COMPLEXITY AND POWER CONSUMPTION IN STOCHASTIC ITERATIVE DECODERS

by

Keyur M Payak

A thesis submitted in partial fulfillment
of the requirements for the degree

of

MASTER OF SCIENCE

in

Electrical Engineering

Approved:

Dr. Chris Winstead Major Professor	Dr. Todd Moon Committee Member
Dr. Sanghamitra Roy Committee Member	Dr. Byron R. Burnham Dean of Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2010

Abstract

Complexity and Power Consumption in Stochastic Iterative Decoders

by

Keyur M Payak, Master of Science

Utah State University, 2010

Major Professor: Dr. Chris Winstead
Department: Electrical and Computer Engineering

Stochastic iterative decoding is a novel method to decode the bits received at the end of a communication channel and to control the rate of error happening in the message bits due to noise being injected into the channel. This decoder uses stochastic computation that is based on manipulation of probabilities from a random sequence of digital bits. Hardware needed for implementing this arithmetic is very simple and can be completely implemented using simple digital complementary metal oxide gates. This helps the decoder to be technology independent, which is a major advantage over its digital and analog counterparts, which are complex and technology dependent. But this decoder presents a new set of problems when nodes in stochastic decoders can get locked to a fixed state if the stochastic streams are correlated due to the presence of cycles in a decoder's factor graph. To overcome this problem, additional logic has to be introduced on every edge of the decoder to break this correlation. This work presents application-specific-integrated-circuit (ASIC) design and simulation of the digital core of a stochastic iterative decoder in 0.18μm technology (Spectre). This thesis also examines gate complexity and power consumption of the decoder with edge-memory, tracking forecast memory, and dual-counter hysteresis techniques in place.

(52 pages)

To my late Grandfather....

Acknowledgments

I would like to thank Dr. Chris Winstead for his valuable suggestions on my thesis topic. His in-depth knowledge in the field of error correction coding and iterative decoder implementations helped me in the simulation and design phase of my thesis. I would also like to thank him for the courses that he taught and the projects that he designed, which helped me gain an insight into analog, mixed signal, and digital systems. I would also like to thank my committee members, Dr. Todd Moon and Dr. Sanghamitra Roy, for their kind support and motivation.

Very special thanks to my parents, my brother, and my fiancee, Nirmal, without whom I could not have moved a foot. My friends (Arun, Madhu, Hemang, Nisha, Balaji, Swadesh) here at USU have been extremely helpful and supportive. I definitely owe them one.

I am also grateful to the Department of Mathematics and Statistics for supporting me financially throughout my coursework here at USU. Special thanks to Richard Cutler, Chris Corcoran, Mevin Hooten, Cindy Moulton, and Nancy Smart.

Keyur M. Payak

Contents

List of Tables

List of Figures

Acronyms

ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
CMOS	Complementary Metal Oxide Semiconductor
DSP	Digital Signal Processor
EM	Edge Memory
LDPC	Low-Density Parity Check Code
LLR	Log-Likelihood Ratio
MAP	Maximum A-Posteriori
MOS	Metal Oxide Semiconductor
RNG	Random Number Generator
SoC	System-on-Chip
SPA	Sum-Product Algorithm
TFM	Tracking Forecast Memory
TSMC	Taiwan Semiconductor Manufacturing Company

Chapter 1

Introduction

1.1 Motivation

Since the advent of Turbo Codes [1] and Low-Density Parity-Check Codes [2] and also with their introduction into a number of digital communuication standards, much research effort is invested into realization of high-speed and low-power architectures for iterative decoding of Turbo and LDPC codes. Stochastic iterative decoding introduces one such architecture to implement high-speed, fully-parallel, and low-power iterative decoder.

Stochastic iterative decoding is a method to decode the bits received at the end of a communication channel and to control the number of errors happening in a channel. Iterative decoding using stochastic computation was first proposed by Gaudet and Rapley for error control decoding, which were based on the technique of manipulation of probabilities from a random sequence of digital bits [3]. Hardware needed for implementing this arithmetic is very simple, consisting of parity-check and equality nodes. These nodes are interlaced in a network in which stochastic bits flow to-and-fro concurrently until a stopping criterion is reached. A decoder using stochastic iterative algorithm can potentially operate at high-speeds with low-power consumption [3]. Stochastic decoding algorithm is an instance of the sum-product algorithm [4], the only difference being that the messages are represented using a Bernoulli sequence of 0s and 1s instead of the conventional method which represented messages using fixed-point arithmatic.

Analog continuous time iterative decoders have been implemented in the past, using metal oxide semiconductor (MOS) transistors operating in sub-threshold regime. This implementation represents internal decoder messages using continuos voltages/currents. Some features of this implementation included low-power, and high-speed [5]. But this implementation is complex and technology dependent. So a "digital-like" implementation of this

analog decoder was desired which would be simple to implement and would be technology independent.

1.2 Contribution of This Thesis

The gray box in fig. 1.1 refers to the emphasis area of this thesis. Stochastic iterative decoders have been implemented using FPGAs in the past [6, 7], but this thesis presents application-specific-integrated-circuit (ASIC) design of all the components of a stochastic iterative decoder and their gate complexity and power consumption are examined. Various techniques are shown to minimize the power consumption at every edge of the decoder. Following are the contributions of this thesis.

1. Complete ASIC implementation of all components of a stochastic iterative decoder in TSMC 0.18μm technology.

2. Gate complexity and power consumption of three variants of the decoder - dual-counter, edge memory, and tracking forecast memory (TFM) are compared and trade-offs are evaluated based on simulation results.

3. Power consumption of the decoder is examined against increasing values of signal-to-noise ratio (SNR) using data obtained from system level C simulations and Spectre simulations. Consequently, a relationship between power consumption and SNR is derived.

4. Power consumption of the decoder is studied for various technologies and feature sizes (0.18μm, 0.35μm and 0.60μm) and is plotted with respect to SNR. It is shown how power consumption of the decoder changes with decreasing transistor channel lengths.

5. Speed and convergence of dual counters and tracking forecast memories are compared and conclusions are derived.

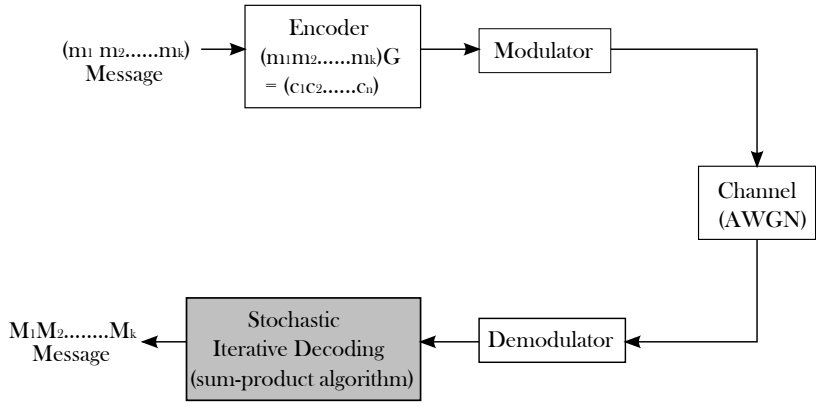

Fig. 1.1: Flow of message in a communication channel.

1.3 Outline of Thesis

Chapter 2 begins with literature review and then goes on to describe the stochastic algorithm and also illustrates the problem of locking or latching. Chapter 3 delves into implementation details for dual-counter, edge-memory, and tracking forecast memory schemes for solving the latching problem. Chapter 4 deduces results and discusses trade-offs between the dual-counter, edge-memory, and TFM schemes. Chapter 5 offers conclusions and future work.

Chapter 2

Literature Review and Stochastic Algorithm

2.1 History

In the last few years, the demand for efficient and reliable digital data transmission has tremendously increased. Researchers answer this problem with devising more efficient coding schemes which are as close as it can get to the shannon limit. Two of such coding schemes are Turbo codes and Low-Density-Parity-Check codes (LDPC), which have outperformed all other existent codes. However, the hardware implementation of such codes is complex and require precision circuitry. Also, with increase in speed and code complexity, power consumption of the decoder increases.

Traditionally, iterative decoders used pure digital logic to implement the message passing algorithm [8–10]. However, with increase in clock frequency the complexity, processing speeds, and power consumption went up which is undesirable for many applications (e.g. cellular phones and other mobile devices). Besides, simplified iterative decoders using regular LDPC codes have also been implemeneted on a DSP [11].

To address this issue, it was shown by Loeliger et al. [12, 13] that iterative decoder modules can be realized using simple analog transistor circuits which exhibit exponential current-voltage relationship. The entire decoder can be implemented using variations of a single circuit using the translinear principle, and hence probabilty propagation on decoder factor graph was made simple using these simple BiCMOS/CMOS analog circuits. Following this observation, a variety of decoders (MAP, Turbo, LDPC, etc.) were implemented using analog voltage/current as an internal metric [14–18]. These implementations revealed following advantages of analog decoders over their digital counterparts.

1. Design of an analog decoder is more neat and systematic as only few basic cells are required and these can be reused over and over again for the factor graph representation of the decoder.

2. Analog decoders have considerably lower power consumption than digital decoders as transistors are operating in subthreshold regime which require very low input currents. This, however, results in low-speed circuits as considerable amount of time is required for charging up circuit parasitics.

3. Internal metrics of the decoder are represented using real numbers (in the form of analog voltages/currents), which are represented using several digital bits in digital decoders. This makes such decoders more elegant [18].

4. The iterations disappear, i.e. the decoder works like a asynchronous circuit that stabilizes itself [12]. This is one of the main attractions for an analog decoder.

5. CMOS analog decoders are ideal for their use in System-On-Chip (SoC) designs as they can be placed alongside other CMOS digital signal processors on the same chip. Slowness of analog decoders can be advantageous in this case as it inherently rejects high frequency noise from adjacent digital and RF circuits [19].

On the other hand analog decoders suffered from following disadvantages.

1. Mismatch, noise, channel length modulation, and other nonlinearities in CMOS circuits affect decoder's performance. Also, decoder's convergence gets degraded if the circuit operates in strong inversion.

2. Analog decoders are not technology-independent, i.e. with ever scaling CMOS technologies, analog decoders does not scale proportionally and the design has to be revised if ever a technology migration has to take place.

To overcome these problems in analog decoders a digital-like version of analog decoder was desired. It was first proposed by Gaudet and Rapley [3] that iterative decoders using

the stochastic computation can operate at high speeds and can be implemented with low-precision digital gates without any loss of benifits offered by analog decoders. This approach offered several advantages.

1. Complete decoder can be realized using simple digital logic gates like XOR, NOR, etc. which are relatively easy to design using modern CAD tools.

2. Stochastic iterative decoders are technology-independent as device sizes would scale proportionally with scaling feature sizes and supply voltage. Thus, a decoder using stochastic computation would be easy to migrate to a different CMOS technology.

3. With the simple implementation using minimum sized transistors for digital gates, the total area of the chip can be predicted to be considerably lower than conventional decoders. Although, this is yet to verified.

Stochastic algorithm was further extended by Rapley et al. and Tehrani et al. [4,20,21] and it was discovered that local processing elements in hardware implementation of this algorithm are prone to locking or latching. This corresponds to nodes in the factor graph getting locked to a fixed state if the incoming stochastic streams to the node are correlated. However, several techniques to this problem were proposed which aimed at breaking the correlation in the stochastic streams. With these techniques inplace, implementations of stochastic decoders on FPGAs have been reported by Winstead et al. and Tehrani et al. [6, 7, 22, 23].

2.2 Brief Review on Error Correction

Figure 2.1 shows how a transmitted BPSK signal gets modified after being injected into noisy channel. The signal can be misinterpreted to be 0 instead of 1 or vice versa at the receiver due to the influence of noise.

In order to overcome this phenomenon, data is embedded with some extra bits which impose rules on the transmitted bit sequence. If any of these rules are violated in the received bit sequence, then the received bit stream is flagged as erroneous. These rules are

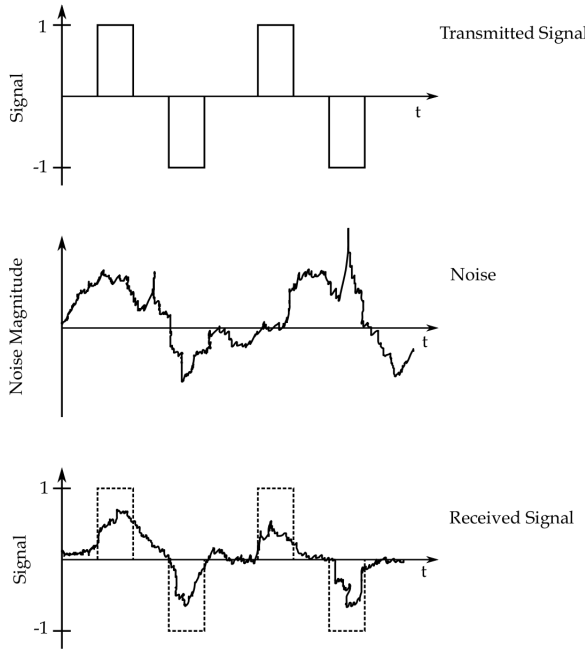

Fig. 2.1: Effect of noise on the transmitted signal.

called as parity checks. A better performance is achieved by having multiple interlocking rules or parity checks. These rules are defined through a matrix called as the parity-check matrix H. For example a parity check matrix could be defined as:

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix}. \tag{2.1}$$

A data sequence $\vec{x}$ contains an error if $H \times \vec{x} \neq 0$ where $\vec{x}$ is the received vector. If $H \times \vec{x} \neq 0$, then the resultant pattern tells the unique position where the single bit error has occured. Let us say the recevied bit sequence $\vec{x}$ is $\begin{bmatrix} 1 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$. Then equation (2.2) indicates that an error occured at position $\begin{bmatrix} 0 & 1 & 1 \end{bmatrix}$.

The rules defined by the parity check matrix H are represented graphically using a factor graph. Figure 2.2 shows a factor graph for the above mentioned parity check matrix. The circles are called as variable nodes and act as input/output nodes and the boxes (+) are called as parity check nodes. There is an edge between a variable node and a check

node if the corresponding parity check matrix contains a 1 on the corresponding row and column.

$$H \times \vec{x} = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}. \tag{2.2}$$

2.3 Stochastic Computation on Factor Graphs

Stochastic algorithm is an approximation to sum-product algorithm [4] that works through probability propagation on edges connecting variable and check nodes in a codes factor graph. These probabilities are updated on every node according to a constraint function and the updated message is sent back on the connecting edges.

2.3.1 Stochastic Computation

Stochastic algorithm performs operations corresponding to sum-product algorithm where internal metrics probabilities are represented using Bernoulli sequence of bits. This algorithm is much simpler to implement at circuit level as it eliminates most of the complicated operations involved in sum-product algorithm (SPA). Using stochastic computation, a probability ρ is represented using the following equation:

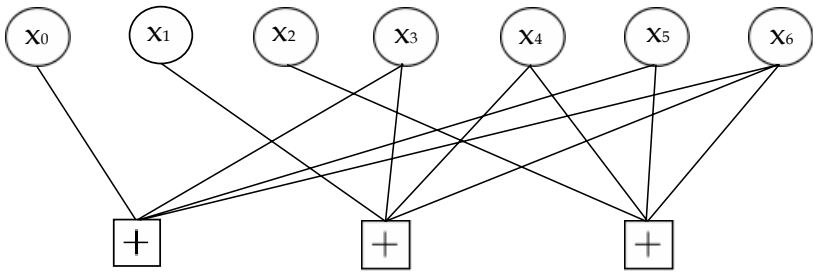

Fig. 2.2: Sample factor graph.

$$\rho = \frac{\text{Number of ones in the stochastic stream}}{\text{Total number of bits in the stochastic stream}}. \tag{2.3}$$

Therefore, in order to represent a probability of 0.75, then there are multiple ways which have same number of ones and zeros but at different locations. Figure 2.3 shows three ways to represent a probability of 0.75. This way of representing probabilities makes operations such as multlipication, division, etc., uncomplicated and can be implemented using simple logic gates.

2.3.2 Decoder Operation

When a codeword is received, the analog noisy value of each bit is converted to a probability of being a binary 1. These analog probabilities are then converted to a digital bit stream which can be done using oversampling A/D converters [3, 21]. This digital bit stream is then converted to a stochastic stream using the structure shown in fig. 2.5.

These random bit streams are then passed on to the equality nodes. Figure 2.4 shows an overall architechture of stochastic decoding. Up/down counters at the output of the decoder are incremented each time a 1 is encountered and decremented each time a 0 is encountered. Finally, comparators are used to make hard decisions from the the extrinsic information from the decoder. Decoding is performed by sending initial probabilities to and fro between the equality and check nodes in the factor graph. Each node performs an update and sends it back to its adjacent nodes which perform their own update and send it back to all other nodes attached to it in the factor graph. This process continues until a specified number of iterations are performed.

2.3.3 Node Updates

Check nodes does an update according to the following equation [4, 5, 20, 21]. This equation is an equivalent to check node update rule defined in conventional SPA.

$$\rho_C = \rho_A(1 - \rho_B) + (1 - \rho_A)\rho_B, \tag{2.4}$$

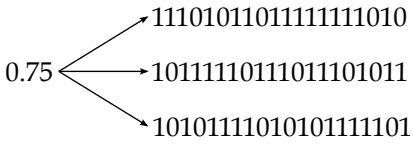

Fig. 2.3: Probability representation in stochastic algorithm.

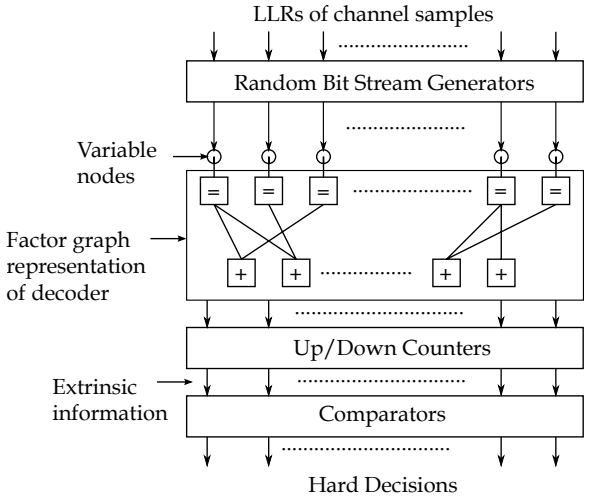

Fig. 2.4: Stochastic iterative decoder architecture.

where ρ_A and ρ_B are corresponding incoming probabilities and ρ_C is the outgoing probability. Figure 2.6 shows the equivalent circuit diagram of a 3-edge stochastic check node.

Similarly, an equality node is defined by the equation:

$$\rho_C = \frac{\rho_A \rho_B}{\rho_A \rho_B + ((1-\rho_A)(1-\rho_B))}. \tag{2.5}$$

Following from this equation, the equivalent circuit of a 3-edge equality node is shown in fig. 2.7. This circuit is equivalent in function to a classic Muller C-element.

2.4 Locking in Stochastic Iterative Decoders

Stochastic update rule at any node of the decoder is given by equation (2.6) [4].

$$C(t) = \begin{cases} f_c(A_r(t), B_r(t)) & \text{if } (\mathrm{A_r(t)}, \mathrm{B_r(t)}) \in \mathrm{S} \\ C_t(t-1)) & \text{otherwise,} \end{cases} \tag{2.6}$$

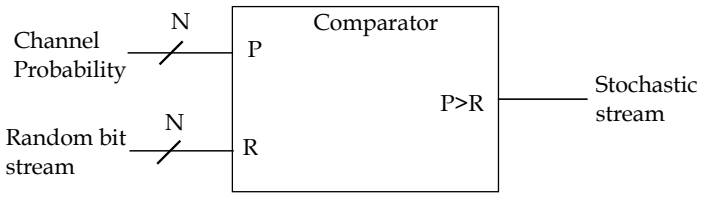

Fig. 2.5: Generation of stochastic stream from channel probabilities.

where $C(t)$ is the transmitted bit from a stochastic gate, $A_r(t)$ and $B_r(t)$ are the received bits at time t. f_c is the constraint function which the gate implements and S is satisfaction set which is a set of all allowed values of $A_r(t)$ and $B_r(t)$.

In a nutshell, every node in the factor graph is defined by a function $f(A, B)$ and if at any time t the node's constraint function is undefined, then a memory element is required to produce the output at time $(t-1)$. This memory element is a simple D flip flop for a parity check node and a J-K flip flop for a equality node (for normalization). This flip-flop memory is the root cause for nodes in the code's factor graph to get locked up. Locking (a.k.a latching) in stochastic iterative decoders is defined as the phenomenon when nodes become reluctant to produce a fixed output regardless of different inputs from the variable nodes. This can happen when the decoders constraint graph contains one or more cycles, due to which incoming messages on some of equality nodes become correlated [4]. Besides, stochastic decoders are also sensitive to the level of random switching activity within the circuit. As the SNR increases, received LLRs can become so large that the corresponding probability is driven to one or zero. In this case random switching events become too rare, which can cause locking of equality nodes and can also lead to slow convergence at high SNRs [20]. Therefore, the memory element present in the nodes of the factor graph have to be eliminated and must be redesigned so that the correlation between stochastic streams is

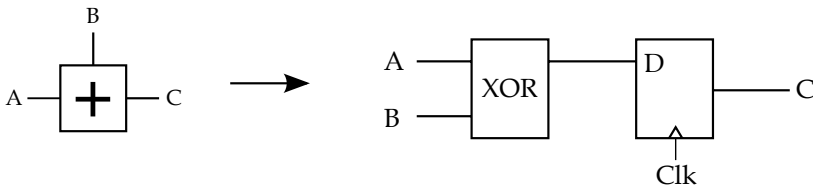

Fig. 2.6: Circuit representation of a check node.

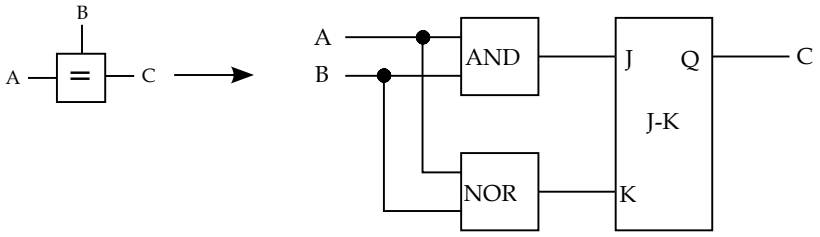

Fig. 2.7: Circuit representation of a 3-edge equality node.

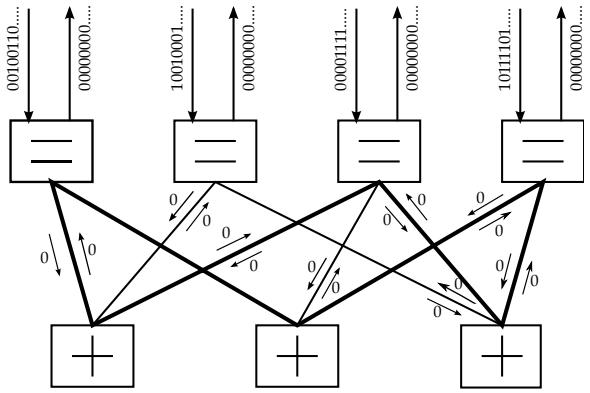

Fig. 2.8: Locking in factor graph.

disrupted. Figure 2.8 shows an example of locking in stochastic decoders. Here bold lines represent a cycle.

Chapter 3

Design of Anti-Locking Stochastic Decoders

Locking in stochastic decoders can be prevented by disrupting the correlation between the incoming messages on equality nodes. This can be done by placing additional logic on the output of a equality node, which will replace the incoming message with a new message, such that no correlation exists between the two. Techniques to prevent locking are often termed as hysteresis techniques and the additional logic that has to placed can therefore be termed as a hysteretic filter [24]. This chapter presents transistor level design of two of the widely used hysteresis techniques: edge memory and dual counter. All designs presented use TSMC 0.18μm technology and are simulated using Spectre.

3.1 Equality and Parity Check Gates

Equations (2.4) and (2.5) define the stochastic update rules for check node and equality node, respectively. Figures 3.1 and 3.2 show how equality nodes and parity check can be realized by using equations (2.4) and (2.5) [25].

3.2 Edge Memory

Edge memories were first introduced by Tehrani et al. [20] and this technique inserts M-bit shift registers at the output of each equality node as shown in fig. 3.3. During a regenerative state, the regenerative bit is added into the shift register, and the output equals the value of the regenerative bit. During a non-regenerative state, a shift-register address is randomly chosen, and the output is set equal to the bit at that address. In some implementations, the selected bit is removed from the shift-register, so that the register's size is reduced by one during a non-regenerative condition. In this thesis, all simulations are done using a fixed 32-bit shift register with regenerative bit adding to the front of the

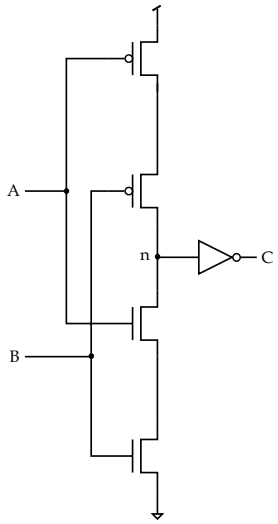

Fig. 3.1: Equality gate.

shift register by shifting the previous bit. A linear feedback shift register is used to initialize the 32-bits in the edge memory with random bits.

A standard edge memory design requires a 32-bit shift register, 32:1 multiplexer for random bit selection from the shift register, a 5-bit random number generator (RNG) used for random addressing, and a 2:1 multiplexer for selection between the regenerative bit and the random bit from the shift register.

3.2.1 Shift Register

A serial-in, parallel-out shift register is required in an edge memory. A N-bit shift register requires placing of N flip-flops in series and drawing parallel outputs from each flip flop. Figure 3.4 represents how a shift register can be constructed from N-transistors and inverters. Charge stored at the gate of an inverter acts as memory.

3.2.2 Random Number Generator

As stated before, during a hold state a random bit is chosen from the edge memory and is passed on to the edge. This random bit selection requires a random number generator which will provide a valid random address, from where the bit will be picked. For a 32-bit shift register, a valid address can be 5 bits long, hence a 5-bit random number generator is

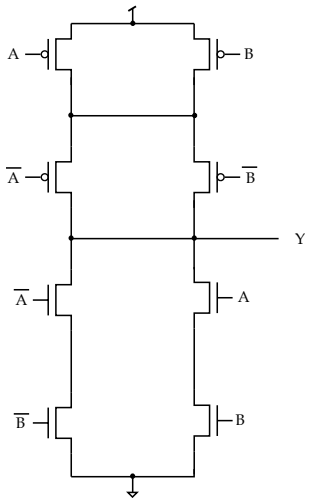

Fig. 3.2: Parity check gate.

required. We choose Galois LFSR implementation for random number generation instead of a conventional LFSR structure as this has considerably less propagation delay [26]. Figure 3.5 shows a 5-bit RNG with taps at positions 2,3,4, and 5. Note that XNOR gates are used to calculate tap values instead of conventional XOR gates, as XOR gates can potentially lock all flip flops in the register to a zero state. This locking can be prevented by a special reset pin for each flip flop, which can be implemented using pass transistor logic, but this comes at the cost of additional transistors per flip flop, and for a 32-bit edge memory, this would mean additional 64 transistors per edge, which can add up to large numbers for bigger codes and hence, is undesirable. LFSRs with XNOR gates are as efficient as with XOR gates, but with a lower gate count, since there is no need for a reset circuitry.

3.2.3 Regenerative Bit Detector

Regenerative bit detector is required to detect whether the incoming bits are equal or not. Figure 3.6 shows a regenerative bit detector. Besides having AND and NOR gates for checking the equality of the two inputs, this detector also generates a input for the select line for 2:1 multiplexer.

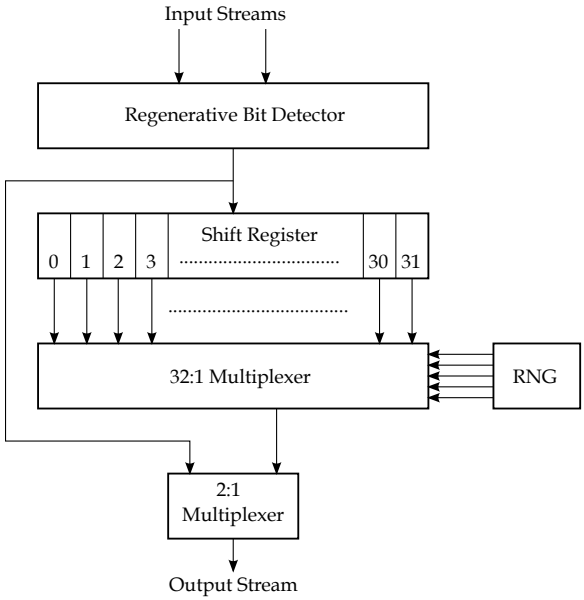

Fig. 3.3: High-level representation of an edge memory.

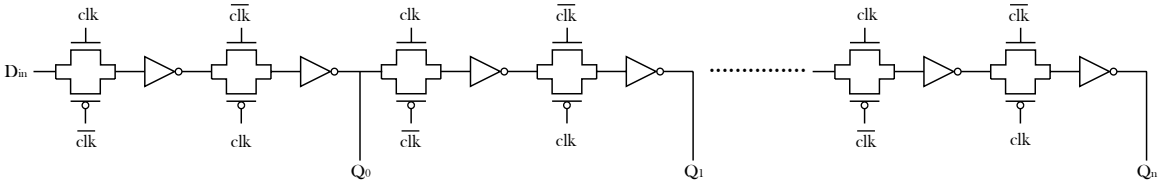

Fig. 3.4: Schematic of a shift register.

3.3 Dual Counter

Dual counter hysteresis was first introduced by Winstead et al. [4]. Figure 3.7 represents a general block diagram of dual counter hysteresis technique. This scheme directly replaces the memory imposed by the equality node. A dual counter architecture maintains two counters, namely one-counter and a zero-counter, which increase by a step size δ, whenever a 1 is encountered for a one-counter (or 0 for a zero-counter). A comparator compares a randomly generated number with the difference of the two counts (added to 0.5) and gives an output bit C. If both inputs to the equality gate are equal, then the counters are updated and the output bit C is the regenerative bit. Otherwise, the output bit is a randomly generated bit from a random number generator (RNG) and the corresponding counter is decremented. In this way, a new bit stream is generated which has same probability and is uncorrelated with the orignal stream.

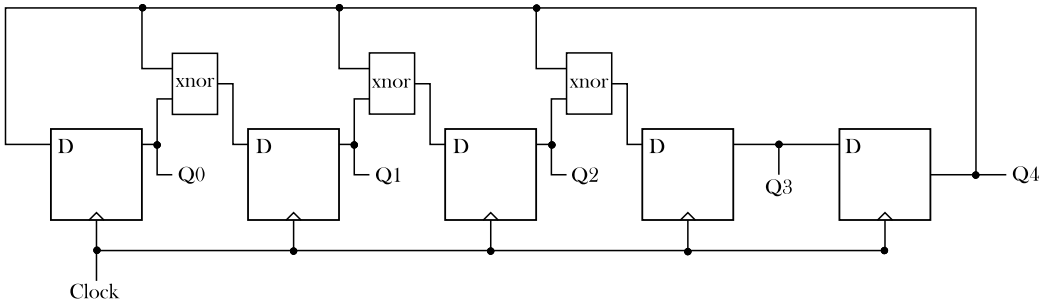

Fig. 3.5: Circuit schematic of a 5-bit Galois random number generator.

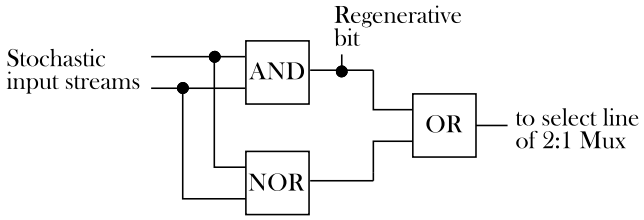

Fig. 3.6: Circuit schematic of a regenerative bit detector.

A standard dual counter is designed using 7-bit zero and one counters, an 8-bit adder to add the zero and one count, a 8-bit RNG to generate a random number at the comparator input, and finally a 8-bit pipelined comparator to compare the random number generators output with that of the adders output, to produce a random bit, C, at the output.

3.3.1 Adder

An 8-bit adder is required to add the zero and one counts. Note that the zero count is inverted to essentially perform a subtraction operation rather than addition. A carry lookahead adder is chosen for this design. For 8-bit additions carry lookahead logic consumes much more gates than a simple ripple carry adder but has reduced propagation delay. Equations (3.1) and (3.2) describe the generate and propogate functions for a stage i and equation (3.3) gives the carry bit for the next stage in the ripple carry adder.

$$g_i = x_i y_i \tag{3.1}$$

$$p_i = x_i + y_i \tag{3.2}$$

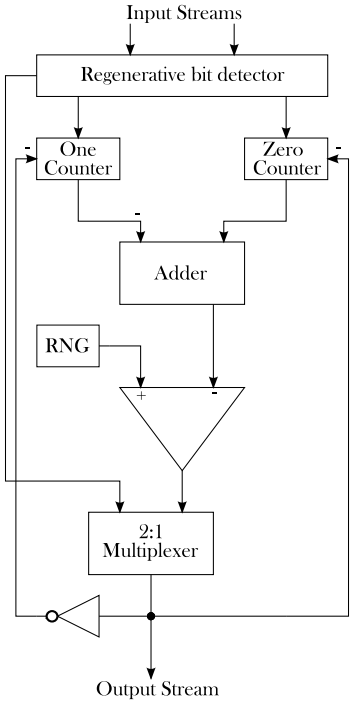

Fig. 3.7: High-level representation of a dual counter scheme.

$$c_{i+1} = g_i + p_i(g_{i-1} + p_{i-1}c_{i-1}) \tag{3.3}$$

Each stage in a ripple carry adder computes the generate, propagate, and the sum. The carryover bit is then rippled over to the subsequent stage. Figure 3.8 shows how each stage can be designed using simple AND, OR, and XOR gates. For 8-bit addition, eight of such stages are cascaded one after the other.

3.3.2 RNG

An 8-bit random number generator is required at the input of the comparator. A Galois LFSR circuit is designed for generating random numbers. Figure 3.5 can be extended to create a 8-bit RNG. The tap positions for a 8-bit RNG will be at 4,5,6, and 8.

3.3.3 Comparator

An 8-bit pipelined comparator is designed for comparison. Pipelining offers several advantages, one of which is reduced power consumption but this comes with overhead in

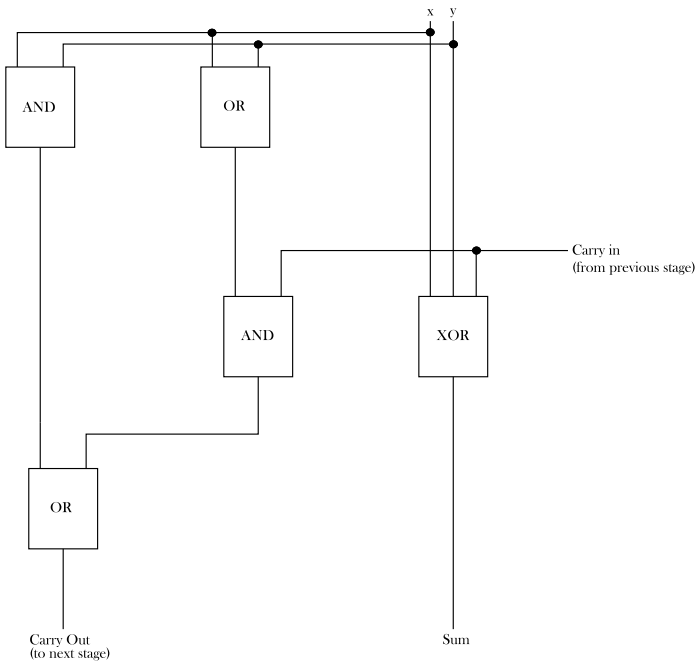

Fig. 3.8: One functional unit of a carry lookahead adder.

circuitry. It takes eight clock cycles to fill the pipleine and once this is done, results come out every clock cycle. Bit-by-bit comparison is made, starting from the most significant bit, the results being stored in intermediate flip flops. So, if the MSB of the first vector is greater than the MSB of the second vector, then all other bit-comparators are shut off and no further comparisons are made. Thus switching of comparators is made conditional which saves considerable power. Finally, the results of all the comparators are added to get the final result. This idea can be extended for a n-bit comparator. Figure 3.9 shows the idea of a pipelined comparator.

3.3.4 Regenerative Bit Detector and Multiplexer

Finally, a regenerative bit detector is used to check for the equality condition in the incoming bits and a 2:1 multiplexer to used to select between the regenerative bit and the random bit. Figure 3.6 shows the implementation of a regenerative bit detector.

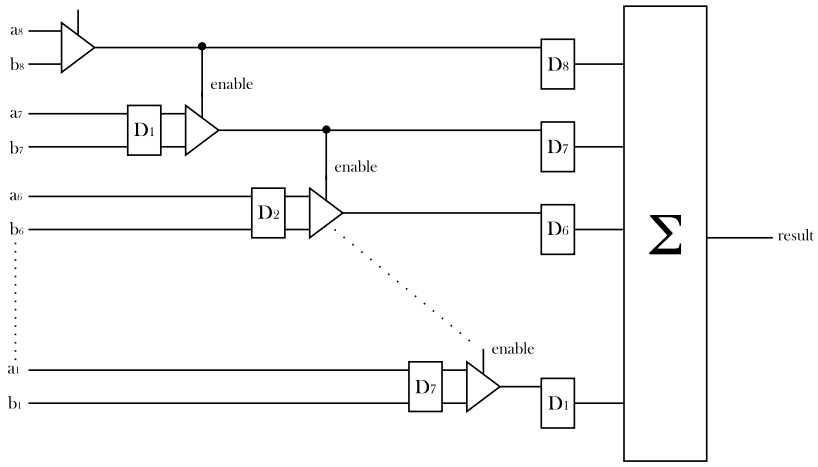

Fig. 3.9: Circuit schematic of a pipelined comparator.

3.4 Low Complexity Tracking Forecast Memories (TFM)

Tracking forecast memories have been recently introduced by Tehrani et al. [22, 23] in order to reduce the complexity inherent in edge memories. Edge memories have to placed on every edge of the decoder and with their high gate complexity these can take up most of the chip area for decoding a modern LDPC code. TFMs were introduced to address this problem and can replace EMs at every edge on the factor graph.

In the conventional sum-product algorithm, a variable nodes outgoing messages are replaced by a new probability mass determined according to equation (2.5). An alternative method, called successive relaxation [4,24] replaces the outgoing probability $\rho(t+1)$ at time $t+1$ according to equation (3.4), where $\rho(t)$ denotes previous probability, β is the relaxation factor lying in the range $0 < \beta < 1$, $r(t)$ is the regenerative bit from the variabe node with $r(t) \in [0, 1]$. Equation (2.5) describes the update rule when during a regular (regenerative) state and equation (3.5) describes the update rule during a hold state. During a hold state, a comparator is used to compare the probability $\rho(t)$ with a random number $R(t)$ and the output of the comparator is a 1 if $\rho(t) > R(t)$ and 0 otherwise.

$$\rho(t+1) = \rho(t) + \beta(r(t) - \rho(t)) \tag{3.4}$$

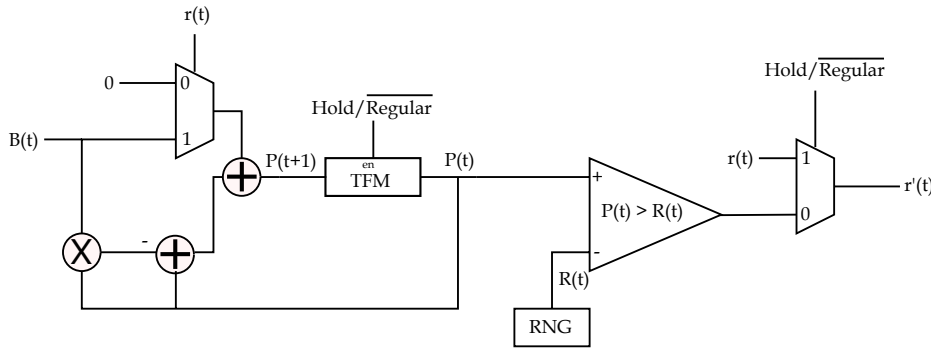

Fig. 3.10: General architecture of a tracking forecast memory following the successive relaxation rule.

$$r'(t) = \begin{cases} 1 & \rho(t) > R(t) \\ 0 & otherwise \end{cases} \tag{3.5}$$

A TFM approach directly converts equation (3.4) into a hardware realization as shown in fig. 3.10. When a *regular* state occurs in a equality node (which is detected by a regenerative bit detector), the TFM gets activated and updates the incoming probability based on equation (3.4). Since $r(t)$ can be either a 0 or a 1, equation (3.4) can take two values:

$$\rho(t+1) = \begin{cases} \rho(t) - \rho(t)\beta & if : r(t) = 0 \\ \rho(t) + \bar{\rho}(t)\beta & if : r(t) = 1, \end{cases} \tag{3.6}$$

where $1 - \rho(t)$ is replaced with $\bar{\rho}(t)$. Following from this equation, lesser complex hardware realization can be achieved since only a single adder will be required which could be used both as adder and a subtractor. Also, it is proposed by Tehrani et al. [22, 27], that if β is a negative factor of 2, then the multiplier in fig. 3.10 can be replaced with a shift register shifting input streams to the right. Thus, if the input stream has to be multplied with $\beta = 2^{-4}$, then the value stored in the shift register has to be shifted five times to the right which would achieve a multiplication of 1/16. A revised low complexity TFM architecture derived from above description is shown in fig. 3.11 [23].

It has been suggested by Tehrani [22] that an 8-bit TFM (relaxation coefficient $\beta = 2^{-4}$)

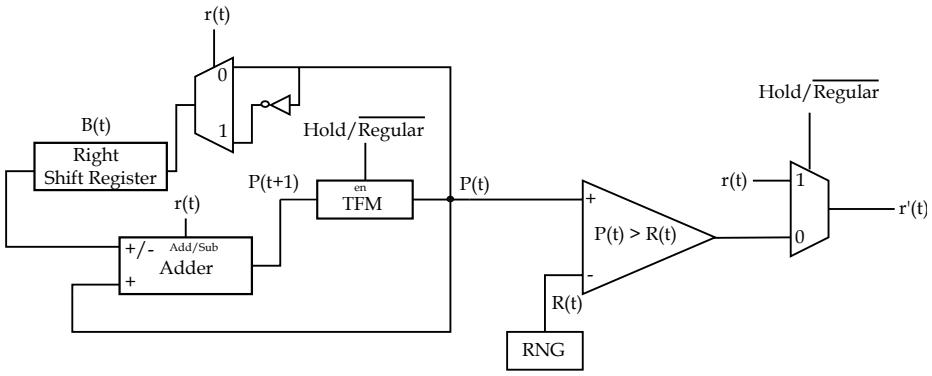

Fig. 3.11: Low complexity architecture of a tracking forecast memory with a consolidated adder/subtractor and a shift register for β multiplication.

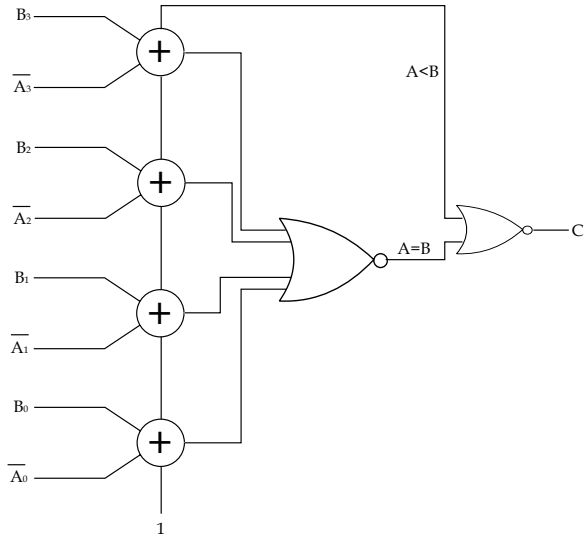

Fig. 3.12: Low complexity comparator circuit.

is sufficient for a practical LDPC decoder and results in a similar BER performance when compared with a 32-bit edge memory. An 8-bit TFM thus requires a 8-bit comparator, 8-bit adder/subtractor, 8-bit LFSR, 4-bit right shift register for achieving a $\beta = 2^{-4}$ and multiplexers. Multiplexers, LFSR, and shift register can be implemented in a similar way as shown in figs. 3.4 and 3.5. Besides, a low complexity comparator circuit and a adder/subtractor circuit are shown in figs. 3.12 and 3.13, respectively.

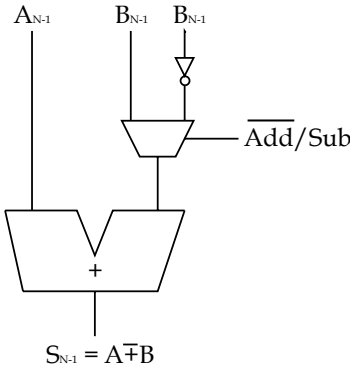

Fig. 3.13: Adder/subtractor unit.

3.5 Scaling of Channel LLRs

Scaling of channel LLRs is another way by which latching problem in stochastic decoders is minimized. Latching gets worse at high SNRs as switching activity of the decoder decreases. Scaling of channel LLRs is done in such a way that it the switching rate of the decoder. Noise dependent scaling is one method to scale the channel LLRs, in which the LLR samples are scaled by a factor proportional to the operating SNR [20]. However, scaling alone cannot solve the locking problem in stochastic decoders and has to be accompanied by a supernode structure which replaces the equality node with a modified structure which breaks the correlation between incoming stochastic streams. Therefore, scaling along with either edge memories or dual counters can overcome latching.

Chapter 4

Results

This chapter organizes results obtained from transistor level designs described in Chapter 3. A comparison between dual counter and edge memory is done on the basis of number of transistors required per edge for both these techniques. Besides, power consumption estimation is also done and based on these analyses several trade-offs are inferred between the two systems. The effect of scaling of process technologies is also studied and its effect on power consumption of the decoder is illustrated.

4.1 Complexity Comparison

Table 4.1 shows a comparison based on transistor counts per edge for a 32-bit edge memory, 7-bit pipelined dual counter, and an 8-bit tracking forecast memory.

4.1.1 Dual Counter and Tracking Forecast Memories

A comparative analysis shows that a 7-bit dual counter requires almost 65% more transistors per edge. A percent figure is also provided alongside each implementation which helps in seperating out which component is using the most transistors. For a dual counter design, the pipelined comparator consumes most transistors (46%) and increases the complexity substantially. However, a pipelined comparator is chosen for minimizing the power consumption, and hence a trade-off has to be made between power and complexity.

4.1.2 Edge Memory and Tracking Forecast Memories

Based on figures shown in Table 4.1, it can be inferred that a degree 3 equality node with an edge memory requires almost 14% more transistors per edge than an equivalent tracking forecast memory. This difference however is not as big as dual counters, but can

make a big difference when longer codes are being considered which have higher degree equality nodes.

Thus, it can be concluded that a tracking forecast memory beats both dual counter and edge memory design when complexity is being considered. However, it has been reported by Tehrani et al. [23] that delay of an 8-bit TFM is considerably higher than an equivalent edge memory.

4.2 Power Consumption and Energy

Power consumption in CMOS circuits can be broadly divided into static power consumption (due to reverse-biased leakage current between the diffused region and the substrate), dynamic power consumption (due to charging and discharging of internal nodes of the circuit), and short circuit power consumption (due to finite rise and fall times of input pulses causing some short-circuit current from V_{dd} to go through to the ground). High frequency switching makes dynamic power consumption a winner in the overall power consumption. Dynamic power consumption in CMOS circuits is given by equation (4.1), where α is the activity factor or the transistion probability, f_c is operating frequency, C is output node capacitance (also called as power dissipation capacitance), and V_{dd} is the supply voltage.

$$P_d = \alpha f_c C V_{dd}^2 \tag{4.1}$$

Table 4.1: Comparison of transistor count per edge between a 32-bit edge memory, 7-bit pipelined dual counter, and a 8-bit tracking forecast memory ($\beta = 2^{-4}$).

Component	**32-bit EM**		**7-bit DC**		**8-bit TFM**	
	Count	%	Count	%	Count	%
Adder			320	18%	208	34%
Comparators			804	46%	180	29%
RNG	78	11%	78	4%	78	12%
Counters			522	30%		
Registers	128	18%			132	21%
Multiplexers	186	26%			12	2%
Initializer	312	44%				
Total Count	**704**		**1724**		**610**	

Notice the direct dependence of power consumption on operating frequency. Operating frequency of the decoder can vary with varying decoder speeds, hence power consumption of decoder will also vary. In order to make the results independent of operating frequency, energy consumption is evaluated for the decoder instead of its power consumption. Thus, equation (4.1) can be modified as following:

$$E = \alpha C V_{dd}^2 = \alpha \int p(t)dt = \alpha V_{dd} \int I(t)dt. \tag{4.2}$$

In order to calculate the energy consumption given by equation (4.2), there are two variables which need to be evaluated. First is the activity factor or α, and second is the integral $\int I(t)dt$.

4.2.1 Activity Factor of a Stochastic Decoder

Using system level C simulations, the activity factor or the switching frequency of several nodes in a stochastic decoder using dual counters are probed against increasing SNRs. Using these simulations it is deduced that activity factor of the decoder goes down with increasing values of SNR, which would infer to proportional decrease in power consumption. This is shown in fig. 4.1. Besides, the C snippet shown in fig. 4.14 illustrates how transitions for a flip flop memory are computed against increasing SNR values in the system level C simulations.

4.2.2 Calculating $\int I(t)dt$

All components designed can be generalized using a pull down and pull up network as shown in fig. 4.2. Everytime there is a switch between the pull up and pull down network, some current is drawn from the supply rails as shown in figs. 4.3 and 4.4. This current contributes to the dynamic power consumption in CMOS circuits. Total current drawn from the power supply can be found out by integrating the current over time.

Current plots generated (using spectre) of all components of the decoder are exported to matlab and are then integrated over time to get the total dynamic current consumption.

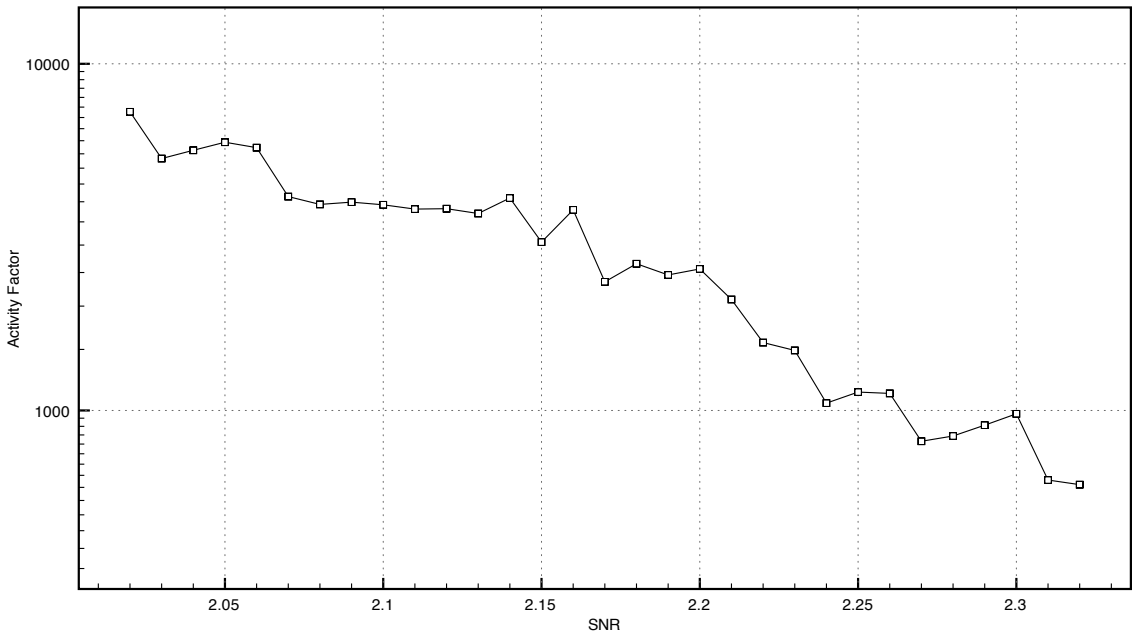

Fig. 4.1: Activity factor (switching rate) per frame vs. SNR for a (2000,1000) LDPC stochastic decoder using dual counter hysteresis.

4.3 Energy: Dual Counter, Edge Memory, and Tracking Forecast Memory

Using the above mentioned analysis techniques, energy requirements for all components are calculated and tabulated in Table 4.2. The numbers show average energy consumption per transition, i.e how much energy is required when a gate output swtiches from a 0 state to a 1 state or vice versa. Power consumed during transistions $0 \rightarrow 1$ and $1 \rightarrow 0$ are different. Results shown in Table 4.2 denote average energy consumption of the two. For circuits with multiple outputs (like the 8-bit adder used in dual-counter) it becomes difficult to track which output is transitioning when. Therefore for such circuits, worst case power consumption is calculated, which can be estimated when all outputs are transitioning from a 0 state to a 1 state or vice versa.

Table 4.2 depicts a comparison of energy consumption of three types of hysteretic filters. An 8-bit TFM outperfoms both dual counter and edge memory for a degree 3 equality node when energy consumption per transition is considered. TFM achieves a factor of 9 (89%) improvement in energy over an equivalent edge memory and by a factor of 3 (68%) over a

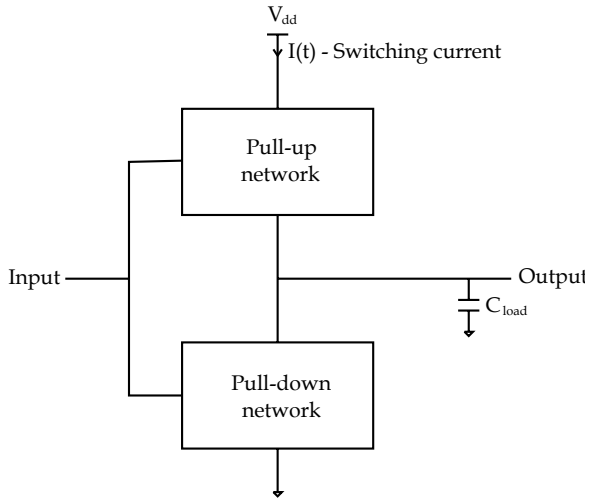

Fig. 4.2: Generalized CMOS network showing pull-up and pull-down networks.

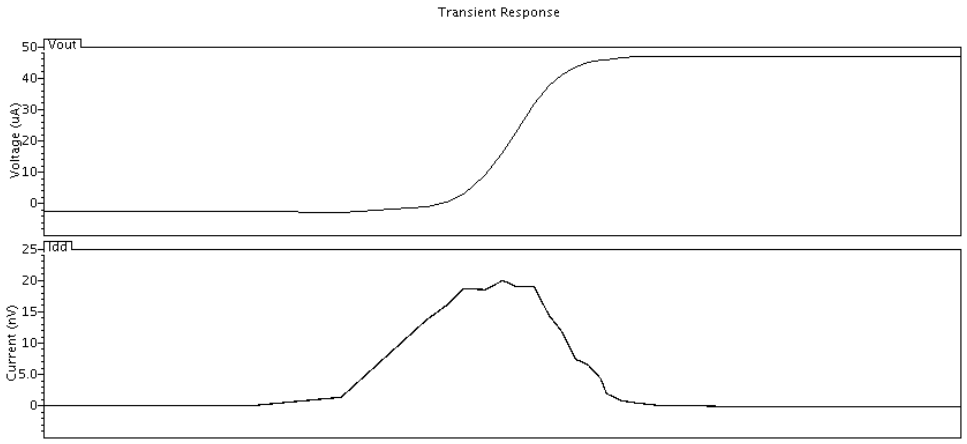

Fig. 4.3: Current spike when an output of a CMOS gate switches from a 0 to 1.

dual counter scheme. Note that these figures are projected for a TFM with $\beta = 2^{-4}$ and will vary if β varies. Higher β will increase both complexity and energy consumption.

4.4 Energy, SNR, and Scaling Process Technologies

Power consumption is a strong function of amount of switching rate inherent in the decoder. More switching means more power consumption. Switching, on the other hand, decreases at high SNR. Figure 4.5 shows how energy consumed per operation decreases with increase in SNR for a typical (2000,1000) LDPC decoder. This implies that at higher SNR there will be proportional saving in the total power. Thus, power consumption is a function

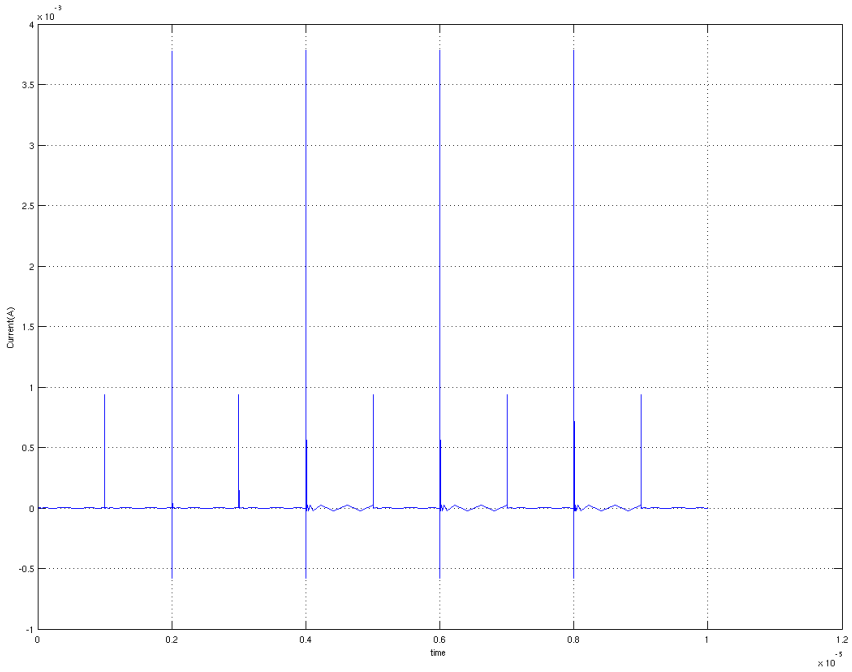

Fig. 4.4: Current spike when an output of a CMOS gate switches from a 0 to 1.

of switching, which in turn is a function of signal-to-noise ratio.

Besides, fig. 4.5 also shows how energy consumption for a 7-bit dual counter hysteretic filter goes down with scaling feature sizes or channel lengths. Four feature sizes are compared (0.6μm, 0.35μm, 0.25μm, and 0.18μm) and results show that energy consumption goes down with newer process technologies. This is a result of proportional decrease in supply voltage which is a major contributor to power consumption in CMOS circuits as shown in equation (4.2). Therefore, scaling to a new technology could also lead to savings in power. It has already been mentioned previously that stochastic iterative decoders scale easily and migration to a newer technology is relatively straightforward. Table 4.3 shows a numerical comparison of a dual counter with scaling technologies.

Y-axis in fig. 4.5 represents energy consumption per bit, which is calculated as follows:

$$E_b = \frac{Energy/transition \times T}{L}, \tag{4.3}$$

Table 4.2: Comparison of energy consumed per transition for a degree 3 ($d_v = 3$) 7-bit dual counter, 32-bit edge memory, and 8-bit tracking forecast memory.

Component	**Edge Memory**	**Dual Counter**	**TFM**
Comparator		2.65 pJ	3.08 pJ
LFSR	5.27 pJ	3.06 pJ	3.06 pJ
Adder		0.47 pJ	0.47 pJ
Up/Down Counter		32.68 pJ	
Shift Registers	96.5 pJ		5.51 pJ
Multiplexers	10.86 pJ		0.288 pJ
Total Energy Consumed	**112.63 pJ**	**38.86 pJ**	**12.40 pJ**

Table 4.3: Numerical comparison between energy consumption for different feature sizes for a 7-bit dual counter hysteretic filter.

Component	**0.6μm**	**0.35μm**	**0.25μm**	**0.18μm**
Comparator	11.04 pJ	7.02 pJ	6.34 pJ	2.65 pJ
LFSR(RNG)	54.49 pJ	23.30 pJ	18.18 pJ	3.06 pJ
Adder	14.15 pJ	7.47 pJ	4.52 pJ	0.47 pJ
Counters	190.35 pJ	102.72 pJ	75.72 pJ	32.68 pJ
Equality	0.69 pJ	0.16 pJ	0.13 pJ	0.003 pJ
Total	270.72 pJ	140.67pJ	104.89 pJ	38.863 pJ

where E_b represents Energy/bit, T is number of transitions, and L represents codelength. This expression can be treated as normalized energy consumption, and therefore can be used to compare energy consumption for different codelengths.

4.5 Speed and Convergence

Figure 4.6 shows BER performance for a (3,6) LDPC (2000,1000) code using 8-bit dual counter hysteresis, 32-bit shift register, and a 8-bit tracking forecast memory ($\beta = 0.0625$). This figure shows that TFM converges much faster than dual counters and edge memories. Initially, at low SNRs, the performance is almost equivalent, but as SNR increases, TFM shows a better BER performance than an equivalent dual counter hysteresis or an edge memory.

Figures 4.7, 4.8, and 4.9 show a plot of BER vs. decoding cycles for fixed SNR values of 2.5dB, 3.0dB, and 3.5dB. Based on these figures, it is clear that for a particular BER, a TFM takes lesser number of decoding cycles than a dual counter. In other words, TFM

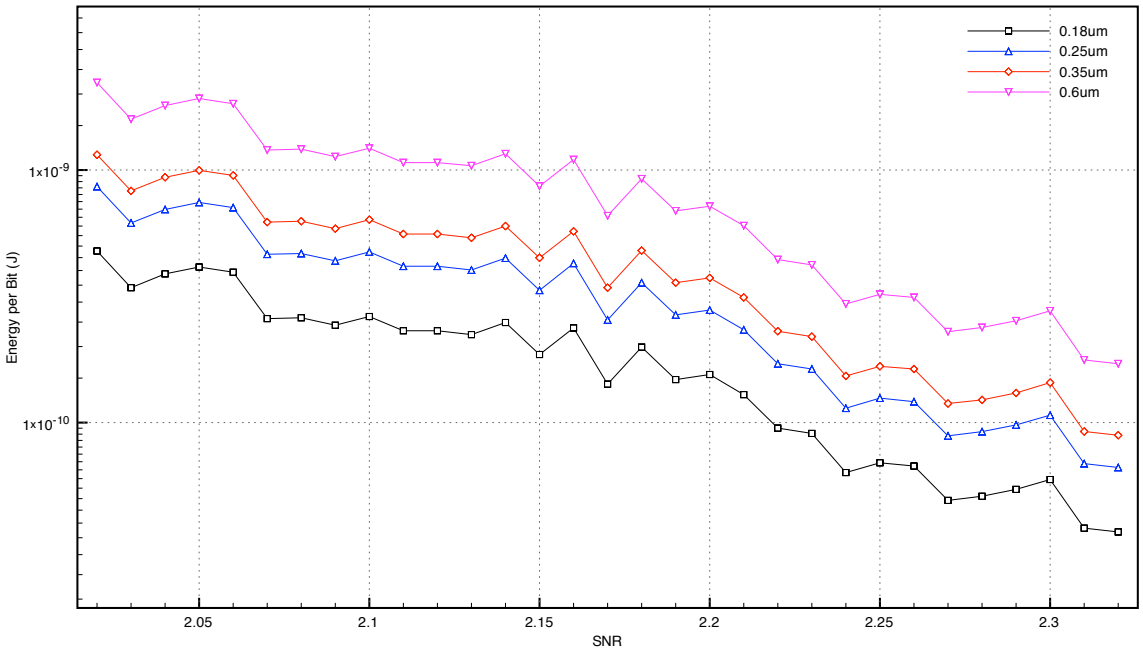

Fig. 4.5: Energy vs. SNR for a (2000,1000) LDPC stochastic decoder. This plot also shows how energy consumption varies with decreasing feature sizes.

outperforms dual counters in speed as well.

4.6 BER Performance vs. Hysteretic Edges

Figure 4.10 shows a performance plot for BER vs. increasing hysteretic edges (H) for a fixed SNR of 2dB. A (3,6) code contains a degree 3 equality node and a degree 6 parity check node, and if H=4, then for a degree 6 parity check node, there are four edges which employ hysteresis and the other two edges follow the regular flip flop rule. From fig. 4.10 two deductions can be made, firstly as more and more hysteresis is introduced, the BER drops down for each case, i.e TFM, EM, and dual counters. Secondly, for a given number of hysteretic edges, TFM has a much better BER performance than the other two in the lot.

4.7 BER Performance and Hysteresis

Figures 4.11, 4.12, and 4.13 show a plot of BER vs. SNR as number of hysteretic edges

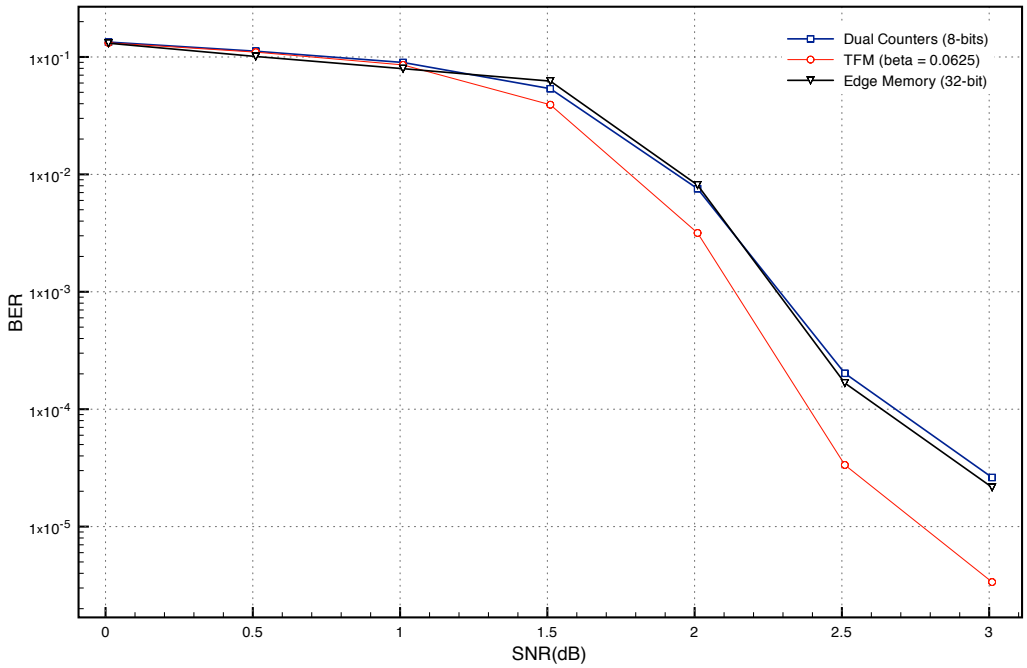

Fig. 4.6: BER performance comparison for an 8-bit dual counter, 32-bit edge memory, and a tracking forecast memory with $\beta = 0.0625$ for a (3,6) LDPC (2000,1000) code.

are increased for a dual counter hysteresis, EM, and TFM. In general, as more and more hysteresis is introduced the convergence gets better and better. For dual counters and TFM, there is a negligible loss in performance for H=5 and H=6. Therefore, performance loss can be traded off with lesser number of transistors per edge. For a long code, this can save up a lot of area, and hence cost. For edge memories, at higher SNRs, a reverse effect starts to happen. As more and more hysteresis is introduced into the graph, the SNR performance actually gets worse especially for cases H=4, H=5, H=6. A parity check node with four hysteretic edges following the edge memory rule actually performs better than a node with six hysteretic edges. Therefore, it is not necessary to for every edge to be a hysteretic edge for a better performance when an edge memory hysteretic filter is used.

4.8 Design Parameters

Table 4.4 shows the design parameters for all four processes chosen in this design [28]. Here, V_{dd} refers to supply voltage, L refers to channel length used, W_p and W_n refer to width

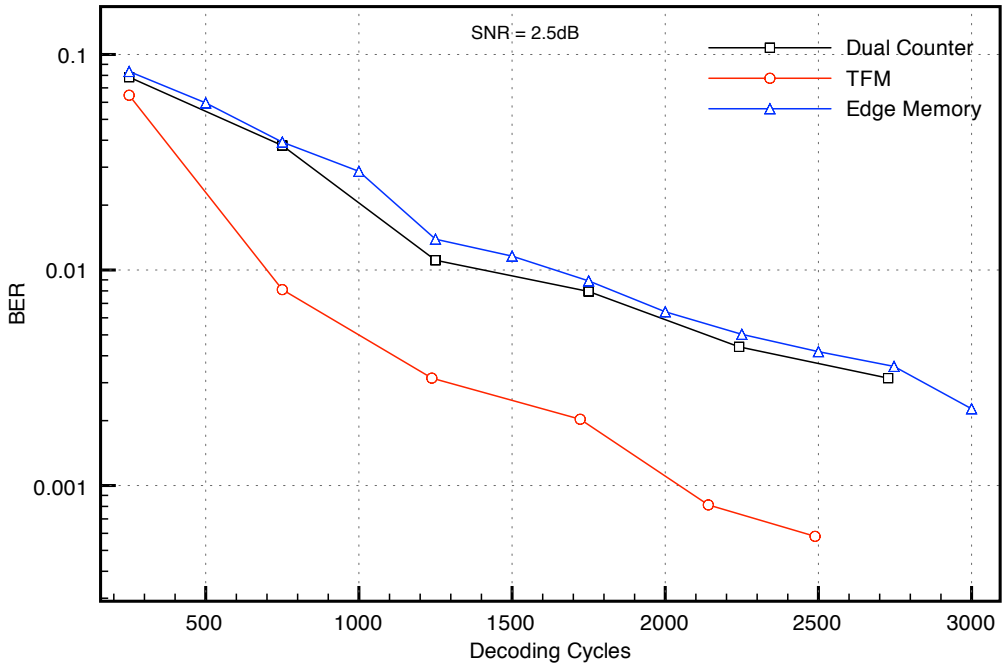

Fig. 4.7: BER vs. decoding cycles at $SNR = 2.5dB$ for an 8-bit dual counter, 32-bit edge memory, and a 8-bit TFM with $\beta = 0.0625$ for a (3,6) LDPC (2000,1000) code.

of PMOS and NMOS devices in the designs. W_p and W_n are chosen so that the switching threshold voltage lies at $V_{dd}/2$ which causes noise margin low to be equal to noise margin high ($NM_L = NM_H$). Also, since mobility of holes (μ_p) is lesser than mobility of electrons (μ_n), PMOS transistors have to be made wider to compensate for lesser mobility. W_p and W_n are chosen so that the switching threshold voltage lies at $V_{dd}/2$ which causes noise margin low to be equal to noise margin high ($NM_L = NM_H$). Process models are chosen from two vendors: Taiwan manufacturing semiconductor company (TSMC) and AMI.

Table 4.4: Parameters for 0.6μm, 0.35μm, 0.25μm, 0.18μm technologies.

Parameters	**0.6μm**	**0.35μm**	**0.25μm**	**0.18μm**
Vendor	AMI	TSMC	TSMC	TSMC
V_{dd}	3.3 V	3.3 V	2.5 V	1.8 V
L	0.6μm	0.35μm	0.25μm	0.18μm
W_p	2.4μm	2μm	1.8μm	1.6μm
W_n	1.5μm	0.3μm	0.36μm	0.6μm

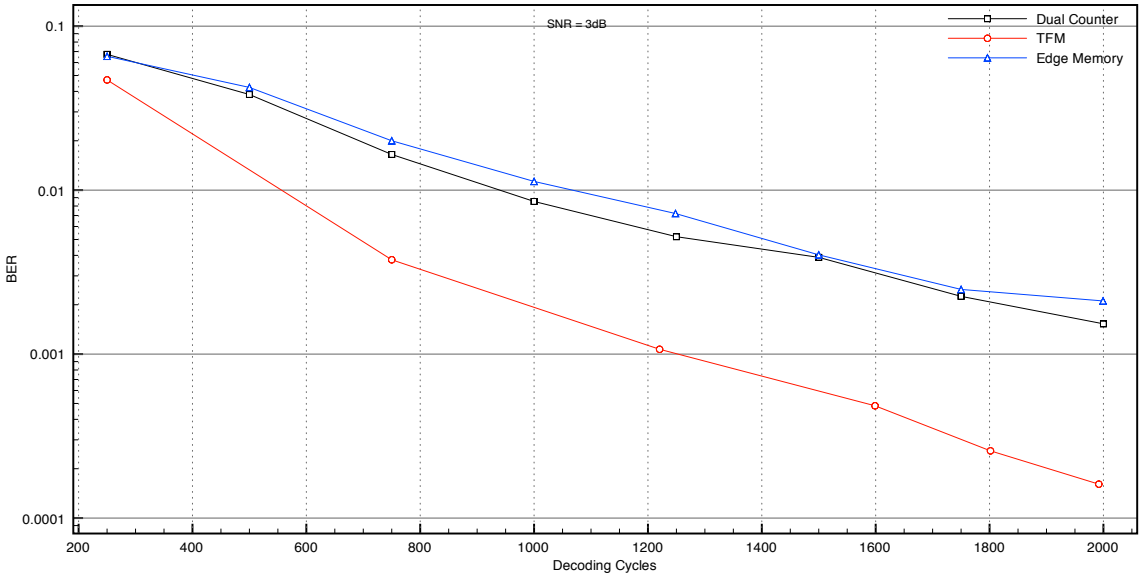

Fig. 4.8: BER vs. decoding cycles at $SNR = 3dB$ for an 8-bit dual counter, 32-bit edge memory, and a 8-bit TFM with $\beta = 0.0625$ for a (3,6) LDPC (2000,1000) code.

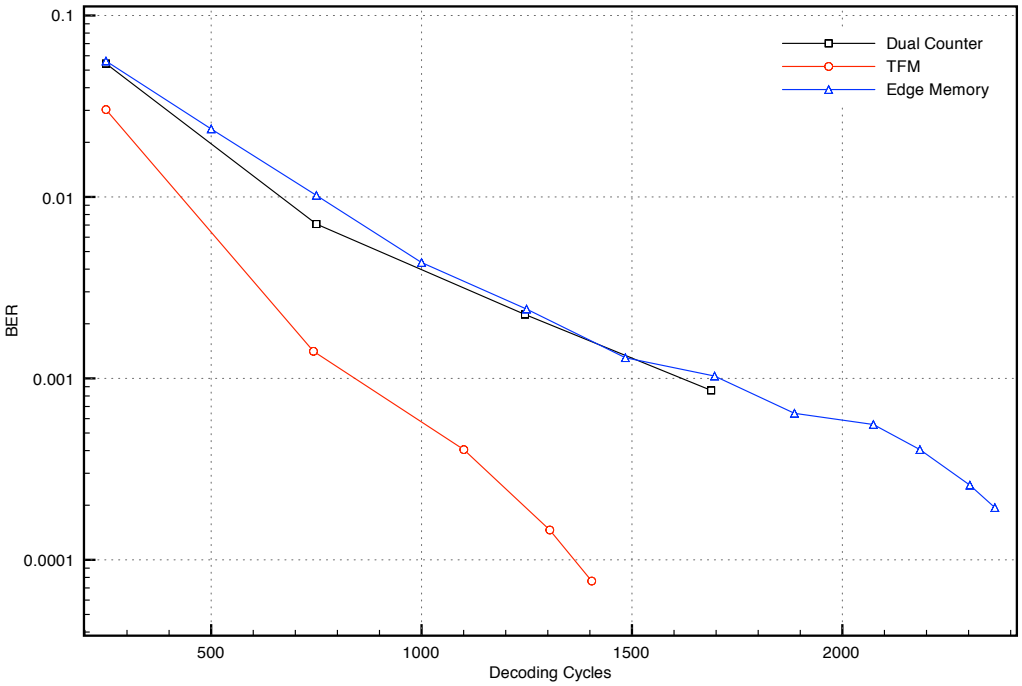

Fig. 4.9: BER vs. decoding cycles at $SNR = 3.5dB$ for an 8-bit dual counter, 32-bit edge memory, and a 8-bit TFM with $\beta = 0.0625$ for a (3,6) LDPC (2000,1000) code.

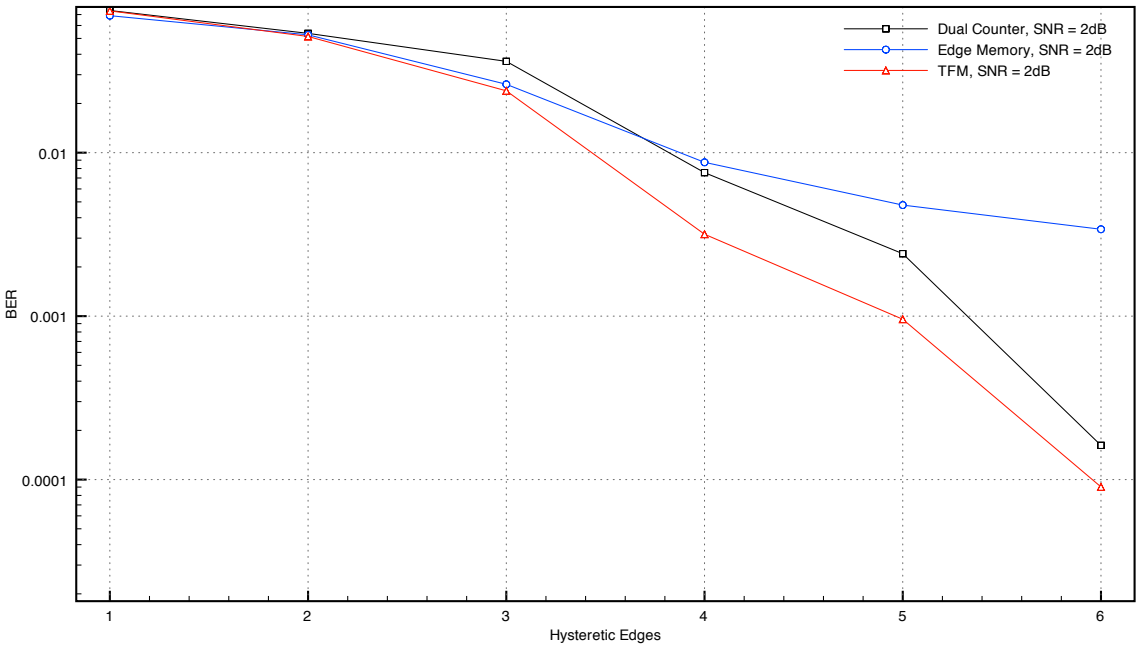

Fig. 4.10: BER vs. number of hysteretic edges at $SNR = 2dB$ for an 8-bit dual counter, 32-bit edge memory, and a 8-bit TFM with $\beta = 0.0625$ for a (3,6) LDPC (2000,1000) code.

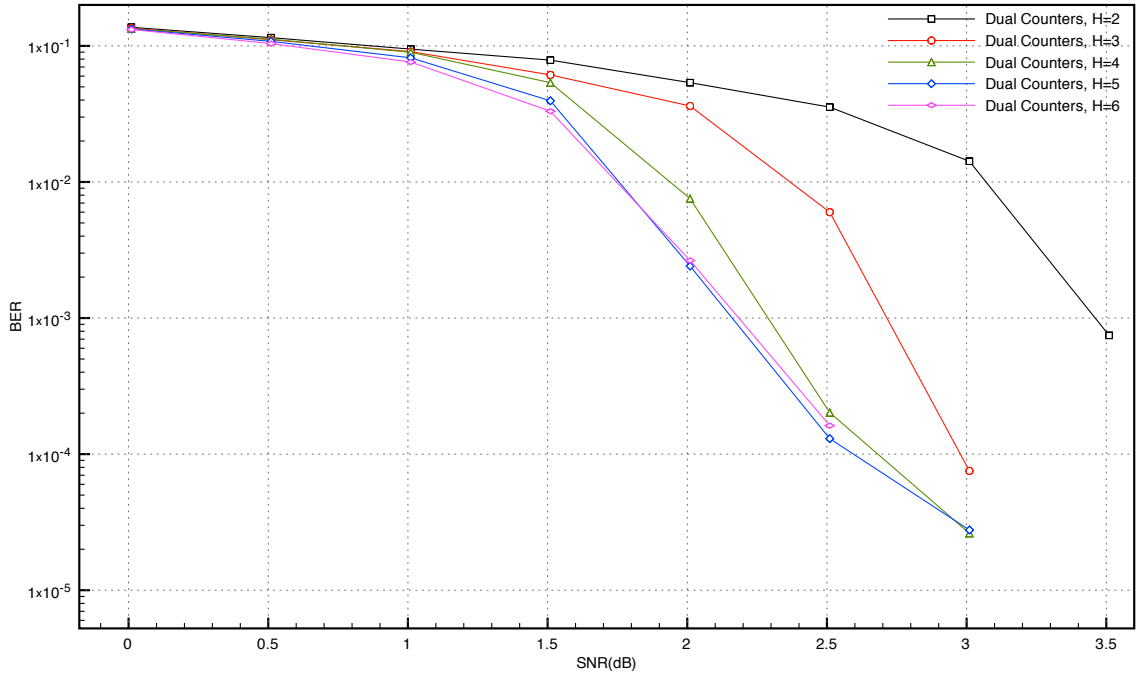

Fig. 4.11: BER vs. SNR with increasing hysteretic edges (H) for an 8-bit dual counter, 32-bit edge memory, and a 8-bit TFM with $\beta = 0.0625$ for a (3,6) LDPC (2000,1000) code.

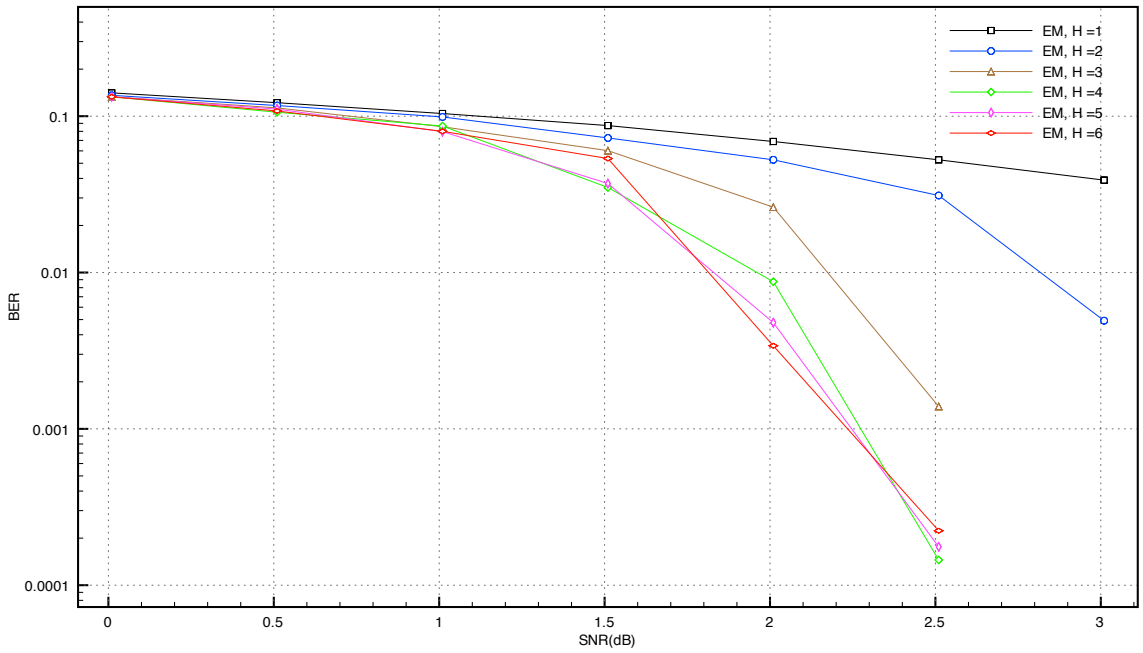

Fig. 4.12: BER vs. SNR with increasing hysteretic edges (H) for an 8-bit dual counter, 32-bit edge memory, and a 8-bit TFM with $\beta = 0.0625$ for a (3,6) LDPC (2000,1000) code.

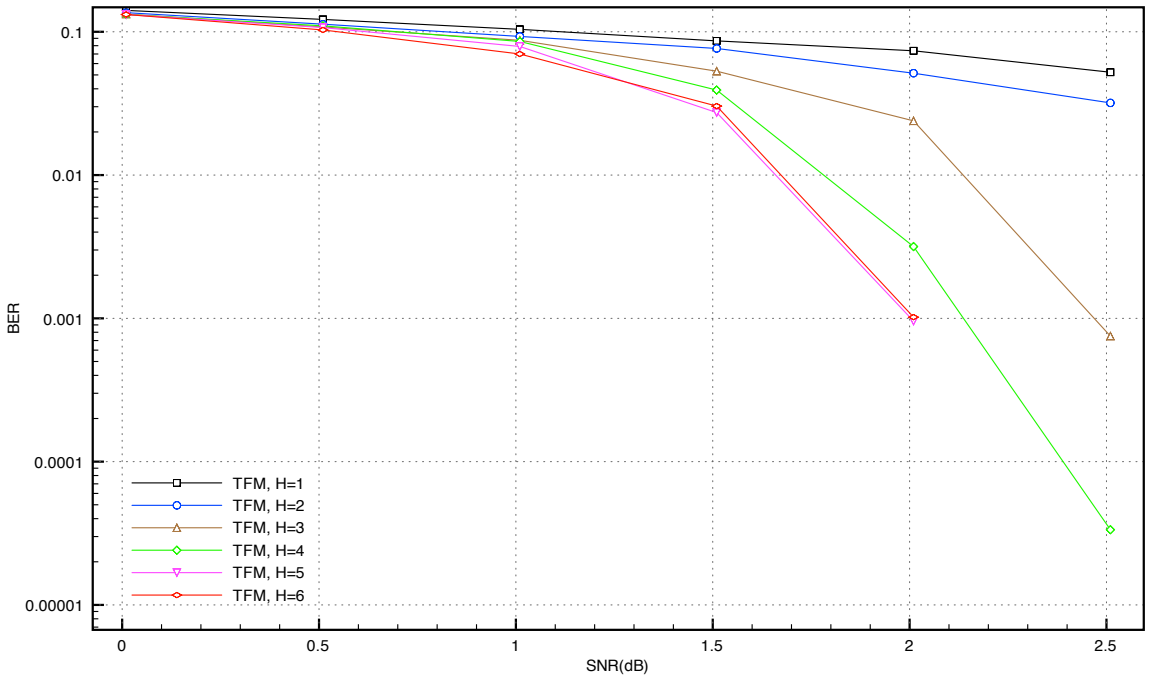

Fig. 4.13: BER vs. SNR with increasing hysteretic edges (H) for an 8-bit dual counter, 32-bit edge memory, and a 8-bit TFM with $\beta = 0.0625$ for a (3,6) LDPC (2000,1000) code.

```
//File - main.c
int main(int argc, char * argv[])
{
 for(int i=min_SNR;i<max_SNR;i++)
 {
    bitgen();
    encode();
    cwcheck();
    transmit();
    initializeDecoder();
    decode();
    makeDecisions();
    .
    .
    .
    .
    .
 }
}
//end of main.c

//File - stochastic_dual.c
//Definitions
int ***TransitionMatrix; //This matrix stores states for each iteration, eqnode, and eqedge
long int transitions=0; //Stores the total transitions at the end of each SNR value

void decode()
{
 .
 .
 .
 for (it=0; it<iterations; it++)
 {
  doOneIteration(it);
  .
  .
  .
 }
 Transitions = ComputeTransitions(TrasitionMatrix);
}

void doOneIteration(int iteration)
{
 .
 .
 .
 for(int n = 0; n<_alist.N; n++)
 {
  for (int e=0;e<_alist.biggest_num_n;e++)
  {
   //Store each flip flop output state into the transition matrix
   TransitionMatrix[iteration][n][e]= ComputeFlipFlopbit(n,e);
  }
 }
}


//Computes total transitions occuring in the output states which are stored in the Transition Matrix
long int ComputeTransitions(int ***TransitionMatrix)
{
 long int transitions = 0;     //Set to zero each SNR value.
 for(int i=0; i<iterations; i++)
 {
  for(int n = 0; n<_alist.N; n++)
  {
   for (int e=0; e<_alist.biggest_num_n; e++)
   {
    //Reached the end of transition matrix - do nothing (since cannot return a value inside a for loop)
    if(i == (iterations-1))
     transitions+=0;

    /*if there is a change between current state and next state between iterations then increment
    transitions*/
    else if(TransitionMatrix[i][n][e] != TransitionMatrix[i+1][n][e])
     transitions+=1;
   }
  }
 }
 return transitions;
}
//end stochastic_dual.c
```

Fig. 4.14: C snippet for calculating output transitions for a flip flop memory in a stochastic iterative decoder.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

This thesis presented a simple implementation of three variants of a stochastic iterative decoder. This was then used to project the transistor count and average energy consumption. It is evident that hardware required for implementation of such decoders is unsophisticated consisting of simple logic gates. Three hysteresis techniques are considered to prevent locking, and their transistor counts, energy consumption, speed, and convergence are recorded. From data analysis, it is inferred that tracking forecast memories are much efficient as far as complexity, power, speed, and convergence is concerned than an equivalent dual counter and edge memory.

When dual counters and edge memories are compared, it can be concluded that dual counter hysteresis requires almost 1000 more transistors than edge-memories (per hysteretic edge). When energy consumed per transition is considered, edge memories are more power hungry than dual counters. Dual counter, on the other hand, needs more transistors to implement, but requires less power. Another advantage with dual counters is that they are not required on every edge of the factor graph. Careful placement of dual counters on specific edges will not only prevent locking, but would also save plenty of transistors per edge. This is not the case with edge memories and tracking forecast memories, as they have to be placed on every edge in the factor graph.

Energy consumption is directly dependent on the decoder speed, so there is always a trade off between the two. Also, the level of switching decreases at high SNRs, which suggests a proportional decrease in the power consumption. Power consumption also goes down with scaling processes, and this could be easily done in stochastic iterative decoders as they scale very easily to a new process. Smart use of components, for example, reusing

the same component for multiple functions, can further reduce the transistor count and power consumption. Table 5.1 provides a summary of the results derived in this thesis.

5.2 Future Work

Designs presented in this thesis can be used to build an actual decoder and the results can be verified with those obtained from system level simulations. Computational units required for designing an actual decoder have been implemented in this thesis and these units can be used to build a practical decoder. This can be done using SKILL scripting which can place these computational units according to code's factor graph. This would eliminate all the manual hardwork required to place all decoding units in order. SKILL scripting could be further used for easy layout of the decoder, which can then be sent for fabrication.

Besides, the designs presented can also be optimized more with respect to propagation delay. If this is done, then decoding speeds can go up considerably. As discussed earlier, the ripple carry adder can be replaced with a carry look ahead adder which has lesses propagation delay. Also, circuits like multiplexers, adders, AND, OR gates could be analysed more to reduce branch and path logical effort which would result in reduced propagation delay.

Component sharing can also lead to savings in transitors (per edge) and static power consumption. Complex components such as comparators can be shared between several nodes and thus will help in reducing the gate complexity.

Table 5.1: Complexity and energy comparison between dual counter, edge memory, and tracking forecast memory.

Hysteretsis Type	**Resolution**	**Transistor count**	**Energy/transition**
Dual Counters	7 bits	1724	38.86 pJ
Edge Memory	32 bits	704	112.63 pJ
Tracking Forecast Memory	8 bits	618	12.40 pJ

References

[1] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo codes," in *IEEE International Conference on Communications*, Geneva, May 1993.

[2] R. Gallager, "Low-Density Parity-Check Codes," *IRE Transactions on Information Theory*, vol. IT-8, pp. 21–28, Jan. 1962.

[3] V. Gaudet and A. Rapley, "Iterative decoding using stochastic computation," *Electronic Letters*, vol. 39, Feb. 2003.

[4] C. Winstead, A. Rapley, V. Gaudet, and C. Schlegel, "Stochastic Iterative Decoders," in *IEEE Symposium on Information Theory (ISIT)*, Adelaide, Sept. 2005.

[5] V. Gaudet, A. Rapley, C. Winstead, and C. Schlegel, "Stochastic iterative decoding on factor graphs," in *Analog Decoder Workshop*, Zurich, Switzerland, Sept. 2003.

[6] S. Tehrani, S. Mannor, and W. Gross, "An area-efficient FPGA-based architecture for fully-parallel stochastic LDPC decoding," in *2007 IEEE Workshop on Signal Processing Systems*, vol. 17-19, pp. 255–260, 2007.

[7] C. Winstead and H. Gao, "A stochastic decoder for a (256,121) block turbo code," 2008 [Online]. Available: `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.109.7316`.

[8] A. Blanksby and C. Howland, "A 690-mw 1-Gb/s 1024-b, rate-1/2 Low-Density Parity-Check Code Decoder," *IEEE Journal of Solid-State Circuits*, vol. 37, Mar. 2002.

[9] A. Blanksby and C. Howland, "Parallel decoding architectures for low density parity check codes," in *IEEE International Symposium on Circuits and Systems*, Sydney, Australia, May 2001.

[10] S. Hong and W. Stark, "Design and implementation of a low complexity VLSI turbo-code decoder architecture for low energy mobile wireless communications," *Journal of VLSI Signal Processing Systems*, vol. 24, pp. 43–57, 2000.

[11] G. Lechner, J. Sayir, and M. Rupp, "Efficient DSP implementation of an LDPC decoder," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Montreal, Canada, May 2004.

[12] H. Loeliger, F. Lustenberger, M. Helfenstein, and F. Tarkoy, "Probability propagation and decoding in analog VLSI," *IEEE Transcations on Information Theory*, vol. 47, no. 2, pp. 837–843, Feb. 2001.

[13] H. Loeliger, F. Tarkoy, E. Tech, A. Lustenberger, and M. Helfenstein, "Decoding in analog VLSI," *IEEE Communications Magazine*, Apr. 1999.

[14] C. Winstead, *Analog Iterative Error Control Decoders*. Ph.D. dissertation, University of Alberta, 2005.

[15] J. Dai, *Design Methodology for Analog VLSI Implementations of Error Control Decoders*. Ph.D. dissertation, The University of Utah, 2002.

[16] C. J. Winstead and C. Schlegel, "Low-voltage CMOS circuits for analog decoders," US Patent US 2007/0 276 895 A9, 2007.

[17] C. Winstead, J. Dai, S. Yu, R. Harrison, C. Myers, and C. Schlegel, "Analog decoding of product codes," in *International Symposium on Information Technology*, 2002.

[18] C. Winstead, J. Dai, W.J. Kim, S. Little, Y.B. Kim, C. Myers, and C. Schlegel, "Analog MAP decoder for (8,4) hamming code in subthreshold CMOS," in *Conference on Advanced Research in VLSI*, 2001.

[19] C. Winstead and C. Schlegel, "Analog decoding: the state of the art," in *International Symposium on Spread Spectrum Techniques and Applications*, Sydney, Sept. 2004.

[20] W. Gross, S. Tehrani, and S. Mannor, "Stochastic decoding of LDPC codes," *IEEE Communication Letters*, vol. 10, pp. 716–718, Oct. 2006.

[21] A. Rapley, V. Gaudet, and C. Winstead, "On the simulation of stochastic iterative decoder architectures," in *Canadian Conference on Electrical and Computer Engineering*, 2005.

[22] S. Tehrani, A. Naderi, G. Kamendje, S. Mannor, and W. Gross, "Tracking forecast memories in stochastic decoders," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 561–564, 2009.

[23] S. Tehrani, A. Naderi, G. Kamendje, S. Mannor, and W. Gross, "Tracking forecast memories in stochastic decoders," *Journal of Signal Processing Systems*, 2010.

[24] S. Tehrani, C. Winstead, W. Gross, S. Mannor, S. Howard, and V. Gaudet, "Relaxation Dynamics in Stochastic Iterative Decoders," *IEEE Transactions on Signal Processing*, vol. 58, pp. 5955–5961, 2010.

[25] C. Winstead, "Error-control decoders and probabilistic computation," in *Tohoku University 3rd SOIM-COE Conference*, Sendai, Japan, Oct. 2005.

[26] I. Goldberg and D. Wagner. (1996) Architectural considerations for cryptanalytic hardware. Technical Report. University of California, Berkeley.

[27] S. Tehrani, A. Naderi, G. Kamendje, S. Hemati, S. Mannor, and W. Gross, "Majority-based tracking forecast memories for stochastic LDPC decoding," *IEEE Transactions on Signal Processing*, vol. 58, pp. 4883–4896, Sept. 2010.

[28] MOSIS [Online]. Available: `http://www.mosis.com/products/fab/vendors/tsmc/tsmc_rules_libs.html`.