A Chaos-based Arithmetic Logic Unit and Implications for Obfuscation

Garrett S. Rose
Air Force Research Laboratory
Information Directorate
Rome, New York 13441 USA
email: garrettrose@ieee.org

Abstract—**It is no secret that modern computer systems are vulnerable to threats such as side-channel attack or reverse engineering whereby sensitive data or code could be unintentionally leaked to an adversary. It is the premise of this work that the mitigation of such security threats can be achieved by leveraging the inherent complexity of emerging chaos-based computing (computer systems built from chaotic oscillators). More specifically, this paper considers a chaos-based arithmetic logic unit which consists of many unique implementations for each possible operation. Generalizing to a chaos-based computer, a large number of implementations per operation can enable the obfuscation of critical code or data. In such a system, any two functionally equivalent operations are unique in terms of control parameters, power profiles, and so on. Furthermore, many possible implementations for each operational code can be leveraged to compile a program that is uniquely defined in terms of what the user knows–such knowledge which itself could be protected via encryption. The frequencies of the various operations are shown to approach that of a probabilistic system as the circuit is allowed to evolve in time. Further, the difficulty of a successful attack is assumed to be directly related to the number of unique op-code sets possible which is shown to grow exponentially with allowed evolution time for the proposed chaos-based arithmetic logic unit.**

Keywords-chaotic systems; logic design; arithmetic and logic units; integrated circuits; security

I. INTRODUCTION

Great strides have been made in recent years to leverage embedded computing resources that improve and simplify many of the things we do. However, the prevalence of computing has also brought along many challenges whereby we must trust that the machines we depend on can secure even our more sensitive information. To meet this challenge, many computer security solutions rely on complex cryptography to encrypt and thus secure sensitive data. However, even the strongest cryptographic techniques are susceptible to attacks via side-channel analysis (SCA) if the underlying hardware and computer architecture are not properly developed [1]. The possibility of side-channel attacks necessitates new computing architectures where all potential leaky channels are protected against the threat of attack.

A well-known SCA technique is differential power analysis (DPA) whereby the power traces of a computer system are monitored over time to obtain sensitive information [1]. It has even been shown that SCA techniques can be used to obtain the secret key from the popular AES encryption standard [2]. Given such vulnerabilities, several techniques exist to minimize the threat of potential side-channel attacks. For example, critical components can be implemented using sub-threshold CMOS [3] or dual-rail logic [4] to reduce the contribution of critical information (e.g. encryption keys) to the overall power dissipation. While such techniques are fairly effective at hiding critical information leaked through side-channels, such gains come either at the cost of significantly reduced performance or increased area overhead.

Code obfuscation techniques have also been developed and explored for mitigating side-channel attacks as well as reducing the threat of reverse engineering [5-6]. Such techniques typically work by inserting random delays constructed from dummy operations and shuffling the order of some instructions [6]. While software-level code obfuscation has been shown to render DPA attacks practically infeasible, the techniques often depend on complex compiler strategies. In contrast, the high complexity of potential operations available in a chaos-based computer system could provide adequate obfuscation due to the inherent complexity of the underlying hardware.

The use of chaotic dynamical systems for computation was first proposed in [7]. The early approaches focused on using chains of coupled chaotic elements to perform arithmetic operations. Later approaches focused more on the implementation of logic gates. A logic gate capable of implementing the $\overline{AB}$, $\overline{A+B}$, $\overline{A}$, and $A \oplus B$ functions using threshold controlled chaotic elements was proposed in [8]. This implementation used a single iteration of the logistic map. In the same paper, a gate using Chua's circuit [9] is described that can implement the functions $\overline{AB}$ and $\overline{A+B}$. A logic gate capable of implementing eight logic functions, AB, $A+B$, $\overline{AB}$, $\overline{A+B}$, $A \oplus B$, $\overline{A \oplus B}$, TRUE and FALSE, was proposed in [10]. This last chaotic gate design leverages multiple iterations of a chaotic oscillator, such as the logistic map, to allow for the implementation of any of the eight functions at different points in time and under different control conditions.

In this work, chaotic logic gates (chaogates) are used to construct a chaos-based arithmetic logic unit. The construction is such that the ALU will, once initialized, evolve through different arithmetic and bit-wise logic operations as the chaotic systems oscillate over time. It is important to point out that chaotic systems are sensitive to even subtle changes in their initial conditions such that the

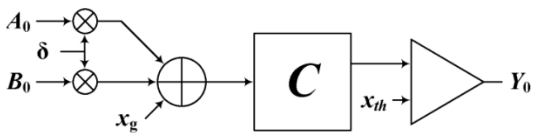

Fig. 1. Two-input chaogate built leveraging a single chaotic state variable. Control variables x_g, x_{th}, and δ determine how the functionality of the gate evolves with time.

state-space is increasingly complex as the oscillator evolves. For the chaos-based ALU, the complex state-space of individual chaogates leads to many different unique implementations for each operation. These different implementations in turn enable the construction of many possible op-code sets using only the ALU hardware as presented. The complexity of this chaos-based ALU and the many possible op-code sets that result are considered as handles in hardware for security applications such as code obfuscation and side-channel attack mitigation.

The remainder of this paper is organized as follows. Section 2 provides some background on chaotic computing and the chaogates considered in this work. The design of the proposed chaos-based ALU is presented in section 3. Section 4 provides some discussion on the implications of the chaos-based ALU for security. Finally, concluding remarks are provided in section 5.

II. Chaos-based Digital Logic

A one-dimensional chaotic logic gate was first described by Ditto et al. [10]. The gate uses a chaotic oscillator (e.g. Chua's function [9] or the logistic map) to determine the functionality of the logic gate at any given time. Consider the schematic of a one-dimensional chaotic logic gate as shown in Fig. 1. A generic chaotic oscillator is represented by the box labeled '*C*'. The signal going into the box from the left is the initial condition, and the output signal coming from the right side of the box is the state variable. The initial condition is formed by the sum of the analog control input x_g and two digital inputs *A* and *B*, each of which are converted to an analog value by multiplication with a constant weighting factor δ. The digital output *Y* is generated using the comparator with a threshold x_{th}.

The logic functionality of the gate can be altered by varying the control input x_g, weighting factor δ, or the threshold x_{th}. If the chaotic oscillator is seen as a pseudorandom number generator, then x_g and δ determine the seed and x_{th} converts the analog output into a digital 0 or 1 [10].

For a generic two-input reconfigurable logic gate, there are sixteen possible logic functions. However, for the one-dimensional chaotic logic gate the input sets {0,1} and {1,0} will produce the same initial condition such that the output for these two input sets will be identical. This restriction on the outputs reduces the number of possible logic operations the gate can perform to eight. These logic functions are as follows: FALSE, AB, $A+B$, $\overline{AB}$, $\overline{A+B}$, $A\oplus B$, $\overline{A\oplus B}$, TRUE.

A simple function that has been shown to exhibit chaotic behavior is the logistic map. The logistic map is the recurrence relation associated with the logistic function, often used to model population growth. The logistic map is typically written as:

$$x_{n+1} = rx_n(1-x_n), \tag{1}$$

where *r* is a positive real number and x_n is the state variable of this particular system. When the logistic map is used to represent population dynamics, *r* represents the combined rate for reproduction and starvation. The state variable x_n represents the ratio of the existing population to the maximum possible population for iteration *n*.

The logistic map is a simple system that acts as a useful example of deterministic chaos. Roughly speaking, a chaotic system is one that is very sensitive to even minute changes in initial conditions. Chaotic systems are also aperiodic in nature. These two properties combined lead to a system that is very difficult to predict without a perfect understanding of the system itself and all of its initial conditions. The logistic map is known to exhibit chaotic behavior when the coefficient *r* is between 3.6 and 4.

Fig. 2 illustrates the evolution of logic functions for a two-input chaogate (Fig. 1) that has been implemented with the logistic map. In this case the coefficient *r* is set to 4 such that the logistic map is chaotic. The different colors in Fig. 2

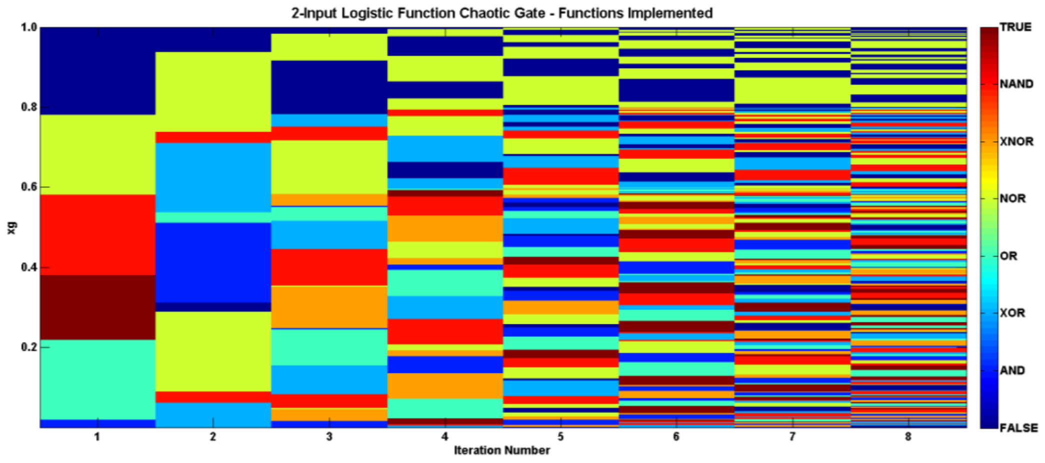

Fig. 2. Evolution of logic functions for a chaogate consisting of a logistic map chaotic oscillator [10].

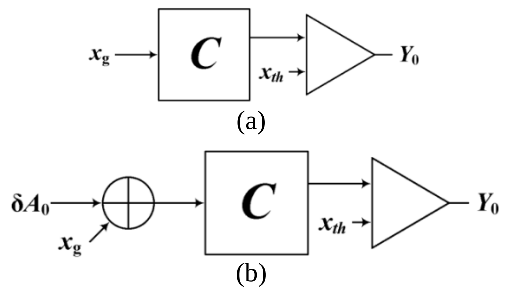

Fig. 3. (a) A zero-input chaogate initialized to only x_g and (b) a one-input chaogate initialized based on sum of x_g and scaled input A.

are used here to represent the eight different two-input logic functions possible with a chaogate constructed such as that shown in Fig. 1.

As can be seen in Fig. 2, different functions can be selected by controlling the variable x_g and the time of evolution or iteration number. It is worth noting the seeming unpredictability in the logic functions realized for any given combination of x_g and iteration number n. Furthermore, the behavior of the chaotic oscillator leads to a variety of unique implementations (in terms of x_g and n) for each possible logic function. It is this rich and yet difficult to predict state-space that can be leveraged to enhance the security of emerging computing systems.

III. A Chaos-based Arithmetic Logic Unit

Given the chaogate as described in [10] and in section 2 of this paper, it is possible to construct multi-input, multi-output logic blocks that are also chaotic in nature. Such chaos-based logic blocks would also exhibit a rich and difficult to predict state-space of multiple implementations of various possible logic operations. Thus, we will describe the construction of a chaos-based arithmetic logic unit consisting of conventional, combinational logic in addition to the chaogates described in section 2.

A. Different Types of Chaogates

Before constructing the chaos-based ALU two additional types of chaogates are considered: one with zero inputs and another that takes only a single digital input. These two additional types of chaogates are illustrated in Fig. 3.

Fig. 3a shows the zero-input chaogate which is initialized only according to the analog control value x_g. As will be discussed later, this zero-input chaogate can be considered almost like a pseudo-random bit generator as the likelihood that the digital output is a logic '1' approaches 50% after many iterations. The one-input chaogate shown in Fig. 3b is initialized based on the sum of x_g and the scaled digital input. Both of the chaogates described in Fig. 3 consist of a thresholding circuit (comparator) that will set the digital output to logic '1' if the oscillator's state variable is above x_{th} and to logic '0' otherwise.

B. An ALU with Chaogates

The proposed chaos-based ALU is illustrated in Fig. 4. All three chaogate types mentioned thus far (0-, 1-, and 2-input) are utilized in the construction of the ALU. Each bit

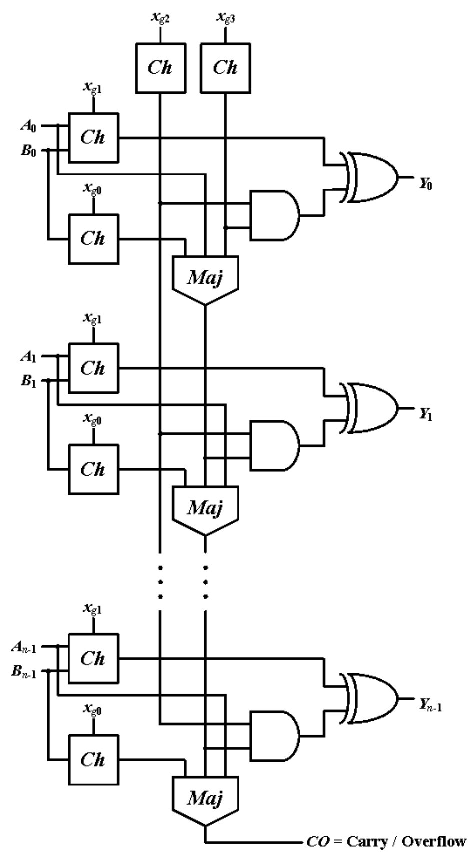

Fig. 4. Chaos-based ALU constructed with 2 selection chaogates for the entire ALU and 2 functional chaogates per bit slice. The gates labeled '*Maj*' are digital majority logic gates.

slice of this particular ALU is essentially a full-adder with the first XOR gate replaced by a two-input chaogate and a one-input chaogate placed between the input B and the majority logic gate used to produce the carry-out bit. Two zero-input chaogates are used to drive a carry-in bit (initialized by x_{g3}), and a second bit (initialized by x_{g2}) is used to control whether the carry path is considered in each bit-slice.

As this simple ALU design is based on a carry-ripple adder, addition or subtraction operations can be implemented when the two-input chaogates implement XOR or XNOR functions, respectively. The exact combination of x_{g1} and the iteration number to produce either an XOR or an XNOR can be determined based on the functional map shown in Fig. 2, assuming the chaotic oscillator is a logistic map.

As an example, for the ALU to implement subtraction the carry-in must be a logic '1', the input B must be inverted, and the carry chain must contribute to the output of each bit slice. The inversion of B is accomplished when the two-input chaogate is an XNOR (equivalent to $A \oplus \acute{B}$) and at the same time the one-input chaogate implements an inverter. The carry-in bit is generated by the zero-input chaogate driven by x_{g3}, which essentially acts like a pseudorandom bit generator. For subtraction, the carry-in bit being high leads to a twos complement of B, assuming all other necessary conditions are met with regards to the other chaogates. Similar

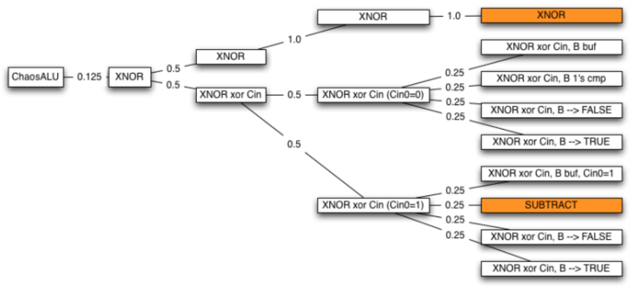

Fig. 5. Partial probability tree showing the necessary configurations to implement a subtraction operation and the bitwise operation XNOR. Based on this tree, XNOR operations occupy 6.25% of the state space while subtraction occupy 0.78% of the space.

reasoning stands for obtaining an addition operation except the two-input gate must be an XOR, the one-input chaogate a buffer, the carry-in bit a logic ‘0’, and the carry path control bit a logic ‘1’.

Bitwise operations are achieved based on the particular logic function implemented by the two-input chaogate and also the criterion that the carry chain be broken. Since the carry-bit and the one-input chaogate do not contribute to the bit slice outputs, bitwise operations will occur more often than the addition and subtraction operations. While eight logic functions are possible for the two-input chaogate only six may be of use as two functions are simply always TRUE and always FALSE.

C. *The Likelihood of Operations*

Before constructing the chaos-based ALU two additional types of chaogates are considered: one with zero inputs and another that takes only a single digital input. These two additional types of chaogates are illustrated in Fig. 3.

It has been mentioned that a chaotic oscillator can be used to construct a pseudo-random number generator. While not truly random, chaogate functionality can be approximated as a probabilistic system in order to aid in the design of chaos-based computing systems such as the ALU presented here. For example, the partial probability tree in Fig. 5 shows the likelihoods for the functions required for all of the chaogates in the ALU in order to implement either a subtraction or a bit-wise XNOR. Following the path to implement subtraction, it is clear the two-input chaogate must implement an XNOR (0.125 likelihood), the carry-in must contribute the summation output (0.5 likelihood), the initial carry-in bit must be a '1' (0.5 likelihood), and the one-input chaogate must implement an inverter (0.25 likelihood). Multiplying out the probabilities along the path leads to a roughly 0.78% likelihood that the chaos-based ALU in Fig. 4 will implement a subtraction operation. This does not mean that the ALU will randomly implement subtraction 0.78% of the time it is used. On the contrary, the chaos-based ALU is deterministic but this probability based model estimates that 0.78% of the ALU operations to appear in the overall state-space will be subtraction. Similarly, 6.25% of the ALU operations are estimated to be bit-wise XNOR.

Table 1 shows the “frequencies of operations” for the eight operations considered for the ALU described in Fig. 4. The frequency of operation represents the percentage of the

TABLE I. ALU FREQUENCIES OF OPERATIONS

Operation	Average Iteration	Median Iteration	Frequency of Operation (%)
ADD	7.87	8	0.77 %
SUB	10.44	13	0.44 %
AND	7.33	7	15.99 %
NAND	9.10	8	4.66 %
OR	7.84	8	15.77 %
NOR	8.20	8	5.32 %
XOR	9.15	10	13.31 %
XNOR	8.57	8	4.31 %

Frequency of operation represents the % of the state-space occupied by a particular operation. Each x_g value is two bits (256 values total) and the circuit can evolve for up to 16 total iterations. These values also consider δ as a 4 bit variable control value. For this example, 65, 536 distinct control configurations define the state-space.

state-space that is occupied by a particular operation; i.e. how many unique control combinations lead to the operation. In the sense that the addition and subtraction operation occur very rarely while the bitwise operations are more common, the results are much as predicted by the probabilistic model. However, there are outliers due to the fact that these numbers were generated by considering only 16 iterations of the chaotic oscillator. Since the number of iterations is small the frequencies of operation don’t exactly match estimates. Such descrepencies are expected as the system is not probabilistic but is based on deterministic chaos.

One final point worth consideration is that the frequency of operation can be used as a metric to guide the design of a chaos-based logic block. For example, it may be desirable to tailor the design of the ALU such that the frequencies of operations for arithmetic operations are more similar to their bit-wise logical counterparts. Such optimization could be accomplished by modifying design parameters such as the threshold (xth) and the scaling parameter (δ) shown in Figs. 1 and 3. The frequencies of operations predicted by the probability model for the chaos-based ALU could be a useful metric to guide a variety of optimization techniques. The details for the optimization of chaos-based computing systems such as the ALU in Fig. 4 is outside the scope of this particular work but is left for future endeavors.

IV. AN EXPONENTIAL NUMBER OF OP-CODE SETS

A motivating factor for chaos-based computing systems is that the complex nature of chaotic computing can lead to improved security. In the case of the chaos-based ALU, or even a chaos-based processor, this improved security could manifest itself in the form of many unique op-code sets. Having multiple op-code sets would be useful to mitigate issues such as side-channel attacks. Further, code obfuscation would benefit from an architecture that provides several unique implementations for each instruction such that unique and thus well obfuscated machine code can be generated.

In Fig. 6 we plot the number of unique op-code sets as a function of iteration number for the chaos-based ALU. For this work, the only operations considered are the eight listed in Table 1: ADD, SUB, AND, NAND, OR, NOR, XOR, and

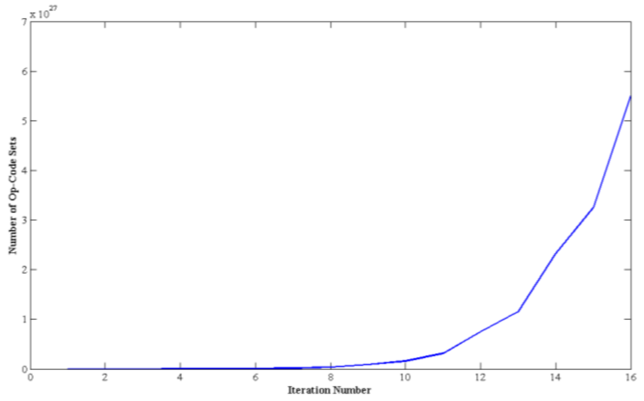

Fig. 6. The total number of possible op-code sets as a function of iteration number. If 16 iterations are allowed, the number of possible op-code sets is on the order of 10^{27} using *only* the hardware described in Fig. 4.

XNOR. The results presented in Fig. 6 demonstrate not only that several op-code sets, i.e. combinations of all possible operations, are possible but that the number of unique op-code sets increases exponentially with the iteration number of the system. This exponential increase in the number of op-code sets is related to the degree of difficulty for someone attempting to guess the functionality of some piece of code.

V. Conclusion

In this paper a novel chaos-based arithmetic logic unit is described. This chaotic ALU is constructed from chaogates that can evolve through different functional configurations. Given the complexity of the state space of a chaogate, it is shown that the state space of the chaos-based ALU is also complex and leads to an exponential number of possible op-code sets as the chaotic oscillators are allowed to evolve over time. It should be pointed out that the number of possible op-code sets, though large, is made possible with a minimal amount of hardware. The complexity of the control values used to generate different operations provides a useful architectural handle for code obfuscation.

Future work could include a study of the uniqueness of different op-code implementations and op-code sets. For example, as the proposed chaos-based ALU could also be useful for mitigating power analysis attacks, uniqueness in terms of power profiles could be considered. Since power considerations require a transistor level design of the system a study of the uniqueness in terms of power profiles is left for future work.

Acknowledgment

The author would like to thank Lok Yan, Bryant Wysocki, and Nathan McDonald of the Air Force Research Laboratory for interesting discussions on this topic. This work was supported by AFOSR LRIR 13RI05COR.

The material and results presented in this paper have been CLEARED (Distribution A) for public release, unlimited distribution by AFRL, case number 88ABW-2014-2101. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of AFRL or its contractors.

References

[1] P. Kocher, J. Jaffe, and B. Jun, “Differential power analysis,” in *Proceedings of Advances in Cryptology (CRYPTO)*, Lecture Notes in Computer Science, vol. 1666, Springer, pp. 388–397, 1999.

[2] K. Schramm, G. Leander, P. Felke, and C. Paar, “A Collision-Attack on AES,” in *Proceedings of Cryptographic Hardware and Embedded Systems (CHES)*, Lecture Notes in Computer Science, vol. 3156, Springer, pp. 163—175, 2004.

[3] S. I. Haider and L. Nazhandali, “Utilizing sub-threshold technology for the creation of secure circuits,” in *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 3182—3185, Seattle, WA, May 2008.

[4] K. Tiri, M. Akmal, and I. Verbauwhede, “A Dynamic and Differential CMOS Logic with Signal Independent Power Consumption to Withstand Differential Power Analysis on Smart Cards,” in *Proc. Eur. Solid-State Circuits Conf. (ESSCIRC)*, Florence, Italy, 2002, pp. 403–406.

[5] C. Linn and S. Debray, “Obfuscation of Executable Code to Improve Resistance to Static Disassembly,” in *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pp. 290—299, New York, NY, 2003.

[6] G. Agosta, A. Barenghi, and G. Pelosi, “A Code Morphing Methodology to Automate Power Analysis Countermeasures,” in *Proceedings of the ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 77—82, San Francisco, CA, June 2012.

[7] S. Sinha and W. L. Ditto, “Dynamics Based Computation,” *Phys. Rev. Lett.*, vol. 81, no. 10, pp. 2156—2159, Sep 1998.

[8] W. L. Ditto, K. Murali, and S. Sinha, “Chaos computing: ideas and implementations,” *Phil. Trans. R. Soc. A*, vol. 366, no. 1865, pp. 653—664, Feb 2008.

[9] L. O. Chua and G. Lin, “Canonical realization of Chua’s circuit family,” *IEEE Trans. Circuits Syst.*, vol. 37, no. 7, pp. 885—902, Jul 1990.

[10] W. L. Ditto, A. Miliotis, K. Murali, S. Sinha, and M. L. Spano, “Chaogates: Morphing logic gates that exploit dynamical patterns,” *Chaos*, vol. 20, no. 037107, Sep 2010.