Chapter 5
Deterministic Approaches to Bitstream Computing

Marc Riedel

Abstract Stochastic logic allows complex arithmetic to be performed with very simple logic, but it suffers from high latency and poor precision. Furthermore, the results are always somewhat inaccurate due to random fluctuations. The random or pseudorandom sources required to generate the representation are costly, consuming a majority of the circuit area (and diminishing the overall gains in area). This chapter reexamines the foundations of stochastic computing and comes to some surprising conclusions. It demonstrates that one can compute deterministically using the same structures that are used to compute stochastically. In doing so, the latency is reduced by an exponential factor; also, the area is reduced significantly (and this correlates with a reduction in power); and finally, one obtains completely accurate results, with no errors or uncertainty. This chapter also explores an alternate view of this deterministic approach. Instead of viewing signals as digital bit streams, we can view them as periodic signals, with the value encoded as the fraction of the time that the signal is in the high (on) state compared to the low (off) state in each cycle. Thus we have a *time-based encoding*. All of the constructs developed for stochastic computing can be used to compute on these periodic signals, so the designs are very efficient in terms of area and power. Given how precisely values can be encoded in the time, the method could produce designs that have much lower latency that conventional ones.

5.1 Introduction

As detailed throughout this book, the topic of stochastic computing has been investigated from many angles, by many different researchers. In spite of the activity, it is fair to say that the practical impact of the research has been modest. In our view, interest has been sustained because of the intellectual appeal of the

Marc Riedel
University of Minnesota, Minneapolis, MN, USA. e-mail: mriedel@umn.edu

paradigm. It presents a completely different way of computing functions with digital logic. Complex functions can be computed with remarkably simple structures. For instance, multiplication can be performed with a single AND gate. Complex functions such as exponentiation, absolute value, square roots, and hyperbolic tangent can each be computed with a very small number of gates [1]. Although this is a claim that can only be justified through design examples, stochastic designs consistently achieve $50\times$ to $100\times$ reductions in gate count over a wide range of applications in signal, image and video processing, compared to conventional binary radix designs [1]. Savings in area correlate well with savings in power, a critical metric.

Note that while stochastic computation is digital – operating on 0's and 1's – and performed with ordinary logic gates, it has an "analog" flavor: conceptually, the computation consists of mathematical operations on real values, the probabilities of the streams. The approach is a compelling and natural fit for computing mathematical functions, for applications such as image processing and neural processing.

The intellectual appeal notwithstanding, the approach has a glaring weakness: the latency it incurs. A stochastic representation is not compact: to represent 2^M distinct numbers, it requires roughly 2^{2M} bits, whereas a conventional binary representation requires only M bits. When computing on serial bit streams, this results in an exponential, near-disastrous increase in latency. The simplicity of the logic generally translates to very short critical paths, so one could, in principle, bump up the clock to very high rates. This could mitigate the increase in latency. But there are practical limitations to increasing the clock rate [2, 3].

Another issue is the cost of generating randomness. Most implementations have used pseudo-random number generators such as linear-feedback shift registers (LFSRs). The cost of these easily overwhelms the total area cost, completely offsetting the gains made in the structures for computation [4, 5]. Researchers have explored sources of true randomness [6, 7]. Indeed, with emerging technologies such as nanomagnetic logic, exploiting true randomness from physical sources could tip the scales, making stochastic computing a winning proposition [8]. Still, the latency and the cost of interfacing random signals with deterministic signals make it a hard sell.

In this chapter, we reexamine the foundations of stochastic computing, and come to some surprising conclusions. Why is computing on probabilities so powerful, conceptually? Why can complex functions be computed with such simple structures? Intuition might suggest that somehow we are harnessing deep aspects of probability theory; perhaps we are computing approximate answers to hard problems efficiently through "sampling" as with Monte Carlo simulations. This intuition is wrong.

The key to the efficiency of stochastic computing is much simpler: it stems from performing streaming computation on values represented by *quantity*, on a uniform representation, rather than representing values by *position*, as they are in binary radix. In this chapter, we demonstrate that, if the computation is properly structured, we can compute deterministically on bit streams using the same structures as we use when computing stochastically.

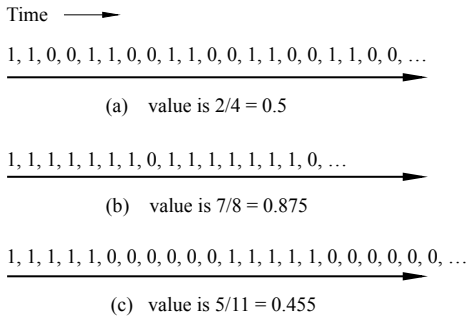


Fig. 5.1: Digital signals encoded as periodic bit streams. The values represented are (a) 0.5, (b) 0.875, and (c) 0.455.

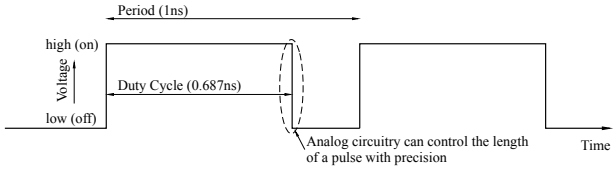
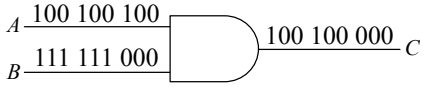


Fig. 5.2: A Pulse-Width Modulated (PWM) signal. The value represented is the fraction of the time that the signal is high in each cycle, in this case 0.687.

$$p_A = 1/3 = 100 \rightarrow 100\ 100\ 100$$
$$p_B = 2/3 = 110 \rightarrow 111\ 111\ 000$$



$$p_C = p_A p_B = 2/9$$

Fig. 5.3: Multiplication with a single AND gate, operating on deterministic periodic bit streams.

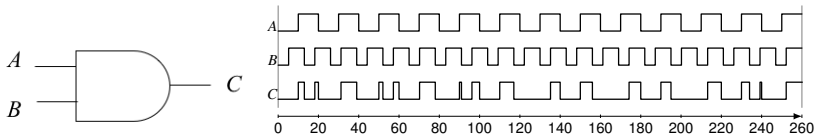


Fig. 5.4: Multiplication with a single AND gate: operating on deterministic periodic signals. *A* represents 0.5 with a period of 20ns; *B* represents 0.6 with a period of 13ns. The output signal *C* from t=0ns to 260ns represents 0.30, the expected value from multiplication of the inputs.

Next, this chapter explores a generalization of this deterministic approach. Instead of computing on bit *streams*, we explore computing on *periodic* signals, with the value encoded as the fraction of the time that the signal is in the high (on) state compared to the low (off) state in each cycle. Consider the examples in Figures 5.1 and 5.2. We will call digital bit streams of this sort **uniform** bit streams. We will call analog signals of this sort **pulse-width modulated (PWM)** signals. By exploiting pulse width modulation, signals with specific values can be generated by adjusting the frequency and duty cycles of the PWM signals. Note that a PWM signal is, in fact, *digital* in the sense that the voltage level is either 0 or 1. However, it represents a real-valued (analog) signal by its duration.

But how can one compute on such periodic signals? Recall that with stochastic logic, multiplication is performed with a single AND gate. Simply connecting two periodic signals to the inputs of an AND gate will evidently not work. With the two signals lining up, multiplying $1/2$ by $1/2$ would produce an output signal equal to $1/2$, not equal to $1/4$, the value required. However, suppose that one adopts the following strategy when generating the bit streams: hold each bit of one stream, while cycling through all the bits of the other stream. Figure 5.3 gives an example. Here the value $1/3$ is represented by the bits 100 repeating, while the value $2/3$ is represented by the 110, clock-divided by three. The result is $2/9$, as expected. This method works in general for all stochastic constructs.

In an analogous way, we can perform operations on PWM signals. For instance, one can use periodic signals with relatively prime frequencies. Figure 5.4 shows an example of multiplying two values, 0.5 and 0.6, represented as PWM signals. The period of the first is 20ns and that of the second is 13ns. The figure shows that, after performing the operation for 260ns, the fraction of the total time the output signal is high equals the value expected when multiplying the two input values, namely 0.3.

The idea of computing on time-encoded signals has a long history [9–12]. We have been exploring the idea of time-based computing with constructs developed for stochastic computing [13, 14]. We note that other researchers have explored very similar ideas in the context of LDPC decoding [15].

As we will argue, compared to computing on stochastic bit streams, we can reduce the latency significantly – by an exponential factor – with deterministic approaches. Of course, compared to binary radix, uniform bit streams still incur high latency. However, with PWM signals, the precision is no longer dependent on the length of pulses, but rather on how accurately the duty cycle can be set.

As technology has scaled and device sizes have gotten smaller, the supply voltages have dropped while the device speeds have improved [16]. Control of the dynamic range in the voltage domain is limited; however, control of the length of pulses in the time domain can be precise [16, 17]. Encoding data in the time domain can be done more accurately and more efficiently than converting signals into binary radix. Given how precisely values can be encoded in time, our method could produce designs that are much faster than conventional ones – operating in the terahertz range. Figure 5.5 compares the conventional approach, consisting of an analog-to-digital converter (ADC) that produces binary radix, to the new methods that we are proposing here.

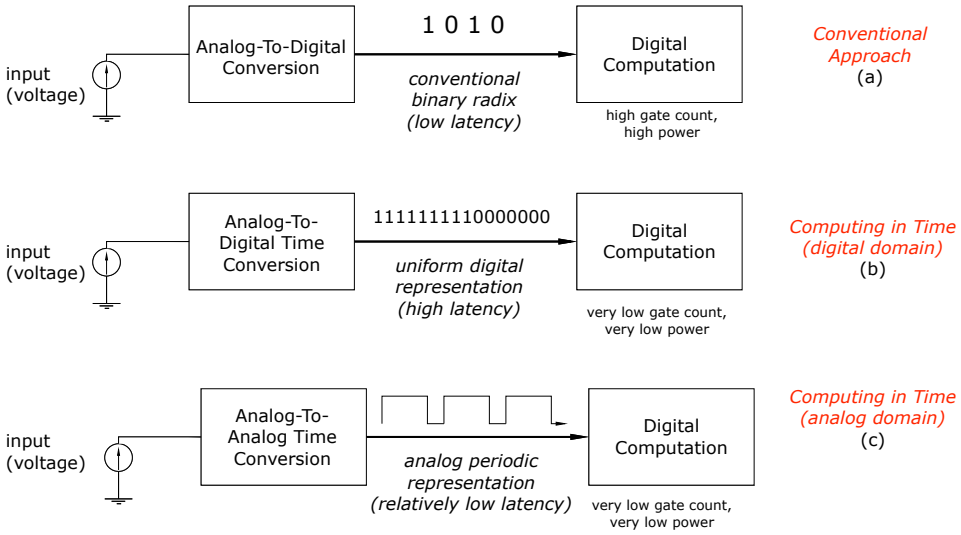


Fig. 5.5: Comparison of (a) the conventional approach, namely digital computation on binary radix; to (b) our methodology on uniform bit streams; and (c) our methodology on pulse-width modulated (PWM) signals.

5.2 A Deterministic Approach

The benefit of stochastic computing is that complex operations can be performed with very simple logic. We point the reader to a subset of the work that demonstrates this: [4,5,18,18–29]. This body of work includes examples of both basic and applied operations, ranging from multiplication, scaled addition, and exponentiation; to polynomial approximations of trigonometric functions; to LDPC decoders; to video processing operations such as edge detection. Across the board, the examples demonstrate a reduction in area by an order of magnitude or more compared to conventional designs.

An obvious drawback of the stochastic paradigm is the high latency that results, due to the length of the bit streams. Another is that the computation suffers from errors due to random fluctuations and correlations between the streams. These effects worsen as the circuit depth and the number of inputs increase [4]. While the logic to perform the computation is simple, generating random or pseudorandom bit streams is costly. Indeed, in prior work, pseudorandom constructs such as linear feedback shift registers (LFSRs) accounted for as much as 90% of the area of stochastic circuit designs [19,20]. This significantly diminishes the area benefits.

In this chapter, we argue that randomness is *not* a requirement for the paradigm. We show that the same computation can be performed on deterministically generated bit streams. The results are completely accurate, with no random fluctuations. Without the requirement of randomness, bit streams can be generated inexpensively. Most importantly, with our approach, the latency is reduced by a factor of approx-

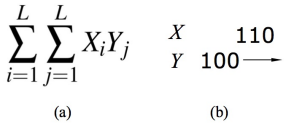


Fig. 5.6: Discrete Convolution. (a) Mathematical operation on two bit streams, X and Y. (b) Intuition: convolution is equivalent to sliding one bit streams past the other.

imately $1/2^n$, where n is the equivalent number of bits of precision. (For example, for the equivalent of 10 bits of precision, the bit stream length is reduced from 2^{20} to only 2^{10}.) As is the case with stochastic bit streams, all bits in our deterministic streams are weighted equally. Accordingly, as is the case with stochastic circuits, our deterministic circuits have a high degree of tolerance to soft errors.

5.2.1 Intuitive View

Conceptually, an operation such as multiplication in stochastic logic works by randomly *sampling* the inputs. This is achieved by randomizing the input bit streams and then intersecting them. This approach is easy to understand but incurs a lot of overhead: one must create the random bit streams, say with constructs such as LFSRs; this is costly. Furthermore, one must do a *lot* of sampling. Indeed, as explained in Section 5.2.2, in order to obtain a result that is equivalent in precision to n bits, one must sample 2^{2n} bits. Randomness requires, in effect, "oversampling" to get a statistically accurate result [5].

But is such randomly sampling necessary? *Why not simply intersect two deterministic bit streams.* Consider the mathematical operation of convolution. Intuitively, it consists of three operations: slide, multiply, and sum. Figure 5.6 illustrates this. If we implement this operation on uniform deterministic bit streams, the result will be equivalent to a stochastic operation.

Example 1

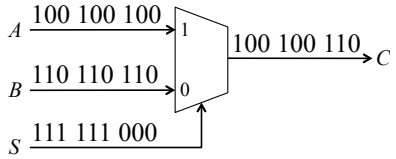


Fig. 5.7: Scaled Addition via Convolution, by Clock Dividing a Signal.

An example of multiplication by "clock-diving" one input stream and repeating the other was shown in Figure 5.3 in the introduction. Suppose that we wish to perform scaled addition, say on inputs $p_A = 1/3$, $p_B = 2/3$, and $p_S = 2/3$. We can divide the bit stream on the select input S to a multiplexer, while the bit streams for the operands A and B repeat:

$$p_A = 1/3 = 100 \rightarrow 100100100$$
$$p_B = 2/3 = 110 \rightarrow 110110110$$
$$p_S = 2/3 = 110 \rightarrow 111111000$$

Figure 5.7 illustrates that the result is $p_C = p_S p_A + (1 - p_S) p_B = 2/9 + 2/9 = 4/9$.

5.2.2 Comparing Stochastic and Deterministic Representations

A stochastic representation maintains the property that each bit of one stream meets every bit of an other stream the same number of times, but this property occurs *on average*, meaning the bit streams have to be much longer than the resolution they represent due to random fluctuations. The bit stream length N required to estimate the average proportion within an error margin ε is

$$N > \frac{p(1-p)}{\varepsilon^2}$$

(This is proved in [5].) To represent a value within a binary resolution $1/2^n$, the error margin ε must equal $1/2^{n+1}$. Therefore, the bit stream must be greater than 2^{2n} uniform bits long, as the $p(1-p)$ term is at most 2^{-2}. This means the length of a stochastic bit stream increases *exponentially* with the desired resolution. This results in enormously long bit streams. For example, if we want to find the proportion of a random bit stream with 10-bit resolution ($1/2^{10}$), we will have to observe at least 2^{20} bits. This is over a thousand times longer than the bit stream required by a deterministic uniform representation.

The computations also suffer from some level of correlation between bit streams. This can cause the results to bias away from the correct answer. For these reasons, stochastic logic has only been used to perform approximate computations. Another related issue is that the LFSRs must be at least as long as the desired resolution in order to produce bit streams that are sufficiently random. A "Randomizer Unit", described in [22], uses a comparator and LFSR to convert a binary encoded number into a random bit stream. Each independent random bit stream requires its own generator. Therefore, circuits requiring i independent inputs with n-bit resolution need i LFSRs with length L approximately equal to $2n$. This results in the LFSRs dominating a majority of the circuit area.

By using deterministic bit streams, we avoid all problems associated with randomness while retaining all the computational benefits associated with a stochastic representation. However, we can use *much* shorter bit streams to achieve the same

precision: to represent a value with resolution $1/2^n$ in a deterministic representation, the bit stream must be 2^n bits long. The computations are also completely accurate; they do not suffer from correlation. The next section discusses three methods for generating independent deterministic bit streams and gives their circuit implementations. Without the requirement of randomness, the hardware cost of the bit stream generators is reduced, so it is a win in every respect.

5.2.3 Deterministic Methods

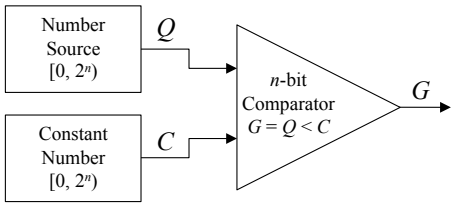


Fig. 5.8: Converter module

We present three alternative approaches to deterministic computation on uniform bit streams. These differ in how the uniform bit streams are generated. We note that the computational structures themselves are identical to those developed for stochastic computing. Accordingly, existing designs for arithmetic, signal processing and video processing can be used. We illustrate the approach with the simplest example: multiplication with an AND gate.

The three methods for generating the uniform bit streams are: (1) using relatively prime lengths; (2) rotation; and (3) clock division. For each method, the hardware complexity of the circuit implementation is given. The computation time of each method is the same. Each method is implemented using a bit stream generated from "converter" modules, illustrated in Figure 5.8. The modules are similar to the "Randomizer Unit" [19]; the difference is that the LFSR is replaced by a deterministic number source.

5.2.4 Relatively Prime Bit Lengths

The "relatively prime"' method maintains independence by using bit streams that have relatively prime lengths. Here the ranges $[0, R_i)$ between converter modules are relatively prime. Figure 5.9 demonstrates the method with two bit streams A and B, one with operand length four and the other with operand length three. The bit streams are shown in array notation to show the position of each bit in time.

$$\begin{array}{l} a_0\,a_1\,a_2\,a_3\,a_0\,a_1\,a_2\,a_3\,a_0\,a_1\,a_2\,a_3 \\ b_0\,b_1\,b_2\,b_0\,b_1\,b_2\,b_0\,b_1\,b_2\,b_0\,b_1\,b_2 \end{array}$$

Fig. 5.9: Two bit streams generated by the "relatively prime" method

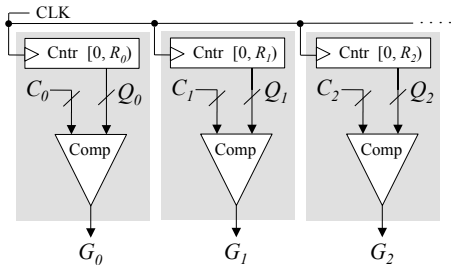


Fig. 5.10: Circuit implementation of the "relatively prime" method

$$\begin{array}{l} a_0\,a_1\,a_2\,a_3\,a_0\,a_1\,a_2\,a_3\,a_0\,a_1\,a_2\,a_3\,a_0\,a_1\,a_2\,a_3 \\ b_0\,b_1\,b_2\,b_3\,b_3\,b_0\,b_1\,b_2\,b_2\,b_3\,b_0\,b_1\,b_1\,b_2\,b_3\,b_0 \end{array}$$

Fig. 5.11: Two bit streams generated by the "rotation" method

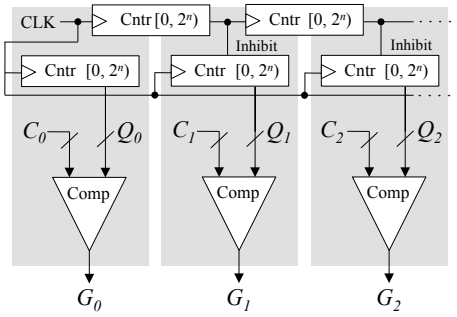


Fig. 5.12: Circuit implementation of the "rotation" method

$$\begin{array}{l} a_0\,a_1\,a_2\,a_3\,a_0\,a_1\,a_2\,a_3\,a_0\,a_1\,a_2\,a_3\,a_0\,a_1\,a_2\,a_3 \\ b_0\,b_0\,b_0\,b_0\,b_1\,b_1\,b_1\,b_1\,b_2\,b_2\,b_2\,b_2\,b_3\,b_3\,b_3\,b_3 \end{array}$$

Fig. 5.13: Two bit streams generated by the "clock division" method

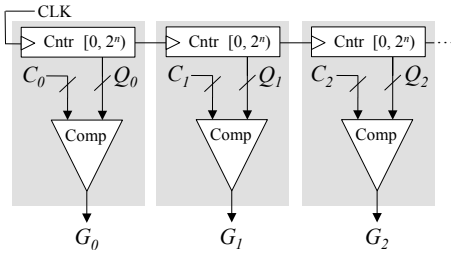


Fig. 5.14: Circuit implementation of the "clock division" method

Independence between bit streams is maintained because the remainder, equal to the overlap between bit streams, always results in a new rotation (or initial phase) of stream. Intuitively, this occurs because the bit lengths share no common factors. This results in every bit of each operand seeing every bit of the other operand. For example, a_0 sees b_0, b_1, and b_2; b_0 sees a_0, a_3, a_2, and a_1; and so on. Using two bit streams with relatively prime bit lengths j and k, the output of a logic gate repeats with period jk. This means that, with multi-level circuits, the output of the logic gates will also be relatively prime. This allows for the same arithmetic logic as a stochastic representation.

A circuit implementation of the "relatively-prime" method is shown in Figure 5.10. Each converter module uses a counter as a number source for iterating through each bit of the stream. The state of the counter Q_i is compared with the stream constant C_i. The relatively prime counter ranges R_i between modules maintain independence. In terms of general circuit components, the circuit uses i counters and i comparators, where i is the number of generated independent bit streams. Assuming the max range is a binary resolution 2^n and all modules are close to this value (i.e., 256, 255, 253, 251...), the circuit contains approximately i n-bit counters and i n-bit comparators.

5.2.5 Rotation

In contrast to the previous method, the "rotation" method allows bit streams of arbitrary length to be used. Instead of relying on relatively prime lengths, the bit streams are explicitly rotated. This requires the sequence generated by the number source to change after it iterates through its entire range. For example, a simple way to generate a bit stream where the stream lengths rotates in time is to inhibit or stall a counter every 2^n clock cycles (where n is the length of the counter). Figure 5.11 demonstrates this method with two bit streams, both of length four.

By rotating bit stream B's length, it is straightforward to see that each bit of one bit stream sees every bit in the other stream. Assuming all streams have the same length, we can extend the example with two bit streams to examples with multiple bit streams; here we would be inhibiting counters at powers of the operand length. This allows the operands to rotate relative to longer bit streams.

A circuit implementation follows from the previous example. We can generate any number of independent bit streams as long as the counter of every ith converter module is inhibited every 2^{ni} clock cycles. This can be managed by adding additional counters between each module. These counters control the phase of each converter module and maintain the property that each converter module rotates relative to the other modules. Using n-bit binary counters and comparators, the circuit requires i n-bit comparators and $2i-1$ n-bit counters. The advantage of using rotation as a method for generating independent bit streams is that we can use operands with the same resolution, but this requires slightly more circuitry than the "relatively-prime" method.

5.2.6 Clock Division

The "clock division" method works by clock dividing operands. Similar to the "rotation" method, it operates on streams of arbitrary lengths. (This method was first seen in Examples 1 and 2 in Section 5.2.1.) Figure 5.13 demonstrates this method with two bit streams, both with bit streams of length four. Bit stream B is clock divided by the length of bit stream A's value.

Assuming all operands have the same length, we can generate an arbitrary number of independent bit streams as long as the counter of every ith converter module increments every 2^{ni} clock cycles. This can be implemented in circuit form by simply chaining the converter module counters together, as shown in Figure 5.14. Using n-bit binary counters and comparators, the circuit requires i n-bit comparators and i n-bit counters. This means the "clock division" method allows operands of the same length to be used with approximately the same hardware complexity as the "relatively-prime" method.

5.2.7 Comparing the Three Deterministic Methods to Stochastic Methods

Table 5.1
Gate count for basic deterministic components, where n is the resolution and i is the number of inputs.

Component	Gate Count
Comparator	$3n$
Counter	$6n$
LFSR	$12ni$

Here we compare the hardware complexity and latency of the deterministic methods with conventional stochastic methods. Perfectly precise computations require the output resolution to be at least equal to the product of the independent input resolutions. For example, with input bit stream lengths of n and m, the precise output contains nm bits.

Consider a stochastic representation implemented with LFSRs. As discussed in Section 5.2.2, a stochastic representation requires bit streams that are 2^{2n} bits long to represent a value with $1/2^n$ precision. In order to ensure that the generated bit streams are sufficiently random and independent, each LFSR must have at least as many states as the required output bit stream. Therefore, to compute with perfect precision each LFSR must have at least length $2ni$.

With our deterministic methods, the resolution n of each of the i inputs is determined by the length of its converter module counter. The output resolution is simply

the product of the counter ranges. For example, with the "clock division" method, each converter module counter is connected in series. The series connection forms a large counter with 2^{ni} states. This shows that output resolution is not determined by the length of each individual number source, but by their concatenation. This allows for a large reduction in circuit area compared to stochastic designs..

Table 5.2
Gate count for stochastic and deterministic bit stream generators, where n is resolution and i is the number of inputs. Latency for each method.

Representation	Method	Gate Count	Latency
Stochastic	Randomizer	$12ni^2 + 3ni$	2^{2ni}
Deterministic	Rel. Prime	$9ni$	2^{ni}
	Rotation	$15ni - 6n$	
	Clock Div.	$9ni$	

To compare the area of the circuits, we assume three gates for every cell of a comparator and six gates for each flip-flop of a counter or LFSR (this is similar to the hardware complexity used in [29] in terms of fanin-two NAND gates). For i inputs with n-bit binary resolution, the gate count for each basic component is given by Table 5.1. Table 5.2 gives the total gate count and bit stream length for precise computations in terms of independent inputs i with resolution n for prior stochastic methods as well as the deteterminstic methods that we propose here. The basic component totals for each deterministic method were discussed in Section 5.2.3. For stochastic methods, we assume that each "Randomizer Unit"' needs one comparator and one LFSR per input.

The equations of Table 5.2 show that our deterministic methods use less area and compute to the same precision, in exponentially less time. It is a win on both metrics, but the reduction in latency is especially compelling. Consider a reduction in latency from $1/2^{20} = 1,048,576$, to just $1/2^{10} = 1,024$!.

5.3 An Analog Approach

Building on the insight that stochastic computation can be implemented deterministically, we explore computation on "Pulse-Width Modulated" (PWM) signals. We encode values as the fraction of the time that the signal is in the high (on) compared to the low (off) state in each cycle. An example was shown in Figure 5.2 in the introduction.

As we will show, the key is choosing different periods for the PWM signals, and letting the system run over multiple cycles. If we choose relatively prime periods and run the signals to their common multiple, we achieve the effect of "convolving"

the signals. This is analogous to the approach that we took with deterministic digital bit streams in Section 5.2.4, where we used relatively prime bit stream lengths.

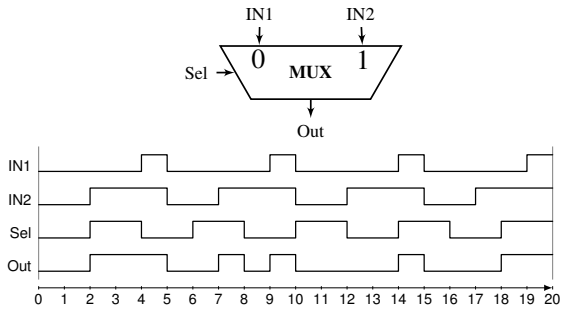


Fig. 5.15: An example of the scaled addition of two PWM signals using a MUX. Here IN1 and IN2 represent 0.2 and 0.6 with a period of 5ns. Sel represents 0.5 with a period of 4ns. The output signal from t=0ns to 20ns represents 0.40 (8ns/20ns=4/10), the expected value from the scaled addition of the inputs.

Figure 5.4 in the introduction showed an example of multiplication on PWM signals. Here we show an example of addition. Recall that with stochastic logic, scaled addition can be performed with a multiplexer (MUX). The performance of a MUX as a stochastic scaled adder/subtracter is insensitive to the correlation between its inputs. This is because only one input is connected to the output at a time [24]. Thus, highly overlapped inputs like PWM signals with the same frequency can be connected to the inputs of a MUX. The important point when performing scaled addition and subtraction with a MUX on PWM signals is that the period of the select signal should be relatively prime to the period of the input signals.

Figure 5.15 shows an example of scaled addition on two numbers, 0.2 and 0.6, represented by two PWM signals. Both have periods of 5ns. A PWM signal with a duty cycle of 50% and period of 4ns is connected to the select input of the MUX. As shown, after performing the operation for 20ns, the fraction of the total time the output signal is high equals the expected value, 0.40.

5.4 Conclusion

While it is easy conceptually to understand how stochastic computation works, randomness is costly. This chapter argues that randomness is not necessary. Instead of relying upon statistical sampling to operate on bit streams, we can explicitly "convolve" them: we slide one operand past the other, performing bitwise operations. We argued that the logic to perform this convolution is less costly than that to generate pseudorandom bit streams. More importantly, we can use much shorter bit streams to achieve the same accuracy as with statistical sampling through randomness. In-

deed, the results of our computation are predictable and completely accurate for all input values.

Of course, compared to a binary radix representation, our deterministic representation is still 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, we require a stream of 2^M bits. In contrast, a stochastic representation requires 2^{2M} bits to achieve the same precision!

We conclude that there is no clear reason to compute on stochastic bit streams. Even when randomness is free, say harvested from thermal noise or some other physical source, stochastic computing entails very high latency. In contrast, computation on deterministic uniform bit streams is less costly, has much lower latency, and is completely accurate.

We do note that there is one drawback to the approach: bit stream lengths grow with each level of logic. This is, in fact, a mathematical requirement. Consider the multiplication of two numbers, each encoded with a precision of n binary bits. Regardless of the encoding, the precision of the result must be greater than the precision of the two operands: up to n^2 bits are required. Stochastic encodings have the same requirement. However, with randomness it is easy to approximate the result, by simply truncating the length of the streams. Accordingly, most stochastic circuits keep constant bit stream lengths regardless of the levels of logic. We concede that there is no straight-forward way to optimally truncate the results of our deterministic computation. See [30] for a discussion of this topic.

This chapter also presented an alternated view of deterministic computation. Instead of streams of digital bits, we can encode data as periodic pulses, with the value represented by the fraction of the duty cycle of each pulse. We are still representing data digitally, though not by an encoding in space, but rather through an encoding in time. With data represented this way, we can use the same theory and all the same constructs developed for stochastic computing on these deterministic, periodic signals.

This time-based approach is motivated by the observation that, as technology has scaled and device sizes have gotten smaller, the supply voltages have dropped while the device speeds have improved. Control of the dynamic range in the voltage domain is limited; however, control of the length of pulses in the time domain can be precise. Given how precisely values can be encoded in the time, the method could produce designs that are much faster than conventional ones – operating in the terahertz range. This remains a work in progress. Potentially, this paradigm could deliver circuits that are as efficient in terms of area and power as stochastic circuits, with considerably lower latency.

References

1. M. H. Najafi, P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. Riedel, "A reconfigurable architecture with sequential logic-based stochastic computing," *J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 4, pp. 57:1–57:28, June 2017. [Online]. Available: http://doi.acm.org/10.1145/3060537
2. E. Friedman, "Clock distribution networks in synchronous digital integrated circuits," *Proceedings of the IEEE*, vol. 89, no. 5, pp. 665–692, May 2001.
3. Y. Jiang, H. Zhang, H. Zhang, H. Liu, X. Song, M. Gu, and J. Sun, "Design of mixed synchronous/asynchronous systems with multiple clocks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014.
4. A. Alaghi and J. P. Hayes, "Survey of stochastic computing," *ACM Transaction on Embedded Computing*, vol. 12, 2013.
5. W. Qian, "Digital yet deliberately random: Synthesizing logical computation on stochastic bit streams," Ph.D. dissertation, University of Minnesota, 2011.
6. N. C. Laurenciu and S. D. Cotofana, "Low cost and energy, thermal noise driven, probability modulated random number generator," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, May 2015, pp. 2724–2727.
7. S. Balatti, S. Ambrogio, R. Carboni, V. Milo, Z. Wang, A. Calderoni, N. Ramaswamy, and D. Ielmini, "Physical unbiased generation of random numbers with coupled resistive switching devices," *IEEE Transactions on Electron Devices*, vol. 63, no. 5, pp. 2029–2035, May 2016.
8. N. Rangarajan, A. Parthasarthy, N. Kani, and S. Rakheja, "Voltage-tunable stochastic computing with magnetic bits," in *2017 IEEE International Magnetics Conference (INTERMAG)*, April 2017, pp. 1–2.
9. R. D'Angelo and S. Sonkusale, "Analogue multiplier using passive circuits and digital primitives with time-mode signal representation," *Electronics letters*, vol. 51, pp. 1754–1755, 2015.
10. V. Ravinuthula, V. Garg, J. G. Harris, and J. Fortes, "Time-mode circuits for analog computation," *International Journal of Circuit Theory and Applications*, vol. 37, pp. 631–659, 2009.
11. K. M. Mhaidat, M. A. Jabri, and D. W. Hammerstrom, "Representation, methods, and circuits for time-based conversion and computation," *International Journal of Circuit Theory and Applications*, vol. 39, pp. 299–311, 2011.
12. Y. P. Tsividis and J. O. Voorman, Eds., *Integrated Continous-Time Filters: Principles, Design and Applications*. iEEE Press, 1993.
13. M. H. Najafi and D. J. Lilja, "High-speed stochastic circuits using synchronous analog pulses," in *ASP-DAC 2017, 22nd Asia and South Pacific Design Automation Conference*, 2017.
14. M. H. Najafi, S. Jamali-Zavareh, D. J. Lilja, M. D. Riedel, K. Bazargan, and R. Harjani, "Time-encoded values for highly efficient stochastic circuits," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 5, pp. 1644–1657, May 2017.
15. K. Cushon, C. Leroux, S. Hemati, S. Mannor, and W. J. Gross, "A min-sum iterative decoder based on pulsewidth message encoding," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 57, no. 11, pp. 893–897, Nov 2010.
16. I. T. R. for Semiconductors (ITRS), "Itrs 2.0," 2015.
17. G. W. Roberts and M. Ali-Bakhshian, "A brief introduction to time-to-digital and digital-to-time converters," *IEEE Transactions on Circuits and System-II*, vol. 57, no. 3, pp. 153–157, January 2010.
18. P. Li, D. Lilja, W. Qian, K. Bazaragan, and M. D. Riedel, "The synthesis of complex arithmetic computation on stochastic bit streams using sequential logic," in *International Conference on Computer-Aided Design*, 2012, pp. 480–487.
19. 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*, vol. 60, no. 1, pp. 93–105, 2011.
20. W. Qian and M. D. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Design Automation Conference*, 2008, pp. 648–653.

21. 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*, 2009, pp. 367–374.
22. W. Qian, M. D. Riedel, H. Zhou, and J. Bruck, "Transforming probabilities with combinational logic," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (to appear)*, 2011.
23. W. Qian, C. Wang, P. Li, D. Lilja, K. Bazaragan, and M. D. Riedel, "An efficient implementation of numerical integration using logical computation on stochastic bit streams," in *International Conference on Computer-Aided Design*, 2012, pp. 156–162.
24. A. Alaghi and J. P. Hayes, "On the functions realized by stochastic computing circuits," in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI*, ser. GLSVLSI '15. New York, NY, USA: ACM, 2015, pp. 331–336. [Online]. Available: http://doi.acm.org/10.1145/2742060.2743758
25. S. S. Tehrani, A. Naderi, G.-A. Kamendje, S. Hemati, S. Mannor, and W. J. Gross, "Majority-based tracking forecast memories for stochastic ldpc decoding," *IEEE Transactions on Signal Processing*, vol. 58, pp. 4883–4896, 2010.
26. B. Gaines, "Stochastic computing systems," in *Advances in Information Systems Science*. Plenum Press, 1969, vol. 2, ch. 2, pp. 37–172.
27. B. Brown and H. Card, "Stochastic neural computation I: Computational elements," *IEEE Transactions on Computers*, vol. 50, no. 9, pp. 891–905, 2001.
28. P. Li, D. J. Lilja, W. Qian, K. Bazargan, and M. D. Riedel, "Case studies of logical computation on stochastic bit streams," in *Lecture Notes in Computer Science: Proceedings of Power and Timing Modeling, Optimization and Simulation Workshop*, G. Goos, J. Hartmanis, and J. Leeuwen, Eds. Springer, 2012.
29. P. Li, W. Qian, and D. J. Lilja, "A stochastic reconfigurable architecture for fault-tolerant computation with sequential logic," *IEEE 30th International Conference on Computer Design (ICCD)*, 2012.
30. M. H. Najafi and D. Lilja, "High quality down-sampling for deterministic approaches to stochastic computing," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2018.