International Journal of Approximate Reasoning 85 (2017) 139–158

Contents lists available at ScienceDirect

International Journal of Approximate Reasoning

www.elsevier.com/locate/ijar

Approximation enhancement for stochastic Bayesian inference ☆,☆☆

Joseph S. Friedman [a,b,*], Jacques Droulez [c], Pierre Bessière [c], Jorge Lobo [d], Damien Querlioz [a]

[a] Centre de Nanosciences et de Nanotechnologies, CNRS, Univ. Paris-Sud, Université Paris-Saclay, 220 rue André Ampère, 91405 Orsay, France
[b] Department of Electrical and Computer Engineering, The University of Texas at Dallas, 800 W. Campbell Rd., Richardson, TX 75080, USA
[c] Institut des Systèmes Intelligents et de Robotique, Université Pierre et Marie Curie, CNRS, 4 Place Jussieu, 75005 Paris, France
[d] Institute of Systems and Robotics, Department of Electrical and Computer Engineering, University of Coimbra, 3030-290 Coimbra, Portugal

ARTICLE INFO

Article history:
Received 15 May 2016
Received in revised form 12 March 2017
Accepted 14 March 2017
Available online 31 March 2017

Keywords:
Stochastic computing
Muller C-element
Bayesian inference
Autocorrelation
Approximate inference

ABSTRACT

Advancements in autonomous robotic systems have been impeded by the lack of a specialized computational hardware that makes real-time decisions based on sensory inputs. We have developed a novel circuit structure that efficiently approximates naïve Bayesian inference with simple Muller C-elements. Using a stochastic computing paradigm, this system enables real-time approximate decision-making with an area-energy-delay product nearly one billion times smaller than a conventional general-purpose computer. In this paper, we propose several techniques to improve the approximation of Bayesian inference by reducing stochastic bitstream autocorrelation. We also evaluate the effectiveness of these techniques for various naïve inference tasks and discuss hardware considerations, concluding that these circuits enable approximate Bayesian inferences while retaining orders-of-magnitude hardware advantages compared to conventional general-purpose computers.

1. Introduction

The development of autonomous robotic systems requires fast, low-power circuits that enable real-time decision-making. Conventional approaches use general-purpose processors with decision algorithms implemented in software and mapped to conventional Boolean logic and arithmetic. Such systems have the ability to perform a wide range of tasks, but are not ideal for any task, including decision-making. For systems dedicated to a single task, specialized hardware can provide optimizations that lead to increased speed, reduced circuit area, and decreased energy consumption.

To autonomously reason and perform actions based on sensory information, Bayesian inference efficiently incorporate information from independent sources [1]. Bayesian inference has been suggested as a fundamental component of biolog-

☆ This work was supported by the EU collaborative FET Project BAMBI FP7-ICT-2013-C, project number 618024 and a public grant overseen by the French National Research Agency (ANR) as part of the "Investissements d'Avenir" program (Labex NanoSaclay, reference: ANR-10-LABX-0035).
☆☆ This paper is part of the Virtual special issue on Special Issue on Unconventional computing for Bayesian inference, edited by Jorge Lobo and João Filipe Ferreira.

* Corresponding author at: Department of Electrical and Computer Engineering, The University of Texas at Dallas, 800 W. Campbell Rd., Richardson, TX 75080, USA.
E-mail addresses: joseph.friedman@utdallas.edu (J.S. Friedman), jacques.droulez@college-de-france.fr (J. Droulez), pierre.bessiere@college-de-france.fr (P. Bessière), jlobo@isr.uc.pt (J. Lobo), damien.querlioz@u-psud.fr (D. Querlioz).

http://dx.doi.org/10.1016/j.ijar.2017.03.007
0888-613X/

Table 1
Muller C-element truth table.

X	Y	Z
0	0	0
0	1	Z_{prev}
1	0	Z_{prev}
1	1	1

ical systems [2–4], and has been successfully applied to robotics and other sensory-motor systems [5]. In such systems, constantly-updated sensory information is provided to a reasoning circuit that makes decisions in response to new information resulting from a constantly-changing environment [6]. For such circuits, there are fundamental trade-offs between reaction time, circuit efficiency, and the level of approximation used for the inference and reasoning. Therefore, the design of systems specifically dedicated to Bayesian inference is an active research area, incorporating analog or digital circuits to enable systems that are more efficient than computers [7–11].

In this context, we recently proposed an especially compact and energy efficient computing system that provides approximate naïve Bayesian inference [12]. In this system, based on Muller C-elements used in a stochastic computing paradigm, the approximate probability of a particular event is calculated based on prior and evidence data. We demonstrated a nearly one-billion-fold improvement in the area-energy-delay product for approximate Bayesian inference, with exceptional robustness to hardware faults. However, we found that bitstream autocorrelation leads to an approximation of the inference, limiting this system to tasks that do not require exact computation.

In this paper, we explore techniques to improve the quality of the approximation by mitigating the bitstream autocorrelation. Circuits that reduce bitstream autocorrelation are presented that remove the inaccuracy at the cost of additional hardware components that reduce the system efficiency. We show that these autocorrelation mitigation techniques are effective in reducing bitstream autocorrelation, even for difficult inference tasks. Finally, we conclude that despite the hardware costs, autocorrelation mitigation permits the use of our stochastic C-element structure for approximations of Bayesian inference with massive efficiency improvements over conventional hardware systems.

2. Bayesian inference with stochastic C-elements

In this section, we summarize the findings of [12]. Approximate Bayesian inference can be performed with a simple cascade of C-elements, with stochastic bitstream inputs and outputs. This system provides several orders of magnitude improvement in computational efficiency, though bitstream autocorrelation limits the accuracy of the approximation.

2.1. Stochastic computing

Stochastic computing enables the efficient approximation of mathematical functions performed on streams of random binary values [13,14]. The value of a stochastic bitstream is the percentage of binary bits that are '1'. For example, "010011010" encodes $\frac{4}{9}$. The random nature of the bitstream prevents the encoding of exact values; the value is an approximation with an accuracy that increases with the bitstream length.

This bitstream encoding enables the efficient realization of various functions, notably multiplication with a single AND gate [14]. For example, consider two input bitstreams to an AND gate "0101010101" and "0000011111", which both encode a value of 0.5. If the bit-pairs are input sequentially into an AND gate, the resulting output is "0000010101", which has a value of 0.3. This is not the exact 0.25 result of the multiplication, but it is as precise as possible given that only ten bits are employed.

Correlations between bitstreams, and within a bitstream, can cause significant errors. For the case of an AND gate with both input bitstreams being "0101010101": the output is also "0101010101", which is equivalent to 0.5. As the bitstreams are correlated with each other, this AND-gate stochastic multiplication does not perform the desired function. Additionally, in cases of autocorrelation with bits interacting within a bitstream, this autocorrelation can impact the output bitstream to produce undesired results. This autocorrelation is the primary challenge that is addressed in this paper, and the resulting imprecision is considered as a tradeoff for decreased power consumption, area, and latency.

2.2. C-element inference

The small area of circuits developed for stochastic computing enables an especially efficient Bayesian inference function. In particular, a Muller C-element is a simple circuit composed of as few as eight transistors that performs the function described by Table 1 [15]. The output Z maintains its state Z_{prev} unless both inputs X and Y are opposite the current output state, in which case the output switches to the shared input value. For C-element input signals with no autocorrelation, the output probability is approximated by [12,16–18]

$$P(Z) = \frac{P(X)P(Y)}{P(X)P(Y) + (1 - P(X))(1 - P(Y))}. \tag{1}$$

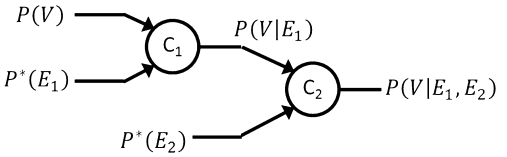

Fig. 1. Cascaded C-elements with one prior and two evidence inputs.

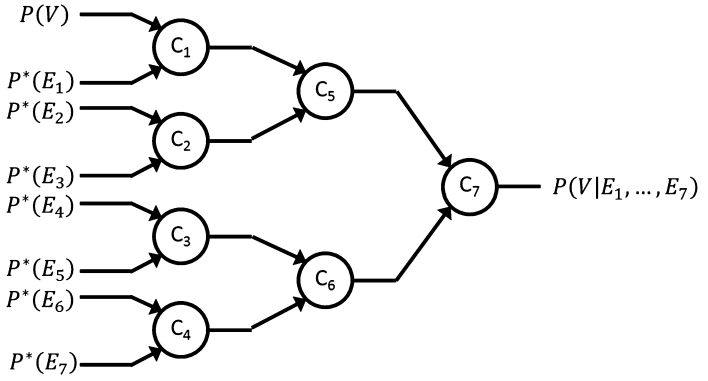

Fig. 2. Cascaded C-element circuit with one prior and seven evidence inputs.

Equation (1) can be shown to be equivalent to a Bayesian inference. Given prior $P(V)$ and evidence input E_1, the probability of an event can be calculated as

$$P(V \mid E_1) = \frac{P(E_1 \mid V)P(V)}{P(E_1 \mid V)P(V) + P(E_1 \mid \overline{V})P(\overline{V})}. \tag{2}$$

Defining a parameter $P^*(E_1)$ by

$$P^*(E_1) \equiv \frac{P(E_1 \mid V)}{P(E_1 \mid V) + P(E_1 \mid \overline{V})}, \tag{3}$$

equation (2) can be rewritten as

$$P(V \mid E_1) = \frac{P^*(E_1)P(V)}{P^*(E_1)P(V) + (1 - P^*(E_1))(1 - P(V))}. \tag{4}$$

Equation (4) can be seen as equivalent to equation (1), with $P(X)$ as the prior, $P(Y)$ as evidence, and $P(Z)$ as the posterior probability of an event. A simple C-element thus performs a complete Bayesian inference with two sources of input information [12].

Multiple C-elements can be cascaded to perform more complex Bayesian inferences with multiple sources of evidence [12]. Incorporating a second evidence input E_2, with E_1 and E_2 conditionally independent and $P^*(E_2)$ defined similarly to $P^*(E_1)$, the posterior probability based on E_1 and E_2 can be determined as

$$P(V \mid E_1, E_2) = \frac{P(V \mid E_1)P^*(E_2)}{P(V \mid E_1)P^*(E_2) + (1 - P(V \mid E_1))(1 - P^*(E_2))}. \tag{5}$$

This is equivalent to two cascaded C-elements, as shown in Fig. 1, and this concept can be extended to an arbitrarily large number of evidence inputs. The C-element tree shown in Fig. 2 performs inference with a prior and seven independent sources of evidence, performing an approximation of

$$P\big(V \mid \{E_1, \ldots, E_i\}\big) = \frac{P(V)\prod_{x=1}^{i} P^*(E_x)}{P(V)\prod_{x=1}^{i} P^*(E_x) + (1 - P(V))\prod_{x=1}^{i}(1 - P^*(E_x))}, \tag{6}$$

with P^* defined above. As a pedagogical example, this circuit can be used to perform spam filtering on the five messages in Table 2, with the presence or absence of eight key words providing evidence to determine whether each message is spam. As shown in the simulation results of Fig. 3, the C-element inference circuit provides approximately correct results. As expected from the discussion of section 2.1, the results become increasingly accurate with longer bitstreams. However, as particularly noticeable for messages D and E, the spam probability resulting from this approximate inference does not approach the correct value. This is a result of the autocorrelation described in the next section.

Table 2
Sample messages for spam detection.

Message	Message text
A	Do **you** want to get **pizza** for lunch?
B	**You** should **check** out these **stochastic** simulations.
C	If **you** want to earn a **fortune**, send a $100 **check** to **Nigeria** and we will **transfer** $10,000 to your account.
D	My **commute** to **Nigeria** includes a **transfer** in Morocco. I will **check** if my flight is on schedule – if so, do **you** want to get **pizza** when I arrive?
E	There is a $10 fee for all **check transfers**.
F	**You!**

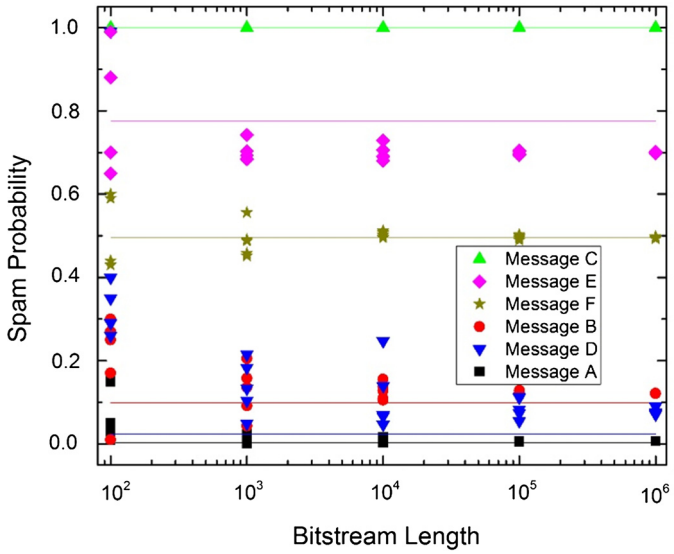

Fig. 3. Symbols: Spam probability value for messages A–E of Table 1 obtained by cascaded C-elements for several simulation runs as a function of observed bitstream length. Horizontal lines: exact inference.

2.3. Bitstream autocorrelation

Bitstream autocorrelation leads not only to slower convergence (*i.e.*, an increased number of bits are required for the output to reach the value produced by an infinite number of bits), but also to convergence toward an incorrect value (*i.e.*, the value to which the output converges is not equivalent to the Bayesian inference expected from equation (6)). This is a result of the expected C-element behavior: unless both inputs are opposite the current state of the output, the next output bit is identical to the previous output bit. In a C-element tree, each C-element stage leads to an increase in bitstream autocorrelation. Therefore, the output of each C-element has longer "domains" of consecutive '1's or '0's than either input.

As mentioned in section 2.2, the expectation that a C-element performs Bayesian inference is premised on the assumption that the input bitstreams of each C-element are uncorrelated. However, as the input bitstreams to C-elements beyond the first stage come from previous C-element output bitstreams, the C-element input bitstreams are autocorrelated in all C-element stages after the first. The fact that the C-element outputs have autocorrelation prevents the use of equation (1) in describing a C-element tree, as C-element output bitstreams with significant autocorrelation are used as inputs to other C-elements. As a result, the cascaded C-element output probabilities do not converge to the "correct" value expected from equation (6), as seen for messages D and E in Fig. 3. This convergence to incorrect values limits the accuracy of the approximate inference, and therefore the utility of the proposed stochastic Bayesian inference system. The remaining sections of this paper address this challenge, providing techniques to mitigate autocorrelation by rerandomizing the bitstreams without modifying the encoded probabilities.

2.4. Computing improvements

In comparison to conventional approaches, the primary benefits provided by this C-element stochastic Bayesian inference system are an increase in fault tolerance and computing efficiency [12]. Whereas a conventional system provides an exact solution after a specific period of time unless a fault occurs, this stochastic inference system provides increasingly accurate solutions as a function of time and is minimally affected by faults.

2.4.1. Fault tolerance

In conventional binary number representations, the unintentional flipping of a single bit drastically alters its value. For example, if the unsigned binary number "11011111" is written as "01011111", this flipping of a single bit causes the value to change from 223 to 95. In contrast, stochastic bitstreams are minimally affected by single bit-flips; in the example above, the value represented by the stochastic bitstream becomes 0.75 rather than 0.875. While still a noticeable error, this change is far less drastic than for the conventional binary number.

This fault tolerance advantage inherent to stochastic bitstream number representations extends to stochastic computation. In particular, when a C-element output does not switch when expected due to insufficient current to overpower its latch, the effect on the output bitstream is minimal. This "not switching" fault leads to an error in a single bit of the bitstream. As an example, [12] considered a C-element with input bitstreams representing the values 0.1 and 0.7. The Kullback–Leibler (KL) divergence, which can be used to measure error, was found to be below 0.01 even for an extremely pessimistic "not switching" fault rate of 99% (realistic fault rates are far less than 1%). This stochastic Bayesian inference system can thus tolerate faults at levels that would cripple a conventional system.

2.4.2. Inference efficiency

This stochastic inference structure provides approximate Bayesian inferences in far less time than required by conventional systems to produce its precise answer, and with significantly less power and area overhead. The accuracy of the stochastic Bayesian inference increases as the number of output bits increases, permitting decision-making based on a sufficiently accurate approximation with minimal hardware costs. In particular, approximate inference with a KL divergence of 0.01 (corresponding to a value of 0.5 being approximated as 0.57) can be achieved with a roughly one billion-fold improvement over the area-energy-delay product of a conventional system.

This comparison was performed in a manner that enables the conclusion to be relevant to most hardware systems [12]. Standard CMOS cells at the 90 nm node were used to determine the area, energy, and delay of both the C-element structure and a conventional floating point structure, thus providing an apples-to-apples comparison. Though the comparison will vary at different CMOS technology nodes or with an FPGA, a several orders-of-magnitude computing efficiency improvement will remain.

3. Proposed approximation enhancement techniques

To reduce bitstream autocorrelation and improve the accuracy of the approximate inference, distinct additions and modifications of the C-element circuit are proposed. The goal of these rerandomization techniques is to generate random bitstreams encoding the same value as a reference bitstream, with shorter domains than the reference bitstream; if the bitstreams are fully rerandomized and the autocorrelations fully removed, all C-elements function according to equation (1) and an accurate inference is achieved.

In this section, six rerandomization techniques are presented at both a conceptual and circuit level – two based on averaging, two based on counting, and two based on shifting access. Three of these are novel; three are interpretations of the hysteretic filters presented by Sharifi Tehrani et al. [19] for stochastic decoding, and the terminology in this section is consistent with theirs. The effectiveness of these techniques for approximate Bayesian inference is evaluated in section 4 for a variety of inference tasks, permitting a comparison of their rerandomization behavior. Their hardware costs are described in section 5, enabling conclusions regarding the preferences amongst the six rerandomization techniques.

3.1. Averaging techniques

An intuitive technique for stochastic bitstream rerandomization is to directly calculate the mean value of a bitstream and to continuously generate a new bitstream with the same mean value. Two averaging techniques are described below, respectively using a moving average and a series of single-run averages.

3.1.1. Moving average

A first intuitive technique is to follow each C-element (instruction i) with a moving average circuit that continually keeps track of the average value of the previous 2^x bits $C_i^*, C_{i-1}^*, \ldots, C_{i-2^x}^*$ of the C-element output C^*. The moving total MT is updated at each clock cycle (instruction ii). This value can then be used in concert with a random number generator to generate a new bit C_i at each clock cycle with the probability of a '1' given by the moving average $\frac{MT}{2^x}$ (instruction iii). This technique can be described by the following procedure:

(i) Normal C-element: $C_i^* = (A_i \wedge B_i) \vee (C_{i-1}^* \wedge (A_i \vee B_i))$

(ii) Update moving total: If $C_i^* \oplus C_{i-2^x}^* = 1$,
- Add 1 to MT if $C_i^* = 1$
- Subtract 1 from MT if $C_i^* = 0$

(iii) Generate output: $Pr(C_i = 1) = \frac{MT}{2^x}$

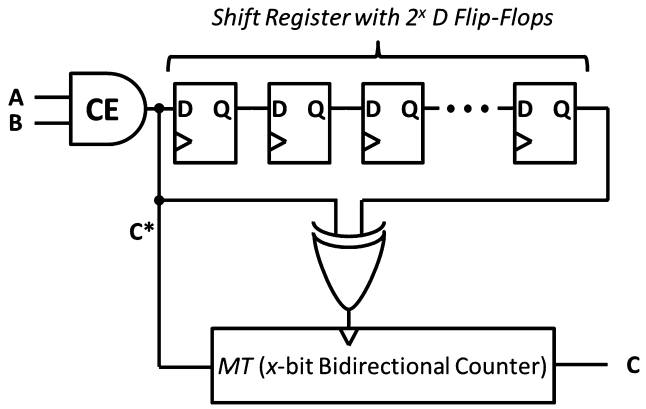

Fig. 4. Moving average rerandomization circuit.

As 2^x bits must enter the moving average circuit before the moving average can be calculated, 2^x clock cycles must pass before a reliable output is provided. There must therefore be either 2^x unreliable output bits or an initial delay of 2^x clock cycles; the latter approach is analyzed in this work. As this 2^x value is small relative to the bitstream lengths analyzed in section 4, this latency is not included in the hardware cost analysis.

From a hardware perspective, this rerandomization procedure requires a C-element, shift register, bidirectional counter, and XOR gate, as shown in Fig. 4. The conventional C-element takes the two input bitstreams to generate C_i^*, which is an input the 2^x-bit shift register. At each clock cycle, bits move from one flip-flop to the next through the shift register. The output of the last flip-flop, $C_{i-2^x}^*$, is then XOR-ed with C_i^* to determine whether the bidirectional counter value *MT* must be updated: if $C_i^* \neq C_{i-2^x}^*$, the bidirectional counter is incremented if $C_i^* = 1$ and decremented if $C_i^* = 0$. The value *MT* stored in the bidirectional counter is used in concert with a random number generator to create the rerandomized output bitstream *C*.

3.1.2. Single-run average

In order to reduce the hardware cost of the averaging, the shift registers can be eliminated and the continuously-updated moving average replaced by a sequence of single-run averages. The initial C-element remains unchanged (instruction *i*). At each clock cycle, the *x*-bit clocked counter *CC* is incremented (instruction *ii*) and the C-element output is added to the single-run total *SRT* (instruction *iii*). Whenever *CC* reaches 2^x, *CC* resets to 0 and an overflow bit is sent to *SRT* and *AVE*. This overflow bit instructs *AVE* take in the value stored by *SRT*, and for *SRT* to reset to 0. This reset process repeats every 2^x clock cycles, with a randomly-generated output bit with '1' probability $\frac{AVE}{2^x}$ produced at each clock cycle.

(*i*) Normal C-element: $C_i^* = (A_i \wedge B_i) \vee (C_{i-1}^* \wedge (A_i \vee B_i))$
(*ii*) Update clocked counter: Add 1 to *CC*
(*iii*) Update single-run total: Add 1 to *SRT* if $C_i^* = 1$
(*iv*) Reset after each run: If $CC = 2^x$,
- Set $AVE = SRT$
- Set $CC = 0$
- Set $SRT = 0$

(*v*) Generate output: $Pr(C_i = 1) = \frac{AVE}{2^x}$

This circuit is far more compact than the moving average. As shown in Fig. 5, this circuit is composed of a C-element, two *x*-bit counters, and an *x*-bit register. The C-element output is an input to *SRT*, which outputs a value to *AVE* on an *x*-bit bus. The clocked counter *CC* continually increments its value, sending an overflow signal that causes the value in *SRT* to shift to *AVE* and for *SRT* to reset to 0. A random number generator generates an output bitstream *C* based on the value stored in *AVE*.

3.2. Counting techniques

Rather than repeatedly averaging the input bitstream, an alternative strategy directly outputs "regenerative" inputs and outputs a randomly generated bit when the inputs are "non-regenerative". For these techniques, "regenerative" inputs are defined as the case of $A_i = B_i$; otherwise, the inputs are referred to as "non-regenerative". A counter keeps track of the difference between the input and output bitstreams and is used to generate appropriate random numbers when the inputs are non-regenerative. These counting techniques are interpretations of the hysteric filters proposed in [19] for a different application.

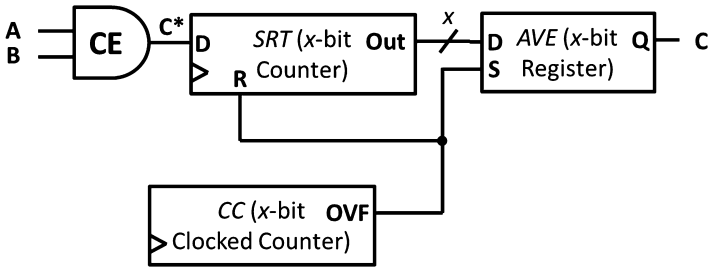

Fig. 5. Single-run average rerandomization circuit.

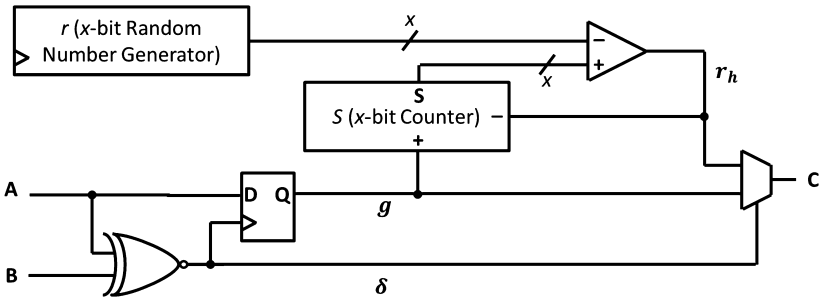

Fig. 6. Single-ended counter rerandomization circuit.

3.2.1. Single-ended counter

The simplest counting rerandomization circuit is the single-ended counter [19]. As mentioned above, the inputs are determined to be regenerative if $A_i = B_i$, in which case δ_i is set to 1. If the inputs are non-regenerative, δ_i is set to 0 (instruction *i*). The "regenerative bit" g is set to the value in A_i if $\delta_i = 1$; otherwise, g retains its previous value (instruction *ii*). At each stage, an x-bit random number r is generated and compared to the x-bit counter value S: if $r < S$, the "non-regenerative bit" r_h is set to 0; otherwise, r_h is set to 1 (instruction *iv*). If the inputs are regenerative (that is, $\delta_i = 1$), the output is set to the regenerative bit g; otherwise, the non-regenerative bit r_h is output as C (instruction *v*). At each clock cycle, the x-bit Sum counter S is incremented if $g = 1$ and decremented if $r_h = 1$; neither, one, or both can occur at any given clock cycle (instruction *vi*). This Sum counter is used to provide a similar number of '1' bits to the generated output bitstream as in a reference C-element bitstream.

(*i*) Check for input equivalence: $\delta_i = \overline{A_i \oplus B_i}$
(*ii*) Conditionally update regenerative bit: If $\delta_i = 1$, $g = A_i$
(*iii*) Generate x-bit random number r
(*iv*) Generate non-regenerative bit:
- $r_h = 1$ if $r < S$
- $r_h = 0$ otherwise

(*v*) Generate output:
- $C_i = g$ if $\delta_i = 1$
- $C_i = r_h$ otherwise

(*vi*) Update sum:
- Add 1 to S if $g = 1$
- Subtract 1 from S if $r_h = 1$

Fig. 6 depicts the circuit at a hardware level. A flip-flop stores the value of A, and its output value g is updated only when $A_i = B_i$ (when $\delta_i = 1$ and the inputs are regenerative). Simultaneously, a random number generator provides an x-bit number r at each clock cycle, which is compared to the x-bit number S by a comparator. The comparator outputs r_h. Depending on the value of δ, the multiplexer outputs either r_h or g. At each clock cycle, the Sum S is also updated if either r_h or g is '1'.

3.2.2. Differential counter

The differential counter is similar to the single-ended counter, but with a more complex counting mechanism [19]. Here, r_h and g values of '0' also cause the Sum counter to be updated: if $g = 1$, S is incremented; if $g = 0$, S is decremented; if $r_h = 0$, S is incremented; if $r_h = 1$, S is decremented (instruction *vi*). As can be seen in Fig. 7, the hardware design is also not significantly changed, the only difference being the replacement of the double-input counter by a double-input up/down counter.

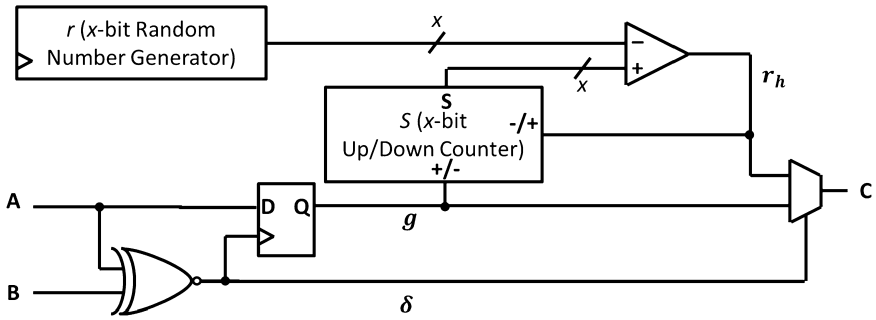

Fig. 7. Differential counter rerandomization circuit.

(*i*) Check for input equivalence: $\delta_i = \overline{A_i \oplus B_i}$
(*ii*) Conditionally update regenerative bit: If $\delta_i = 1$, $g = A_i$
(*iii*) Generate x-bit random number r
(*iv*) Generate non-regenerative bit:
- $r_h = 1$ if $r < S$
- $r_h = 0$ otherwise

(*v*) Generate output:
- $C_i = g$ if $\delta_i = 1$
- $C_i = r_h$ otherwise

(*vi*) Update sum:
- Add 1 to S if $g = 1$
- Subtract 1 from S if $g = 0$
- Add 1 to S if $r_h = 0$
- Subtract 1 from S if $r_h = 1$

3.3. Shifting access techniques

A third strategy for reducing autocorrelation and rerandomizing a bitstream is to store regenerative bits in a controlled shift register and access the bits in an order different from a reference C-element bitstream. In these techniques, the output is never a randomly-generated bit – rather, the output is a bit chosen from the shift register in a manner particular to the specifics of the technique.

3.3.1. Edge memory

An edge memory outputs a bit selected randomly from a shift register at each non-regenerative clock cycle [19,20]. As previously, the inputs are determined to be regenerative if $A_i = B_i$, in which case $\delta_i = 1$ (instruction *i*). If the inputs bits are regenerative, the shift register data is shifted by one bit, with the current input A_i stored in E_0 (instruction *ii*) and output as C_i (instruction *iv*). For cases where the input bits are non-regenerative ($\delta_i = 0$), a randomly chosen bit stored in shift register is sent to the output (instructions *iii*–*iv*).

(*i*) Check for input equivalence: $\delta_i = \overline{A_i \oplus B_i}$
(*ii*) Conditionally shift 2^x-bit shift register: If $\delta_i = 1$,
- Set each bit E_j to the value stored in E_{j-1}
- Set bit $E_0 = A_i$

(*iii*) Generate non-regenerative bit:
- Choose a bit E_r randomly from E

(*iv*) Generate output:
- $C_i = E_0$ if $\delta_i = 1$
- $C_i = E_r$ if $\delta_i = 0$

As shown in Fig. 8, the central hardware component of the edge memory is a 2^x-bit shift register E. The bits are shifted to the right whenever $A_i = B_i$, and A_i is stored in the leftmost flip-flop of the shift register. For use with non-regenerative inputs, a random number with x bits is generated that refers to a randomly chosen bit E_r in E. The final multiplexer then chooses whether to output the regenerative bit E_0 or the non-regenerative bit E_r.

3.3.2. Circular buffer

The final rerandomization technique explored in this paper is the circular buffer, which functions similarly to the edge memory without random number generation. (See Fig. 9.) Similar to the edge memory, the bits in the shift register E are

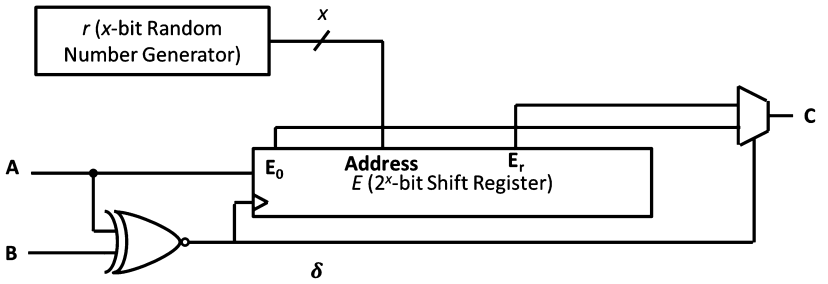

Fig. 8. Edge memory rerandomization circuit.

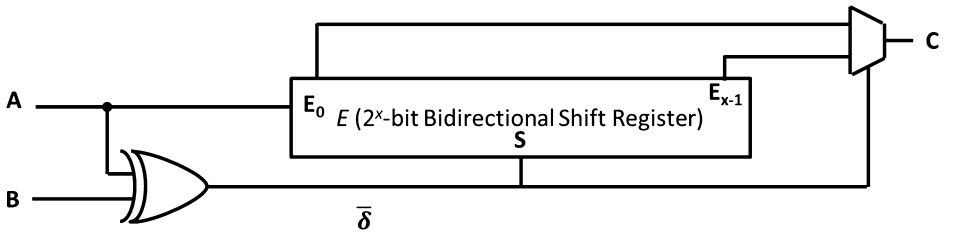

Fig. 9. Circular buffer rerandomization circuit.

shifted to the right and the current input A_i is stored in E_0 (the left-most flip-flop of E) if $\delta_i = 1$ (instructions *i*–*ii*). If $\delta_i = 0$, a circular motion is made with the bits shifted to the left and the bit in the left-most flip-flop is shifted to the right-most flip-flop (instruction *ii*). The output multiplexer then chooses between E_0 and E_{2^x-1} depending on the value of δ.

(*i*) Check for input equivalence: $\delta_i = \overline{A_i \oplus B_i}$
(*ii*) Shift x-bit shift register:
- If $\delta_i = 1$:
  - Set each bit E_j to the value stored in E_{j-1}
  - Set bit $E_0 = A_i$
- If $\delta_i = 0$:
  - Set each bit E_j to the value stored in E_{j+1} and $E_{2^x-1} = E_0$

(*iii*) Generate output:
- $C_i = E_0$ if $\delta_i = 1$
- $C_i = E_{2^x-1}$ if $\delta_i = 0$

4. Rerandomization circuit effectiveness

To demonstrate the effectiveness of these rerandomization circuits for reducing autocorrelation and enhancing the accuracy of the Bayesian inference approximation, the performance is analyzed for several sample tasks. These tasks were performed with C-elements discussed in [12] replaced by the rerandomization circuits, collectively labeled "ECE" for "extended C-element". In all of these tasks, eight input bitstreams are randomly generated with particular probabilities. The results are all based on ten simulation runs with one million bits and a 200,000 bit initialization period not included in the statistical analysis; this bitstream length was found to be appropriate for all rerandomization circuits and analyses, but is not optimized for minimal delay and energy. The input pseudorandomly-generated bitstreams have negligible autocorrelation and cross-correlation. The error bars in the graphs denote the standard deviation. Varying numbers of bits x were considered for the various rerandomization circuits and inference tasks in order to thoroughly explore the behavior while minimizing the excessive simulation times required by large rerandomization circuits.

When evaluating the simulation results, it is crucial to consider that the hardware component count is exponentially related to the number of bits x that characterizes the rerandomization circuit for the circuits with a shift register (moving average, edge memory, circular buffer). In contrast, the other three circuits (single run average, single counter, differential counter) exhibit a linear relationship between x and the hardware component count. Therefore, the inference accuracy observed for the edge memory, moving average, and circular buffer must outweigh this hardware cost to be deemed superior to the simpler rerandomization circuits.

4.1. Control task autocorrelation

We first consider a simple situation where all inputs to the circuit of Fig. 2 represent 50% (0.5), and the C-elements have been replaced by ECEs. In this case, all signals in the circuits should also represent 50%. Due to the symmetry in the task,

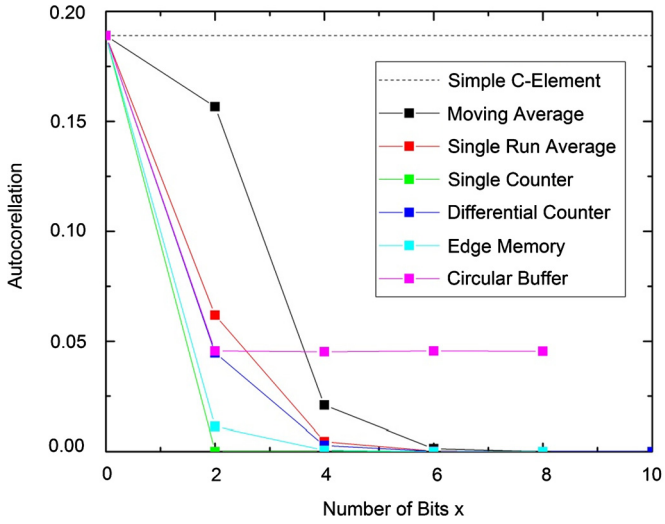

Fig. 10. Autocorrelation of rerandomization circuits after one stage.

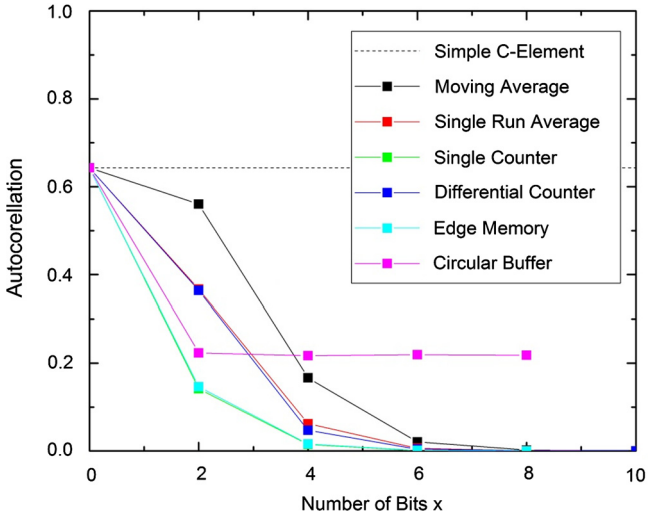

Fig. 11. Autocorrelation of rerandomization circuits after three stages.

all ECEs provide this output value, but with varying degrees of autocorrelation. Fig. 10 presents the autocorrelation of the output of the circuit as a function of the bit numbers x in the various ECEs, after one stage of ECE, while Fig. 11 presents the autocorrelation of the output after the three ECE stages. A horizontal line depicts the autocorrelation obtained using simple C-elements. As all circuits provide the same output value, comparing the autocorrelation provides a fair assessment of the ECE effectiveness.

Under these uniform conditions, the ECEs behave differently from each other, in a relatively counterintuitive fashion. First, although they are based on a mathematically similar concept, the moving average and single run average provide remarkably different results. Both converge to a zero autocorrelation as x is increased, but the single run converges notably faster. This can be interpreted as follows. Autocorrelated signals are prone to presenting long sequences of '0's and '1's. The moving average, by computing an average continuously, tends to be influenced by such sequences and to reproduce them, while the single average more naturally cuts such sequences. As the moving average requires 2^x bits of memory and the single average requires only x bits of memory, this suggests the vast superiority of the single run average over the moving average.

Despite the fact that the differential counter is a more complicated version of the single counter, the differential counter converges similarly to single run average, while the single counter converges faster. Although it relies on a complex concept and requires 2^x bits of memory, the edge memory converges similarly to the single counter.

The circular buffer behaves differently from all other circuits: although for $x = 2$ it is one of the best circuits, it is the only circuit that does not converge to zero autocorrelation as x is increased. The fact that this circuit is not capable of removing autocorrelation is due to the fact that it is the only circuit that does not feature a random number generator.

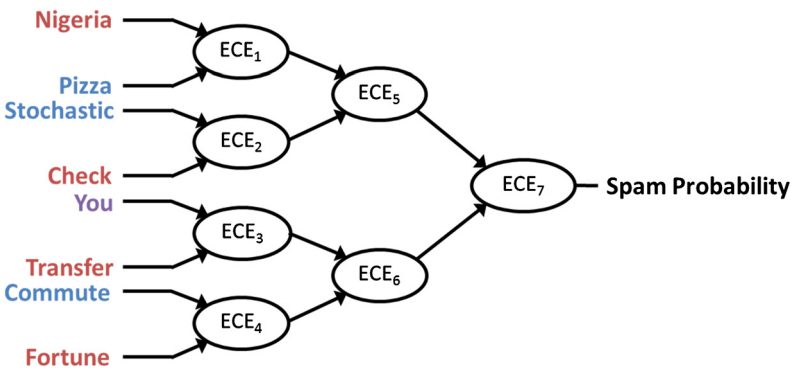

Fig. 12. Spam filter inference task with extended C-element rerandomization circuits.

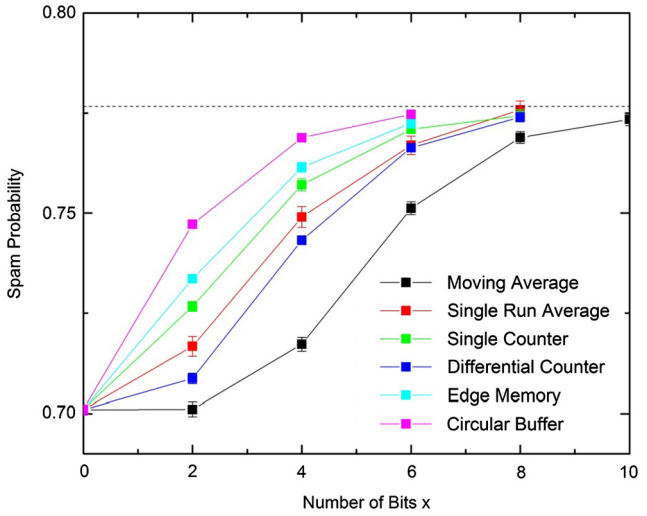

Fig. 13. Spam filter results for message E with all rerandomization circuits (moving average, edge memory, and circular buffer require 2^x bits of memory).

However, we see in the following subsections that these autocorrelation results do not necessarily translate to practical inference tasks.

4.2. Bayesian inference task #1: spam filter

As discussed in section 2.2 and shown in Fig. 3, the spam filter output converges toward an incorrect value for messages D and E in Table 2. These spam filtering inference tasks are therefore used to evaluate the ability of these rerandomization circuits to improve the accuracy of the inference approximation.

4.2.1. Task description

In the spam filter tasks described thoroughly in [12], stochastic bitstreams are input to the Bayesian inference circuit that represent the presence of various words in a message and their "spamicity". According to the errorless mathematical calculation of the inference using equation (6) and the contrived spamicity data, message E is determined to be spam with a 77.7% probability and message D is given a 2.4% probability of being spam. However, the simple C-element circuit converges to probabilities of 70.1% and 7.8%, respectively. The approximate inference is enhanced by replacing the simple C-elements with the rerandomization (ECE) circuits in Fig. 12.

4.2.2. Results for message E

As shown in the simulation results of Fig. 13, the various rerandomization circuits successfully mitigate the autocorrelation and enable an accurate approximation of the inference. As the number of bits in the rerandomization circuits increases, the output spam probability approaches the mathematically computed probability. However, surprisingly, the results are very different than what would have been expected from Figs. 10 and 11. For this task, the circular buffer, one of the worst circuits according to Figs. 10 and 11, performs far better than the other rerandomization circuits, providing increased accuracy with fewer bits in its shift register. However, this accuracy comes at the cost of a greater number of hardware components. Whereas the edge memory also performs quite well, it is notable that the hardware-expensive moving average

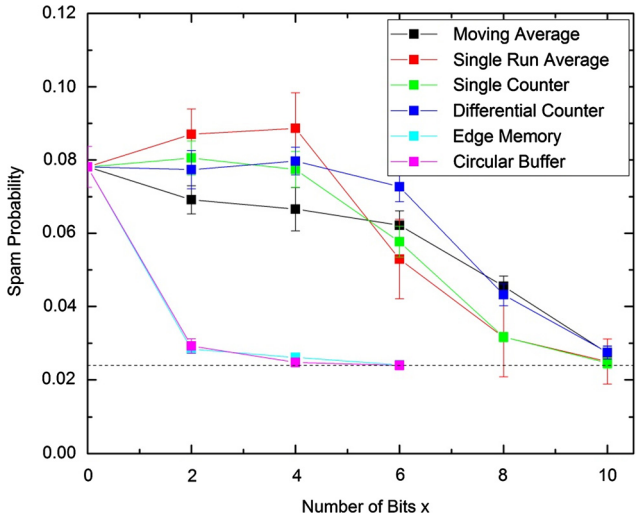

Fig. 14. Spam filter results for message D with all rerandomization circuits (moving average, edge memory, and circular buffer require 2^x bits of memory).

performs quite poorly, consistent with our interpretation of Figs. 10 and 11. The rest of the curves are consistent, but do not map perfectly to Figs. 10 and 11.

4.2.3. Results for message D

Fig. 14 shows that rerandomization circuits also provide enhanced approximations for the Bayesian spam filter for message D. The behavior is qualitatively different from Figs. 10 and 11, as it is clearly differentiated between the circular buffer and edge memory, which produce a nearly correct result with only two bits (four-bit shift register), and the other four circuits which require ten bits. As with message E, the moving average does not perform sufficiently well to make up for its large hardware cost.

4.3. Bayesian inference task #2: uniform tree

Whereas the spam filter simulations demonstrate the ability of the rerandomization circuits to enable accurate inference approximations, it is also enlightening to consider the behavior more systematically. This task involves a uniform tree having inputs with equal probabilities.

4.3.1. Task description

This task uses the same circuit structure as the spam filter, with all inputs having a value 0.4. Therefore, by Bayes' rule in equations (1) and (2), the expected output probability after the first stage is 30.7%, after the second stage is 16.5%, and after the third stage is 3.8%. The simple C-element consistently overestimates the output probability, producing a final result of 9.2%. This inference task thus provides a good opportunity to compare the capabilities of the various rerandomization circuits in a systematic manner.

4.3.2. Results

As shown in Fig. 15, the edge memory and single counter perform particularly well. The single run average and differential counter perform nearly as well, and the moving average lags far behind. Most interestingly, the circular buffer provides excellent results with two bits (four-bit register), but does not converge to the mathematically correct value, as in the uniform case of Figs. 10 and 11.

4.4. Inference task #3: symmetric chain

A third set of inference tasks involves a chain of extended C-elements. This technique, though not ideal for performing complex inferences [12], provides insight for evaluating the behavior of the rerandomization circuits. In particular, the symmetry of the inputs implies that a correct output is precisely 50%, and the approximation accuracy can be easily judged by the distance from this expected value.

4.4.1. Task description

This inference task uses the chain of extended C-elements shown in Fig. 16. The first four inputs have a probability of α, and the final four inputs have a probability $1-\alpha$. As a result of this symmetry, equation (2) suggests an expected output of 50%. However, due to the fact that the inputs are sequential, there is an internal node with an extreme probability. For

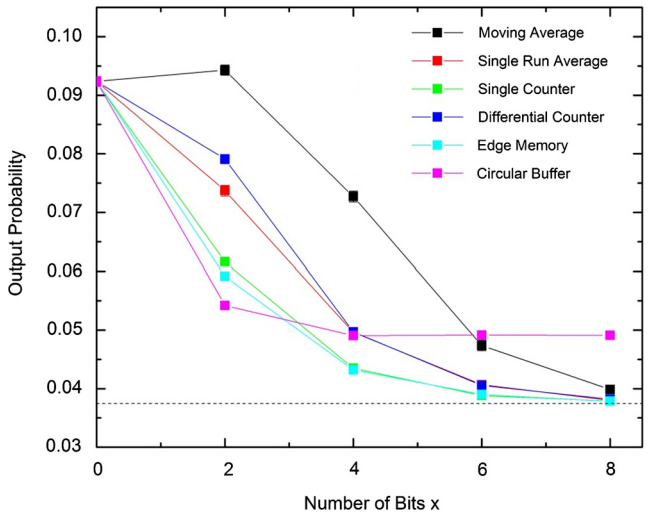

Fig. 15. Inference tree output with all rerandomization circuits for all inputs having a value 0.4 (moving average, edge memory, and circular buffer require 2^x bits of memory).

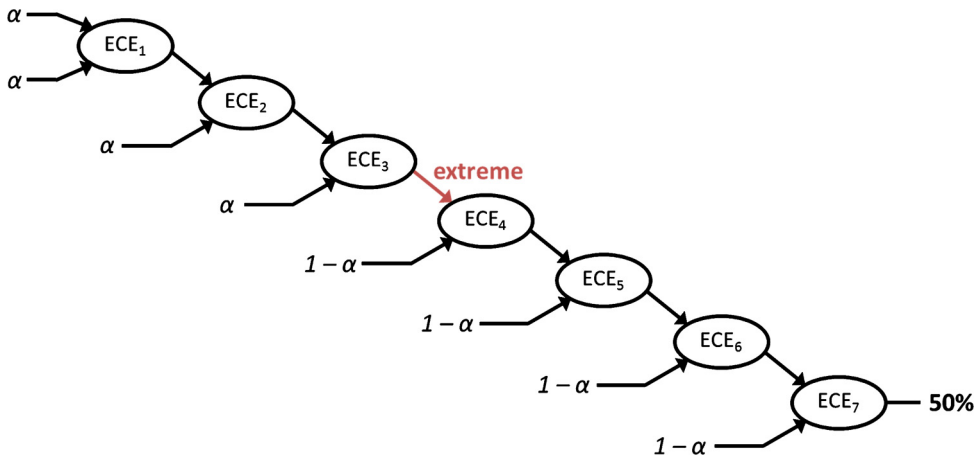

Fig. 16. Symmetric chain inference task with extended C-element rerandomization circuits.

$\alpha = 40\%$, the most extreme internal node has an expected probability of 16.5%; for $\alpha = 10\%$, the most extreme internal node has an expected probability of 0.015%. These extreme probabilities imply that there is minimal switching in the bitstream, leading to long domains and large autocorrelation. Therefore, a simple C-element provides output probabilities of 32.9% and 0.5%, respectively, demonstrating the need for bitstream rerandomization.

4.4.2. Results for moderate inputs

The large autocorrelation in the inference chain is shown in Fig. 17 to be overcome by the rerandomization circuits for $\alpha = 40\%$. The circular buffer provides by far the best results, reaching 49.4% with just two bits (four-bit register). The edge memory performs barely better than the single counter, with the single-run average and differential counter slightly behind. Similar to the other tasks, the moving average circuit performs quite poorly.

4.4.3. Results for extreme inputs

The more extreme case of $\alpha = 10\%$ requires larger circuits with more rerandomization bits to provide an accurate approximate inference. As can be observed in Fig. 18, all of the rerandomization circuits successfully converge to the correct output probability, albeit with greater than 10 bits (1,024 bit-shift register). Comparing amongst the various circuits, the results are similar to the previous case, the primary difference being improved performance of the single-run average relative to the single counter. These results also highlight the challenge of performing inferences with extreme probabilities.

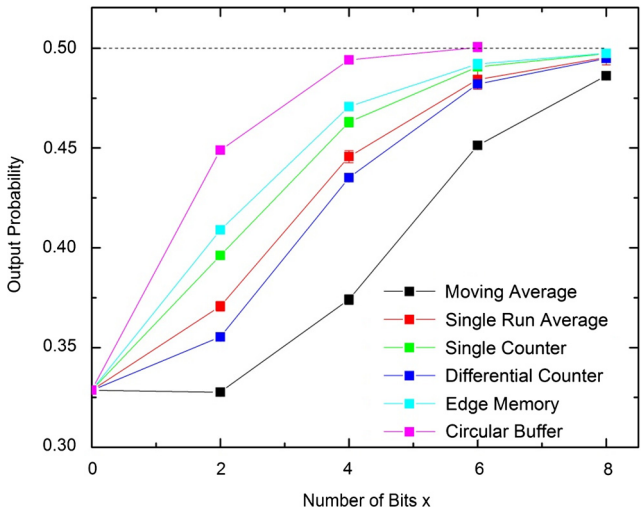

Fig. 17. Symmetric chain results for $\alpha = 40\%$ with all rerandomization circuits (moving average, edge memory, and circular buffer require 2^x bits of memory).

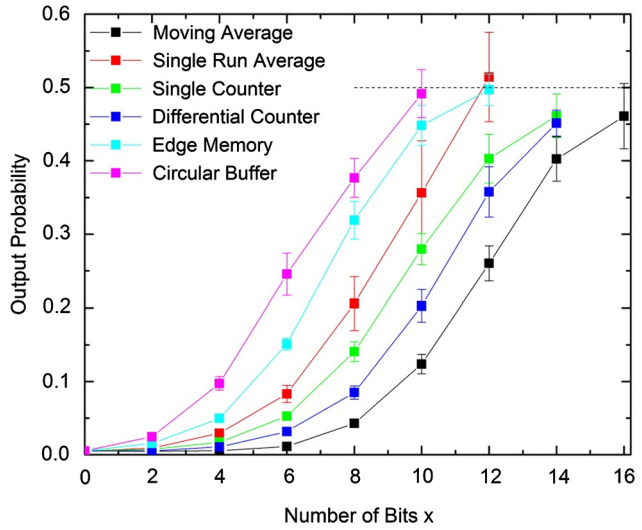

Fig. 18. Symmetric chain results for $\alpha = 10\%$ with all rerandomization circuits (moving average, edge memory, and circular buffer require 2^x bits of memory).

5. Hardware cost considerations

The primary benefits of this stochastic approximate Bayesian inference paradigm are the hardware efficiency and fault tolerance [12,21]. As discussed previously, these benefits require an approximation of the inference rather than a direct computation. To proceed towards a practical system, the utility of these approaches can be quantified through a comparison of the inference accuracy and hardware costs.

Using reconfigurable logic devices (FPGAs) we were able to have working prototypes and analyze resource usage and performance. We have synthesized all of the rerandomization circuits for various numbers of bits to provide an analysis of their relative hardware costs in terms of area, speed, and power consumption. In order to enable a fair comparison as in section 2.4, all circuits have been synthesized and tested in the same way: logic synthesis using Altera Quartus II 14.0 IDE targeting a board with a Cyclone IV FPGA (EP4CE115F29C7), and Mentor Graphics Modelsim with PowerPlay Power Analyzer from the Quartus IDE to provide power consumption estimates. The synthesis is restricted to only use generic LEs (FPGA basic internal blocks, or logic elements) so that the resource usage resulting from the synthesis can be used as a relative measure for circuit area. For the power analysis ModelSim test vectors, a VHDL test bench was used to generate pseudo-random numbers with uniform distribution, and inputs with a 50% toggle rate, providing a worst-case estimation of the power dissipation. While in some applications FPGAs might be used as the end target device for their flexibility, the relative hardware costs can also provide guidelines for their expected behavior in other environments, in particular

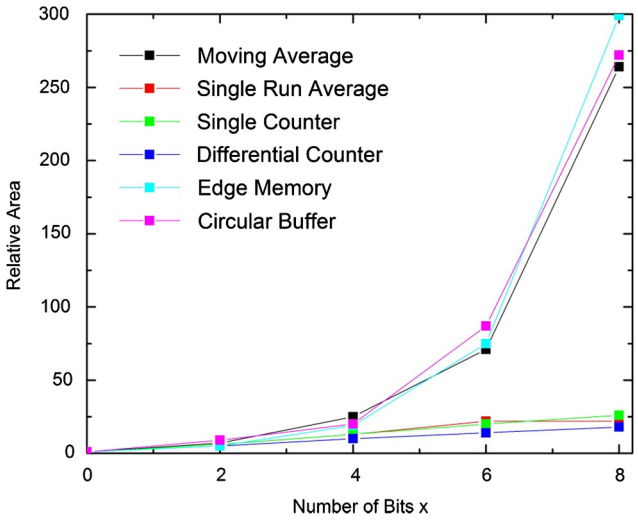

Fig. 19. Area overhead of rerandomization circuits relative to a single C-element.

application-specific integrated circuits (ASICs), which would constitute the optimal implementation of our scheme. The results in this section do not include the cost of the random number generators, which are discussed in the Discussion section instead.

5.1. Area utilization

The relative area utilization of the rerandomization circuits is shown in Fig. 19. Computed as the total number of logic elements (LEs) in the circuit, a zero-bit rerandomization circuit refers to a simple C-element. The area number includes both combinational elements and registers. The relative proportion varies considerably depending on the ECEs. For $x = 16$, the relative proportion of combinational to register elements is 0.04 for moving average, 1.4 for single run average, 3.3 for single counter, 2.3 for differential counter, 0.7 for edge memory, and 1.0 for circular buffer. Notice that a single LE can be used for some logic and a register, so using this total there is in fact an overlap. Our area metric does not double count in case of overlap.

For both counter circuits and the single-run average, the area is shown to increase linearly; the area increases exponentially for the moving average, edge memory, and circular buffer circuits. This exponential relationship is a result of the 2^x-bit shift registers that comprise the bulk of the circuit area. As can be readily discerned, these circuits are not efficiently implemented in hardware with more than six bits (64-bit shift register).

5.2. Computation speed

In stochastic computing, the precision of a signal increases with each bit in a bitstream. Therefore, the clock speed at which a stochastic computing system outputs bits is critical for enabling high-speed information processing. As shown in Fig. 20, the counters produce bits fairly slowly, whereas the single-run average suffers a gentler decline in speed. The maximum frequency plotted in the graph is the smallest clock period at which the circuit can operate without inducing errors, producing an output bit at each clock cycle. For the simple C-element circuit, the maximum FPGA frequency is 727 MHz.

5.3. Power dissipation

Power dissipation is a primary consideration in the development of efficient computing technologies, and is particularly relevant for battery-limited autonomous systems. As shown in Fig. 21, the power dissipation grows exceptionally large for the shifting access techniques with a large number of bits. These power data consider the FPGA dynamic power at a frequency of 50 MHz, as the static power is primarily consumed by inactive components of the reconfigurable circuit. The moving average, edge memory, and circular buffer circuits requires significantly more power than the other three circuits, which all require $\sim 20\times$ more power than a simple C-element.

5.4. Total efficiency

The area-energy-delay product (AEDP) is an effective means of evaluating the total efficiency of a computing system for general-purpose applications. The energy is computed by multiplying the maximum frequency by the power dissipation

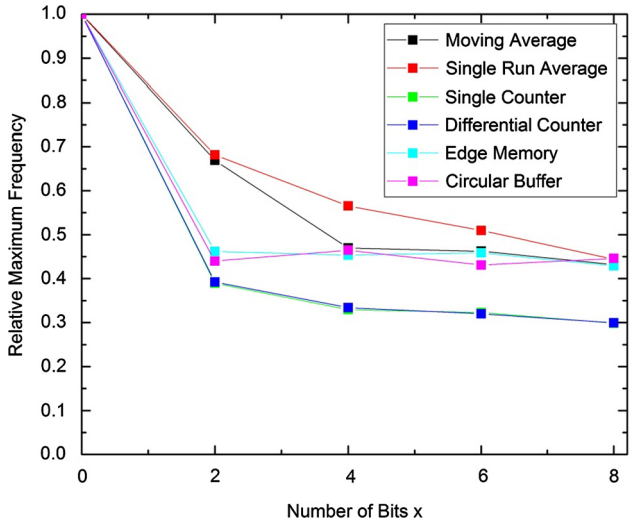

Fig. 20. Maximum operating frequency of rerandomization circuits relative to a single C-element.

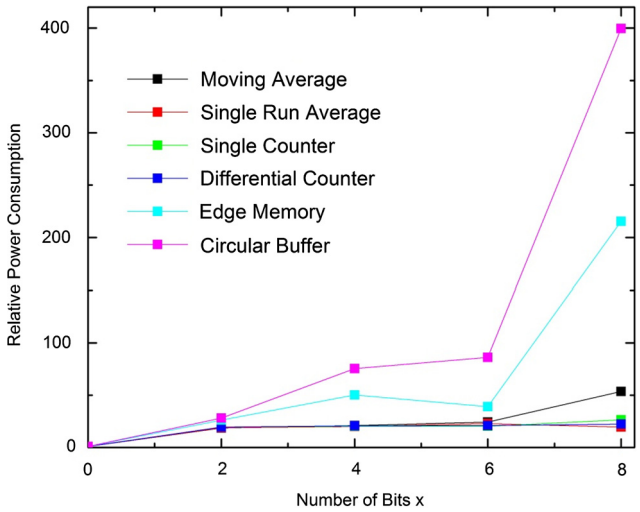

Fig. 21. Power dissipation of each rerandomization circuit with a clock frequency of 50 MHz, relative to a single C-element.

at this maximum circuit frequency. As can be observed on the logarithmic scale of Fig. 22, the AEDP of the counters and single-run average is roughly one thousand times larger than a simple C-element. The relative AEDP of the edge memory is similar up to four bits (sixteen-bit shift register), and increases dramatically for larger circuits. The circular buffer has a consistently larger hardware cost, though its ability to function without a random number generator provides a significant advantage. For all these circuits, an analysis of the total efficiency in concert with the rerandomization effectiveness enables comparisons of overall utility.

6. Discussion

This paper has proposed several rerandomization circuits for stochastic approximate Bayesian inference, and demonstrated their capability of improving the approximation accuracy. All the circuits have been shown to provide the desired results without inducing an excessive hardware cost, and the choice of rerandomization circuit is determined by the specific inference task and hardware specifications. Depending on the cost of the random number generator, and the potential availability of an analog random number generator, all the rerandomization circuits should be considered besides the moving average. Though the diversity of results for the various inference tasks prevents a strong claim regarding the absolute superiority of one of the circuits on all criteria, important conclusions can be drawn:

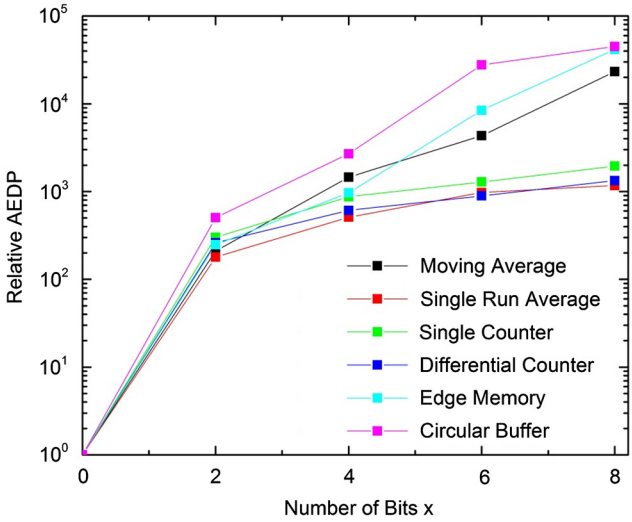

Fig. 22. Area-energy-delay product (AEDP) of each rerandomization circuit relative to a single C-element.

- *Averaging techniques*: The single-run average provides more accurate inference approximations than the moving average, despite the exponential difference in hardware resources of the two techniques. This counterintuitive improved capability for autocorrelation reduction is a result of differences in the averaging mechanism. For the moving average, the shift registers cause the average to be gradually and continuously updated, following the behavior of the input signal and retaining its correlations. In contrast, the single-run average provides abrupt shifts in the averages, enabling longer-range decorrelation within the bitstream. As the hardware cost of the moving average rerandomization circuit is exponentially larger, the single-run average is *superior* to the moving average in all aspects.
- *Counting techniques*: The inference approximations with the single counter are more accurate than with the differential counter, despite contrary findings in [19] where they were used for stochastic decoding. However, the accuracy difference between the two circuits is relatively small. As the differential counter is found to have a smaller hardware cost, neither circuit can immediately be considered superior.
- *Shifting access techniques*: The edge memory and circular buffer both provide excellent autocorrelation mitigation. Given the high effectiveness with a small number of bits, the high hardware cost may acceptable for some systems. In particular, the four-bit (sixteen-bit shift register) edge memory provides highly accurate approximate inferences with a relatively small hardware cost. The circular buffer is in general also highly effective, and the lack of a random number generator is a significant strength. However in situations with uniform or relatively uniform probabilities, the circular buffer is the only solution that is incapable of eliminating autocorrelation entirely. Therefore, the edge memory appears to be a more reasonable choice in general.
- *Comparison to conventional systems*: As already mentioned, the final efficiency of a Bayesian inference circuit based on stochastic computing depends on the expected precision of the output. When exact results are required, conventional deterministic processing techniques are more appropriate than the stochastic C-element inference due to the inability of stochastic computing systems to provide exact results. For approximate inference, the required precision determines the relative efficiency of the C-element structures relative to conventional deterministic circuits, thus dictating which is more suitable to that problem. In general, when high precision is required (low KL divergence to exact inference), deterministic circuits are more efficient than stochastic circuits. By contrast, for approximate inference sufficient for real-time decision-making (*i.e.*, KL divergence to exact inference <0.01), stochastic inference can be considerably more efficient than conventional circuits: we find that our autocorrelation mitigation circuits enable stochastic approximate Bayesian inference circuits with several orders of magnitude hardware efficiency improvements as compared to conventional systems with exact floating point computation. As demonstrated through simulation, eight-bit rerandomization circuits without shift registers (and four-bit edge memories and circular buffers) can produce accurate approximate inferences for relatively difficult tasks: for very long bitstreams, the KL divergence from an exact inference is generally much smaller than 0.01. In practical situations, with shorter bitstreams, the dominant error source will therefore be the input bitstream length, and inference accuracy can therefore be chosen based on input bit length. As shown in Fig. 19, the hardware cost of such circuits is about one thousand times larger than that of a simple C-element. However, following the approach introduced in [12], for the examples studied in the present paper, we find that the AEDP of stochastic approximate Bayesian inferences with rerandomization can be approximately 100,000 times smaller than a conventional exact Bayesian inference computation while maintaining KL divergence to exact inference <0.01. In other terms: the ECEs allow reaching inference with KL divergence <0.01 in situations where simple C elements cannot, while retaining considerable benefits in AEDP compared with floating point-based systems. In terms of energy alone, our circuits are

$\sim$50$\times$ more efficient than floating point operation, and in terms of area are $\sim$700x more efficient (also based on the analysis of [12] and our current results).

As a strategy for low precision inference, it is also worthwhile to consider conventional circuit structures with less precision than the double-precision floating point format; for example, fixed point or single-precision floating-point provides an intermediate level of precision with an intermediate hardware cost. Based on the literature, such strategies can increase energy efficiency by factors between two and ten [22–24]. Stochastic circuits remain beneficial, but less so when compared with full precision floating point computation. Additionally, low precision deterministic circuits lack the fault tolerance of stochastic systems as well as the ability of stochastic computation to adjust precision through control of the bitstream length.

Naïve Bayesian inference can thus be efficiently approximated by this stochastic computing system. Though the moving average technique is inferior, the other rerandomization circuits exhibit promising characteristics for enabling accurate approximations. The range of probabilities that may be input to the circuit should determine the choice between a shift register-based technique with a small number of bits and a counter or single-run average with additional bits. Further, it is worth considering the possibility of introducing rerandomization only at select nodes of the system where particularly large autocorrelations are expected.

The choice of x value is a critical design choice. A reasonable design rule is to choose the smallest value such that for long bitstreams, the KL divergence from an exact inference is smaller than the targeted threshold for an application (*e.g.*, 0.01). Therefore, in practical situations, the dominant error source is the input bitstream length, and accuracy can be chosen based on the input bitstream length, consistent with the basic principles of stochastic computing.

We should remark that the discussion of hardware overhead has heretofore considered only the rerandomization logic and memory; excluding the circular buffer, each rerandomization circuit requires an additional random number generator. For the counters and edge memory, the random number generator must randomly generate an x-bit binary number. This task is conventionally achieved with x unbiased single-bit random number generators. For the moving average and single run average circuits, the random number generator must provide a one-bit number with a particular probability with x-bit precision. This can be achieved with x unbiased single-bit digital random number generators and a small combinational logic circuit, or perhaps with a single analog random number generator.

The development of low energy random number generators is current a highly active area of research in the field of nanotechnology [25–30]. Novel devices and materials enable new types of random number generation that may be highly energy efficient. With continually reduced device dimensions, the energy barriers for switching between binary states can be reduced significantly [31,32]. Particularly exciting technologies include memristors [29,30], magnetic tunnel junctions [26–28], and single-electron transistors [25]. As these approaches continue to develop and improve, it is expected that the hardware costs will be small and have therefore not been included in the hardware analysis.

We have also assumed that the input bitstreams are neither biased nor autocorrelated. Our system is not resilient to input bias; any bias of the input bitstreams leads to bias in the output bitstreams. To remove bias, input bitstreams can be whitened through the use of XOR gates on several independent randomly generated bitstreams at the cost of additional area and energy consumption in bitstream generation. Input autocorrelation, on the other hand would be naturally improved by the ECEs, as shown during our analysis. In case of strong input autocorrelation, additional stages between the input and an input of probability 0.5 can also naturally suppress autocorrelation.

Given the disparate mechanisms underlying each of the rerandomization circuits, it is natural to consider the use of multiple different types of ECEs within a single Bayesian inference circuit. While numerous combinations can be conceived, a particularly compelling combination is the edge memory in concert with the single run average. As shown in Figs. 23 and 24, spam filter simulations were performed on the three-stage ECE tree for message E of Table 2, as in section 4.2.2. In Fig. 23, the four ECEs in the first stage are single run average circuits, the two ECEs in the second stage are edge memories, and the one ECE in the third stage is a single run average circuit. This organization is switched in Fig. 24, with four edge memories in the first stage, two single run average circuits in the second stage, and an edge memory in the third and final stage. For both mixed simulations, various bit length combinations are evaluated for both the single run average and edge memory – all single run average circuits have one particular bit length, and all edge memories have another bit length. The number of bits is noted in terms of the size of the memory; therefore, the edge memory sizes are written in terms of 2^x rather than x (*e.g.*, 64 edge memory bits corresponds to six edge memory bits in the notation of the simulations of section 4). In both graphs, the result from Fig. 13 for an ECE inference tree composed solely of single run average circuits is shown as a reference.

Figs. 23 and 24 interestingly show that the autocorrelation mitigation provided by a large edge memory can be traded off for that provided by a large single run average circuit. For example, the 256-bit edge memory in concert with a two-bit single average is seen in Fig. 23 to produce a result similar to the case of a 64-bit edge memory in concert with a four-bit single run average. The insertion of an edge memory generally provides superior behavior to a tree composed solely of single run average circuits. It is particularly noteworthy from Fig. 24 that with a sufficiently large edge memory (*e.g.*, 256 bits), the impact of the single run average circuits are negligible. This inspires further consideration of a tree in which some of the C-elements are simple C-elements rather than ECEs; however, it is likely that the performance and energy efficiency can be maximized with a balanced circuit composed of ECEs with similar hardware costs.

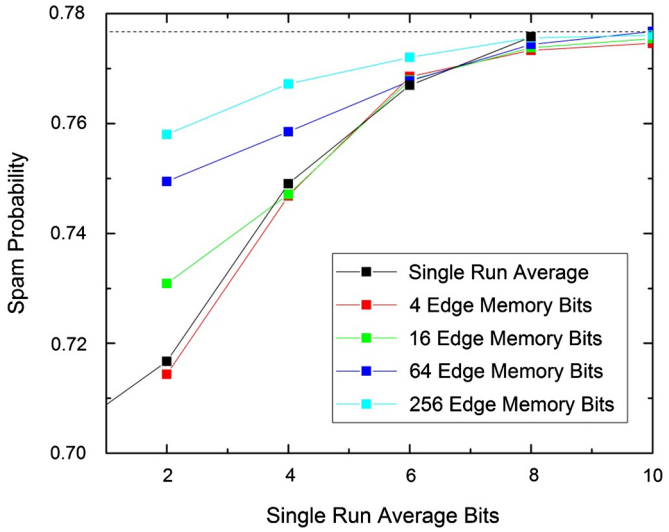

Fig. 23. Mixed simulation for spam message E with edge memories at stage two and single run average rerandomization circuits at stages one and three.

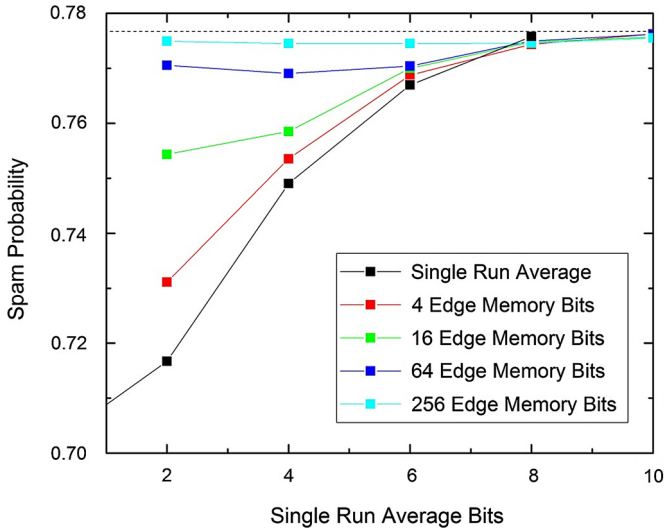

Fig. 24. Mixed simulation for spam message E with single run average rerandomization circuits at stage two and edge memories at stages one and three.

7. Conclusions

Circuits with simple C-elements can perform stochastic approximate Bayesian inference with exceptional efficiency benefits for inference tasks that do not require exact results. In this paper, we have shown that this approximate Bayesian inference scheme can be extended to provide a much higher degree of accuracy while retaining a five orders of magnitude efficiency advantage over conventional computing systems. In particular, the hardware analysis and inference simulation results indicate that the preferred solutions are counters or single-run average circuits with six-to-eight bits or circular buffers or edge memories with up to four bits.

These stochastic approximate Bayesian inference concepts are therefore promising techniques for the next generation of efficient and accurate autonomous robotic systems. Extending this stochastic approximate inference task further, our current work investigates the use of these extended C-elements in more sophisticated stochastic circuits for non-naïve Bayesian inferences.

Acknowledgement

The authors would like to acknowledge the contribution of Awais Aslam in the FPGA implementation.

References

[1] P. Bessière, C. Laugier, R. Siegwart, Probabilistic Reasoning and Decision Making in Sensory-Motor Systems, Springer-Verlag, Berlin, Heidelberg, 2008.
[2] W.J. Ma, J.M. Beck, P.E. Latham, A. Pouget, Bayesian inference with probabilistic population codes, Nat. Neurosci. 9 (Nov. 2006) 1432–1438.
[3] J. Laurens, J. Droulez, Bayesian processing of vestibular information, Biol. Cybern. 96 (Apr. 2007) 389–404.
[4] A. Houillon, P. Bessière, J. Droulez, The probabilistic cell: implementation of a probabilistic inference by the biochemical mechanisms of phototransduction, Acta Biotheor. 58 (Sep. 2010) 103–120.
[5] O. Lebeltel, P. Bessière, J. Diard, E. Mazer, Bayesian robot programming, Auton. Robots 16 (2004) 49–79.
[6] P. Bessière, J.-M. Ahuactzin, K. Mekhnacha, E. Mazer, Bayesian Programming, Chapman and Hall/CRC, ISBN 9781439880326, 2013, CAT# K13774.
[7] B. Vigoda, Analog Logic: Continuous-Time Analog Circuits for Statistical Signal Processing, 2003.
[8] I. Pournara, C.-S. Bouganis, G.A. Constantinides, FPGA-accelerated Bayesian learning for reconstruction of gene regulatory networks, in: FPL, 2005, pp. 323–328.
[9] M. Lin, I. Lebedev, J. Wawrzynek, High-throughput Bayesian computing machine with reconfigurable hardware, in: FPGA, 2010.
[10] P. Mroszczyk, P. Dudek, The accuracy and scalability of continuous-time Bayesian inference in analogue CMOS circuits, in: ISCAS, 2014, pp. 1576–1579.
[11] D. Querlioz, O. Bichler, A.F. Vincent, C. Gamrat, Bioinspired programming of memory devices for implementing an inference engine, Proc. IEEE 103 (8) (2015) 1398–1416.
[12] J.S. Friedman, L.E. Calvet, P. Bessière, J. Droulez, D. Querlioz, Bayesian inference with Muller C-elements, IEEE Trans. Circuits Syst. I 63 (6) (2016) 895–904.
[13] J. Von Neumann, Probabilistic logics and the synthesis of reliable organisms from unreliable components, Autom. Stud. 34 (1956) 43–98.
[14] B. Gaines, Stochastic computing systems, in: J.T. Tou (Ed.), Advances in Information Systems Science, 1969, pp. 37–172.
[15] D.E. Muller, Theory of Asynchronous Circuits, 1955.
[16] V. Gaudet, A. Rapley, Iterative decoding using stochastic computation, Electron. Lett. 39 (3) (2003).
[17] C. Winstead, S. Howard, A probabilistic LDPC-coded fault compensation technique for reliable nanoscale computing, IEEE Trans. Circuits Syst. II 56 (6) (Jun. 2009) 484–488.
[18] C. Winstead, C-element multiplexing for fault-tolerant logic circuits, Electron. Lett. 45 (19) (2009) 969–970.
[19] S. Sharifi Tehrani, C. Winstead, W.J. Gross, S. Mannor, S.L. Howard, V.C. Gaudet, Relaxation dynamics in stochastic iterative decoders, IEEE Trans. Signal Process. 58 (11) (2010) 5955–5961.
[20] S. Sharifi Tehrani, W.J. Gross, S. Mannor, Stochastic decoding of LDPC codes, IEEE Commun. Lett. 10 (10) (2006) 716–718.
[21] G. Tziantzioulis, A.M. Gok, S.M. Faisal, N. Hardavellas, S. Memik, S. Parthasarathy, b-HiVE: a bit-level history-based error model with value correlation for voltage-scaled integer and floating point units, in: DAC, 2015, p. 105.
[22] J.Y.F. Tong, D. Nagle, R. a Rutenbar, Reducing power by optimizing the necessary precision/range of floating-point arithmetic, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 8 (3) (2000) 273–286.
[23] G. Govindu, L. Zhuo, S. Choi, P. Gundala, V.K. Prasanna, Area and power performance analysis of a floating-point based application on FPGAs, in: Annu. High Perform. Embed. Comput. Work. (HPEC), MIT Lincoln Lab, 2003.
[24] S. Gupta, A. Agrawal, K. Gopalakrishnan, P. Narayanan, Deep learning with limited numerical precision, in: ICML, vol. 37, 2015.
[25] K. Uchida, T. Tanamoto, S. Fujita, Single-electron random-number generator (RNG) for highly secure ubiquitous computing applications, Solid-State Electron. 51 (2007) 1552–1557.
[26] W.H. Choi, Y. Lv, J. Kim, A. Deshpande, G. Kang, J. Wang, C.H. Kim, A magnetic tunnel junction based true random number generator with conditional perturb and real-time output probability tracking, in: IEDM, 2014, pp. 315–318.
[27] X. Fong, M. Chen, K. Roy, Generating true random numbers using on-chip complementary polarizer spin-transfer torque magnetic tunnel junctions, in: DRC, 2014, pp. 103–104.
[28] A. Fukushima, T. Seki, K. Yakushiji, H. Kubota, H. Imamura, S. Yuasa, K. Ando, Spin dice: a scalable truly random number generator based on spintronics, Appl. Phys. Express 7 (2014) 83001.
[29] S. Gaba, P. Sheridan, J. Zhou, S. Choi, W. Lu, Stochastic memristive devices for computing and neuromorphic applications, Nanoscale 5 (13) (Jul. 2013) 5872–5878.
[30] S. Balatti, S. Ambrogio, Z. Wang, D. Ielmini, True random number generation by variability of resistive switching in oxide-based devices, IEEE J. Emerg. Sel. Top. Circuits Syst. 5 (2) (2015) 214–221.
[31] R. Landauer, Irreversibility and heat generation in the computing process, IBM J. Res. Dev. 5 (3) (1961) 261–269.
[32] J.M.R. Parrondo, J.M. Horowitz, T. Sagawa, Thermodynamics of information, Nat. Phys. 11 (2) (2015) 131–139.