Design and Implementation of Discrete-Time Chaos-Based Systems for Hardware Security Applications

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Aysha Siddique Shanta

August 2020

© by Aysha Siddique Shanta, 2020
All Rights Reserved.

I dedicate this work to my beloved father, Md Siddique Hossain, without whom this day would not have been possible and to my wonderful husband, Md Arifur Rahman, who stood by me through thick and thin.

Acknowledgments

I would like to thank my advisor Dr. Garrett S. Rose for giving me the opportunity to work in his research group and for providing guidance throughout my PhD career at University of Tennessee, Knoxville (UTK). I will always be indebted to him for believing in me when I was going through a rough patch in life.

I am thankful to Dr. Jayne Wu, Dr. Jinyuan Stella Sun and Dr. Hoon Hwangbo for serving on my PhD committee and dedicating their valuable time in helping and guiding me through the process.

I am extremely grateful to Dr. Hairong Qi and Dr. Leon Tolbert for their support and help during my time as a student at UTK. I would also like to thank Dr. Syed Kamrul Islam for giving me the opportunity to come to UTK for higher studies.

I would like to thank the Department of Electrical Engineering and Computer Science at University of Tennessee, Knoxville for their excellent service throughout the course of my degree. I would especially like to thank Dana Bryson, Linda Robinson and Melanie Kelley for helping me with useful information and reimbursement procedures.

I am grateful to have had the opportunity to work with wonderful people during my graduate studies. I would like to extend my gratitude to Md Sakib Hasan, Md. Badruddoja Majumder, Samira Shamsir, Mohammad Habib Ullah Habib, Khandaker Abdullah Al Mamun, Ava Hedayatipour, Shaghayegh Aslanzadeh, George Niemela, Sagarvarma Sayyaparaju, Mohammad Aminul Haque and Sherif Amer.

Finally, I would like to thank my family and friends for their support throughout different phases in life. These people made me the person I am today. This work would not have been possible without their endless love, support and kindness.

Abstract

Security of systems has become a major concern with the advent of technology. Researchers are proposing new security solutions in order to meet the area, power and performance specifications of the systems. The additional circuitry required for security purposes can consume significant area and power. This work proposes a solution which utilizes discrete-time chaos-based systems to address multiple hardware security issues. The nonlinear dynamics of chaotic maps is leveraged to build a system that mitigates IC counterfeiting, overbuilding, disables hardware Trojan insertion and enables authentication of connecting devices (such as IoT and mobile). Chaos-based systems are also used to generate pseudo-random numbers (PRN) for cryptographic applications.

The chaotic map is the building block for the design of discrete-time chaos-based oscillator. The analog output of the oscillator is converted to digital value using a comparator or analog-to-digital converter in order to build logic gates or generate PRNs. The chaos-based logic gate is reconfigurable since parameters in the circuit topology can be altered to implement multiple Boolean functions using the same system. The tuning parameters are iteration number, control input, bifurcation parameter and threshold voltage of the comparator.

The proposed PURCS system is a hybrid between standard CMOS logic gates and reconfigurable chaos-based logic gates where original gates are replaced by chaos-based gates. The system works in two modes: logic locking and authentication. In logic locking mode, the goal is to ensure that the system achieves logic obfuscation in order to mitigate IC counterfeiting. The secret key for logic locking is made up of the tuning parameters of the chaotic oscillator. Each gate has 10-bit key which ensures that the key space is large which exponentially increases the computational complexity of any attack. In authentication

mode, the aim of the system is to provide authentication of devices so that adversaries cannot connect to devices to learn confidential information. Chaos-based computing system is susceptible to process variation which can be leveraged to build a physical unclonable function (PUF). The proposed system demonstrates near ideal PUF characteristics which means systems with large number of primary outputs can be used for authenticating devices.

Table of Contents

1 Introduction **1**

1.1 Motivation and Research Goals 1

1.1.1 Chaos-based Logic Gates and Functionality Space 1

1.1.2 Random Number Generation 3

1.1.3 IC Counterfeiting 3

1.1.4 Authentication 5

1.1.5 Physically Unclonable and Reconfigurable System 6

1.2 Dissertation Overview 6

2 Background **7**

2.1 Introduction to Chaos Theory 7

2.1.1 Classification of Chaotic Map 8

2.1.2 Study of Chua's Circuit 8

2.2 Introduction to Chaos Computing 11

2.2.1 Continuous-time Nonlinear System 13

2.2.2 Discrete-time Nonlinear System 16

2.2.3 Digital Logic Obtained by Varying Iteration Number 19

2.2.4 Distinguishing Between (0,1) and (1,0) Input Pairs 20

2.2.5 Design of Multi-Input Multi-Output Logic Functions 21

2.3 Concluding Remarks 22

3 Design of Chaotic Oscillator and Chaos-Based Logic Gate Using Three Transistor Chaotic Map **24**

3.1 Design of Chaotic Map . . . 25
3.1.1 DC Transfer Characteristics of the Map . . . 26
3.2 Design of Chaotic Oscillator . . . 27
3.2.1 Iterating through the Map . . . 30
3.2.2 Sensitivity to Initial Condition . . . 30
3.2.3 Ergodicity of the Chaotic Map . . . 30
3.2.4 Bifurcation Diagram . . . 31
3.2.5 Lyapunov Exponent . . . 31
3.3 Reconfigurable Chaos-based Logic Gates . . . 33
3.3.1 Introduction . . . 33
3.3.2 Design of Reconfigurable Chaos-based Gates . . . 35
3.3.3 Complex Functions Obtained Using Single Chaotic Element . . . 40

4 Expansion of Functionality Space Using Three Transistor Chaotic Map 45
4.1 Introduction . . . 45
4.2 Expansion of Design Space . . . 46
4.2.1 Comparison of Area and Power Overhead . . . 47
4.2.2 Design Space Enhancement . . . 49
4.3 Application . . . 52

5 Four Gate Transistor Negative Differential Resistance (NDR) Based Discrete-Time Chaotic Map 53
5.1 Background . . . 53
5.2 Four Terminal Transistor (G^4FET) . . . 54
5.2.1 G^4FET Operation . . . 56
5.3 G^4FET Based Negative Differential Resistance . . . 59
5.4 G^4NDR Based Chaotic Map . . . 61
5.4.1 Design of Chaotic Oscillator Using G^4NDR Based Map . . . 62
5.4.2 Bifurcation Diagram and Lyapunov Exponent . . . 62
5.5 Design of Logic Gates Using G^4NDR Based Map . . . 63
5.6 Expansion of Design Space Using G^4NDR Map . . . 63

6 Pseudo-Random Number Generation (PRNG) Using Three Transistor Chaotic Map **71**

6.1 Introduction . . . 71

6.2 True Random Number Generator (TRNG) . . . 72

6.3 Pseudo-Random Number Generator (PRNG) . . . 73

6.3.1 Linear PRNG . . . 74

6.3.2 Nonlinear PRNG . . . 74

6.4 Proposed Lightweight and Reconfigurable PRNG . . . 75

6.4.1 Chaotic Oscillator for PRNG Design . . . 76

6.4.2 Correlation Coefficient . . . 79

6.4.3 Seed Sensitivity . . . 81

6.4.4 National Institute of Standards and Technology Tests . . . 81

6.5 Overhead Analysis . . . 84

6.6 Application Lies in Security of IoT Devices . . . 86

7 Physically Unclonable and Reconfigurable Computing System (PURCS) **87**

7.1 IC Counterfeiting and Logic Locking . . . 87

7.1.1 IC Counterfeiting . . . 87

7.1.2 Background of Logic Locking . . . 89

7.1.3 Threat Model for Logic Locking . . . 90

7.1.4 IC Design Flow . . . 90

7.1.5 Issues Mitigated by Logic Locking . . . 91

7.1.6 Types of Logic Locking . . . 91

7.1.7 Techniques of Inserting Key Gates . . . 93

7.1.8 Types of Key Gates . . . 93

7.2 Authentication and Physical Unclonable Functions (PUFs) . . . 96

7.2.1 Authentication of Devices . . . 96

7.2.2 Types of Authentication Architectures . . . 97

7.2.3 Process variation . . . 98

7.2.4 Background of Physical Unlconable Functions (PUFs) . . . 98

7.2.5 Classification of PUFs . . . 100
7.2.6 Chaos-based PUFs . . . 104
7.2.7 Authentication Protocol . . . 104
7.2.8 PUF Metrics . . . 106
7.3 Proposed Computing System (PURCS) . . . 107
7.3.1 Design of Chaotic Oscillator for PURCS System . . . 108
7.3.2 Bifurcation Diagram . . . 108
7.3.3 Chaos-Based Logic Gate Implementation . . . 110
7.3.4 Characterization of the Chaos-based Logic Gate . . . 112
7.3.5 Functions Generated by the Logic Gate in Different ICs . . . 113
7.3.6 Reliability of the Functions Generated from Logic Gate . . . 113
7.3.7 Creating the Characterization Table . . . 115
7.3.8 Replacement Algorithm . . . 117
7.3.9 Configuration Key of the Hybrid Circuit . . . 119
7.4 Simulation and Results . . . 121
7.4.1 Logic Locking Results . . . 121
7.4.2 Authentication Results . . . 123
7.5 Security Performance . . . 125
7.5.1 Security of Logic Locked Circuits . . . 125
7.5.2 Security of PUFs . . . 133
7.6 Overhead Analysis . . . 136
7.7 Modes of Operation . . . 137
7.7.1 Logic Locking Mode . . . 139
7.7.2 Authentication Mode . . . 139

8 Contribution and Future Work **140**
8.1 Original Contribution . . . 140
8.2 Future Work . . . 142

Bibliography **143**

Appendices **164**

A Random Number Generator 165

A.1 SKILL Code for Modeling the Chaotic Oscillator 165

A.2 Matlab Code for Plotting the Bifurcation Diagram 169

A.3 Matlab Code for Plotting the Lyapunov Exponent 171

A.4 Matlab Code for Generating the Random Numbers 175

B PURCS System 183

B.1 SKILL Code for Modeling the Chaotic Oscillator 183

B.2 Matlab Code for Creating the Characterization Table 185

B.3 SKILL Code for Generating Data for Monte Carlo Simulation 186

B.4 Python Code for Calculating the Controllability 192

B.5 Python Code for Calculating the Observability 195

B.6 Python Code for Calculating the Testability 200

B.7 Python Code for Solving a Given Netlist 204

B.8 Python Code for Replacing the Gates Based on Testability 207

B.9 Python Code for Calculating the Hamming Distance 209

Vita **218**

List of Tables

2.1 Necessary and sufficient conditions to obtain different logic functions from the nonlinear dynamical element [43]. 17

2.2 One specific solution to yield different logic functions using logistic map [43]. 18

2.3 Necessary conditions to implement logic functions from nonlinear systems for varying iterations [43]. 19

2.4 Necessary and sufficient conditions to build half-adder from the nonlinear dynamical element [43]. 22

3.1 Evolution of chaotic oscillator output with number of iterations. (V_c= 690 mV, $C = (111010)_2 = 58$, $V_{th} = 1.03\ V$, $n = 5$) [63]. 42

3.2 Different configurations for 3-input 1-output instructions [63]. 42

3.3 Design of 1-bit full-adder using chaotic oscillator (V_c= 677.5 mV and $C = (011001)_2 = 25$). 43

3.4 Design of 3×8 decoder using chaotic oscillator (V_c= 710 mV and $C = (011001)_2 = 25$). 44

4.1 Overhead comparison of proposed work with previous work [150]. 48

5.1 Evolution of analog output from chaotic oscillator with different iterations ($\mu_1 = 0.95\ M\Omega$, $\mu_2 = 0$ V, $\mu_3 = 0\ V$, $C_b = 0$, $V_{th} = 1.25\ V$). Functions are represented in decimal value. 70

5.2 Three different configurations for six logic functions. 70

6.1 Correlation coefficient between the original sequence and the three cases. . . 82

6.2 NIST test results of the proposed PRNG [148]. 85

6.3 Overhead comparison of established PRNGs with the proposed design [148]. 85

7.1 Evolution of chaotic oscillator output with number of iterations. (V_c= 552 mV, $C_b = 1$, $V_{th} = 625\ mV$) [149]. 111
7.2 Characterization table of the 2-input reconfigurable chaos-based logic gate for $V_{th} = 625\ mV$ [149]. 111
7.3 Characterization table of the reconfigurable chaos-based logic gate in different chips for $V_{th} = 625\ mV$ [149]. 114
7.4 Percentage of gates replaced to reach almost 50% Hamming distance in ISCAS'85 benchmark circuits [149]. 122
7.5 Results of standard PUF metrics for the benchmark circuits using testability based replacement method [149]. 124
7.6 SAT attack on a logic locked circuit shown in Fig. 7.16b. [182]. 130
7.7 Result of machine learning based modeling attacks on PURCS system using testability based replacement method [149]. 138
7.8 An estimate of transistor count in different logic locking schemes [149]. . . . 138
7.9 Estimation of transistor count for both PUF and logic locking in a system with 30-bit key/challenge [149]. 138

List of Figures

2.1 Chua's autonomous circuit [108]. 9
2.2 Double scroll attractor using Chua's circuit. 9
2.3 Chaotic Chua's attractor implemented in circuit to build logic gates [115]. . . 16

3.1 Three transistor chaotic map circuit designed in 65 nm process [150, 46]. . . 25
3.2 DC transfer characteristic of the three transistor map circuit [150]. 26
3.3 Traditional chaotic oscillator using buffer [85]. 28
3.4 Chaotic oscillator using two map circuits [150, 46]. 28
3.5 Iterating through the chaotic map. 28
3.6 Sensitivity of the chaotic oscillator to slight change in initial condition. . . . 32
3.7 (a)-(d): Sampled output of chaotic oscillator for $V_{in} = 750\ mV$ showing different states as V_c is changed (a) Period 2 (b) Period 4 (c) Period 8 (d) Chaotic region. 32
3.8 Bifurcation diagram of the chaotic oscillator [148]. 34
3.9 Lyapunov exponent of the chaotic oscillator [148]. 34
3.10 A basic 2-input chaogate using a single chaotic oscillator [139]. 34
3.11 Chaotic oscillator is shown in the dotted box. Two-input chaos-based logic gate including digital encoding of the inputs and digital decoding of the outputs [86]. 36
3.12 Two-input reconfigurable chaos-based logic gate demonstrating all the tuning parameters in red circles. 36
3.13 Three-bit DAC for encoding digital inputs to analog values [86]. 38
3.14 Decoder which converts final states to digital outputs [86]. 38

3.15 Multi-input multi-output reconfigurable chaos-based logic gate [63]. 40
3.16 1-bit full-adder designed using chaotic oscillator [63]. 42
3.17 3×8 decoder designed using chaotic oscillator [63]. 43

4.1 Transient response of the chaotic oscillator displaying that V_c is changed cycle to cycle [150]. 48
4.2 Comparison between the functionality space of the proposed work with existing work for varying number of iterations [150]. 51
4.3 Number of individual functions increases linearly with the total functionality space [150]. 51

5.1 Four gate transistor (G^4FET) (a) Structure (b) Symbol [7]. 55
5.2 Cross-section of the G^4FET device [64]. 55
5.3 Traditional 2-terminal JFET NDR [163]. 55
5.4 Tunable NDR made with n- channel and p- channel G^4FETs (a) Circuit (b) Symbol [9]. 58
5.5 Transfer characteristics of G^4NDR. 58
5.6 G^4NDR based discrete-time chaotic map which has three bifurcation parameters (a) Circuit (b) Symbol. 60
5.7 Transfer curve of the G^4NDR map circuit for varying μ_1. 60
5.8 G^4NDR based chaotic oscillator using buffer in the feedback. 64
5.9 G^4NDR based chaotic oscillator using another G^4NDR map in the feedback. 64
5.10 (a) Bifurcation diagram and (b) Lyapunov exponent of the chaotic oscillator for varying μ_1; $\mu_2 = 0\ V$, $\mu_3 = 0\ V$. 65
5.11 (a) Bifurcation diagram and (b) Lyapunov exponent of the chaotic oscillator for varying μ_2; $\mu_1 = 1\ M\Omega$, $\mu_3 = 0\ V$. 66
5.12 (a) Bifurcation diagram and (b) Lyapunov exponent of the chaotic oscillator for varying μ_3; $\mu_1 = 1\ M\Omega$, $\mu_2 = 0\ V$. 67
5.13 G^4NDR based reconfigurable logic gate. 67
5.14 Comparison of functionality space among proposed and previous works. . . . 70

6.1 4-bit linear feedback shift register [151]. 73
6.2 Chaotic oscillator. The dotted lines represent the chaotic map [148]. 78
6.3 Proposed pseudo-random number generator [148]. 78
6.4 Distribution of correlation coefficient. 80
6.5 Seed sensitivity of the PRNG for slight changes in the seed. 80

7.1 IC design flow with logic locking capabilities [132]. 91
7.2 (a) Unlocked circuit (b) Locked circuit with three XOR/XNOR gates [132]. 92
7.3 Arbiter PUF [54]. 103
7.4 Generic strong PUF based authentication protocol [161]. 103
7.5 Reconfigurable 2-input chaos-based logic gate designed using chaotic oscillator, comparators and DAC [149]. 109
7.6 Transient response of the chaotic oscillator [149]. 109
7.7 Number of AND, OR and XOR functions in 10 chips [149]. 114
7.8 Effect of temperature variation on the oscillator output ($V_{in} = 193.2\ mV$ and $V_c = 490\ mV$). 116
7.9 Effect of supply voltage variation on the oscillator output ($V_{in} = 193.2\ mV$ and $V_c = 520\ mV$). 116
7.10 Algorithm for replacing gates using testability method [149]. 120
7.11 (a) Unobfuscated circuit from ISCAS'85 benchmark suite (b) Obfuscated circuit where two NAND gates are replaced with reconfigurable chaos-based logic gates and the secret key is 20 bits long [149]. 120
7.12 Hamming distance vs. percentage of gates replaced in ISCAS'85 benchmark circuits (a) Gates replaced randomly (b) Gates replaced by measuring testability [149]. 122
7.13 Post-processing scheme for authentication [149]. 124
7.14 Circuit to determine distinguishing input patterns (DIPs) [182]. 127
7.15 Logic locking example (a) Original circuit (b) Locked circuit [160]. 127
7.16 Logic locking second example (a) Original circuit (b) Locked circuit [182]. . . 130

7.17 SAT attack implemented on C432 benchmark circuit for varying percentage of gates and key sizes [149]. 130
7.18 (a) Logic locking mode (b) Authentication mode. 139

Chapter 1

Introduction

Hardware is a fundamental and root of trust component in any security system. Recently, many software-based security solutions have been migrated to hardware in order to mitigate security threats. All cryptographic protocols rely on two basic assumptions: 1) read-proof hardware that prevents the adversary from learning the contents stored 2) tamper-proof hardware that prevents the adversary from altering the information stored. Existing cryptographic algorithms are stored in tamper-proof hardware which the adversary has knowledge of, but cannot change the content. There is also a secret key involved which is stored in read-proof as well as tamper-proof hardware which the adversary does not know about and cannot change either. It has been brought into light that hardware is also vulnerable to security threats and that adversaries can learn information from the systems using reverse engineering, testing and side-channel analysis.

1.1 Motivation and Research Goals

1.1.1 Chaos-based Logic Gates and Functionality Space

In the 1960s, Gordon Moore predicted that the number of transistors in a chip will double every two years. This hypothesis has been true for a long time. As technology is scaling, designers are able to integrate more transistors in an IC but it comes at a cost of short channel effects and heat production in a chip due to increased density of transistors per

square area. A solution is to build gates that can implement multiple functionalities using the same system by altering the circuit parameters.

Researchers have studied chaos theory for the past few decades and proposed to build a chaos-based computing system which utilizes chaos-based logic gates. These logic gates are reconfigurable and capable of implementing multiple functions by tweaking different parameters in the system. This feature of chaos-based logic gates can be leveraged to reduce the number of gates in a system. Chaos-based logic gates are comparable to look-up tables (LUTs). The hardware required for increasing the number of inputs scales exponentially in LUTs due to usage of SRAMs, MUXs and inverters. In chaos-based logic gates, the only component that needs to be scaled for increased input is the digital-to-analog-converter (DAC) which is used at the input of the chaotic oscillator and the rest of the system remains unchanged. It is also possible to design complex functions (multi-input multi-output) such as adder, subtractor, decoder and encoder using the complex dynamical systems.

Chaos-based logic gates have additional overhead compared to conventional CMOS logic gates. One way to justify the additional hardware cost is to ensure that a single chaos-based logic gate possesses a huge functionality space. Here, functionality means the Boolean function as well as the configuration of the chaos gate in order to achieve that function. The functionality space is large if there are numerous ways to execute the same Boolean function.

This work explores two types of discrete-time chaotic maps, three transistor map and a G^4NDR based map. The goal is to design a chaos-based logic gate which has multiple tuning parameters to change the logic function of the gate. The parameters are bias voltages (bifurcation parameters), threshold voltage, control bit and iteration number. Bias voltages are chosen from the chaotic region using the bifurcation diagram. Changing the bias voltage in each cycle of the chaotic oscillator can lead to an increase in the entire functionality space and number of individual functions with respect to iteration number. The three transistor map has one bifurcation parameter whereas the G^4NDR map has three independent bifurcation parameters. The three bifurcation parameters enhance the functionality space without using a very high value of iteration number.

The increase in functionality space can be applied to security of computing systems against power analysis based side channel attack. The attacker learns the power profile of

the executed instructions on the processor and tries to decipher the code by making an educated guess based on the collected data. Chaos-based logic gates can help in mitigating this attack since different functions can be executed using the same reconfigurable block and the power signature of different instructions will be difficult to classify.

1.1.2 Random Number Generation

Good quality random number generators (RNG) are required for cryptographic applications. Linear RNGs repeat after a short period which makes it unsuitable for security applications. If the period needs to be extended then the circuit design becomes very complex and performance overhead increases. Chaotic systems are nonlinear dynamical systems which maps inputs to outputs in a deterministic but random manner. A lot of research has been done on using chaotic systems such as logistic map, tent map and sine map to be used as a pseudo-random number generator (PRNG). These maps are mathematical equations which can require a handsome amount of hardware to be implemented in an IC.

The goal of this work is to use a simple three transistor chaotic map which portrays chaotic behavior because it has non-monotonic transfer characteristics. A chaotic oscillator is used to map the inputs to outputs. The output of the oscillator is analog so the output is converted to digital value using 10-bit ADC. The least significant bit (LSB) or the 10^{th} bit of the ADC is used since it has the most entropy. A single chaotic oscillator is not sufficient for the sequence to pass all the NIST tests. Two chaotic oscillators are used and the outputs are XORed to ensure that the generated sequence passes the NIST tests and that the sequence can be used for security applications. G^4NDR based discrete-time chaotic map can also be used to generate random numbers.

1.1.3 IC Counterfeiting

Silicon fabrication facilities are very expensive to build, maintain, operate and manage due to which many U.S. high-tech companies have gone fabless in recent years. In 2015, the cost of building a new semiconductor fab was approximately calculated to be $5 billion dollars along with large recurring maintenance costs [185]. The cost will keep on increasing as the

technology scales down to smaller technology nodes. This has led to the globalization of the Integrated Circuit (IC) supply chain and the fabrication of the chips are outsourced to multiple facilities in the design flow process. However, the globalization of IC design is making it easier for adversaries in the supply chain to overbuild ICs, pirate ICs and add hardware Trojans to the design. The semiconductor industry loses $4 billion dollars annually due to counterfeiting issues [87, 78].

Researchers have proposed various solutions to mitigate counterfeiting such as watermarking, split manufacturing, IC camouflaging and logic locking. Among all these techniques, logic locking has gained the most attention because it can protect the IC content throughout the entire IC design flow. Logic locking adds extra gates to the IC design in order to hide the original functionality of the design. If conventional XOR/XNOR gates are added, then the attacker can reverse-engineer the layout in order to figure out the netlist. The attacker can apply different keys and match it with the outputs of a functional IC bought from the market. The XOR gates are just standard gates added to the design and attackers can utilize techniques such as FIBing to bypass the XOR logic.

The goal of this work is to provide logic obfuscation by replacing some standard CMOS gates with chaos-based logic gates. The chaos-based logic gates will be difficult to bypass because they replace original circuitry. Moreover, each chaos-based logic gate has 10-bit key whereas each XOR gate can only provide 1-bit key. The 10-bit key is made up of the tuning parameters in the circuit topology such as iteration number, control bit, bifurcation parameter and threshold voltage of the comparator. Hamming distance between correct and wrong output is measured and the aim of the design is to ensure that half of the output bits are corrupted. In order to achieve 50% Hamming distance with lower overhead, the chaos-based logic gates are replaced using testability based method. Boolean SAT attack has been implemented on a single benchmark circuit and the attack complexity increases exponentially due to increase in number of replaced gates and increase in key size for a fixed percentage of gates replaced.

1.1.4 Authentication

Due to technology advancement, smart devices have become a major part of everyone's life. Application of smart devices can range from mobile phones to smart cities. All the devices are connected to each other and it is essential to ensure that information is being exchanged between two authentic parties. Authentication of devices is important otherwise attackers can steal confidential information such as passwords. The devices are usually used in an untrusted zone and the adversary might have physical access to the system. The solution to saving confidential information is to install a key unique to each device which can be achieved by using a physical unclonable function (PUF).

PUFs demonstrate unique characteristics in each IC due to the process variation in the manufacturing process. Each device will respond differently to inputs because of slight changes in the internal characteristics of the device due to fabrication procedure. The input to a PUF is called "challenge" and the output is called "response". If the PUF has many challenge-response pairs, then it is impossible for the attacker to model the PUF. Even the manufacturer will not be able to replicate the same PUF if the same manufacturing process is executed.

Chaotic systems are vulnerable to small changes in the initial condition. During the manufacturing process, the internal characteristics of the circuit will be slightly altered which means that the same circuit in different ICs will respond differently to inputs. The hybrid system of CMOS gates and chaos-based logic gates, discussed earlier, can be used as a PUF. The chaotic oscillator will map the current states to future states in a unique manner in each IC due to process variation. The output of the system requires post-processing to be used for authentication purposes. The post-processing technique adds very little overhead to the design. The system demonstrates ideal PUF characteristics such as uniqueness, bit-aliasing and uniformity. The immunity of the hybrid system has been tested against common machine learning based attacks.

1.1.5 Physically Unclonable and Reconfigurable System

A single computing system is proposed that can provide both authentication and logic locking. It is possible to provide both features because chaos-based logic gates are used in the design. Each chip needs to be characterized after fabrication since process variation will change the functionality of the system. The correct keys can be stored in a tamper-proof memory for correct functionality in logic locking mode. In authentication mode, the keys and primary inputs make up the "challenges" to the system. The "challenges" are provided from the verifier using the CRP table in the authentication protocol. The PURCS system behaves like PUF due to the unclonability feature present due to the effects of process variation. The overhead of the system is significantly lower compared to a system that contains hardware for both logic locking and authentication purposes.

1.2 Dissertation Overview

The dissertation is organized as follows. Chapter 2 introduces chaos theory and how chaos can be used to make computing systems. Chapter 3 discusses the three transistor discrete-time chaotic map and oscillator and focuses on features such as bifurcation diagram, Lyapunov exponent and transfer characteristics of the map. The functionality enhancement technique of the three transistor chaotic system is discussed in chapter 4. In chapter 5, a novel G^4NDR based discrete-time chaotic map is introduced and its functionality space is explored. Pseudo-random number generator designed using the three transistor chaotic map is discussed in chapter 6. Chapter 7 elaborates on how a single system containing chaos-based logic gates can be utilized to obtain both authentication of devices and logic obfuscation to mitigate counterfeiting. Chapter 8 contains the contribution of the entire dissertation and discusses the future direction of the proposed work.

Chapter 2

Background

2.1 Introduction to Chaos Theory

Chaos has generated a lot of research interest in the past few years. Chaos is present in many disciplines such as engineering, biology, chemistry, medicine and physics. Chaotic phenomena is found in electronic circuits, lasers, chemical systems, hearts and brains. The discovery of chaos along with quantum mechanics and theory of relativity are considered the three monumental scientific findings of the twentieth century. Lorenz described chaos as a deterministic non-periodic flow of a system's state through its entire state space [100]. Chaotic behavior is present in a range of natural and made-made systems starting from weather to integrated circuits. Chaotic systems have two important features:

1. High sensitivity to initial conditions.

2. The behavior is apparently random but deterministic.

Chaotic systems are hard to predict, difficult to study and change their behavior very fast over time. Chaotic systems are aperiodic in nature and even though they are finite, they never converge to any value. Chaotic behavior of systems open up new areas of applications in the field of engineering, annealing noise of networks, chaos-based communication systems and chaotic neural networks [72, 69]. Design of chaos-based systems should have a simple structure and most importantly they should be compatible with standard CMOS technologies. Deterministic chaos can be used to implement electronic

random signal generators by leveraging their sensitivity to initial conditions which is also known as the butterfly effect. A chaotic system is ideal for encryption schemes since it is extremely difficult to predict their long-term behavior [11, 12].

2.1.1 Classification of Chaotic Map

Chaotic maps can be classified into two types.

One-dimensional (1D) Chaotic Map

One-dimensional chaotic maps are mathematical systems that capture the evolution of a single variable in discrete time. Examples of 1D map are Gauss map, logistic map and tent map. 1D maps are easy to implement and have simple structures. However, they suffer from some weaknesses such as small parameter number, the outputs are predictable with low reverse-engineering cost and their dynamic range is limited.

High-dimensional (HD) Chaotic Map

High-dimensional (HD) maps are mathematical systems which model the evolution of at least two variables in discrete time. Examples of HD maps are Lorenz system, Henon map, Chen and Lee system and hyperchaotic systems. HD maps have better chaotic performance and their chaotic sequences are harder to predict in comparison to 1D maps. However, HD maps are difficult to implement in hardware and they incur high computational cost. This limitation does not allow designers to use them in real-time applications.

2.1.2 Study of Chua's Circuit

The study of the complex behavior of nonlinear dynamical systems can be easily done by using Chua's autonomous circuit as shown in Fig. 2.1. Chua's circuit was the first chaotic system that has been derived [32], diligently studied [31] and physically confirmed [108]. The simple circuit can be used to study the characteristics of dynamical systems such as period doubling, bifurcation, chaos and attractors. Chua's circuit represents a continuous-time dynamical system. The circuit is made up of a capacitor, C_2 and inductor, L in

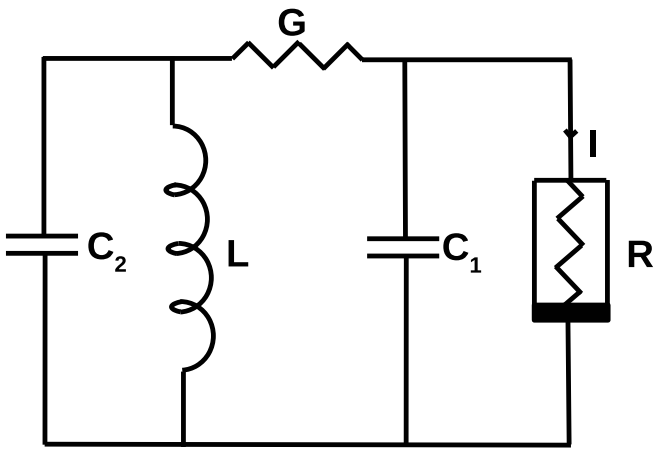


Figure 2.1: Chua's autonomous circuit [108].

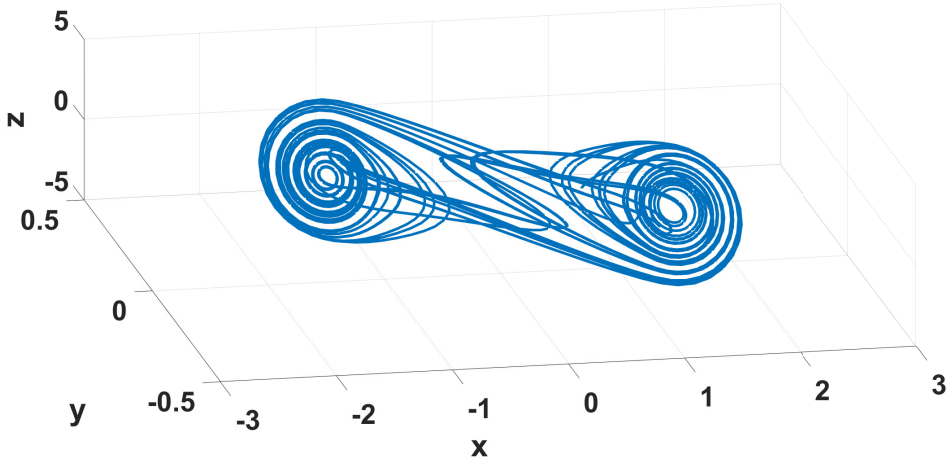


Figure 2.2: Double scroll attractor using Chua's circuit.

parallel which makes up the lossless resonant component. The conductance, G provides the coupling between nonlinear resistor, R and the capacitor, C_1 in parallel with it. The transfer characteristic of the nonlinear resistor is a three-segmented piecewise curve.

The nonlinear element, R can be designed in several ways in practice. It has been designed with a single operational amplifier (op-amp), two diodes and resistors [109], two bipolar transistors, two diodes and resistors [31] and two op-amps and six resistors [84]. Initially, the system demonstrates a transient response by finally settling into a steady-state response. The steady-state response is also known as the *attractor* of the system which means that the passive part of the system can have many different initial conditions but eventually the system converges to the same steady-state behavior. The control parameter in the circuit is the conductance, G and small or large changes in its values causes the current and voltage of the passive elements to portray the same values after a certain period. The period of repetition of the values is the same as the driving force so the system is called a period-one attractor. As the value of the conductance increases, the period of repetition doubles from the period-one solution and the system displays a discontinuous change in the response known as bifurcation.

Period doubling is a universal phenomenon in nonlinear dynamical systems. The same behavior can be observed in many physical systems such as the forced movement of pendulum. Period doubling cascade is a common behavior of chaos-based systems. In any chaotic circuit, it can be seen that period-two doubles to period-four and period-four doubles to period-eight. Small changes in the control parameter can create an endless series of period doublings representing all the characteristics of the chaotic attractor. An interesting property of chaos is that infinite number of periodic solutions from all periods exist with chaos. Slight tuning of the control parameter can result in absence of chaos and instead a periodic solution appears. The chaotic attractor can suddenly reappear after the conductance is changed minutely. This behavior is known as the appearance of periodic window [70]. The existence of the periodic windows prove that a chaotic solution is available rather than noise. A demonstration of double scroll chaotic attractor using Chua's circuit is shown in Fig. 2.2.

2.2 Introduction to Chaos Computing

Chaos-based systems generate sequences and their sensitivity to initial conditions allows them to switch between different sequences exponentially fast. Even lower dimensional chaotic systems can exhibit a variety of behaviors depending on their initial condition, function of time or parameters. Chaos-based systems are popular for their rich temporal behavior and determinism for applications in computation with contrast to stochastic and linear systems. Linear systems will not be able to generate all the possible logic functions because their temporal patterns are inherently limited. As a result, linear systems do not have versatility and reconfigurability. On the other hand, stochastic systems contain many temporal patterns but the sequences are not deterministic so they cannot be used for computing purposes.

Nonlinearity is an essential feature in order to generate all the logic functions but *chaos* may not always be required. In some maps, all the functions exist only when the system is chaotic. The amount of nonlinearity necessary for producing all the Boolean functions depends on the chosen system and the technique that is used to generate the input-output mapping. It is also possible that some systems will generate logic functions without actually being in the chaotic region.

Boolean functions can be generated from chaotic systems by tuning the control parameters in both continuous-time and discrete-time systems. Sensitivity to initial conditions can be leveraged to achieve fast switching between all the logic gates generated by chaos-based system. The chaos-based computing systems can be made of arrays of chaotic elements that can be individually programmed to generate all the possible Boolean functions. The array of chaotic elements can then be used to perform higher order functions such as memory, arithmetic logic and input/output operations and the system will be able to switch rapidly between different function executions. The chaotic computing system will be able to provide the reconfigurability of field programmable gate arrays (FPGA), the speed and optimization of application specific integrated circuits (ASIC) and the general usage of central processing unit (CPU) within the same architecture [45]. Chaos-based logic gates

can be beneficial for many research areas such as hardware security, reliability enhancement and reconfigurable logic.

It was proposed in 1998 that chaotic systems can be used to design computing systems [154]. When the research on chaos computing started, the motivation was to prove that chaotic systems can be exploited to perform universal computing. The main objective was to leverage the sensitivity to initial conditions and sequence generation features. In later years, researchers found out that chaotic elements are reconfigurable and are able to generate all the logic functions by using a thresholding mechanism [113, 155]. Traditional programmable gate array elements achieve their reconfigurability and flexibility through the usage of multiple single function logic gates whereas chaotic elements are able to perform several functionalities using the same chaotic element. Sinha *et al.* demonstrated that dynamical systems are able to encode numbers, perform arithmetic operations such as multiplication and addition and also calculate least common multiplier from a list of integers [154]. Murali *et al.* built an universal NOR gate using continuous-time chaotic Chua's circuit [115]. Rizk *et al.* designed a universal logic gate which can perform all logic functions such as AND, OR, NAND, NOR and XOR using Chua's circuit [136]. Chaos-based system has been used to design flip-flops which is a building block for memory elements [27].

The method of thresholding provides a flexible tool for controlling chaos and there can be a wide variety of applications ranging from engineering to biology [114]. The strategy is to monitor one state variable and changing its value when it exceeds a certain threshold. In this type of control mechanism, computation and run time knowledge is not required to gain the desired controllability. The method only involves monitoring a single state variable and other parameters are not changed in the system. The thresholding mechanism does not stabilize unstable periodic orbits instead, it focuses on clipping the desired time sequence and enforces a periodicity of the sequence by resetting the initial conditions. Chaos-based systems are ideal candidates for this controlling mechanism because it possesses a wide range of temporal patterns which can be clipped to demonstrate different behaviors.

When chaos-based logic gates are implemented in VLSI technology, the input and output should have the same level which allows a direct connection between different gates without

the need of level converters. In [137], researchers proposed a method that allows continuous-time chaos-based system to be morphed into any logic function by using thresholding mechanism while keeping the input and output level unaltered.

In the earlier days, two-input reconfigurable chaotic logic gates (*chaogate*) were built and were capable of generating all the Boolean functions [117, 115, 116]. Researchers have also demonstrated the design of multiple-input logic gates using chaotic elements which reduces area overhead and propagation delay in circuits. Multiple-input chaotic elements would have a wider range of applications, more power efficient and demonstrate higher performance.

The advantages of using *chaogate* are:

- Fast switching between different logic functions using the same chaos-based logic gate.
- Ability to perform complex instructions which require complex design of logic gates.
- Reduction in transistor count for performing complex tasks.

Dynamical systems can be of two types, continuous-time and discrete-time. A discrete-time system is also known as an iterated map. In order to design a nonlinear iterated map, the output of the map has to feedback to the input. A dynamical system maps the initial condition to future states which is identical to a function mapping its inputs to the outputs. Examples of both types of systems are described in sections below.

2.2.1 Continuous-time Nonlinear System

Digital logic can be obtained from a continuous-time nonlinear system. The nonlinear system is defined by the following equation:

$$\frac{d\mathbf{x}}{dt} = \mathbf{F}(\mathbf{x}, t). \tag{2.1}$$

where $\mathbf{x}$ are the state variables x_1, x_2, x_3....x_N and $\mathbf{F}$ is the nonlinear function. In this system, if x_1 is chosen to be thresholded by E, then $x_1 = E$ whenever x_1 is greater than E.

In continuous-time systems, 2-input 1-output logic functions are realized by choosing an input-dependent threshold voltage represented by:

$$E = V_c + I_1 + I_2. \tag{2.2}$$

where V_c represents the dynamic control signal which chooses the functionality of the processor and I_1 and I_2 are inputs to the logic gate. As mentioned earlier, I_1 or I_2 has a value of 0 when logic input is zero and has a value of V_{in} when logic input is high. Threshold, E is equal to V_c when I_1 and I_2 are (0, 0), V_c+V_{in} when I_1 and I_2 are (0, 1) or (1, 0) and V_c+2V_{in} when I_1 and I_2 are (1, 1).

The output, V_0 is interpreted as logic 0 if $x_1 < E$ and $V_0 = 0$. The logic output is 1, if $x_1 > E$ and $V_0 = (x_1\text{–}E) \sim V_{in}$. For example, in order to implement a NOR gate ($V_c = V_{NOR}$), the following equations must hold true:

1. when inputs are (0, 0), logic output is 1, which implies that $E = V_{NOR}$, output is $V_0 = (x_1\text{–}E) \sim V_{in}$.

2. when inputs are (0, 1) or (1, 0), logic output is 0, which implies that $E = V_{NOR}$+V_{in}, $x_1 < E$ so output is $V_0 = 0$.

3. when inputs are (1, 1), logic output is 0, which implies that $E = V_{NOR}$+2V_{in}, $x_1 < E$ so output is $V_0 = 0$.

An example of a continuous-time chaotic map is the double-scroll chaotic Chua's circuit which is represented by the following set of differential equations:

$$\dot{x_1} = \alpha(x_2 - x_1 - g(x_1)). \tag{2.3}$$

$$\dot{x_2} = x_1 - x_2 + x_3. \tag{2.4}$$

$$\dot{x_3} = -\beta x_2. \tag{2.5}$$

where x_1, x_2 and x_3 are the input variables, α and β are system parameters and $g(x)$ is a characteristic function described below in equation 2.6.

$$g_x = m_1 x + 1/2(m_0 - m_1)(|x+1| - |x-1|). \tag{2.6}$$

Realization of Multiple-Input Logic Gates

Previously, 2-input logic gates have been discussed and in this section we discuss about designing 3-input logic gates using chaotic elements. Here, continuous-time system is chosen as discussed in section 2.2.1 to demonstrate the functionality of 3-input logic gates. In this system, if a state variable, x_1 is chosen to be thresholded, then whenever the value of the variable exceeds the threshold, E, x_1 resets to E. The value of E is selected based on the logic inputs, I_1, I_2 and I_3. The equation of the threshold for 3-input logic gate is given by:

$$E = V_c + I_1 + I_2 + I_3. \tag{2.7}$$

where V_c is the dynamic control signal determining the functionality of the system. By changing the value of V_c, the logic operation can be changed.

I_1 or I_2 or I_3 has a value of 0 when logic input is zero and has a value of V_{in} when logic input is high. Threshold, E is equal to V_c when I_1, I_2 and I_3 are (0, 0, 0), V_c+V_{in} when I_1, I_2 and I_3 are (0, 0, 1) or (0, 1, 0) or (1, 0, 0), V_c+$2V_{in}$ when I_1, I_2 and I_3 are (0, 1, 1) or (1, 0, 1) or (1, 1, 0) and V_c+$3V_{in}$ when all the inputs are (1, 1, 1).

As discussed before, the output is interpreted as logic 0 if $x_1 < E$ and $V_0 \sim 0$. The logic output is 1 if $x_1 > E$, and $V_0 = (x_1\text{–}E) \sim V_{in}$. In order to design logic gates like NOR and NAND, knowledge of the dynamics of the nonlinear system is required in order to choose the values of V_c and V_0.

The differential equations 2.3, 2.4 and 2.5 are used to represent the double-scroll chaotic Chua's attractor. The chaotic system that implements the attractor is shown in Fig. 2.3. In the figure, $E = V_c + I_1 + I_2 + I_3$ known as the dynamically varying threshold voltage. Voltage, V_T represents the signal from the threshold controller and V_0 is the difference voltage signal. The state variable, x_1 which is represented by voltage V_1 is thresholded by the control circuit with voltage E setting different threshold voltages. Equation 2.4 is changed from

$dx_2/dt = x_1 - x_2 - x_3$ to $dx_2/dt = E - x_2 - x_3$ only if $x_1 > E$, otherwise, no controlling action takes place.

The classical Chua's circuit in Fig. 2.3 works in the following manner:

1. When the value of E is greater than the value of V_1, diode, D turns on. The circuit now acts as a voltage follower and voltage, V_T is equal to V_1.

2. When the value of E is less than the value of V_1, diode, D is off and voltage, V_T is the same as voltage, E.

2.2.2 Discrete-time Nonlinear System

In discrete-time dynamical systems, the chaos is generated by using a set of differential equations known as a chaotic map. A discrete-time chaotic map can be used where the state is represented by variable, x. In order to generate responses by utilizing the different patterns generated by chaotic circuits, a thresholding mechanism can be used to generate AND, OR, NAND, NOR, XOR and NOT functions [44]. The simple steps used to make logic gates from chaotic systems is given below:

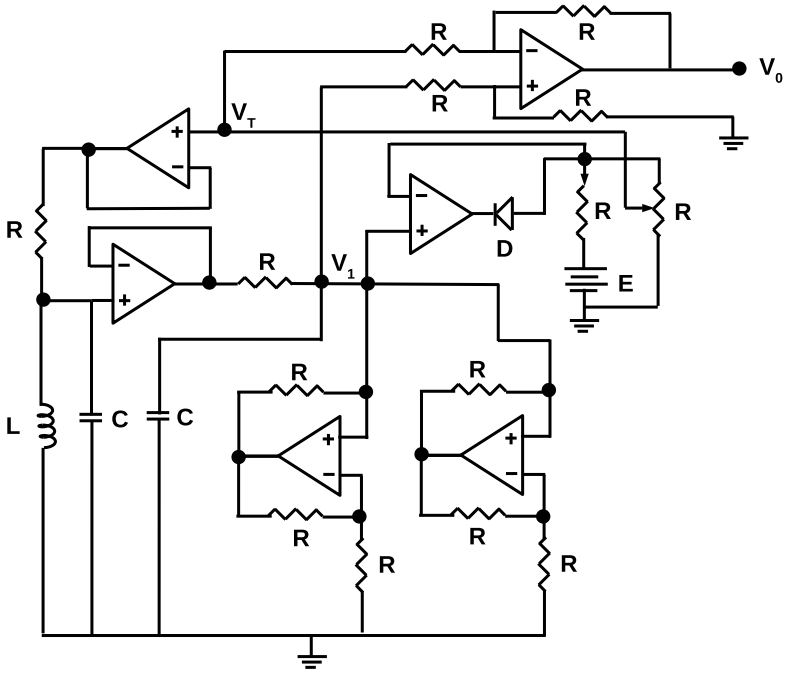


Figure 2.3: Chaotic Chua's attractor implemented in circuit to build logic gates [115].

Table 2.1: Necessary and sufficient conditions to obtain different logic functions from the nonlinear dynamical element [43].

Logic Gate	I_1, I_2	**Output**	**Necessary condition**
AND	(0,0)	0	f(x_{AND}) $< E$
	(0,1), (1,0)	0	f(x_{AND}+V_{in}) $< E$
	(1,1)	1	f(x_{AND}+2V_{in}) $\geq E$
OR	(0,0)	0	f(x_{OR}) $< E$
	(0,1), (1,0)	1	f(x_{OR}+V_{in}) $\geq E$
	(1,1)	1	f(x_{OR}+2V_{in}) $\geq E$
XOR	(0,0)	0	f(x_{XOR}) $< E$
	(0,1), (1,0)	1	f(x_{XOR}+V_{in}) $\geq E$
	(1,1)	0	f(x_{XOR}+2V_{in}) $< E$
NOR	(0,0)	1	f(x_{NOR}) $\geq E$
	(0,1), (1,0)	0	f(x_{NOR}+V_{in}) $< E$
	(1,1)	0	f(x_{NOR}+2V_{in}) $< E$
NAND	(0,0)	1	f(x_{NAND}) $\geq E$
	(0,1), (1,0)	1	f(x_{NAND}+V_{in}) $\geq E$
	(1,1)	0	f(x_{NAND}+2V_{in}) $< E$

1. Inputs:

$$x \rightarrow x_0 + I_1 + I_2. \tag{2.8}$$

$$x \rightarrow x_0 + I. \tag{2.9}$$

Equation 2.8 is the state equation for two-input logic gates and similarly equation 2.9 represents the state condition for single input gates. Here x_0 is the initial state of the system, $I = 0$ for logic input zero and $I = V_{in}$ for logic input high.

2. Dynamical system:

$$x \rightarrow f(x). \tag{2.10}$$

Here $f(x)$ represents a highly nonlinear function.

3. Threshold mechanism:

$$V_0 = \begin{cases} 0, & \text{if } f(x) \leq E \\ f(x) - E, & \text{otherwise.} \end{cases} \tag{2.11}$$

Here, E represents the threshold.

In a chaotic system, if the initial state is required to be set accurately, then a controlling mechanism is required. A threshold controller can be used to set the initial state, x_0. It is also important to ensure that the input and output have equivalent definitions for different logical operations. In order to ensure this condition, same V_{in} is used throughout the system and this will allow the output of one chaotic element to couple with the input of another chaotic element. An example of discrete-time chaotic map is the well-known 1-D function called logistic map which is implemented using the following equation:

$$f(x) = ax(1 - x). \tag{2.12}$$

where $x \in [0, 1]$ and the variable, a determines whether the system is in chaotic or periodic region. The characteristic of logistic map is analogues to biological and physical phenomena. Nonlinear oscillators and electrical circuits portray such phenomena. The logistic map generates sequences in real number domain and needs to be converted to binary representation in order to be used in different applications.

Table 2.1 shows the necessary conditions that must be fulfilled in order to implement the different logic gates. If logistic map is used as the chaotic element, then typical values of x_0 and E are shown in Table 2.2. The value of a from equation 2.12 is equal to 4 and V_{in} equals 1/4 for all logic gates [43].

Table 2.2: One specific solution to yield different logic functions using logistic map [43].

Operation	**AND**	**OR**	**XOR**	**NAND**
x_0	0	1/8	1/4	3/8
E	3/4	11/16	3/4	11/16

In chaos computing, a set of initial conditions and parameters need to be determined to produce logic functions from a specific chaotic system. A set of parameters can produce the results for a logical OR or a logical AND. The parameters can be stored as a "lookup-table" for future computations. There might be conditions or parameters that do not generate desired logical functions and those parameters can be discarded. The chosen set of parameters should lead to the correct output.

2.2.3 Digital Logic Obtained by Varying Iteration Number

In addition to perturbing the initial condition of the nonlinear dynamical system, digital logic can also be obtained from the time evolution of the states. This concept is stating that logic gate can be obtained if the initial condition is kept fixed and output is sampled at different iterations, then it is possible to obtain multiple logic functions. The advantage of this technique is that a single system will be able to perform complex operations efficiently.

In section 2.2.2, it was shown how different logic functions like AND, OR and XOR were obtained by sampling at a fixed iteration. With varying iterations, it is possible to obtain an AND logic in the first iteration and an OR logic in the second iteration. Hence, a sequence of logical or mathematical operations can be performed as smoothly as a single operation. As discussed previously, for two-input logic operation the initial state is given by the following equation:

$$x \rightarrow x_0 + I_1 + I_2. \tag{2.13}$$

Here, x_0 is the initial state, $I = 0$ for logic input zero and $I = V_{in}$ for logic input high.

Table 2.3: Necessary conditions to implement logic functions from nonlinear systems for varying iterations [43].

Logic	**AND**	**OR**	**XOR**	**NOR**	**NAND**
Iteration n	1	2	3	4	5
Input $(0,0)$	x_1=f$(x_0) < E$	f$(x_1) < E$	f$(x_2) < E$	f$(x_3) > E$	f$(x_4) > E$
Input $(0,1),(1,0)$	x_1=f$(x_0 + V_{in}) < E$	f$(x_1) > E$	f$(x_2) > E$	f$(x_3) < E$	f$(x_4) > E$
Input $(1,1)$	x_1=f$(x_0 + 2V_{in}) > E$	f$(x_1) > E$	f$(x_2) < E$	f$(x_3) < E$	f$(x_4) < E$

After n iterations, the output is determined by using the following equations:

$$V_0 = \begin{cases} 0, & \text{if } f_n(x) \leq E \\ f_n(x) - E, & \text{otherwise.} \end{cases} \tag{2.14}$$

Here, E is the threshold.

The main difference from the previous approach is that n is the tuning parameter and it is varied to implement different logic functions. In earlier approach, n was held constant at the first iteration, i.e. $n = 1$ and V_c was tuned to change the mapping of input-output patterns. The inputs set up the initial condition of the system using equation 2.13. The system evolves over n iterations and reaches the final state, x_n. The final state is compared with a predetermined threshold, E. If the state is greater than E, the output is 1, otherwise the output is 0. Table 2.3 shows the necessary conditions to implement logic functions from nonlinear dynamical systems by altering the iteration number and keeping the initial condition fixed.

2.2.4 Distinguishing Between (0,1) and (1,0) Input Pairs

Section 2.2.2 describes a discrete-time nonlinear system that cannot distinguish between the input pairs (0,1) and (1,0). There might be cases where it is required that the two pairs generate different logical outputs. In this technique, a two-input logic gate is considered with inputs, I_1 and I_2. The initial condition is generated by the following equation:

$$x \to x_0 + X_1 + X_2. \tag{2.15}$$

where $X_1 = 0$ when $I_1 = 0$ and $X_1 = V_{in}$ when $I_1 = 1$. $X_2 = 0$ when $I_2 = 0$ and $X_2 = 2V_{in}$ when $I_2 = 1$. V_{in} is a positive constant.

All the input combinations are considered below:

1. $I_1 = I_2 = 0$ so $x = x_0{+}0{+}0 = x_0$.
2. $I_1 = 1$, $I_2 = 0$ so $x = x_0 + V_{in} + 0 = x_0 + V_{in}$.
3. $I_1 = 0$, $I_2 = 1$ so $x = x_0 + 0 + 2V_{in} = x_0 + 2V_{in}$.

4. $I_1 = 1$, $I_2 = 1$ so $x = x_0 + V_{in} + 2V_{in} = x_0 + 3V_{in}$.

The above encoding of the inputs treats (0,1) and (1,0) as separate input combinations and hence can implement asymmetric logic functions. Logic outputs are determined in the same manner as in equation 2.11. Symmetric gates can also be evaluated using this scheme which is capable of generating all the 16 possible Boolean functions in contrast to the maximum 8 functions when (0,1) and (1,0) were equivalent. The designer needs to determine values for the threshold, E, constant V_{in} and initial state, x_0 to implement the logic gates.

Another method of distinguishing the input pairs is to utilize the iteration number, n by using the following equation:

$$n \rightarrow n + X_1 + X_2. \tag{2.16}$$

The above equation yields four distinct conditions.

1. $I_1 = I_2 = 0$ so $n = n{+}0{+}0 = n$.

2. $I_1 = 1$, $I_2 = 0$ so $n = n + V_{in} + 0 = n + V_{in}$.

3. $I_1 = 0$, $I_2 = 1$ so $n = n + 0 + 2V_{in} = n + 2V_{in}$.

4. $I_1 = 1$, $I_2 = 1$ so $n = n + V_{in} + 2V_{in} = n + 3V_{in}$.

After n evolution in time, the final state is achieved and $f_n(x)$ is the desired output.

2.2.5 Design of Multi-Input Multi-Output Logic Functions

Combinational logic can be implemented by a single chaotic element. An example can be bit-by-bit arithmetic addition which involves two logic gates, an AND gate to generate the carry and an XOR gate to generate the sum. Using the previously mentioned scheme, the half-adder operation can be obtained in consecutive iterations. Table 2.4 shows the necessary conditions for implementing half-adder using a single chaotic element.

Table 2.4: Necessary and sufficient conditions to build half-adder from the nonlinear dynamical element [43].

Logic Operation	I_1, I_2	**Output**	**Initial State**	**Necessary Condition**
Carry	(0,0)	0	x_0	x_1=f(x_0) $\leq E$
	(0,1)	0	$x_0 + V_{in}$	x_1=f($x_0 + V_{in}$) $\leq E$
	(1,0)	0	$x_0 + 2V_{in}$	x_1=f($x_0 + 2V_{in}$) $\leq E$
	(1,1)	1	$x_0 + 3V_{in}$	x_1=f($x_0 + 3V_{in}$) $> E$
Sum	(0,0)	0	x_0	x_2=f(x_0) $\leq E$
	(0,1)	1	$x_0 + V_{in}$	x_2=f($x_0 + V_{in}$) $> E$
	(1,0)	1	$x_0 + 2V_{in}$	x_2=f($x_0 + 2V_{in}$) $> E$
	(1,1)	0	$x_0 + 3V_{in}$	x_2=f($x_0 + 3V_{in}$) $\leq E$

Moreover, the implementation of full-adder requires two half-adders along with an OR gate. A full-adder requires five gates in total including two XOR gates, two AND gates and an OR gate. If logistic map is used as the chaotic oscillator, then only three iterations are required to implement the full-adder operation. This technique allows combinational circuits to be implemented with fewer computational components and without cascading.

2.3 Concluding Remarks

Previous sections demonstrated how a single computing system made with nonlinear dynamical system can implement all the Boolean functions. The nonlinear responses generated by the system is utilized to generate functions. In contrast, field programmable gate arrays use multiple single purpose gates which makes the reconfiguration feature wasteful of area on a chip and degrades the performance of the system. Preliminary goal was to prove that chaotic elements can be used to design computers which provides better performance and flexibility. Systems that combine CMOS circuits along with programmable chaos-based circuits are also a design option.

A major drawback of using chaotic-based logic gates is that the hardware cost is larger compared to traditional logic gates. In [35], it is discussed that the op-amp based design

of Chua's circuit along with the capacitor and inductor area, a total of 52 transistors are required to implement the *chaogate*. This limitation of chaos-based systems can be overcome by increasing the number of inputs of the logic gate.

Chapter 3

Design of Chaotic Oscillator and Chaos-Based Logic Gate Using Three Transistor Chaotic Map

****Portions of this chapter were published in:**

[150] Aysha S. Shanta, Md Badruddoja Majumder, Md Sakib Hasan, Mesbah Uddin, and Garrett S. Rose. "Design of a reconfigurable chaos gate with enhanced functionality space in 65nm cmos." *In 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1016-1019. IEEE, 2018.

[148] Aysha S. Shanta, Md Sakib Hasan, Md Badruddoja Majumder, and Garrett S. Rose. "Design of a Lightweight Reconfigurable PRNG Using Three Transistor Chaotic Map." *In 2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 586-589. IEEE, 2019.

[63] Md Sakib Hasan, Md Badruddoja Majumder, Aysha S. Shanta, Mesbah Uddin, and Garrett S. Rose. "A Chaos-Based Complex Micro-instruction Set for Mitigating Instruction Reverse Engineering." *Journal of Hardware and Systems Security* (2019): 1-17.

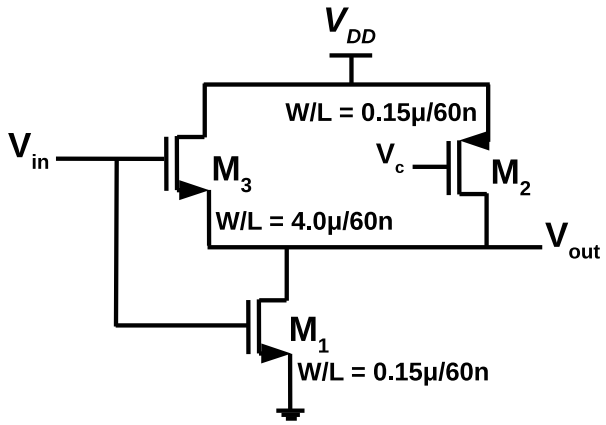


Figure 3.1: Three transistor chaotic map circuit designed in 65 nm process [150, 46].

3.1 Design of Chaotic Map

In this work, a three transistor CMOS circuit is implemented in 65 nm process which can be used as a map to generate chaotic signals [46, 80]. The circuit is extremely compact which makes it suitable for integration in VLSI chips with area and power concerns. The nonlinear map circuit is analog since the state variables are analog. A one-dimensional discrete-time chaotic series can be generated by a simple iterative method:

$$x_{n+1} = f(x_n). \tag{3.1}$$

where $f(x)$ represents a nonlinear function. Several analog circuits have been proposed in order to generate chaotic signals [164, 48, 37]. The CMOS implemented circuits are usually designed to imitate one of the popular chaos maps, such as, tent map or logistic map. Chaos maps can be achieved with V-shaped functions [37]. The branches of the V-shaped curve can be generated by different functions which allows a degree of freedom in constructing the map circuit. This kind of design is more suitable for applications in stochastic neural networks where accuracy of the map function and the statistical properties of the generated signal are not important as long as the map is able to generate random-like signals. The V-shape can be designed using two circuits and the nonlinear function achieved will be the DC characteristics of the final circuit.

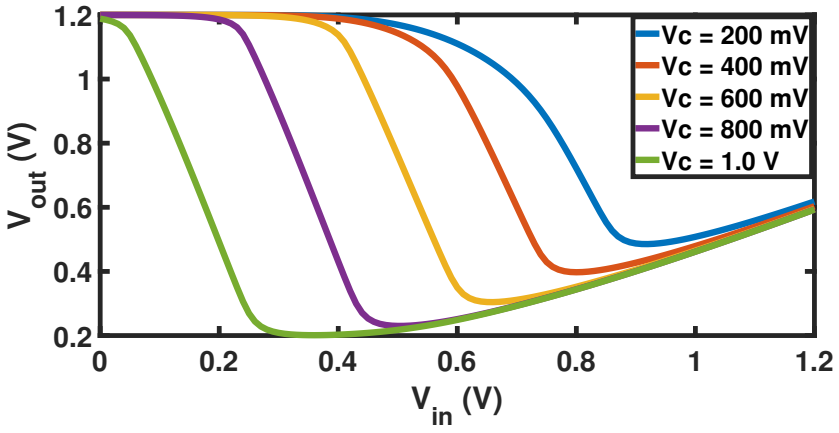


Figure 3.2: DC transfer characteristic of the three transistor map circuit [150].

Chaos map circuits are expected to portray a non-monotonic characteristic where the 'average steepness' of the generated chaotic signal is greater than unity. The chaotic map circuit used in this work is shown in Fig. 3.1. The DC transfer characteristics of the map circuit for different values of the bias voltage voltage, V_c is shown in Fig. 3.2. As can be seen from the figure, the three transistor circuit generates a V-shaped map. The negative slope is constructed using a common-source amplifier and the positive slope can be obtained from a source follower. The circuits are connected in parallel so that the final response is a summation of their DC characteristics. In the three transistor map circuit, transistor M_3 acts as the source follower with M_1 as load and transistor M_1 acts as the common source amplifier with M_2 as load. The two sub-circuits share the same input and output in creating the V-shaped transfer curve. The operation of the map circuit is explained below.

3.1.1 DC Transfer Characteristics of the Map

When V_{in} is less than the threshold voltage of transistor M_1, the output V_{out} remains at V_{DD} = 1.2 V. At this point, transistor M_3 is turned-off. As the value of V_{in} increases, V_{out} starts to decrease as transistors M_1 and M_2 act like an inverter. The slope of the transfer curve depends on the sizes of M_1 and M_2 and the value of the bias voltage, V_c. At some point, the value of V_{out} falls below V_{in} such that the gate-source voltage of transistor, M_3 becomes larger than its threshold voltage and the transistor starts conducting. If the value of V_{in} is

increased further, V_{out} increases since M_3 acts like a source follower. The current through transistor M_1 keeps increasing due to increase in V_{out}. The slope of V_{out} against V_{in} can be close to unity if the aspect ratio of width to length of transistor M_3 is much larger than M_1.

Process variation might affect the steepness of the slope of the transfer curve so it is important to make the design robust. The physical implementation of the two map circuits will never be identical due to component mismatch. Despite this mismatch, the mapping function will result in chaotic behavior. In this design, the controllability and flexibility exists since the bias voltage, V_c of the map circuit can be adjusted to get different transfer curves as shown in Fig. 3.2. The bias voltage, also known as the bifurcation parameter, determines whether the chaotic generator produces periodic or chaotic signals. The controllability of the circuit can be essential for applications such as chaotic neural networks [6].

3.2 Design of Chaotic Oscillator

Traditionally, two sample-and-hold circuits were required in addition to the map circuit to design the chaos generator in order to generate discrete-time chaotic signals [37]. The schematic of the conventional chaotic oscillator design is shown in Fig. 3.3. It is assumed that the map circuit has high-impedance input. Two non-overlapping clocks, ϕ_1 and ϕ_2 are used in order to avoid race conditions. Initially, the input of the map is equal to x_n. The nonlinear map circuit evaluates the initial voltage and output, x_{n+1} is generated which is sampled on the capacitor, C_1. On the second phase, ϕ_2, the value of x_{n+1} is transferred to capacitor, C_2 where it is held constant at the input of the map circuit for the next iteration. In some cases, the buffer is not used and a large ratio of C_1/C_2 is used in order to reduce charge-sharing error of the second sample-and-hold circuit.

The settling time, t_{set} of the map circuit along with C_1 sets the limit on the minimum period of ϕ_1. The disadvantage is that the time required to perform the second sample-and-hold reduces the maximum operating frequency of the chaotic oscillator. In this work, a chaotic oscillator with two identical map circuits is used so that the maximum frequency, $f_{max} = 1/t_{set}$. The schematic of the proposed chaotic oscillator is shown in Fig. 3.4. Each

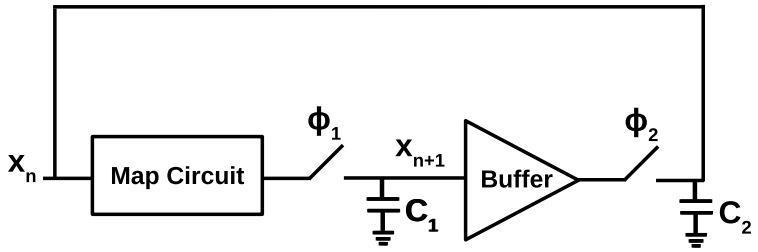


Figure 3.3: Traditional chaotic oscillator using buffer [85].

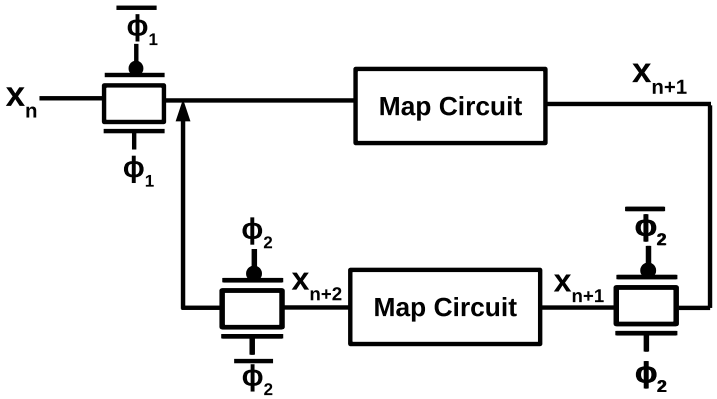


Figure 3.4: Chaotic oscillator using two map circuits [150, 46].

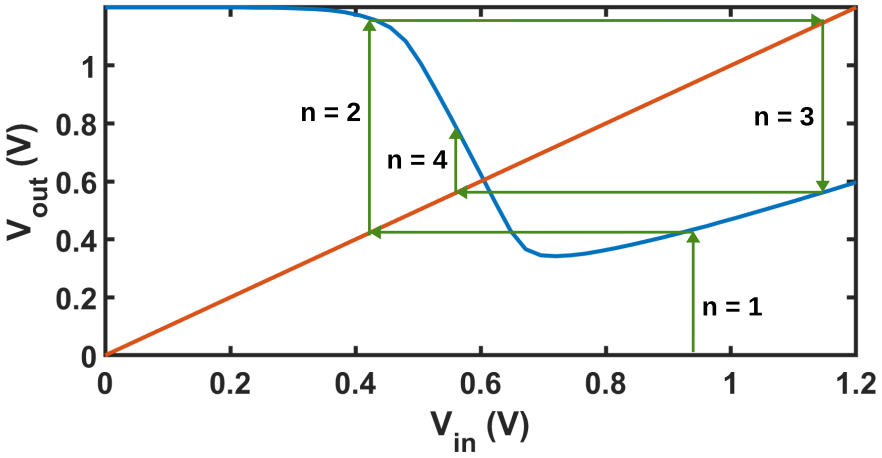


Figure 3.5: Iterating through the chaotic map.

map implements a nonlinear function which helps to generate chaotic signals. The discrete-time dynamical system can only map analog inputs to analog outputs. The replacement of buffer with another map circuit doubles the rate at which the map iterates which in turn helps to speed up the computation process.

On the first phase, ϕ_1, initial condition, x_n is applied to the forward path map circuit and it produces the next response by using equation 3.1. On phase, ϕ_2, the next response is generated from the second map circuit in the feedback path by using the following equation:

$$x_{n+2} = f(x_{n+1}). \tag{3.2}$$

The response, x_{n+2} is then applied to the forward path map circuit again and the process continues to generate new responses. The frequency and duty cycle of the clocks are defined by the RC time constant of the sample-and-hold circuits and the propagation delay of the system. Pass transistors are used to select which map circuit produces the response. This method of using two map circuits is efficient since all the transistors are part of the map circuit or act as switches and the system does not consume extra power and area due to sample-and-hold buffer.

The chaotic oscillator has been designed in 65 nm process. The size of the transistors used in the map circuit are: M_1: 0.15 μm/60 nm, M_2: 0.15 μm/60 nm and M_3: 4 μm/60 nm. The switches in the chaotic oscillator are implemented by pass transistors consisting of $n-MOS$ and $p-MOS$ transistors. The size of $p-MOS$ transistor is 8 μm/100 nm and the size of $n-MOS$ transistor is 4 μm/100 nm. C_1 and C_2 are implemented by utilizing the inherent input capacitance of the map circuits which contributes to saving area on the chip. The oscillator is made up of 12 transistors including the switches. If a more compact design is required, the switches can be implemented with a single transistor.

In practical implementation of the chaotic oscillator, two major challenges can be the frequency of operation and the power supply required to generate the chaotic signals. In [80], Juncu *et al.* fabricated the chaos generator in 0.6 μm process. It was observed the circuit gave a constant performance upto 2.5 MHz and beyond this frequency the generator was still chaotic but had less amplitude since the sample-and-hold circuits were not fully

charged or discharged during the clock cycle. The reason for limited operating frequency can be off-chip capacitances and bonding pads. The power consumption of the circuit is another concern and depends on the current flowing from V_{DD} to ground through transistors M_1 and M_3 for large values of V_{in}.

3.2.1 Iterating through the Map

Fig. 3.5 demonstrates how the different values are generated from the chaotic map. In iteration, $n = 1$, $V_{in} \approx 0.95\ V$ and $V_{out} \approx 0.4\ V$. When $n = 2$, the previous output is the new input to the map, thus $V_{in} \approx 0.4\ V$ and the output $V_{out} \approx 1.18\ V$. The loop continues until the desired number of iterations are completed.

3.2.2 Sensitivity to Initial Condition

The chaotic oscillator's sensitivity to initial conditions is shown in Fig. 3.6 for $V_c = 520\ mV$. The initial seed is changed from 600 mV to 601 mV and it can be seen that around 40^{th} iteration, the output of the oscillator starts to diverge from each other. Sensitivity and reconfigurability of the map means that there can be a huge number of initial conditions and control parameters to generate random sequences which means that the key space will be large and it will be able to withstand statistical attacks from the adversaries in case the map is used to design pseudo-random number generators (PRNGs).

3.2.3 Ergodicity of the Chaotic Map

Ergodicity is defined in statistics as a random phenomena where the time average of one sequence of events is the same as the average of the entire sequence. The ergodicity can be expressed by the distribution of the state value of the chaotic map. The ergodicity of a map can also be demonstrated by fixing the bifurcation parameter and choosing a random initial value. The map will be iterated for a few thousand iterations and the distribution of the map is plotted against the iteration number, n. The results in Fig. 3.7 shows the values of V_c for which the map is chaotic and the ergodicity is better. When $V_c = 180\ mV$, the output oscillates between two values. As the value of V_c is increased, the periodicity of the

oscillator increases and at one point, it enters into the chaotic region. The first 100 iterations are discarded in order to avoid the transient phase.

3.2.4 Bifurcation Diagram

Bifurcation diagram helps in understanding the dynamics of nonlinear dynamical systems. The bifurcation diagram of the chaotic oscillator is shown in Fig. 3.8. For small bias voltages, V_c, the system displays periodic behavior. The number of functions that the circuit can implement does not increase exponentially in the periodic region. As V_c increases, a period-doubling cascade takes place and the system moves into chaotic region. In this region, the number of logic functions that the system can implement exponentially increases. Further increase in the bias voltage leads the system into periodic region again and the process of chaotic and periodic region shifts continues. There are mainly two chaotic regions in the bifurcation diagram. The first region starts at 527 mV and ends at 622.5 mV. The second chaotic region lies between 890 mV and 952.5 mV. The bias voltage has been chosen from the first chaotic region since the span of the output voltage is larger and ranges from 294 mV to 1.194 V. The initial value of the map is also chosen from this output range. The delay of the circuit is higher for large values of bias voltage so the value of V_c is chosen from the first region only. The first 1000 iterations are discarded to avoid the transient phase and a total of 4000 iterations have been used to generate the bifurcation diagram.

3.2.5 Lyapunov Exponent

Lyapunov exponent (LE) is the average convergence or divergence of states due to small changes in the initial conditions. A positive Lyapunov exponent means that the system is in chaotic region and that the perturbations in initial conditions diverge the states by a significant amount. It indicates that a chaotic system becomes more unpredictable if the LE is more positive. If a nonlinear circuit is not sensitive to initial conditions, then the following statements are true.

1. The nonlinear system is incapable of generating multiple logic functions.

2. The Lyapunov exponent is zero or negative.

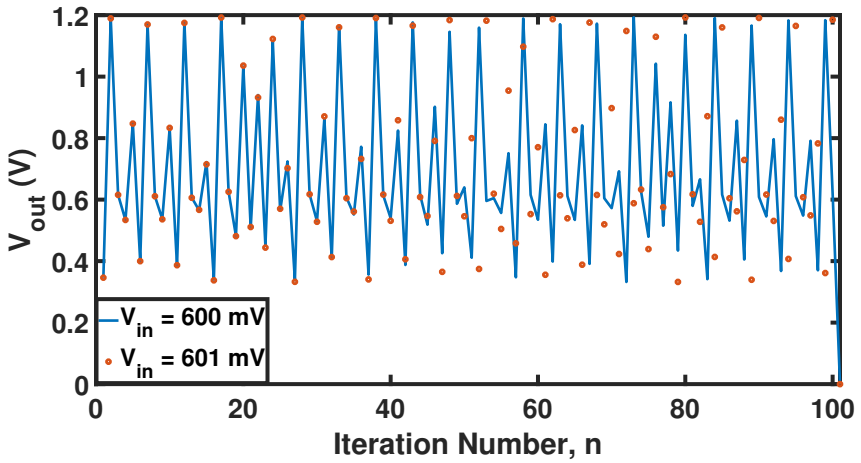


Figure 3.6: Sensitivity of the chaotic oscillator to slight change in initial condition.

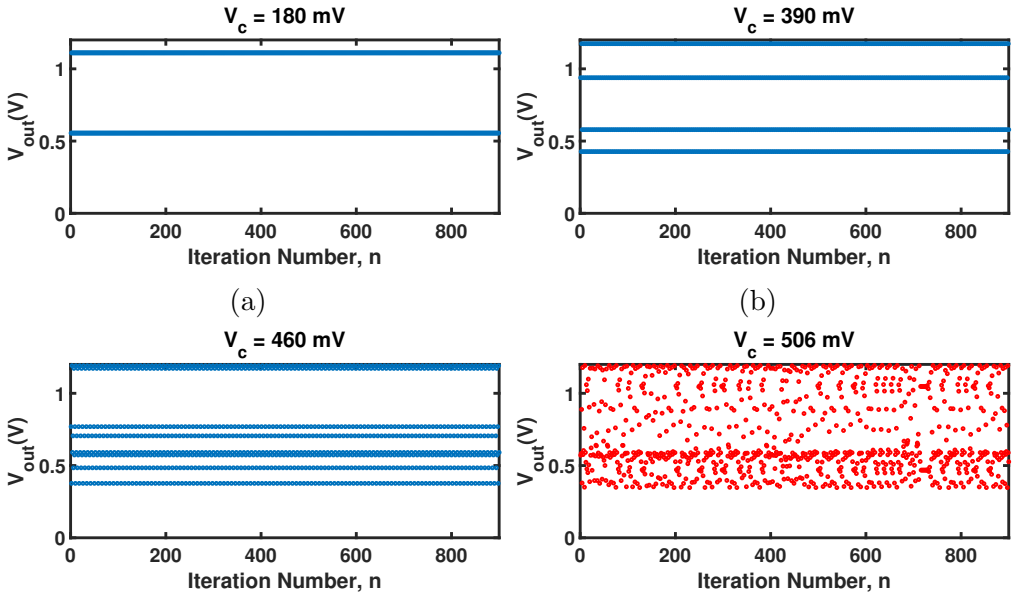


Figure 3.7: (a)-(d): Sampled output of chaotic oscillator for $V_{in} = 750\ mV$ showing different states as V_c is changed (a) Period 2 (b) Period 4 (c) Period 8 (d) Chaotic region.

The equation for the Lyapunov exponent is given below:

$$\lambda = \lim_{n\to\infty} \frac{1}{n} \sum_{i=0}^{n-1} ln|f'(x_i)|. \tag{3.3}$$

The diagram of Lyapunov exponent is shown in Fig. 3.9. It can be seen that the values of positive Lyapunov exponent correlates to the chaotic regions in the bifurcation diagram. The perturbation value of the initial condition used in this work is 1 mV. In order to calculate the Lyapunov exponent, 3000 iterations were considered out of which 300 iterations were truncated in order to eliminate the transient phase.

3.3 Reconfigurable Chaos-based Logic Gates

3.3.1 Introduction

Chaos-based logic gates have many advantages over traditional logic circuits. A single chaos-based logic gate can implement all the universal logic functions such as AND, OR, XOR, NAND, NOR and XNOR. In conventional reconfigurable logic gates such as look-up tables (LUTs), the need for additional hardware grows exponentially with the increase in number of inputs. Therefore, using chaos-based logic gates saves significant area for large number of inputs. The apparent randomness of chaos-based logic gates helps to mitigate power analysis based side-channel attacks [105, 104]. Implementation of chaos-based logic gates have a high probability of useful functions but there will still exist a handsome number of unused or "garbage" functions which can be utilized to obfuscate the desired computation.

Ditto *et al.* made the concept of chaos computing concrete by building logic gates called *chaogates*. The gate utilizes a chaotic oscillator, C which can be Chua's function or logistic map to generate logic functions from the logic gate at a given time. The schematic of one-dimensional *chaogate* is shown in Fig. 3.10. The initial condition of the chaotic oscillator is made up of the summation of the analog control input, x_g and the two inputs, A and B. The digital inputs are converted to analog values by multiplying with the constant weighting factor, δ. The digital output, Y is generated by using a comparator with threshold voltage, V_{th}. The logic function generated from the *chaogate* can be altered by varying the parameters,

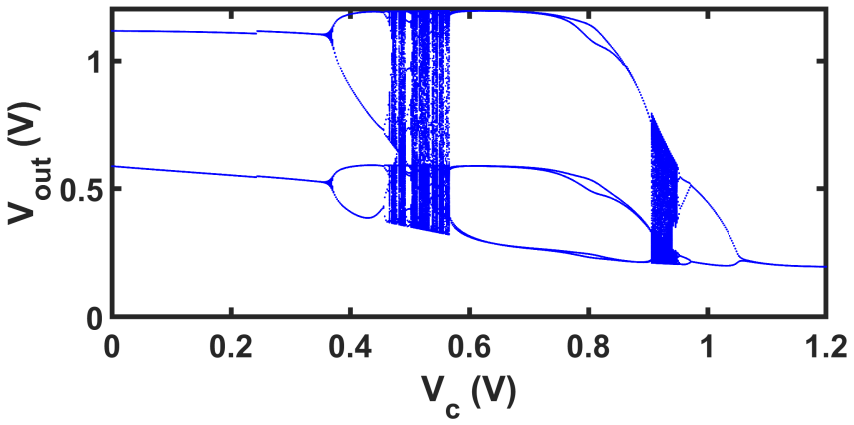


Figure 3.8: Bifurcation diagram of the chaotic oscillator [148].

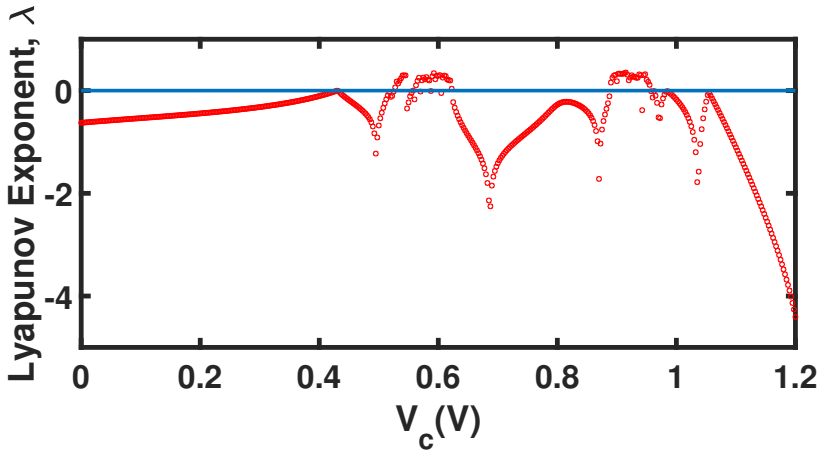


Figure 3.9: Lyapunov exponent of the chaotic oscillator [148].

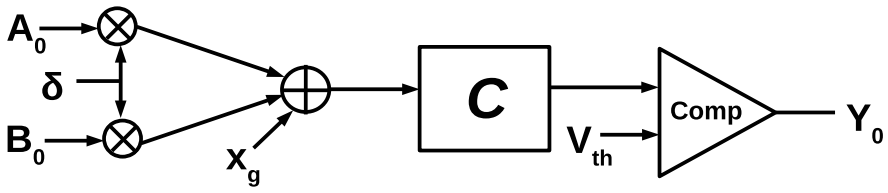


Figure 3.10: A basic 2-input chaogate using a single chaotic oscillator [139].

control input, x_g, weighting factor, δ and the threshold voltage, V_{th}. The chaotic oscillator can be seen as a pseudo-random number generator where x_g and δ make up the initial seed and V_{th} converts the analog output into digital voltages, 0 or 1.

It is possible to generate 16 different logic functions from a 2-input reconfigurable chaos-based logic gate. The disadvantage of using the one-dimensional *chaogate* is that the logic inputs {0, 1} and {1, 0} will produce the same initial condition for the chaotic oscillator. As a result, the generated digital logic output will be the same. This limitation on the outputs reduces the possible generated number of functions from 16 to 8. The generated logic functions are: TRUE, AB, $A + B$, $\overline{AB}$, $\overline{A + B}$, $A \oplus B$, $\overline{A \oplus B}$ and FALSE.

Previous research in chaos-based systems demonstrated the existence of inherent obfuscation available across multiple abstraction layers. Rose implemented a chaos-based arithmetic logic unit (ALU) and displayed how each function can be obtained in multiple ways by changing the control input, x_g, weighting factor, δ, threshold voltage, V_{th} and iteration number, n [139]. Moreover, since the ALU is designed with chaotic oscillators, there is no pattern to relate the various ways a specific function is implemented.

The major design limitations of a chaos-based system is the probability of generation of the desired function, circuit size and performance. Considering applications in hardware security, it is imminent that a chaos-based circuit will generate huge number of garbage functions which are not part of the desired instruction set. The two methods of changing the probability of functions in a chaos-based system are:

1. Tuning the comparator threshold value.

2. Modifying the chaos-based logic circuit.

3.3.2 Design of Reconfigurable Chaos-based Gates

In Fig. 3.11, the digital inputs of a logic gate are encoded as initial conditions to the dynamical system and the digital output are decoded from the final state to generate a digital function. The encoder converts digital inputs to analog inputs and the decoder converts the analog output of the map to binary outputs. There are four control inputs,

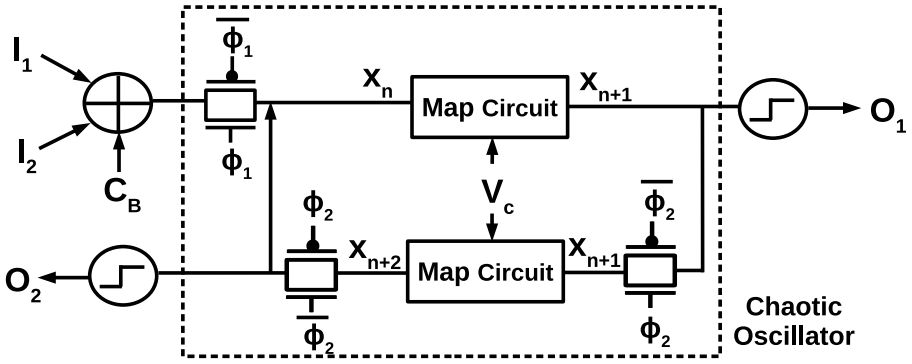


Figure 3.11: Chaotic oscillator is shown in the dotted box. Two-input chaos-based logic gate including digital encoding of the inputs and digital decoding of the outputs [86].

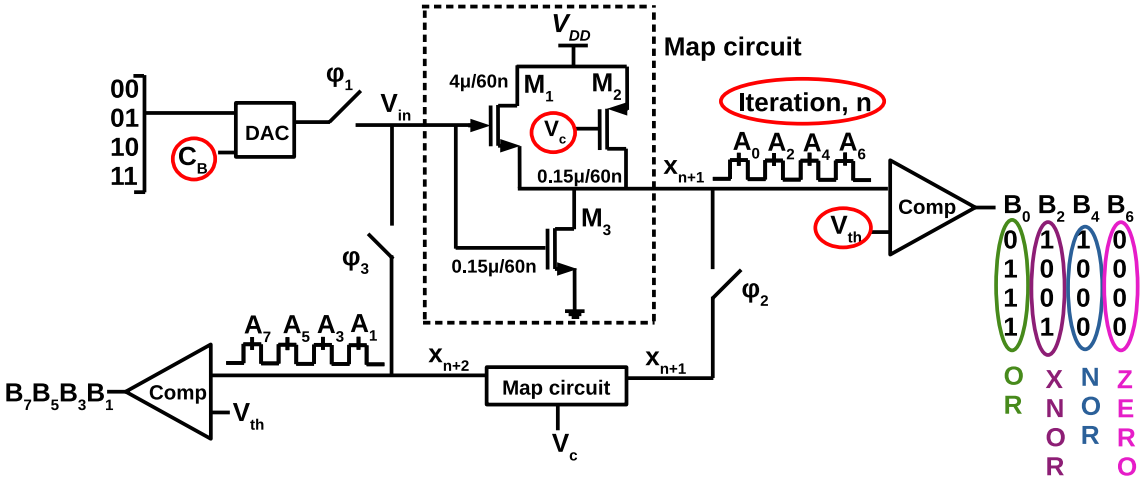


Figure 3.12: Two-input reconfigurable chaos-based logic gate demonstrating all the tuning parameters in red circles.

C_B, V_c, V_{th} and n available in the design to tune the block diagram to generate different functions as illustrated in Fig. 3.12.

The nonlinear dynamical system accepts data bits and control bits simultaneously and control bits can be different in different clock cycles. As a result, different functions are achievable at each clock cycle. The control input, C_B changes the initial condition of the dynamical system as a way of reconfiguring the block diagram. It is a known fact that small changes in the initial condition of the nonlinear dynamical systems can change the final state by a huge amount. However, this sensitivity to initial conditions puts a limit on the maximum number of iterations on the iterated map so that random noise will not be able to change the future state. The iteration number should be high enough so that the change in the future states created by C_B can take place but not so high that random noise can change the final states of the system and make the computation unreliable.

The bifurcation parameter of a nonlinear dynamical system determines whether the system is periodic or chaotic. The bifurcation parameter is typically an analog voltage such as gate voltage of a transistor. The control input, V_c changes the qualitative behavior of the map circuit. This control input can be tuned to change the way the map circuit connects the inputs to the outputs. Additional auxiliary circuitry is required to convert the digital input to analog value for application as bifurcation parameter.

Encoder/Digital-to-Analog Converter (DAC)

In the design of two-input reconfigurable chaos-based logic gate, two inputs, I_1 and I_2 along with a control input, C_B make up the initial condition for the map circuits. There are various encoding schemes present in literature. Fig. 3.13 shows a simple 3-bit digital-to-analog converter (DAC) to convert the digital inputs to an analog value. The transistors are turned on or off based on the data input values of I_1, I_2 and C_B. The transistors alter the value of resistor ratio and hence produces a voltage proportional to the digital data applied at the inputs. The 2-input reconfigurable chaos-based logic gate can be easily changed to multi-input system by using the initial control bits as data bits. Extra input lines can also be added to the encoder by increasing the input size of the DAC.

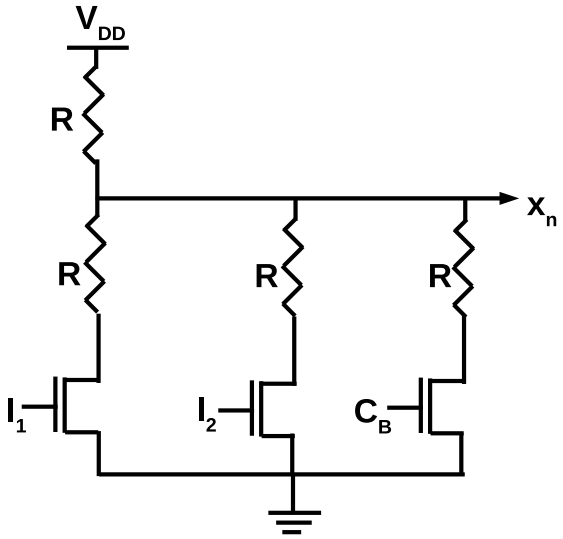


Figure 3.13: Three-bit DAC for encoding digital inputs to analog values [86].

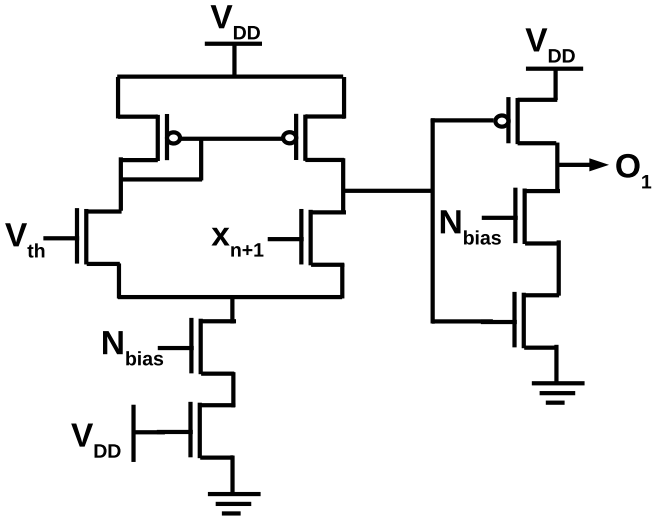


Figure 3.14: Decoder which converts final states to digital outputs [86].

The initial condition, x_n can be calculated by the DAC using data and control bits. Let us consider a bits of data, D and b bits of control, C and the voltage span is between 0 and V_{DD}. The equation for calculating x_n is given below:

$$x_n = V_{DD} \times \frac{\sum_{j=1}^{b} 2^{j-1} C_j + \sum_{i=1}^{a} 2^{i+b-1} D_i}{2^{a+b}}. \tag{3.4}$$

Here, C_1 and D_1 are the least significant bits.

Decoder/Analog-to-Digital Converter (ADC)

In order to decode the analog state of the map circuit to digital output, the output state space is divided into two parts. One partition decides if the digital output is 1 and the other decides if it is 0. A simple thresholding circuit is shown in Fig. 3.14. The decoder circuit either generates V_{DD} or 0 V which represents a 1 or 0 bit respectively. The output depends on the final state of the map circuit, either x_{n+1} or x_{n+2} depending on which map circuit is sampled. Any threshold voltage, V_{th} could be chosen to divide the output state space of the map. The critical point of the V-shaped transfer curve shown in Fig. 3.2 can be a suitable choice for partition between 0 and 1. The main point to note is that the partitioning should be able to retain the entropy in the system [33]. In order to ensure a quick transition region at V_{th} and to reduce the decoder's dependence on temperature variations, an analog comparator circuit is used which compares the analog output of the map against an internally generated threshold voltage.

The digital output from the final state of the map circuit is obtained by a simple thresholding mechanism. The state space is partitioned in two parts. The equation for the partition is given below:

$$O_{n+1} = \begin{cases} 1, & \text{if } x_{n+1}/x_{n+2} > V_{th} \\ 0, & \text{otherwise.} \end{cases} \tag{3.5}$$

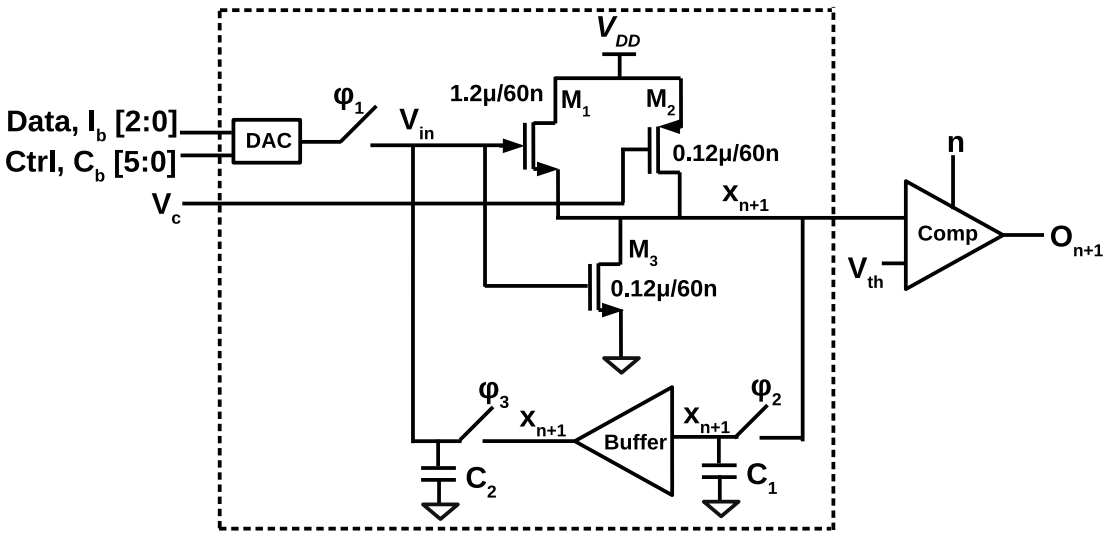


Figure 3.15: Multi-input multi-output reconfigurable chaos-based logic gate [63].

The digital outputs produced by the two map circuits are usually different. The functions change at each iteration if the map circuits are in the chaotic region. If the control inputs put the map circuit in an unstable regime and the orbits are guided towards V_{th}, then the binary output is more susceptible to noise because a small noise can now grow quickly until it switches the final state to an incorrect region producing an incorrect digital output.

3.3.3 Complex Functions Obtained Using Single Chaotic Element

As discussed in section 2.2.5, chaotic oscillators can be used to build multi-input multi-output functions such as adders, decoders etc. Flexibility which means different configurations to achieve the same logic function can be achieved by tuning only a few parameters in the chaotic system. Implementing complex functions by using a single digital system can reduce area and power issues which arise due to technology scaling. The complex instructions generated from chaos-based systems are more uniform than its traditional counterparts. Presence of uniformity makes it difficult for adversaries to decode an instruction by power analysis based side channel attack. This work has explored 3-input multiple output (1 - 8) instructions using a single discrete-time chaos-based system [63]. The tuning parameters in the circuit topology are control bit, threshold voltage of the comparator, iteration number, initial seed value and bifurcation parameter.

The number of functions that a single chaos-based logic gate can implement increases with increasing the number of iterations. It is seen that when the system is operating in chaotic region, a different function is generated in each iteration. Other circuit parameters can be changed dynamically in order to implement different functionalities. It should be noted that all the generated functions will not be reliable or usable due to noise or instability.

Fig. 3.15 shows the design of multi-input and multi-output reconfigurable chaos-based logic gate. The dotted lines enclose the three-transistor map circuit, DAC, buffer and sample-and-hold circuits. A 3-input gate is designed whose inputs are 3-bit data (D), 6-bit control (C) and bifurcation parameter, V_c. A 9-bit digital-to-analog converter (DAC) is required to convert the data bits and control bits to the initial state of the map. The analog output of the map circuit is sampled at each iteration, n and converted to digital voltage using a threshold voltage, V_{th} in the comparator. The size of the transistors in the three transistor map circuit are - M_1: 1.2 μm/60 nm and M_2, M_3: 0.12 μm/60 nm. The size of the transistors are chosen after simulating with several different geometries and finding the optimum case in regards to area, power, width of the chaotic region in the bifurcation diagram and delay.

In order to generate the chaotic signals, an oscillator is required so that the output of the map circuit is fed back to the input. The feedback mechanism requires a buffer, two capacitors and two transmission gates. ϕ_1 is used to apply the initial input to the map circuit. When ϕ_2 is high, the analog output of the map circuit is sampled onto capacitor, C_1 and when ϕ_3 is high, the output is held at C_2.

The functionality generated from the chaos-based logic gate can be varied by changing the threshold voltage, control bit, iteration number and bifurcation parameter. The three transistor map has been designed in IBM 65 nm process. All the other blocks, DAC, comparator, buffer, switches and clocking circuitry has been designed using Verilog-A. The entire reconfigurable logic gate has been simulated using Cadence Spectre.

The input-output values for different values of V_c is obtained from Cadence Virtuoso. A MATLAB code has been developed which varies the four tuning parameters of the design and searches for the configuration of the instruction required. The number of iterations has been contrained within 21 iterations. Higher iteration number might be distorted by noise so it is wise to decide on a suitable limit for n. The evolution of 3-input 1-output AND

Table 3.1: Evolution of chaotic oscillator output with number of iterations. (V_c= 690 mV, $C = (111010)_2 = 58$, $V_{th} = 1.03\ V$, $n = 5$) [63].

x_n (V)	x_{n+1} (V)				
	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$
0.136(000)	1.2(1)	0.52(0)	0.36(0)	1.06(1)	0.43(0)
0.287(001)	1.19(1)	0.51(0)	0.39(0)	0.91(0)	0.35(0)
0.437(010)	0.72(0)	0.27(0)	1.19(1)	0.51(0)	0.38(0)
0.587(011)	0.24(0)	1.2(1)	0.52(0)	0.37(0)	1.02(0)
0.737(100)	0.27(0)	1.19(1)	0.51(0)	0.38(0)	0.96(0)
0.888(101)	0.34(0)	1.13(1)	0.47(0)	0.56(0)	0.25(0)
1.038(110)	0.42(0)	0.79(0)	0.3(0)	1.18(1)	0.51(0)
1.188(111)	0.51(0)	0.39(0)	0.93(0)	0.36(0)	1.04(1)

Table 3.2: Different configurations for 3-input 1-output instructions [63].

Operation	**Configurations**											
	Config.1				**Config.2**				**Config.3**			
	V_c	C	V_{th}	n	V_c	C	V_{th}	n	V_c	C	V_{th}	n
AND	0.69	58	1.02	5	0.74	63	1.17	3	0.62	0	0.69	4
OR	0.71	40	0.3	3	0.65	26	0.3	5	0.62	46	0.27	8
XOR	0.65	0.42	28	5	0.67	41	0.51	7	0.73	63	0.5	15
NAND	0.74	63	0.32	5	0.66	61	0.34	4	0.62	0	0.4	3
NOR	0.63	34	0.61	7	0.69	54	1.17	8	0.74	63	1.19	1
XNOR	0.65	28	0.69	6	0.66	63	0.81	15	0.69	63	0.82	13

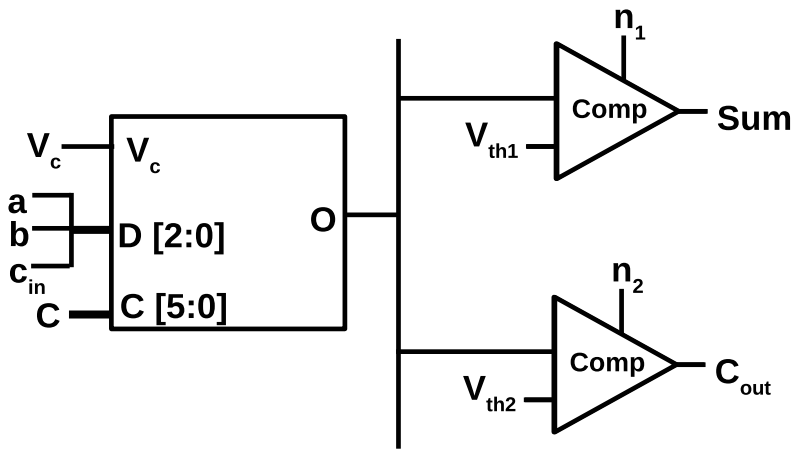


Figure 3.16: 1-bit full-adder designed using chaotic oscillator [63].

Table 3.3: Design of 1-bit full-adder using chaotic oscillator (V_c= 677.5 mV and $C = (011001)_2 = 25$).

Input (a, b, c_{in})	$V_{th} = 497.5\ mV$	$n = 16$	$V_{th} = 502.5\ mV$	$n = 17$
	Sum		**Carry**	
0.0587(000)	0.4954(0)		0.5023(0)	
0.2090(001)	0.5095(1)		0.4394(0)	
0.3593(010)	0.5146(1)		0.4170(0)	
0.5096(011)	0.4287(0)		0.8044(1)	
0.6599(100)	0.9574(1)		0.3767(0)	
0.8102(101)	0.2458(0)		1.1961(1)	
0.9605(110)	0.4153(0)		0.8636(1)	
1.1108(111)	1.1962(1)		0.5148(1)	

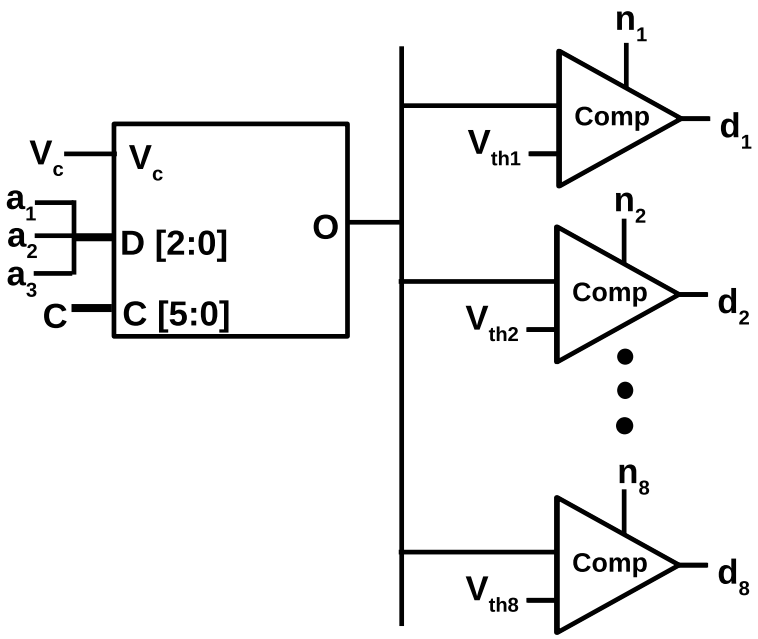


Figure 3.17: 3×8 decoder designed using chaotic oscillator [63].

gate is shown in Table 3.1 column 5. Table 3.2 shows different configurations for generating standard 3-input 1-output instructions using the chaotic oscillator.

Multi-output functions/instructions like adder, subtractor, decoder and encoder uses a common pair of V_c and C for all the output functions. For example, a 3-to-8 decoder uses the same V_c and C but changes V_{th} and n to generate the 8 possible outputs. This is how it is possible to use a single chaotic element to generate multiple instructions. The schematic of 1-bit full adder and 3×8 decoder is shown in Fig. 3.16 and Fig. 3.17 respectively. Table 3.3 shows one of the configurations for designing 1-bit full adder and the generated analog voltages. Table 3.4 shows the configuration for the design of 3-to-8 decoder where V_c and C has been kept constant.

Table 3.4: Design of 3×8 decoder using chaotic oscillator (V_c= 710 mV and $C = (011001)_2$ = 25).

Input (a_1, a_2, a_3)	**Decoder Output**	**Configuration**	
		V_{th} (V)	n
0.0587(000)	d_1	1.1875	11
0.2090(001)	d_2	0.9150	19
0.3593(010)	d_3	0.5150	3
0.5096(011)	d_4	1.14	7
0.6599(100)	d_5	1.1675	2
0.8102(101)	d_6	0.7475	9
0.9605(110)	d_7	0.4975	6
1.1108(111)	d_8	1.18	4

Chapter 4

Expansion of Functionality Space Using Three Transistor Chaotic Map

****Portions of this chapter were published in:**

[150] Aysha S. Shanta, Md Badruddoja Majumder, Md Sakib Hasan, Mesbah Uddin, and Garrett S. Rose. "Design of a reconfigurable chaos gate with enhanced functionality space in 65nm cmos." *In 2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 1016-1019. IEEE, 2018.

4.1 Introduction

Despite its tremendous success, binary computers fail to meet the needs of today's requirements and specifications. A binary computer contains arrays of transistors which acts as switches. In 1965, Gordon Moore predicted that the density of transistors in a chip will double every two years [102]. The growth in the transistor count ensured that computer hardware is able to meet the needs of the trending applications. Nowadays, it is becoming increasingly difficult to increase the density of transistors as Moore's law predicted. Moreover, with increasing density of transistors in a chip, heat production and sink and power consumption are becoming huge challenges. Mobile applications and embedded systems are examples of applications that require low power consumption. Researchers have invented

new ways to improve the performance of computing systems without increasing the number of transistors. One such alternative was to increase the amount of computation performed by devices without adding more hardware [81]. Multi-valued digital circuits have been developed to increase bit-handling capacity of each device [138].

Instead of studying transistors as binary switches, the nonlinear dynamics in the transistor circuits and their ability to perform intrinsic computation should be analyzed. Dynamics of a circuit can be defined as mapping initial states to future states. A nonlinear circuit can have complex dynamics and contain many different functions. Each different function generated means that the mapping of initial state to future state is different. It is seen that the intrinsic functions present in a nonlinear system is able to generate many logic functions using the same circuit. The data and the control bits are combined to make the initial conditions of the system and finally the final state is converted to a binary value using a comparator.

In [85], it has been shown that a simple nonlinear circuit contains an infinite number of functions. The number of logic functions that a chaotic circuit can implement increases exponentially with time. All the functions generated from the circuit are not accessible due to issues such as noise and instability of the functions. The chaos-based computing system opens up options for designing extremely low-power circuits that are capable of performing multiple functions using the same system. The nonlinear dynamical system can be seen as a reconfigurable computing system where different functions can be selected by changing parameters in the circuit topology. The simplest way of selecting a new function is changing the initial condition of the dynamical system which is made up of data and control bits. Data bits are the logic inputs that is applied to the chaos-based logic gate and control bit alters the internal mapping of the system to select a logic function. The output of the map is fed to a threshold circuit which generates a 1-bit digital output.

4.2 Expansion of Design Space

A conventional CMOS logic gate requires less number of transistors compared to reconfigurable chaos-based logic gate. One possible way to compensate for this lacking is to make

sure that the functionality space of the chaos-based gate is huge. This work proposes to increase the design space by varying the threshold voltage of the comparator and using a new bias voltage in each iteration. The total number of functions generated exponentially increases compared to previous work found in literature which only demonstrates linear growth. The number of individual functions such as OR, XOR and AND also increases with the increase in design space.

The characteristics of the chaotic map and oscillator are discussed in sections 3.1 and 3.2 respectively. The characteristic of the map circuit can be changed by changing the bias voltage, V_c as shown in Fig. 3.2. Fig. 3.4 represents the chaotic oscillator where clocks ϕ_2 and $\overline{\phi_2}$, are two non-overlapping clocks to control the feedback in the loop containing the two map circuits. Clock ϕ_1 initializes the system to an initial condition and the map starts iterating. When the map is in the long period regime or in chaotic region, at each iteration a new set of outputs are produced and the generated sequence has a long period. Iteration number is another method to tune the circuit to implement different logic functions. Since there are two maps in the chaotic oscillator, two bias voltages can be applied to generate different logic functions. Although, in this work, only one bias voltage has been used by shorting the bias voltages together.

4.2.1 Comparison of Area and Power Overhead

This work has been simulated in 65 nm CMOS process [150]. The supply voltage used in this technology node is 1.2 V. The chaotic oscillator has smaller area and power consumption compared to previous work found in literature. The previously designed chaotic oscillator was designed in 0.6 μm process and used a supply voltage of 5 V. The capacitors for sample-and-hold circuits are implemented using the internal capacitance of the map circuits. The absence of capacitors in the design leads to huge saving in the chip area.

The total power consumption of the chaotic oscillator in this work is 18.4 μW whereas the power consumption in the 5 V process was 7.85 mW. The consumption in the 5 V is higher since huge amount of current flows from V_{DD} to ground as the applied input voltage increases. The consumed area of the proposed design is only 0.556 μm^2 whereas the area

Table 4.1: Overhead comparison of proposed work with previous work [150].

	Previous Work [80]	**This Work**
Technology	0.6 μm	65 nm
Supply Voltage	5 V	1.2 V
Area	32×19 μm	0.556 μm^2
Power	7.85 mW (experimental)	18.4 μW (simulated)

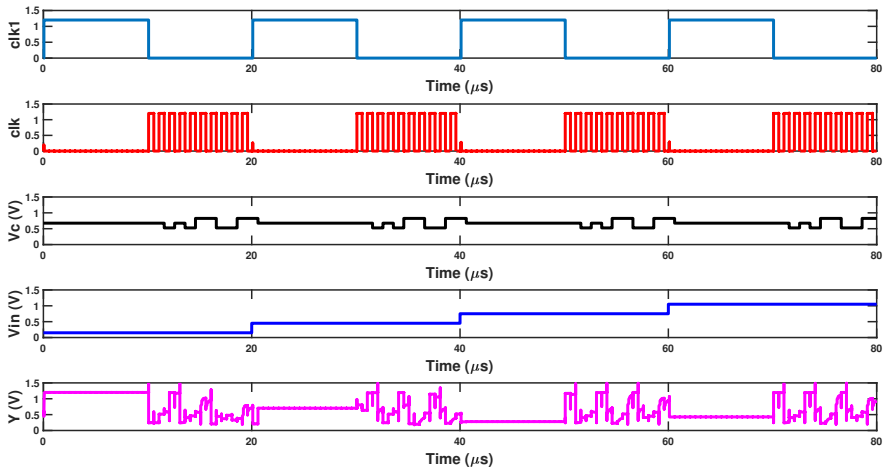


Figure 4.1: Transient response of the chaotic oscillator displaying that V_c is changed cycle to cycle [150].

in 0.6 μm process was 32×19 μm. The calculation of the area does not include the pass transistors. Table 4.1 shows the comparison between the two designs.

4.2.2 Design Space Enhancement

It has been seen in literature that simple nonlinear dynamical systems are able to generate infinite number of functionalities [85]. The term "functionality" does not only represent particular logic functions but also considers multiple ways of achieving the functions. All the functions generated by a dynamical system might not be usable due to instability of the outputs or presence of noise. Kia *et al.* implemented the three transistor chaotic map to produce a large functionality space where the authors mapped the control inputs and the digital inputs using a digital-to-analog converter (DAC) (shown in Fig. 3.13) and converted the analog output of the chaotic oscillator into digital output by using a comparator (shown in Fig. 3.14). The output of the oscillator is compared to a threshold voltage in order to compute the digital value using equation 3.5.

Kia *et al.* increased the functionality space by using 4 data inputs along with 16 control bits. Different digital output can be obtained in different iterations since the output might change from the previous one. Let c be the control input, n be the number of iterations and b represent the bias voltage (V_c). In previous work, the threshold voltage has been fixed to 2.5 V. The equation for representing the functionality space from [85] is as follows:

$$f(n) = b \times \ 2^c \times n. \tag{4.1}$$

From equation 4.1, it can be seen that the functionality space increases linearly with the increase in number of iterations, n. The space is limited by the number of control bits used in the DAC and the number of bias voltage levels for a fixed number of iterations. Moreover, the bifurcation diagram in Fig. 3.8 shows that not all bias voltages lead to chaotic behavior. The number of possible bias voltages that can be used to increase the design space is limited.

The size of the DAC used in the design depends on the number of input and control bits. Although, there is a limitation on the achievable resolution of the DAC due to signal to noise ratio (SNR). This issue poses a limitation on the number of control bits, c that can be used

in the design. The only option left is to have more number of iterations in the design for an optimal choice of values of b and c. As shown by equation 4.1, the increase in n only has a linear effect on the increase of functionality space. This means that the iteration number has to be huge in order to gain larger design space.

In this work, new techniques have been implemented to increase the design space exponentially. The bias voltage is randomly varied cycle to cycle in order to increase the functionality space with increasing iteration number, instead of using a fixed bias voltage. In this design, both the map circuits are connected to the same bias voltage, V_c. However, applying different bias voltage might increase the design space significantly. We have chosen three bias voltages in this design by partitioning 1.2 V into eight partitions. Three bias voltages out of eight have displayed more chaotic behavior than others. The values of the bias voltages used are: 525 mV, 675 mV and 825 mV.

Multiple threshold voltages are applied to the comparator to extract the digital output from the system instead of using a fixed threshold voltage. Four threshold voltage values have been chosen to expand the functionality space which are: 150 mV, 450 mV, 750 mV and 1.05 V. The four analog voltages chosen to represent the inputs, $(0,0)$, $(0,1)$, $(1,0)$ and $(1,1)$ are 150 mV, 450 mV, 750 mV and 1.05 V respectively.

The transient response of the chaotic oscillator with varying bias voltage is shown in Fig. 4.1. Whenever clk_1 is high, a new analog input, V_{in} is applied to the chaotic oscillator. When clk_1 is low, clk runs through different iterations to generate the analog output voltages. As can be seen in the figure, V_c changes randomly in each cycle and the final output, Y demonstrates a chaotic behavior.

The equation for representing the functionality space of the proposed design is as follows:

$$f(n) = b^n \times \ N_{V_{th}} \times n. \tag{4.2}$$

where b is the number of bias voltage levels, $N_{V_{th}}$ is the number of threshold voltage levels and n is the number of iterations.

Fig. 4.2 displays the entire functionality space with increasing number of iterations in the proposed and existing work ([85]). The increase in functionality space with respect to

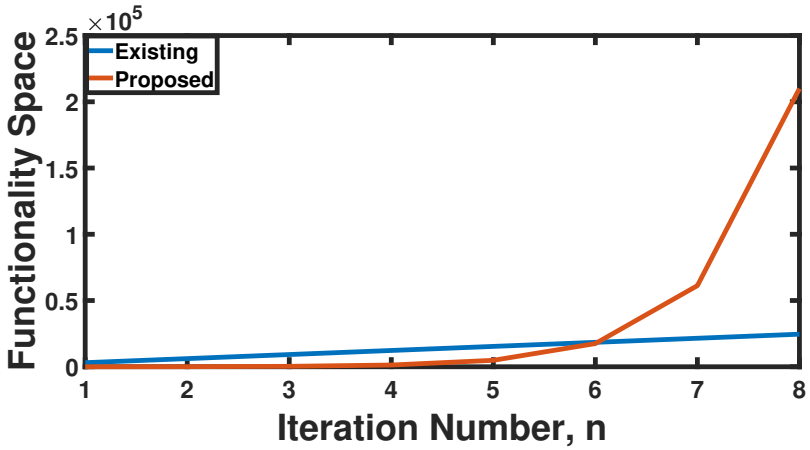


Figure 4.2: Comparison between the functionality space of the proposed work with existing work for varying number of iterations [150].

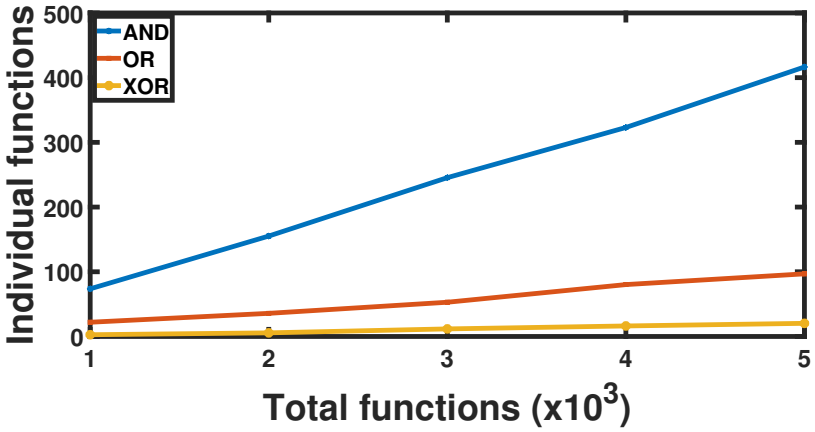


Figure 4.3: Number of individual functions increases linearly with the total functionality space [150].

iteration number is linear for the existing work and represented by the blue curve. On the other hand, the increase in functionality space of the proposed system is exponential and represented by red curve. It can be seen that the functionality space for the first six iterations is higher in existing work. After the 6^{th} iteration, the proposed work picks up and in the 8^{th} iteration, the functionality space has reached almost three times the existing work.

It is important to ensure that the individual functions also increase with the increase in functionality space. In Fig. 4.3, it can be seen that the individual functions, AND, OR and XOR increases with the enhanced design space. This result ensures that the individual functions grow linearly with the increase in iteration number. In order to generate Fig. 4.3, SKILL file has been written where the circuit output has been probed in total of 2000 times for four inputs representing (0,0), (0,1), (1,0) and (1,1). Each input has 10 iterations each. New sequence of V_c has been applied 500 times and a total of 2000 ($= 500 \times 4$) sequences are generated. The file is imported into Matlab where four different threshold voltages are applied at least 20 times after the sequence is varied and the average result of the total functions is plotted.

4.3 Application

Security is an important design consideration in modern digital systems. Side channel attack is an attack mounted on a system where the attacker collects information about the power profile of several instructions and tries to learn the content of the processor data. Researchers have proposed several strategies to mitigate against side channel attack. Chaos-based systems can help to mitigate power analysis based side channel attack due to the enhanced design space. Chaotic computations generates a random power signature which is difficult to decipher by the attacker.

Chapter 5

Four Gate Transistor Negative Differential Resistance (NDR) Based Discrete-Time Chaotic Map

5.1 Background

It is becoming very difficult to integrate more transistors on a chip as technology scaling has slowed down in the past decade. Researchers have dived into producing new solutions to achieve higher performance without increasing the transistor count. One possible solution is to develop a system or a device that can perform multiple computations [86]. G^4FET is a silicon-on-insulator (SOI) four gate device that has been utilized in various digital and analog applications since it requires less number of transistors [8]. SOI technology mitigates a lot of issues associated with bulk silicon scaling. The advantages of using SOI technology are reduced parasitic capacitance, increased switching speed, ideal device isolation, resolved latch-up issue, excellent sub-threshold slope and reduced leakage current [65].

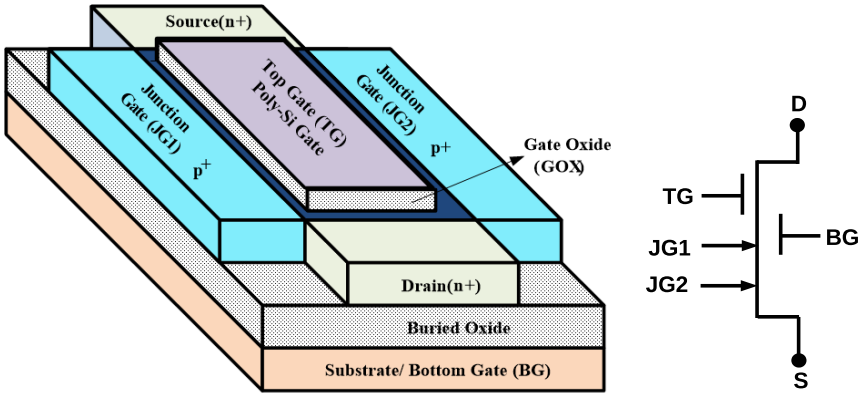
Some of the traditional chaotic maps are logistic map, tent map and sine map. These maps incur a lot of hardware overhead since they are represented by ideal mathematical equations. This is why researchers have dived into discovering new transistor level maps with good chaotic behavior. A large functionality space is available from the transistor level

chaotic maps and these functions can be leveraged in the field of hardware security. Most of the chaotic maps in literature which require less hardware can only provide one bifurcation parameter. The functionality space is mainly extended by increasing the iteration number which was seen in section 4.2.2. However, there is a limitation on how much the iteration number can be increased. As the number of iterations increase, the reliability of the system reduces and it becomes more susceptible to noise. It has become essential to develop a discrete-time chaotic map with simple hardware whose functionality space can be increased by utilizing the various control knobs in the designed system. In this work, a G^4NDR based chaotic map is proposed whose transfer characteristics is similar to that of logistic map. The map demonstrates excellent chaotic property and has wider chaotic region in the bifurcation diagram compared to the three-transistor map used in previous chapters 3 and 4.

5.2 Four Terminal Transistor (G^4FET)

G^4FET is a multiple terminal SOI device which is used in applications where less transistors are required. The designed circuits are able to produce more functionality [23, 9, 8]. The device can be fabricated in partially/fully depleted SOI process. The channel conductance can be altered using four terminal gates that can be independently biased [47]. A traditional inversion-mode p-channel SOI MOSFET with two body terminals on the opposite sides of the channel can be used to construct an accumulation/depletion mode n-channel G^4FET.

The symbol and structure of the n-channel G^4FET is shown in Fig. 5.1. The cross-section of the G^4FET device displaying all the four terminals is shown in Fig. 5.2. The junction gates of G^4FET are formed from the p^+ doped source and drain of the traditional MOSFET. The channel width can be controlled using the two lateral gates whose functionality is similar to JFET gates. The channel width can be decreased by reverse-biasing the junction gates. The top gate (TG) is similar to the conventional accumulation-mode MOSFET gate. The substrate or bottom gate (BG) *imitates* a fourth MOS gate and can be biased using the bottom gate terminal due to the presence of buried oxide. The vertical gates are used to change the accumulation/inversion/depletion of free carriers in the silicon epi layer near the



(a) (b)

Figure 5.1: Four gate transistor (G^4FET) (a) Structure (b) Symbol [7].

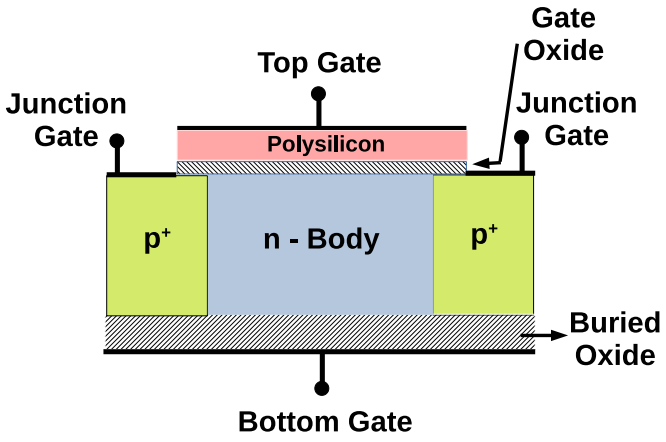


Figure 5.2: Cross-section of the G^4FET device [64].

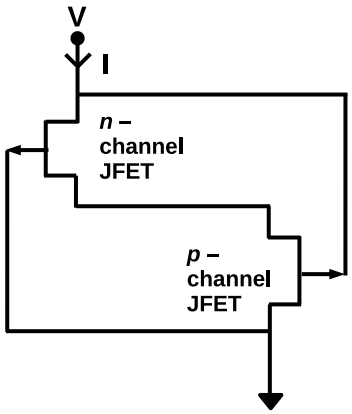


Figure 5.3: Traditional 2-terminal JFET NDR [163].

top/bottom oxide interfaces [62]. The body contacts are highly doped to make ohmic contact with the channel.

The function of G^{4}FET is controlled by two JFET gates and two MOSFET gates that can be biased independently. The conductive path in original MOSFET relates to the gate length whereas the JFET channel length is altered by adjusting the width of the MOSFET. In G^{4}FET, the gate length is the width of the original MOSFET and gate width is defined by MOSFET's gate length. The drain current comprises of majority carriers flowing from one body contact to the other (here, the body contacts of the MOSFET are used as drain and source) and it is inversely proportional to the length of the G^{4}FET. The current in G^{4}FET flows in perpendicular direction to that of an inversion-mode MOSFET. G^{4}FET combines both MOS and JFET characteristics by allowing both surface and volume conduction. The top and bottom gates work like MOSFET while the lateral gates work as JFET. The threshold voltage of the vertical gates are influenced by the lateral gates. There are three conduction paths in G^{4}FET. They are:

1. Top surface conduction near gate-oxide interface.

2. Bottom surface conduction near buried-oxide interface.

3. Volume conduction away from vertical oxide interfaces.

G^{4}FETs provide the flexibility to design multiple input systems with reduced transistor count in comparison to its CMOS counterparts. CMOS technology scaling should enhance the performance of the G^{4}FET transistor as MOS gate length shrinks. This means that the channel will be easier to pinch-off if body doping is optimized [23]. Moreover, it does not require special fabrication steps to manufacture the device.

5.2.1 G^{4}FET Operation

The G^{4}FET works as an accumulation mode MOSFET and it has two junction gates on opposite sides of the channel. The junction gates act as JFETs by altering the potential distribution within the body by utilizing the induced lateral depletion regions. In a partially-depleted body, if the amount of reverse-bias voltage ($|V_{JG}|$) applied to the junction gates

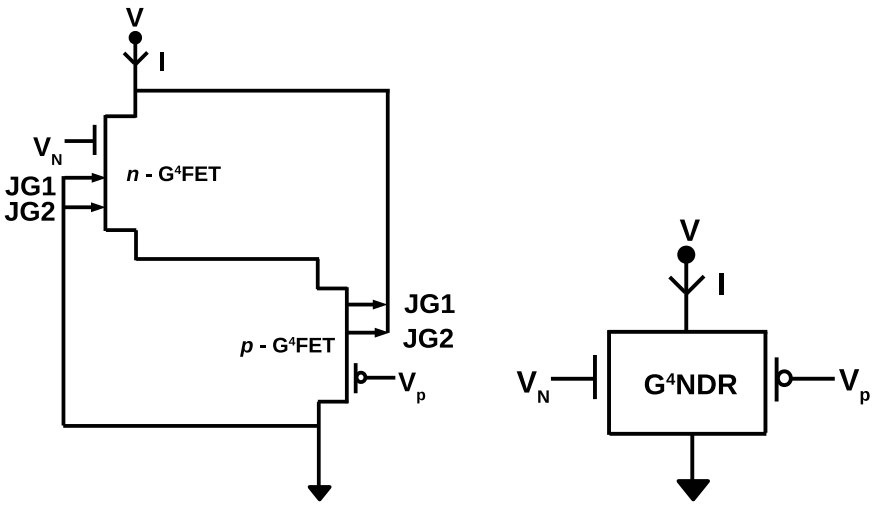
is high, it can switch the transistor operation from normally-on to normally-off. If $|V_{JG}|$ is increased further, it can modulate the threshold voltage associated with TG.

When $V_{JG} = 0\ V$, the transistor is normally-on because the electron concentration (depends on the doping level) in the body enables conduction of carriers. As V_{JG} is decreased from $0\ V$ to a negative value, e.g. $-1.5\ V$, depletion region in the junction gates increase in width and start suppressing the flow of carriers in the body which in turn reduces the current. If $V_{JG} = -2\ V$, the G^{4}FET becomes normally-off. At this point, the body is fully-depleted and the drain current depends only on the electrons accumulated at the front gate which is controlled by the voltage, V_{TG}. If $V_{JG} < -2\ V$, the body potential reduces and charge coupling occurs between front gate and junction gates making it harder to accumulate electrons at the front-interface. This phenomenon results in an increase in threshold voltage.

In an n- channel G^{4}FET, when the top gate is biased negatively relating to a maximum depth of the depletion region, the conductive channel shrinks and repelled towards the back gate. The front interface is depleted of carriers and the conductive path reduces gradually. At a certain point, the path vanishes and the device turns off completely.

When the back-gate is in strong accumulation-mode, the device will not be turned-off even when large negative voltages are applied at the junction gates and the front gate. However, the lateral gates and the top gate have control on the current level. The G^{4}FET can be used in this mode for high current applications. When negative voltage is applied to back-gate, the back interface is depleted of carriers and the channel thickness decreases. The device can be easily turned off even when moderate values of voltages are applied to the lateral gates and top gate [34]. When the back interface and lateral gates are depleted, the conductive path will not be able to cover the whole body of the transistor, even if the top gate is accumulated.

In a partially depleted SOI MOSFET, it is not possible for the vertical gates to achieve full depletion. The current can only be switched off by adding the lateral junctions and keeping the body narrow in a G^{4}FET. The coupling of depletion and vertical regions allow for full depletion even before the lateral or vertical depletion regions merge.



(a) (b)

Figure 5.4: Tunable NDR made with n- channel and p- channel G^4FETs (a) Circuit (b) Symbol [9].

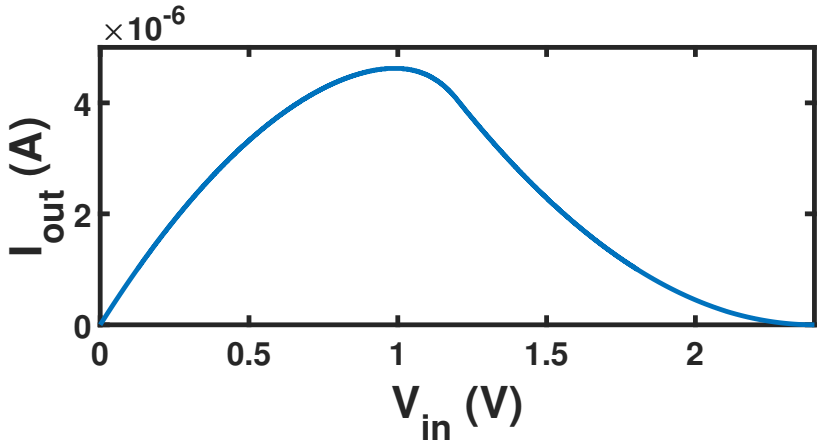


Figure 5.5: Transfer characteristics of G^4NDR.

5.3 G^{4}FET Based Negative Differential Resistance

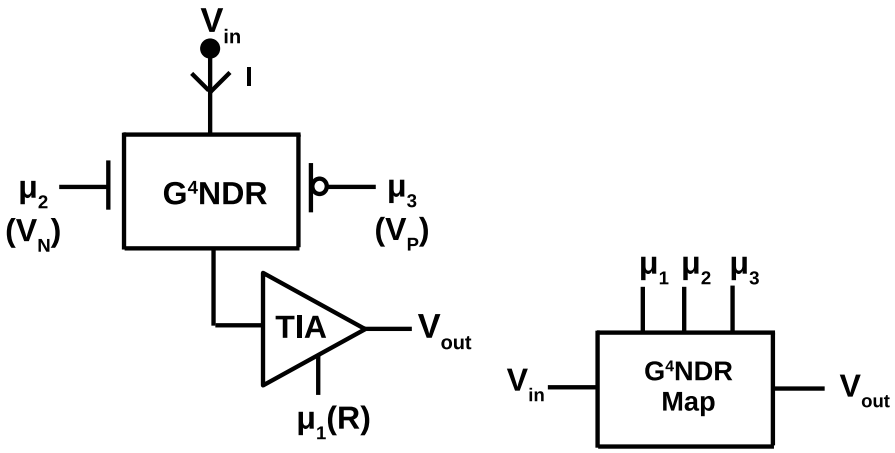
Resistance is always present in electronic devices. The resistance of materials change with varying temperatures. Most materials obey Ohm's law at ambient temperature but some tend to show a negative resistance. Ohms's law states that in order to determine the resistance of a component, voltage should be applied across it and current is measured flowing through it. The resistance equals to the voltage divided by the current as shown in the following equation:

$$R = V/I. \tag{5.1}$$

Many objects have a constant resistance and the current tends to display a linear relationship with changes in the applied voltage. The linear relationship holds true only if the temperature is kept constant. If the temperature is varied slightly, obtained resistance values will be different. All the devices do not seem to portray a linear relationship. For example, a diode made with silicon dioxide has very high resistance for applied voltages below 700 mV. It is possible for devices to demonstrate different values of resistance as the applied voltage is altered. The Ohm's law holds true for these cases because the resistance is measured with constant circuit parameters. This is an example of static resistance because the voltage is kept constant while current is measured [42].

Dynamic resistance refers to a resistance that is measured due to changes in circuit parameters. Therefore, dynamic resistance is defined as the change in voltage divided by the resulting change in current. In a resistor, the dynamic resistance and static resistance are equal but in a diode the dynamic resistance is a function of the applied voltage. Most of the devices have positive dynamic resistance but there are devices which display a negative dynamic resistance. Negative dynamic resistance happens when the current starts decreasing with increase in the applied voltage although the measured static resistance will be positive for a given voltage.

Tunnel diode might be the only device whose physical characteristics display negative resistance due to its physical mechanism. The traditional lambda diode consists of complementary JFETs and it exhibits a region with negative differential resistance [163]. The



(a) (b)

Figure 5.6: G^{4}NDR based discrete-time chaotic map which has three bifurcation parameters (a) Circuit (b) Symbol.

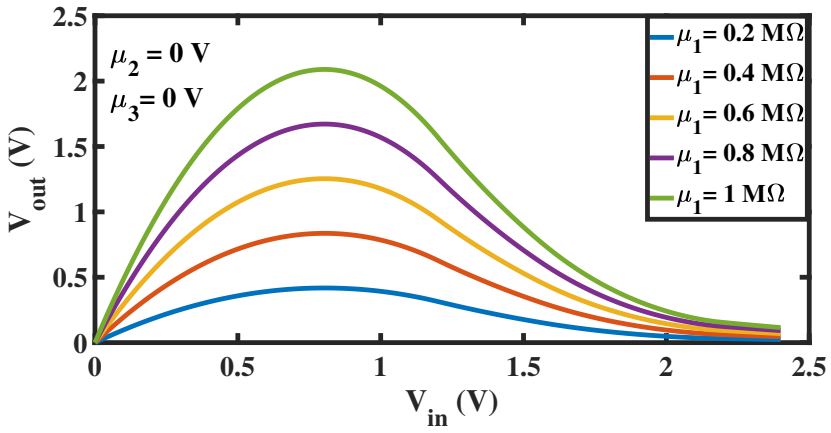


Figure 5.7: Transfer curve of the G^{4}NDR map circuit for varying μ_1.

peak currents and peak voltages are dependent on the pinch-off voltage and transconductance of the transistors. The current in lambda diode approaches zero as voltage increases until it reaches a breakdown point of the FET. At this point, the current starts increasing again. The conventional two-terminal NDR is shown in Fig. 5.3. The n- channel JFET is fabricated using two types of diffusion, n^+ diffusion for the source and drain contacts and p diffusion for the gate contact. Similarly, the p-channel JFET has n^+ diffusion for the gate and back-gate contact while the p diffusion forms the source and drain contacts.

A voltage-controlled negative differential resistance (VC-NDR) circuit using n-channel and p-channel G^{4}FETs was first proposed in [9]. G^{4}NDR has been developed using the same idea but the JFETs are replaced with complementary G^{4}FETs. The circuit and symbol of G^{4}NDR is shown in Fig. 5.4. The resulting current changes with applied voltage is shown in Fig. 5.5. The junction gates of the G^{4}FETs are tied together and act as a single gate in the NDR. The back-gate is connected to ground and not shown for simplicity. The resulting G^{4}NDR is a four-terminal device which is made up of the top gates and the tied junction gates. The top gate voltages are denoted by V_N and V_P respectively. In G^{4}NDR, the transconductance and pinch-off voltage vary with the applied voltage on the MOS front gates. This is why G^{4}NDR properties can be tuned using both V_N and V_P. If V_P increases, the peak current and valley voltage of the G^{4}NDR decreases since the pinch-off voltage of the G^{4}FET is reduced. V_N controls the conductance of n- channel G^{4}FET and modulates the peak current since the front gate oxide interface changes from accumulation to depletion. The sizes used for G^{4}FETs are p- channel: 0.35 μm/1.2 μm and n- channel: 0.35 μm/3.4 μm. All the simulations have been performed using the MOS-JFET macromodel in [65].

5.4 G^{4}NDR Based Chaotic Map

A novel chaotic map has been designed by implementing a voltage-controlled negative differential resistance (VC-NDR) circuit consisting of two G4FETs and a transimpedance amplifier which converts the generated current to voltage using a variable gain, R. The transfer characteristics of the G^{4}NDR map is very similar to the behavior of logistic map. The map circuit and symbol are shown in Fig. 5.6. G^{4}NDR map is better than traditional chaotic

maps such as tent map, sine map or logistic map. Traditional maps require high overhead to implement the idealized mathematical equations in hardware. Moreover, G^{4}NDR map has three independent bifurcation parameters (μ_1, μ_2, μ_3) instead of one parameter provided by the conventional maps and the three transistor map. The proposed chaotic map shows excellent chaotic property and the bifurcation diagram shows wider chaotic regions. Fig. 5.7 shows the transfer characteristics of the proposed map where μ_1 is varied from 0.2 $M\Omega$ to 1 $M\Omega$ and μ_2 and μ_3 are fixed at 0 V. The differentiable unimodal characteristic of the transfer curve is ideal for chaotic operation.

5.4.1 Design of Chaotic Oscillator Using G^{4}NDR Based Map

Fig. 5.8 shows the chaotic oscillator with sample-and-hold circuits and a buffer in the feedback whereas the oscillator in Fig. 5.9 utilizes another G^{4}NDR map in the feedback path. When the chaotic oscillator is in chaotic region, it is capable of generating a new analog voltage in every iteration. The oscillator in Fig. 5.8 works by applying an initial input through clock, ϕ_1. After application of the initial value, ϕ_1 is disabled and clocks, ϕ_2 and ϕ_3 are used to apply the generated output of the G^{4}NDR map to the input of the forward path map. Only one output can be sampled in each iteration. The oscillator in Fig. 5.9 works in a similar way but the second map in the feedback path is able to generate a new chaotic value which can be sampled as well.

5.4.2 Bifurcation Diagram and Lyapunov Exponent

The importance of bifurcation diagrams and Lyapunov exponent in chaotic systems has been discussed in detail in section 3.2. As mentioned earlier, the G^{4}NDR chaotic map has three bifurcation parameters, μ_1, μ_2 and μ_3. The bifurcation diagrams for the three parameters are shown in Fig. 5.10a, 5.11a and 5.12a. In Fig. 5.10a, the bifurcation diagram has been simulated for changing μ_1. The first 1000 iterations are ignored because the values are in transient state. The three bifurcation diagrams clearly demonstrate the periodic and chaotic regions. The corresponding Lyapunov exponent diagrams are shown in Fig. 5.10b, 5.11b and 5.12b. It can be seen from the figures that the Lyapunov exponent has a positive value

whenever the chaotic oscillator is in chaotic region. Lyapunov exponent demonstrates zero or negative value for periodic regions.

5.5 Design of Logic Gates Using G^{4}NDR Based Map

Previously in section 4.2.2, the design space has been increased by varying the bifurcation parameter in each iteration and changing the threshold voltage of the comparator. The procedure to design reconfigurable chaos-based logic gates has been discussed in section 3.3.2. The schematic for the design of chaos-based gate is shown in Fig. 5.13 where the DAC is used to convert the digital input to analog voltage which is the initial seed (x_0) for the chaotic oscillator. A control bit, C_b is used to change the initial value of the seed slightly in order to generate a different chaotic sequence. Data, I_b is the digital input applied to the logic gate. At each iteration, the analog voltage from the chaotic oscillator is converted to a digital output by using a comparator with threshold voltage, V_{th}.

This work focuses on designing a 2-bit chaos-based logic gate with 1-bit control. Therefore, a 3-bit DAC is needed to convert the initial digital voltage to an analog value. The chaos-based logic gates are reconfigurable and are capable of implementing all the 16 functions. The functions are numbered in decimal starting from 0 (0000) to 15 (111). Table 5.1 demonstrates the evolution of analog voltages from the chaotic oscillator with changing iteration number. The values of the tuning parameters are $\mu_1 = 0.95\ M\Omega$, $\mu_2 = 0\ V$, $\mu_3 = 0\ V$, $V_{th} = 1.25\ V$ and $C_b = 0$. The table shows that the functions vary with each iteration. For example, when iteration number , $n = 4$, the generated function is NAND and the decimal value is 14 (1110). Multiple configurations are possible in order to obtain the logic functions from the chaotic oscillator. Three different configuration for six different functions (AND, OR, XOR, NAND, NOR and XNOR) are shown in Table 5.2.

5.6 Expansion of Design Space Using G^{4}NDR Map

The benefits of expanding the design space has already been discussed in chapter 4.

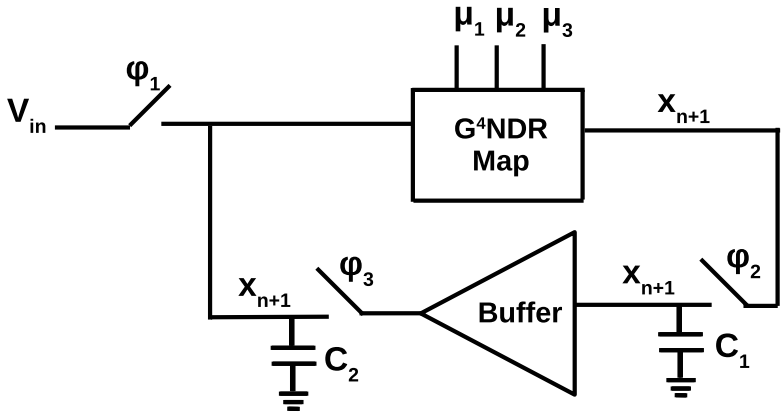


Figure 5.8: G^{4}NDR based chaotic oscillator using buffer in the feedback.

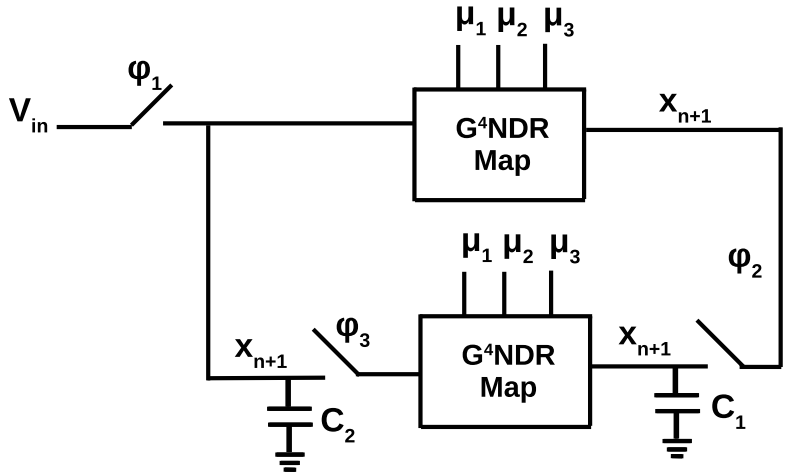
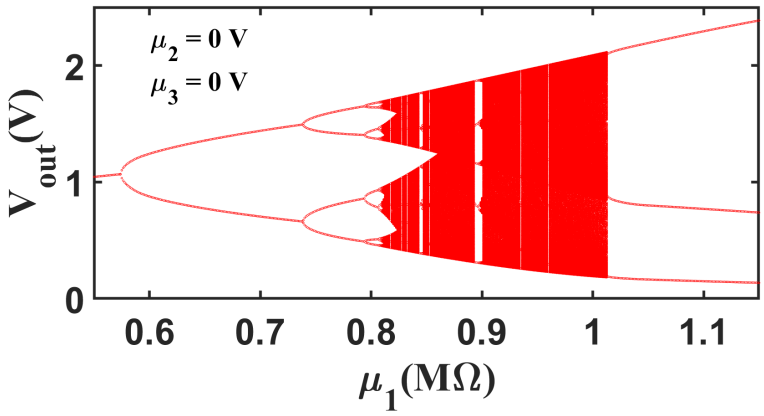
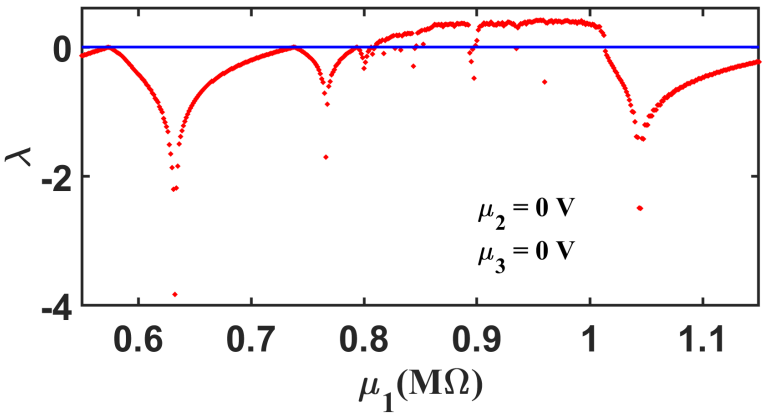


Figure 5.9: G^{4}NDR based chaotic oscillator using another G^{4}NDR map in the feedback.

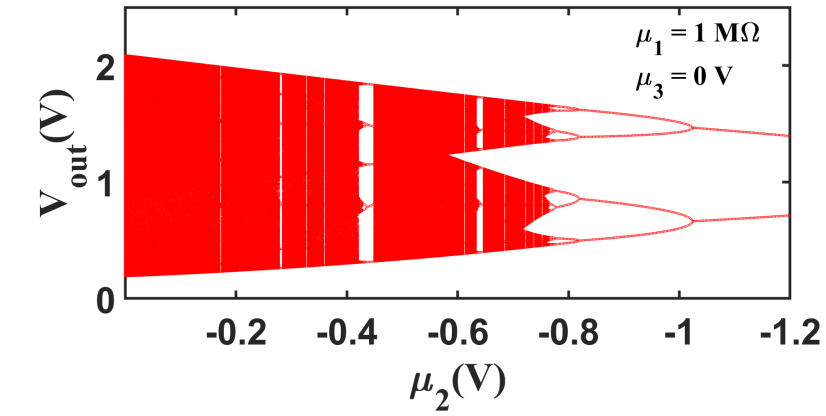


(a)

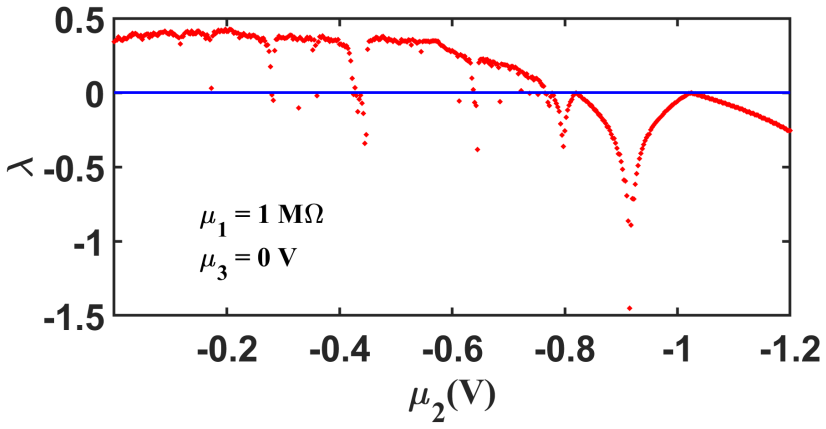


(b)

Figure 5.10: (a) Bifurcation diagram and (b) Lyapunov exponent of the chaotic oscillator for varying μ_1; $\mu_2 = 0\ V$, $\mu_3 = 0\ V$.

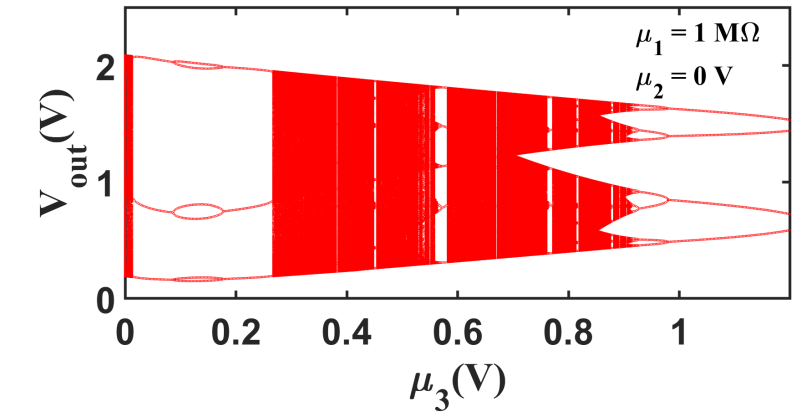


(a)

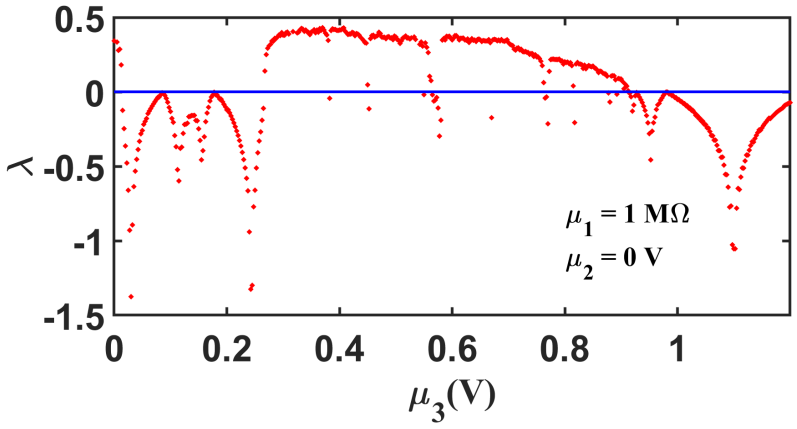


(b)

Figure 5.11: (a) Bifurcation diagram and (b) Lyapunov exponent of the chaotic oscillator for varying μ_2; $\mu_1 = 1\ M\Omega$, $\mu_3 = 0\ V$.



(a)



(b)

Figure 5.12: (a) Bifurcation diagram and (b) Lyapunov exponent of the chaotic oscillator for varying μ_3; $\mu_1 = 1\ M\Omega$, $\mu_2 = 0\ V$.

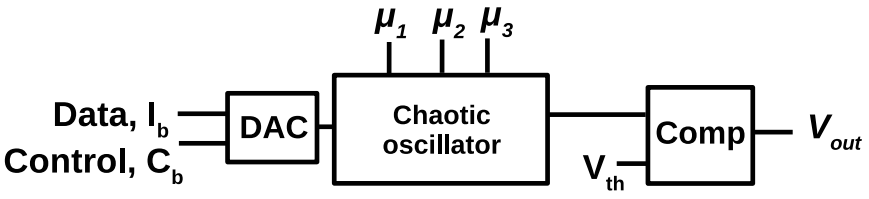


Figure 5.13: G^{4}NDR based reconfigurable logic gate.

In [80], the parameters that are changed to enhance the design space are control bit, c, number of iterations, n and bifurcation parameter, μ. Let N_μ be the available number of voltage levels in the bifurcation diagram where the system is chaotic. The functionality space, F_1(n) in [80] can be expressed as:

$$F_1(n) = n \times 2^c \times N_\mu. \tag{5.2}$$

In [150], control bit is not used but the threshold voltage is varied. The functionality space was extended by varying the bifurcation parameter in each iteration. Let N_{vth} be the number of threshold voltage levels. The functionality space, F_2(n) in [150] can be expressed as:

$$F_2(n) = n \times N_{vth} \times N_\mu^n. \tag{5.3}$$

The three transistor chaotic map shown in Fig. 3.1 has one bifurcation parameter. On the other hand, the proposed G^{4}NDR based chaotic map has three independent bifurcation parameters which can be leveraged to increase the functionality space. There are two types of oscillator that utilizes G^{4}NDR based chaotic map. The oscillator in Fig. 5.8 has only one chaotic map in the forward path. There are 6 tuning parameters in the design, bifurcation parameters (μ_1, μ_2 and μ_3), control bit (c), iteration number (n) and threshold voltage (V_{th}). The bifurcation parameters are varied cycle to cycle. The functionality space, F_3(n) for the chaotic oscillator in Fig. 5.8 can be expressed as:

$$F_3(n) = N_{v_{th}} \times 2^c \times N_{\mu_1}^n \times N_{\mu_2}^n \times N_{\mu_3}^n \times n. \tag{5.4}$$

The functionality space can be further expanded if the chaotic oscillator in Fig. 5.9 is used which utilizes another chaotic map in the feedback path. The forward path map and the feedback path map each has three independent bifurcation parameters. The functionality space, F_4(n) for the chaotic oscillator in Fig. 5.9 can be expressed as:

$$F_4(n) = N_{v_{th}} \times 2^c \times N_{\mu_1}^{2n} \times N_{\mu_2}^{2n} \times N_{\mu_3}^{2n} \times n. \tag{5.5}$$

The functionality space for the four equations 5.2 - 5.5 is shown in Fig. 5.14. The functionality space is significantly larger for G^4NDR based chaotic map due to the presence of multiple bifurcation parameters. The bottom gates of the G^4FET can be utilized to further extend the design space instead of connecting them to ground terminal. It is highly desirable to have a large design space so that multiple reliable functions can be chosen to implement a single logic function. This feature can be utilized to mitigate power analysis based side channel attacks [105].

Table 5.1: Evolution of analog output from chaotic oscillator with different iterations ($\mu_1 = 0.95\ M\Omega$, $\mu_2 = 0$ V, $\mu_3 = 0\ V$, $C_b = 0$, $V_{th} = 1.25\ V$). Functions are represented in decimal value.

x_n (V)	x_{n+1} **(V)**				
	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$
0.1(00)	0.46(0)	1.62(1)	0.63(0)	1.89(1)	0.31(0)
0.757(01)	1.98(1)	0.24(0)	1.02(0)	1.84(1)	0.35(0)
1.414(10)	1.01(0)	1.86(1)	0.34(0)	1.33(1)	1.2(0)
2.071(11)	0.19(0)	0.82(0)	1.98(1)	0.24(0)	1.01(0)
Func (dec)	4	10	1(AND)	14($NAND$)	0

Table 5.2: Three different configurations for six logic functions.

Operation	**Configurations**																	
	Configuration 1						**Configuration 2**						**Configuration 3**					
	$\mu_1(M\Omega)$	$\mu_2(V)$	$\mu_3(V)$	C_b	$V_{th}(V)$	n	$\mu_1(M\Omega)$	$\mu_2(V)$	$\mu_3(V)$	C_b	$V_{th}(V)$	n	$\mu_1(M\Omega)$	$\mu_2(V)$	$\mu_3(V)$	C_b	$V_{th}(V)$	n
AND	0.97	0	0	0	1.1	3	0.94	-0.3	0	1	1.5	5	0.99	-0.3	0.3	0	1.4	7
OR	0.94	0	0	1	1.2	7	26	0.96	-0.3	0	0	0.7	0.99	-0.3	0.3	0	0.7	5
XOR	0.95	0	0	1	0.6	4	41	0.97	-0.3	0	0	0.7	0.99	-0.3	0.3	0	0.7	1
NAND	0.95	0	0	0	1.4	6	61	0.95	-0.3	0	1	0.5	0.99	-0.3	0.3	1	1.1	5
NOR	0.93	0	0	1	1.4	6	54	0.9	-0.3	0	0	1.4	0.99	-0.3	0.3	1	1.6	7
XNOR	0.98	0	0	1	1.6	3	63	0.92	-0.3	0	1	1.4	0.99	-0.3	0.3	0	1.6	6

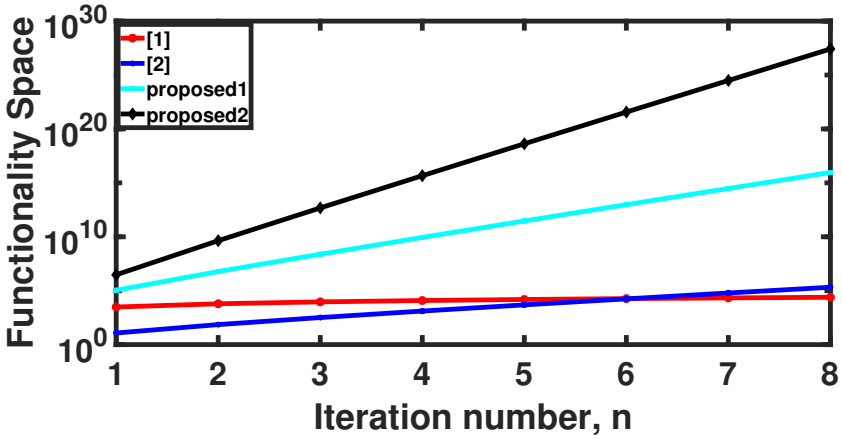


Figure 5.14: Comparison of functionality space among proposed and previous works.

Chapter 6

Pseudo-Random Number Generation (PRNG) Using Three Transistor Chaotic Map

****Portions of this chapter were published in:**

[148] Aysha S. Shanta, Md Sakib Hasan, Md Badruddoja Majumder, and Garrett S. Rose. "Design of a Lightweight Reconfigurable PRNG Using Three Transistor Chaotic Map." *In 2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 586-589. IEEE, 2019.

6.1 Introduction

Intel incorporated a random number generator (RNG) in their motherboards, desktop systems and chipsets. The RNG was introduced in 1999 and it was the first of Intel's family of primitives which was used for hardware security [79]. In order to define randomness, it is important to get a grasp on the idea of unpredictability. The simplest idea of randomness states that the sequence is non-deterministic, uniformly distributed and has independent bits over an infinite length of data set. In statistical definition, it means there is no correlation between the numbers. Random numbers find its applications in many areas such as games of

chance, simulations, secure communication systems, diagnostics and sampling. The security of cryptographic systems relies on the true randomness of the numbers used. Random numbers are intrinsic to some algorithms such as Digital Signature Algorithm (DSA) or protocols such as zero knowledge. In smart card based applications, good quality random numbers might be required for protection against side-channel attacks [134]. Cryptographic key exchange mechanisms and integrity mechanisms require good quality random numbers [20]. Random numbers can also protect against traceability attacks on RFID systems [98].

Random number generators are classified into two classes:

1. True random number generator (TRNG)
2. Pseudo-random number generator (PRNG)

6.2 True Random Number Generator (TRNG)

TRNGs generate random numbers by leveraging the real physical systems that exhibit unpredictable and uncontrollable behavior such as quantum effects, timing user process, electronics noise, shot noise, thermal noise and radioactive decay. TRNGs have good statistical properties for use in cryptographic systems and prevents attack from adversaries.

TRNGs were first developed using continuous-time chaotic systems and now they are designed using discrete-time systems [170, 119]. However, chaotic map based TRNGs are vulnerable to process variations that are inherent in the IC fabrication. These non-ideal effects can severely degrade the performance of the chaotic oscillator rendering it useless. The trajectories of the chaotic oscillator can diverge after a certain number of iterations or may converge to stable points in the orbit. Research has shown that the most compact hardware implementation and sufficient randomness can be obtained from piece-wise linear 1D maps. Researchers have developed ways to make the design robust and proposed ideas to mitigate unwanted effects due to process variation by improving design techniques [179, 111]. True random bit generators can also be implemented with continuous-time mathematical models but the challenge is the bit rate is slow. In this regard, discrete-time models are

preferred due to their inherent rich and complex behavior. Discrete-time chaotic maps can be easily implemented in hardware using field-programmable gate arrays (FPGAs) [167].

The disadvantage of using TRNGs is that they are expensive, slow and susceptible to external synchronization. They require complex designs and hardware is essential for random number extraction. Moreover, the sequence from the TRNG is uncontrollable since it is hugely vulnerable to environmental noise. Hence, the bitstream is not applicable for applications where strong stable randomness quality is desired. These challenges motivated researchers to develop and design pseudo-random number generators.

6.3 Pseudo-Random Number Generator (PRNG)

PRNGs require an initial seed value to start the random number generation. PRNGs are deterministic state transitions $f : x \rightarrow x$ mapping between values in a finite state space if a new seed value is not applied. The seed is usually selected randomly. The generated sequence is called "pseudo-random" since the same sequence can be generated from the PRNG if the same initial conditions are applied. The generated sequence of a PRNG is periodic which means the sequence will repeat itself after a certain number of cycles. There are many advantages of PRNGs, such as, easy and fast implementation, cheap and it does not require any hardware. The numbers produced by the PRNG can be predicted if the underlying function is not complex or if the seed is known [38]. Shannon's entropy states that the entropy of the output depends on the entropy of the seed [56]. In many cases, the seed of the PRNG is generated using a TRNG.

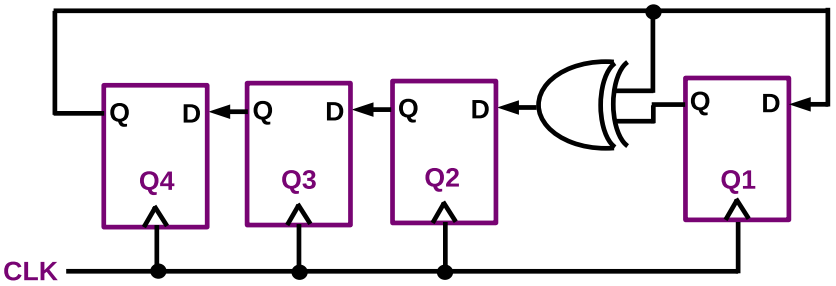


Figure 6.1: 4-bit linear feedback shift register [151].

PRNGs can be used for key generation, initialization vectors for encryption algorithm and produce random sequences for cryptographic challenge-response pairs [74]. Recently, PRNGs are implemented on FPGAs but the disadvantage is that the generated sequence need to pass all the NIST tests by utilizing few resources and at the same time achieve a desirable throughput of random bits.

6.3.1 Linear PRNG

Linear feedback shift registers (LFSRs) have been used for producing pseudo-random sequences for applications in stream cipher and are suitable for high speed and low power specifications [151]. The schematic of LFSR is shown in Fig. 6.1. The design of LFSR is based on shift registers whose input is a linear function of the previous state and input bit is generated by XORing some bits of the overall value of the shift register. The operation of the registers are deterministic so the sequence generated by the register is dependent on the current or previous state. After an initial value is applied to the LFSR, it can be clocked to produce the bitstream. LFSRs are less expensive to implement, have low complexity, high speed and good statistical properties [66]. However, the LFSR is easily breakable by attackers using Berlekamp-Massey algorithm [19].

6.3.2 Nonlinear PRNG

Literature review brought into light that metastable [172], jitter [121] and chaotic systems [50, 18, 180] can be used as entropy sources in the design of TRNG. Chaotic systems are preferred for random number generation due to characteristics such as unpredictability, random behavior, vulnerability to initial conditions and tuning parameters. Shanon states that chaotic systems are relatable to cryptographic systems because they provide confusion and diffusion properties of encryption algorithms [118].

Nonlinear PRNGs can be implemented using nonlinear recurrence equations. Nonlinear feedback shift registers (NLFSRs) have long period and high throughput [53]. Blum's nonlinear generator uses modulus operation and square function in order to increase unpredictability [24]. Other methods include digitizing a chaotic map which suffers from

truncation error and high hardware complexity [15, 58]. There exists a trade-off between randomness and hardware cost which depends on the desired precision in operation. If the precision is reduced to increase the throughput of the system, the truncation would harm the entropy of the sequence which is not desirable for security applications.

An important feature that PRNGs should possess is long cycle length, i.e. the number of outputs after which the sequence starts to repeat itself. If the generated sequence has a short cycle then different portions of the data might be encoded with the same key [107]. In order to alleviate this problem, researchers have proposed different techniques such as reseeding method which masks the last 5 least significant bits at a specified time and tries to induce noise sensitivity [90]. In [92], the period is made long by coupling linear generators with nonlinear generators. Yang *et al.* introduced noise such as Gaussian and uniform distribution to make the sequence unpredictable [181]. In [4], the period of the sequence has been increased by utilizing nonlinear truncation.

6.4 Proposed Lightweight and Reconfigurable PRNG

Chaotic systems can be extremely useful for encryption and secure communication purposes since it requires two properties: good statistical feature and sensitivity to initial conditions. In encryption algorithms, the chaotic system is the main component that generates the pseudo-random sequences. The American mathematician, Shannon who is known as the founder of information theory stated that if a random sequence is used to generate a secret key then a slight change in key should result in considerable change in the output value. Nonlinear dynamical systems possess this exact quality and is thus ideal for encryption purposes [147]. In [75], researchers have used the output of one chaotic map to control the parameters of another chaotic map. This method improves sensitivity to initial conditions, randomness, produces wider chaotic region and possesses more complex chaotic phenomenon.

It is important that the right kind of chaotic system is chosen for pseudo-random number generation such as piecewise linear chaotic map [158, 159], tent map [5] and Henon map [162]. Common chaotic maps can be used for PRNG design such as logistic map [126, 83]. Researchers have also used oscillators with frequency dependent negative resistances (FDNR)

element to achieve a simple and fast RNG. In [145], researchers have designed a hyperchaotic system to design PRNG. A method to generate binary sequence from real number sequence using logistic map has been proposed by Biance [21]. In order to achieve high speed, the chaotic map is iterated to generate the pseudo-random sequence. Chaos-based PRNGs are generated by sampling the trajectory of the map.

There are disadvantages to the chaos-based PRNG as well. Chaotic systems are based on dynamical systems which require s set of real numbers as input and output. The system designed from chaos can require considerable storage space, they are usually slow and it is possible that the chaotic properties are disturbed during the computation procedure [60]. Theoretically, chaotic states in very close proximity to each other should never overlap with each other, but in practice due to finite precision of the computing systems, it is possible they might coincide with each other [125]. The finite precision issue also creates a problem when the rounding method at the sender and receiver side are different. In cryptographic applications, it is important that the sender and receiver side have the same set of data. In order to avoid this problem, the generators can be integrated with fixed finite precision of N bits. For example, in [181] the precision of the system has been chosen to be 43 bits. However, this fix can lead to problems of its own such as short cycles and degradation of chaotic behavior [39].

6.4.1 Chaotic Oscillator for PRNG Design

In this work, two one-dimensional chaotic maps have been connected in cascade to design the chaotic oscillator. 1-D systems are mathematical systems which evolve in time as iteration number increases. The analog sequence generated from the chaotic oscillator is sensitive to small changes in the initial condition which is an essential characteristic of a good PRNG. A conventional approach of generating random sequences is using two discrete-time maps and comparing the obtained sequences. Common chaotic maps such as logistic map are able to produce only 1 bit per iteration but in this design since two chaotic maps are utilized, it is possible to extract 2 bits per iteration.

The chaotic oscillator is shown in Fig. 6.2. It can be seen that two map circuits are used, one in the forward path and the other in the feedback path. Non-overlapping clocks (clk_a

and $nclk_a$) are used to sample-and-hold the outputs of the chaotic oscillator. The initial input, also known as the seed, is applied when clk_{init} is high. The first sampled output is out_1. Out_1 reaches the input of the map circuit in the feedback path when clk_a is high. The feedback map circuit produces an output called out_2. Out_2 is the second sampled output from the circuit and when $nclk_a$ becomes high out_2 becomes the input of the map circuit in the forward path. This cycle continues until the required amount of data is generated.

There are multiple tunable voltages in the proposed system such as input seed, V_{in} and two bias voltages V_{c1} and V_{c2}. The bias voltages can be tied together or different voltages can be applied in order to generate new sequences. The proposed system is reconfigurable because slight change in bias voltage produces a brand new sequence. In order to ensure that the design produces pseudo-random sequences, the bias voltage should be chosen from the chaotic region. The chaotic region can be estimated from the bifurcation diagram and Lyapunov exponent from Fig. 3.8 and Fig. 3.9 respectively. In this work, a single bias voltage, V_c is used where the Lyapunov exponent has the most positive value i.e. 592.5 mV.

The proposed pseudo-random number generator is shown in Fig. 6.3. The sequence generated from a single chaotic oscillator is not random enough to pass all the NIST tests. In order to generate good pseudo-random sequence, two chaotic oscillators, CO_A and CO_B are used. CO_A produces 2 million analog values and CO_B produces 1 million values. In Fig. 6.3, clk_1 has double the frequency of clk_2 so that Sel pin of the analog MUX (same frequency as clk_1) can choose the output of the chaotic oscillator to transfer to the input of the analog-to-digital converter (ADC). The proposed scheme uses a single ADC in order to reduce area and power overhead.

Every other analog value is sampled from oscillator, CO_A starting at the 2^{nd} position until 1 million data is sampled. The first 1 million values are sampled from CO_B. The ADC converts the analog voltages from the oscillators into digital values and stores only the 10^{th} bit in the 2-bit shift register (SR). The 10^{th} bit from the ADC is used to create the random sequence because it provides the highest entropy. When two values are available in the shift register, the values are XORed to enhance the randomness of the sequence and the final sequence from the *output* pin is saved.

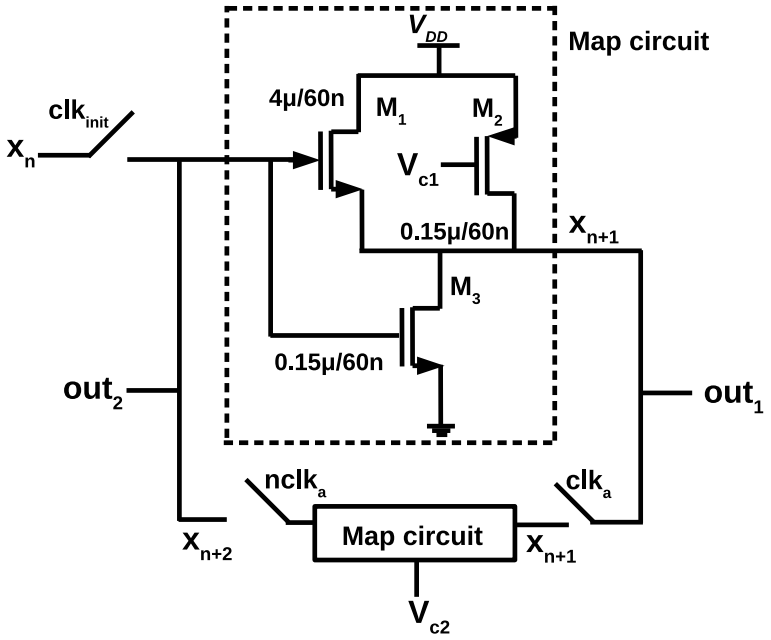


Figure 6.2: Chaotic oscillator. The dotted lines represent the chaotic map [148].

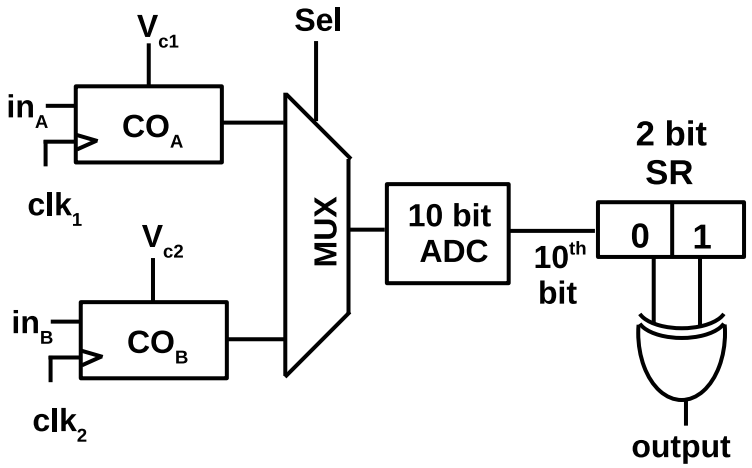


Figure 6.3: Proposed pseudo-random number generator [148].

An analog seed is applied to the proposed PRNG where $in_A = in_B$. The seed is chosen from the output span of the oscillators. In this case, the range is from 310 mV to 1.194 V. The same bias voltage has been used for both chaotic oscillators, i.e. $V_{c1} = V_{c2} = 592.5$ mV. The value of V_c has been chosen using the help of bifurcation diagram and Lyapunov exponent. After the sequences are generated, correlation and NIST tests are performed on the sequences to validate their quality and usage in security applications.

6.4.2 Correlation Coefficient

Correlation coefficient measures the statistical relationship and dependence between two or more random sets of data [176]. If two data sets, x and y are considered, the equations for the correlation coefficient is as follows:

$$cov(x, y) = E\{(x - E(x))(y - E(y))\} \tag{6.1}$$

$$r_{xy} = \frac{cov(x, y)}{\sqrt{D(x)}\sqrt{(D(y)}} \tag{6.2}$$

$$E(x) = \frac{1}{M}\sum_{i=1}^{M} x_i \tag{6.3}$$

$$D(x) = \frac{1}{M}\sum_{i=1}^{M}(x_i - E(x))^2 \tag{6.4}$$

In order to test the correlation coefficient of the data generated from the proposed PRNG, 10 sequences of length 10100 were generated. The first 100 data has been ignored to avoid transient response. 10 sequences can be combined in 45 ways if sets of 2 are formed. The seed has been changed to generate a new sequence. The seed values start from 350 mV to 800 mV with increments of 50 mV. Ideally, the correlation coefficient of two random data sets will be zero. Fig. 6.4 shows the distribution of correlation coefficient of the generated sequences. It can be seen from the figure that all the coefficients are either 0 or close to 0. It indicates that the PRNG has good correlation features.

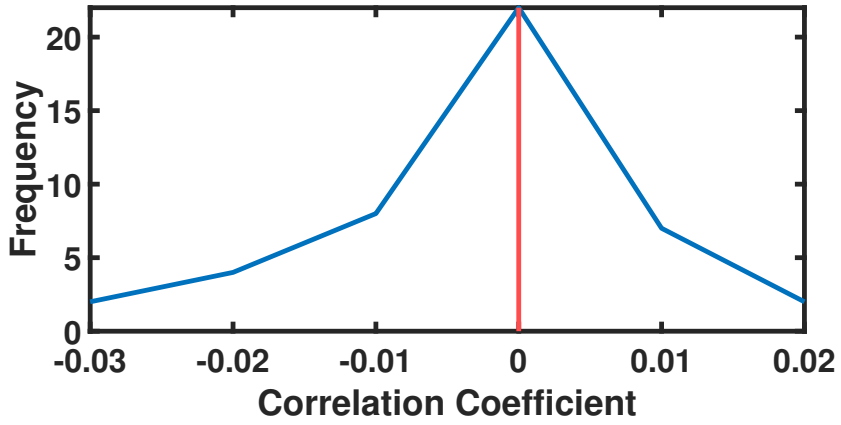


Figure 6.4: Distribution of correlation coefficient.

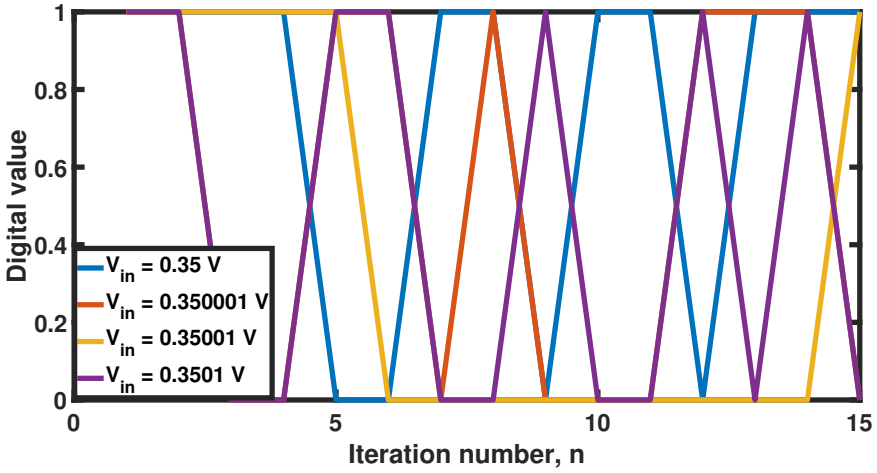


Figure 6.5: Seed sensitivity of the PRNG for slight changes in the seed.

6.4.3 Seed Sensitivity

Seed sensitivity is essential in the design of PRNG because a tiny difference in the seed should cause the output to change significantly. In order to test the sensitivity, four different seeds are applied to the PRNG with very close values, 0.35 V, 0.350001 V, 0.35001 V and 0.3501 V. Fig. 6.5 shows the seed sensitivity of the proposed PRNG for the first 15 iterations. It can be seen that the sequences start deviating within the first few iterations.

The correlation coefficient has also been calculated comparing the slightly changed seeds with the original seed. 10000 iterations have been considered in order to calculate the correlation coefficient. Table 6.1 shows the correlation coefficient of the three cases and as expected all the values are close to zero. Thus, the proposed PRNG has high seed sensitivity.

In some cases, the sequence generated from the map does not possess enough randomness and unpredictability to be used in cryptographic applications. Post-processing of data might be required such as bit counting redundancy reduction technique [36].

6.4.4 National Institute of Standards and Technology Tests

National Institute of Standards and Technology (NIST) tests measure the contingency of the PRNG sequence obtained from the generator. Each test in the suite measures the statistics of the features that the generated sequences possess. High value of statistics means an absence of contingency in the sequence. The test result is positive when the calculated probability of the obtained statistics is compared with the statistical significance value, α which is usually equal to 0.01. If the value of the probability is greater than 0.01, then the sequence passes the test. The sequence that has at least one negative test result is not random. If the p-value of a sequence is 1, then the sequence is perfectly random and p-value of 0 indicates that the sequence is not random. There are 15 tests in the NIST 800.22 test suite to test the randomness of sequence for applications in cryptography. The minimum pass rate for each test is in the tolerance range calculated by equation 6.5.

$$Tolerance = (1-\alpha) \pm 3\sqrt{\alpha(1-\alpha)/T}. \tag{6.5}$$

where T = number of sequences to be tested.

Table 6.1: Correlation coefficient between the original sequence and the three cases.

	Case 1	**Case 2**	**Case 3**
Correlation coefficient	0.0043	0.0040	0.0169

The sequence of values that do not pass the tests are not suitable for security applications since the lack of certain statistical properties might help the attacker to predict future values of the sequence. However, algorithms that do pass the tests does not necessarily provide enhanced security features for use in cryptographic algorithms. NIST tests help in designing new PRNGs, identifying weak binary sequences, studying standard PRNGs, verifying that the PRNG has been implemented properly and analyzing the degree of unpredictability of the current PRNGs.

All the tests in the NIST suite [1] are described briefly.

1. **Frequency Monobit Test:** This test measures whether the proportion of ones and zeroes is the same in the generated sequence. The test calculates if the number of ones is close to 0.5.

2. **Frequency Test within a Block:** This test measures whether the frequency of ones in an M-bit block is $M/2$. The default value of M is 128.

3. **Cumulative Sums Test:** This test calculates the partial sum of the sequence under test and compares it to the expected cumulative sum of a random sequence.

4. **Runs Test:** Runs test determines the total number of runs of identical bits where run means an uninterrupted sequence. A run of length, n means that the sequence has n identical bit bounded by different bits before and after. Basically, this test calculates whether the switching between zeros and ones are too slow or too fast.

5. **Serial Test:** The purpose of the test is to find out all possible overlapping m-bit patterns in the overall bit stream. Random sequences are uniform in manner so every m-bit pattern has the same chance of occurring as every other m-bit pattern.

6. **Rank Test:** Rank test looks for linear dependence among fixed length strings of the sequence under test.

7. **DFT Test:** DFT test can indicate if there are periodic features i.e. repetitive patterns in the sequence.

8. **Longest Run of Ones Test:** This test checks if the longest run of ones of the generated sequence is similar to a random sequence.

9. **Non-overlapping Template Matching Test:** This test finds the number of occurrences of non-periodic patterns and fails the test if the sequence has too many of these patterns.

10. **Overlapping Template Matching Test:** This test fails those sequences that lack the expected number of runs of ones of a given length.

11. **Maurer's Universal Statistical Test:** This test measures whether the sequence can be reduced in size significantly without losing valuable information.

12. **Approximate Entropy Test:** This test compares the number of occurrences of overlapping blocks in two adjacent lengths with the expected result of a random sequence. The default length of each chunk of data is 10 bits.

13. **Random Excursions Test:** This test calculates the number of visits to a particular state in a random walk and checks if it exceeds the expected value.

14. **Random Excursions Variant Test:** This test checks if there are deviations from the expected number of occurrences of different states in the random walk.

15. **Linear Complexity Test:** This test determines if the generated sequence is complex enough to be considered random. The complexity is measured by using Berlekamp-Massey algorithm.

In this work, the generated pseudo-random sequence passes 15 out of 15 NIST tests. 100 1 million sequences have been generated in order to validate the NIST tests. 100 different

sequences have been generated using 100 different initial seeds. Table 6.2 shows the results of the NIST tests. Asterisk(*) means that average of multiple results are shown.

6.5 Overhead Analysis

PRNGs have been designed based on popular chaotic maps such as logistic map, tent map and piecewise linear map. CMOS implementations of these maps can have huge area overhead. In many cases, in order to reduce area consumption only one value of bifurcation parameter is chosen in the chaotic map. The proposed design retains all the control parameters which makes the design reconfigurable and helps to generate new sequences. Different bias voltage combination in the chaotic oscillators can generate more sequences. The body bias voltage of the transistors can also be used as an added control parameter.

The generated pseudo-random sequences need to pass the NIST test so that it can be used in security applications. There are multiple post-processing schemes discussed in the literature but they add more complexity to the design. Post-processing schemes can add to the delay of the system on top of being area and power hungry. This work proposes a simple post-processing technique for the generated data. After the analog data is generated from the two chaotic oscillators, the analog MUX selects which data gets to reach the input of the ADC for digital conversion. The least significant bit (LSB) of the digital data is stored in a two-bit shift register and XORing is carried out on the stored bits. This technique adds very little power and area overhead.

The PRNG has been designed in 65 nm CMOS process with a supply voltage of 1.2 V. The area of the chaotic oscillator is 0.556 μm^2. A low area and power consuming SAR ADC can be used for digital conversion [101]. The ADC only consumes 0.132 mm^2 of area and 1.6 mW of power. The overhead of the entire system including the chaotic oscillators, analog MUX, 10-bit ADC and 2-bit shift register is approximately 0.132 mm^2 and the power consumption is 2.12 mW. Table 6.3 demonstrates that the proposed system consumes less area and power compared to prior work.

The throughput of the chaotic oscillator is 330 MS/s but the throughput of the ADC is only 100 MS/s. The throughput of the entire system is limited by the ADC's throughput.

Table 6.2: NIST test results of the proposed PRNG [148].

NIST test	**P-value**	**Pass-rate**
Frequency	0.0966	0.99
Block frequency (m = 128)	0.3838	0.98
Cumulative sums*	0.4356	0.99
Runs	0.7792	1
Longest runs of ones (M = 10000)	0.8832	1
Rank	0.4190	1
DFT	0.8514	0.99
Non-overlapping template* (m = 9)	0.4994	0.98
Overlapping template (m = 9)	0.0329	1
Universal (L = 7, Q = 1280)	0.6371	0.99
Approximate entropy (m = 10)	0.2368	0.99
Random excursion*	0.2558	0.99
Random excursion variant*	0.2213	0.99
Serial* (m = 16)	0.52155	0.99
Linear complexity (M = 500)	0.3191	0.98

Table 6.3: Overhead comparison of established PRNGs with the proposed design [148].

	Technology (nm)	**Supply Voltage** (V)	**Area** (mm^2)	**Power** (mW)	**Throughput** (MS/s)
[124]	350	3.3	0.752	56	40
[124]	180	1.8/3.3	0.126	22	100
[91]	180	1.8	0.275	13.9	6400
[181]	180	1.8	0.767	37	120
This work	**65**	**1.2**	**0.132**	**2.12**	**100**

If 10 bits from the ADC are used, then the throughput of the system can be increased to 1000 MS/s. A more complex post-processing scheme need to be used in order to pass all the NIST tests. If only the 10^{th} bit of the ADC is utilized, then a Flash ADC can be used to increase throughput but it comes at a cost of more area and power consumption.

6.6 Application Lies in Security of IoT Devices

In 1999, Keven Ashton introduced the term Internet of Things (IoT) when he was working at MIT's AutoID [16]. IoT devices became increasingly popular due to the rapid advancement in technology and standardized communication protocols. These objects can communicate amongst each other and are managed by computers. It has been anticipated that by the year 2020, there will be 20 billion connected devices in the IoT domain. One of the major concerns of IoT devices is the security threat on all the connected devices. The security concerns can range from threat of cyber-theft, financial transactions and personal privacy. IoT devices can communicate with each other without human interaction and absence of human control. Such devices typically have area and power consumption limitations so traditional cryptographic security schemes will be expensive given the constraints. Therefore, the proposed lightweight PRNG will be perfect for security applications in IoT devices due to its lightweight nature and inherent dynamics.

Chapter 7

Physically Unclonable and Reconfigurable Computing System (PURCS)

****Portions of this chapter is to be published in:**

[149] Aysha S. Shanta, Md Badruddoja Majumder, Md Sakib Hasan, and Garrett S. Rose. "Physically unclonable and reconfigurable computing system (PURCS) for hardware security applications ." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, (accepted).

7.1 IC Counterfeiting and Logic Locking

7.1.1 IC Counterfeiting

Counterfeit ICs include remarked, recycled, tampered and overproduced ICs and the estimated annual loss due to these ICs is over $169 billion dollars. The counterfeit ICs sold in the market can have national security implications. Semiconductor Industry Association (SIA) estimated that 15% of the replacement and spare semiconductors purchased by the

Pentagon are counterfeit chips [52]. IC and Intellectual Property (IP) designers are re-evaluating their trust on hardware due to globalization of the supply chain. The hardware is prone to new attacks everyday such as IP piracy and reverse engineering [10].

The are multiple entities involved in the IC design and fabrication process such as the foundry, the designer and the end-user [184]. The designers can use third-party IP blocks in their systems. The foundry is untrusted because it is located outside the company and rouge people in the foundry can reverse engineer the design from the mask, steal IP design or produce extra chips to illegally sell in the market. End-users are also untrusted because they can try to reverse engineer the contents of the chip to gain technology or design information. Reverse engineering can help in obtaining design details by using IC imaging and probing. The process of reverse engineering includes multiple steps such as depackaging the IC, delayering and imaging layers and finally analyzing the bundle of extracted images to find design details [166].

The trend of IC piracy and reverse engineering to produce counterfeit ICs have raised severe concerns in the industry. The counterfeiting can happen in several forms such as:

1. Design houses can sell IPs to third parties without vendor's knowledge.

2. IPs being used without paying the appropriate fees to the IP vendor.

3. Fabrication houses can sell illegally cloned copies of the design without the designer's knowledge in the market.

4. Companies can perform reverse engineering of a fabricated IC to decode the contents of GDS-II files.

5. Adversaries in the supply chain can insert malicious hardware into the design without the designer knowing about it.

All the countries involved in the IC design flow does not have strict laws against IP theft except USA and Japan. Hence, it becomes, IC/IP designer's responsibility to protect their designs. IP/IC piracy affects IP vendors, chip designers and system designers by depriving them of the generated revenue. Several countermeasures have been introduced in order to

mitigate the consequence of counterfeiting, reverse engineering and insertion of malicious content into the design such as IC metering [10], watermarking [82], split manufacturing [76], IC camouflaging [131] and logic locking [127, 132]. Watermarking is a passive technique that can help detect piracy but fails to protect against it. Metering utilizes a set of protocols, either active or passive, which assigns a unique ID for each chip after fabrication. Split manufacturing and IC camouflaging protects only against the untrusted foundry and untrusted user respectively. Logic locking has gained the most attention amongst all the techniques because it can lock the functionality of the IC while it is being passed through different phases of production and the attacks such as overbuilding, IP piracy and reverse engineering can be mitigated.

7.1.2 Background of Logic Locking

One possible method to protect the functionality of the IC is by inserting additional gates to lock the IC such that the circuit will only produce correct outputs when a specific input is applied at these gates. The IC will be locked when it passes through the untrusted design flow in order to conceal its contents. The claim of logic locking concept is that if the key and input space is large, the adversary will have to spend exponential time in figuring out the secret key. Logic locking prevents cloning, Trojan insertion, reverse engineering and overbuilding of ICs. The designer provides the correct key to the end-user so that the circuit produces correct outputs on application of the valid key. The keys can be stored in a tamper proof memory or physical unclonable functions (PUFs) can be used to generate keys so that the attacker cannot gain direct access to the secret key value. If the key values are unavailable to the attacker, the netlist of the design obtained by stealing or reverse engineering is useless.

In an encrypted IC, wrong key should generate wrong output for the entire space of input patterns. In case a wrong key produces correct output, then the logic locking method is not strong and the adversary will gain advantage. If an invalid key only changes a few of the output bits, then the adversary can tolerate the wrong outputs. If a wrong key affects or flips all the output bits, then the wrong output is an inverted version of the correct output. Therefore, an invalid key should alter half of the output bits, which is why the Hamming distance between correct and wrong outputs should be 50%. 50% Hamming distance should

provide sufficient obscurity to the attacker. It is also important that fewer gates need to be inserted in order to reach 50% Hamming distance [128].

7.1.3 Threat Model for Logic Locking

- The end-user and foundry are untrusted.
- The designer is trusted.
- The adversary knows about the logic locking algorithm and the location of the key gates. The only unknown piece of information is the secret key.
- The adversary has access to a reverse engineered netlist and a functional IC. The netlist is obtained from the IC design or by reverse engineering the mask, layout or manufactured IC. The adversary uses simulation tools on the locked netlist and generates the output patterns from the functional IC for specific input patterns.

7.1.4 IC Design Flow

The IC design flow is shown in Fig. 7.1. The designer and IP provider are trusted in logic locking threat model. The foundry is not trusted since there might be rouge agents who can steal or reverse engineer the design. The designer encrypts the design and after synthesizing sends the layout masks to the foundry. The key has not been loaded into the tamper-evident memory at this stage since it can be recovered by the adversary in the foundry. The foundry manufactures the chip and sends it back to the design house. The designer then loads the key into the memory to enable correct functionality of the IC. After the key is loaded, the designer blows out the fuses in the read/write circuit in order to disable access to the memory contents by the user. The designer performs validation tests on the functional IC and if the ICs pass the test, the chips are packaged and sold in the market.

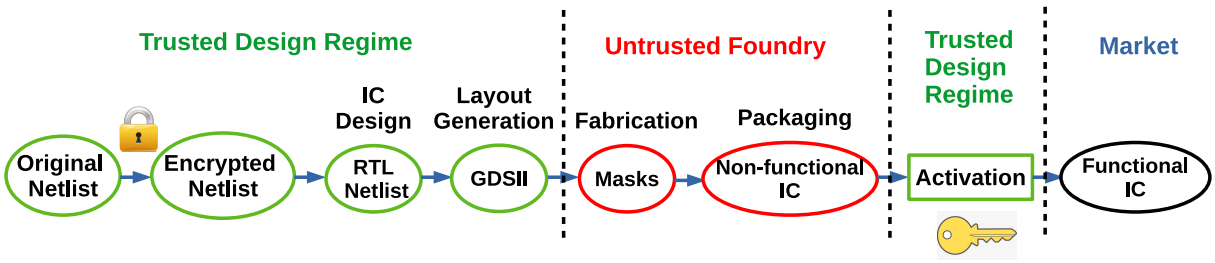


Figure 7.1: IC design flow with logic locking capabilities [132].

7.1.5 Issues Mitigated by Logic Locking

Hardware Trojans

Logic locking prevents the adversary from inserting malicious circuitry into the design because it is harder to find a location to insert Trojans. The key gates change the propagation of the signals in the design in a different manner which is unknown to the attacker.

Overbuilding

The foundry can produce extra chips without the knowledge of the design house. Their intent is to sell the ICs to the market. However, the ICs cannot be unlocked without the secret key and will be useless.

Reverse engineering and IP piracy

If the adversary steals the netlist or gains access to it by reverse engineering, the netlist will be useless without information about the secret key.

7.1.6 Types of Logic Locking

Logic locking can be classified into two types: combinational and sequential.

Sequential Logic Locking

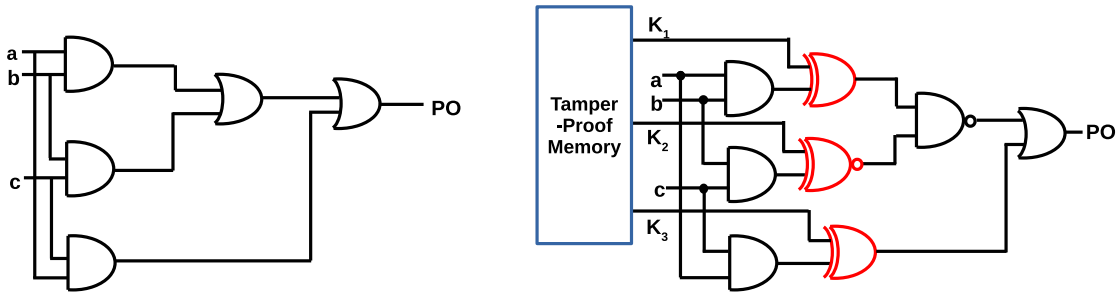
Sequential logic locking requires additional black states to be inserted in the state transition graphs [10, 28]. The modified state transition graph is designed in such a way that a valid state is reached only on application of valid key sequence. If the key is withdrawn from the

design, the system transitions into a black state again. Examples of sequential primitives are obfuscated finite state machines [28]. Another sequential logic locking approach is to replace parts of the design using programmable logic such as look-up tables (LUTs) [17]. In this way, the IP owner hides the design from rouge agents in the fabrication stages. The concealed design is later configured into programmable logic and the circuit will only function correctly if these replaced elements are configured properly.

Combinational Logic Locking

In combinational logic locking, XOR/XNOR gates, AND/OR gates and multiplexers are inserted into the design to hide the functionality of the design. One of the inputs of the key gates act as the 'key' input. The user can configure these key gates as buffers or inverters depending on the value of the key inputs. An example of unlocked and locked circuit is shown in Fig. 7.2. The correct key input for the circuit to be functioning properly is $1, 0, 0$ for $K1$, $K2$ and $K3$ respectively which is loaded by the user in the chip's memory. The locked IC can be activated either prior or post-fabrication test. If the keys are loaded remotely into the chip, then secure communication infrastructure is necessary.

Memory elements can also be used for logic locking in addition to combinational and sequential elements [17]. The circuit will produce correct outputs only when the memory element is programmed correctly. Although, usage of memory elements will lead to significant performance overhead.



(a) (b)

Figure 7.2: (a) Unlocked circuit (b) Locked circuit with three XOR/XNOR gates [132].

7.1.7 Techniques of Inserting Key Gates

There are several techniques to insert key gates into the IC design such as random (RLL) [140], strong interference based logic locking (SLL) [183] and fault analysis based (FLL) [132]. In order to insert key gates, the position of the node inside the design needs to be determined. When an incorrect key is applied, it should be able to alter most of the output bits for most of the input patterns. Key gates can be inserted randomly but it does not necessarily produce wrong output for the wrong key since the effect of wrong key might be damped in the propagation path to the primary outputs. Fault analysis based logic locking assumes that if an incorrect key is applied it activates a fault in one of the nodes. When a wrong key is applied, a stuck-at-1 (s-a-1) or a stuck-at-0 (s-a-0) fault will be activated when XOR/XNOR gates are used for logic locking. Sometimes, a wrong key can result in generation of the correct output. Therefore, it is important that the effect of an excited fault is propagated to primary outputs by using non-controlling inputs at the key gates. All the inputs cannot guarantee the presence of non-controlling values on the propagation path, so an incorrect key will not always corrupt the output. Care should also be taken when multiple key gates are inserted or replaced in the design since the effect of one gate can nullify the effect of the other gates.

7.1.8 Types of Key Gates

XOR/XNOR based

XOR/XNOR gates can be inserted in a design to hide the functionality of the circuit. The gates are inserted into the design after logic synthesis and before physical synthesis. One of the inputs of the key gate is part of the 'secret key' and the other one is part of the original circuit design. If the key gate is an XOR gate, then the key value is 1 and if it is an XNOR gate then the value is 0 for the circuit to perform correctly. Keys are stored in tamper-evident memory by the trusted party and the circuit will not function correctly unless a valid key is loaded in the on-chip memory.

If the key gates are left in the design without further improvements, then the attackers can extract the key value by finding the XOR and XNOR gates in the design by reverse-engineering the layout. The effort of the attackers can be increased by replacing the XOR/XNOR gates with other gates such as AND, OR and NAND. Another method could be to change the position of the inverters so that the polarity of the key gates are altered [183]. However, the latter method suffers from significant area and power overhead in order to realize De Morgan's rules.

MUX based

Multiplexers (MUXs) are inserted in the logic paths of a system in MUX based logic locking [132, 127, 177]. One input to the MUX is the original logic signal, the other input is an arbitrary internal node and the select pin of the MUX is the key input. The original signal can be connected to either the first or the second input of the MUX. This offers the flexibility that the key bit can be 0 or 1. If the applied key is not correct, the MUX selects a random node for determining the primary output. However, the output might not be corrupted since the original node might have the same value as the random node. Designers should also consider that a combinational loop is not formed between the original node and the random node. MUX based logic locking requires more gates to be inserted to reach 50% Hamming distance.

In [175], camouflage connectors and multiplexers are connected to design a 'configurable logic unit' which can be used to replace logic gates in the design. An m-input 1-output 'configurable logic unit' can perform 2^{2^m} functions using 2×2^m camouflage connectors and 2^m-to-1 MUXs. Hence, a 4-to-1 MUX can be tuned to produce all the possible 16 Boolean functions that a 2-input 1-output gate can perform. It is better than configurable CMOS gates which are able to perform only 3 functionalities: NAND, NOR and XOR [131].

LUT based

An n-input look-up table requires 2^n memory cells and 2^n-1 2-to-1 MUXs. In [17, 96], researchers have proposed using look-up table (LUT) based logic locking and established methods for replacing gates in the netlist. The scheme is secure because the contents of

the LUTs are unknown after reverse engineering. In LUT based obfuscation, the selected standard logic gates are replaced by LUTs. The functionality of the gates remain concealed from the manufacturer. LUTs are then configured in a trusted facility after the fabrication process is over.

After fabrication, the foundry can perform tests on the circuit on any mode (any logic out of 16 possible functions) of the reconfigurable logic. The end-user will determine the final usage or logic implementation of the reconfigurable block. Personnel in the designing and manufacturing phase does not have to know the desired function of the LUT and cannot tamper with the reconfigurable blocks. The reconfigurable block will act as a black-box to the supply chain attacker. He knows the function of the LUT but does not have knowledge about the final implementation. Therefore, he cannot perform tests, probing, and reverse engineering because the LUT content has not been finalized [95]. Reconfigurable logic block possesses three important qualities:

1. virtual "black-box"

2. polynomial slowdown

3. preserves the functionality

LUTs are better as key gates compared to XOR/XNOR gates for several reasons. First, when XOR gates are used as key gates, they violate the outputs for wrong key application but they are just gates added to the design. The entire layout is sent to the foundry for fabrication so the rouge agent in the foundry can reverse engineer the layout to learn the entire contents of the design and learn the location of the XOR gates. The XOR gates can simply be removed from the design or attackers can use a FIBing technique where additional wires are added to bypass the locking scheme. LUTs lock the design and replace gates in the design with generic memory structure. The contents of the memory are unknown to the manufacturer. Second, each XOR gate addition to the circuit adds 1-bit key to the design whereas LUTs provide a larger key space which increases the attacker's work.

There are disadvantages to using LUTs. The use of RAMs will increase the performance overhead since additional mask layers are required. Utilizing LUTs for obfuscation purposes

can result in huge area and delay overhead. The area overhead of the LUTs exponentially increases with the increase in input size. The area overhead constraint limits the number of gates that can be replaced with LUTs. Moreover, the timing or delay requirements limit the placement of LUTs in critical paths.

7.2 Authentication and Physical Unclonable Functions (PUFs)

7.2.1 Authentication of Devices

Nowadays, the emergence of smart devices along with embedded memories is affecting our daily lives. It has opened opportunities for great applications such as smart city and home automation. The security of the devices is a huge challenge and opened a field for security researchers to dive into. An illegitimate or unauthenticated device will be able to share false information to disturb the functions of cyber-physical systems. If the security of these devices are not handled or taken care of, then these devices can potentially distribute confidential information to untrusted parties. Since these systems are mobile, the threat model should include special cases where the devices will be operating in untrusted environment and the attacker has physical access to the device. As a result, authentication is important for data protection and to prevent man-in-the-middle attacks.

The solution is to install a secret key unique to each device in order to secure the communication. Previously, the norm was to save the secret key in a non-volatile memory such as electrically erasable programmable read-only memory (EEPROM) or battery-backed static random access memory (SRAM) and perform cryptographic algorithms such as encryption or digital signature to authenticate a device to ensure protection of confidential information. New techniques were necessary to be developed because of shortcomings in the previous methods such as huge amount of area and power consumption. Moreover, battery-backed RAMs can be read after storing keys for a long time and non-volatile memories are prone to invasive attacks since the keys are stored in digital format [13, 14, 26]. In order

to secure the key, expensive tamper-proof circuits need to be installed which needs to be battery powered continuously.

7.2.2 Types of Authentication Architectures

Two-types of authentication architectures exist and these can be classified as: hardware and software-aided architectures.

Software-aided Architectures

Software security methods are more flexible and easier to implement, update and manage but they are built on the assumption that the underlying hardware is secure which is not guaranteed in most cases. The software architectures utilize the conventional cryptographic algorithms but the strength of the algorithm can be adjusted to fit the operating requirements.

The disadvantage of software based architecture is that they require huge amount of computation and consumes a lot of space in the memory. Moreover, they are susceptible to hardware cloning attacks. Software implementations require less resources to reverse-engineer and the source codes stored in the memory of the system can be easily accessed and tampered by attackers. When the codes with timing and power traces are compiled into assembly language instructions, they become susceptible to power and delay analysis based side channel attacks. The aim of the researchers in sensor networks is to build a lightweight and feasible but computationally less intensive cryptographic algorithms. It is also mandatory to ensure that key distribution overhead is within a tolerable limit and solutions have been proposed by reducing the key size [94, 120]. From a security perspective, the dilemma between computation overhead and security level will always persist.

Hardware-aided Architectures

Hardware based cybersecurity utilizes secure hardware to protect the system while managing to keep the performance overhead minimum. The problem arises because the security of the system relies on the assumption that a "safe" key exists. The secret keys are stored in the

non-volatile memories in order to use in cryptographic primitives. Cryptographic keys are long digital bit strings of data with truly random and entropy features. The vulnerabilities arise because of insecure storage and transmission of personal identifiable information and the lack of strong encryption and authentication schemes. A system that does not require the key to be stored in a memory and generates a signature due to the physical disorder of the system and provides tamper-resistance as an innate property is required for the security of systems. This is why hardware-based architectures utilize Physical Unlconable Functions.

7.2.3 Process variation

The inevitable random fluctuations in silicon fabrication process results in random deviations in interconnect and device properties. This random fluctuations lead to deviation of circuit parameters from nominal values. The process variation can be divided into types: *die-to-die* and *within-die* [186]. The radius of *die-to-die* variation is larger than the die size and includes wafer-to-wafer, within-wafer, lot-to-lot and fab-to-fab. All the circuits in the die are affected equally. The variation that exists between circuits of the same die is known as *within-die* variation. The variations can occur due to reasons such as, manufacturing instrument or layout technique of the designer. Random variations can also occur due to changes in device length, doping concentration variation and fluctuations in oxide thickness.

Let the varied process parameter be denoted by X and this can represent the threshold voltage or the channel length. The sources of variation can be represented by an equation:

$$X = X_0 + \Delta X = X_0 + X_g + X_s + X_r. \tag{7.1}$$

where X_0 represents the mean value of X, X_g represents the global variation, X_s represents the intra-chip variation and X_r represents random fluctuations.

7.2.4 Background of Physical Unlconable Functions (PUFs)

PUFs use process variation in the fabrication procedure to generate unique signature for each chip [161]. The input to a PUF is called a "*challenge*" and the output is called a "*response*". The PUF can be modeled if all the challenge-response pairs (CRPs) are known.

The *response* of the PUF is dependent on its physical structure and is difficult to clone. Even the original manufacturer with the same photolithography masks cannot create an identical device because of the unpredictable and uncontrollable nature of process variation. A PUF circuit is inherently noisy due to the time dependent variations of the physical characteristics of the system. While using PUFs, the noise is eliminated by applying the same *challenge* multiple times and using only those *response* bits that remained stable.

Modern cryptographic implementations in hardware is facing increasing number of attacks [173]. Side-channel attacks (SCAs) are able to retrieve secret information from memories where the cryptographic keys are stored [157, 165]. There is continuous challenge to keep the memory safe from adversaries. There is an increasing demand for cryptography in resource-constrained devices such as IoT, implantable medical devices and wearables. Researchers are putting in more effort into building PUFs for developing secure electronics because PUFs provide keys without the need of a storage element (memory) for secret keys.

One of the earliest works related to PUFs was proposed by Lofstrom *et al.* in the year 2000 where mismatch in silicon devices were used for identification of ICs [99]. The process variation in PUFs can lead to intrinsic and random alterations in the characteristics of CMOS circuits such as changes in threshold voltage of the MOSFETs, metal resistivity and effective channel length of the transistors. The PUF outputs are not only dependent on the applied inputs but also on the internal characteristics of the circuit. The secret key generated by PUF is only available for usage when the chip is turned on and running. Since the IC probes into the random variations caused by the manufacturing process, the secret is difficult to clone [161]. The adversary has to put in more work since they have to mount an attack while the IC is running, making sure that they don't disturb the internal characteristics of the chip, otherwise, the secret key will be changed or lost. Tapping into the device might cause a capacitance change which in turn causes the output of the PUF to change. Another advantage of PUF is that extra steps in the fabrication process, or testing or programming is not required. PUF technology uses simple digital circuits which are easy to manufacture and consume less area compared to EEPROMs or RAMs that require anti-tamper circuits to protect the hardware. The fabrication cost of EEPROM is expensive since it needs additional mask layers and RAMs require an always-on power source.

In addition to studying silicon PUFs which rely on delay and timing information, researchers have also dived into PUFs that exploit physical characteristics of devices to generate unique *responses*. For example, acoustic PUFs measure the acoustic reflections of an item and coating PUFs measure the capacitance of the coating layer that covers the chip [156, 169]. Pappu *et al.* developed an optical PUF that uses speckle patterns of medium for laser light [122].

As technology is scaling, it is becoming increasingly difficult to control power and performance dependent parameters. Variability is now an unavoidable characteristic of CMOS processes [25]. The unique *responses* generated by PUFs finds its applications in device authentication, encrypted storage, active metering of ICs to prevent overproduction, trusted configuration of FPGAs and secure software execution on processors [130]. Guajardo *et al.* used the PUF for remote service authentication, protection of intellectual property (IP) and secret key storage [59].

7.2.5 Classification of PUFs

PUFs can be classified into two types: weak PUF and strong PUF [68]. The weak PUF has limited number of CRPs and portrays a linear relationship with the number of components whose behavior depends on process variation whereas strong PUFs have a huge number of CRPs and displays an exponential relationship with the components that exhibit PUF characteristics. PUFs provide a hardware aided architecture for authentication of devices which makes the computation efficient and reduces the usage of memory. Weak PUFs are usually used for key storage and strong PUFs are used for authentication purposes. A disadvantage is that the PUF characteristics are susceptible to external environmental conditions such as temperature and supply voltage variation.

Weak PUF

Weak PUFs, also known as Physically Obfuscated Keys (POKs), can directly digitize the unique fingerprint of the circuit and then the digital value is used for cryptographic applications. The PUF will replace the secure non-volatile memory that would contain the

secret key. Once the key is extracted from the weak PUF, it is stored in a volatile memory during operation. The key must be kept secret since weak PUF can generate only a small number of *responses*. The *responses* are unpredictable and are dependent on the process variation. If the attacker gets access to the key, then the attacker can use it to mimic the operation of the PUF. If the physical size of the PUF is finite then all the CRPs can be measured within polynomial time. As a result, the peripheral parts of the PUF are designed with a fuzzy extractor (FE) which limits direct access to the original PUF response [29].

In [89], Layman *et al.* proposed a weak PUF called Static Random Access Memory (SRAM PUF) which is composed of symmetric cross-coupled inverters. Once the power of the SRAM PUF is turned on, the cells in memory will randomly be pushed towards a logical 1 or 0 due to process variation in the manufacturing procedure. The *response* space is the number of cells in the SRAM and the *challenge* is turning on the SRAM. SRAM PUF generates only one CRP which can be used for key generation rather than device authentication. Other examples of weak PUF include latch PUF [61], butterfly PUF [88], D flipflop PUF [171] and bus keeper PUF [153].

Weak PUFs are ideal for key generation purposes because ideally they have fixed "challenge bits". The PUF provides secure key storage facility and the secret key is never revealed during usage. The key generated by the PUF usually requires post-processing through error-correction codes (ECC) for cryptographic applications. ECC is needed because the digital bits of the PUF output might flip due to noise and other environmental factors. Once a stable set of output bits are obtained from the PUF, they can be used for many cryptographic applications. Weak PUFs can also be used for authentication purposes even though they are unable to generate a huge pool of CRPs. The PUF can be used in conjunction with HMAC/AES implementation to achieve authentication at the cost of additional hardware required for the cryptographic protocol which in turn makes this technique area and power hungry.

Strong PUF

Strong PUFs differ from weak PUFs because strong PUFs have a large pool of CRPs and devices can be authenticated directly without the need of cryptographic hardware. The

benefit is that the adversary cannot cover the entire *challenge-response* space in order to model the PUF and mimic the outputs. The output of the PUF does not reveal information about the internal characteristics or behavior of the circuit. The output of the strong PUF does not need to be kept secret as the weak PUF. The strong PUF should have readout access and readout time limitations so that the attacker cannot enumerate through the entire *challenge-response* space in polynomial time. Since a strong PUF does not need cryptographic hardware to provide authentication, the system should be able to prevent unauthorized access to the internal circuitry.

Gassend *et al.* proposed a strong PUF called Arbiter PUF which utilizes the delay due to process variation between identical paths to generate *responses*. The schematic of Arbiter PUF is shown in Fig. 7.3. Arbiter PUF uses multiplexers to switch between the paths [54]. Arbiter PUF is designed in the layout level to eliminate bias in the delay paths. The select signal of the multiplexers is the *challenge* to the PUF. The delay paths are fed into an arbiter which can be a D flipflop or SR latch. When the input of the PUF changes, the signals propagate through two different delay paths. The arrival times of the two input signals will vary due to process variation. The *response* is one if the signal arrives at input D of the flipflop first, otherwise the signal is zero. This interpretation holds true as long the setup time of the latch is negligible compared to the delay difference. This type of PUF is known as a strong PUF because it can generate many CRPs and can be used for authentication purposes. Arbiter PUF is appropriate for applications in resource-constrained platforms such as RFIDs. Ring oscillator (RO) PUF is another example of a strong PUF which is easier for implementation in ASIC and FPGAs compared to Arbiter PUFs. However, RO PUFs are slower, more area consuming and power hungry. RO PUF are applicable for secure processor designs [144]. Other examples of strong PUFs are optical PUF [122], lightweight secure PUF [106] and XOR arbiter PUF [71].

Although weak PUFs are typically used to generate secure keys due to their small number of CRPs. Strong PUFs can also be used to generate keys but the challenge is correcting the bit errors in the *responses* obtained from the PUF. Pappu *et al.* proposed a "pattern matching" technique to correct the output errors of the PUFs [123].

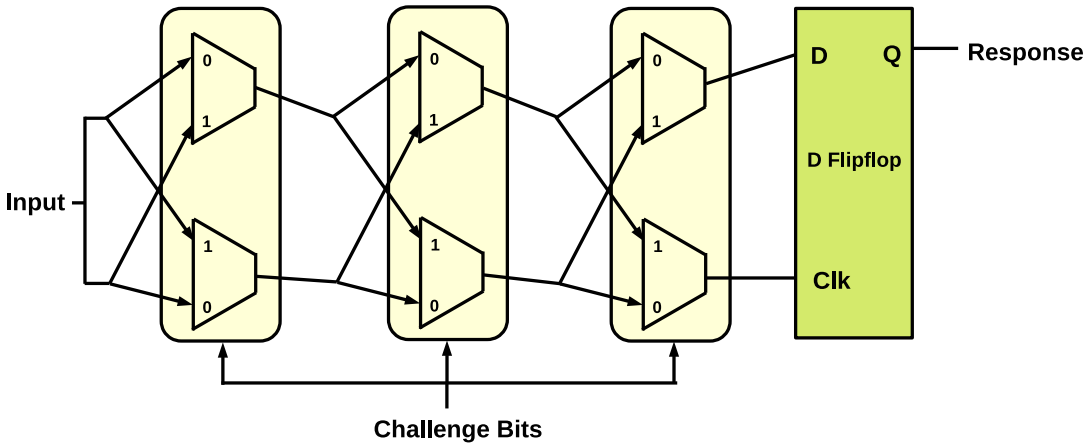


Figure 7.3: Arbiter PUF [54].

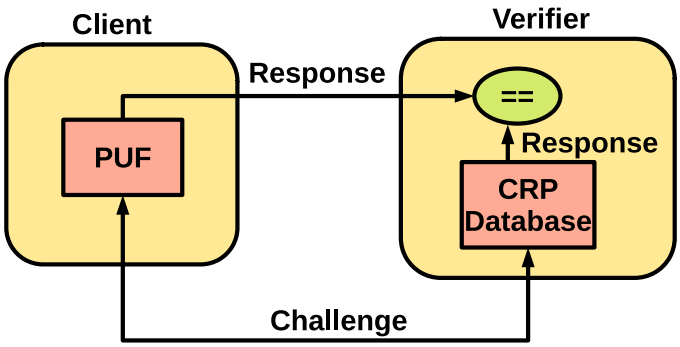


Figure 7.4: Generic strong PUF based authentication protocol [161].

7.2.6 Chaos-based PUFs

As previously discussed, the definition of chaos applies to deterministic nonlinear systems whose outputs are susceptible to small changes in initial conditions. If the initial conditions stem from physical conditions in the hardware, the system can be used as a PUF. Deterministic circuits can be very easy to design and the chaotic system produces time series which is unpredictable due to dependence on initial conditions which in turn changes the circuit's state in time.

A chaotic system is vulnerable to the tolerances of different parameters of the components in the design. The sensitivity arises from all the physical parameters affected by manufacturing process. Any change in the circuit's structure influences the characteristics of the chaotic system and opens the doorway to expand the *challenge* space. The same reason can be extended to explain why some circuits with same placement and routing behaves differently. After a certain number of iterations, it is possible for chaotic systems to display random behavior which can be used to design random number generators.

It has already been discussed that PUFs are vulnerable to attacks such as man-in-the-middle, emulation, reconfiguration and reverse engineering. In order to prevent these susceptibilities, the chaotic signals generated from logistic map are applied as *challenges* to the RO-PUF [168] to generate random numbers for cryptographic applications. In [57], researchers have used switchable chain ring oscillators to generate cryptographic keys. Miura *et al.* introduced a chaos PUF for mitigating counterfeit attacks where inductive coupling introduces wireless electromagnetic interactions between the package or board and chip into a PUF [110]. Chen demonstrated a way to mitigate machine learning based modeling attacks on PUFs by integrating an arbiter PUF with a converter and chaotic circuit [30]. In [97], modeling attack of CRPs are alleviated by using a CNT PUF with Lorenz chaotic circuit which enhances the difference in *responses* for similar *challenges*.

7.2.7 Authentication Protocol

Since the PUF acts as a "black-box", the verifier only has access to a set of CRPs and cannot determine the *response* to a brand new *challenge*. A typical authentication protocol is shown

in Fig. 7.4. The client side has a strong PUF that can generate many CRPs. The verifier has a database of the CRPs generated from the same strong PUF. The process of creation of the database is known as *enrollment* which is typically done by a trusted third party. The second phase is called *verification* where the client requests to be authenticated and the verifier chooses a random CRP from the database. The verifier sends the selected *challenge* to the client and keeps the corresponding *response* to be verified later. The client generates a *response* from the PUF by using the *challenge* it received from the verifier. The client sends back the generated *response* to the verifier and the verifier now matches the two *responses*, one received from the client and the other stored in the database. If the two responses match (i.e. the response bits are close enough), the device is authenticated. PUF authentication allows an error threshold which decreases the stringency on the stability requirement and eliminates the need for error correction. In low-cost authentication schemes, the verifier allows an error tolerance threshold or multiple attempts to authenticate the device before rejecting it as a fake. Error tolerance is a much preferred technique. As an example, the authentication method can set a threshold of 32 bits for verification of 128 bit *response* [40].

A strong PUF contains an exponential number of CRPs with respect to components whose characteristics depend on process variation. The verifier cannot allocate enough memory to save the entire CRP space of the strong PUF. A CRP cannot be used twice in an authentication protocol, since the attacker can send a previously authenticated *response* to decipher the CRP model. Periodic refreshing of the CRP database is mandatory. It is also necessary to ensure that the PUF produces the same *response* for a given *challenge* and is not affected by environmental conditions. It is important to keep in mind that each client will have a unique PUF so they can be authenticated individually. The verifier can store CRP tables for each client that connects to the system or network.

It is assumed that the attacker will only have read-only access to the data being transported between the client and the verifier. In order to prevent the adversary from impersonating the valid client, acknowledgments of the transported information through the public channel must be verified in a time bounded private channel before the information can be used for further authentication steps.

7.2.8 PUF Metrics

There are several metrics in order to evaluate the PUF design. Some of the most commonly used metrics are described below.

Uniqueness

Uniqueness is the ability of the PUF to generate unique signatures on different ICs for the same design. In order to measure uniqueness, the inter-chip Hamming distance is calculated between two responses generated from the same PUF on different chips while the applied *challenge* is fixed. Ideally, the uniqueness should be 50% which means each PUF generates a unique response. It is not an estimation of the actual probability of inter-chip process variation. The equation is given below for k chips:

$$Uniqueness = \frac{2}{k(k-1)} \sum_{i=1}^{k-1} \sum_{j=i+1}^{k} \frac{HD(R_i, R_j)}{n} \times 100\%. \tag{7.2}$$

where R_i and R_j are n-bit responses of chips i and j for a given challenge.

Uniformity

Uniform distribution of the number of ones and zeros is required in a PUF *response*. Ideally, the value of uniformity should be 50%. The equation for uniformity is defined as the percentage Hamming Weight (HW) of the response:

$$Uniformity = \frac{1}{n} \sum_{l=1}^{n} r_{i,l} \times 100\%. \tag{7.3}$$

where i represents the PUF instance and $r_{i,l}$ represents the l^{th} bit of an n-bit response. Hamming Weight is the Hamming distance from an all-zero string of the same length.

Bit-aliasing

Bit-aliasing occurs when different PUFs produce nearly identical *responses* which is an unwanted phenomena. Ideally, the value of bit-aliasing should be 50%. The equation for bit-aliasing is as follows:

$$Bit - aliasing = \frac{1}{k}\sum_{i=1}^{k} r_{i,l} \times 100\%. \tag{7.4}$$

where bit-aliasing of the l^{th} bit of an n-bit PUF response is measured by calculating the Hamming Weight (HW) across k devices and $r_{i,l}$ is the l^{th} bit of a response from chip i.

7.3 Proposed Computing System (PURCS)

In this work, a physically unclonable and reconfigurable computing system is proposed which provides both device authentication and mitigates IC counterfeiting via logic locking. The main component of the system is a three transistor chaotic map. In order to generate chaotic signals, a chaotic oscillator has been designed which has a map circuit in the forward path as well as the feedback path. The oscillator can be used to produce logic functions by altering four tuning parameters called the iteration number, control bit, bifurcation parameter and threshold voltage. The computing system is not built entirely with reconfigurable chaos-based logic gates but is a hybrid of standard CMOS logic gates and chaos-based logic gates. The purpose of using reconfigurable logic gate is that a single system can implement multiple functions. Replacing some of the logic gates in a system will allow the system to observe logic locking and the tuning parameters in the oscillator can act as the secret key. Each chaos-based logic gate has a 10-bit key. The key size increases with the number of replaced chaos-based gates in the system. The key space is large if sufficient gates are replaced and the attacker will have to spend exponential time in figuring out the secret key.

The computing system is called physically unclonable since process variation can be used to generate unique keys for each chip. The same system can be used to generate challenge-response pairs to authenticate devices. The obtained responses needs post-processing to achieve ideal PUF metrics such as uniformity, uniqueness and bit-aliasing. The computing system can be used for authentication where the key and primary input bits make up the *challenges* and primary outputs are the *responses*. Process variation in the manufacturing process ensures that different ICs will have different CRPs.

In this work, ISCAS'85 benchmark circuits are used to demonstrate that logic obfuscation and authentication can be performed using the same system. Hamming distance is calculated between the correct output and wrong outputs to ensure that 50% of the bits are corrupted on application of wrong key. The proposed design has low overhead compared to conventional circuits that contains both logic locking and authentication circuits.

7.3.1 Design of Chaotic Oscillator for PURCS System

In this work, the same chaotic map as shown in Fig. 3.1 has been used to produce chaotic signals. The reconfigurable 2-input chaos-based logic gate is designed using chaotic oscillator, DAC and comparators as shown in Fig. 7.5. Digital inputs made up of 2-bit data and 1-bit control are converted to analog voltage using a 3-bit DAC in order to apply to the input of chaotic map in the forward path. The map circuit produces two analog voltages as outputs from the two map circuits. Therefore, two comparators are used to convert the analog output to digital voltage so that the system can be used to generate Boolean functions. The system produces two outputs per iteration since two chaotic maps are used.

Fig. 7.6 shows the transient response of the chaotic oscillator where the applied bias voltage, V_c is 564 mV. An input, V_{in} of 750 mV is applied to the map when ϕ_1 is high. When ϕ_1 is low, ϕ_2 and ϕ_3 starts oscillating producing outputs O_{n+1} and O_{n+2}. A total of 10 output values of O_{n+1} and O_{n+2} can be obtained since ϕ_2 and ϕ_3 have five iterations each.

7.3.2 Bifurcation Diagram

The bifurcation diagram of the chaotic oscillator represents the periodic and chaotic regions as shown in Fig. 3.8. The $x-axis$ is the bifurcation parameter, V_c and $y-axis$ is the output voltage V_{out} of the chaotic oscillator which is the combination of O_{n+1} and O_{n+2}. The bifurcation diagram is obtained by applying an input voltage of 750 mV. The bifurcation parameter, V_c is swept from 0 V to 1.2 V with 2 mV increments. The bifurcation diagram has been plotted for 900 iterations.

The circuit has a period of two for initial values of V_c. The circuit can implement two functions in this region if all other tuning parameters are unchanged. The chaotic oscillator

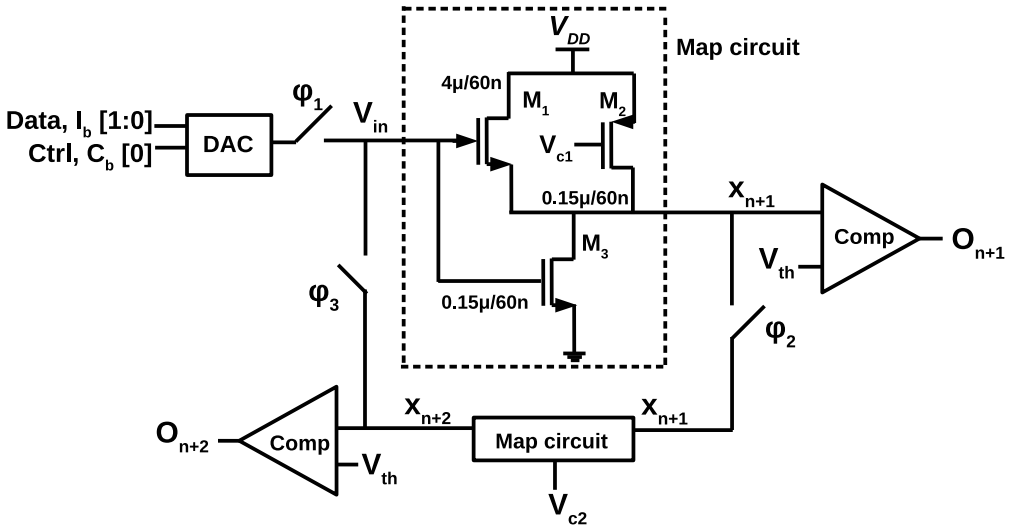


Figure 7.5: Reconfigurable 2-input chaos-based logic gate designed using chaotic oscillator, comparators and DAC [149].

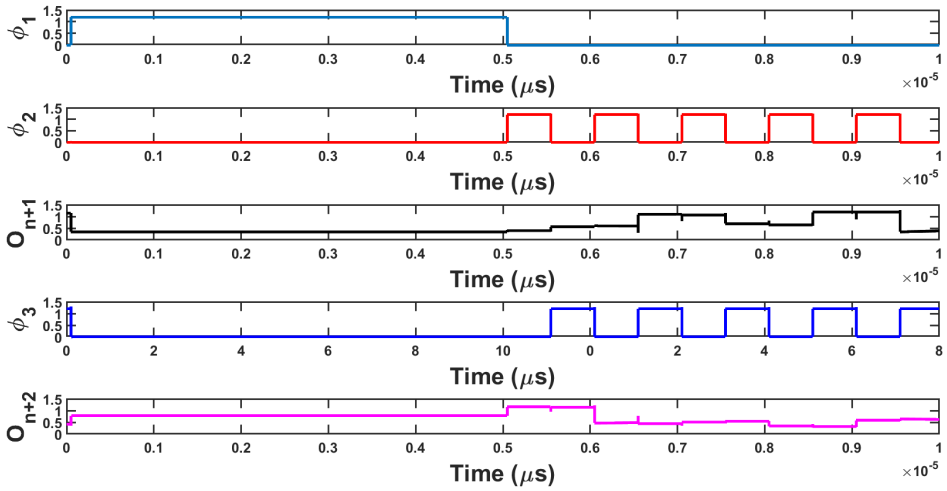


Figure 7.6: Transient response of the chaotic oscillator [149].

enters into the chaotic region after period-doubling cascade. The first chaotic region occurs when the value of V_c is between 470 mV and 566 mV. The second chaotic region occurs between 906 mV and 948 mV. In this work, the values of V_c has been chosen from the first chaotic region because the delay of the oscillator increases as V_c increases. Sixteen values of V_c has been chosen and the values are 486 mV, 488 mV, 490 mV, 506 mV, 510 mV, 512 mV, 518 mV, 520 mV, 524 mV, 526 mV, 528 mV, 538 mV, 540 mV, 546 mV, 556 mV and 564 mV. The 16 values represent 4 bits in the secret key of the chaos-based gate.

7.3.3 Chaos-Based Logic Gate Implementation

It has been discussed already that a chaos-based system can be implemented to design logic gates in section 3.3. The chaos-based logic gate is reconfigurable because of the multiple tuning parameters available in the circuit topology. Nonlinear system can be used to build logic gates that can implement multiple Boolean functions using the same building block. In this work, the chaos-based system is used as 2-input reconfigurable digital block which is capable of implementing all the possible 16 Boolean functions. The different functions can be achieved by changing the tuning parameters: bifurcation parameter, control bit, threshold voltage and iteration number.

The digital inputs are converted to analog values by using a DAC for applying to the chaotic oscillator. The oscillator in turn produces analog outputs and they are converted to digital values using comparators which utilizes a thresholding mechanism. The threshold voltage, V_{th} is chosen to be 625 mV by taking the median of the values obtained from 65 different ICs. The logic gate has a digital value of "1" if the analog voltage is greater than V_{th} and "0" otherwise.

Chaotic systems are sensitive to small changes in the initial conditions. The digital input to the chaos-based logic gate is represented by I_b and the control bit by C_b. The control bit perturbs the analog voltage going into the map, called V_{in}, to a slightly different value. The range of the output voltage is between 193.2 mV and 1.2 V. When C_b is equal to 0, the values of V_{in} are 193.2 mV (00), 480.9 mV (01), 768.5 mV (10) and 1.056 V (11). When the value of C_b is 1, the values of V_{in} changes to 337 mV (00), 624.7 mV (01), 912.3 mV (10) and 1.2 V (11).

Table 7.1: Evolution of chaotic oscillator output with number of iterations. (V_c= 552 mV, $C_b = 1$, $V_{th} = 625\ mV$) [149].

x_n (V)	x_{n+1} **(V)**				
	$n = 1$	$n = 2$	$n = 3$	$n = 4$	$n = 5$
0.337(00)	1.11(1)	0.58(0)	0.54(0)	0.72(1)	0.34(0)
0.625(01)	0.35(0)	1.18(1)	0.62(0)	0.42(0)	1.13(1)
0.912(10)	0.49(0)	0.91(1)	0.44(0)	1.08(1)	0.55(0)
1.2(11)	0.63(1)	0.39(0)	1.16(1)	0.62(0)	0.41(0)
Function (dec)	9 (XNOR)	6 (XOR)	1 (AND)	10	4

Table 7.2: Characterization table of the 2-input reconfigurable chaos-based logic gate for V_{th} = 625 mV [149].

Key (dec)	**Control inputs**			**Function (dec)**
	$V_c(mV)$	n	C_b	
1	486	19	1	7 (OR)
17	524	19	0	6 (XOR)
55	538	20	0	15 (1111)
91	546	21	0	1 (AND)
141	518	23	0	10 (1010)
478	556	33	1	0 (0000)
625	524	38	0	12 (1100)

In a chaos-based system, the system evolves in time with increasing number of iterations. The functions obtained from one iteration can be different from the next iteration. As a result, iteration number is another tuning parameter that can be used to change the functionality of the logic gate. In this work, a total of 50 analog voltages were generated from 25 iterations. The first 18 values are not used since they represent values in the transient period. The last 32 iterations are used to make digital gates. Two-bit chaos-based reconfigurable logic gate can implement $2^{2^2} = 16$ different functions. The generated functions are numbered in decimal starting from 0 (0000) to 15 (1111).

Table 7.1 shows the evolution of the chaotic oscillator for different iterations. All the other parameters are fixed such as $V_c = 552\ mV$, $V_{th} = 625\ mV$ and $C_b = 1$. In the first iteration, an XNOR (9) gate is generated and in the second iteration an XOR (6) gate is produced. Five different iterations produce five different functionalities.

7.3.4 Characterization of the Chaos-based Logic Gate

The different tuning parameters are converted to digital values in order to use the chaotic oscillator as a digital system. 16 values of V_c were chosen in section 7.3.2 using the bifurcation diagram. The threshold voltage of the comparator was chosen to be 625 mV by taking the median value of all the analog voltages across the different chips. The chaos-based logic gate has one control bit which changes the initial condition of the chaotic oscillator to generate a new pattern of voltages. The output of a chaotic oscillator evolves in time with increasing number of iterations. The last 32 out of 50 iterations are considered to design the logic gate. The digital configuration key of the chaos-based logic gate is made up of 10 bits where V_{th} has 0 bits, C_b has 1 bit, V_c has 4 bits and iterations number has 5 bits. The key has the following configuration:

$$key(10) = V_c(4), n(5), V_{th}(0), C_b(1). \tag{7.5}$$

Table 7.2 shows a part of the characterization table for the 2-input reconfigurable chaos-based logic gate in a single chip. It can be seen that changes in tuning parameters changes

the functions implemented by the logic gate. The first column in the table represents the 10-bit key in decimal value. There are 1024 possible combinations of key for each chaos-based logic gate. The table shows that when key = 1, the generated function is an OR gate.

7.3.5 Functions Generated by the Logic Gate in Different ICs

Process variation in deep sub-micron technologies like 65 nm is a major concern. It is possible that characteristics of a transistor will vary significantly in different ICs due to process variation. Monte Carlo simulations have been performed across 65 chips to study the effect of process variation on the functionality generated by the chaos-based logic gate. Chaotic oscillators are susceptible to minute changes in the initial condition because of which the functions produced in different chips vary significantly. Table 7.3 demonstrates how the functions vary in different chips for the same tuning parameters. When key = 20, the functions generated in three different ICs are 9, 2 and 8. It means that different chips will have different key to execute the same functionality. Therefore, each chip must be characterized individually to configure the correct key. Monte Carlo simulations showed that mismatch variation in each IC is negligible. Therefore, characterizing one chaos-based logic gate from one chip should be sufficient to obtain the characterization table.

The reconfigurable chaos-based logic gates are built for use in logic locking. It is essential that each chaos-based logic gate can implement the necessary functions such as AND, XOR and OR. Fig. 7.7 shows the number of AND, OR and XOR functions for 10 different ICs. The number of obtained functions vary in each chip but it is possible to implement all the necessary Boolean functions using chaos-based logic gate.

7.3.6 Reliability of the Functions Generated from Logic Gate

The most important feature of chaotic systems are that they are sensitive to small perturbations in initial conditions. It has been evaluated that a mere 3 $^{\circ}C$ increase in temperature results in different functionality from the same chip. The effect of temperature variation is shown in Fig. 7.8. The key combinations become unreliable and implementing logic functions utilizing the chaotic system becomes a challenge. Logic functions are realized

Table 7.3: Characterization table of the reconfigurable chaos-based logic gate in different chips for $V_{th} = 625\ mV$ [149].

Key (dec)	**Control inputs**			**Function (dec)**		
	$V_c(mV)$	n	C_b	IC 1	IC 2	IC 3
20	528	19	1	9	2	8
73	510	21	0	10	6	0
112	520	22	1	9	3	11
151	538	23	0	8	0	13
259	488	27	0	2	4	0
412	546	31	1	11	2	10

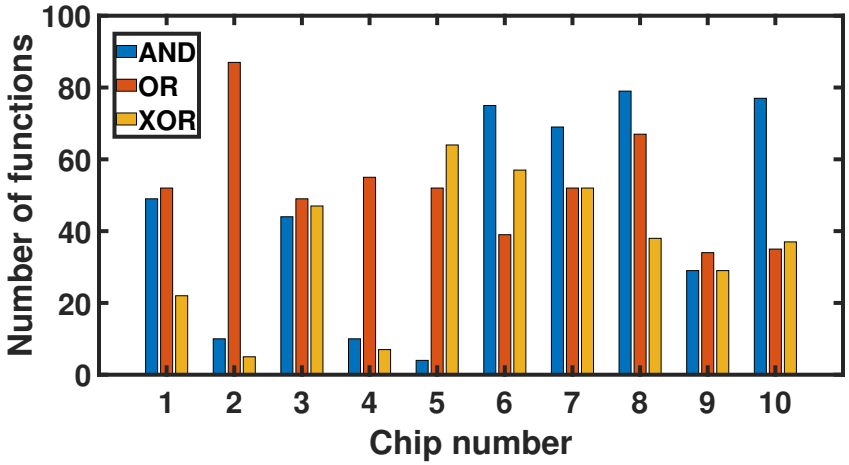


Figure 7.7: Number of AND, OR and XOR functions in 10 chips [149].

from the oscillator by applying a digital value at the input of DAC which converts it to an analog voltage. The analog voltage acts as the seed to the oscillator. As discussed earlier in section 3.3.2, a comparator with threshold voltage V_{th} is required to convert the analog output of the oscillator to digital value. The analog voltages that are in close proximity to the threshold voltage can lead to potential unreliable functions. Outputs that have high noise margin from V_{th} generate reliable functions.

A study has been performed on the reliability performance of the proposed chaos-based logic gate due to changes in supply voltage and temperature. Ideally, the functions should be the same for the same key for varying environmental conditions. The temperature has been altered from 0 $°C$ to 100 $°C$ and the supply voltage (1.2 V) has been changed by 4%. The effect of slight supply voltage variation is shown in Fig. 7.9. The characterization table in Table 7.2 has been re-simulated for random temperature and supply voltage variations in the chosen range. 50 new characterization tables are generated and compared against the original table to check the number of functions that remain stable for the same key configuration. The reliability test shows that 211 out of 1024 functions are stable for 4% supply voltage fluctuation and 10% voltage variation leads to 206 stable functions. The reliable key size for each chaos-based logic gate is almost 8 bits. When the chaos-based logic gate is characterized, only the reliable key configurations should be chosen to generate the desired function.

7.3.7 Creating the Characterization Table

In order to create the characterization table, a separate chaos-based logic gate will be placed in the manufactured chip. The characterization will be performed by a trusted manufacturer or designer. Each chaos-based logic gate has 12-bit input in total including the 10-bit key and 2-bit digital input. The 2-bit input has 4 combinations and the 10-bit key has 1024 input combinations. A total of 4096 $(4 * 1024)$ combinations need to be explored to make one characterization table for a specific supply voltage and temperature pair. Standard scan chain based mechanism can be used to apply the inputs serially to the logic gate.

The same number of input vectors should be applied for a separate voltage and temperature pair in order to create a new characterization table. After generating at least

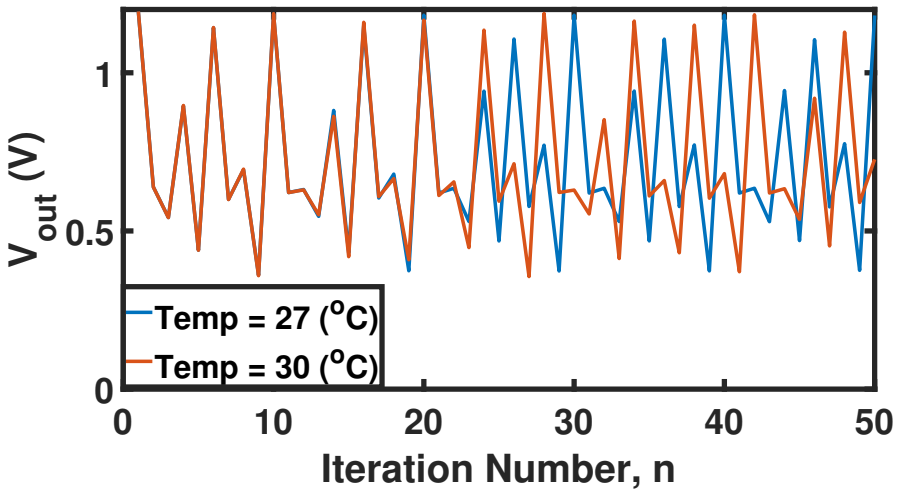


Figure 7.8: Effect of temperature variation on the oscillator output (V_{in} = 193.2 mV and V_c = 490 mV).

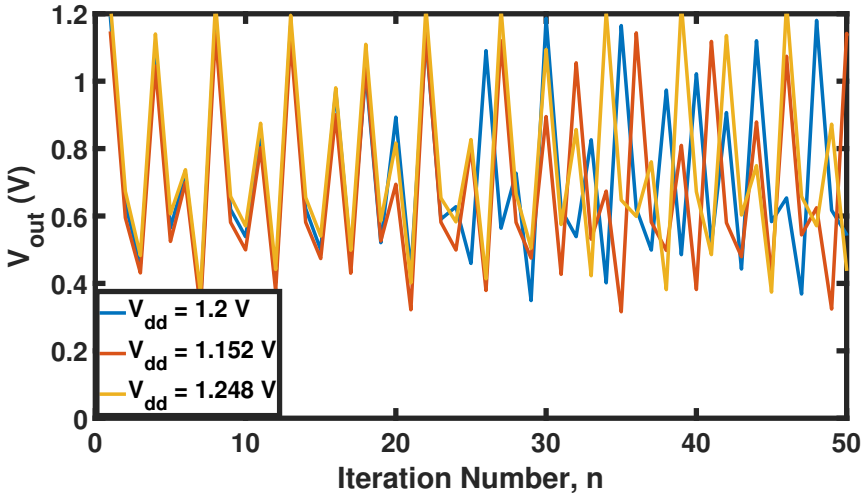


Figure 7.9: Effect of supply voltage variation on the oscillator output (V_{in} = 193.2 mV and V_c = 520 mV).

50 different characterization tables for 50 different voltage and temperature pairs, the tables will be compared to find the functions which remained stable. These stable functions are the reliable functions for that particular chip. Monte Carlo simulations revealed that the intra-chip variation is insignificant and does not affect the functions of the logic gate so one block needs to be characterized in each chip. Each chip must be characterized separately due to the effects of process variation. After characterization of each chip is complete, access to the chaotic block is disabled ending further characterization attempts. This can be achieved by burning supporting fuses or laser burning the access wires [67].

7.3.8 Replacement Algorithm

In a computing system, if all the logic gates were replaced by chaos-based logic gates, then the overhead will be significantly large. The aim of this work is to design a system which is a mixture of standard CMOS logic gates and 2-input reconfigurable chaos-based logic gates. There are two types of key gate inclusion method: insertion and replacement. It is desirable to use an efficient algorithm to insert/add the chaos-based logic gates. Conventional logic obfuscation techniques uses different insertion techniques such as fault-analysis based (FLL), random (RLL) and strong interference based (SLL). The fraction of gates that needs to be inserted is determined by calculating the Hamming distance between correct and wrong outputs. The most popular technique for replacement based logic locking is by calculating the controllability and observability of the nodes. Controllability and observability are two important characteristics of a logic design.

Controllability: Controllability measures how easily the inputs of a logic gate can be controlled. The input patterns that are never produced at the input of the node are determined and if the number is high, it means that the gate is less controllable. If the node only has primary inputs connected to it, it means that the gate is very controllable and all the input pattern combinations can be applied to the gate. However, if a node is buried inside the logic network, it is possible that the node might not encounter all the possible input patterns even though all the input patterns have been applied to the primary inputs.

Controllability is measured by calculating the probability of different input combinations occurring at the inputs of the gate. Each of the input combinations should occur with equal probability for a gate to be controllable. The equation for controllability is given below:

$$C = log_4 \frac{1}{p_0^2 + p_1^2 + p_2^2 + p_3^2}. \tag{7.6}$$

where p_0, p_1, p_2 and p_3 is the probability of occurrence of (0,0), (0,1), (1,0) and (1,1) at the inputs of the selected gate and if the probability is equal for all possible input combinations, then the gate has a controllability of 1.

NOT gates can be replaced in a circuit in two ways. They can be implemented using XOR gates where one of the inputs is tied to the supply voltage or 'HI'. If a buffer needs to be replaced, then one of the inputs of the XOR gate is connected to the ground node or 'LO'. Another way to implement a NOT gate is to tie the inputs of NAND/NOR gates. In the latter case where the inputs are tied, the equation for controllability will be changed to the following:

$$C = log_2 \frac{1}{p_0^2 + p_1^2}. \tag{7.7}$$

where p_0 and p_1 is the probability of (0,0) and (1,1) at the input of the selected gate.

Observability: Observability measures how easy it is to propagate the values of inaccessible gates to the primary outputs. Gates that are near the primary outputs are highly observable but they are less controllable. Similarly, gates near the primary inputs are more controllable rather than observable. Observability is also defined by values where the certain output values do not occur. If the node is connected to primary outputs, then the node is very observable since every output value can be observed at the primary outputs. If the output of a gate does not alter other nodes, then the node is not observable.

Observability ensures that the output value of internal nodes are propagated to the primary outputs without being blocked by outputs of other gates. For example, an AND/NAND blocks the propagation of the other signal if one of the inputs is 0. Similarly, OR/NOR should have a value of 0 in order to propagate the value of the other input to the

primary output. Observability is measured independently for each primary output and the average value is calculated at the end.

Testability: In replacement based algorithm, each node is ranked based on the controllability of the input nodes and observability of the output nodes. Controllability and observability is measured by applying 1000 random input patterns to the system under consideration. Each gate in the system is sorted from high to low based on the testability of the gates which is the product of controllability and observability. The most testable gates are replaced by chaos-based logic gates in order to create the hybrid system. The algorithm for replacement is shown in Fig. 7.10.

Example of Replacement Algorithm on C17 Benchmark Circuit

Fig. 7.11 shows a circuit from the ISCAS'85 benchmark suite. This circuit is used as an example to demonstrate replacement based algorithm using testability. Gate 3 from Fig. 7.11a is chosen for testability calculation. The circuit has 5 inputs so there are 32 possible input combinations. Probability of input combinations are evaluated for nodes, d and n_3. The probability of occurrence of (0,0), (0,1), (1,0) and (1,1) are calculated to be $\frac{1}{8}$, $\frac{3}{8}$, $\frac{1}{8}$, $\frac{3}{8}$, respectively. The value of controllability is calculated to be 0.84 using equation 7.6. Observability is calculated by measuring the probability of propagation of the value of node, n_2 to the primary outputs. The output of n_2 will propagate to primary output, out_1 if node, n_1 is 1. Probability of n_1 being 1 is the observability of gate 3 to out_1 which is $\frac{3}{4}$. Similarly, the probability of node, n_4 being 1 is the observability of gate 3 to out_2 which is $\frac{7}{8}$. Observability is calculated by averaging the observability of the individual primary outputs.

7.3.9 Configuration Key of the Hybrid Circuit

The functionality of the reconfigurable chaos-based logic gate depends on the application of a valid key. Due to process variation, the correct key will be different for different chips. Post-manufacturing characterization is required to find the correct key for a particular chip.

The keys for AND, OR, XOR, NAND, NOR and XNOR gates are represented by K_{and}, K_{or}, K_{xor}, K_{nand}, K_{nor} and K_{xnor}. The secret key for each chip will be a combination of the

Step 1.
Solve the circuit for 1000 random input patterns
Step 2. ***Controllability***
for all n ϵ nodes **do**
    calculate the probability of (0,0), (0,1), (1,0), (1,1) at input
    calculate controllability = $\log_4(1/(p_0^2+p_1^2+p_2^2+p_3^2))$
    if same input
        calculate the probability of (0,0), (1,1) at input
        calculate controllability = $\log_2(1/(p_0^2+p_1^2))$
    end if
end for
Step 3. ***Observability***
for all n ϵ nodes **do**
    find paths to the primary outputs
    for all paths
        find the type of gates in the path to the output
        find the probability of occurrence of '1' in the other input for AND/NAND and '0' for OR/NOR
    end for
    calculate observability by finding the average of all outputs
end for
Step 4. ***Testability***
for all n ϵ nodes **do**
    calculate testability = controllability x observability
end for
return the sorted gate index list

Figure 7.10: Algorithm for replacing gates using testability method [149].

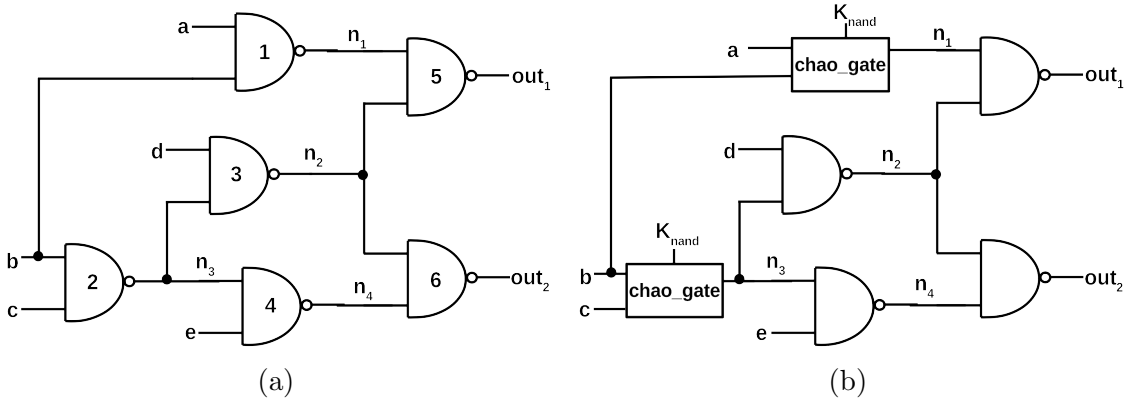


Figure 7.11: (a) Unobfuscated circuit from ISCAS'85 benchmark suite (b) Obfuscated circuit where two NAND gates are replaced with reconfigurable chaos-based logic gates and the secret key is 20 bits long [149].

keys of individual gates that are replaced. Fig. 7.11b shows the logic locked circuit where two NAND gates have been replaced by 2-input reconfigurable chaos-based logic gates. K_{nand} is the key for each gate which is made of V_c, V_{th}, C_b and n. The secret key of the entire circuit is 20 bits. The circuit will produce incorrect functionality if wrong key is applied. The correct key is made up of two K_{nand} and can be written as:

$$K_{correct} = \{K_1, K_2\} : K_1, K_2 \in K_{nand}. \tag{7.8}$$

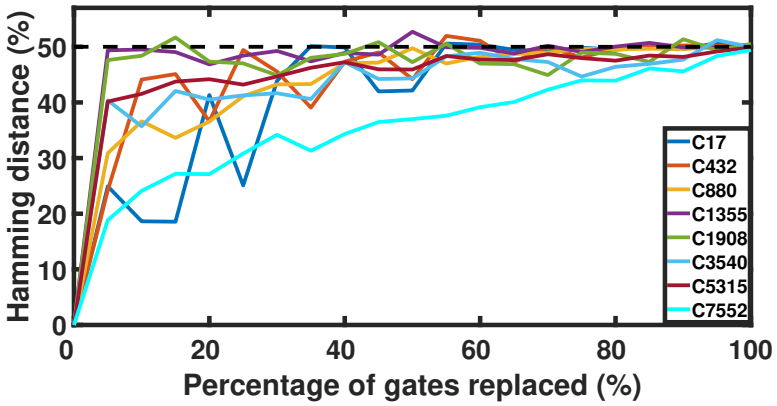
7.4 Simulation and Results

Eight circuits have been chosen from ISCAS'85 combinational benchmark suite. The circuits are C17 (6 gates), C432 (231 gates), C880 (462 gates), C1355 (590 gates), C1908 (1087 gates), C3540 (2050 gates), C5315 (2981 gates) and C7552 (4056 gates). All the gates have been converted to 2-input and 1-output since the reconfigurable chaos-based logic gates have the same configuration.

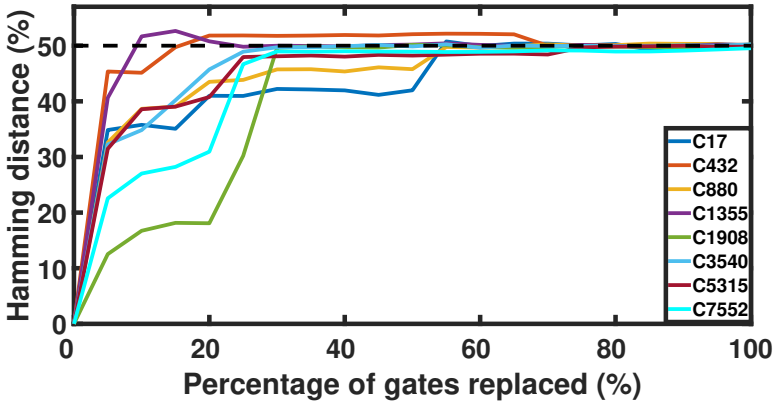
7.4.1 Logic Locking Results

Application of wrong key should produce incorrect output for all input patterns in a logic locked circuit. An invalid key should change half of the output bits so that the calculated Hamming distance between the correct output and the wrong output is 50%. The reconfigurable chaos-based logic gates have been replaced using two methods: calculating the testability of all the nodes in the circuit and random replacement. Random replacement might not alter the output bits when wrong key is applied. When testability based replacement algorithm is used, fewer gates are required to achieve 50% Hamming distance. The testability of all the nodes are calculated and sorted from high to low. The gates with the highest testability values are replaced with chaos-based logic gates.

Fig. 7.12 shows the plot of Hamming distance with 5% increments in the replacement percentage of the gates. Correct and random keys were applied to the logic locked circuit in order to measure the Hamming distance. The two replacement methods, random and testability based techniques are compared. The slope of the lines indicate the performance



(a)



(b)

Figure 7.12: Hamming distance vs. percentage of gates replaced in ISCAS'85 benchmark circuits (a) Gates replaced randomly (b) Gates replaced by measuring testability [149].

Table 7.4: Percentage of gates replaced to reach almost 50% Hamming distance in ISCAS'85 benchmark circuits [149].

Benchmark	**Random Replacement** (%)	**Testability Based Replacement** (%)
$C17$	70	55
$C432$	80	20
$C880$	85	55
$C1355$	55	25
$C1908$	70	30
$C3540$	95	30
$C5315$	95	25
$C7552$	100	30

difference of the techniques. If the line is steep, 50% Hamming distance is reached with fewer amount of replaced gates and the performance overhead will be smaller as seen in Fig. 7.12b. Once a circuit reaches, 50% mark, the Hamming distance value does not deviate much even though more gates are inserted in testability based replacement method. The advantage is that the key size can be increased without ruining the 50% Hamming distance.

Table 7.4 compares the percentage of gates that need to be replaced to reach 50% Hamming distance. The second column displays the percentage of total gates that need to be replaced by using random replacement technique whereas the third column displays the percentage required for testability based replacement technique. It can be clearly seen that random replacement technique requires more percentage of gates than testability based technique. The results in the table also indicate that the percentage of gates that need to be replaced to achieve 50% Hamming distance depends on the circuit topology.

7.4.2 Authentication Results

This work demonstrates that any system that has reconfigurable chaos-based logic gates can be used for authentication purposes. The benchmark circuits display PUF characteristics as there are significant changes to the outputs of the system due to process variation. Different key and input combinations are considered as *challenges* and the outputs are *responses*. Fig. 7.13 represents the post-processing scheme required for a benchmark circuit to reach ideal PUF metrics. The *response* is calculated for 10 random input patterns applied to the circuit. When clk_1 is high, the initially generated output bits are XORed with 0. When clk_1 stops, clk_2 starts and the result of the first iteration are XORed with the newly generated output by applying a second set of inputs. This process continues until 10 outputs are XORed and the result is stored in a register. The output stored in the register has the same number of output bits as the original circuit. After completion of 9 cycles, clk_2 stops and clk_3 starts in order to calculate the final *response*. The final *response* bits are half in number compared to original number of bits due to XORing amongst each other. For example, if the immediate result stored in the register has 4 bits, then bit 1 is XORed with bit 3 and bit 2 is XORed with bit 4. The final *response* will be 2 bits.

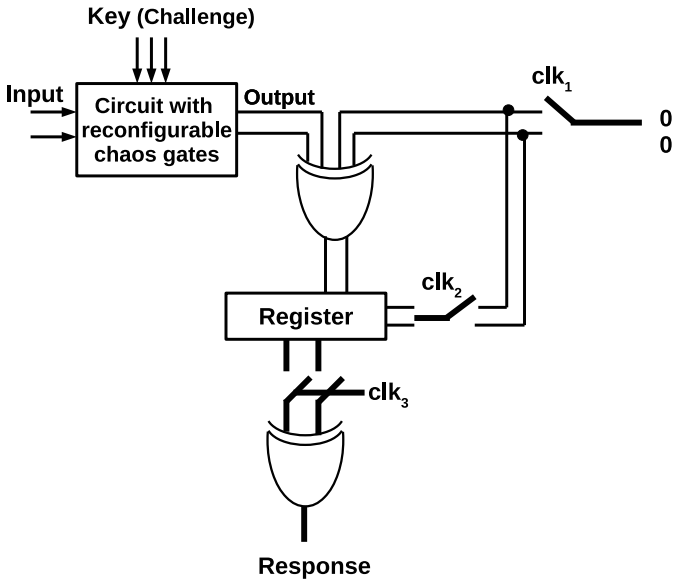


Figure 7.13: Post-processing scheme for authentication [149].

Table 7.5: Results of standard PUF metrics for the benchmark circuits using testability based replacement method [149].

Benchmark	**Uniqueness** (%)	**Uniformity** (%)	**Bit-Aliasing** (%)
$C17$	50.01	49.93	50.13
$C432$	49.99	50.03	50.03
$C880$	50.01	49.99	49.87
$C1355$	50.01	50.05	50.05
$C1908$	49.99	49.82	49.82
$C3540$	50.03	50.01	50.01
$C5315$	49.85	49.52	48.71
$C7552$	49.99	50.04	49.07

The post-processing scheme is required to ensure that the outputs of the benchmark circuits meet the ideal PUF metrics described in section 7.2.8. The three PUF metrics uniqueness, bit-aliasing and uniformity are calculated after the final *response* is generated from the post-processing scheme. Table 7.5 displays the results of the PUF metrics after the circuit gates have been replaced using testability based replacement algorithm. Percentage of gates replaced in the circuits is the same as the third column of Table 7.4. The results are obtained after post-processing scheme in Fig. 7.13 has been applied. All the metrics have the ideal value of 50% which means that the circuits can also be used for authentication of devices. The circuits with large number of outputs can be considered a strong PUF since they will have a large CRP space.

7.5 Security Performance

7.5.1 Security of Logic Locked Circuits

The main aim of using logic locking is to hamper the adversary's ability to change the contents or reverse engineer the functionality of an IC. The foundry can attempt to produce extra ICs without the owner's permission, but the chip will not function properly without the knowledge of the correct key. The adversary can make an educated guess about the key but the key space is extremely large and it would take exponential time and resources to decipher the valid key. Moreover, when standard gates are replaced with chaos-based logic gates, each IC will have a different valid key due to the effect of process variation. The innate complexity of a chaos-based system can be leveraged to suppress information leakage to the attackers.

Reconfigurable chaos-based logic gates look similar to the attacker in the layout-level geometry. It will be difficult for the adversary to learn the functionality of each block. If XOR/XNOR gates are used, the attacker can easily learn the entire netlist using reverse engineering techniques. XOR/XNOR gates provide only 1-bit key after insertion whereas each chaos-based logic gate provides 10-bit key. Chaos-based logic gates are able to generate all the possible functions which poses a huge burden on the attacker. If 10 gates are replaced

with chaos-based logic gate, then the secret key will be 100 bits long. In practice, the chip will have billions of transistors and a considerable amount of gates will be replaced by the chaos-based logic gates. The key size increases with increase in number of gates replaced. The key can be stored in a tamper-proof memory. After characterizing each IC, the designer can blow out the fuses of read/write circuitry which provides access to the memory.

Some of the common attacks against logic locked circuits are discussed below.

Side Channel Attack (SCA)

This procedure utilizes side channel information such as timing and power in order to break security primitives. When gates are replaced in an obfuscated netlist, the attackers look for power signatures when the replaced gate tries to perform a different functionality. Chaos-based reconfigurable gates can successfully mitigate power analysis based side channel attack. The attacker is not able to distinguish between different functionalities by applying popular machine learning algorithms [105, 104].

Brute Force Attack (BFA)

The attacker can directly try all the possible combinations of functionality that an obfuscated gate can perform [129]. The attacker iterates through each possible combination and simulates the locked IC to generate the output and compares the generated output with a functional IC bought from the market. If the generated output matches the output of the functional IC, then the secret key is revealed.

The functionality of each chaos-based reconfigurable logic gate appears as 16 different functionalities for the attacker. The brute force effort of the attacker will be $16^N = 2^{4N}$ where N is the number of gates replaced in the design except the gates that are directly connected to the primary output. Such an attack poses an exponential effort on the part of the attacker especially if the value of N is large.

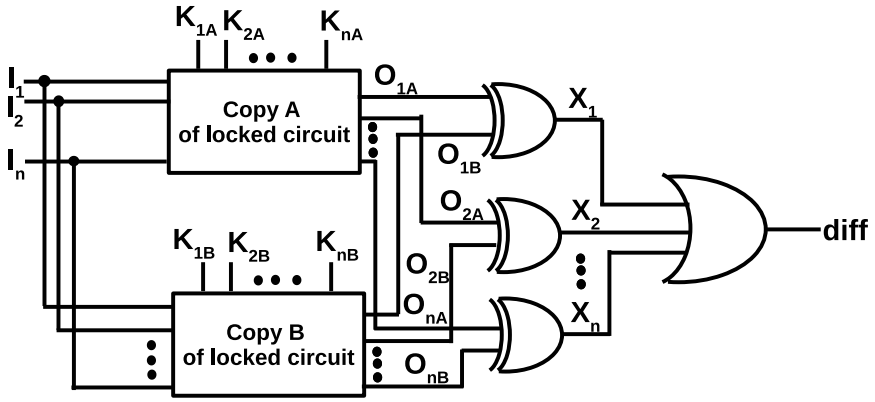
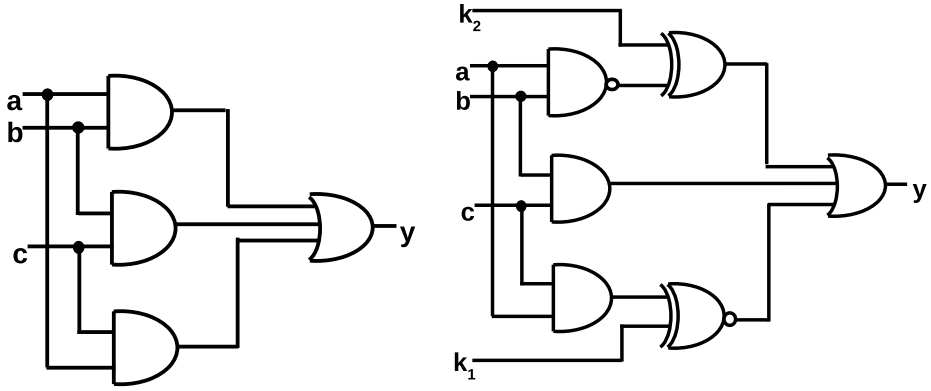


Figure 7.14: Circuit to determine distinguishing input patterns (DIPs) [182].



(a) (b)

Figure 7.15: Logic locking example (a) Original circuit (b) Locked circuit [160].

Boolean SAT Attack

It is essential that the proposed computing system is resistant against Boolean SAT attack for the logic locking scheme to be successful [160]. The SAT attack model assumes two things:

1. The attacker has access to the mask information and layout. The gate-level netlist can be reverse-engineered from the available information [165].

2. Random input patterns can be applied to the IC and there is a golden IC (bought from the open market) from which the outputs can be compared to the outputs obtained from the locked IC.

The attacker can only apply a certain number of input patterns to the activated chip and determine the correct key based on these results. If a set of input-output patterns are given, a SAT solver will be able to find a key that satisfies the applied patterns only. The key might generate correct output for the observations made but the problem is that the key needs to fulfill all the possible input-output pairs. Moreover, verifying that the correct key is extracted might require application of all possible input patterns. For example, if there are M input patterns in a system then 2^M input patterns must be applied in order to verify the correctness of the key. This method seems impractical since the number of input patterns can be significantly high depending on the size of the system.

SAT attack finds distinguishing input patterns (DIPs) by comparing two outputs generated from two different keys. The generated outputs are matched with the output from a golden IC. If the output is wrong, the key is eliminated [160]. A single DIP is capable of eliminating multiple incorrect keys [182]. The algorithm continues until all the DIPs are evaluated. After the program converges, the keys that are not eliminated are the probable correct keys for the locked chip being considered. The correct key satisfies all the possible input-output patterns of the locked IC. SAT attack needs only a small amount of observations from the golden IC in order to extract the secret key.

The DIPs are found by utilizing a circuit shown in Fig. 7.14. The same primary inputs are applied to the two copies of the locked circuit for different set of keys. The outputs from the two locked circuits are XORed and then Ored to generate the *diff* signal. The

Conjunctive Normal Form (CNF) of the resulting system is created and passed to the SAT solver. The SAT solver then finds an input (DIP) that outputs $diff = 1$ where the outputs of the two circuits are different. The DIP is then applied to the activated IC to find the correct output. The correct output is used to eliminate the incorrect keys. In this manner, more DIPs are generated in order to eliminate more key values. A new value is passed on to the SAT solver in every iteration and the formula is updated. The SAT attack stops when new DIPs are not found and all the incorrect keys have been discarded [182].

An example of original circuit and locked circuit is shown in Fig. 7.15. Suppose the SAT solver generates a DIP of $(a, b, c) = (1, 0, 1)$ in the first iteration for which the output, y is evaluated to be 1 from the original circuit. Initially, the key used is $(k_1, k_2) = (0, 1)$ which yields an output, $y = 0$ from the locked chip. The key pair $(0, 1)$ does not represent the correct key pair. In the second iteration, the DIP is equal to $(a, b, c) = (0, 0, 0)$ for which y is equal to 0. The SAT solver terminates because the only key pair that satisfies the original output value is $(k_1, k_2) = (1, 1)$. The fault-analysis attack cannot be carried out on the locked circuit in Fig. 7.15b because the key values cannot be propagated to the primary output. In earlier days, the only way to retrieve the secret key from the given locked circuit was brute-force attack by searching the entire key space. Now, it can be seen that Boolean SAT attack is able to extract the secret key.

Another example of SAT attack on a logic locked circuit is shown in Fig. 7.16. Table 7.6 represents the output of the original circuit, y shown in Fig. 7.16a. The output of the locked circuit for different key values are also shown in the following columns. There are 3 key inputs so there are 8 possible combinations of the key. The key values are represented as k_0, k_1, k_2...k_7. The SAT solver takes four iterations to decipher the correct key. In iteration 1, $(a, b, c) = (0, 1, 1)$ which helps in eliminating key, k_4 since the output is wrong. In the second and third iteration, key values, k_1 and k_7 are also eliminated for DIPs with values $(a, b, c) = (1, 1, 1)$ and $(a, b, c) = (1, 0, 1)$ respectively. The DIP, $(a, b, c) = (1, 0, 0)$ used in the fourth iteration eliminates all the incorrect keys and identifies the correct key, $k_6 = 110$. The attack would been much faster if the first DIP chosen was $(a, b, c) = (1, 0, 0)$. The execution time is dependent on the order of the DIPs applied to the SAT solver. The DIPs are chosen

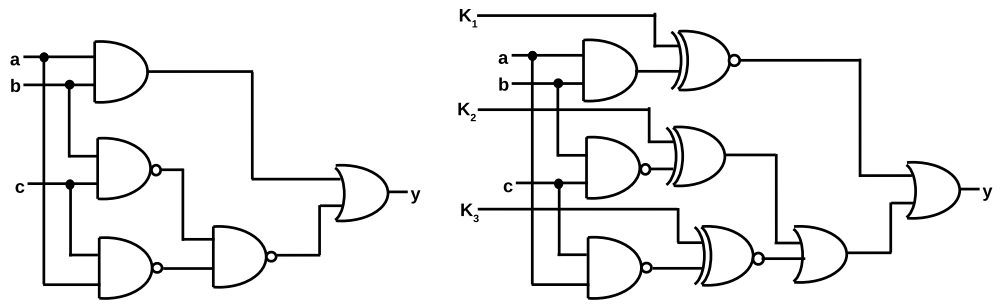


Figure 7.16: Logic locking second example (a) Original circuit (b) Locked circuit [182].

Table 7.6: SAT attack on a logic locked circuit shown in Fig. 7.16b. [182].

No.	**a**	**b**	**c**	**y**	**Output y for different key values**								**Eliminated keys**
					k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	
0	0	0	0	0	1	1	1	1	1	1	0	1	
1	0	0	1	0	1	1	1	1	1	1	0	1	
2	0	1	0	0	1	1	1	1	1	1	0	1	
3	0	1	1	1	1	1	1	1	0	1	1	1	iteration 1: k_4
4	1	0	0	0	1	1	1	1	1	1	0	1	iteration 4: all correct
5	1	0	1	1	1	1	1	1	1	1	1	0	iteration 3: k_7
6	1	0	1	1	1	1	0	1	1	1	1	1	
7	1	1	1	1	1	0	1	1	1	1	1	1	iteration 2: k_1

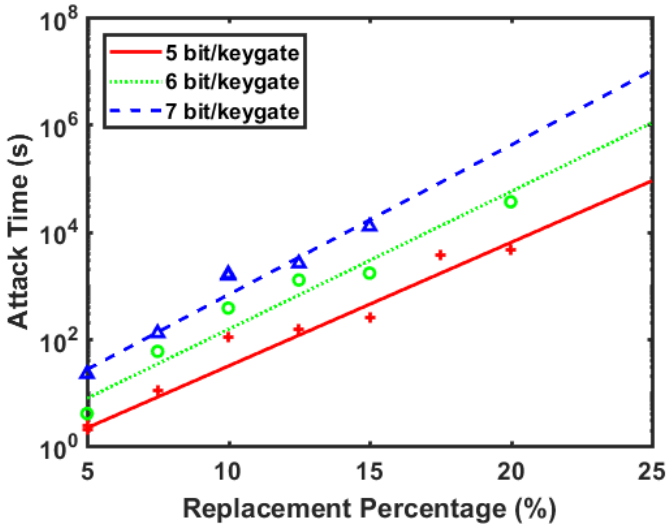


Figure 7.17: SAT attack implemented on C432 benchmark circuit for varying percentage of gates and key sizes [149].

randomly. If the selected DIP can eliminate more incorrect key values per iteration, then the attack time will be much faster.

Researchers have proposed multiple solutions in literature to make the locked chip resistant against SAT attack. SARLock proposed a lightweight technique to increase the number of DIPs required to extract the secret key which increases the attack time exponentially [182]. Anti-SAT is another lightweight circuit block that aims to mitigate the SAT attack by increasing the total number of iterations required to obtain the correct key [178]. In Anti-SAT, the execution time of the attack is an exponential function of the key size in the circuit. Unfortunately, SAT attack mitigation techniques are vulnerable to "removal" attacks. Therefore, researchers have proposed a solution that creates densely populated cyclic circuits that are obfuscated. Cyclic obfuscation is achieved by adding dummy gates and wires to the original circuit and creating logical loops that cannot be solved by the SAT solver since a directed acyclic graph (DAG) does not exist [146]. Double DIP is another technique that eliminates two wrong keys per iteration ensuring that it does not take exponential time to retrieve the correct key by using SAT attack [152].

In traditional logic locking schemes, insertion based method is utilized and each key gate adds only a single key bit. This poses a restriction on the number of key size that is required to make the attack computationally infeasible for an adversary. Even the look-up-table (LUT) based logic locking scheme fails to provide the desired flexibility in terms of key size since the memory required increases exponentially with the increase in number of key bits. Chaos-based logic gates provide the flexibility of increasing the key size without adding a substantial hardware cost.

The complexity of the logic locking scheme, discussed in section 7.3.8, has been analyzed by implementing SAT attack. The attack tool is available online at [2]. The machine used for carrying out the attack on the proposed computing system is Intel Xeon E5-2660 CPU with 2.6 GHz clock frequency and 125 GB memory. The attack complexity is measured by the time it takes to recover the secret key for different percentage of chaos-based logic gates replaced in the system. Chaos-based logic gates are capable of generating multiple functionality for varying input key bits. The key space of the chaos-based logic gates can be increased by replacing more gates. SAT attack has been performed on C432 benchmark

circuit for varying percentage of chaos-based gates present in the system and varying key sizes. The results are displayed in Fig. 7.17.

It is seen from the results that the attack complexity increases by increasing the number of chaos-based gates used in the system for a given key size. Attack complexity can also be enhanced when the number of key gates replaced is fixed but the key size of each chaos-based logic gate is increased. The proposed computing system is vulnerable to process variation so the secret key is different for each chip. If an adversary is able to extract the secret key by utilizing SAT attack, the key is only valid for that particular chip. The attacker will have to perform the time intensive procedure of deciphering the key for each individual chip.

Reconfigurable chaos-based logic gates have a non-uniform functionality space which adds to the SAT attack complexity. Most of the generated functions from each chaos-based logic gates are either ones or zeros. The desired functions such as OR, NOR, AND, NAND, XOR and XNOR have less probability of existing in the entire functionality space. For example, if a 2-input chaos-based logic gate with 10-bit key has uniform functionality space, each logic function will occur 64 times and will have a distribution of 1/16. In reality, the average probability of finding a desired function is much less than 1/16 as can be seen in Fig. 7.7. As a result, discovering the correct key by eliminating all the wrong keys might take additional effort from the attacker's side.

Chaos-based logic gates also have unreliable functions in the functionality space which are affected by change in temperature and supply voltage. This undesired feature can be utilized to prevent SAT attack from extracting the right key. It is significantly important that the generated functions from the chaos-based logic gates remain stable under varying environmental conditions. Reliable functions should only be considered while characterizing the proposed computing system. While executing the SAT attack on the system, it is highly probable that the SAT attack will converge to an unreliable key. A similar work is found in literature where researchers have used functions that have morphing feature and the SAT tool converges to the wrong key [133]. SAT solvers need to be modified in order to accommodate the morphing/unreliable functions that is an inherent quality of dynamic circuits.

7.5.2 Security of PUFs

The large benchmark circuits can be used for authentication purposes in the PURCS system when appropriate amount of gates are replaced by chaos-based logic gates. The generation of *responses* depends on the intrinsic characteristic of the chaotic oscillator. Physical attacks by the adversary will be difficult to execute without hindering the operation of the system. It has already been shown in literature that chaos-based logic gates can mitigate power analysis based side channel attacks [105]. Common machine learning attacks have been executed on the PURCS system to test its immunity against modeling attacks.

Classification Algorithm

Classification algorithms are used to classify test data based on the training data that is built from the data sets. The training data set in supervised learning usually has a x, y format. Here, x represents an instance and y represents its class. A class is determined by the classifier based on the training data for a random instance of x. Several classifiers are used in machine learning algorithms and their performance is judged based on implementation cost, speed, accuracy and the nature of the problem. Some of the classification algorithms are described briefly.

Logistic Regression (LR): Logistic Regression is a statistical method for analyzing the data set and it helps in predicting a double branched outcome [22]. It performs the job of a binary classifier but is closely related to a regression model. It analyzes the data set and assigns a probability to each data point so that it belongs to one of the two classes. LR utilizes signum or logit function to perform the classification. LR is a powerful algorithm since it has a small feature set and is used widely due to its easy-to-implement algorithm and fast convergence.

K-Nearest Neighbor (K-NN): K-NN is a non-parametric lazy supervised algorithm and one of the fundamental tools used for classification [51]. The algorithm is called non-parametric because data generalization is not needed and it does not make any assumptions about the data. The entire training data set needs to be stored for the classification algorithm to work. This feature makes the algorithm computationally expensive. A sample is identified

based on its distance from each of the training sets. The test samples are assigned to the most common class among the K closest training samples. Different values of K can be chosen based on the application. It is also possible to choose different distance measurement methods for multi-dimensional data sets. Some of the algorithms used for calculating the distances in K-NN are Correlation, Euclidean and Cosine distance functions [174, 112, 41]. In this work, Euclidean distance has been used and it is measured as the distance between two instances $\mathbf{x^1}$ and $\mathbf{x^2}$. The equation for calculating the distance is as follows:

$$d_{euc}(\mathbf{x^1}, \mathbf{x^2}) = \sqrt{\sum_{i=1}^{L} (x_i^1 - x_i^2)^2}, \tag{7.9}$$

where x^1 and x^2 have L features and x_i^1 and x_i^2 are the i-th sample points.

Decision Tree (DT): A decision/classification tree is a simple representation for classifying data sets. Each internal node of the tree is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the output feature. Otherwise, the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes. In order to predict a response, the decisions in the tree must be followed from the root node down to a leaf node which contains the result of the classification [143].

Support Vector Machine (SVM): SVM is a very commonly used classification algorithm [77]. It performs binary classification by using two classes to categorize training sets. One category belongs to a specific class and the other category combines all the other classes. It is a non-probabilistic classifier where the test sample falls into one of the two classes. The SVM classifier works by drawing a line into the 2D space which contains the training samples and makes a clean partition between them. New samples are classified based on the side of the line it falls into. SVM is also capable of classifying multi-class data sets by utilizing kernel functions such as Gaussian radial basis function [73]. There are multiple binary classifiers in SVM such as 'onevsall', 'onevsone', 'denser random' and 'binary complete'. For a K-way multiclass problem, 'onevsall' and 'onevsone' train k and $\frac{k(k-1)}{2}$ binary classifiers, respectively.

Naive Bayes (NB): Naive Bayes is a conditional probability model based on Bayes' theorem with additional simplifying assumptions. Given a problem instance to be classified, represented by a vector $\mathbf{x} = (x_1, \ldots, x_n)$ representing some n features (independent variables), it assigns probabilities to the instance, $p(C_k|x_1, \ldots, x_n)$ for each of k possible outcomes. It labels the data as belonging to the class with the highest probability. The combination of Bayes' theorem and very simplistic conditional independence assumptions leads to the problem boiling down to calculating the value of $p(C_k) \prod_{i=1}^{n} p(x_i|C_k)$ for each class. Finally, the class with maximum value is chosen [135].

AdaBoostM2 Ensemble: AdaBoost is short for *Adaptive Boosting* and is a machine learning meta-algorithm. It can be used in combination with various types of learning algorithms in order to enhance performance. The output of other learning algorithms (weak learner) is converted into a weighted sum and this value is the final output of the classifier. AdaBoost is an adaptive algorithm because weak learners can be altered for those instances which were wrongly classified by previous classifiers. The disadvantage is that the algorithm is susceptible to outliers and noisy data sets. Although, the individual learners can be weak, if the performance of each one is better than a random guess, then the final model will be forced to converge to a strong learner [49].

Machine Learning Based Modeling Attack on the PURCS System

Attacks on PUFs depends on many factors such as skills of the attacker, cost, speed, accuracy of the tolls and instruments used, knowledge about the PUF behavior and information leakage of the devices. Due to increased speed, precision and affordability of advanced tools, side channel analysis is becoming more executable in order to decipher the secret key of the PUF. The security of PUFs lies in the assumption that the PUF cannot be modeled and that the attacker does not have access to the entire challenge-response space. Researchers are able to model PUFs using machine learning techniques [141]. If the PUF circuit is simple, attackers can construct an accurate timing model and learn the behavior of the PUF after collecting many CRPs [93]. In Arbiter PUF, delay of the two paths could be linearly added to determine the *response* bit [55]. These limitations led to the discovery of more "nonlinear" effects to make the modeling attacks non-executable. Studies have shown that previously

resilient strong PUFs were susceptible to a combination of side-channel attacks and machine learning attacks [103, 142]. Post-processing the PUF *responses* might be required to scramble the output bits in order to mitigate the attacks but it comes at a cost of reduced effectiveness in an authentication protocol. The post-processing scheme helps in reducing the accuracy of the machine learning attacks but it also increases the intrinsic PUF error resulting in reduced stability.

The immunity of the proposed hybrid system has been tested against common machine learning attacks such as Logistic Regression, AdaBoostM2 Ensemble, K-Nearest Neighbor, Decision Tree and Support Vector Machine. The machine learning toolbox from MATLAB has been used [3]. The modeling attack has been implemented on all the combinational benchmark circuits and the results are shown in Table 7.7. 5000 challenge-response pairs are collected from the PURCS system. 80% of the data has been used for training the classifiers and remaining data has been used for testing. If the PUF is robust against modeling attacks, then a new response from an unknown challenge should be close to a random guess. The testing accuracy of a new data point should be 50% ideally. The results demonstrate that the testing accuracy is close to 50% for the eight benchmark circuits which proves that the system is immune to ML attacks.

7.6 Overhead Analysis

Reconfigurable chaos-based logic gates are comparable to reconfigurable barrier based or look-up table based logic locking. Look-up tables require SRAMs and MUXs which significantly add to the area and power consumption. The size of LUTs increase if the input size increases. Each reconfigurable chaos-based logic gate performs multiple functions similar to LUTs. The advantage of using chaos-based logic gates is that they can be easily transformed into multi-input gates with a slight increase in the overhead.

Table 7.8 gives an estimation of the number of transistors with the increase in key size. When XOR/XNOR based logic locking is performed, XOR/XNOR gates are added to the system in addition to the existing circuitry whereas LUTs or chaos-based logic gates are replaced into the original design. It is approximated that XOR gates require 8 transistors

and provides a 1-bit key for each additional XOR gate inserted in the design. In order to obtain a 50 bit key, 400 additional transistors are inserted into the original design. For the purpose of comparison, we are assuming that chaos-based logic gates and LUTs are actual overheads to the design. The overhead for a single 10-input LUT which uses 2 : 1 MUXs (4 transistors), SRAM cells (6 transistors) and inverters (2 transistors) is 10256 transistors. The chaos-based logic gate has 10-bit key and uses 68 transistors which includes the 3-bit DAC (3 transistors), comparators (9 transistors in each comparator), clocking circuit (35 transistors) and the chaotic oscillator (12 transistors including the pass gates in the sample-and-hold circuit). It can seen from the table that XOR based and LUT based systems require more transistors compared to chaos-based logic gate in order to achieve the same key size.

The core of the system is an analog circuit which portrays different characteristics in different chips due to process variation. The functionality of LUTs does not vary in different chips unless different values are configured post-fabrication by the trusted designer. The chaotic oscillator provides ideal PUF characteristics which allows the same system to be used for device authentication. The proposed circuit is area and power efficient since EEPROM/RAMs are not required to store the key bits for authenticating connecting devices.

Table 7.9 shows the transistor count for the $C17$ benchmark circuit for a 30-bit key/challenge. The table shows a comparison of transistor count if both logic locking and authentication circuits are present in a system. Table 7.4 states that 55% of gates need to be replaced in $C17$ benchmark circuit which means three gates. Three NAND gates in the circuit are replaced and the key/challenge size is 30 bits. The circuit produces 1 response bit after post-processing. The results in the table shows that the proposed chaos-based system utilizes significantly less number of transistors compared to a system which incorporates traditional logic locking and PUFs in the same system.

7.7 Modes of Operation

There are two modes of operation of the proposed computing system: logic locking mode and authentication mode.

Table 7.7: Result of machine learning based modeling attacks on PURCS system using testability based replacement method [149].

Benchmark	**LR**		**KNN (k=1)**		**DT**		**SVM**		**NB**		**AdaB. Ensem.**	
	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test	Train	Test
$C17$	54.57	49.7	84.97	50.4	79.27	50.3	60.45	50.3	58.95	49.5	79.3	51
$C432$	63.01	49.5	84.75	50.67	81.57	50.3	85.11	49.63	78.18	50.33	93.03	51.07
$C880$	64.51	50.11	85.1	50.33	82.76	50.12	85.08	50.07	84.88	49.8	96.83	50.02
$C1355$	58.28	50.49	85.04	49.4	82.46	49.58	84.86	50.02	84.2	50.18	96.04	50.45
$C1908$	61.21	49.69	84.85	49.88	82.86	49.9	84.98	48.53	84.99	48.83	97.38	50.53
$C3540$	64.19	50.02	85.07	50.45	83.14	50.57	84.8	49.63	84.74	49.85	98.11	50.76
$C5315$	51.79	50	85.14	50.5	83.93	50.44	85.02	50	83.9	49.91	88.49	49.96
$C7552$	51.63	50.07	85.05	49.79	84.05	49.94	84.85	49.94	84.77	49.98	90.05	49.66

Table 7.8: An estimate of transistor count in different logic locking schemes [149].

Key Size	**Transistor count**		
	XOR based	**10-input LUT based**	**10-bit Chaos-based**
50	400	51280	340
100	800	102560	680
200	1600	205120	1360

Table 7.9: Estimation of transistor count for both PUF and logic locking in a system with 30-bit key/challenge [149].

Transistor count				
Arb + XOR	**Arb + LUT**	**Ring + XOR**	**Ring + LUT**	**Chaos-based**
518	31022	1452	31956	240

7.7.1 Logic Locking Mode

In logic locking mode, the main goal is to obfuscate the design to avoid IC counterfeiting. Fig. 7.18a shows the schematic of logic locking mode. The secret key of the system is stored in tamper-proof memory and it is the combination of the configuration key of the reconfigurable chaos-based logic gates. Each chaos-based gate has 10-bit key and the key space increases with increase in number of chaos-based gates in the system. Hamming distance has been calculated between correct and wrong outputs in order to ensure that half of the bits are flipped. After post-fabrication characterization, the correct keys are stored in a tamper-proof memory for correct operation of the system.

7.7.2 Authentication Mode

In authentication mode, the design exploits the fact that chaos-based gates are susceptible to small changes in the initial condition. Process variation in the manufacturing process changes the internal characteristics of the gates slightly which in turn changes how the inputs are mapped to outputs in the chaos-based gates. This feature ensures that the key is unique to each IC. The input is kept fixed and the *challenges* are applied based on the authentication protocol discussed in section 7.2.7. After post-processing, the *responses* from the systems demonstrates almost ideal PUF characteristics. Fig. 7.18b shows the schematic of the system in authentication mode.

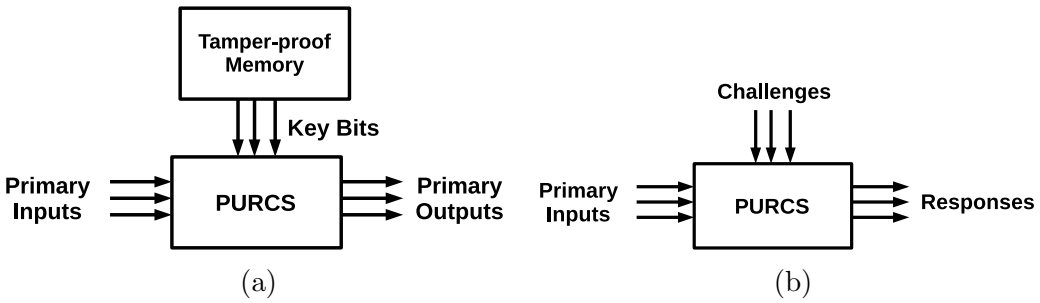


Figure 7.18: (a) Logic locking mode (b) Authentication mode.

Chapter 8

Contribution and Future Work

8.1 Original Contribution

The goal of this work is to utilize discrete-time chaos-based systems for various hardware security applications. The following points demonstrate the contributions of this dissertation.

- The three transistor chaotic map has been redesigned and simulated in 65 nm process to optimize the power and area consumption. Multi-input and multi-output reconfigurable chaos-based logic gates have been designed. It has been demonstrated that simple as well as complex functions can be generated from a single chaos-based system which aids in reducing the number of transistors used in a chip.

- The functionality space of chaos-based system using three transistor map has been increased by varying the bias voltage in each iteration. Three different bias voltages has been chosen from the chaotic region and four different threshold voltage levels has been chosen to obtain different Boolean functions. Results show that the entire functionality space increases exponentially with iteration number. The number of individual functions such as AND, OR and XOR seem to portray an upward rise with increase in design space. This huge functionality space helps to mitigate power analysis based side channel attack.

- A novel G^4NDR based discrete-time chaotic map has been developed. The map has three independent bifurcation parameters which can be biased separately. The

bifurcation diagram and Lyapunov exponent demonstrates excellent chaotic property and wider chaotic regions. The functionality space of the G^4NDR based logic gate is significantly large compared to the three transistor map based gate. This limitation is mainly due to the presence of only one bifurcation parameter in three transistor chaotic map. Reconfigurable chaos-based logic gates has been designed using the novel map circuit and different configurations are demonstrated for implementing the standard logic gates.

- A lightweight and reconfigurable pseudo-random generator has been designed using the three transistor based chaotic oscillator. The analog output of the chaotic oscillator is converted to a digital value by using a 10-bit ADC where the 10^{th} bit is used for random number generation. The output generated from a single chaotic oscillator is not random enough to pass all the NIST tests. The output from two chaotic oscillators are XORed and the final output passes all the tests. This proves that the sequence is random and can be used for cryptographic applications.

- A physically unclonable and reconfigurable computing system has been designed which can mitigate IC counterfeiting by replacing original CMOS gates in the circuit with chaos-based logic gates. The gates are replaced using testability based method in order to ensure that fewer gates needs to be replaced to achieve 50% Hamming distance. Each chaos-based logic gate has 10-bit key. The same system can be used for authentication of devices since chaos-based logic gates are vulnerable to process variation which can be utilized to generate a unique signature in each chip. The *challenges* are made up of the key and primary inputs and the *responses* are the primary outputs of the system. The system has to be large enough so that the CRP space is large in order to be used for authentication. After post-processing of the *responses*, the system demonstrates near ideal PUF characteristics. It has also been shown that the system is resistant against Boolean SAT attack and common machine learning based modeling attacks.

8.2 Future Work

Chaos-based system can be a huge asset for mitigating hardware security issues. There is a lot of scope for expanding this work for future research purposes. Some of the future research directions are:

- A new PRNG design can be explored where two different chaotic maps will be coupled. The output of one map can be used as a bifurcation parameter for the other map. This technique should help in increasing the entropy of the generated sequence. The chaotic region in the bifurcation diagram is expected to be larger than the current three transistor chaotic oscillator which makes the PRNG more tunable. The final chaotic oscillator will be more sensitive to initial conditions and have more complex behaviour.

- The PURCS system must be resistant against SAT attack for the logic locking scheme to be successful. This work has shown that the system is resistant against SAT attack using only one benchmark circuit. The SAT attack tool needs to be modified so that it can work for reconfigurable/morphing logic. The SAT tool is not equipped to handle systems which might have unreliable functionality space.

- Extensive analysis should be performed on the chaos-based systems to improve the reliability of the generated output voltages from the chaotic oscillator. In this work, some preliminary simulations have been performed to determine the stability of the functions due to temperature and supply voltage variations.

- The entire PURCS system can be fabricated and an exhaustive testing methodology can be developed in order to characterize the functionality of the entire system. It is important to know the effects of surrounding environment on the chaotic system and calculate the reliability of the functions under different conditions.

Bibliography

[1] (2010). Nist sp 800-22: Download documentation and software. https://csrc.nist.gov/projects/random-bit-generation/documentation-and-software. 82

[2] (2015). Decryption tool binaries and benchmark circuits. https://bitbucket.org/spramod/host15-logic-encryption. 131

[3] (2020). Matlab machine learning toolbox. https://www.mathworks.com/help/stats/index.html. 136

[4] Addabbo, T., Alioto, M., Fort, A., Pasini, A., Rocchi, S., and Vignoli, V. (2007). A class of maximum-period nonlinear congruential generators derived from the rényi chaotic map. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 54(4):816–828. 75

[5] Addabbo, T., Alioto, M., Fort, A., Rocchi, S., and Vignoli, V. (2006). The digital tent map: Performance analysis and optimized design as a low-complexity source of pseudorandom bits. *IEEE Transactions on Instrumentation and Measurement*, 55(5):1451–1458. 75

[6] Aihara, K., Takabe, T., and Toyoda, M. (1990). Chaotic neural networks. *Physics letters A*, 144(6-7):333–340. 27

[7] Akarvardar, K., Blalock, B., Chen, S., Cristoloveanu, S., Gentil, P., and Mojarradi, M. (2006a). Digital circuits using soi four-gate transistor. In *2006 8th International Conference on Solid-State and Integrated Circuit Technology Proceedings*, pages 1867–1869. IEEE. xv, 55

[8] Akarvardar, K., Chen, S., Blalock, B., Cristoloveanu, S., Gentil, P., and Mojarradi, M. (2005). A novel four-quadrant analog multiplier using soi four-gate transistors (g/sup 4/-fets). In *Proceedings of the 31st European Solid-State Circuits Conference, 2005. ESSCIRC 2005.*, pages 499–502. IEEE. 53, 54

[9] Akarvardar, K., Chen, S., Vandersand, J., Blalock, B., Schrimpf, R., Prothro, B., Britton, C., Cristoloveanu, S., Gentil, P., and Mojarradi, M. (2006b). Four-gate transistor voltage-controlled negative differential resistance device and related circuit applications. In *2006 IEEE international SOI Conferencee Proceedings*, pages 71–72. IEEE. xv, 54, 58, 61

[10] Alkabani, Y. and Koushanfar, F. (2007). Active hardware metering for intellectual property protection and security. In *USENIX security symposium*, pages 291–306. 88, 89, 91

[11] Alvarez, G. and Li, S. (2006). Some basic cryptographic requirements for chaos-based cryptosystems. *International journal of bifurcation and chaos*, 16(08):2129–2151. 8

[12] Amigo, J., Kocarev, L., and Szczepanski, J. (2007). Theory and practice of chaotic cryptography. *Physics Letters A*, 366(3):211–216. 8

[13] Anderson, R. and Kuhn, M. (1996). Tamper resistance-a cautionary note. In *Proceedings of the second Usenix workshop on electronic commerce*, volume 2, pages 1–11. 96

[14] Anderson, R. and Kuhn, M. (1997). Low cost attacks on tamper resistant devices. In *International Workshop on Security Protocols*, pages 125–136. Springer. 96

[15] Andrecut, M. (1998). Logistic map as a random number generator. *International Journal of Modern Physics B*, 12(09):921–930. 75

[16] Ashton, K. et al. (2009). That 'internet of things' thing. *RFID journal*, 22(7):97–114. 86

[17] Baumgarten, A., Tyagi, A., and Zambreno, J. (2010). Preventing ic piracy using reconfigurable logic barriers. *IEEE Design & Test of Computers*, 27(1):66–75. 92, 94

[18] Beirami, A., Nejati, H., and Massoud, Y. (2008). A performance metric for discrete-time chaos-based truly random number generators. In *2008 51st Midwest Symposium on Circuits and Systems*, pages 133–136. IEEE. 74

[19] Berlekamp, E., Fredricksen, E., and Proto, R. (1974). Minimum conditions for uniquely determining the generator of a linear sequence. *Utilitas Math*, 5:305–315. 74

[20] Bernstein, G. M. and Lieberman, M. A. (1990). Secure random number generation using chaotic circuits. *IEEE Transactions on Circuits and Systems*, 37(9):1157–1164. 72

[21] Bianco, M. E. and Mayhew, G. L. (1994). High speed encryption system and method. US Patent 5,365,588. 76

[22] Bishop, C. M. (2006). *Pattern recognition and machine learning*. springer. 133

[23] BLALOCK, B. J., CRISTOLOVEANU, S., DUFRENE, B. M., Allibert, F., and MOJARRADI, M. M. (2002). The multiple-gate mos-jfet transistor. *International journal of high speed electronics and systems*, 12(02):511–520. 54, 56

[24] Blum, L., Blum, M., and Shub, M. (1986). A simple unpredictable pseudo-random number generator. *SIAM Journal on computing*, 15(2):364–383. 74

[25] Borkar, S. (2005). Designing reliable systems from unreliable components: the challenges of transistor variability and degradation. *Ieee Micro*, 25(6):10–16. 100

[26] Bowman, K. A., Duvall, S. G., and Meindl, J. D. (2002). Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *IEEE Journal of solid-state circuits*, 37(2):183–190. 96

[27] Cafagna, D. and Grassi, G. (2005). Chaos-based computation via chua's circuit: Parallel computing with application to the sr flip-flop. In *International Symposium on Signals, Circuits and Systems, 2005. ISSCS 2005.*, volume 2, pages 749–752. IEEE. 12

[28] Chakraborty, R. S. and Bhunia, S. (2009). Harpoon: an obfuscation-based soc design methodology for hardware protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502. 91, 92

[29] Chang, C.-H., Zheng, Y., and Zhang, L. (2017). A retrospective and a look forward: Fifteen years of physical unclonable function advancement. *IEEE Circuits and Systems Magazine*, 17(3):32–62. 101

[30] Chen, L. (2018). A framework to enhance security of physically unclonable functions using chaotic circuits. *Physics Letters A*, 382(18):1195–1201. 104

[31] Chua, L., Komuro, M., and Matsumoto, T. (1986). The double scroll family. *IEEE transactions on circuits and systems*, 33(11):1072–1118. 8, 10

[32] Chua, L. O. (1992). *The genesis of Chua's circuit*. Electronics Research Laboratory, College of Engineering, University of 8

[33] Collet, P. and Eckmann, J.-P. (2007). *Concepts and results in chaotic dynamics: a short course*. Springer Science & Business Media. 39

[34] Cristoloveanu, S., Blalock, B., Allibert, F., Dufrene, B., and Mojarradi, M. (2002). The four-gate transistor. In *32nd European Solid-State Device Research Conference*, pages 323–326. IEEE. 57

[35] Cruz, J. M. and Chua, L. O. (1993). An ic chip of chua's circuit. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 40(10):614–625. 22

[36] de la Fraga, L. G., Torres-Pérez, E., Tlelo-Cuautle, E., and Mancillas-López, C. (2017). Hardware implementation of pseudo-random number generators based on chaotic maps. *Nonlinear Dynamics*, 90(3):1661–1670. 81

[37] Degaldo-Restituto, M., Medeiro, F., and Rodriguez-Vazquez, A. (1993). Nonlinear switched-current cmos ic for random signal generation. *Electronics Letters*, 29(25):2190–2191. 25, 27

[38] Deng, L.-Y. and Lin, D. K. (2000). Random number generation for the new century. *The American Statistician*, 54(2):145–150. 73

[39] Desnos, K., El Assad, S., Arlicot, A., Pelcat, M., and Menard, D. (2014). Efficient multicore implementation of an advanced generator of discrete chaotic sequences. In *The 9th International Conference for Internet Technology and Secured Transactions (ICITST-2014)*, pages 31–36. IEEE. 76

[40] Devadas, S., Suh, E., Paral, S., Sowell, R., Ziola, T., and Khandelwal, V. (2008). Design and implementation of puf-based" unclonable" rfid ics for anti-counterfeiting and security applications. In *2008 IEEE international conference on RFID*, pages 58–64. IEEE. 105

[41] Deza, M. M. and Deza, E. (2009). Encyclopedia of distances. In *Encyclopedia of distances*, pages 1–583. Springer. 134

[42] Dick, S. (1992). Exploring negative resistance: the lambda diode. In *Elektor Electronics*, page 54. 59

[43] Ditto, W. L., Miliotis, A., Murali, K., Sinha, S., and Spano, M. L. (2010). Chaogates: Morphing logic gates that exploit dynamical patterns. *Chaos: An Interdisciplinary Journal of Nonlinear Science*, 20(3):037107. xii, 17, 18, 19, 22

[44] Ditto, W. L., Murali, K., and Sinha, S. (2007). Chaos computing: ideas and implementations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 366(1865):653–664. 16

[45] Ditto, W. L., Murali, K., and Sinha, S. (2009). Construction of a chaotic computer chip. In *Applications of Nonlinear Dynamics*, pages 3–13. Springer. 11

[46] Dudek, P. and Juncu, V. (2003). Compact discrete-time chaos generator circuit. *Electronics Letters*, 39(20):1431–1432. xiv, 25, 28

[47] Dufrene, B., Akarvardar, K., Cristoloveanu, S., Blalock, B., Gentil, R., Kolawa, E., and Mojarradi, M. (2004). Investigation of the four-gate action in g/sup 4/-fets. *IEEE transactions on electron devices*, 51(11):1931–1935. 54

[48] Eguchi, K., Ueno, F., Tabata, T., Zhu, H., and Inoue, T. (2000). Simple design of a discrete-time chaos circuit realizing a tent map. *IEICE transactions on electronics*, 83(5):777–778. 25

[49] Eibl, G. and Pfeiffer, K.-P. (2005). Multiclass boosting for weak classifiers. *Journal of Machine Learning Research*, 6(Feb):189–210. 135

[50] Ergun, S. and Ozoguz, S. (2007). A chaos-modulated dual oscillator-based truly random number generator. In *2007 IEEE International Symposium on Circuits and Systems*, pages 2482–2485. IEEE. 74

[51] Fix, E. (1951). *Discriminatory analysis: nonparametric discrimination, consistency properties*. USAF school of Aviation Medicine. 133

[52] Force, S. A.-C. T. (2013). Winning the battle against counterfeit semiconductor products. *White Paper, Semiconductor Industry Association*, pages 4–8. 88

[53] Gammel, B. M., Gottfert, R., and Kniffler, O. (2006). An nlfsr-based stream cipher. In *2006 IEEE International Symposium on Circuits and Systems*, pages 4–pp. IEEE. 74

[54] Gassend, B., Clarke, D., Van Dijk, M., and Devadas, S. (2002). Silicon physical random functions. In *Proceedings of the 9th ACM conference on Computer and communications security*, pages 148–160. ACM. xvi, 102, 103

[55] Gassend, B., Lim, D., Clarke, D., Van Dijk, M., and Devadas, S. (2004). Identification and authentication of integrated circuits. *Concurrency and Computation: Practice and Experience*, 16(11):1077–1098. 135

[56] Gentle, J. E. (2006). *Random number generation and Monte Carlo methods*. Springer Science & Business Media. 73

[57] Gołofit, K. and Wieczorek, P. Z. (2019). Chaos-based physical unclonable functions. *Applied Sciences*, 9(5):991. 104

[58] González, J. A. and Pino, R. (1999). A random number generator based on unpredictable chaotic functions. *Computer Physics Communications*, 120(2-3):109–114. 75

[59] Guajardo, J., Kumar, S. S., Schrijen, G.-J., and Tuyls, P. (2008). Brand and ip protection with physical unclonable functions. In *2008 IEEE International Symposium on Circuits and Systems*, pages 3186–3189. IEEE. 100

[60] Guyeux, C., Wang, Q., and Bahi, J. M. (2010). Improving random number generators by chaotic iterations application in data hiding. In *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, volume 13, pages V13–643. IEEE. 76

[61] Habib, B., Kaps, J.-P., and Gaj, K. (2015). Efficient sr-latch puf. In *International Symposium on Applied Reconfigurable Computing*, pages 205–216. Springer. 101

[62] Hasan, M. S., Mahbub, I., Islam, S. K., and Rose, G. S. (2018a). A mos-jfet macromodel of soi four-gate transistors (g 4 fet) to aid innovative circuit design. In *2018 IEEE 13th Dallas Circuits and Systems Conference (DCAS)*, pages 1–4. IEEE. 56

[63] Hasan, M. S., Majumder, M. B., Shanta, A. S., Uddin, M., and Rose, G. S. (2019). A chaos-based complex micro-instruction set for mitigating instruction reverse engineering. *Journal of Hardware and Systems Security*, pages 1–17. xii, xv, 24, 40, 42, 43

[64] Hasan, M. S., Rahman, T., Islam, S. K., and Blalock, B. B. (2017). Numerical modeling and implementation in circuit simulator of soi four-gate transistor (g4fet) using multidimensional lagrange and bernstein polynomial. *Microelectronics journal*, 65:84–93. xv, 55

[65] Hasan, M. S., Shamsir, S., Shawkat, M. S. A., Garcia, F., Islam, S. K., and Rose, G. S. (2018b). Macromodel of g4fet enabling fast and reliable spice simulation for innovative circuit applications. *International Journal of High Speed Electronics and Systems*, 27(03n04):1840015. 53, 61

[66] Hazwani, S., Khan, S., Siddiqi, M. U., Al-Khateeb, K. A., Habaebi, M. H., and Shahid, Z. (2014). Randomness analysis of pseudo random noise generator using 24-bits lfsr. In *2014 5th International Conference on Intelligent Systems, Modelling and Simulation*, pages 772–774. IEEE. 74

[67] Helfmeier, C., Nedospasov, D., Tarnovsky, C., Krissler, J. S., Boit, C., and Seifert, J.-P. (2013). Breaking and entering through the silicon. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 733–744. 117

[68] Herder, C., Yu, M.-D., Koushanfar, F., and Devadas, S. (2014). Physical unclonable functions and applications: A tutorial. *Proceedings of the IEEE*, 102(8):1126–1141. 100

[69] Herrera, R., Suyama, K., Horio, Y., and Aihara, K. (1999). Ic implementation of a switched-current chaotic neuron. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 82(9):1776–1782. 7

[70] Hobson, P. and Lansbury, A. (1996). A simple electronic circuit to demonstrate bifurcation and chaos. *Physics Education*, 31(1):39. 10

[71] Hoffman, C., Gebotys, C. H., Aranha, D. F., Cortes, M. L., and Araujo, G. (2019). Circumventing uniqueness of xor arbiter pufs. In *The Euromicro Conference on Digital System Design (DSD) Euromicro Conference on Digital System Design*. 102

[72] Horio, Y., Aihara, K., and Yamamoto, O. (2003). Neuron-synapse ic chip-set for large-scale chaotic neural networks. *IEEE Transactions on Neural Networks*, 14(5):1393–1404. 7

[73] Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE transactions on Neural Networks*, 13(2):415–425. 134

[74] Hu, H., Liu, L., and Ding, N. (2013). Pseudorandom sequence generator based on the chen chaotic system. *Computer Physics Communications*, 184(3):765–768. 74

[75] Hua, Z. and Zhou, Y. (2015). Dynamic parameter-control chaotic system. *IEEE transactions on cybernetics*, 46(12):3330–3341. 75

[76] Jagasivamani, M., Gadfort, P., Sika, M., Bajura, M., and Fritze, M. (2014). Split-fabrication obfuscation: Metrics and techniques. In *2014 IEEE international symposium on hardware-oriented security and trust (HOST)*, pages 7–12. IEEE. 89

[77] Joachims, T. (1998). Text categorization with support vector machines: Learning with many relevant features. In *European conference on machine learning*, pages 137–142. Springer. 134

[78] Jose, S. (2008). Innovation is at risk as semiconductor equipment and materials. *Semiconductor Equipment and Material Industry (SEMI)*. 4

[79] Jun, B. and Kocher, P. (1999). The intel random number generator. *Cryptography Research Inc. white paper*, 27:1–8. 71

[80] Juncu, V., Rafiei-Naeini, M., and Dudek, P. (2006). Integrated circuit implementation of a compact discrete-time chaos generator. *Analog Integrated Circuits and Signal Processing*, 46(3):275–280. 25, 29, 48, 68

[81] Kahng, A. B. (2013). The itrs design technology and system drivers roadmap: Process and status. In *Proceedings of the 50th Annual Design Automation Conference*, page 34. ACM. 46

[82] Kahng, A. B., Lach, J., Mangione-Smith, W. H., Mantik, S., Markov, I. L., Potkonjak, M., Tucker, P., Wang, H., and Wolfe, G. (1998). Watermarking techniques for intellectual property protection. In *Proceedings of the 35th annual Design Automation Conference*, pages 776–781. ACM. 89

[83] Kanso, A. and Smaoui, N. (2009). Logistic chaotic maps for binary numbers generations. *Chaos, Solitons & Fractals*, 40(5):2557–2568. 75

[84] Kennedy, M. P. (1993). Three steps to chaos. ii. a chua's circuit primer. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 40(10):657–674. 10

[85] Kia, B., Lindner, J. F., and Ditto, W. L. (2016). A simple nonlinear circuit contains an infinite number of functions. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 63(10):944–948. xiv, 28, 46, 49, 50

[86] Kia, B., Mobley, K., and Ditto, W. L. (2017). An integrated circuit design for a dynamics-based reconfigurable logic block. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 64(6):715–719. xiv, 36, 38, 53

[87] KPMG (2005). Managing the risks of counterfeiting in the information technology industry. 4

[88] Kumar, S. S., Guajardo, J., Maes, R., Schrijen, G.-J., and Tuyls, P. (2008). The butterfly puf protecting ip on every fpga. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, pages 67–70. IEEE. 101

[89] Layman, P. A., Chaudhry, S., Norman, J. G., and Thomson, J. R. (2004). Electronic fingerprinting of semiconductor integrated circuits. US Patent 6,738,294. 101

[90] Li, C.-Y., Chen, J.-S., and Chang, T.-Y. (2006). A chaos-based pseudo random number generator using timing-based reseeding method. In *2006 IEEE International Symposium on Circuits and Systems*, pages 4–pp. IEEE. 75

[91] Li, C.-Y., Chen, Y.-H., Chang, T.-Y., Deng, L.-Y., and To, K. (2011). Period extension and randomness enhancement using high-throughput reseeding-mixing prng. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 20(2):385–389. 85

[92] Li, C.-Y., Chou, H.-P., Deng, L.-Y., Shiau, J.-J. H., and Lu, H. H.-S. (2012). Non-linear pseudo-random number generators via coupling dx generators with the logistic map. In *Anti-counterfeiting, Security, and Identification*, pages 1–5. IEEE. 75

[93] Lim, D., Lee, J. W., Gassend, B., Suh, G. E., Van Dijk, M., and Devadas, S. (2005). Extracting secret keys from integrated circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(10):1200–1205. 135

[94] Liu, A. and Ning, P. (2008). Tinyecc: A configurable library for elliptic curve cryptography in wireless sensor networks. In *Proceedings of the 7th international conference on Information processing in sensor networks*, pages 245–256. IEEE Computer Society. 97

[95] Liu, B. and Wang, B. (2014). Embedded reconfigurable logic for asic design obfuscation against supply chain attacks. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE. 95

[96] Liu, B. and Wang, B. (2015). Reconfiguration-based vlsi design for security. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 5(1):98–108. 94

[97] Liu, L., Huang, H., and Hu, S. (2017). Lorenz chaotic system-based carbon nanotube physical unclonable functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(7):1408–1421. 104

[98] Liu, Z. and Peng, D. (2006). True random number generator in rfid systems against traceability. In *CCNC 2006. 2006 3rd IEEE Consumer Communications and Networking Conference, 2006.*, volume 1, pages 620–624. IEEE. 72

[99] Lofstrom, K., Daasch, W. R., and Taylor, D. (2000). Ic identification circuit using device mismatch. In *2000 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No. 00CH37056)*, pages 372–373. IEEE. 99

[100] Lorenz, E. N. (1963). Deterministic nonperiodic flow. *Journal of the atmospheric sciences*, 20(2):130–141. 7

[101] Ma, J., Guo, Y., Li, L., Wu, Y., Cheng, X., and Zeng, X. (2011). A low power 10-bit 100-ms/s sar adc in 65nm cmos. In *2011 9th IEEE International Conference on ASIC*, pages 484–487. IEEE. 84

[102] Mack, C. A. (2011). Fifty years of moore's law. *IEEE Transactions on semiconductor manufacturing*, 24(2):202–207. 45

[103] Mahmoud, A., Rührmair, U., Majzoobi, M., and Koushanfar, F. (2013). Combined modeling and side channel attacks on strong pufs. *IACR Cryptology ePrint Archive*, 2013:632. 136

[104] Majumder, M. B., Hasan, M. S., Shanta, A., Uddin, M., and Rose, G. (2019). Design for eliminating operation specific power signatures from digital logic. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pages 111–116. ACM. 33, 126

[105] Majumder, M. B., Hasan, M. S., Uddin, M., and Rose, G. S. (2018). Chaos computing for mitigating side channel attack. In *2018 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 143–146. IEEE. 33, 69, 126, 133

[106] Majzoobi, M., Koushanfar, F., and Potkonjak, M. (2008). Lightweight secure pufs. In *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 670–673. IEEE Press. 102

[107] Masoodi, F., Alam, S., and Bokhari, M. (2012). An analysis of linear feedback shift registers in stream ciphers. *International Journal of Computer Applications*, 46(17):46–49. 75

[108] Matsumoto, T. (1984). A chaotic attractor from chua's circuit. *IEEE Transactions on Circuits and Systems*, 31(12):1055–1058. xiv, 8, 9

[109] Matsumoto, T., Chua, L., and Komuro, M. (1985). The double scroll. *IEEE Transactions on Circuits and Systems*, 32(8):797–818. 10

[110] Miura, N., Takahashi, M., Nagatomo, K., and Nagata, M. (2017). Chaos, deterministic non-periodic flow, for chip-package-board interactive puf. In *2017 IEEE Asian solid-state circuits conference (A-SSCC)*, pages 25–28. IEEE. 104

[111] Moddemeijer, R. (1989). On estimation of entropy and mutual information of continuous distributions. *Signal processing*, 16(3):233–248. 72

[112] Msgna, M., Markantonakis, K., and Mayes, K. (2014). Precise instruction-level side channel profiling of embedded processors. In *International conference on information security practice and experience*, pages 129–143. Springer. 134

[113] Munakata, T., Sinha, S., and Ditto, W. L. (2002). Chaos computing: implementation of fundamental logical gates by chaotic elements. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 49(11):1629–1633. 12

[114] Murali, K. and Sinha, S. (2003). Experimental realization of chaos control by thresholding. *Physical Review E*, 68(1):016210. 12

[115] Murali, K., Sinha, S., and Ditto, W. L. (2003a). Implementation of nor gate by a chaotic chua's circuit. *International Journal of Bifurcation and Chaos*, 13(09):2669–2672. xiv, 12, 13, 16

[116] Murali, K., Sinha, S., and Ditto, W. L. (2003b). Realization of the fundamental nor gate using a chaotic circuit. *Physical Review E*, 68(1):016205. 13

[117] Murali, K., Sinha, S., and Ditto, W. L. (2005). Construction of a reconfigurable dynamic logic cell. *Pramana*, 64(3):433–441. 13

[118] Murillo-Escobar, M., Cruz-Hernández, C., Abundiz-Pérez, F., and López-Gutiérrez, R. M. (2015). A robust embedded biometric authentication system based on fingerprint and chaotic encryption. *Expert Systems with Applications*, 42(21):8198–8211. 74

[119] Nejati, H., Beirami, A., and Ali, W. H. (2012). Discrete-time chaotic-map truly random number generators: design, implementation, and variability analysis of the zigzag map. *Analog Integrated Circuits and Signal Processing*, 73(1):363–374. 72

[120] Oliveira, L. B., Aranha, D. F., Gouvêa, C. P., Scott, M., Câmara, D. F., López, J., and Dahab, R. (2011). Tinypbc: Pairings for authenticated identity-based non-interactive key distribution in sensor networks. *Computer communications*, 34(3):485–493. 97

[121] Özkaynak, F. (2014). Cryptographically secure random number generator with chaotic additional input. *Nonlinear Dynamics*, 78(3):2015–2020. 74

[122] Pappu, R., Recht, B., Taylor, J., and Gershenfeld, N. (2002). Physical one-way functions. *Science*, 297(5589):2026–2030. 100, 102

[123] Paral, Z. and Devadas, S. (2011). Reliable and efficient puf-based key generation using pattern matching. In *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 128–133. IEEE. 102

[124] Pareschi, F., Setti, G., and Rovatti, R. (2010). Implementation and testing of high-speed cmos true random number generators based on chaotic systems. *IEEE transactions on circuits and systems I: regular papers*, 57(12):3124–3137. 85

[125] Patidar, V., Sud, K. K., and Pareek, N. K. (2009). A pseudo random bit generator based on chaotic logistic map and its statistical testing. *Informatica*, 33(4). 76

[126] Pellicer-Lostao, C. and López-Ruiz, R. (2008). Pseudo-random bit generation based on 2d chaotic maps of logistic type and its applications in chaotic cryptography. In *International Conference on Computational Science and Its Applications*, pages 784–796. Springer. 75

[127] Plaza, S. M. and Markov, I. L. (2015). Solving the third-shift problem in ic piracy with test-aware logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(6):961–971. 89, 94

[128] Rajendran, J., Pino, Y., Sinanoglu, O., and Karri, R. (2012a). Logic encryption: A fault analysis perspective. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 953–958. IEEE. 90

[129] Rajendran, J., Pino, Y., Sinanoglu, O., and Karri, R. (2012b). Security analysis of logic obfuscation. In *Proceedings of the 49th Annual Design Automation Conference*, pages 83–89. ACM. 126

[130] Rajendran, J., Rose, G. S., Karri, R., and Potkonjak, M. (2012c). Nano-ppuf: A memristor-based security primitive. In *2012 IEEE Computer Society Annual Symposium on VLSI*, pages 84–87. IEEE. 100

[131] Rajendran, J., Sam, M., Sinanoglu, O., and Karri, R. (2013a). Security analysis of integrated circuit camouflaging. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 709–720. ACM. 89, 94

[132] Rajendran, J., Zhang, H., Zhang, C., Rose, G. S., Pino, Y., Sinanoglu, O., and Karri, R. (2013b). Fault analysis-based logic encryption. *IEEE Transactions on computers*, 64(2):410–424. xvi, 89, 91, 92, 93, 94

[133] Rangarajan, N., Patnaik, S., Knechtel, J., Karri, R., Sinanoglu, O., and Rakheja, S. (2018). Opening the doors to dynamic camouflaging: Harnessing the power of polymorphic devices. *arXiv preprint arXiv:1811.06012*. 132

[134] Rankl, W. and Effing, W. (2004). *Smart card handbook*. John Wiley & Sons. 72

[135] Rish, I. et al. (2001). An empirical study of the naive bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, volume 3, pages 41–46. 135

[136] Rizk, M. R., Nasser, A.-M. A., El-Badawy, E.-S. A., and Abou-Bakr, E. (2012a). A new approach for obtaining all logic gates using chua's circuit: Advantages and disadvantages. In *2012 International Conference on Computer and Communication Engineering (ICCCE)*, pages 730–733. IEEE. 12

[137] Rizk, M. R., Nasser, A.-M. A., El-Badawy, E.-S. A., and Abou-Bakr, E. (2012b). A new approach for obtaining all logic gates using chua's circuit with fixed input/output levels. In *2012 Japan-Egypt Conference on Electronics, Communications and Computers*, pages 12–17. IEEE. 13

[138] Romero, M. E., Martins, E. M., dos Santos, R. R., and Gonzalez, M. E. D. (2013). Universal set of cmos gates for the synthesis of multiple valued logic digital circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61(3):736–749. 46

[139] Rose, G. S. (2014). A chaos-based arithmetic logic unit and implications for obfuscation. In *2014 IEEE Computer Society Annual Symposium on VLSI*, pages 54–58. IEEE. xiv, 34, 35

[140] Roy, J. A., Koushanfar, F., and Markov, I. L. (2008). Epic: Ending piracy of integrated circuits. In *Proceedings of the conference on Design, automation and test in Europe*, pages 1069–1074. ACM. 93

[141] Rührmair, U., Sehnke, F., Sölter, J., Dror, G., Devadas, S., and Schmidhuber, J. (2010). Modeling attacks on physical unclonable functions. In *Proceedings of the 17th ACM conference on Computer and communications security*, pages 237–249. ACM. 135

[142] Rührmair, U., Xu, X., Sölter, J., Mahmoud, A., Koushanfar, F., and Burleson, W. (2013). Power and timing side channels for pufs and their efficient exploitation. *IACR Cryptology ePrint Archive*, 2013:851. 136

[143] Safavian, S. R. and Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674. 134

[144] Sahoo, D. P., Mukhopadhyay, D., and Chakraborty, R. S. (2013). Design of low area-overhead ring oscillator puf with large challenge space. In *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–6. IEEE. 102

[145] Senouci, A., Benkhaddra, I., Boukabou, A., Bouridane, A., and Ouslimani, A. (2014). Implementation and evaluation of a new unified hyperchaos-based prng. In *2014 26th International Conference on Microelectronics (ICM)*, pages 1–4. IEEE. 76

[146] Shamsi, K., Li, M., Meade, T., Zhao, Z., Pan, D. Z., and Jin, Y. (2017). Cyclic obfuscation for creating sat-unresolvable circuits. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 173–178. 131

[147] Shannon, C. E. (1949). Communication theory of secrecy systems. *Bell system technical journal*, 28(4):656–715. 75

[148] Shanta, A. S., Hasan, M. S., Majumder, M. B., and Rose, G. S. (2019). Design of a lightweight reconfigurable prng using three transistor chaotic map. In *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 586–589. IEEE. xii, xiii, xiv, xvi, 24, 34, 71, 78, 85

[149] Shanta, A. S., Majumder, M. B., Hasan, M. S., and Rose, G. S. (2020). Physically unclonable and reconfigurable computing system (purcs) for hardware security applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*. xiii, xvi, xvii, 87, 109, 111, 114, 120, 122, 124, 130, 138

[150] Shanta, A. S., Majumder, M. B., Hasan, M. S., Uddin, M., and Rose, G. S. (2018). Design of a reconfigurable chaos gate with enhanced functionality space in 65nm cmos. In *2018 IEEE 61st International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 1016–1019. IEEE. xii, xiv, xv, 24, 25, 26, 28, 45, 47, 48, 51, 68

[151] Sharaf, M., Mansour, H. A., Zayed, H. H., and Shore, M. (2005). A complex linear feedback shift register design for the a5 keystream generator. In *Proceedings of the Twenty-Second National Radio Science Conference, 2005. NRSC 2005.*, pages 395–402. IEEE. xvi, 73, 74

[152] Shen, Y. and Zhou, H. (2017). Double dip: Re-evaluating security of logic encryption algorithms. In *Proceedings of the on Great Lakes Symposium on VLSI 2017*, pages 179–184. 131

[153] Simons, P., van der Sluis, E., and van der Leest, V. (2012). Buskeeper pufs, a promising alternative to d flip-flop pufs. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 7–12. IEEE. 101

[154] Sinha, S. and Ditto, W. L. (1998). Dynamics based computation. *physical review Letters*, 81(10):2156. 12

[155] Sinha, S. and Ditto, W. L. (1999). Computing with distributed chaos. *Physical Review E*, 60(1):363. 12

[156] Škorić, B., Tuyls, P., and Ophey, W. (2005). Robust key extraction from physical uncloneable functions. In *International Conference on Applied Cryptography and Network Security*, pages 407–422. Springer. 100

[157] Skorobogatov, S. P. (2005). Semi-invasive attacks: a new approach to hardware security analysis. 99

[158] Stojanovski, T. and Kocarev, L. (2001). Chaos-based random number generators-part i: analysis [cryptography]. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48(3):281–288. 75

[159] Stojanovski, T., Pihl, J., and Kocarev, L. (2001). Chaos-based random number generators. part ii: practical realization. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 48(3):382–385. 75

[160] Subramanyan, P., Ray, S., and Malik, S. (2015). Evaluating the security of logic encryption algorithms. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143. IEEE. xvi, 127, 128

[161] Suh, G. E. and Devadas, S. (2007). Physical unclonable functions for device authentication and secret key generation. In *2007 44th ACM/IEEE Design Automation Conference*, pages 9–14. IEEE. xvi, 98, 99, 103

[162] Suneel, M. (2009). Cryptographic pseudo-random sequences from the chaotic hénon map. *Sadhana*, 34(5):689–701. 75

[163] Takagi, H. and Kano, G. (1975). Complementary jfet negative-resistance devices. *IEEE Journal of Solid-State Circuits*, 10(6):509–515. xv, 55, 59

[164] Tanaka, H., Sato, S., and Nakajima, K. (2000). Integrated circuits of map chaos generators. *Analog integrated circuits and signal processing*, 25(3):329–335. 25

[165] Torrance, R. and James, D. (2009). The state-of-the-art in ic reverse engineering. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 363–381. Springer. 99, 128

[166] Torrance, R. and James, D. (2011). The state-of-the-art in semiconductor reverse engineering. In *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 333–338. IEEE. 88

[167] Tuncer, T. (2015). Implementation of duplicate trng on fpga by using two different randomness source. *Elektronika ir Elektrotechnika*, 21(4):35–39. 73

[168] Tuncer, T. (2016). The implementation of chaos-based puf designs in field programmable gate array. *Nonlinear Dynamics*, 86(2):975–986. 104

[169] Tuyls, P., Škorić, B., Stallinga, S., Akkermans, A. H., and Ophey, W. (2005). Information-theoretic security analysis of physical uncloneable functions. In *International Conference on Financial Cryptography and Data Security*, pages 141–155. Springer. 100

[170] Valtierra, J. L., Tlelo-Cuautle, E., and Rodríguez-Vázquez, Á. (2017). A switched-capacitor skew-tent map implementation for random number generation. *International Journal of Circuit Theory and Applications*, 45(2):305–315. 72

[171] Van der Leest, V., Schrijen, G.-J., Handschuh, H., and Tuyls, P. (2010). Hardware intrinsic security from d flip-flops. In *Proceedings of the fifth ACM workshop on Scalable trusted computing*, pages 53–62. ACM. 101

[172] Vasyltsov, I., Hambardzumyan, E., Kim, Y.-S., and Karpinskyy, B. (2008). Fast digital trng based on metastable ring oscillator. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 164–180. Springer. 74

[173] Wang, A., Chen, M., Wang, Z., and Wang, X. (2013). Fault rate analysis: breaking masked aes hardware implementations efficiently. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 60(8):517–521. 99

[174] Wang, L., Zhang, Y., and Feng, J. (2005). On the euclidean distance of images. *IEEE transactions on pattern analysis and machine intelligence*, 27(8):1334–1339. 134

[175] Wang, X., Jia, X., Zhou, Q., Cai, Y., Yang, J., Gao, M., and Qu, G. (2016a). Secure and low-overhead circuit obfuscation technique with multiplexers. In *Proceedings of the 26th edition on Great Lakes Symposium on VLSI*, pages 133–136. ACM. 94

[176] Wang, Y., Liu, Z., Ma, J., and He, H. (2016b). A pseudorandom number generator based on piecewise logistic map. *Nonlinear Dynamics*, 83(4):2373–2391. 79

[177] Wendt, J. B. and Potkonjak, M. (2014). Hardware obfuscation using puf-based logic. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 270–271. IEEE. 94

[178] Xie, Y. and Srivastava, A. (2018). Anti-sat: Mitigating sat attack on logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(2):199–207. 131

[179] Yalçin, M. E. (2007). Increasing the entropy of a random number generator using n-scroll chaotic attractors. *International Journal of Bifurcation and Chaos*, 17(12):4471–4479. 72

[180] Yalcin, M. E., Suykens, J. A., and Vandewalle, J. (2004). True random bit generation from a double-scroll attractor. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(7):1395–1404. 74

[181] Yang, H.-T., Huang, J.-R., and Chang, T.-Y. (2004). A chaos-based fully digital 120 mhz pseudo random number generator. In *The 2004 IEEE Asia-Pacific Conference on Circuits and Systems, 2004. Proceedings.*, volume 1, pages 357–360. IEEE. 75, 76, 85

[182] Yasin, M., Mazumdar, B., Rajendran, J. J., and Sinanoglu, O. (2016). Sarlock: Sat attack resistant logic locking. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 236–241. IEEE. xiii, xvi, 127, 128, 129, 130, 131

[183] Yasin, M., Rajendran, J. J., Sinanoglu, O., and Karri, R. (2015). On improving the security of logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(9):1411–1424. 93, 94

[184] Yasin, M. and Sinanoglu, O. (2015). Transforming between logic locking and ic camouflaging. In *2015 10th International Design & Test Symposium (IDT)*, pages 1–4. IEEE. 88

[185] Yeh, A. (2012). Trends in the global ic design service market. *DIGITIMES research*. 3

[186] Zuchowski, P. S., Habitz, P. A., Hayes, J. D., and Oppold, J. H. (2004). Process and environmental variation impacts on asic timing. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 336–342. IEEE. 98

Appendices

A Random Number Generator

A.1 SKILL Code for Modeling the Chaotic Oscillator

The value of V_c has been partitioned by 2 mV in the range of 0 V to 1.2 V. Two SKILL files are required to complete the modeling due to the limitation on the size of .csv file.

File 1

```
declare(Vc[241])
;For two new VCs, exploring chaotic behavior, VC=0.625, 0.65
declare(in[1201])
declare(out[1201])

for(p 0 length(in)-1
       ;in[p]=p*0.1;
  in[p]=p*0.001;
       ;fprintf(myport2 "%d," RN[p])
  )
for(p 0 length(Vc)-1
       ;in[p]=p*0.1;
  Vc[p]=p*0.0025;

       ;fprintf(myport2 "%d," RN[p])
  )
myport1=outfile("/home/ashanta1/seneca/PRNG new analysis/
   Lookup_table_generation_skill/lookup_mwscas_ckt_trans_first_geo6_1.csv")

for(i 0 length(Vc)-1
simulator( 'spectre )
design(
   "/data1/aysha/simulation/map_ckt_with_trans_gate_geo6/spectre/schematic
/netlist/netlist")
```

```
resultsDir( "/data1/aysha/simulation/map_ckt_with_
trans_gate_geo6/spectre/schematic" )
modelFile(
   '("/data1/IBM_PDK_MOD/cmos10lpe/V1.5.0.0RF/Spectre/models/design.scs"
      "")
)
stimulusFile( ?xlate nil
   "/data1/aysha/simulation/map_ckt_with_trans_
   gate_geo6/spectre/schematic/netlist/_graphical_stimuli.scs")
;analysis('tran ?stop "20n" ?errpreset "moderate" )
analysis('dc ?param "Vin" ?start "0" ?stop "1.2"
     ?step "1m" )
desVar( "Vin" 0 )
desVar( "Vc" Vc[i] )
envOption(
  'firstRun t
  'analysisOrder list("dc" "tran")
)
temp( 27 )
run()
for(j 0 length(in)-1
out[j]=value(VDC("/out") in[j] )
;print(xinit)
fprintf(myport1 "%e," Vc[i])
fprintf(myport1 "%e," in[j])
fprintf(myport1 "%e," out[j])
fprintf(myport1 "\n")
;plot(getData("/net10") )
)
)
close(myport1)
```

File 2

```
declare(Vc[240])

;For two new VCs, exploring chaotic behavior, VC=0.625, 0.65
declare(in[1201])
declare(out[1201])

for(p 0 length(in)-1
       ;in[p]=p*0.1;
  in[p]=p*0.001;
       ;fprintf(myport2 "%d," RN[p])
  )


for(p 0 length(Vc)-1
       ;in[p]=p*0.1;
  Vc[p]=p*0.0025+0.6025;

       ;fprintf(myport2 "%d," RN[p])
  )

myport1=outfile("/home/ashanta1/seneca/PRNG new analysis/
   Lookup_table_generation_skill/lookup_mwscas_ckt_trans_first_geo6_2.csv")

for(i 0 length(Vc)-1
simulator( 'spectre )
design( "/data1/aysha/simulation/map_ckt_
with_trans_gate_geo6/spectre/schematic/netlist/netlist")

resultsDir( "/data1/aysha/simulation/map_ckt_
with_trans_gate_geo6/spectre/schematic" )
```

```
modelFile(
   '("/data1/IBM_PDK_MOD/cmos10lpe/V1.5.0.0RF/Spectre/models/design.scs"
      "")
)
stimulusFile( ?xlate nil
   "/data1/aysha/simulation/map_ckt_with_trans
   _gate_geo6/spectre/schematic/netlist/_graphical_stimuli.scs")
;analysis('tran ?stop "20n" ?errpreset "moderate" )
analysis('dc ?param "Vin" ?start "0" ?stop "1.2"
     ?step "1m" )
desVar( "Vin" 0 )
desVar( "Vc" Vc[i] )
envOption(
  'firstRun t
  'analysisOrder list("dc" "tran")
)

temp( 27 )
run()

for(j 0 length(in)-1
out[j]=value(VDC("/out") in[j] )
;print(xinit)
fprintf(myport1 "%e," Vc[i])
fprintf(myport1 "%e," in[j])
fprintf(myport1 "%e," out[j])
fprintf(myport1 "\n")
)
)
close(myport1)
```

A.2 Matlab Code for Plotting the Bifurcation Diagram

```
M1=csvread('lookup_mwscas_ckt_trans_first_geo6_1.csv');
M2=csvread('lookup_mwscas_ckt_trans_first_geo6_2.csv');

count=0;
res=1201;
start=1000;
iter=4000;

Vc1=0:2.5e-3:0.6;
for i=1:length(Vc1)

   Vc_ta_1{i}=M1((i-1)*res+1:(i-1)*res+res,2:3);
   %Vc_ta_4{i} =round(Vc_ta_4{i},3);
end

Vc2=602.5e-3:2.5e-3:1.2;
for i=1:length(Vc2)

   Vc_ta_2{i}=M2((i-1)*res+1:(i-1)*res+res,2:3);
   %Vc_ta_4{i} =round(Vc_ta_4{i},3);
end

for i=1:length(Vc1)
ta=Vc_ta_1{i};
%vc=0.375;
Vin=ta(:,1);
Vout=ta(:,2);

in=0.6;
for j=1:iter
```

```
    %vout(j)=Vout(in_ind);
    vout(j)= interp1(Vin,Vout,in);
    %[val in_ind]=min(abs(vout(j)-Vin));
    in=vout(j);
end
vout=vout(start:end);
bi_fur{i}=vout;

    %figure
    %plot(vout,'ro')
    vc_plot=ones(1,length(vout))*Vc1(i);
    plot(vc_plot,vout,'ro','markersize',2)
    hold on
end

for i=1:length(Vc2)
ta=Vc_ta_2{i};
%vc=0.375;
Vin=ta(:,1);
Vout=ta(:,2);

in=0.6;
for j=1:iter

    %vout(j)=Vout(in_ind);
    vout(j)= interp1(Vin,Vout,in);
    %[val in_ind]=min(abs(vout(j)-Vin));
    in=vout(j);
end
vout=vout(start:end);
bi_fur{i}=vout;
```

```
    %figure
    %plot(vout,'ro')
    vc_plot=ones(1,length(vout))*Vc2(i);
    plot(vc_plot,vout,'ro','markersize',2)
    hold on
end

xlabel('Bifurcation Parameter, V_{c} (V)',
    'fontweight','bold','fontsize',55)
ylabel('Output, V_{out} (V)', 'fontweight','bold','fontsize',55)

set(gca,'FontWeight','Bold','FontSize',45, 'LineWidth', 3)
ylim([0 1.2])
```

A.3 Matlab Code for Plotting the Lyapunov Exponent

```
%set up the number of data and control bits
n_data=2;
n_control=1;
n_bit=n_data+n_control;
iter=3000;
trun=300;

in=0.2;
%in2=in+0.01;
delta=0.001;
%csv files with transmission gate
M1=csvread('lookup_mwscas_ckt_trans_first_1.csv');
M2=csvread('lookup_mwscas_ckt_trans_first_2.csv');
```

```
%finding range
vout1=M1(:,3);
vout2=M2(:,3);
[val1 ind1]=min(vout1);
[val2 ind2]=min(vout2);
if val1<val2
   ind=ind1;
   Vcmin=M1(ind,1);
   vin_min=M1(ind,2);
   vout_min=val1;
else
   ind=ind2;
   Vcmin=M2(ind,1);
   vin_min=M2(ind,2);
   vout_min=val2;
end

Vc1=0:2.5e-3:600e-3;
for i=1:length(Vc1)

   Vc_ta_1{i}=M1((i-1)*1201+1:(i-1)*1201+1201,2:3);

end

Vc2=602.5e-3:2.5e-3:1.2;
for i=1:length(Vc2)

   Vc_ta_2{i}=M2((i-1)*1201+1:(i-1)*1201+1201,2:3);

end
```

```
%%%%%%%%%%%% first table
vo_a=[];
Vc_plot=[];
LE_plot=[];
for i=1:length(Vc_ta_1)

   ta=Vc_ta_1{i};
   Vc_plot=[Vc_plot;Vc1(i)];
   Vin=ta(:,1);
   Vout=ta(:,2);

   vi=in;
   %vi2=in2;
   sum=0;
   for ii=1:iter
      vo= interp1(Vin,Vout,vi);
      vo2= interp1(Vin,Vout,vi+delta);
      if ii>trun
      sum=sum+log(abs((vo2-vo)/(delta)));
      end
      vi=vo;
      %vi2=vo2;
   end
   if vo2~=vo
      %LE=log(abs((vo2-vo)/(in2-in)))/iter;
      LE=sum/(iter-trun);
   else
      LE=-5;
   end
   %LE=log(abs((vo2-vo)/(in2-in)))/iter;
   LE_plot=[LE_plot;LE];
   %vo_a=[vo_a;vo];
```

```
end

%%%%%%%%%%%%% second table
for i=1:length(Vc_ta_2)
   ta=Vc_ta_2{i};
   Vc_plot=[Vc_plot;Vc2(i)];
   Vin=ta(:,1);
   %[val ind]=min(abs(Vin-0.1932));
   Vout=ta(:,2);

   vi=in;
   sum=0;
   for ii=1:iter
      vo= interp1(Vin,Vout,vi);
      vo2= interp1(Vin,Vout,vi+delta);
      if ii>trun
      sum=sum+log(abs((vo2-vo)/(delta)));
      end
      vi=vo;
      %vi2=vo2;
   end
   if vo2~=vo
      %LE=log(abs((vo2-vo)/(in2-in)))/iter;
      LE=sum/(iter-trun);
   else
      LE=-5;
   end
   LE_plot=[LE_plot;LE];
end

figure
ind=find(LE_plot>0);
```

```
chaotic_points=Vc_plot(ind);
stem(chaotic_points)

figure
plot(Vc_plot,LE_plot,'ro','markersize',1.2, 'Linewidth',6)
hold on
plot(Vc_plot,zeros(length(Vc_plot),1),'Linewidth',4.5)

xlabel('V_{c}(V)', 'fontweight','bold','fontsize',60)
ylabel('Lyapunov Exponent, \lambda', 'fontweight','bold','fontsize',55)
set(gca,'FontWeight','Bold','FontSize',40,'LineWidth', 6)
ylim([-5 1])
```

A.4 Matlab Code for Generating the Random Numbers

```
M1=csvread('lookup_mwscas_ckt_trans_first_1.csv');
M2=csvread('lookup_mwscas_ckt_trans_first_2.csv');

count=0;
res=1201;
seq_len = 2*10^6;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
start=10000; %start is the number of extra sequences generated
iter=seq_len+start; %total number of binary values generated is iter +
   start
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
seed= [0.35:0.008:1.2]; %initial seed value, Vin
seed = seed(1:100);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
num_Vc=1; %no. of different Vc(s) to be applied to the map circuits
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
num_seq = length(seed);
bin_seq_tot_fh = zeros(1, num_seq*seq_len/2);
bin_seq_tot_alter = zeros(1, num_seq*seq_len/2);
bin_seq_tot_xor = zeros(1, num_seq*seq_len/2);
bin_seq_tot = zeros(1, num_seq*seq_len);
anal_seq_tot = zeros(1, num_seq*seq_len);
starting1 = 1;
ending = seq_len;
starting_half1 = 1;
ending_half1 = seq_len/2;
starting_half2 = 1;
ending_half2 = seq_len/2;
for p = 1:num_seq
   initial = seed(p)

%%%%%%%%%%%%%%%%%% Single Vc is considered
if num_Vc==1
   Vc_app=0.575; %applied Vc

   %ADC set up
   vmin=0.31;
   vmax=1.195;
   nb=10; %no. of bits in ADC
   nlev=2^nb;
   resol=(vmax-vmin)/nlev;

   Vc1=0:2.5e-3:0.6;
   for i=1:length(Vc1)
      Vc_ta_1{i}=M1((i-1)*res+1:(i-1)*res+res,2:3);
   end

   Vc2=602.5e-3:2.5e-3:1.2;
```

```
    for i=1:length(Vc2)
        Vc_ta_2{i}=M2((i-1)*res+1:(i-1)*res+res,2:3);
    end
    Vc_ta=[Vc_ta_1 Vc_ta_2];
    Vc_a=[Vc1 Vc2];
    [val, ind]=min(abs(Vc_a-Vc_app));
    ta=Vc_ta{ind};
    Vin=ta(:,1);
    Vout=ta(:,2);


    in=initial;
    for j=1:iter
        vout(j)= interp1(Vin,Vout,in);
        temp=ADC(resol,vout(j),nb,vmin);
        vout_bin(j)=temp(end);
        in=vout(j);
    end
    anal_seq=vout(start+1:end); %analog values
    bin_seq=vout_bin(start+1:end); %binary sequence
end

%%%%%%%%%%%%%%% Double Vc is considered
if num_Vc==2
    Vc_app_1=0.575;
    Vc_app_2=0.615;

    %ADC set up
    vmin=0.29; %minimum of the two values
    vmax=1.2; %maximum of the two values
    nb=10; %ADC bits
    nlev=2^nb;
```

```
resol=(vmax-vmin)/nlev;

Vc1=0:2.5e-3:0.6;
for i=1:length(Vc1)
   Vc_ta_1{i}=M1((i-1)*res+1:(i-1)*res+res,2:3);
end

Vc2=602.5e-3:2.5e-3:1.2;
for i=1:length(Vc2)
   Vc_ta_2{i}=M2((i-1)*res+1:(i-1)*res+res,2:3);
end
Vc_ta=[Vc_ta_1 Vc_ta_2];
Vc_a=[Vc1 Vc2];


[val, ind_1]=min(abs(Vc_a-Vc_app_1));
ta_1=Vc_ta{ind_1};
Vin1=ta_1(:,1);
Vout1=ta_1(:,2);

[val, ind_2]=min(abs(Vc_a-Vc_app_2));
ta_2=Vc_ta{ind_2};
Vin2=ta_2(:,1);
Vout2=ta_2(:,2);

in= initial;
for j=1:iter
   if mod(j,2)==1
      vout(j)= interp1(Vin1,Vout1,in);
   else
      vout(j)= interp1(Vin2,Vout2,in);
   end
```

```
        temp=ADC(resol,vout(j),nb,vmin);
        vout_bin(j)=temp(end);
        in=vout(j);
    end

    anal_seq=vout(start+1:end); %analog values
    bin_seq=vout_bin(start+1:end); %binary sequence
end

bin_seq_tot(starting1:ending) = bin_seq;
anal_seq_tot(starting1:ending) = anal_seq;
starting1 = starting1 + seq_len;
ending = ending + seq_len;

bin_seq_alter = bin_seq(2:2:seq_len);
bin_seq_tot_alter(starting_half1:ending_half1) = bin_seq_alter;
starting_half1 = starting_half1 + seq_len/2;
ending_half1 = ending_half1 + seq_len/2;

bin_seq_fh = bin_seq(1:seq_len/2);
bin_seq_tot_fh(starting_half2:ending_half2) = bin_seq_fh;

bin_seq_xor = zeros(1,seq_len/2);
a = 1;
b = 2;
for i = 1:seq_len/2
    bin_seq_xor(i) = bitxor(bin_seq(a),bin_seq(b));
    a = a+2;
    b= b+2;
end
bin_seq_tot_xor(starting_half2:ending_half2) = bin_seq_xor;
```

```
starting_half2 = starting_half2 + seq_len/2;
ending_half2 = ending_half2 + seq_len/2;

end

prob_alter=(sum(bin_seq_tot_alter)/length(bin_seq_tot_alter))*100
   %probability of ones in the sequence
prob_fh=(sum(bin_seq_tot_fh)/length(bin_seq_tot_fh))*100
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%writing to a file
str = [' ', ' ', ' ' ];
filename = 'ds_fh_100mil.txt';
studID = fopen(filename,'wt');
bin_seq_tot_fh = dec2bin(bin_seq_tot_fh);

temp = bin_seq_tot_fh(1:24);
fprintf(studID, str);
fprintf(studID,'%s\n',temp);
temp = zeros(1,25);
start = 25;
ending = 49;
for i = 1:length(bin_seq_tot_fh)/25-1
   temp = bin_seq_tot_fh(start:ending);
   start = start + 25;
   ending = ending + 25;
fprintf(studID, str);
fprintf(studID,'%s\n',temp);

end
temp = bin_seq_tot_fh(start:end);
fprintf(studID, str);
fprintf(studID,'%s\n',temp);
```

```
fclose(studID);


str_alter = [' ', ' ', ' '];
file_alter = 'ds_alt_100mil.txt';
studID_alter = fopen(file_alter,'wt');
bin_seq_tot_alter = dec2bin(bin_seq_tot_alter);
temp_alter = bin_seq_tot_alter(1:24);
fprintf(studID_alter, str_alter);
fprintf(studID_alter,'%s\n',temp_alter);
temp_alter = zeros(1,25);
start_alter = 25;
ending_alter = 49;
for i = 1:length(bin_seq_tot_alter)/25-1
   temp_alter = bin_seq_tot_alter(start_alter:ending_alter);
   start_alter = start_alter + 25;
   ending_alter = ending_alter + 25;
fprintf(studID_alter, str_alter);
fprintf(studID_alter,'%s\n',temp_alter);
end
temp_alter = bin_seq_tot_alter(start_alter:end);
fprintf(studID_alter, str_alter);
fprintf(studID_alter,'%s\n',temp_alter);
fclose(studID_alter);


str_xor = [' ', ' ', ' '];
file_xor = 'ds_xor_100mil.txt';
studID_xor = fopen(file_xor,'wt');
bin_seq_tot_xor = dec2bin(bin_seq_tot_xor);
temp_xor = bin_seq_tot_xor(1:24);
fprintf(studID_xor, str_xor);
fprintf(studID_xor,'%s\n',temp_xor);
```

```
temp_xor = zeros(1,25);
start_xor = 25;
ending_xor = 49;
for i = 1:length(bin_seq_tot_xor)/25-1
   temp_xor = bin_seq_tot_xor(start_xor:ending_xor);
   start_xor = start_xor + 25;
   ending_xor = ending_xor + 25;
fprintf(studID_xor, str_xor);
fprintf(studID_xor,'%s\n',temp_xor);
end
temp_xor = bin_seq_tot_xor(start_xor:end);
fprintf(studID_xor, str_xor);
fprintf(studID_xor,'%s\n',temp_xor);
fclose(studID_xor);
```

B PURCS System

B.1 SKILL Code for Modeling the Chaotic Oscillator

```
declare(Vc[1])
Vc[0]=520m
myport1=outfile("/home/ashanta1/seneca/Monte Carlo 2
   output/Threshold/ctrl_0_Vc_520m_50_iter_threshold.csv")
cycle = 25
per_a= 1u
per=2*cycle*per_a

del=50n
ttr=100p
for( k 0 0
simulator( 'spectre )
design(
   "/data1/aysha/simulation/chaos_gate/spectre/schematic/netlist/netlist")
resultsDir( "/data1/aysha/simulation/chaos_gate/spectre/schematic" )
modelFile(
   '("/data1/IBM_PDK_MOD/cmos10lpe/V1.5.0.0RF/Spectre/models/design.scs"
      "")
)
stimulusFile( ?xlate nil
   "/home/ashanta1/seneca/Monte Carlo 2
      output/Threshold/stimuli_constantVc_4_input.scs")
analysis('tran ?stop "4*per+per/2" ?errpreset "conservative" )
desVar( "per_a" per_a )
desVar( "ttr" ttr )
desVar( "delay_a" del )
desVar( "del" del )
desVar( "per" per )
```

```
desVar( "Vc" Vc[k] )
envOption(
   'analysisOrder list("tran" "dc")
)
option( ?categ 'turboOpts
   'proc_affinity "8"
   'numThreads "32"
   'mtOption "Manual"
   'uniMode "APS"
)
temp( 27 )
run()
selectResult( 'tran )
;plot(getData("/clk") getData("/Vc") getData("/G0") getData("/net29")
    getData("/Y") getData("/clka") getData("/clkb") getData("/IN")
    getData("/OUT") getData("/nclkb") )
for(i 0 3
  for(j 0 cycle-1
    to1=i*per+(del+per/2)
    to2=j*per_a+(0.25*per_a)
    tsamp1=to1+to2
    y1=value(v("/out1" ?result "tran") tsamp1)
    fprintf(myport1 "%e," y1)
    tsamp2=tsamp1+per_a/2
    y2=value(v("/out2" ?result "tran") tsamp2)
    fprintf(myport1 "%e," y2)
    )
  fprintf(myport1 "\n")
  )
)
close(myport1)
```

B.2 Matlab Code for Creating the Characterization Table

```
%key structure(N-bit): Vc(p-bit)|n(q-bit)|Vth(r-bit)|Cb(s-bit)

%parameter
p=4; %Vc
q=5; %iteration
r=0; %threshold
s=1; %control bit
N=p+q+r+s;
Vc=[486,488,490,506,510,512,518,520,524,528,526,538,540,546,556,564];
Vth=0.6207; %determined by calculating the median of all the values in
   32 files
Cb=[0,1];
len_Vc=2^p;
len_Cb=2^s;
len_Vth=2^r;
len_n=2^q;

nchip=1;
for chip_no=1:nchip
%for chip_no=1:nchip
   temp_char=zeros(2^N,6); %filed for key,individual parameter,function
   for i=1:len_Vc %iterating through number of Vc
      for j=1:len_Cb %iterating through number of Cb
        infile=csvread(strcat(".\Raw data nominal
           edited\ctrl_",int2str(Cb(j)),
        "_Vc_",int2str(Vc(i)),"m_50_iter_edited.csv"));
         field_Vc=flip(de2bi(i-1,p));
         field_Cb=flip(de2bi(j-1,s));
         for k=1:len_n %iterating through n
            field_n=flip(de2bi(k-1,q));
```

```
            for l=1:len_Vth %iterating through Vth
               field_Vth];
               key=[field_n field_Vc field_Cb];
               key_index=bi2de(flip(key))+1;
               index_mat_funct=k+18:50:200;%index of 00,01,10,11 in
                  the raw file
               funct=infile(chip_no,index_mat_funct);
               %thresh=(l*(1.2)/(2^r));
               thresh=Vth(l);
               funct=(funct>thresh).*1; %convert analog output to
                  binary
               funct=bi2de(flip(funct));
               temp_char(key_index,1)=key_index;
               temp_char(key_index,2)=Vc(i);
               temp_char(key_index,3)=k+18;
               temp_char(key_index,4)=thresh;
               temp_char(key_index,5)=Cb(j);
               temp_char(key_index,6)=funct;
            end
         end
      end
   end
   csvwrite(strcat(".\Characterization_nominal_median
   \characterization_last_32_iteration_nominal_med.csv"),temp_char;
end
```

B.3 SKILL Code for Generating Data for Monte Carlo Simulation

```
NumRuns = 25  ;number of iteration (2 output chaos gates have double
   iterations)
MC     = 100   ;number of chips analyzed (# Monte Carlo sims)
```

```
per_a=1u
per = 2*NumRuns*per_a
del=50n
ttr=100p
Vc_temp = 486m

simTime = 4*per + per/2 ;4 inputs analyzed all at once
sprintf(simTstr "%e" simTime)

;====================== Set to XL mode
   ========================================
ocnSetXLMode()
ocnxlProjectDir( "/data1/aysha/mc_results" )
ocnxlTargetCellView( "chaos_redesigned" "chaos_gate" "adexl" )
ocnxlResultsLocation( "/data1/aysha/mc_results" )
ocnxlSimResultsLocation( "/data1/aysha/mc_results" )

;====================== Tests setup
   ============================================

ocnxlBeginTest("chaos_redesigned:chaos_gate:1")
simulator( 'spectre )
design( "chaos_redesigned" "chaos_gate" "schematic")
;resultsDir(
   "/data1/aysha/mc_results/chaos_gate_single_output/spectre/schematic"
   )
modelFile(
   '("/data1/IBM_PDK_MOD/cmos10lpe/V1.5.0.0RF/Spectre/models/design.scs"
      "")
)
stimulusFile( ?xlate nil
```

```
   "/home/ashanta1/seneca/Monte Carlo 2 output/Monte
      Carlo/stimuli_constantVc_2_output_ctrl_0_mc.scs")
analysis('tran ?stop simTstr ?errpreset "conservative" )

desVar( "per_a" per_a )
desVar( "ttr" ttr )
desVar( "delay_a" del )
desVar( "del" del )
desVar( "per" per )
desVar( "Vc" Vc_temp )


printf("\n\n\n\nhere %d\n\n\n\n" NumRuns)

envOption(
  'analysisOrder list("tran" "pz" "dcmatch" "stb" "envlp" "ac" "dc"
     "noise" "xf" "sp" "pss" "pac" "pstb" "pnoise" "pxf" "psp" "qpss"
     "qpac" "qpnoise" "qpxf" "qpsp" "hb" "hbac" "hbnoise" "sens")
)
temp( 27 )

option( ?categ 'turboOpts
  'apsplus t
      'mtOption "Auto"
  'uniMode "APS"
)


run()
selectResult( 'tran )
```

```
;plot(getData("/clk") getData("/G0") getData("/net29")
   getData("/clka") getData("/clkb") getData("/IN") getData("/nclkb")
   getData("/out1") getData("/out2"))


for(ind1 1 4
ind3 = 0
  for(ind2 1 NumRuns
    to1 = (ind1-1)*per+del+per/2
    to2 = (ind2-1)*per_a + per_a/4
    tsamp1 = to1 + to2
    sprintf(Oout "value(v(\"/out1\" ?result \"tran\") %e)" tsamp1)
    sprintf(Ooutn "out%d_%d" (ind1-1) ind3 )
    ocnxlOutputExpr( Oout ?name Ooutn ?plot t ?save t)
    ind3 = ind3 + 1
    tsamp2 = tsamp1 + per_a/2
    sprintf(Oout "value(v(\"/out2\" ?result \"tran\") %e)" tsamp2)
    sprintf(Ooutn "out%d_%d" (ind1-1) ind3 )
    ocnxlOutputExpr( Oout ?name Ooutn ?plot t ?save t)
    ind3 = ind3 + 1

)
)

ocnxlEndTest() ;

;====================== Model Group setup
   ========================================

;====================== Corners setup
   ========================================
```

```
;====================== Job setup
   ================================================

ocnxlJobSetup( '(
  "blockemail" "1"
  "configuretimeout" "300"
  "distributionmethod" "Local"
  "lingertimeout" "300"
  "maxjobs" "4"
  "name" "ADE XL Default"
  "preemptivestart" "1"
  "reconfigureimmediately" "1"
  "runtimeout" "-1"
  "showerrorwhenretrying" "1"
  "showoutputlogerror" "0"
  "startmaxjobsimmed" "1"
  "starttimeout" "300"
  "usesameprocess" "1"
) )

;====================== Disabled items
   ========================================

;====================== Run Mode Options
   ====================================
sprintf(num_of_MC "%d" MC)
ocnxlMonteCarloOptions( ?mcMethod "all" ?mcNumPoints num_of_MC
   ?mcNumBins "" ?mcStopEarly "0" ?mcStopMethod "Significance Test"
   ?samplingMode "random" ?saveProcess "1" ?saveMismatch "0"
   ?useReference "0" ?donominal "1" ?saveAllPlots "0" ?monteCarloSeed
   "" ?mcStartingRunNumber "" ?dumpParamMode "yes" )
```

```
;====================== Starting Point Info
   ======================================

;====================== Run command
   ============================================
ocnxlRun( ?mode 'monteCarlo ?nominalCornerEnabled t ?allCornersEnabled
   t ?allSweepsEnabled t)
ocnxlOutputSummary(?yieldSummary t ?exprSummary nil ?specSummary nil
   ?detailed nil)


;====================== save data points
   ======================================

sprintf(strYield "/data1/aysha/mc_results/yield_chaos_gate.csv")
sprintf(strTrans "/data1/aysha/mc_results/outputs_chaos_gate.csv")
sprintf(dtTranFile
   "/data1/aysha/mc_results/mc_chaos_gate_50_iter_100_chip_Vc_486m_ctrl_0.csv")
sprintf(detailFile "/data1/aysha/mc_results/chaos_gate_det_raw.csv")
ocnxlExportOutputView(strYield "Yield")
ocnxlExportOutputView(dtTranFile "Detail-Transpose")
ocnxlExportOutputView(detailFile "Detail")

axlOutputsExportToFile(ocnxlGetSession() strTrans)

;====================== End XL Mode command
   ==================================

ocnxlEndXLMode()
```

B.4 Python Code for Calculating the Controllability

```
import csv
import sys
import math

def controllability(netarray, filename):
  rows = []
  fields = []
  with open(filename, 'r') as csvfile:
    # creating a csv reader object
    csvreader = csv.reader(csvfile)

    # extracting field names through first row
    fields = next(csvreader)

    # extracting each data row one by one
    for row in csvreader:
      rows.append(row)

    final = []
    flag = 0

    nodes = []
    for j in range(0,len(netarray)):
      nodes.append(netarray[j][1]);


    for j in range(0,len(netarray)):
      inp1 = netarray[j][2];
      inp2 = netarray[j][3];
```

```
#count1 and count2 represent the column index
count1 = 0
count2 = 0
col1 = []
col2 = []
for i in range (0,len(fields)):
  if inp1 == fields[i]:
    count1 = i;
  if inp2 == fields[i]:
    count2 = i;

#extracting the values of columns in col1 and col2
for i in range(0, len(rows)):
  col1.append(rows[i][count1])
  col2.append(rows[i][count2])

#a,b,c,d keeps track of how many combinations of 00, 01, 10 and
   11 exists
a = 0
b = 0
c = 0
d = 0
temp = []
for i in range(0,len(col1)):
  if (col1[i] == '0' and col2[i] == '0'):
    a = a + 1
  elif (col1[i] == '0' and col2[i] == '1'):
    b = b + 1
  elif (col1[i] == '1' and col2[i] == '0'):
    c= c + 1
  else:
    d = d + 1
```

```
        temp.append(a)
        temp.append(b)
        temp.append(c)
        temp.append(d)
        final.append(temp)

    #logth contains the value of controllability
    prob = []
    logth = 0
    for j in range(0,len(final)):
        inp1 = netarray[j][2];
        inp2 = netarray[j][3];
        if inp1 == inp2:
            flag = 1
        else:
            flag = 0
        x = final[j]
        sum = 0
        for k in range(0,4):
            temp = x[k]/float(len(rows))
            sum = sum + temp*temp
        if flag == 0:
            logth = math.log((1/sum),4)
        else:
            logth = math.log((1/sum),2)
        prob.append(logth)
    index = sorted(range(len(prob)), key=lambda k: prob[k], reverse =
        True)

return prob, nodes
```

B.5 Python Code for Calculating the Observability

```
import csv
import sys

def recurse(node,path,temp,netarray,outpname,inp1,inp2):
  #global netarray,outpname,inp1,inp2
  temp_copy=temp[0:]
  #find children
  index1 = [index for index in range(len(inp1)) if inp1[index] == node]
  index2 = [index for index in range(len(inp2)) if inp2[index] == node]
  index=set(index1+index2)
  child=[netarray[i][1] for i in index]

  for i in range(0,len(child)):
    if child[i] not in outpname:
      temp.append(child[i])
      recurse(child[i],path,temp[0:],netarray,outpname,inp1,inp2)
      temp=temp_copy[0:]

    else:
      temp.append(child[i])
      path.append(temp)
      temp=temp[0:-1]

  return path


def cond_observ(node,path,netarray,outpname,outp,inp1,inp2,gate):
  #global netarray,outpname,outp,inp1,inp2,gate
  dict={'AND':1,'NAND':1,'OR':0,'NOR':0}
  name_secinp=[]
```

```
    val_secinp=[]
    for i in range(0,len(path)):
      tmp_name_secinp=[]
      tmp_val_secinp=[]
      for j in range(0,len(path[i][:])):
        ind=outp.index(path[i][j])
        tmp_gate=gate[ind]
        if (j==0):
          if((tmp_gate!='XOR') and (tmp_gate!='XNOR')):
               if(inp1[ind]!=node):
                    tmp_name_secinp.append(inp1[ind])
                    tmp_val_secinp.append(dict[tmp_gate])
               elif(inp2[ind]!=node):
                    tmp_name_secinp.append(inp2[ind])
                    tmp_val_secinp.append(dict[tmp_gate])
        else:
          if((tmp_gate!='XOR') and (tmp_gate!='XNOR')):
               if(inp1[ind]!=path[i][j-1]):
                    tmp_name_secinp.append(inp1[ind])
                    tmp_val_secinp.append(dict[tmp_gate])
               elif(inp2[ind]!=node):
                    tmp_name_secinp.append(inp2[ind])
                    tmp_val_secinp.append(dict[tmp_gate])
      name_secinp.append(tmp_name_secinp)
      val_secinp.append(tmp_val_secinp)
    return name_secinp,val_secinp

def different_output_paths(path, outpname):
    num_output = len(outpname)
    lists = []

    for i in range(0,num_output):
```

```
    gatelist_temp=[]
    for j in range(0, len(path)):
      if path[j][-1] == outpname[i]:
        gatelist_temp.append(path[j])
    if (len(gatelist_temp)!=0):
      lists.append(gatelist_temp)
  return lists

def final_count_code(nodename,nodevalue,filename):
  c = any( isinstance(e, list) for e in nodename)
  rows = []
  fields = []
  with open(filename, 'r') as csvfile:
    # creating a csv reader object
    csvreader = csv.reader(csvfile)

    # extracting field names through first row
    fields = next(csvreader)

    # extracting each data row one by one
    for row in csvreader:
      rows.append(row)

    if c == False:
      col = []
      index = []
      for k in range(0,len(nodename)):
        for i in range(0,len(fields)):
          if nodename[k] == fields[i]:
            col.append(i)

      final_count = 0
```

```
    nodevalues = []
    for i in range(0,len(nodevalue)):
      nodevalues.append(str(nodevalue[i]))

    for i in range(0,len(rows)):
      val = []
      for k in range(0,len(col)):
        val.append(rows[i][col[k]])
      if val == nodevalues:
        index.append(i)

    final_count = len(index)

  else:
    col_final = []
    index = []
    for k in range(0,len(nodename)): #length of list
      col = []
      for p in range(0,len(nodename[k])): #length of sublist
        for i in range(0,len(fields)):
          if nodename[k][p] == fields[i]:
            col.append(i)
      col_final.append(col)

    final_count = 0
    nodevalues = []
    for i in range(0,len(nodevalue)):
      temp = []
      for k in range(0,len(nodevalue[i])):
        temp.append(str(nodevalue[i][k]))
      nodevalues.append(temp)
```

```
        for i in range(0,len(rows)):
          for k in range(0,len(col_final)):
            val = []
            for p in range(0,len(col_final[k])):
              val.append(rows[i][col_final[k][p]])
            if val == nodevalues[k]:
              index.append(i)
        index = set(index)
        final_count = len(index)/float(len(rows))

    return final_count

def observability(netarray, outpname, filename):
    inp1=[]
    for i in range(0,len(netarray)):
      inp1.append(netarray[i][2])

    inp2=[]
    for i in range(0,len(netarray)):
      inp2.append(netarray[i][3])

    outp=[]
    for i in range(0,len(netarray)):
      outp.append(netarray[i][1])

    gate=[]
    for i in range(0,len(netarray)):
      gate.append(netarray[i][0])

    cand_outp = outp
    obs_res=[0]*len(cand_outp)
```

```
  for i in range(0,len(cand_outp)): #iterating over all gate output
    print("operating on gate output:",i)
    tarnode=cand_outp[i]
    path = recurse(tarnode,[],[],netarray,outpname,inp1,inp2)
    lists = different_output_paths(path, outpname)
    temp_calc=[0]*(len(outpname)+1)
    if tarnode in outpname:
      temp_calc[0]=1
    for j in range(0,len(lists)):
      testlist=lists[j]
      nodename,nodevalue=cond_observ(tarnode,
      testlist,netarray,outpname,outp,inp1,inp2,gate)
      fcount=final_count_code(nodename,nodevalue,filename)
      temp_calc[1+j]=fcount
    sum_temp=sum(temp_calc)/float(len(outpname))
    obs_res[i]=sum_temp
  return obs_res, cand_outp
```

B.6 Python Code for Calculating the Testability

```
from observability_final import observability
from controllability_final import controllability
from bench_to_spice_format import extract_from_bench
from bench_to_spice_format import write_in_scp
from logicsolver import mixed_logic_solver
import numpy as np
from random import *
import csv
import sys
import math
```

```
def netlist_to_netarray(fnetlist):
  netarray=[] # 2D list for gate description
  netname=[]
  with open(fnetlist) as fnet:
    for line in fnet:
      '''
      line=line.replace(" ","")
      line=line.strip("\n")
      '''
      count=0
      temp=[]
      for word in line.split():
        #print(word)
        temp.append(word)
        if count!=0:
          if word not in netname:
            netname.append(word)
        count=count+1
      netarray.append(temp)
  return netarray

def generating_truth_table(fspice, inpname, outpname, inpval, rows):
  filename = 'bool_truthtable_'+fspice+'.csv'
  netarray=netlist_to_netarray(fspice)
  f=open(filename,'w')
  netval = mixed_logic_solver(fspice,inpname,inpval,outpname)
  net_keys = list(netval.keys())
  for j in range(len(net_keys)):
    f.write(str(net_keys[j]))
    #f.write(net_keys[j])
    f.write(',')
  f.write('\n')
```

```
linp = len(inpname)
print('generating boolean truth table .....')
for i in range(0,rows):
  print('truth table row:',i)
  cryptogen = SystemRandom()
  inpval = [cryptogen.randrange(2) for i in range(linp)]
  #adding HI and LO if there exists NOT gates in the schematic
  for i in range (0,len(inpname)):
    if inpname[-2] == 'HI' and inpname[-1] == 'LO':
      del inpval[-1]
      del inpval[-1]
      inpval.append(1)
      inpval.append(0)
      break
    else:
      for i in range (0,len(netarray)):
        if netarray[i][3] == 'LO':
          del inpval[-1]
          inpval.append(0)
          break
      for i in range (0,len(netarray)):
        if netarray[i][3] == 'HI':
          del inpval[-1]
          inpval.append(1)
          break
  #print("inpval", inpval)
  netval = mixed_logic_solver(fspice,inpname,inpval,outpname)
  #print(netval)
  for j in netval:
    output = netval[j]
    #print(output)
    f.write(str(output))
```

```
            f.write(',')
        f.write('\n')

    f.close()
    return filename

def testability(fbench,bench_no,rows):
    inpname,keyname,outpname,net = extract_from_bench(fbench)
    print(inpname,len(inpname))
    fspice = write_in_scp(fbench,bench_no)
    netarray=netlist_to_netarray(fspice)
    #print(netarray)
    linp=len(inpname)
    #print("length of input name",linp)
    cryptogen = SystemRandom()
    inpval = [cryptogen.randrange(2) for i in range(linp)]
    #adding HI and LO if there exists NOT gates in the schematic
    for i in range (0,len(netarray)):
        if netarray[i][3] == 'HI':
            inpname.append('HI')
            inpval.append(1)
            break
    for i in range (0,len(netarray)):
        if netarray[i][3] == 'LO':
            inpname.append('LO')
            inpval.append(0)
            break
    filename =  generating_truth_table(fspice, inpname, outpname,
       inpval, rows) #HI and LO is already appended in inpname
    print("truthtable generation complete")
    print("observability in progress.....")
    obs, outp = observability(netarray, outpname, filename)
```

```
  print("observability calculation complete")
  print("controllability in progress....")
  prob, nodes= controllability(netarray, filename)
  print("controllability calculation complete")
  test = []
  for i in range(0,len(obs)):
    temp = obs[i]*prob[i]
    test.append(temp)
  index = sorted(range(len(test)), key=lambda k: test[k], reverse =
     True)
  return index
```

B.7 Python Code for Solving a Given Netlist

```
def logic_func(*args):
  arg=[]
  for val in args:
    arg.append(val)
  if arg[0]=='AND':
    outp=arg[1] & arg[2]
  elif arg[0]=='OR':
    outp=arg[1] | arg[2]
  elif arg[0]=='XOR':
    outp=arg[1] ^ arg[2]
  elif arg[0]=='NAND':
    outp=(~(arg[1] & arg[2]))%2
  elif arg[0]=='NOR':
    outp=(~(arg[1] | arg[2]))%2
  elif arg[0]=='XNOR':
    outp=(~(arg[1] ^ arg[2]))%2
  return outp
```

```
def mixed_logic_solver(fnetlist,inpname,inpval,outpname):

  netname=[] # list of all nets in the netlist
  netarray=[] # 2D list for gate description
  with open(fnetlist) as fnet:
    for line in fnet:
      count=0
      temp=[]
      for word in line.split():
        #print(word)
        temp.append(word)
        if count!=0:
          if word not in netname:
            netname.append(word)
        count=count+1
      netarray.append(temp)

  inpdict={}
  #creating dictionary for input net name and value
  for i in range(0,len(inpname)):
    inpdict[inpname[i]]=inpval[i]


  netval={} #dictionary for keeping the value of each net
  netstatus={} #dictionary for keeping the updated status
     (valid/invalid) of each net
  for i in range(0,len(netname)):

    #initializing value of each net
    if netname[i] in inpname:
      netval[netname[i]]=inpdict[netname[i]]
```

```
  else:
    netval[netname[i]]=0
  #print(netval)
  #initializing status of each net
  if netname[i] in inpname:
    netstatus[netname[i]]=1
  else:
    netstatus[netname[i]]=0

#solver
#iterating each gate
i=0
while (i<len(netarray)):
  gate_name=netarray[i][0]
  o_p=netarray[i][1]
  i_p1=netarray[i][2]
  i_p2=netarray[i][3]


  #if the inputs of the gate are evaluated yet
  if (netstatus[i_p1]==1)&(netstatus[i_p2]==1):

    netval[o_p]=logic_func(gate_name,netval[i_p1],netval[i_p2])
    netstatus[o_p]=1
    i=i+1
  else:
    #print(False)
    #taking the element to the last of the list for considering
       later
    temp=netarray[i]
    del(netarray[i])
    netarray.append(temp)
```

```
    #print(netarray)
    #print("netval:", netval)
    #print("netstatus:", netstatus)
  outpval=[0]*len(outpname)
  for i in range(0,len(outpname)):
    outpval[i]=netval[outpname[i]]
  #print(netval['s'])
  #print(netval['co'])
  return netval
```

B.8 Python Code for Replacing the Gates Based on Testability

```
from random import *
def testable_chao_netlist(fnetlist,clist):
  #fnetlist='ISCAS85.scp'
  netarray=[]
  #read original netlist file
  with open(fnetlist) as fnet:
      for line in fnet:
        '''
        line=line.replace(" ","")
        line=line.strip("\n")
        '''
        count=0
        temp=[]
        for word in line.split():
          #print(word)
          temp.append(word)
          # if count!=0:
            # if word not in netname:
              # netname.append(word)
```

```
          count=count+1
        netarray.append(temp)
#print(netarray)

n=len(netarray)
#clist=[] # gate no. to be replace
klist=[]
kname=[]

for i in range(0,len(clist)):
  #print(clist)
  kname.append(netarray[clist[i]][0])
  #print(kname)
  netarray[clist[i]][0]='CHAO'
  netarray[clist[i]].append('k'+str(clist[i]+1))
  klist.append('k'+str(clist[i]+1))
  #print(klist)



#write modified netlist to another file
fmod=open('netlist_testable_replacement.scp','w')
for i in range(0,n):
  for j in range(0,len(netarray[i])):
    fmod.write(netarray[i][j])
    fmod.write(' ')
  if i<=n:
    fmod.write('\n')
return klist,kname
close(fmod)
```

B.9 Python Code for Calculating the Hamming Distance

```
from testability_chaogate_replacement import testable_chao_netlist
from random_chaogate_replacement import random_chao_netlist
from testability_final import testability
from logicsolver_chao import mixed_logic_solver
from logicsolver_chao import logic_func
from bench_to_spice_format import extract_from_bench
from finding import finding_index
from random import *
import math

#netlist is in .scp format
def create_netarray(netlist):
  netname = []
  netarray = []
#making netarray from scp
  with open(netlist_original) as fnet:
       for line in fnet:
         count=0
         temp=[]
         for word in line.split():
           #print(word)
           temp.append(word)
           if count!=0:
             if word not in netname:
               netname.append(word)
           count=count+1
         netarray.append(temp)
  return netarray

cryptogen = SystemRandom()
```

```
################INPUTS#####################################
bench_no = [3540]
no_chip=65
replace_percentage =
   [0.05,0.10,0.15,0.20,0.25,0.30,0.35,0.40,0.45,0.50,
0.55,0.60,0.65,0.70,0.75,0.80,0.85,0.90,0.95,1.00]
#replace_percentage =
   [0.10,0.20,0.30,0.40,0.50,0.60,0.70,0.80,0.90,1.00]
#replace_percentage = [0.05, 0.15, 0.25]
no_of_inputs = [200] #different number of inputs where for each input
   key varies to generate challenge
no_of_char_files = 1
observation=500; # no. of measurement to find the hamming distance
   between responses from correct and random key
rows = [5000]
###########################################################
kb=10 #keybits
####################finding 10 random chips##############
char_file_no = []

cc = [cryptogen.randrange(1,66)]
for p in range(0,no_of_char_files):
  while (cc in char_file_no):
       cc = [cryptogen.randrange(1,66)]
    #print("cc",cc)
  char_file_no.append(cc)
print("characterization file",char_file_no)
###########################################################
for bb in range(0,len(bench_no)):

  netlist_bench='./Benchmark Circuits/c'+str(bench_no[bb])+'.bench'
```

```
inpname,keyname,outpname,net = extract_from_bench(netlist_bench)
   #extracting inpname, keyname and outpname from bench
linp=len(inpname)
loutp = len(outpname)
print('length of output ' + str(loutp))
inpsize = 2**linp
print("length of input",linp)

netlist_original = './Benchmark Circuits/c'+str(bench_no[bb])+'.scp'
netarray = create_netarray(netlist_original)
print("netarray", netarray)
#############generating different inputs to apply to the
   circuit###############################
inpval = []
temp = []
for i in range (0,len(netarray)):
  if netarray[i][3] == 'HI':
     inpname.append('HI')
     break
for i in range (0,len(netarray)):
  if netarray[i][3] == 'LO':
     inpname.append('LO')
     break

temp = [cryptogen.randrange(2) for i in range(linp)]
for p in range(0,no_of_inputs[bb]):
  while (temp in inpval):
     temp = [cryptogen.randrange(2) for i in range(linp)]

  #adding HI and LO if there exists NOT gates in the schematic
  if inpname[-2] == 'HI' and inpname[-1] == 'LO':
     temp.append(1)
```

```
            temp.append(0)
        else:
            if inpname[-1] == 'HI':
                temp.append(1)
            if inpname[-1] == 'LO':
                temp.append(1)
        inpval.append(temp)
########################################
    print("bench number", bench_no[bb])
    print("input value",inpval)
    print("length of input value", len(inpval))
    print("input name", inpname)
    print("output name", outpname)
    print("input size", inpsize)

    fnetlist='netlist_testable_replacement.scp'
    complete_list = testability(netlist_bench,bench_no[bb],rows[bb])
    print("complete_list", complete_list)
    len_complete = len(complete_list)
    print("length of complete list", len_complete)

    for aa in range(0,len(replace_percentage)):
        len_clist = int(math.ceil(replace_percentage[aa]*len_complete))
        print("length of replaced list", len_clist)
        clist = []
        for i in range(0,len_clist):
            clist.append(complete_list[i])
        print("replaced gates", clist)

        [keyname,keygate]=testable_chao_netlist(netlist_original,clist)
        print("keyname",keyname)
        print("keygate", keygate)
```

```
keyval=[0]*len(keyname)
lkey=len(keyname)

hd_filename='ent_hamming_distance_testability_c'
+str(bench_no[bb])+'_perc'+' '+str(replace_percentage[aa])+'.csv'
   #chip_number in different columns
f_hd=open(hd_filename,'w')
for p in range(0, len(char_file_no)):
  char_file='./Characterization 65 chip
     new/characterization_65_chip_last
  _32_iteration_'+str(char_file_no[p][0])+'.csv'
  a, b, c, d, e, f = finding_index(char_file) #and, or, xor,
     nand, nor, xnor
  keydict={'AND':a,'OR':b,'XOR':c,'NAND':d,'NOR':e, 'XNOR':f}
  print("chip number", char_file_no[p][0])
  ###uncomment this for netlist with key gate
  keyval_correct=[0]*len(keygate)
  for i in range(0,len(keygate)):
    keyval_correct[i]=keydict[keygate[i]]
  #print("correct keyval",keyval_correct)
  keyval=[0]*len(keyval_correct)
  ###
  cum_HD=0.0
  cum_HD_ent = 0.0
  cum_HD_inp_ent = []
  for i in range(0,no_of_inputs[bb]):
    cum_HD_inp=0.0
    # cum_HD_ent = 0.0
    #print(i)
    resp_correct=mixed_logic_solver(char_file,
    fnetlist,inpname,inpval[i],keyname,keyval_correct,outpname)
    #print("correct_response", resp_correct)
```

```
    for j in range(0,observation):
      #print(j)
      for jj in range(0,lkey):
        keyval[jj]=randint(0,(2**kb)-1)
      #print("random keyval",keyval)
      resp=mixed_logic_solver(char_file,
    fnetlist,inpname,inpval[i],keyname,keyval,outpname)
      #print("wrong_response", resp)
      temp_sum=0.0
      temp_sum_ent = 0.0
      for k in range(0,loutp):
        temp_sum=temp_sum+abs(resp[k]-resp_correct[k])
        temp_sum_ent=temp_sum_ent+abs(resp[k]-resp_correct[k])
      temp_sum=temp_sum/loutp
      temp_sum_ent=temp_sum_ent/loutp
      #print(temp_sum)
      cum_HD=cum_HD+temp_sum
      cum_HD_inp=cum_HD_inp+temp_sum
      #print(cum_HD)
      if temp_sum_ent > 0.5:
        temp_sum_ent = 1-temp_sum_ent
      cum_HD_ent=cum_HD_ent+temp_sum_ent
    cum_HD_inp=(cum_HD_inp*100)/(observation)
    print("cum_HD_input",cum_HD_inp)
    if cum_HD_inp > 50:
      cum_HD_inp = 100-cum_HD_inp
    cum_HD_inp_ent.append(cum_HD_inp)
print("cum_HD_inp_ent",cum_HD_inp_ent)
cum_HD=(cum_HD*100)/(observation*no_of_inputs[bb])
cum_HD_ent=(cum_HD_ent*100)/(observation*no_of_inputs[bb])
sum = 0.0
for i in range(0,len(cum_HD_inp_ent)):
```

```
        sum = sum + cum_HD_inp_ent[i]
      avg = sum/len(cum_HD_inp_ent)

      print(cum_HD)
      print(cum_HD_ent)
      print(avg)
      f_hd.write(str(cum_HD)+'\n')
      f_hd.write(str(cum_HD_ent)+'\n')
      f_hd.write(str(avg)+'\n')

    f_hd.close()

fnetlist='netlist_random_replacement.scp'

for aa in range(0,len(replace_percentage)):
    replace = int(math.ceil(replace_percentage[aa]*len(netarray)))
    [keyname,keygate]=random_chao_netlist(netlist_original,replace)
    print("keyname",keyname)
    print("keygate", keygate)
    keyval=[0]*len(keyname)
    lkey=len(keyname)

    hd_filename='ent_hamming_distance_random_c'+str(bench_no[bb])+'_perc'+'
        '+str(replace_percentage[aa])+'.csv' #chip_number in different
        columns
    f_hd=open(hd_filename,'w')
    for p in range(0, len(char_file_no)):
      char_file='./Characterization 65 chip
          new/characterization_65_chip_last_
      32_iteration_'+str(char_file_no[p][0])+'.csv'
      a, b, c, d, e, f = finding_index(char_file) #and, or, xor,
          nand, nor, xnor
```

```
keydict={'AND':a,'OR':b,'XOR':c,'NAND':d,'NOR':e, 'XNOR':f}
print("chip number", char_file_no[p][0])
###uncomment this for netlist with key gate
keyval_correct=[0]*len(keygate)
for i in range(0,len(keygate)):
  keyval_correct[i]=keydict[keygate[i]]
#print("correct keyval",keyval_correct)
keyval=[0]*len(keyval_correct)
###
cum_HD=0.0
cum_HD_ent = 0.0
cum_HD_inp_ent = []
for i in range(0,no_of_inputs[bb]):
  cum_HD_inp=0.0
  # cum_HD_ent = 0.0
  #print(i)
  resp_correct=mixed_logic_solver(char_file,
  fnetlist,inpname,inpval[i],keyname,keyval_correct,outpname)
  #print("correct_response", resp_correct)
  for j in range(0,observation):
    #print(j)
    for jj in range(0,lkey):
      keyval[jj]=randint(0,(2**kb)-1)
    #print("random keyval",keyval)
    resp=mixed_logic_solver(char_file,
   fnetlist,inpname,inpval[i],keyname,keyval,outpname)
    #print("wrong_response", resp)
    temp_sum=0.0
    temp_sum_ent = 0.0
    for k in range(0,loutp):
      temp_sum=temp_sum+abs(resp[k]-resp_correct[k])
      temp_sum_ent=temp_sum_ent+abs(resp[k]-resp_correct[k])
```

```
            temp_sum=temp_sum/loutp
            temp_sum_ent=temp_sum_ent/loutp
            #print(temp_sum)
            cum_HD=cum_HD+temp_sum
            cum_HD_inp=cum_HD_inp+temp_sum
            #print(cum_HD)
            if temp_sum_ent > 0.5:
              temp_sum_ent = 1-temp_sum_ent
            cum_HD_ent=cum_HD_ent+temp_sum_ent
          cum_HD_inp=(cum_HD_inp*100)/(observation)
          #print("cum_HD_input",cum_HD_inp)
          if cum_HD_inp > 50:
            cum_HD_inp = 100-cum_HD_inp
          cum_HD_inp_ent.append(cum_HD_inp)
        #print("cum_HD_inp_ent",cum_HD_inp_ent)
        cum_HD=(cum_HD*100)/(observation*no_of_inputs[bb])
        cum_HD_ent=(cum_HD_ent*100)/(observation*no_of_inputs[bb])
        sum = 0.0
        for i in range(0,len(cum_HD_inp_ent)):
          sum = sum + cum_HD_inp_ent[i]
        avg = sum/len(cum_HD_inp_ent)

        print(cum_HD)
        print(cum_HD_ent)
        print(avg)
        f_hd.write(str(cum_HD)+'\n')
        f_hd.write(str(cum_HD_ent)+'\n')
        f_hd.write(str(avg)+'\n')
    f_hd.close()
```

Vita

Aysha S. Shanta received her B.Sc in Electrical and Electronic Engineering (EEE) from BRAC University in 2012. She worked as a Lecturer at BRAC University for 3 years before coming to USA for pursuing higher studies. She is currently working towards her PhD in Electrical Engineering from University of Tennessee, Knoxville (UTK). She has worked as a Graduate Teaching Assistant for four years at UTK and received Outstanding Graduate Teaching Assistant award twice. Her research interests include VLSI design, carbon nano-electrode based biosensors, hardware security and chaos-based computing systems.