Project Description

SHF Medium: Digital Yet Deliberately Random –

Synthesizing Logical Computation on Stochastic Bit Streams

1 Introduction

Humans are accustomed to counting in a positional number system – decimal radix. Nearly all computer systems operate on another positional number system – binary radix. From the standpoint of *representation*, such positional systems are compact: given a radix b, one can represent b^n distinct numbers with n digits. Each choice of the digits $d_i \in \{0, \ldots, b-1\}$, $i = 0, \ldots, n-1$, results in a different number N in $[0, \ldots, b^n - 1]$:

$$N = \sum_{i=0}^{n-1} b^i d_i.$$

However, from the standpoint of *computation*, positional systems impose a burden: for each operation such as addition or multiplication, the signal must be "decoded," with each digit weighted according to its position. The result must be "re-encoded" back in positional form. Any student who has designed a binary multiplier in a course on logic design can appreciate all the complexity that goes into wiring up such an operation.

Consider instead digital computation that is based on a *stochastic* representation of data: each real-valued number x ($0 \leq x \leq 1$) is represented by a sequence of random bits, each of which has probability x of being one and probability $1 - x$ of being zero. These bits can either be serial streaming on a single wire or in parallel on a bundle of wires. When serially streaming, the signals are probabilistic in *time*, as illustrated in Figure 1(a); when in parallel, they are probabilistic in *space*, as illustrated in Figure 1(b). Throughout this proposal, we frame the discussion in terms of serial bit streams. However, our approach is equally applicable to parallel wire bundles. Indeed, we have advocated this sort of stochastic representation for technologies such as nanowire crossbar arrays [27].

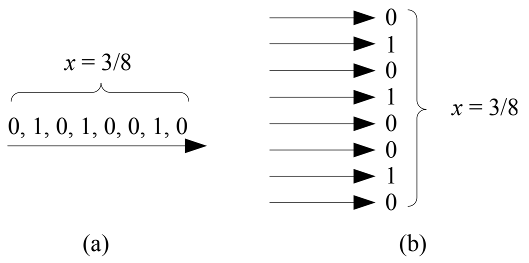
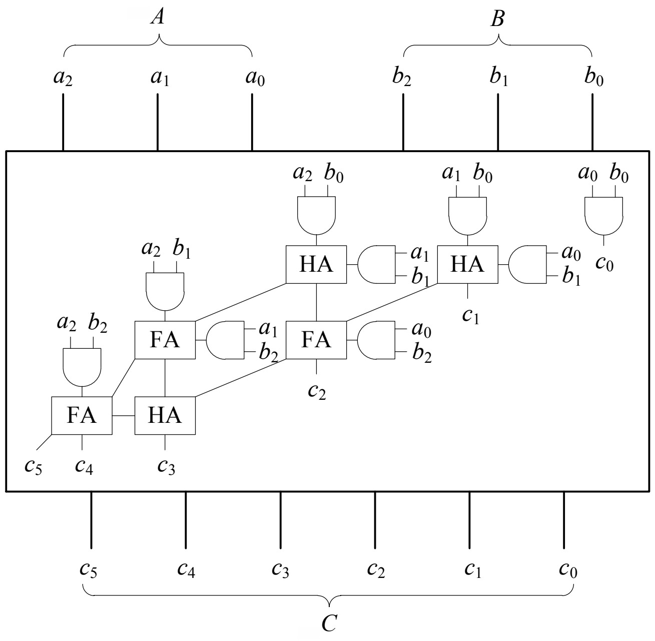


Figure 1: Stochastic representation: (a) A stochastic bit stream; (b) A stochastic wire bundle. A real value x in the unit interval $[0, 1]$ is represented as a bit stream or a bundle. For each bit in the bit stream or the bundle, the probability that it is one is x.

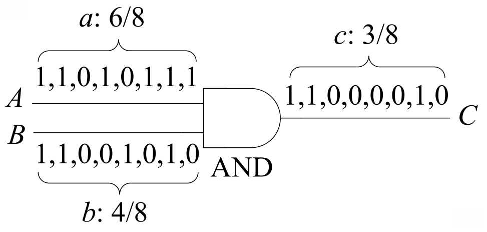
Consider the operation of multiplication implemented conventionally versus stochastically. Figure 2(a) shows a conventional design for a 3-bit carry-save multiplier, operating on binary radix-encoded numbers. It consists of 9 AND gates, 3 half adders and 3 full adders, for a total of perhaps 30 gates. Figure 2(b) shows a stochastic multiplier: it consists of but a *single* AND gate. The inputs are two independent input stochastic bit streams A and B. The number represented by the output stochastic bit stream C is

$$\begin{aligned} c &= P(C = 1) = P(A = 1 \text{ and } B = 1) \\ &= P(A = 1)P(B = 1) \\ &= a \cdot b. \end{aligned} \tag{1}$$

The probability of getting a one at the output, $P(C = 1)$, is equal to the probability of simultaneously getting ones at the inputs, namely, $P(A = 1)$ times $P(B = 1)$. So the AND gate multiplies the two values represented by the stochastic bit streams. In the figure, with bit streams of length 8, the values have a resolution of 1/8. Multiplication is simple and efficient in the stochastic representation precisely because the representation is uniform; no decoding and no re-encoding are required to operate on the values.


A
B
a_2 a_1 a_0 b_2 b_1 b_0
HA
FA
c_5 c_4 c_3 c_2 c_1 c_0
C

(a) Multiplication with a **conventional representation**: a carry-save multiplier, operating on 3-bit binary radix encoded inputs A and B. Each full adder (FA) module can be implemented with two XOR gates, two AND gates, and one OR gate. Each half adder (HA) can be implemented with one XOR gate and one AND gate.



(b) Multiplication with a **stochastic representation**: an AND gate. The inputs are stochastic bit streams A and B and the output is a stochastic bit stream C. Here, the probability of A is 6/8 and that of B is 4/8. The probability of $C = 6/8 \times 4/8 = 3/8$, as expected.

Figure 2: Multiplication Operation: $A \times B = C$. (a) A conventional implementation of multiplication. (b) A stochastic implementation of multiplication.

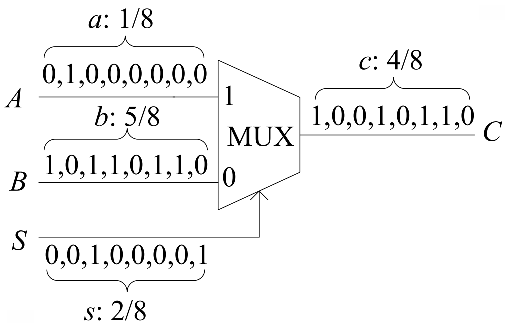


Figure 3: Scaled addition on stochastic bit streams, with a multiplexer (MUX). Here the inputs are 1/8, 5/8, and 2/8. The output is $2/8\times1/8+(1-2/8)\times5/8 = 4/8$, as expected.

Consider the operation of addition implemented stochastically. It is not feasible to add two probability values directly; this could result in a value greater than one, which cannot be represented as a probability value. However, we can perform *scaled* addition. Figure 3 shows a scaled adder operating on real numbers in the stochastic representation. It consists of a multiplexer (MUX), a digital construct that selects one of its two input values to be the output value, based on a third "selecting" input value. For the multiplexer shown in Figure 3, S is the selecting input. When $S = 1$, the output $C = A$. Otherwise, when $S = 0$, the output $C = B$. The Boolean function implemented by the multiplexer is $C = (A \wedge S) \vee (B \wedge \neg S)$. With the assumption that the three input stochastic bit streams A, B, and S are independent, the number represented by the output stochastic bit stream C is

$$\begin{aligned} c &= P(C = 1) \\ &= P(S = 1 \text{ and } A = 1) + P(S = 0 \text{ and } B = 1) \\ &= P(S = 1)P(A = 1) + P(S = 0)P(B = 1) \\ &= s \cdot a + (1 - s) \cdot b. \end{aligned} \tag{2}$$

Thus, with this stochastic representation, the computation performed by a multiplexer is the scaled addition of the two input values a and b, with a scaling factor of s for a and $1 - s$ for b.

Now consider the problem of designing digital circuits that operate on stochastic bit streams. We focus on combinational circuits, that is to say, memoryless digital circuits built with logic gates such as AND, OR, and NOT. For such circuits, suppose that we supply stochastic bit streams as the inputs; we will observe stochastic bit streams at the outputs. Accordingly, combinational circuits can be viewed as constructs that accept real-valued probabilities as inputs and compute real-valued probabilities as outputs.[1] An illustration of such computation is shown Figure 4. The circuit, consisting of an AND gate and an OR gate, accepts ones and zeros and produces ones and zeros, as any digital circuit does. If we set the input bits x_1, x_2 and x_3 to be one randomly and independently with specific probabilities, then we will get an output y that is one with a specific probability. For instance, given input probabilities $x_1 = 1/2$, $x_2 = 1$ and $x_3 = 1/4$, the circuit in Figure 4 produces an output y with probability 5/8. The figure illustrates computations with bit lengths of 8.

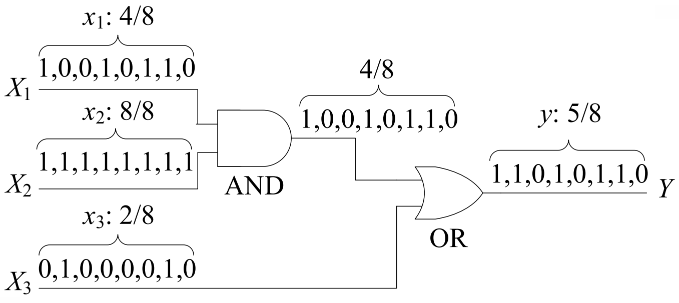


Figure 4: An example of logical computation on stochastic bit streams, implementing the arithmetic function $y = x_1x_2 + x_3 - x_1x_2x_3$. We see that, with inputs $x_1 = 1/2$, $x_2 = 1$ and $x_3 = 1/4$, the output is 5/8, as expected.

Compared to a binary radix representation, a stochastic representation is not very compact. With M bits, a binary radix representation can represent 2^M distinct numbers. To represent real numbers with a resolution of 2^{-M}, i.e., numbers of the form $\frac{a}{2^M}$ for integers a between 0 and 2^M, a stochastic representation requires a stream of 2^M bits. The two representations are at opposite ends of the spectrum: conventional binary radix is a maximally compressed, positional encoding; a stochastic representation is an uncompressed, uniform encoding.

2 Fault Tolerance

A stochastic representation, although not very compact, has an advantage over binary radix in terms of error tolerance. Suppose that the environment is noisy: bit flips occur and these afflict all the bits with equal probability. Compare the two representations for a fractional number of the form $\frac{a}{2^M}$ for integers a between 0 and 2^M. With a binary radix representation, in the worst case, the most significant bit gets flipped, resulting in a change of $\frac{2^{M-1}}{2^M} = \frac{1}{2}$. In contrast, with a stochastic representation, all the bits in a stream of length 2^M have equal weight. Thus, a single bit flip always results in a change of $\frac{1}{2^M}$, which is small in comparison.

Figure 5 illustrates the fault tolerance that our approach provides. The circuit in Figure 5(a) is a stochastic implementation while the circuit in Figure 5(b) is a conventional implementation. Both circuits compute the function:

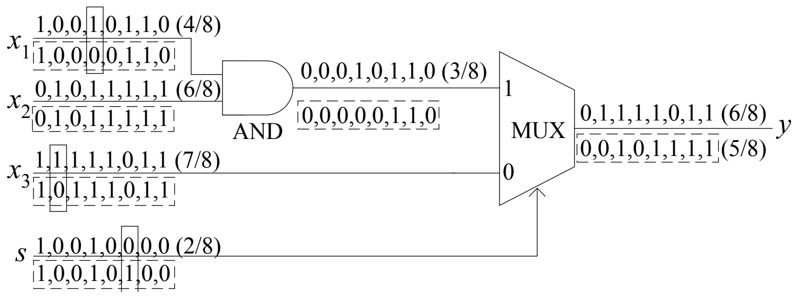
$$y = x_1x_2s + x_3(1 - s).$$

Consider the stochastic implementation. Suppose that the inputs are $x_1 = 4/8$, $x_2 = 6/8$, $x_3 = 7/8$, and $s = 2/8$. The corresponding bit streams are shown above the wires. Suppose that the environment is noisy and bit flips occur at a rate of 10%; this will result in approximately three bit flips for the stream lengths shown. A random choice of three bit flips is shown in the figure. The modified streams are shown below the wires. With these bit flips, the output value changes but by a relatively small amount: from 6/8 to 5/8.

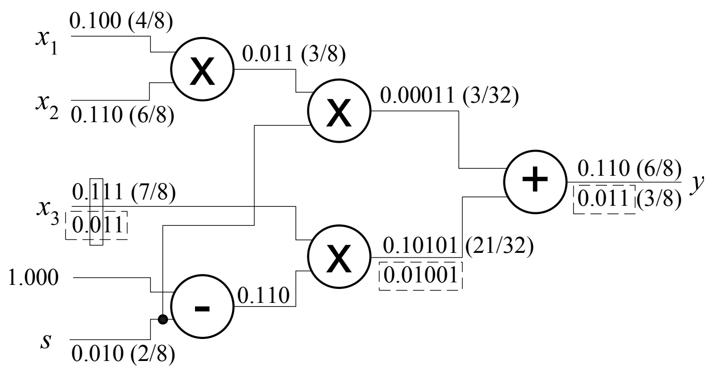
In contrast, Figure 5(b) shows a conventional implementation of the function with multiplication and addition modules operating on a binary radix representation: the real numbers $x_1 = 4/8$, $x_2 = 6/8$, $x_3 = 7/8$, and $s = 2/8$ are encoded as $(0.100)_2$, $(0.110)_2$, $(0.111)_2$, and $(0.010)_2$, respectively. The correct result is $y = (0.110)_2$, which equals 6/8. In the same situation as above, with a 10% rate of bit flips, approximately

[1]Throughout the proposal, when we say "logical computation" or just "computation" on stochastic bit streams we mean combinational logic operating on such bit streams. When we say "probability" without further qualification, we mean the probability of obtaining a one.

one bit will get flipped. Suppose that, unfortunately, this is the most significant bit of x_3. As a result, x_3 changes to $(0.011)_2 = 3/8$ and the output y becomes $(0.011_2) = 3/8$. This is a much larger error than we expect with the stochastic implementation.



(a) Stochastic implementation of the function $y = x_1x_2s + x_3(1 - s)$.



(b) Conventional implementation of the function $y = x_1x_2s + x_3(1 - s)$, using binary radix multiplier, adder and subtractor units.

Figure 5: A comparison of the fault tolerance of stochastic logic to that of conventional logic. The original bit sequence is shown above each wire. A bit flip is indicated with a solid rectangle. The modified bit sequence resulting from the bit flip is shown below each wire and indicated with a dotted rectangle.

With the stochastic representation, noise does not introduce more randomness. The bit streams are random to begin with, biased to specific probability values. Rather, noise distorts the bias, producing streams with different probabilities than intended. However, this change is small. With a bit flip rate of ϵ, the change is bounded by ϵ.[2]

The task of *analyzing* combinational circuitry operating on stochastic bit streams is well understood [26]. For instance, it can be shown that, given an input x, an inverter (i.e., a NOT gate) implements the operation $1 - x$. Given inputs x and y, an OR gate implements the operation $x + y - xy$. Analyzing the circuit in Figure 4, we see that it implements the function $x_1x_2 + x_3 - x_1x_2x_3$. Aspects such as signal correlations of reconvergent paths must be taken into account. Algorithmic details for such analysis were first fleshed out by the testing community [36]. They have also found mainstream application for tasks such as timing and power analysis [20, 23].

3 Synthesis

In this proposal, we will explore the more challenging task of *synthesizing* logical computation on stochastic bit streams that implements the functionality that we want. Naturally, since we are mapping probabilities to probabilities, we can only implement functions that map the unit interval $[0, 1]$ onto the unit interval $[0, 1]$. Based on the constructs for multiplication and scaled addition shown in Figures 2(b) and 3, we can readily

[2]In fact, with a bit flip rate of ϵ, a number p in the stochastic representation is biased to a number $p(1-\epsilon)+(1-p)\epsilon$, a change of $(1 - 2p)\epsilon$ in the value.

implement polynomial functions of a specific form, namely polynomials with non-negative coefficients that sum up to a value no more than one:

$$g(t) = \sum_{i=0}^{n} a_i t^i$$

where, for all $i = 0, \ldots, n$, $a_i \geq 0$ and $\sum_{i=0}^{n} a_i \leq 1$.

For example, suppose that we want to implement the polynomial $g(t) = 0.3t^2 + 0.3t + 0.2$ through logical computation on stochastic bit streams. We first decompose it in terms of multiplications of the form $a \cdot b$ and scaled additions of the form $sa + (1 - s)b$, where s is a constant:

$$g(t) = 0.8(0.75(0.5t^2 + 0.5t) + 0.25 \cdot 1).$$

Then, we reconstruct it with the following sequence of multiplications and scaled additions:

$$\begin{aligned} w_1 &= t \cdot t, \\ w_2 &= 0.5w_1 + (1 - 0.5)t, \\ w_3 &= 0.75w_2 + (1 - 0.75) \cdot 1, \\ w_4 &= 0.8 \cdot w_3. \end{aligned}$$

The circuit implementing this sequence of operations is shown in Figure 6. In the figure, the inputs are labeled with the probabilities of the bits of the corresponding stochastic streams. Some of the inputs have fixed probabilities and the others have variable probabilities t. Note that the different lines with the input t are each fed with *independent* stochastic streams with bits that have probability t.

What if the target function is a polynomial that is not decomposable this way? Suppose that it maps the unit interval onto the unit interval but it has some coefficients less than zero or some greater than one. For instance, consider the polynomial $g(t) = \frac{3}{4} - t + \frac{3}{4}t^2$. It is not apparent how to construct a network of stochastic multipliers and adders to implement it.

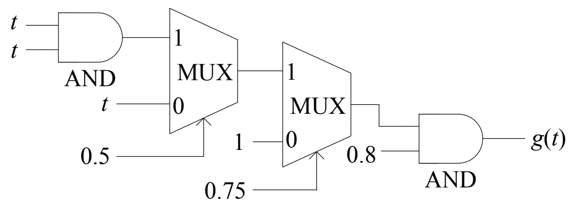


Figure 6: Computation on stochastic bit streams implementing the polynomial $g(t) = 0.3t^2 + 0.3t + 0.2$.

In prior work, we have proposed a general method for synthesizing arbitrary univariate polynomial functions on stochastic bit streams [31]. A necessary condition is that the target polynomial maps the unit interval onto the unit interval. Our major contribution is to show that this condition is also sufficient: we provide a constructive method for implementing any polynomial that satisfies this condition. Our method is based on some novel mathematics for manipulating polynomials in a special form called a *Bernstein polynomial.* In [34] we showed how to convert a general power-form polynomial into a Bernstein polynomial with coefficients in the unit interval. In [29] we showed how to realize such a polynomial with a form of "generalized multiplexing."

We illustrate the basic steps of our synthesis method with the example of $g(t) = \frac{3}{4} - t + \frac{3}{4}t^2$.

1. Manipulate the polynomial into a form with coefficients that are all in the unit interval:

$$g(t) = \frac{3}{4} \cdot [(1 - t)^2] + \frac{1}{4} \cdot [2t(1 - t)] + \frac{1}{2} \cdot [t^2].$$

   Note that the coefficients are $\frac{3}{4}$, $\frac{1}{4}$ and $\frac{1}{2}$, all of which are in the unit interval.

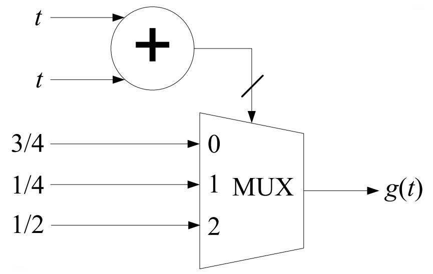


Figure 7: A generalized multiplexing circuit implementing the polynomial $g(t) = \frac{3}{4} - t + \frac{3}{4}t^2$.

2. Implement the polynomial with a multiplexing circuit, as shown in Figure 7. The block labeled "+" counts the number of ones among its two inputs; this is either 0, 1, or 2. The multiplexer selects one of its three inputs as its output according to this value. Note that the inputs with probability t are each fed with *independent* stochastic streams with bits that have probability t.

Generalizing the last example, we claim that we can implement any polynomial with the construct shown in Figure 8. The block labeled "+" has n inputs $X_1, \ldots, X_n$ and $\lceil \log_2(n+1) \rceil$ outputs. It consists of combinational logic that computes the weight of the inputs, that is to say, it counts the number of ones in the n Boolean inputs $X_1, \ldots, X_n$. The multiplexer (MUX) shown in the figure has "data" inputs $Z_0, \ldots, Z_n$ and the $\lceil \log_2(n+1) \rceil$ outputs of the weight counter as the selecting inputs. If the outputs of the weight counter are valued k $(0 \le k \le n)$, then the output Y of the multiplexer is set to Z_k. The constructs implements a polynomial of the form

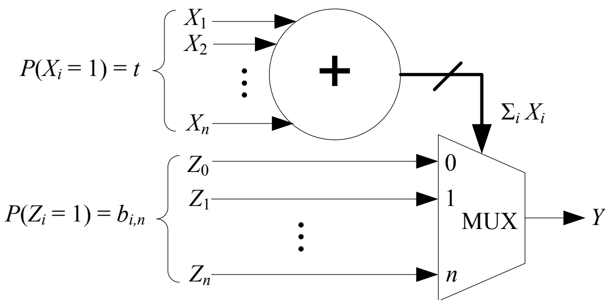


Figure 8: Combinational logic that implements a general polynomial of the form $B_n(t) = \sum_{i=0}^{n} b_{i,n} B_{i,n}(t)$
.

$$B_n(t) = \sum_{i=0}^{n} b_{i,n} B_{i,n}(t),$$

where

$$B_{k,n}(t) = \binom{n}{k} t^k (1-t)^{n-k},$$

with all coefficients $b_{i,n}$ in the unit interval. (This is a Bernstein polynomial representation.)

4 A Case Study: Synthesizing Fault-Tolerant Circuits

The *gamma correction function* is a nonlinear operation used to code and decode luminance and tristimulus values in video and still-image systems. It is defined by a power-law expression

$$V_{\text{out}} = V_{\text{in}}^{\gamma},$$

where V_{in} is normalized between zero and one [19]. We apply a value of $\gamma = 0.45$, which is the value used in most TV cameras. Consider the non-polynomial function

$$f_2(x) = x^{0.45}.$$

We can approximate this function by a polynomial of degree 6 with the coefficients:

$$b_0 = 0.0955, b_1 = 0.7207, b_2 = 0.3476, b_3 = 0.9988,$$
$$b_4 = 0.7017, b_5 = 0.9695, b_6 = 0.9939. \quad \square$$

Setting the corresponding probabilities of obtain a one to these values in the multiplexing construct in Figure 8, we obtain a stochastic implementation of the gamma correction function.

To evaluate the robustness of our method, we analyze the effect of soft errors. These are simulated by independently flipping the input bits for a given percentage of the computing elements. For example, if 5% noise was added to the circuit, this implies that 5% of the total number of input bits are randomly chosen and flipped. We compare the effect of soft errors on our implementation to that on conventional implementations.

The images in Figure 9 illustrate the fault tolerance of stochastic computation. When soft errors are injected at rate of 15%, the image generated by the conventional method is full of noisy pixels, while the image generated by the stochastic method is still recognizable.

5 Input/Output Interface

The premise for the synthesis method in this project is that the inputs and outputs to combinational circuitry consist of stochastic bit streams (or, equivalently, of stochastic bits on parallel wire bundles). Either the inputs are presented in this form or else they must be encoded this way, say from binary radix. Either the outputs are usable in this form or they must be decoded, say back into binary radix.

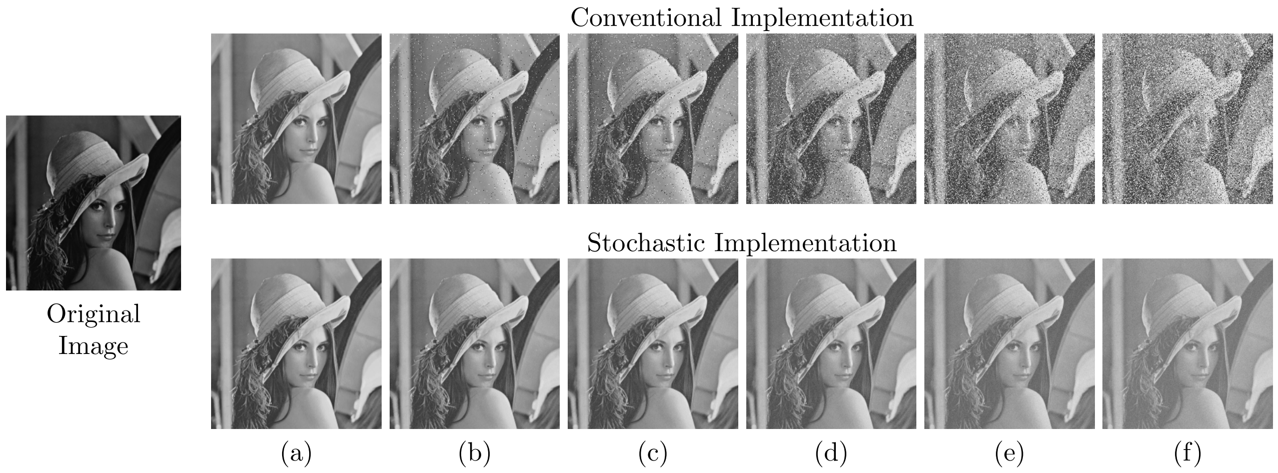


Figure 9: Fault tolerance for the gamma correction function. The images in the top row are generated by a conventional implementation. The images in the bottom row are generated by our stochastic implementation. Soft errors are injected at a rate of (a) 0%; (b) 1%; (c) 2%; (d) 5%; (e) 10%; (f) 15%.

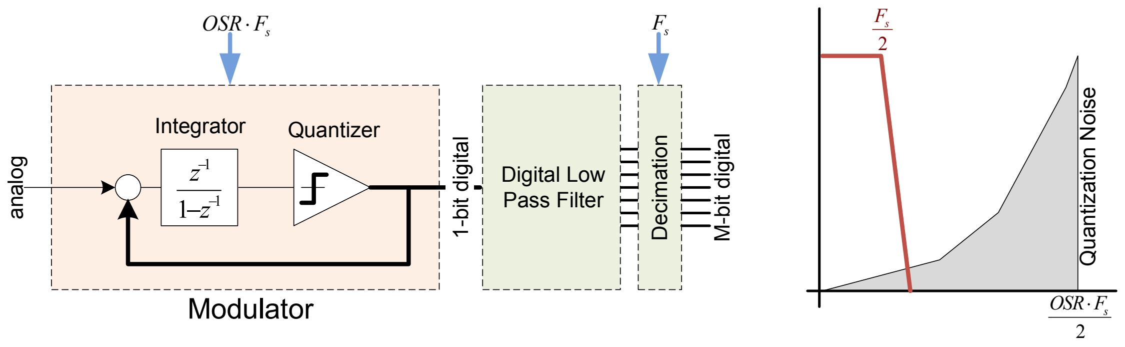


Figure 10: Block diagram for a sigma-delta A/D and the resulting noise shaping and decimation used

5.1 A/D and D/A Interfaces via Sigma-Delta Modulation

For a variety of circuit applications, for instance in sensors and embedded systems, the inputs are obtained from physical measurements in analog form, so as real-valued numbers. For our stochastic framework, we must transform such inputs into stochastic bit streams. The obvious technique for this task is to use a conventional analog-to-digital (A/D) converter followed by a multi-bit digital to single-bit stochastic converter [9].

In this project, we will develop an alternate, more elegant method that directly converts analog quantities to stochastic digital (A/SD) and stochastic digital to analog (SD/A). The A/SD and SD/A converters proposed here are based on single-bit oversampled converters that are oven called sigma-delta ($\Sigma\Delta$) converters [8, 12, 13, 24, 37, 46].

Sigma-delta converters utilize feedback to introduce noise shaping in the frequency domain such that quantization noise, which is inversely proportional to the conversion resolution, decreases rapidly with increased oversampling (OSR). Figure 10 shows the block diagram for a 1^{st}-order $\Sigma\Delta$ A/D. The modulator is usually followed by a low-pass filter and decimation stage. The resulting noise shaping and reduction in the in-band quantization noise is also shown on the right-hand side of Figure 10. In Figure 11 we have plotted the probability density function of the output voltage of integrator, i.e., before the quantizer, for a 2nd order $\Sigma\Delta$ modulator, for a input voltage equal to 0.277350. The comparator basically quantizes this signal over time to give a mean value of 0.277340. We note that this voltage "looks" like a stochastic value. In Figure 12 we plot the Fast Fourier Transform (FFT) of 1 bit output of the $\Sigma\Delta$ modulator in dB and log(frequency). We note that the quantization noise is moved to the higher frequencies. The flattening from 10^{-4} to 10^{-2}

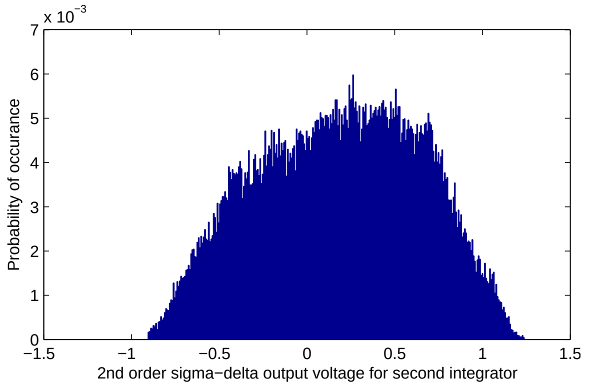


Figure 11: 2nd integrator voltage PDF for 2 stage $\Sigma\Delta$

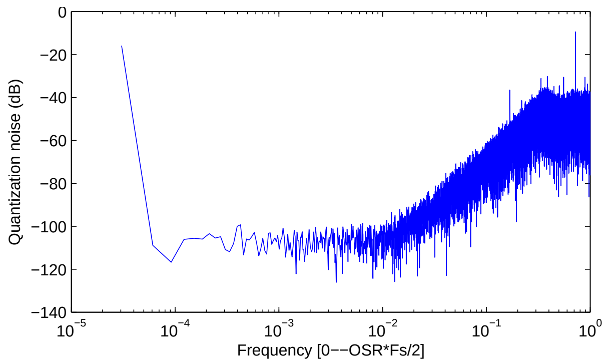


Figure 12: Quantization noise for $\Sigma\Delta$

is due the inclusion of KT/C thermal noise in our model. Additionally, the peak at lower frequencies, near DC, is due to the windowing impact of the DC quantity of 0.277350 at the input.

In our framework, we represent values stochastically, i.e., a value of 0.25 is represented by 25% of 1 and 75% of zeros. To represent this value accurately we need to over sample, i.e., operate the 1 bit stochastic value at a higher rate than our original value being represented. This is exactly what a modulator within a $\Sigma\Delta$ converter does. The $\Sigma\Delta$ modulator effectively use pulse-density modulation, i.e., a value of 0.5 is represented by an equal number of ones and zeros in the time domain. However, it exploits the frequency domain to reduce the oversampling ratio (OSR). This reduction in OSR compared to conventional randomization can significantly reduce the power consumption of the overall system.

We envision the overall system to look as follows: a $\Sigma\Delta$ analog-to-digital modulator, followed by the stochastic engine, which in turn in followed by a $\Sigma\Delta$ digital-to-analog modulator. To ensure that this entire setup is workable we need to ensure that no mathematical manipulation alters the frequency property of the signal at the final output so that we are able to use a simple $\Sigma\Delta$ digital-to-analog modulator.

In comparison to a tradition $\Sigma\Delta$ converter there is one unique property that needs to be presented by any A/SD converter. When two stochastic values that represent the same analog or multi-level digital value their stochastic representatives need to be uncorrelated. Unlike digital pseudo random generators analog circuits have "real noise" therefore $\Sigma\Delta$ modulator outputs for the same value are likely to have some variance. However, to ensure that the correlation is small we may be required to add additional randomness to these converters. There have been many method proposed over time including dithering and chaos to decorrelate the quantization noise from the input in $\Sigma\Delta$ converter [14, 37], including random dither at the input, at the quantizer, partial positive feedback etc. We will exploit these techniques to develop novel A/SD and SD/A converters that provide the correct translation including the additional decorrelation properties.

Part of the research explorations necessary to develop successful A/SD and SD/A converters include developing low power, low area designs for the $\Sigma\Delta$ analog-to-digital and digital-to-analog modulators, understanding the interplay between the modulators and the mathematical manipulations within the stochastic framework and developing dithering and chaotic techniques that result in stochastic representations of same value being sufficiently uncorrelated.

5.2 Stochastic Bit Streams from Physical Sources

For a variety of novel technologies, it may be possible to generate stochastic bit streams directly from random sources. For example, in [6], the authors describe a scheme for exploiting the intrinsic thermal noise of nanoscale CMOS devices as the random source. They call their approach Probabilistic CMOS (PCMOS). A PCMOS switch is an inverter with the input coupled to a noise source, as shown in Figure 13. With the input

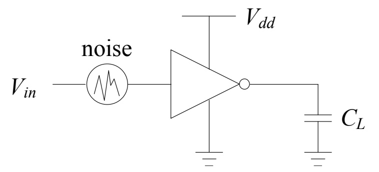


Figure 13: A PCMOS switch. It consists of an inverter with its input coupled to a noise source.

V_{in} set to 0 volts, the output of the inverter has a certain probability p $(0 \le p \le 1)$ of being at logical one. Suppose that the probability density function of the noise voltage V is $f(V)$ and that the trip point of the inverter is $V_{dd}/2$, where V_{dd} is the supply voltage. Then, the probability that the output is one equals the probability that the input to the inverter is below $V_{dd}/2$, or

$$p = \int_{-\infty}^{V_{dd}/2} f(V)\,\mathrm{d}V.$$

Thus, with a given noise distribution, p can be modulated by changing V_{dd}.

Figure 14 illustrates the process of generating stochastic bit streams in a more general context. In each clock cycle, a random source generates a value R obeying a certain probability density function $f(R)$. A comparator compares the value R with a constant value C: it outputs a one if $R < C$ and a zero otherwise. The output of the comparator is a stream of random bits that have probability

$$p = \int_{-\infty}^{C} f(R)\,\mathrm{d}R \tag{3}$$

of being one.

With such physical mechanisms, the random source may be cheap but the constant value may be expensive to implement. With PCMOS, the constant value C corresponds to a supply voltage [6]. Providing different supply voltages is comparatively expensive. If the application requires many stochastic bit streams with different probabilities, many constant values are required. The cost of generating these directly might be prohibitive.

In this proposal, we will explore synthesis strategy to mitigate this issue: we we develop a method for synthesizing combinational logic to transform a set of stochastic bit streams representing a limited number of probabilities into stochastic bit streams representing other target probabilities.

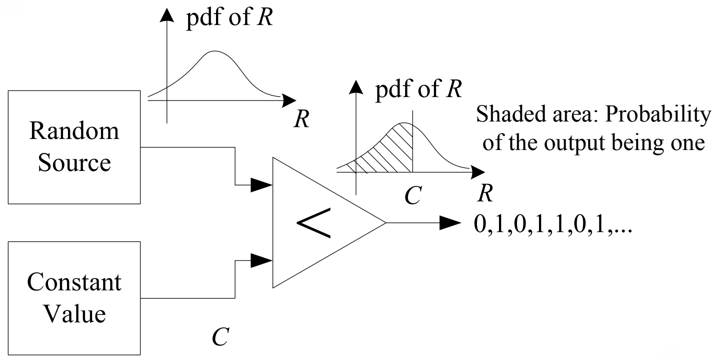


Figure 14: Generating stochastic bit streams from random or pseudo-random sources.

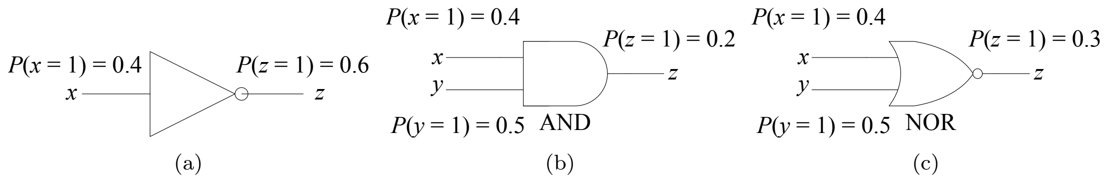


Figure 15: An illustration of transforming a set of source probabilities into new probabilities with logic gates. (a): An inverter implementing $p_z = 1 - p_x$. (b): An AND gate implementing $p_z = p_x \cdot p_y$. (c): A NOR gate implementing $p_z = (1 - p_x) \cdot (1 - p_y)$.

Example 1

Suppose that we have a set of source probabilities $S = \{0.4, 0.5\}$. As illustrated in Figure 15, we can transform this set into new probabilities:

1. *Given an input x with probability 0.4, an inverter will have an output z with probability 0.6 since*

$$P(z = 1) = P(x = 0) = 1 - P(x = 1). \tag{4}$$

2. *Given inputs x and y with independent probabilities 0.4 and 0.5, an AND gate will have an output z with probability 0.2 since*

$$\begin{aligned} P(z=1) &= P(x=1, y=1) \\ &= P(x=1)P(y=1). \end{aligned} \tag{5}$$

3. *Given inputs x and y with independent probabilities 0.4 and 0.5, a NOR gate will have an output z with probability 0.3 since*

$$\begin{aligned} P(z=1) = P(x=0, y=0) &= P(x=0)P(y=0) \\ &= (1-P(x=1))(1-P(y=1)). \end{aligned}$$

Thus, using combinational logic, we obtain the set of probabilities $\{0.2, 0.3, 0.6\}$ *from the set* $\{0.4, 0.5\}$. □

Motivated by these examples, we consider the general problem of synthesizing combinational logic that transforms a set of *source* probabilities into different *target* probabilities.

Example 2
Suppose that the target probability value 0.757. *With our method, we synthesize the circuit shown in Figure 16.*

In prior work, we considered different scenarios in terms of whether the source probabilities are *specified* and whether they can be *duplicated* [33]. In the case that the source probabilities are not specified and can be duplicated, we provide a specific choice, the set $\{0.4, 0.5\}$; we show how to synthesize logic that transforms probabilities from this set into arbitrary decimal probabilities. In this project, we will consider the more general problem of how to synthesize combinational logic to transform a set of source probabilities $S = \{p_1, p_2, \ldots, p_n\}$ into a target probability q.

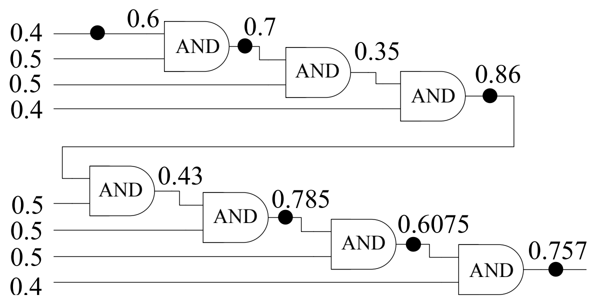


Figure 16: A circuit transforming the set of source probabilities $\{0.4, 0.5\}$ into a decimal output probability of 0.757. (Note that here we use a black dot to represent an inverter.)

6 Trading off Fault Tolerance vs. Bit Stream Length

In the introduction, we pointed out two major disadvantages of positional number systems compared to a unary representation: (1) the encoding / decoding overhead and the complexity of operations, and (2) poor fault tolerance, especially when most significant digits are faulty. The major issue causing the complexity of operations stems from costly carry propagation operations to convert the results of addition and multiplication into the canonical representation of the number. Conceptually, issues (1) and (2) are not tightly intertwined and one could come up with alternative encodings that harness the advantages of both unary and positional systems.

6.1 Positional Probabilities Number System

We propose a hybrid representation called *positional probabilities* that addresses a major shortcoming of the unary system: its poor representation compactness (linear vs. logarithmic in positional systems). As an example, consider number N representing a real number between 0 and 1 with an accuracy of 2^{-10}. To represent this number, we need 10 binary digits in a positional system and $2^{10} = 1024$ bits to represent it using the unary system. A hybrid encoding can be built using the following encoding:

$$N = p_{-1}.b^{-1} + p_{-2}.b^{-2} + p_{-3}.b^{-3} + \cdots p_{-k}.b^{-k} \tag{6}$$

where b is the base (e.g., $b = 2$), k is the number of digits, and $0 \le p_i \le 1$ are *real digits* approximated by fractions. The physical implementation of numbers in this system could be a combination of the bundle and the stream models of Figure 1. Each p_i is represented using a stream of zeros and ones, and k bundles of wires are used to carry the digits' bit streams. We can tradeoff the level of fault tolerance vs. representation compactness.

Figure 17 shows an example in which a real number between 0 and 1 is represented using base $b = 2$ and $k = 3$ with bit stream length of 16. However, for an accuracy of 2^{-10}, we have to wait for $2^{10}/2^3 = 128$ bits on each wire before deciding the value being transmitted (compare this to 1024 bits needed for a unary encoding). Hence to transmit the number, we need to wait for 128 clock cycles, each called a *slice*, containing three 0/1 bits on the $k = 3$ wires. After the end of the 128 cycles, the summation of 1's on wire i divided by 128 is the value of p_i. It is important to note that a single slice does not carry much useful information about the original number in the same way as a single bit in a unary system does not. It is the aggregate of the bit streams over many slices that provides an accurate value of the number.

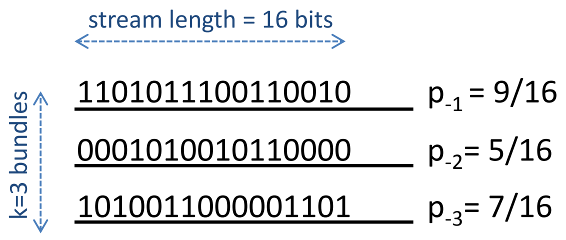


Figure 17: Three bundles of wire representing the number $9/16 \times b^{-1} + 5/16 \times b^{-2} + 7/16 \times b^{-3}$

In terms of fault tolerance, the positional probabilities system is better than binary, but worse than unary. In the example above, if we assume 10% bit flips, then the positional probabilities system is going to see an expected $128 \times 3 \times 0.1 = 38.4$ bit flips, the binary positional system is going to see one bit flip, and the unary system is going to see an expected $1024 \times 0.1 = 102.4$ bit flips. In the worst case for the positional probabilities encoding and the binary, all bit flips happen on the most significant digits. The absolute worst case error values will be 0.5, $38.4/128 \times 2^{-1} = 0.15$, and $102.4/1024 = 0.1$ for the binary, the positional probabilities, and the unary systems respectively. Changing k helps us tradeoff fault tolerance for better representation compactness.

Since our operations are performed on probabilities, no costly carry propagation operations are required. In the new encoding we can perform scaled addition and multiplication as:

$$\begin{aligned} b^{-1}(X+Y) &= b^{-1}[x_{-1}.b^{-1} + x_{-2}.b^{-2} + x_{-3}.b^{-3} + \cdots + y_{-1}.b^{-1} + y_{-2}.b^{-2} + y_{-3}.b^{-3} + \cdots] \\ &= (x_{-1} + y_{-1}).b^{-2} + (x_{-2} + y_{-2}).b^{-3} + \cdots \end{aligned} \tag{7}$$

$$\begin{aligned} X \times Y &= [x_{-1}.b^{-1} + x_{-2}.b^{-2} + x_{-3}.b^{-3} + \cdots].[y_{-1}.b^{-1} + y_{-2}.b^{-2} + y_{-3}.b^{-3} + \cdots] \\ &= (x_{-1}.y_{-1}).b^{-2} + (x_{-1}.y_{-2} + x_{-2}.y_{-1}).b^{-3} + (x_{-1}.y_{-3} + x_{-2}.y_{-2} + x_{-3}.y_{-1}).b^{-4} \cdots \end{aligned} \tag{8}$$

Both scaled addition and multiplication operations are equivalent to the single bit stream model, hence all techniques listed in Sections 3 and 4 can still be used with the new encoding. It is important to note that the scaled addition involves performing independent scaled additions on corresponding bit streams of the two numbers with no carry propagation between the k lines. Similarly, the multiplication operation involves large AND-OR gates the size of which grows asymptotically at a rate of $O(k^2)$, but does not involve carry propagation between the lines. The AND-OR logic can be broken down into smaller gates and pipelined, which may not be necessary given the limited number of wire bundles. Furthermore, one can easily cut on the number of terms, as one would in a binary multiplication with limited number of digits (e.g., if we only keep $k = 3$ bundles, then the terms with the multiplier b^{-4} get eliminated).

6.2 Non-Integer Base

We can further tune the tradeoff between fault tolerance and compactness by changing the base b to a real value between 1 and 2. In the positional probabilities encoding examples above, we used $k = 3$ to represent a number using 128 slices to meet the 2^{-10} resolution requirement. As we decrease b, the number of digits (wires) needed to represent a number in one slice increases. For example, using $b = 1.5$, we need

$\lceil -\log_b(2^{-10}) \rceil = 18$ digits to represent a number between 0 and 1 with an accuracy of 2^{-10} using only 0/1 digits.

The extra digits compared to $k = 10$ when $b = 2$ provide spatial redundancy that can tolerate permanent faults if random encoding of bits are used. Unlike $b = 2$ where $b^{-j} > \sum_{i=j+1}^{k} b^{-i}$, in $1 < b < 2$, we have $b^{-j} < \sum_{i=j+1}^{k} b^{-i}$, which means if a line i is stuck at 0, then the less significant digits can collectively add up to compensate the faulty line. Considering the fact that number representations in fractional bases are not unique, the effect of faults can be diminished over large samples of randomized encodings. As an example, the number 0.5 can be represented as 0.1000_2 with an accuracy of 1/16. If the most significant digit is stuck at 0, then the closest we can get to 0.5 is $0.0111_2 = 0.4375$. However, if base $b = 1.5$ is used, then 7 digits are required for an accuracy of 1/16, and many encodings of 0.5 satisfy the accuracy requirement, including $0.0100001_{1.5}$, $0.0011001_{1.5}$, $0.0001111_{1.5}$, and many more. Given the non-unique representations of a number, even if one or more lines are faulty, other representations of the number that do not use that particular line can partially compensate for the error caused by the faulty line.

Our goal is to study the exact relationship between parameters b, k and bit stream length so that we can formulate the synthesis of any combinational computations with specific resolution and fault tolerance requirements as an optimization problem and synthesize the circuit for it. Extending the number system to a signed-digit is another area that we intend to study. If instead of sending 0's and 1's we send +1's and -1's, we can represent numbers in the range [-1,+1]. Scaled addition and multiplication have to be defined in such a way that they be closed on the range [-1,+1] absent any bit flips. If bit flips cause the value to saturate beyond ± 1, the stochastic computations on individual bit streams are still valid, but the output unit has to detect an overflow or underflow due to errors.

7 Sequential Constructs

Prior research has has explored finite-state machines (FSMs) as sequential constructs for computation on stochastic bit streams Brown and Card proposed the stochastic computation of arithmetic functions with sequential logic, in the form of a one-dimensional, or linear, finite state machine (FSM) [4]. The basic topology is shown in Fig. 18. The machine has a set of states $S_0, S_1, \ldots, S_{N-1}$ arranged in a linear sequence. X is the input of the state machine. X' represents the negation of X. Given the current state S_i $(0 < i < N-1)$, the next state will be S_{i-1} if $X = 0$ and will be S_{i+1} if $X = 1$ (assuming that the machine is not in state S_0 or in state S_{N-1}, respectively).

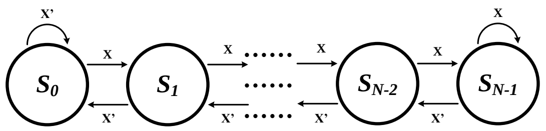


Figure 18: A generic linear state transition diagram.

With a stochastic encoding, X takes the form of a stochastic bit stream. As a result the state transition process is a special type of a Markov chain. The output Y of this state machine, not shown in Fig. 18, is a function of the current state: for some states the output Y is one and for the others it is zero. Thus, the output Y is also a stochastic bit stream. Based on different choices of the set of states that set the output Y to one, this linear FSM can be used to implement different functions.

For example, suppose that the output Y is one when the current state is in the set $\{S_{N/2}, \ldots, S_{N-1}\}$ and is zero when the current state is in the set $\{S_0, \ldots, S_{N/2-1}\}$. Then the functional relation between X and Y is approximately a tanh function [4],

$$y \approx \frac{e^{\frac{N}{2}x} - e^{-\frac{N}{2}x}}{e^{\frac{N}{2}x} + e^{-\frac{N}{2}x}}, \tag{9}$$

Through quadratic programming, the parameters of the linear FSM proposed by Brown and Card can be set so that it computes other functions, such as the absolute value function. However, due to the limited degree of freedom, relatively few target functions can be implemented efficiently with the linear FSM topology. For example, the target function,

$$T(x) = e^{-3x},$$

cannot be implemented efficiently by a linear FSM. Here the optimal solution obtained through quadratic programming for a 32-state linear FSM. The approximation error of this optimal solution is 2.0×10^{-3}. Even if we use a 64-state, a 128-state, or a 256-state FSM, the approximation error is still greater than 1.8×10^{-3}.

This project will explore a 2D mesh FSM topology for logical computation on stochastic bit streams, shown in Fig. 19. By adding an extra dimension in the state transition diagram, we significantly increase the degree of freedom when it comes to synthesizing functions.

The proposed topology has in total $M \times N = 2^R$ states, where R is a positive integer. We set $M = 2^{\lfloor \frac{R}{2} \rfloor}$, and $N = 2^{\lceil \frac{R}{2} \rceil}$. Both X and K are inputs of this state machine. X' and K' indicate the negation of X and K, respectively. The output Y of this state machine, not shown in Fig. 19, is a function of the current state. The inputs X and K consist of stochastic bit streams. We define the probability that each bit in the input stream X is one to be P_X, the probability that each bit in the input stream K is one to be P_K, the probability that each bit in the corresponding output stream Y is one to be P_Y, and the probability that the current state is $S_{i,j}(0 \leq i \leq M-1, 0 \leq j \leq N-1)$ under the input probability P_X and P_K to be $P_{i,j}(P_X, P_K)$.

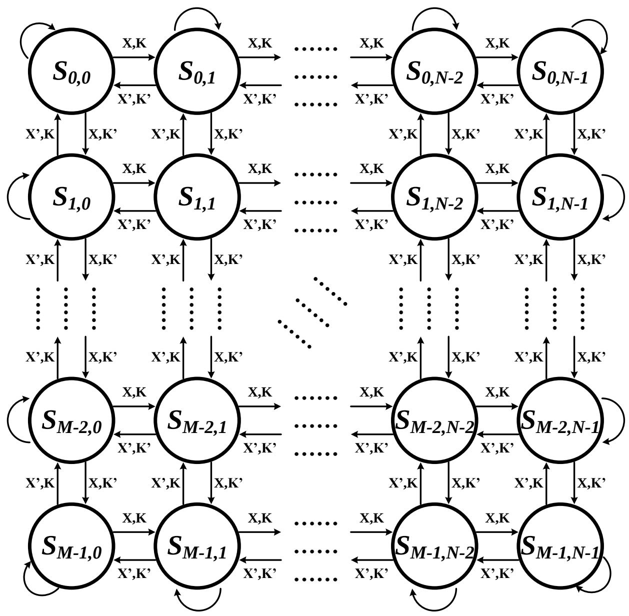


Figure 19: The state transition diagram of the 2D mesh FSM.

Based on the theory of Markov chains, the system shown in Fig. 19 is ergodic and will have a single stable hyperstate. The individual state probabilities $P_{i,j}(P_X, P_K)$ in the hyperstate must sum to unity over all $S_{i,j}$. Additionally, in the steady-state the probability of transitioning from state $S_{i,j}$ to its adjacent state in the next row, $S_{i+1,j}$, must equal the probability of transitioning from state $S_{i+1,j}$ to state $S_{i,j}$, and the probability of transitioning from state $S_{i,j}$ to its adjacent state in the next column, $S_{i,j+1}$, must equal the probability of transitioning from state $S_{i,j+1}$ to state $S_{i,j}$.

With a greater degree of freedom, we expect that 2D FSMs can implement functions that cannot be accurately implemented by 1D FSMs. Compared to stochastic implementations based on combinational logic, we expect those based on 2D mesh FSMs will require less hardware. We will also study different topologies of FSMs, including hypercubes and circulant graphs.

8 Context and Impact of Proposed Research

In a sense, the approach that we are advocating here is simply a highly redundant, probabilistic encoding of data. And yet, our synthesis methodology is a radical departure from conventional approaches. By transforming computations from the deterministic Boolean domain into arithmetic computations in the probabilistic domain, circuits can be designed with very simple logic. Such stochastic circuits are much more tolerant of errors. Since the accuracy depends only on the statistical distributions of the random bit streams, this fault tolerances scales gracefully to very large numbers of errors.

The proposed research builds upon significant prior work, but it is broad and transformative in scope because it establishes a new paradigm for *synthesis*. A sequence of early papers established the concept of logical computation on stochastic bit streams [11, 35]. These papers discussed basic operations such as multiplication and addition. Later papers delved into more complex operations, including exponential functions and square roots [15, 44]. In [4], the authors discuss the implementation of basic arithmetic operations as well as complex ones, including hyperbolic functions, with stochastic bit streams. They also discuss different forms of stochastic representation, including a "bipolar" representation for negative values. Much of the interest in computing with stochastic bit streams stems from the field of neural networks, where the concept is known as "pulsed" or "pulse-coded" computation [3, 7].

In fact, the general concept of stochastic computing dates back even earlier, to work by J. von Neumann in the 1950's [45]. He applied probabilistic logic to the study of thresholding and multiplexing operations on bundles of wires with stochastic signals. As he eloquently states in the introduction to his seminal paper, "Error is viewed not as an extraneous and misdirected or misdirecting accident, but as an essential part of the [design]." We find this view, that randomness and noise are integral to computation, to be compelling in the modern era of nanoscale electronics.

We point to two recent research efforts that embrace randomness in circuit and system design. In [6], the authors propose a construct that they call probabilistic CMOS (PCMOS) that generates random bits from intrinsic sources of noise. In [5], PCMOS switches are applied to form a probabilistic system-on-a-chip (PSOC); this system provides intrinsic randomness to the application layer, so that it can be exploited by probabilistic algorithms. In [25] and [10], the authors propose a methodology for designing stochastic processors, that is to say, processors that can tolerate computational errors caused by hardware uncertainties. They strive for a favorable trade-off between reliability and power consumption.

In contrast to architectural and system-level approaches, we are proposing a fundamentally new strategy for structuring logical computation. The strategy for computation has a pseudo *analog* character, reminiscent of computations performed by physical systems such as electronics on continuously variable signals such as voltage. In our case, the variable signal is the probability of obtaining a one in a stochastic yet *digital* bit stream. Indeed, our system could be built from ordinary, cheap digital electronics such as CMOS.

This is certainly counterintuitive: why impose an analog view on digital values? As we have outlined in this proposal, it might often be advantageous to do so, both from the standpoint of the hardware resources required as well as the error tolerance of the computation. Many of the functions that we seek to implement for computational systems such as signal processing are *arithmetic* functions, consisting of operations like addition and multiplication. Complex functions, such as exponentials and trigonometric functions, are generally computed through polynomial approximations, so through multiplications and additions. As we have argued, these operations are very natural and efficient when performed on stochastic bit streams.

We are the first to tackle the problem of synthesizing arbitrary polynomial functions through logical computation on stochastic bit streams. The synthesis results for our stochastic implementations of a variety of functions are convincing. The area-delay product is comparable to that of conventional implementations with adders and multipliers. Since stochastic bit streams are uniform, with no bit privileged above any other, the computation is highly error tolerant. As higher and higher rates of bit flips occur, the accuracy degrades gracefully.

Indeed, computation on stochastic bit streams could offer tunable precision: as the length of the stochastic bit stream increases, the precision of the value represented by it also increases. Thus, without hardware redesign, we have the flexibility to tradeoff precision and computation time. In contrast, with a conventional binary-radix implementation, when a higher precision is required, the underlying hardware has to be redesigned. Note that we have been evaluating our stochastic implementations under the conservative assumption that the clock rate will be the same as that of a conventional implementation. However, with much simpler hardware we could potentially implement computation with much higher clock rates, particularly if it is pipelined.

9 Educational and Outreach Plan

In a debate with an alchemist in 1628, the great French mathematician René Descartes denied the claim that probabilities are as good as certainties in science. Ever since, there has been a lingering stigma associated with estimations and approximations. Those who can, calculate things exactly. Those who can't, simulate and guess. Of course, in many disciplines of science and engineering, probabilistic analysis has become indispensable. However, it is generally applied as a tool for *characterizing uncertainty*: one postulates a definite model and then affixes uncertainties and error margins. In the physical and biological sciences, statistical analysis of data is pervasive. However, such analysis generally is applied as a tool for *inference*: given noisy experimental data, one attempts to extract information that is beyond the reach of direct measurements.

This project advocates stochastic methodologies for *design*. An important goal is to incorporate this viewpoint into the teaching curriculum in electrical and computer computer engineering. Starting with our undergraduate classes – computer engineering, logic design, microcontrollers, and electronics – to our graduate-level classes – VLSI CAD, architecture, analog, and "circuits and biology" – we will teach the students basic probability and develop stochastic concepts such as fault-tolerance, redundancy, and error-correction. Specifically, through these courses, we will we develop the broad theme of computing reliably with unreliable components and computing in terms of statistical distributions.

9.1 Minority Involvement Plan

The PIs will work with the University of Minnesotas College of Science and Engineering Diversity and Outreach program to involve underrepresented students in research. This program manages the NSF-funded North Star STEM Alliance–Minnesotas Louis Stokes Alliance for Minority Participation (LSAMP). One of the core principles of the Diversity and Outreach program is that Mentoring and introduction of research opportunities early in the undergraduate career is the best practice for retention. Through participation in the North Star programs, the students will present their research to North Star fellows to demonstrate their research. They can choose from a selection of outreach events that are provided by the North Star program including a Kickoff Day at the beginning of each year and a spring symposium in the spring semester to showcase research opportunities at the university. Each student will participate in one of these events during their fellowship. The undergraduate students attending these presentations are encouraged by North Star program to seek research positions in labs. North Star also supplies funding for underrepresented students to attend conferences when mentored by a graduate student to increase the exposure of the students to the research community beyond the Universitys laboratories.

9.2 Undergraduate Involvement in Research

The University of Minnesota offers many research opportunities for undergraduate research. Undergraduate research is supported by the university through the Undergraduate Research Opportunity Program. This is a competitive program that requires the students to write a proposal which gets reviewed and scored. The UROP program funds approximately 80% of the applications providing the students with $1400 stipend and $300 for lab supplies. These students generally are mentored by a graduate student in the lab. This provides graduate students the opportunity to learn mentoring skills and to develop interest in their field. The undergraduates can present their research at the end of the year in an undergraduate research symposium.

9.3 K-12 Outreach Plan

The College of Science Engineering (CSE) offers a summer high school student outreach program, Exploring Careers in Engineering and Physical Science (ECEPS). This program offers students a handson introduction to engineering, science and math opportunities on the University of Minnesota Twin Cities campus by providing the students tours, along with short projects, in different labs around the campus. This program is designed to appeal to and reach both girls and underrepresented minorities with an interest in the STEM disciplines. In particular, two of the four possible oneweek sessions are devoted to girls only.

10 Results of Prior NSF Support

Marc Riedel has two active NSF grants: 1) CAREER Award 0845650: "Computing with Things Small, Wet, and Random – Design Automation for Digital Computation with Nanoscale Technologies and Biological Processes." ($500,000 from Sept. 2009 to Aug. 2014). 2) EAGER grant CCF-0946601 "Synthesizing Signal Processing Functions with Biochemical Reactions" (with Keshab Parhi – $200,000 from Aug. 2009 to July, 2011). These awards have established novel and transformative approaches to design automation guided by physical views of computation. A broad theme is the application of expertise from an established field, digital circuit design, to new fields, such as nanotechnology and synthetic biology. The results have been published in [1, 2, 16–18, 27, 28, 30–32, 34, 38, 39].

Kia Bazargan had the NSF grant CCF-0347891, "CAREER: Computer-Aided Design of Mixed ASIC/ Reconfigurable Fabrics of the Nanometer Era." ($400,000 from January 15, 2005 to December 31, 2008). The HARP project was partially funded by this grant. The work was very well received at the FPGA05 conference [43], and later at Xilinx. The developed CAD algorithms in this effort have been distributed widely through the web (http://www.ece.umn.edu/users/kia/Download). Researchers from many universities and companies within and outside the US have downloaded our tool. The work was extended and published in the IEEE TCAD journal [47], and the method ultimately adopted in Xilinxs Virtex-V chips. Other projects partially funded by this grant include: statistical routing for FPGAs [40], statistical skew assignment algorithm for FPGAs [40, 41], estimation and optimization of noise reliability in circuits [42], and SPFD rewiring of FPGAs [21, 22].

Ramesh Harjani has not had NSF funding in the past five years, but had significant support up until 1996.

David J. Lilja has been PI or co-PI on numerous NSF-supported projects. A project related to the research in this proposal is, “NER: Designing Reliable Computers Using Molecular Nanotechnology” (CCR-0210197, 8/02 – 7/04, $70,000). This exploratory project introduced the concept of NanoBoxes as a possible abstraction for constructing reliable finite state machines for use in computer systems fabricated from future nano-devices. It provided partial support for one recently graduated Ph.D. student and partial support for another current Ph.D. student and spin-off projects for several M.S. students. This one-year effort resulted in two conference publications, two workshop presentations, a journal publication, and led to a three-year contract from the Semiconductor Research Corporation (SRC) to continue and expand this work. This project also led directly to the PI’s current project, NER: Characterizing and Modeling Magnetic Tunnel Junction Devices for a Spintronics-based Processor (ECS-0609023, 6/06 – 5/08, $100,000, co-PI: Jianping Wang).

References Cited

[1] M. Altun and M. D. Riedel. Lattice-based computation of Boolean functions. In *Design Automation Conference*, pages 609–612, 2010.

[2] J. Backes and M. D. Riedel. Reduction of interpolants for logic synthesis. In *International Conference on Computer-Aided Design*, 2010.

[3] C. Bishop. *Neural Networks for Patten Recognition.* Clarendon Press, 1995.

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

[5] L. Chakrapani, P. Korkmaz, B. Akgul, and K. Palem. Probabilistic system-on-a-chip architecture. *ACM Transactions on Design Automation of Electronic Systems*, 12(3):1–28, 2007.

[6] S. Cheemalavagu, P. Korkmaz, K. Palem, B. Akgul, and L. Chakrapani. A probabilistic CMOS switch and its realization by exploiting noise. In *IFIP International Conference on VLSI*, pages 535–541, 2005.

[7] S. Deiss, R. Douglas, and A. Whatley. A pulse coded communications infrastructure for neuromorphic systems. In *Pulsed Neural Networks*, chapter 6, pages 157–178. MIT Press, 1999.

[8] Frank Dropps and Ramesh Harjani. Gain calibration technique for increased resolution in FRC data converters. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 53(11), November 2006.

[9] B. Dufort and G. W. Roberts. *Analog Test Signal Generation Using Periodic $\Sigma\Delta$-Encoded Data Streams.* Kluwer Academic, 2000.

[10] P. S. Duggirala, S. Mitra, R. Kumar, and D. Glazeski. On the theory of stochastic processors. In *International Conference on Quantitative Evaluation of Systems*, pages 292–301, 2010.

[11] B. Gaines. Stochastic computing systems. In *Advances in Information Systems Science*, volume 2, chapter 2, pages 37–172. Plenum Press, 1969.

[12] Ramesh Harjani. *The Circuits and Filters Handbook*, chapter Integrated Analog-to-Digital Converters. CRC/IEEE Press, 1995.

[13] Ramesh Harjani and Tom Lee. FRC: A method for extending the resolutionof nyquist rate converters using oversampling. *IEEE Transactions on Circuits and Systems II*, pages 482–494, April 1998.

[14] Soren Hein. Exploiting chaos to suppress spurious tone in general double loop sigma-delta modulators. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 40(10), October 1993.

[15] C. Janer, J. Quero, J. Ortega, and L. Franquelo. Fully parallel stochastic computation architecture. *IEEE Transactions on Signal Processing*, 44(8):2110–2117, 1996.

[16] H. Jiang, A. P. Kharam, M. D. Riedel, and K. K. Parhi. A synthesis flow for digital signal processing with biomolecular reactions. In *IEEE International Conference on Computer-Aided Design*, pages 417–424, 2010.

[17] H. Jiang, M. D. Riedel, and K. K. Parhi. Digital signal processing with biomolecular reactions. In *IEEE Workshop on Signal Processing Systems*, pages 237–242, 2010.

[18] A. Kharam, H. Jiang, M. D. Riedel, and K. Parhi. Binary counting with chemical reactions. In *Pacific Symposium on Biocomputing*, 2011.

[19] D. Lee, R. Cheung, and J. Villasenor. A flexible architecture for precise gamma correction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 15(4):474–478, 2007.

[20] J.-J. Liou, K.-T. Cheng, S. Kundu, and A. Krstic. Fast statistical timing analysis by probabilistic event propagation. In *Design Automation Conference*, pages 661–666, 2001.

[21] P. Maidee and K. Bazargan. A generalized and unified SPFD-based rewiring technique. In *International Conference on Field Programmable Logic and Applications*, pages 305–310, 2007.

[22] P. Maidee and K. Bazargan. A fast SPFD-based rewiring technique. In *Asia South-Pacific Design Automation Conference*, 2010.

[23] R. Marculescu, D. Marculescu, and M. Pedram. Logic level power estimation considering spatiotemporal correlations. In *International Conference on Computer-Aided Design*, pages 294–299, 1994.

[24] Kavita Nair and Ramesh Harjani. A 96dB SFDR 50Ms/s digitally enhanced CMOS pipelined A/D converter. In *IEEE International Solid-State Circuits Conference*, 2004.

[25] S. Narayanan, J. Sartori, R. Kumar, and D. Jones. Scalable stochastic processors. In *Design, Automation and Test in Europe*, pages 335–338, 2010.

[26] K. P. Parker and E. J. McCluskey. Probabilistic treatment of general combinational networks. *IEEE Transactions on Computers*, 24(6):668–670, 1975.

[27] W. Qian, J. Backes, and M. D. Riedel. The synthesis of stochastic circuits for nanoscale computation. *International Journal of Nanotechnology and Molecular Computation*, 1(4):39–57, 2010.

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

[29] W. Qian and M. D. Riedel. The synthesis of robust polynomial arithmetic with stochastic logic. In *Design Automation Conference*, pages 648–653, 2008.

[30] W. Qian and M. D. Riedel. Synthesizing cubes to satisfy a given intersection pattern. In *International Workshop on Logic and Synthesis*, pages 217–224, 2010.

[31] W. Qian and M. D. Riedel. Synthesizing logical computation on stochastic bit streams. *submitted to Communications of the ACM*, 2010.

[32] W. Qian and M. D. Riedel. Two-level logic synthesis for probabilistic computation. In *International Workshop on Logic and Synthesis*, pages 95–102, 2010.

[33] W. Qian, M. D. Riedel, K. Barzagan, and D. Lilja. The synthesis of combinational logic to generate probabilities. In *International Conference on Computer-Aided Design*, pages 367–374, 2009.

[34] W. Qian, M. D. Riedel, and I. Rosenberg. Uniform approximation and Bernstein polynomials with coefficients in the unit interval. *European Journal of Combinatorics*, 32(3):448–463, 2011.

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

[36] J. Savir, G. Ditlow, and P. H. Bardell. Random pattern testability. *IEEE Transactions on Computers*, 33(1):79–90, 1984.

[37] Richard Schreier and Gabor C. Temes. *Understanding Delta-Sigma Data Converters*. Wiley-IEEE Press, 2004.

[38] P. Senum and M. D. Riedel. Rate-independent biochemical computational modules. In *Proceedings of the Pacific Symposium on Biocomputing*, 2011.

[39] A. Shea, B. Fett, M. D. Riedel, and K. Parhi. Writing and compiling code into biochemistry. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 456–464, 2010.

[40] S. Sivaswamy and K. Bazargan. Statistical generic and chip-specific skew assignment for improving timing yield of FPGAs. In *International Conference on Field Programmable Logic and Applications*, pages 429–434, 2007.

[41] S. Sivaswamy and K. Bazargan. Statistical analysis and process aware routing and skew assignment for FPGAs. *ACM Transactions on Reconfigurable Technology Systems*, 1(1):1–35, 2008.

[42] S. Sivaswamy, K. Bazargan, and M. Riedel. Estimation and optimization of reliability of noisy digital circuits. In *International Symposium on Quality of Electronic Design*, pages 213–219, 2009.

[43] S. Sivaswamy, G. Wang, C. Ababei, K. Bazargan, R. Kastner, and E. Bozorgzadeh. Harp: Hardwired routing pattern FPGAs. In *International Symposium on Field Programmable Gate Arrays (FPGA)*, pages 21–29, 2005.

[44] S. Toral, J. Quero, and L. Franquelo. Stochastic pulse coded arithmetic. In *International Symposium on Circuits and Systems*, volume 1, pages 599–602, 2000.

[45] J. von Neumann. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In *Automata Studies*, pages 43–98. Princeton University Press, 1956.

[46] Feng Wang and Ramesh Harjani. *Design of Modulators for Oversampled Converters*. Kluwer Academic Publishers, 1998.

[47] G. Wang, S. Sivaswamy, C. Ababei, K. Bazargan, R. Kastner, and E. Bozorgzadeh. Statistical analysis and design of harp FPGAs. *IEEETransactions on Computer-Aided Design of Integrated Circuits and Systems*, 25:2088–2102, 2006.

Project Summary

SHF Medium: Digital Yet Deliberately Random –

Synthesizing Logical Computation on Stochastic Bit Streams

Most digital systems operate on a positional representation of data, such as binary radix. A positional representation is a compact way to encode signal values: in binary radix, 2^n distinct values can be represented with n bits. However, operating on it requires complex logic: in each operation such as addition or multiplication, the signal must be "decoded," with the higher order bits weighted more than the lower order bits. This project advocates an alternative representation: random bit streams where the signal value is encoded by the probability of obtaining a one versus a zero. This representation is much less compact than binary radix. However, complex operations can be performed with very simple logic. In particular, *arithmetic* functions, consisting of operations like addition and multiplication can be implemented very efficiently. Complex functions, such as exponentials and trigonometric functions, can be computed through polynomial approximations.

Because a stochastic representation is uniform, with all bits weighted equally, it is highly tolerant of soft errors (i.e., bit flips). Computation on stochastic bit streams offers tunable precision: as the length of the stochastic bit stream increases, the precision of the value represented by it also increases. Thus, without hardware redesign, one has the flexibility to tradeoff precision and computation time. In contrast, with a conventional binary-radix implementation, when a higher precision is required, the underlying hardware has to be redesigned.

This project will develop and apply a unified framework for synthesizing such computation from the circuit level to the architectural and system level. A synthesis strategy for combinational constructs will will be developed, based on polynomial approximations. A synthesis strategy for sequential constructs will be developed based on mesh and circulant topologies for finite-state machines. Hybrid encoding schemes, in which multiple stochastic bit streams are assigned positional weighting, will be explored. Analog expertise will be applied to the task of converting analog input and output signals into and out of stochastic bit streams: specifically, a method will be developed that directly converts analog quantities to bit streams based on single-bit oversampling with *sigma-delta* converters. Also, techniques for transforming probability values with combinational logic will be developed. Architectures that are tailored to specific domains such as scientific computing will be explored; in particular the project will target applications that are data-intensive yet probabilistic in nature.

Intellectual Merit: This new approach to circuit design forms the basis of a novel view of computation: instead of transforming definite inputs into definite outputs, circuits and systems transform probability values into probability values; so, conceptually, real-valued probabilities are both the inputs and the outputs. The computation has a pseudo *analog* character, reminiscent of computations performed by physical systems such as electronics on continuously variable signals such as voltage. In the new paradigm, the variable signal is the probability of obtaining a one versus a zero in a stochastic yet *digital* bit stream. Indeed, the system can be built from ordinary digital electronics such as CMOS. Thus, the design methodology imposes an analog view on top digital values. (Of course, the digital values themselves are an abstraction sitting on top of analog voltages values.) With this paradigm change, expertise in analog computation can be brought to bear on the design of robust digital systems.

Broader Impact: If successful, the proposed research will provide an integrated design paradigm for robust digital computation, applicable to both existing technologies such as CMOS as well novel nanoscale technologies. An important goal of the project is to communicate the goals and the impetus for interdisciplinary research to a wide audience. A new course will be developed, titled "*Circuits, Computation, and Biology*" offered jointly through the ECE Department and the new Biomedical Informatics and Computational Biology Program at the University of Minnesota. Building upon current research efforts that include female students, underrepresented students will be recruited into the project.

Keywords: Stochastic Computing, Probabilistic Circuits, Fault-Tolerant Circuits, Analog Circuits