2009 International Symposium on Nonlinear Theory and its Applications
NOLTA'09, Sapporo, Japan, October 18-21, 2009

Exploiting Noise in Computation

Ferdinand Peper

National Institute of Information and Communications Technology, Nano ICT Group
588-2 Iwaoka, Iwaoka-cho, Nishi-ku, Kobe, 651–2492 Japan
Email: peper@nict.go.jp

Abstract—Noise and Fluctuations are usually considered obstacles in the operation of electronic and mechanical devices, and most strategies to deal with them revolve around suppressing them. This paper focuses on systems that employ different strategies, i.e., strategies that can exploit the properties of noise to improve the efficiency of operations. These strategies may be an important ingredient in the designs of computers with devices of nanometer-scale feature sizes.

1. Introduction

Scientists and engineers are trained to think of noise as an unwelcome phenomenon that needs to be suppressed in order to decrease its ratio to signal levels. Various techniques are used to deal with noise, like Error Control Codes, which encode information redundantly such that its original contents can be recovered in case of data corruption. These traditional techniques, however, have their limitations, especially when signal levels become very low, as is increasingly the case in applications like integrated electronics. When scales are in the range of nanometers, noise takes on a major role, and it is quite remarkable that biological organisms can cope so well with it. Molecular motors, which play an important role in cellular transport and in muscular movements, typically work in an environment full of Brownian motion and still manage to conduct their tasks with a remarkable efficiency. The spiking activity of neurons in the brain is another example in which noise appears to have only limited negative effects on the tasks being conducted. What are the secrets of these natural feats and can engineers use similar mechanisms when designing complex systems? In this paper, we discuss computation methods that aim to exploit noise actively, as an alternative to the usual way of just coping with the deleterious effects of noise.

2. Brownian circuits

Brownian circuits use discrete undividable units (called *tokens*) as signals, which are allowed to fluctuate in random semi-controlled ways within the confines of the circuit topology. There is no clock in Brownian circuits, unlike in traditional logic circuits in which the clock is responsible for making all circuit elements work in lock-step. The lack of a clock is usually indicated by the term *asynchronous timing*. Brownian circuits actively exploit fluctuations of signals, rather than treating them as a nuisance: they use fluctuations as a mechanism to make signals randomly search their way through a circuit from the input side of a circuit along a computational path until the system settles in an output state. Though a random search is regarded as time-inefficient in computational models, it does have the advantage of allowing signals to travel to certain locations in a circuit without there being required an explicit mechanism to accomplish this. Deadlocks, which are common in token-based circuits, are one of the situations that can be avoided or resolved through random search. Without fluctuations, tokens will move in one direction (forward), and the only way out of a deadlock is via additional pathways in a circuit to redirect deadlocked tokens away from their static state. Enter fluctuations, however, and tokens can suddenly backtrack out of deadlocks, without the need for such pathways.

Logic circuits can be constructed from a few primitives, like an AND-gate and a NOT-gate, and this is why these gates are called *universal*. Logic gates only work when signals are synchronized, so the presence of a clock is implicitly assumed. Absent a clock, additional functionality is required to make up for the lack of synchronization functionality. Because of this, the primitives for token-based circuits tend to be more complex than those for logic circuits. Examples of primitives for token-based circuits are the so-called FORK, P-MERGE, and TRIA (see [4]), which together form a universal set. The FORK and the P-MERGE primitive have both three input- or output-wires, but the TRIA has six, which is a substantial increase in complexity over AND-gates and NOT-gates. Surprisingly, it has been found that the presence of fluctuations of tokens in circuits allows a lower number of primitives, each with less wires. In [7] only one primitive is required to achieve universality, but this primitive has still six input or output wires. A different set of primitives for Brownian circuits, which requires two primitives—each though with less wires—is proposed in [3, 5] and discussed in this paper.

The first primitive is the *Hub*, which contains three wires that are bidirectional (Fig. 1). There will be at most one token at a time on any of the Hub's wires, and this token can move to any of the wires due to its fluctuations.

The second primitive is the *Conservative Join (CJoin)*, which has two input wires and two output wires (Fig. 2).

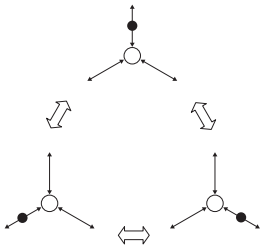

Figure 1: Hub and its transitions. Fluctuations cause a token to move between the Hub's three wires in any order.

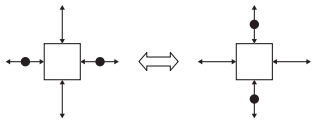

Figure 2: CJoin and its transitions. If there is a token on only one input wire, this token remains pending until a token arrives on the other wire. These two tokens will then result in one token on each of the two output wires. The reverse transition may also take place.

The CJoin can be interpreted as a synchronizer of two tokens passing through it. Tokens may fluctuate on the input wires of a CJoin, but once processed by the CJoin, they will be placed on the output wires, where they will also fluctuate. The operation of the CJoin may also be reversed. When connecting CJoins to each other, we should make sure that input terminals face output terminals. Hubs, having bi-directional wires, may be connected in any way to CJoins or other Hubs.

Fig. 3 gives a circuit constructed from Hubs and CJoins, which acts like a Half-Adder [5]. Input and output to this

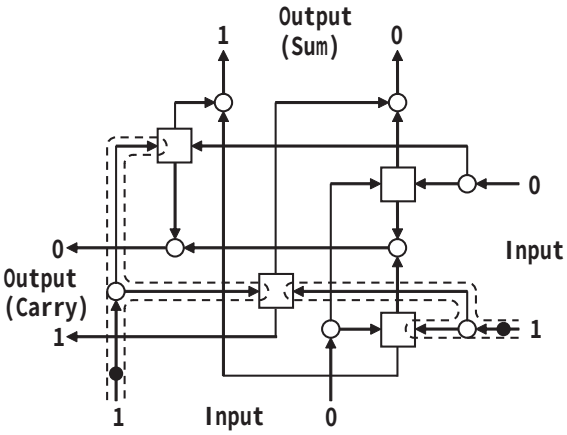

Figure 3: Half-Adder constructed from four CJoins and eight Hubs. The tokens still at the input terminals encode the input signals 1 and 1.

Half-Adder is represented through *dual-rail encoding*, a well-known way to encode information in token-based circuits. Dual-rail encoding represents a binary signal through two wires: the *0-wire*, which encodes the signal 0 when it contains a token, and the *1-wire* doing the same for the signal 1. The absence of tokens on both the 0-wire as the 1-wire at a certain time indicates a spacer between signals. An erroneous situation occurs when there is a token on the 0-wire at the same time as a token on the 1-wire, so this is to be avoided. The input to the Half-Adder in Fig. 3 is 1 at one input and 1 at the other input, and the corresponding tokens on the respective 1-wires are allowed to fluctuate in areas that are indicated by dashed lines. These areas indicate the searching space of the tokens. There is only one CJoin where these two different areas come together, and it is this CJoin that will eventually process the two input tokens when they are present at the same time at the CJoin's input terminal.

After the CJoin has operated on the input tokens, the circuit will be in the state shown in Fig. 4. The tokens can now

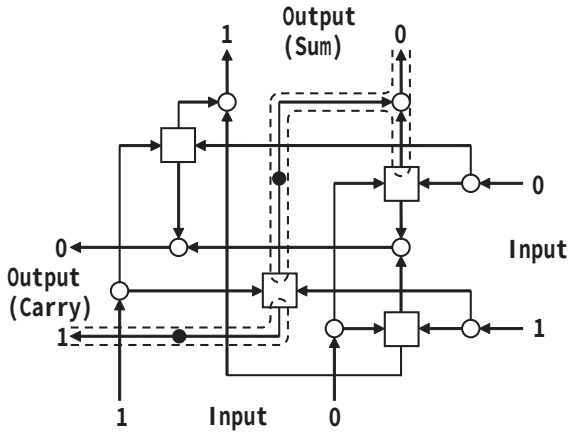

Figure 4: Half-Adder after the two input tokens have been processed by a CJoin.

fluctuate in the post-operation areas indicated by dashed lines, and these areas include the output terminals 0-Sum and 1-Carry of the circuit. It is reasonable to assume that these are the outputs of the circuits, were it not for the continuously fluctuating behavior of the tokens. The CJoin, it must be noted, is reversible in its operations, so the two output tokens can just as well be reverted into the original input state of the circuit. Obviously, this is undesirable behavior for circuits, and in order to make it more in line with conventional circuits, we introduce so-called *Ratchets* (Fig. 5). A ratchet is placed on a wire of a circuit and it

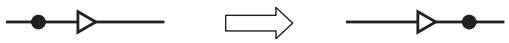

Figure 5: A ratchet is indicated as a triangle on a wire, and its transition. A token can move over the ratchet in its preferred direction, but backwards movement is impossible.

restricts a signal's movement on that wire to one direction (forward). Ratchets are usually placed in a circuit such as to bias signals toward the circuit's output terminals. On wires where no searching behavior is required, ratchets can be freely placed. Fig. 6 shows the Half-Adder circuits with ratchets placed in selected locations. On wires involved in searching behavior of fluctuating tokens ratchets are unwelcome, because they restrict random search to certain directions, thereby jeopardizing the circuit's operation. This is the reason why there are no ratchets in the circuit at the

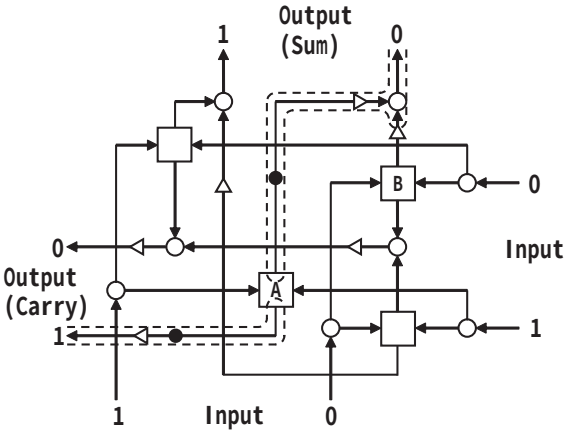

Figure 6: Ratchets placed on wires inside a Half-Adder restrict movements of tokens in the circuit. After the tokens have been processed by CJoin *A*, their fluctuations will be limited as they move over more and more ratchets. Initially CJoin *B* becomes unreachable for the right-most token, due to the ratchet placed at the upper output wire of *B*.

input sides of CJoins.

Once the tokens are near the output terminals, they have hardly any space left to fluctuate (Fig. 7), and it is in this state that we consider the circuit operation finished. As

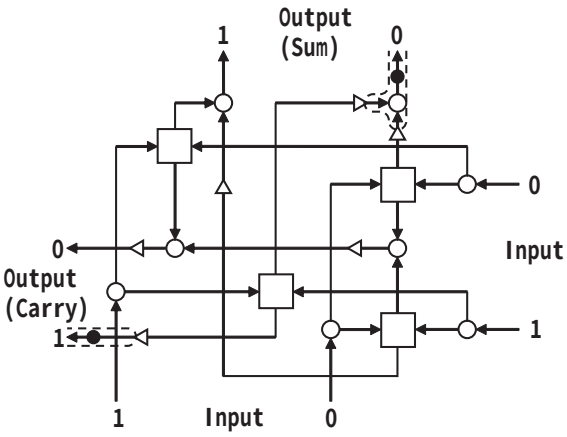

Figure 7: Tokens at output terminal in a ratcheted circuit. The searching space of the tokens has been restricted by the ratchets nearest to the output terminals.

seen above, ratchets offer much potential to speed up the operation of Brownian circuits; they do so by restricting the Brownian search process at selected locations. Another method to speed up operation is to bias tokens away from the centers of wires and toward input and output terminals of CJoins, where the probability of running into a CJoin with a token at its other input terminal is highest. Chaos has been used in [1] with this purpose in mind.

When fluctuations of tokens are inhibited, the tokens become unable to search in the circuit for a CJoin at which they can be matched. Consequently, a deadlock will take place at the input terminal of a CJoin if only one token has arrived at it, because a second token would be needed at the other input terminal to make the CJoin process the tokens. When no fluctuations of signals are allowed, the Hub and CJoin fail to form a universal set of primitives for token-based circuits. Fluctuations thus add functionality to circuits, and because of them primitives require less complex designs to become universal.

The role of fluctuations in simplifying primitives in Brownian circuits also extends to other computational models. The Cellular Automaton (CA) model in [3] achieves computational universality by simulating Brownian circuits on its cell space. This CA updates its cells in an asynchronous way, i.e., by randomly selecting one cell in each step and updating it if the states of itself and of its direct neighbors match the transition rules of the model. Asynchronous updating does not rely on a central clock signal to time the updates of cells, and consequently it tends to require an increased number of cell states and transition rules to deal with all possible orders in which updates can take place. The Brownian CA in [3], being asynchronously timed, should be no different in this regard, but nevertheless it requires only three states for cells and only three transition rules. This is much less than the computationally universal asynchronous CA models discovered thus far. The asynchronous CA in [6], for example, requires 16 cell states and 6 transition rules.

The ability to reduce complexity in models may also be useful in physical implementations of Brownian circuits, which has been the reason to study realizations of Brownian circuits by Single Electron Tunneling (SET) technology. Electrons in such implementations would then be modeled by tokens, and the stochastic nature of tunneling would be compatible with the asynchronous nature of Brownian circuits. Additionally, fluctuations—usually considered a curse in SET technology—would have the potential to be used in a positive way. Tentative steps have been set in this direction through the simulation of SET implementations of Brownian circuits [8], though concrete applications have not yet been demonstrated. An early proposal of the use of fluctuations in SET technology is the simulated annealing process of a Boltzmann machine in [10]. This method utilizes fluctuations to search in an energy landscape, but it is somewhat focused on neural networks, rather than on arithmetic operations.

3. Encoding Signals by Noise

Another way to use noise in computation has recently been proposed in [2]. In this scheme, which is loosely inspired by the stochasticity of brain signals, different levels of signals (like a zero or a one) are encoded through different independent noise sources. Since these noise sources are statistically independent, their correlation, averaged over time, is zero, and this forms the basis for defining these sources as orthogonal. This method resembles *Code*

Division Multiple Access (CDMA) encoding [9], which is used in wireless communication because of its good noise robustness.

The scheme in [2] uses reference signals that are orthogonal; for binary signals we use the signals $H(t)$ and $L(t)$ to encode the 1 and the 0, respectively. A signal input to a circuit, say $X(t)$, is encoded in terms of the reference signals, so, $X(t) = L(t)$ or $X(t) = H(t)$. Multiplying $X(t)$ by a reference signal and averaging the resulting product over time, we obtain a value that is near zero—in case $X(t)$ is different from the reference signal—or much higher—otherwise. This time-average is mapped through the step function S into one of the values 0 and 1, respectively. A binary INVERTER-gate with input $X(t)$ and output $Y(t)$ can then be described by the following equation:

$$Y(t) \;=\; S\,(\,\langle\, X(t)\,H(t)\,\rangle\,)\;L(t) \;+\; S\,(\,\langle\, X(t)\,L(t)\,\rangle\,)\;H(t)$$

The input signal is multiplied by $H(t)$ and $L(t)$ respectively, giving rise to two terms, one of which will map to 0 and the other to 1. These numbers, when multiplied by the reference signals $L(t)$ and $H(t)$, then result in the desired output signal of the gate. This gate can be implemented as a simple analog circuit using an analog multiplier to multiply the input signal $X(t)$ to a reference signal $H(t)$ or $L(t)$, whereas the time-average of the product is obtained by an RC-circuit. The function S can be implemented by an analog switch that is set according to whether its input is above or below a certain value [2]. In a similar way as with the INVERTER we obtain an AND-gate:

$$\begin{aligned} Y(t) \;=\;& S\,(\,\langle\, X_1(t)\,H(t)\,\rangle\;\langle\, X_2(t)\,H(t)\,\rangle\,)\;H(t) \\ &+ S\,(\,\langle\, X_1(t)\,L(t)\,\rangle + \langle\, X_2(t)\,L(t)\,\rangle\,)\;L(t) \end{aligned}$$

This gate can be built by the same type of analog circuit elements as those used in the INVERTER.

The encoding of signals by noise in the above way brings with it a distinct advantage: noise added to signals tends to be orthogonal to those signals and will be automatically removed by the averaging process. This may lead designers to accept a decreased signal level with respect to the noise level, with the promise of a reduced power consumption. An overhead in time and hardware resources appears hard to avoid, though, due to the averaging process and the necessity in operations to effectively decode signals through the step function S. Yet the method may form the inspiration for schemes that incorporate noise as a natural given, and it may ultimately lead to a deeper understanding of the efficient use of noise in biological neural systems.

4. Discussion and Conclusions

Traditional design methods of computational structures, in which noise and fluctuations are considered nuisances that should be suppressed, may need reconsideration. As feature sizes in integrated circuitry approach nanometer scale sizes, phenomena occurring at these scales will need to be actively exploited. Nature serves as a great inspiration in this respect: biological organisms have evolved in an environment full of noise, and they are thus likely to contain mechanisms that are not only able to deal with noise, but also to actively exploit it. Biological organisms are very complex machines, the control of which involves an intricate level of information processing. The methods discussed in this paper may give a glimpse as to how noise and fluctuations can play a positive role to this end.

Acknowledgments

The contributions of Jia Lee have been indispensable in the development of Brownian Circuits, and I like to thank him for the many discussions and close collaborations.

References

[1] H. Ando, F. Peper, and K. Aihara. Chaotic synchronization and de-synchronization for a token-based computational architecture. In *Proc. NOLTA*, 2009.

[2] L.B. Kish. Noise-based logic: Binary, multi-valued, or fuzzy, with optional superposition of logic states. *Physics Letters A*, 373:911–918, 2009.

[3] J. Lee and F. Peper. On brownian cellular automata. In *Proc. of Automata 2008*, pages 278–291, UK, 2008. Luniver Press.

[4] J. Lee, F. Peper, S. Adachi, and K. Morita. Universal delay-insensitive circuits with bi-directional and buffering lines. *IEEE Trans. Computers*, 53(8):1034–1046, 2004.

[5] J. Lee, F. Peper, et. al. Brownian Circuits — Part II: Efficient Designs and Brownian Cellular Automata. *In preparation*, 2009.

[6] F. Peper, J. Lee, F. Abo, T. Isokawa, S. Adachi, N. Matsui, and S. Mashiko. Fault-tolerance in nanocomputers: a cellular array approach. *IEEE Transaction on Nanotechnology*, 3(1):187–201, 2004.

[7] F. Peper, J. Lee, et. al. Brownian circuits — Part I: Concept and basic designs. *In preparation*, 2009.

[8] S. Safiruddin, S.D. Cotofana, F. Peper, and J. Lee. Building blocks for fluctuation based calculation in single electron tunneling technology. In *Proc. 8th Int. Conf. on Nanotechnology (IEEE Nano)*, pages 358–361, 2008.

[9] A.J. Viterbi. *CDMA: Principles of Spread Spectrum Communication*. Prentice Hall, 1995.

[10] T. Yamada, M. Akazawa, T. Asai, and Y. Amemiya. Boltzmann machine neural network devices using single-electron tunnelling. *Nanotechnology*, 12(1):60–67, 2001.