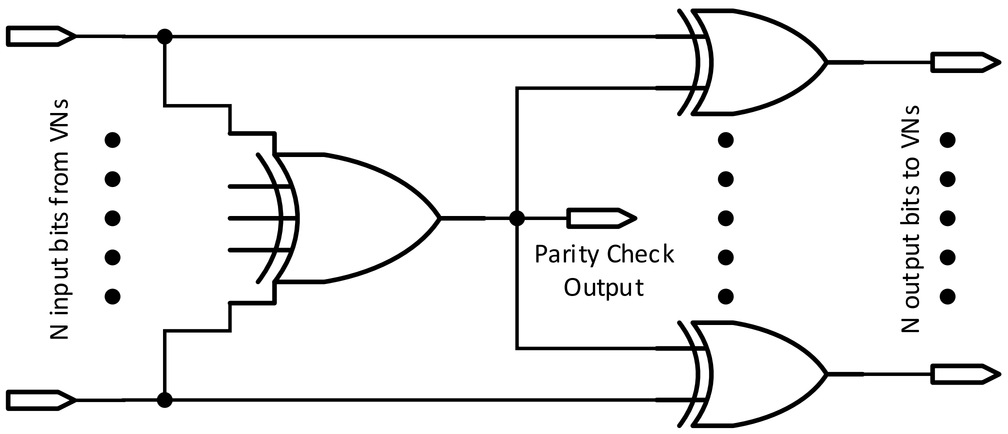

Figure 5.3: General Architecture of a Check Node.

5.3 Variable Nodes

The variable nodes are implemented using the Majority Tracking Forecast Memory (MTFM) presented in Chapter 3 in Figures 3.10 and 3.12. Some of the VNs of the (1056,704) WiMAX LDPC code are of degree 2, some are of degree 3 and some are of degree 4. Degree-3 VNs are shown in Figure 3.13. A degree-2 VN has the same structure as a degree-3 VN but with only 2 inputs instead of 3. Figure 5.4 shows a degree-2 sub-node of a degree-2 VN. The full circuit for the degree-2 VN is shown in Figure 5.5.

The degree-4 VN is a bit different than the degree-2 and degree-3 VNs. The reason is that for nodes of degrees higher than 3, it becomes hard for the VN to output a regenerative bit because it requires that 3 VN edges entering the VN along with the bit from the stochastic stream to have the same value, otherwise the VN goes into the "hold state". So, to increase the possibility of outputting a regenerative bit, we divide the degree-4 sub-node into 2 degree-2 sub-nodes. So, for 4 inputs entering the VN, each two inputs are compared to each other, if they are equal, a regenerative bit is generated, otherwise the sub-node uses a small internal memory (IM) composed of a 2-bit shift register to get a previous

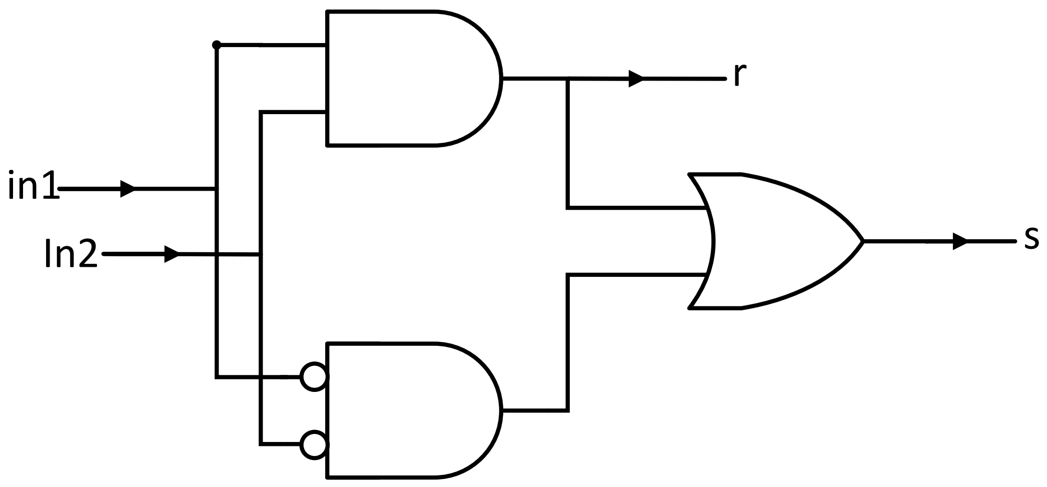

Figure 5.4: Degree-2 Sub-node of the Degree-2 VN.

regenerative bit. The IM is only updated in case the sub-node has a regenerative bit. In case of a "hold state" a bit is randomly drawn from the IM. The circuit for the degree-2 sub-node with IM is shown in Figure 5.6. The architecture of the IM is shown in Figure 5.7. Next, we check if the outputs of the two degree-2 sub-nodes have the same value using a circuit similar to the circuit in Figure 5.4. If so, we have a regenerative bit for this VN edge. Otherwise, this edge is in a hold state and we use the MTFM to give the output. Figure 5.8 shows the full degree-4 VN.

5.4 Random Number Generator

In our chip, we need many random numbers in several blocks. We need random numbers for the stochastic stream generation. In addition, we need random numbers inside the VNs in the MTFM and IMs.

In theory, we want the stochastic streams to be independent, as well as the VNs. Hence, we should use a RNG for every input probability from the channel to generate the

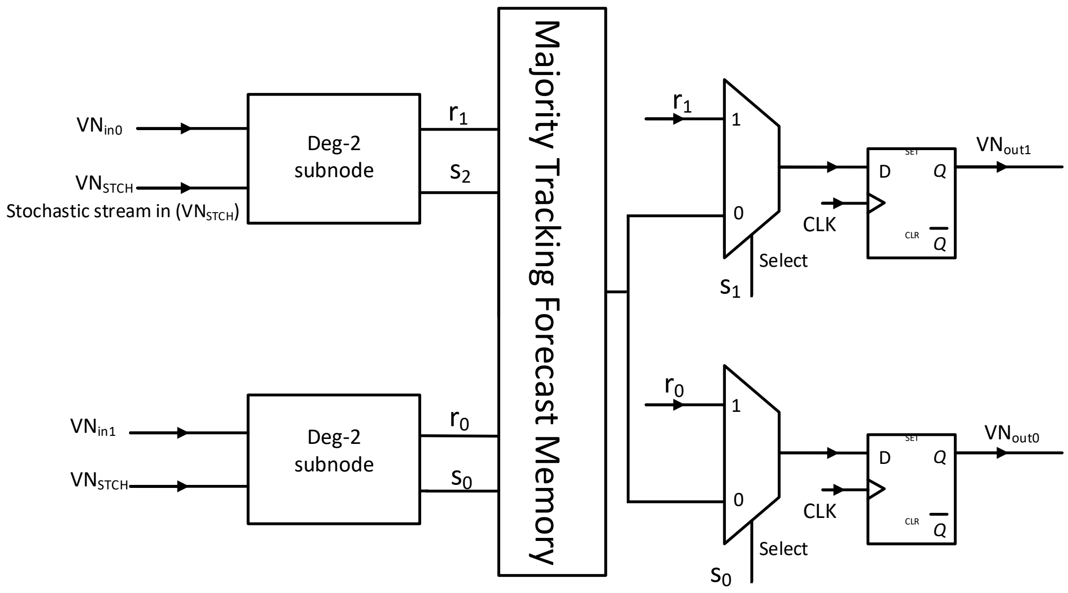

Figure 5.5: Degree-2 VN.

stochastic stream. However, this will result in at least, 1056 RNGs and hence, the total area consumed by the RNGs will be large. As a result, we want to find a way to use less number of RNGs without introducing dependency among the stochastic streams and the VN nodes. To achieve independent VN nodes, every two VNs sharing the same CN should have different RNG. By investigating the H-matrix and the factor graph of the code, we found that the minimum distance for 2 VNs to be sharing the same CN is 25. For example, the first 25 variable nodes don't share any CNs. This means that we can use the same RNG for them. As a result, we divided the VN nodes into groups of 25 VN nodes sharing the same RNG. However, this was true for the whole H matrix except for 2 consecutive nodes sharing the first CN node, the 747$^{\text{th}}$ and the 748$^{\text{th}}$ VNs (counting of VNs starts from 0). To solve this problem, we decreased the size of each group from 25 to 22. This makes the two VN nodes lie in different groups and have different RNGs. So the final number of RNGs used is equal to $1056/22 = 48$ RNGs.

There are many techniques to implement a random number generator in hardware. In our chip, we use linear feedback shift register shown in Figure 5.9. Since, the maximum

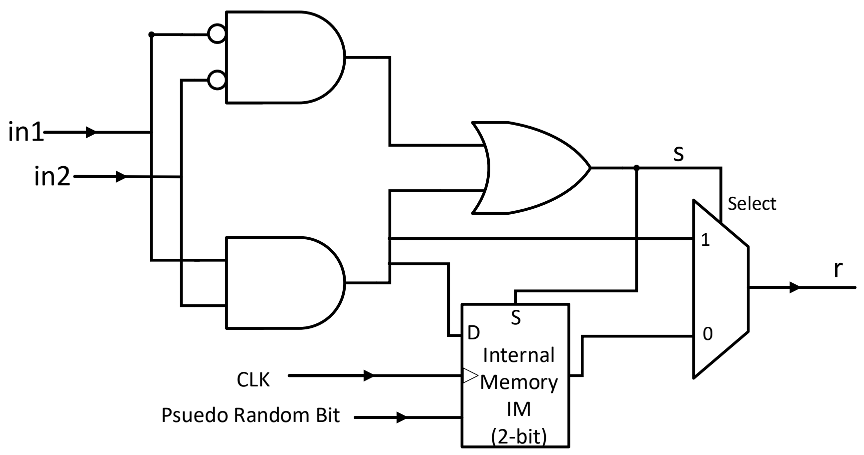

Figure 5.6: Degree-2 Sub-node with IM of the Degree-4 VN.

number of decoding cycles is 700, therefore, the number of the flip-flops in the shift register has to be greater than $\log_2 700$, (i.e., greater than 10). We use 12-bit and 13-bit shift registers, then, we XOR their outputs. 6 bits out of the 12 are used to generate the stochastic streams as the input probability is represented in 6 bits. The comparator used in the stochastic stream generation is shown in Figure 2.4. The rest of the bits are used inside the VN nodes for the IM and MTFM.

5.5 Full Chip Layout

Figure 5.10 shows the circuit layout after place and route from Cadence Encounter software. Figure 5.11 shows the final layout sent to fabrication after connecting the bondpads and after design rule checking (DRC) using Calibre tool.

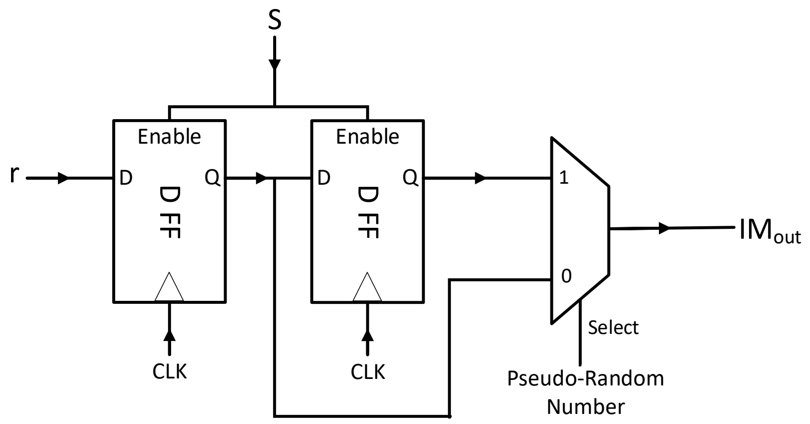

Figure 5.7: Architecture of the IM.

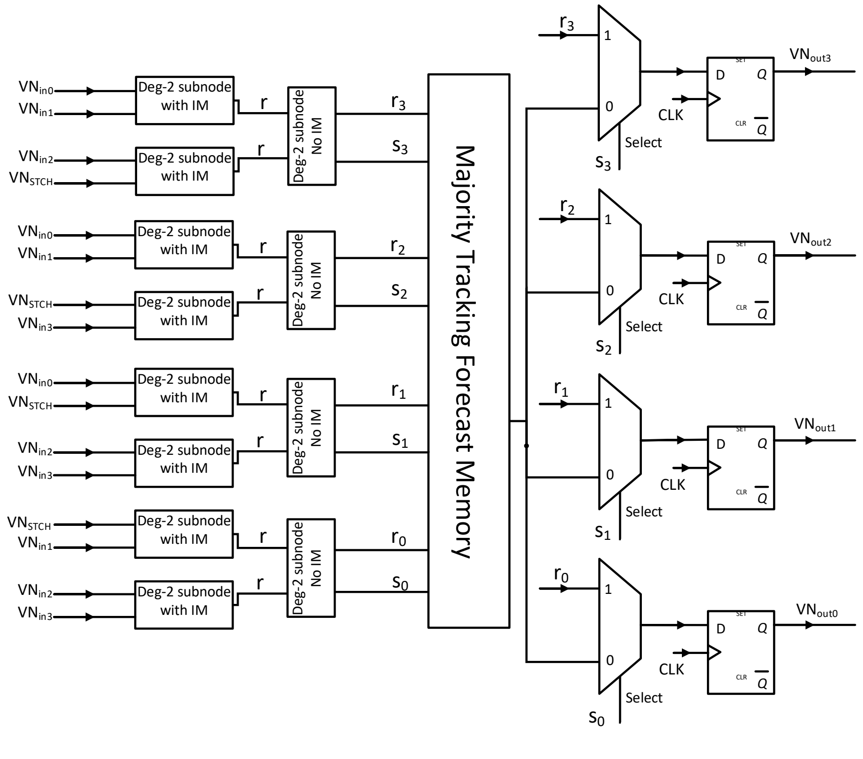

Figure 5.8: Degree-4 VN.

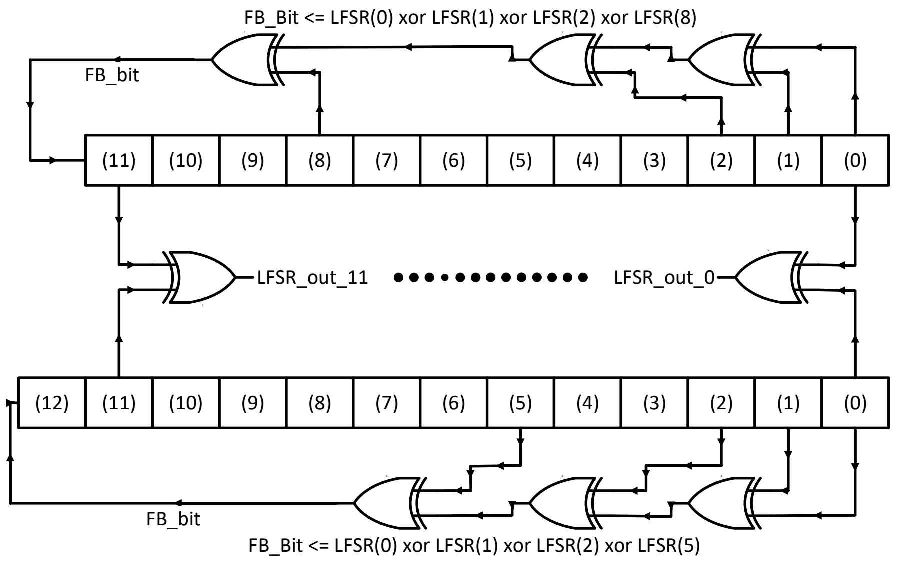

Figure 5.9: Architecture of a 12-bit Linear Feedback Shift Register.

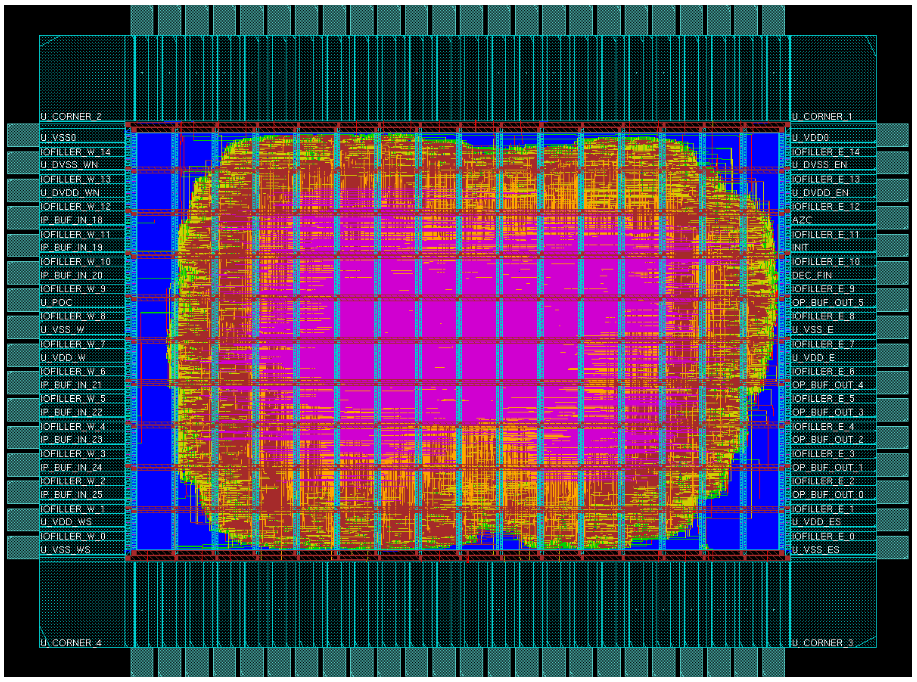

Figure 5.10: Chip Layout from Cadence Encounter.

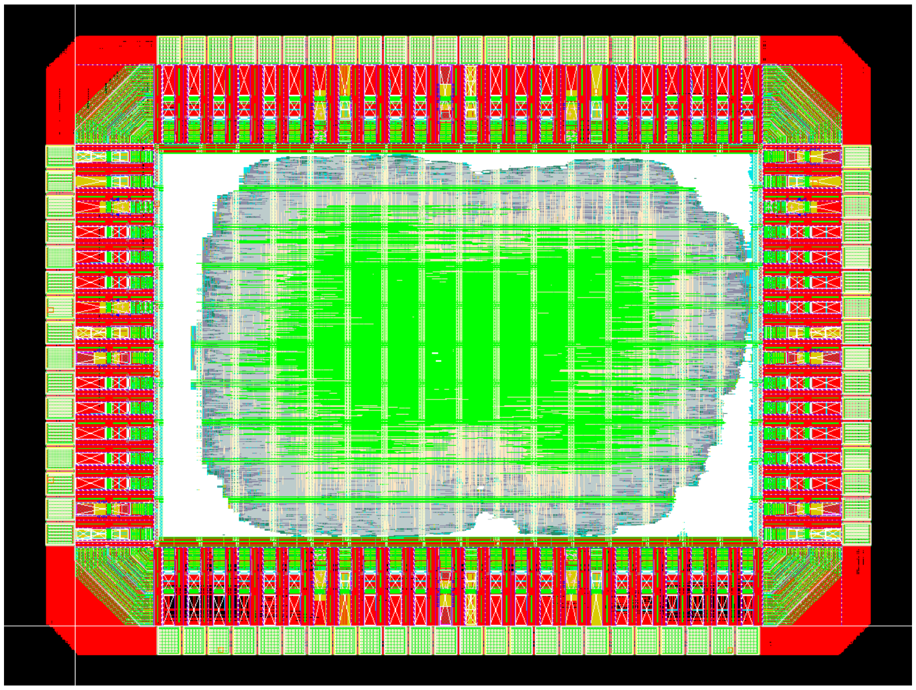

Figure 5.11: Final Chip Layout submitted to Fabrication.

Chapter 6

Conclusion and Future Work

In this thesis, we show that stochastic decoding is very tolerant to soft errors and has high capability of correcting them while giving a good performance near to the ideal one without errors. We started by introducing stochastic computing in Chapter 2. We discussed its strengths and weaknesses. We also introduced some of its promising applications. In Chapter 3, we discussed the stochastic LDPC decoding. We also introduced the general architecture used to build stochastic LDPC decoders. In Chapter 4, we introduce two models to emulate the soft errors that result from V_{DD} scaling or from overclocking the system. We also present the simulations results for these error models and proved that these results nearly give the same performance as the ideal case without errors. These results show that the stochastic decoding algorithm is very tolerant to soft errors. This can allow us to loosen the design constraints on stochastic decoders and push the limits of power or speed while achieving a good performance. We designed and implemented a stochastic LPDC decoder for the WiMAX (1056,704) LDPC code to test this on silicon.

Stochastic computing in general is a very promising research area and it is becoming a hot topic nowadays. Moreover, a lot of research can be done on the topic of stochastic decoding of LDPC codes and its tolerance to errors. Future work includes:

6.1 Testing Chip

The chip we implemented is still in fabrication. We plan to test it once it is back and verify our simulation results. An FPGA will be used to setup the testing environment for the chip.

6.2 Full Chip

As discussed in Chapter 6, the implemented chip is designed to test the tolerance of stochastic decoding to soft errors. Hence, designing a full working chip with a very high throughput or a scaled down V_{DD} utilizing the tolerance of the stochastic LDPC decoders to soft errors would be a valuable research work and is expected to give very good results compared to fabricated chips in the literature. It will also help meet the new communication standards with high speed low power LDPC decoders.

6.3 Channel Noise

The model we use for the channel noise is the AWGN channel. It would be great of we can study other different channel models and their effects on stochastic LDPC decoders. Moreover, it would be interesting to study the effect of the phase noise on stochastic LDPC decoders. In addition, we need to study how we can minimize those noise effects utilizing the strengths of stochastic LDPC decoding.

6.4 Stochastic Decoding for Galois Field Greater Than Two

In this thesis we have been working with stochastic decoding of LDPC codes with binary Galois field (GF-(2)). We think, it would good if we can apply stochastic decoding on

Galois field greater than 2. This can also open the door to applying quantum computing techniques on stochastic computing algorithms in general and stochastic decoding in particular [28]. In quantum computations, a qubit or a quantum bit can represent a 1 or a 0 or both at the same time. As a result, Quantum bits can replace the stochastic stream of bits and hence, quantum computations can replace the normal stochastic decoding operations.

References

[1] Brian R Gaines et al. Stochastic computing systems. *Advances in information systems science*, 2(2):37–172, 1969.

[2] Vincent C Gaudet and Anthony C Rapley. Iterative decoding using stochastic computation. *Electronics Letters*, 39(3):299–301, 2003.

[3] Frank R Kschischang, Brendan J Frey, and H-A Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

[4] Robert Gallager. Low-density parity-check codes. *IRE Transactions on information theory*, 8(1):21–28, 1962.

[5] David JC MacKay and Radford M Neal. Near shannon limit performance of low density parity check codes. *Electronics letters*, 32(18):1645, 1996.

[6] Saeed Sharifi Tehrani, Warren J Gross, and Shie Mannor. Stochastic decoding of ldpc codes. *IEEE Communications Letters*, 10(10):716–718, 2006.

[7] Saeed Sharifi Tehrani, Shie Mannor, and Warren J Gross. Survey of stochastic computation on factor graphs. In *Multiple-Valued Logic, 2007. ISMVL 2007. 37th International Symposium on*, pages 54–54. IEEE, 2007.

[8] Saeed Sharifi Tehrani, Shie Mannor, and Warren J Gross. Fully parallel stochastic ldpc decoders. *IEEE Transactions on Signal Processing*, 56(11):5692–5703, 2008.

[9] S. S. Tehrani, A. Naderi, G. A. Kamendje, S. Mannor, and W. J. Gross. Tracking forecast memories in stochastic decoders. In *2009 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 561–564, April 2009.

[10] Saeed Sharifi Tehrani, Ali Naderi, Guy-Armand Kamendje, Saied Hemati, Shie Mannor, and Warren J Gross. Majority-based tracking forecast memories for stochastic ldpc decoding. *IEEE Transactions on Signal Processing*, 58(9):4883–4896, 2010.

[11] Saeed Sharifi Tehrani, Ali Naderi, Guy-Armand Kamendje, Shie Mannor, and Warren J Gross. Tracking forecast memories for stochastic decoding. *Journal of Signal Processing Systems*, 63(1):117–127, 2011.

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

[13] Naoya Onizawa, Vincent C Gaudet, Takahiro Hanyu, and Warren J Gross. Asynchronous stochastic decoding of low-density parity-check codes. In *Multiple-Valued Logic (ISMVL), 2012 42nd IEEE International Symposium on*, pages 92–97. IEEE, 2012.

[14] Naoya Onizawa, Warren J Gross, Takahiro Hanyu, and Vincent C Gaudet. Lowering error floors in stochastic decoding of ldpc codes based on wire-delay dependent asynchronous updating. In *Multiple-Valued Logic (ISMVL), 2013 IEEE 43rd International Symposium on*, pages 254–259. IEEE, 2013.

[15] Xin-Ru Lee, Chih-Lung Chen, Hsie-Chia Chang, and Chen-Yi Lee. A 7.92 gb/s 437.2 mw stochastic ldpc decoder chip for ieee 802.15. 3c applications. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62(2):507–516, 2015.

[16] Isaac Perez-Andrade, Xin Zuo, Robert G Maunder, Bashir M Al-Hashimi, and Lajos Hanzo. Analysis of voltage-and clock-scaling-induced timing errors in stochastic ldpc decoders. In *Wireless Communications and Networking Conference (WCNC), 2013 IEEE*, pages 4293–4298. IEEE, 2013.

[17] Christiane L Kameni Ngassa, Valentin Savin, and David Declercq. Faulty stochastic ldpc decoders over the binary symmetric channel. In *Turbo Codes and Iterative Information Processing (ISTC), 2014 8th International Symposium on*, pages 112–116. IEEE, 2014.

[18] Xin Zuo, Isaac Perez-Andrade, Robert G Maunder, Bashir M Al-Hashimi, and Lajos Hanzo. Improving the tolerance of stochastic ldpc decoders to overclocking-induced timing errors: A tutorial and a design example. *IEEE Access*, 4:1607–1629, 2016.

[19] Lav R Varshney. Performance of ldpc codes under faulty iterative decoding. *IEEE Transactions on Information Theory*, 57(7):4427–4444, 2011.

[20] The IEEE P802.3an 10GBASE-T Task Force [Online]. Available: www.ieee802.org/3/an.

[21] The IEEE 802.16 Working Group [Online]. Available: http://www.ieee802.org/16/.

[22] John Von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. *Automata studies*, 34:43–98, 1956.

[23] John William Esch. -rascel-a programmable analog computer based on a regular array of stochastic computing element logic. 1969.

[24] Phil Mars and Wolfgang J Poppelbaum. *Stochastic and deterministic averaging processors*. Number 1. Peter Peregrinus Press, 1981.

[25] Armin Alaghi, Cheng Li, and John P Hayes. Stochastic circuits for real-time image-processing applications. In *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, pages 1–6. IEEE, 2013.

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

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

[28] Armin Alaghi. *The Logic of Random Pulses: Stochastic Computing.* PhD thesis, University of Michigan, 2015.

[29] Brian R Gaines et al. Stochastic computing systems. *Advances in information systems science*, 2(2):37–172, 1969.

[30] Saeed Sharifi Tehrani. *Stochastic Decoding of Low-Density Parity-Check Codes.* PhD thesis, McGill University, 2011.

[31] Jan M Rabaey, Anantha P Chandrakasan, and Borivoje Nikolic. *Digital integrated circuits*, volume 2. Prentice hall Englewood Cliffs, 2002.

[32] Jeffery A Dickson, Robert D McLeod, and HC Card. Stochastic arithmetic implementations of neural networks with in situ learning. In *Neural Networks, 1993., IEEE International Conference on*, pages 711–716. IEEE, 1993.

[33] Young-Chul Kim and Michael A Shanblatt. Architecture and statistical model of a pulse-mode digital multilayer neural network. *IEEE transactions on neural networks*, 6(5):1109–1118, 1995.

[34] Ao Ren, Zhe Li, Yanzhi Wang, Qinru Qiu, and Bo Yuan. Designing reconfigurable large-scale deep learning systems using stochastic computing. In *Rebooting Computing (ICRC), IEEE International Conference on*, pages 1–7. IEEE, 2016.

[35] Tarik Hammadou, Magnus Nilson, Amine Bermak, and Philip Ogunbona. A 96/spl times/64 intelligent digital pixel array with extended binary stochastic arithmetic. In *Circuits and Systems, 2003. ISCAS'03. Proceedings of the 2003 International Symposium on*, volume 4, pages IV–IV. IEEE, 2003.

[36] 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, 2011.

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

[38] Xin Li, Weikang Qian, Marc D Riedel, Kia Bazargan, and David J Lilja. A reconfigurable stochastic architecture for highly reliable computing. In *Proceedings of the 19th ACM Great Lakes symposium on VLSI*, pages 315–320. ACM, 2009.

[39] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[40] Niclas Wiberg. Codes and decoding on general graphs. 1996.

[41] Chris Winstead, Vincent C Gaudet, Anthony Rapley, and Christian Schlegel. Stochastic iterative decoders. In *Information Theory, 2005. ISIT 2005. Proceedings. International Symposium on*, pages 1116–1120. IEEE, 2005.

[42] Chris Winstead. Error-control decoders and probabilistic computation. In *Tohoku Univ. 3rd SOIM-COE Conf., Sendai, Japan*, 2005.

[43] Vincent C Gaudet and Warren J Gross. Switching activity in stochastic decoders. In *Multiple-Valued Logic (ISMVL), 2010 40th IEEE International Symposium on*, pages 167–172. IEEE, 2010.

[44] Assem Hussein, Mohamed Elmasry, and Vincent Gaudet. On the fault tolerance of stochastic decoders. In *Multiple-Valued Logic (ISMVL), 2017 IEEE 47th International Symposium on*. IEEE, 2017. accepted.