A Survey of Computation-Driven Data Encoding

Weikang Qian[◇#], Runsheng Wang[†#], Yuan Wang[†], Marc Riedel[‡*], Ru Huang[†*]
[◇]UM-SJTU Joint Inst. and MoE Key Lab of Artificial Intelligence, Shanghai Jiao Tong Univ., China
[†]Inst. of Microelectronics and MoE Key Lab of Microelectronics Devices and Circuits, Peking Univ., China
[‡]Dept. of Electrical and Computer Engineering, Univ. of Minnesota, U.S.A.
Email: [◇]qianwk@sjtu.edu.cn, [†]{r.wang, wangyuan, ruhuang}@pku.edu.cn, [‡]mriedel@umn.edu
[#]first authors; [*]corresponding authors

***Abstract*—Although the metal-oxide-semiconductor field-effect transistor (MOSFET) has been the dominant device for modern very-large scale integration (VLSI) circuits for more than six decades, with the dawning of a post-Moore era, researchers are trying to find replacements. A foundation of modern digital computing is the encoding of digital values through a binary radix representation. However, as we enter into the post-Moore era, the challenges of increasing power density, signal noise, and device unreliability raise the question of whether this basic way of encoding data is still the best choice, particularly with novel electronic devices. Prior work has shown that binary radix encoding has some disadvantages. We argue that it is crucial to rethink the necessity of using this representation in the post-Moore era. In this paper, we review some recent development on computation-driven data encoding. We begin with stochastic encoding, a representation proposed a long time ago, discussing both its advantages and disadvantages. Then, we review several recent breakthroughs with variations of stochastic encoding that mitigate many of its disadvantages. Finally, we conclude the paper by extrapolating future directions for effective computation-driven data encoding.**

***Index Terms*—stochastic computing, stochastic encoding, deterministic unary encoding, low discrepancy stochastic encoding**

I. Introduction

Since the demonstration of the first MOSFET in 1947, it has been the fundamental building block of modern VLSI chips. The scale of the transistor has been shrinking in a trend known as Moore's law for more than 50 years. In recent years, the power density of chips has increased to the point where further scaling is very challenging and might become a physical impossibility. For a post-Moore era, researchers are actively seeking replacements for MOSFETs. Many new devices have been proposed, such as carbon nanotube field-effect transistors (CNFETs) [1], negative capacitance transistors [2], memristors [3], etc.

Through the history of the semiconductor industry, a rigid foundation has been the way data is encoded for computation. Since the first successful release of microprocessor by Intel in 1971, binary radix encoding has dominated the design of arithmetic circuits. Binary radix is a type of deterministic, positional encoding. As shown in Fig. 1, different bit positions have different weights. Its key advantage is the encoding efficiency: with n bits, we can encode all integers in the range $[0, 2^n - 1]$. All students of computer engineering learn how to build basic arithmetic circuits, such as adders and multipliers, based on this encoding. The arithmetic-logic units (ALUs) of all modern computing systems are predicated on it.

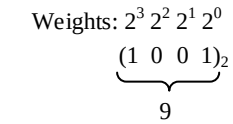

Figure 1: An example of binary radix encoding.

However, as we enter into the post-Moore era, the challenges of the increasing power density, signal noise, and device unreliability raise the question of whether this encoding is still the best choice. Prior research has shown that binary radix has some significant drawbacks [4]. One is that most arithmetic circuits require a large area. Multiplication is an example. When we design a binary multiplier, we need to multiply each bit of the multiplier with the multiplicand and then add all the partial products together. This results in a large area overhead, stemming from the weighted nature of the encoding. Although the partial products obtained from the bit-wise AND are encoded in binary radix form, they need to be further added together to produce the final result. This step could be viewed as "re-encoding" the result into the binary radix form. It accounts for most of the hardware cost of a binary multiplier. Another drawback is the weak error tolerance. With binary radix encoding, the most significant bit has the largest weight. If, due to a noisy environment or an unreliable device, this bit gets flipped, this will result in a large error in the encoded value.

To mitigate the above issues, we argue that it is crucial to rethink the necessity of using a binary radix encoding to represent data. In recent years, there has been growing interest in revisiting previously proposed alternative data encoding methods and proposing new ones for arithmetic computation. In this article, we review some recent developments and extrapolate from them. A particular encoding that was proposed a long time ago that has seen a resurgence of interest recently is stochastic encoding. We first give an overview of stochastic encoding in Section II. With several drawbacks of stochastic encoding recognized, variations of it have been proposed recently. We review these in Section III. Finally, in Section IV, we conclude the paper by discussing possible future directions for developing truly effective computation-driven data encoding.

II. Stochastic Encoding

An encoding method that has found renewed interest recently is *stochastic encoding* [4]–[6]. It was first proposed

in the 1960s [7], [8]. Like binary radix, 0s and 1s are used to represent data. However, the positions of 0s and 1s are random and the data is encoded through the ratio of 1s to the total number of bits in the sequence. As shown in Fig. 2(a), a random sequence with three 1s out of eight bits encodes the value $3/8$. This type of encoding can only represent values in the range $[0, 1]$. It is known as *unipolar format encoding*. Another form is *bipolar format encoding*, in which the ratio p of 1s to the total number of bits encodes the value $(2p-1)$. Thus, the range of the representation is $[-1, 1]$. Stochastic bit sequences can either be generated serially in time as a bit streams (see Fig. 2(a)) or parallel in space as a bit bundles (see Fig. 2(b)) [9]. In what follows, we mainly focus on the serial streaming form and refer to these as *stochastic bit streams*. Unless otherwise specified, we use the unipolar format.

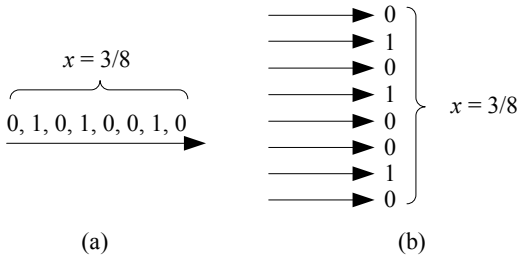

Figure 2: Stochastic encoding: (a) A stochastic bit stream; (b) A stochastic bit bundle.

A. Advantages of Stochastic Encoding

Stochastic encoding has the following advantages.

1) Enabling the design of simple circuits to realizes complex arithmetic functions. A widely-used example is multiplication in the unipolar format, implemented by a single AND gate, as shown in Fig. 3. Assuming that the two input stochastic bit streams are independent, it is easy to see that the probability of obtaining a 1 at the output of the AND gate equals the product of the probabilities of obtaining 1s at the two inputs of the gate. The hardware cost is far less than that of the binary multiplier. In the bipolar format, multiplication is implemented with a single XNOR gate.

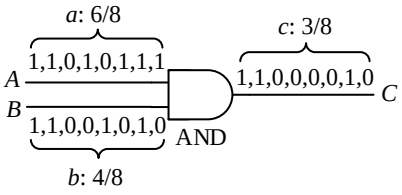

Figure 3: An AND gate realizes multiplication for stochastic encoding.

2) Strong fault tolerance. By its encoding nature, every bit in the sequence contributes the same weight to the encoded value. Given a sequence of length N, the weight is $1/N$. Thus, if a bit flip occurs, the change to the final encoded value is small and independent of the location of the bit flip. This is in contrast to binary radix encoding, in which bits at a higher positions have larger weights, so a flip in these causes a larger error.
3) Tunable precision without hardware change. In the form of serial streaming, the same hardware can be used for any data precision the user requires. To increase or decrease the precision, only the stream length needs to be increased or decreased without changing the circuit. In contrast, for binary radix-based design, when the data precision is changed, the circuit must be redesigned.
4) Progressive precision. This property means that the first few bits of a stochastic bit stream can yield a rough approximation of the final number [10]. For example, the bit stream 0110111001011001 encodes the value $9/16$. The first 2, 4, and 8 bits correspond to the bit streams 01, 0110, and 01101110, respectively. They encode the values $1/2$, $1/2$, and $5/8$, respectively, which are good approximation of the actual value $9/16$. This property can sometimes be exploited to use a short bit stream to represent the final value.

B. Disadvantages of Stochastic Encoding

While the advantages are compelling, stochastic encoding has several distinct disadvantages.

1) The low encoding efficiency. For stochastic encoding to represent any integer in the range $[0, 2^n - 1]$, it requires the sequence length to be $2^n - 1$. This is in contrast to the binary radix encoding, which only requires n bits. With serial bit streams, the computation time is proportional to the stream length; this results in long computation time.
2) Stochastic variation. A stochastic bit stream is usually generated with a module called *stochastic number generator (SNG)*. A typical SNG design is shown in Fig. 4. Each bit in the stream is the comparison result between a k-bit uniformly distributed random number R and a k-bit constant C. The bit is 1 if $R < C$ and 0 otherwise. It is easy to see that each bit in the stream has the probability of $C/2^k$ to be a 1. To produce a bit stream for a value $p \in [0, 1]$, we just choose the constant $C = 2^k p$. Based on this mechanism, we can see that a stochastic bit stream only ensures that each bit in the stream has the probability of p to be a 1. However, for any generated bit stream of length N, it likely will not have *exactly* pN 1s. Randomness is an inherent error source with stochastic encoding.

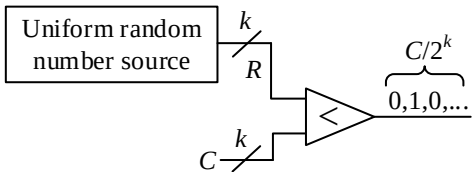

Figure 4: Stochastic number generator.

3) Precision loss when doing addition. A basic stochastic encoding can only represent values in the range of $[0, 1]$ in the unipolar format or $[-1, 1]$ in the bipolar format. This causes a problem when implementing addition. That is, addition $c = a + b$ cannot be exactly realized. This

is because given a and b within the valid representation range of stochastic encoding, it is possible that their sum is outside the range and hence, there is no way to calculate that sum. Thus, with stochastic encoding, addition is typically implemented in a scaled version by using a multiplexer (MUX) as shown in Fig. 5. We typically choose the selecting signal so that it has a probability of $1/2$ to be a 1. In this case, the output of the MUX is the scaled sum $c = \frac{1}{2}(a + b)$. Due to the scaling, the addition with stochastic encoding entails precision loss. The precision of the input is $1/N$, where N is the bit stream length. However, the precision of the output is $2/N$ since we need to multiply the final result by 2 to obtain $a + b$.

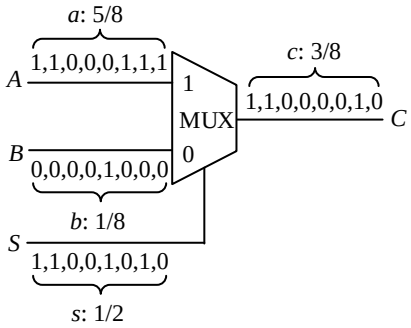

Figure 5: A multiplexer realizes a scaled addition for stochastic encoding.

III. Variations of Stochastic Encoding

In this section, we review several recently proposed variations of stochastic encoding. These address some of the disadvantages discussed in the previous section.

A. Deterministic Unary Encoding

As we mentioned in Section II-B, one disadvantage of stochastic encoding is that it suffers from error caused by stochastic variation. To overcome this drawback, a *deterministic unary encoding* was proposed [11]. An example is shown in Fig. 6(a). Such an encoding is same as stochastic encoding in that every bit has the same weight contribution to the final value, which equals $1/N$, where N is the length of the bit stream. However, the key difference lies in the determinism in generating the bit streams. If we want to encode a value $p \in [0, 1]$ with a stream of length N, as shown in Fig. 6(a), the stream is composed of pN 1s followed by $(1 - p)N$ 0s. In this case, the ratio of 1s exactly equals the target value to be encoded. Such a special stream could be easily generated by replacing the uniform random number source in Fig. 4 by a digital counter.

Since the deterministic unary encoding is same as stochastic encoding in that each bit has the same weight contribution, it is possible to use the same computing circuit as stochastic encoding, e.g., using an AND gate for multiplication. However, directly using two deterministic bit streams may not produce the correct result. An example is shown in Fig. 6(b). In this example, the output is $2/4$, which does not equal the product of the two input values. In order to ensure the correct computation, the inputs should maintain the property that every bit of one stream should be paired with every bit of the other exactly once. This essentially requires the repeat of the bits in each stream in some way. An example is shown in Fig. 6(c): each a_i $(0 \le i \le 3)$ is paired with each b_j $(0 \le j \le 3)$ exactly once. To generate these streams, three methods are proposed in [11], which are using relatively prime bit length, rotation, and clock division (hereafter referred to as *deterministic manipulation methods*). The streams in Fig. 6(c) are produced using the clock division method, which divides the frequency of the bit stream a by 4 to obtain the frequency of the bit stream b. Using a pair of streams of the form shown in Fig. 6(c), an AND gate can correctly realize multiplication, as shown in Fig. 6(d).

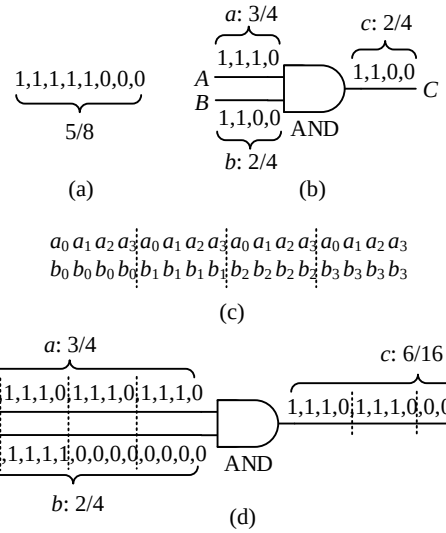

Figure 6: Deterministic unary encoding: (a) basic encoding form, which has all 1s appear before all 0s; (b) an improper use of deterministic unary encoding causing a wrong multiplication result; (c) bit streams generated by the clock division method so that every bit in one stream is paired with every bit of the other exactly once; (d) deterministic unary encoding for correct multiplication by the clock division method.

Recent work has further developed the concept of deterministic unary encoding. Najafi *et al.* proposed using analog pulse width modulated (PWM) signals [12]. The value encoded by the deterministic unary encoding is the fraction of the time the signal is high. Thus, PWM signals encode value through their duty cycles. In order to use these PWM signals to perform correct computation, the key is to choose the correct clock cycles (or frequencies) of the input PWM signals. For example, it is obvious that for multiplication, if the frequencies of the two input signals are the same, an AND gate will not correctly implement multiplication. The authors described a few guidelines for the choice of the clock cycles. One basic guideline is that the clock cycles should be relatively prime, or *inharmonic* using signal processing terminology. Another important guideline is that, to achieve the best accuracy, the operation should be run for the common multiple of the input periods. With the use of analog signal, one clock cycle

corresponds to many clock cycles in the deterministic unary encoding and hence, the computation time, together with the energy consumption, is significantly reduced.

Najafi and Lilja noted that the deterministic unary encoding does not possess the progressive precision property of stochastic encoding. To address this problem, they proposed to combine pseudo-randomized sequence with the deterministic manipulation methods [13].

Recent works considered deterministic unary encoding represented as parallel bundles, a form called *thermometer coding* [14]. Mohajer *et al.* proposed a design method that performs proper routing to implement monotonically increasing functions with thermometer coding [14]. Non-monotonic functions can be implemented with additional XOR gates. They showed that the proposed design has a much smaller area-delay-product than the conventional binary design and stochastic design. Zhang *et al.* exploited the regular pattern of thermometer coding and proposed an efficient generator for it, which can significantly reduce the energy consumption compared to the traditional bit stream generators [15].

B. Low Discrepancy Stochastic Encoding

Although stochastic encoding has the progressive precision property, when the bit stream is short, the difference between the value encoded by the bit stream and the accurate value could still be very large. This is due to the large random fluctuation of the pure random sequence. To improve the progressive precision, *low-discrepancy (LD) bit streams*, in which the 0s and 1s are more uniformly spaced, were introduced to replace the conventional stochastic bit streams [16]. The LD bit streams can be generated by comparing each number in an *LD sequence* to a fixed constant value. For example, the upper part of Fig. 7(a) shows a type of LD sequence called Sobol sequence. It has the property that for any $0 \leq m \leq n$, the first 2^m numbers include all values in the set $\{0, 1 \cdot 2^{n-m}, 2 \cdot 2^{n-m}, \ldots, (2^m - 1) \cdot 2^{n-m}\}$, where n is the number of bits in the binary encoding of the values [17]. If the target value is $1/2$, then we compare each number in the sequence with the constant $\frac{1}{2} \cdot 2^n = 4$. The resulting bit stream is shown in the lower part of Fig. 7(a). As we can see, the first 2, 4, and 8 bits all encode the correct value $1/2$. For comparison, the upper part of Fig. 7(b) shows a pseudo-random sequence generated by a linear-feedback shift register. In order to generate the same target value $1/2$, each number in the sequence is also compared with the value 4. The resulting bit stream is shown in the lower part of Fig. 7(b). Now, the first 2, 4, and 8 bits encode the values 1, $3/4$, and $1/2$, respectively. Obviously, its progressive procession property is far worse than the LD bit stream.

There are many different types of LD sequences, originally developed for quasi-Monte Carlo (QMC) sampling [18]. A few of them have been applied to generate LD bit streams for stochastic encodings. An LD bit stream generator based on Halton LD sequence was proposed in [16]. However, it has a large area overhead. A later work proposed to use Sobol sequence generator [19], which is more area- and power-efficient. The work [17] further proposed to combine the deterministic manipulation methods with Sobol LD bit streams. One particular design considered is to combine the rotation method with the Sobol sequence generator. Given this combination, the resulting design enjoys two benefits: 1) like deterministic encoding, the computation is accurate when the bit stream length equals the product of the input bit stream lengths; 2) the proposed sequence converges to the accurate result much faster than the previous methods, including the method integrating the rotation method with the pseudo-random sequence [13]. Consequently, for a fixed accuracy requirement, the proposed method requires a much shorter bit stream length, which leads to the reduction in energy consumption.

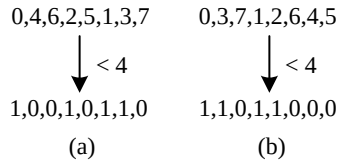

Figure 7: Comparison between low-discrepancy sequence and random sequence. (a) Sobol low-discrepancy sequence and the generated bit stream; (b) A pseudo-random sequence and the generated bit stream.

In [20], another way to efficiently generate LD bit streams was proposed. The idea is to partition the entire bit stream into multiple groups and evenly distribute the 1s among the groups. It combines an even-distribution (ED) encoder with an inter-group randomizer and an intra-group randomizer. The ED encoder makes 1s evenly distributed within all groups, the inter-group randomizer shuffles the order of the groups, and the intra-group randomizer scrambles the bits within a group.

C. Sign-Magnitude Stochastic Encoding

With the unipolar format, stochastic encoding can only represent values in the range $[0, 1]$. Thus, it cannot represent negative values. To solve this problem, the bipolar encoding format was proposed. With this, the representation range becomes $[-1, 1]$. However, with bipolar format, the precision reduces to half of that of the unipolar format: A stochastic bit stream of length N can only represent values of precision $2/N$ in bipolar format.

Zhakatayev *et al.* recently proposed a variation of stochastic encoding called *sign-magnitude stochastic encoding* [21]. It encodes a signed value with a sequence of N bits. The first bit is the sign bit, which represents a positive (resp. negative) value when it is 0 (resp. 1), and the rest $(N - 1)$ bits encode the magnitude just like stochastic encoding. For example, the bit stream shown in Fig. 8 encodes the value $-3/4$. The multiplication on the new encoding applies an XOR gate to the sign bits of the two input sequences to produce the sign bit of the output sequence. It applies AND gates to handle the rest bits in the two input sequences to produce the magnitude part of the output sequence. This encoding method allows the representation of negative values with almost the same precision as the unipolar-format stochastic encoding.

D. Stochastic Encoding with Infinite Range

As we mentioned in Section II-B, one drawback of stochastic encoding is the precision loss when doing addition. The

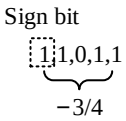

Figure 8: An example of sign-magnitude stochastic encoding.

essential reason is because the encoding only allows us to represent values in a limited range. This issue was considered during the early days of stochastic computing and several variations of stochastic encoding that could represent values with infinite range were proposed. One representation is the single-line unipolar stochastic encoding with infinite range. In this representation, a random sequence with the ratio of 1s as p and the ratio of 0s as $(1-p)$ encodes the value $p/(1-p)$. Thus, any value in the range $[0, +\infty]$ could be encoded. Under this encoding, normal addition, multiplication, and division could be implemented by circuits based on JK flip-flops [8]. The same representation was rediscovered later by Min *et al.* [22].

Another possible representation is a two-line stochastic encoding [8]. Each line represents a value in the bipolar format and the final encoded value is the ratio between X and Y, where X and Y are the bipolar numbers encoded by the upper and lower lines, respectively. An example of such a encoding is shown in Fig. 9(a). It can be seen that the range of the values that could be encoded now becomes $[-\infty, +\infty]$. The same representation was rediscovered later by Canals *et al.* [23], under the name *extended stochastic logic*. Under this encoding, normal addition could be implemented with the circuit shown in Fig. 9(b). Indeed, the value of the output is

$$\frac{t}{u} = \frac{\frac{1}{2}(ps + qr)}{\frac{1}{2}qs} = \frac{p}{q} + \frac{r}{s},$$

which is the sum of the two inputs. Also, multiplication and division could be implemented easily, as shown in Figs. 9(c) and (d), respectively. Each of them only uses two XNOR gates.

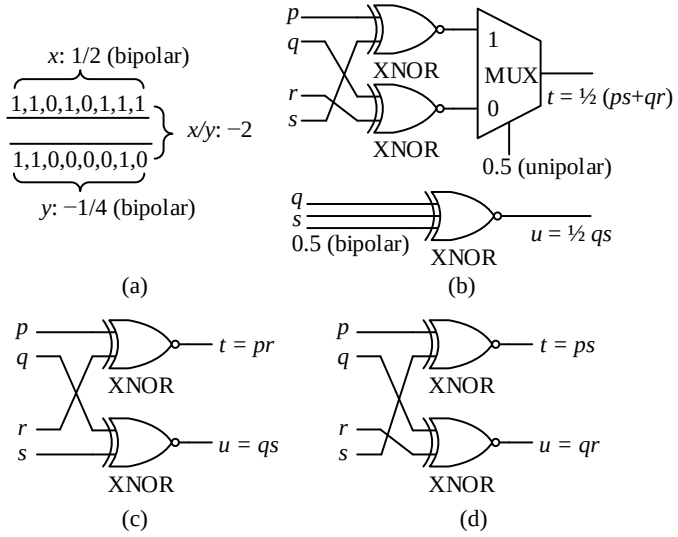

Figure 9: A two-line stochastic encoding. (a) an illustration of the encoding; (b) the addition circuit; (c) the multiplication circuit; (d) the division circuit.

E. Integral Stochastic Encoding

As we mentioned in Section II-B, one fundamental drawback of stochastic encoding is its low encoding efficiency and hence, the long runtime. Besides this, addition with stochastic encoding results in precision loss. Ardakani *et al.* proposed an *integral stochastic encoding* to address these two problems jointly [24]. It can be viewed as a mix of binary radix encoding and stochastic encoding. An example is shown in Fig. 10(a). One still uses a sequence of random numbers to encode a value. However, the difference is that each number in the sequence is not a binary 0-1 value, but an integral value in the range $[0, m]$. The encoding essentially uses multiple wires to represent the integral value through binary radix encoding. With this, the encoding efficiency is improved. For example, with $m = 3$, a sequence of length 5 is enough to represent any value in the range $[0, 15]$. In contrast, using stochastic encoding, the length should be 15, which is much longer. Another benefit of this encoding is that the range of the values that can be encoded is not limited. Thus, we could implement exact addition. The addition of the two values represented by the integral stochastic form can be realized by a binary radix adder that performs number-wise addition, as shown in Fig. 10(b). Multiplication requires a binary multiplier, which seems to nullify the benefit of reduced hardware cost brought by the stochastic encoding. However, if one input sequence to the multiplier can be fixed as the binary stochastic bit stream, then the multiplication could still be implemented quite efficiently with several AND gates.

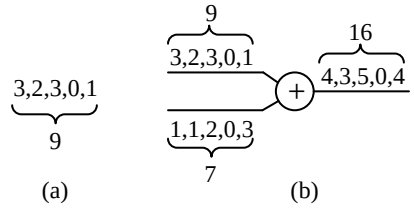

Figure 10: Integral stochastic encoding. (a) an illustration of the encoding; (b) the addition circuit.

Although the hardware cost of integral stochastic encoding is larger than that of the basic stochastic encoding, the latency is significantly reduced. As a result, the total energy consumption is smaller than that of the basic stochastic encoding [24].

IV. Conclusion

In this paper, we reviewed some recent advances in computation-driven data encoding. Binary radix encoding has dominated traditional arithmetic circuit design since the early days of the semiconductor industry. However, in the post-Moore era, some disadvantages of this encoding are manifest, especially its weak fault tolerance. Stochastic encoding has been the subject of much research for post-Moore computing. Stochastic encoding and binary radix encoding can be viewed as two extremes of the encoding spectrum. On the one end, binary radix encoding is positional, deterministic, and compact. On the other end, stochastic encoding is uniformly-weighted, random, and not compact.

Stochastic encoding permits complex operations to be performed with very simple logic. While this is compelling for many applications, there are significant disadvantages. Recent research has proposed variations on the encoding that mitigate many of these disadvantages. A trend seems to be to hybrid encodings, with some positional aspects and some stochastic aspects. For example, deterministic unary encoding combines the uniform weight property of stochastic encoding with the deterministic property of binary radix encoding. Integral stochastic encoding extends the stochastic encoding by replacing each bit in the sequence with a binary-radix encoded integral value.

Despite these recent advances, new computation-driven data encoding methods are still in an early exploration stage. We believe that in the post-Moore era, research on these topics has considerable merit. It may provide alternative solutions to the challenges facing the semiconductor industry, enabling innovations at the device and architecture levels. From the research work surveyed in this article, we can frame questions about future directions:

1) The representation range. Should negative values be considered? Should the encoding only represent value within a limited ranged? The bipolar-format stochastic encoding and the sign-magnitude stochastic encoding both try to extend the representation range by including negative values. The two-line stochastic encoding with infinite range further enlarges the representation range to $[-\infty, +\infty]$.
2) The hardware cost of designing arithmetic circuit. The encoding essentially needs to support efficient hardware design. Ideally, the hardware should has small area, computation time, power consumption, and energy consumption. Stochastic encoding delivers small area and power consumption. However, due to the low encoding efficiency, its needs a long computation time and hence results in high energy consumption. To overcome these issues, low-discrepancy bit streams are proposed to accelerate the convergence speed. Integral stochastic encoding is proposed to shorten the bit stream length at the cost of a slightly increased area.
3) Error tolerance. In the post-Moore era, error tolerance is becoming crucial. On the one hand, for Internet-of-Things applications, the supply voltage may be significantly reduced to minimize power consumption, which at the same time significantly affects signal integrity. On the other hand, some emerging devices are inherently unreliable due to their manufacturing processes. Therefore, in the post-Moore era, computation is more and more subject to error. Thus, error tolerance is an important factor that needs to be considered when developing new data encodings.

ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China (NSFC) under grant no. 61472243 and 61204042.

REFERENCES

[1] R. Martel, T. Schmidt *et al.*, "Single- and multi-wall carbon nanotube field-effect transistors," *Applied Physics Letters*, vol. 73, no. 17, pp. 2447–2449, 1998.
[2] J. C. Wong and S. Salahuddin, "Negative capacitance transistors," *Proceedings of the IEEE*, vol. 107, no. 1, pp. 49–62, 2019.
[3] D. Strukov, G. Snider *et al.*, "The missing memristor found," *Nature*, vol. 453, pp. 80–83, 2008.
[4] W. Qian, X. Li *et al.*, "An architecture for fault-tolerant computation with stochastic logic," *IEEE Transactions on Computers*, vol. 60, no. 1, pp. 93–105, 2011.
[5] W. Qian and M. Riedel, "The synthesis of robust polynomial arithmetic with stochastic logic," in *Design Automation Conference*, 2008, pp. 648–653.
[6] A. Alaghi, W. Qian, and J. P. Hayes, "The promise and challenge of stochastic computing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (in press)*, 2017.
[7] W. J. Poppelbaum, A. Dollas, J. B. Glickman, and C. Otoole, "Statistical processors," in *Advances in Computers*, M. C. Yovits, Ed., 1976, vol. 17, pp. 187–230.
[8] B. R. Gaines, "Stochastic computing systems," in *Advances in information systems science*. Springer, 1969, pp. 37–172.
[9] L. Miao and C. Chakrabarti, "A parallel stochastic computing system with improved accuracy," in *International Workshop on Signal Processing Systems*, 2013, pp. 195–200.
[10] A. Alaghi, C. Li, and J. P. Hayes, "Stochastic circuits for real-time image-processing applications," in *Design Automation Conference*, 2013, pp. 136:1–136:6.
[11] D. Jenson and M. Riedel, "A deterministic approach to stochastic computation," in *International Conference on Computer-Aided Design*, 2016, pp. 102:1–102:8.
[12] M. H. Najafi, S. Jamali-Zavareh *et al.*, "Time-encoded values for highly efficient stochastic circuits," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 5, pp. 1644–1657, 2017.
[13] M. H. Najafi and D. Lilja, "High quality down-sampling for deterministic approaches to stochastic computing," *to appear in IEEE Transactions on Emerging Topics in Computing*, 2017.
[14] S. Mohajer, Z. Wang, and K. Bazargan, "Routing magic: Performing computations using routing networks and voting logic on unary encoded data," in *International Symposium on Field-Programmable Gate Arrays*, 2018, pp. 77–86.
[15] Y. Zhang, R. Wang *et al.*, "A parallel bitstream generator for stochastic computing," in *Silicon Nanoelectronics Workshop*, 2019, pp. 1–2.
[16] A. Alaghi and J. P. Hayes, "Fast and accurate computation using stochastic circuits," in *Design, Automation, and Test in Europe*, 2014, pp. 76:1–76:4.
[17] M. H. Najafi, D. Lilja, and M. Riedel, "Deterministic methods for stochastic computing using low-discrepancy sequences," in *International Conference on Computer-Aided Design*, 2018, pp. 51:1–51:8.
[18] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, 1992.
[19] S. Liu and J. Han, "Energy efficient stochastic computing with Sobol sequences," in *Design, Automation, and Test in Europe*, 2017, pp. 650–653.
[20] K. Kim, J. Lee, and K. Choi, "An energy-efficient random number generator for stochastic circuits," in *Asia and South Pacific Design Automation Conference*, 2016, pp. 256–261.
[21] A. Zhakatayev, S. Lee *et al.*, "Sign-magnitude SC: Getting 10x accuracy for free in stochastic computing for deep neural networks," in *Design Automation Conference*, 2018, pp. 158:1–158:6.
[22] S.-J. Min, E.-W. Lee, and S.-I. Chae, "A study on the stochastic computation using the ratio of one pulses and zero pulses," in *International Symposium on Circuits and Systems*, vol. 6, 1994, pp. 471–474.
[23] V. Canals, A. Morro *et al.*, "A new stochastic computing methodology for efficient neural network implementation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 3, pp. 551–564, 2016.
[24] A. Ardakani, F. Leduc-Primeau *et al.*, "VLSI implementation of deep neural network using integral stochastic computing," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 25, no. 10, pp. 2688–2699, 2017.