Asynchronous physical unclonable function using FPGA-based self-timed ring oscillator

Roshan Silwal

The University of Toledo

Follow this and additional works at: http://utdr.utoledo.edu/theses-dissertations

Recommended Citation

This Thesis is brought to you for free and open access by The University of Toledo Digital Repository. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of The University of Toledo Digital Repository. For more information, please see the repository's About page.
A Thesis

entitled

Asynchronous Physical Unclonable Function using FPGA-based Self-Timed Ring Oscillator

by

Roshan Silwal

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the

Master of Science Degree in Electrical Engineering

Dr. Mohammed Y Niamat, Committee Chair

Dr. Robert C. Green II, Committee Member

Dr. Weiqing Sun, Committee Member

Dr. Patricia R. Komuniecki, Dean
College of Graduate Studies

The University of Toledo

August 2013
An Abstract of

Asynchronous Physical Unclonable Function using FPGA-based Self-Timed Ring Oscillator

by

Roshan Silwal

Submitted to the Graduate Faculty as partial fulfillment of the requirements for the Master of Science Degree in Electrical Engineering

The University of Toledo

August 2013

Field Programmable Gate Array (FPGA) security has emerged as a challenging security paradigm in system design. Systems implemented on FPGAs require secure operations and communication. There is a growing concern over the security attributes of FPGAs regarding protecting and securing information processed within them, protecting designs during distribution and protecting intellectual property rights. One of the important aspects of improving the trustworthiness level of FPGAs is enhancing the physical security of FPGAs. A Physical Unclonable Function (PUF) provides a means to enhance physical security of Integrated Circuits (ICs) against piracy and unauthorized access. PUFs exploit the inherent and embedded randomness that occurs during the fabrication process of silicon devices.

This thesis presents a novel FPGA-based PUF design technique using asynchronous logic. Significant process variations exist in IC fabrication, which makes each IC unique in its delay characteristics. The statistical delay variation in transistors and wires across FPGA chips is exploited through identically laid-out asynchronous ring oscillators. The asynchronous ring oscillators generate oscillations of varying frequencies
when the oscillators are identically mapped on a semiconductor device. These varying frequencies produced by identically mapped self-timed ring oscillators are used to generate unique PUF response bits, which are used in device authentication and cryptographic applications such as generating secret keys and True Random Number Generator (TRNG). Experimental analysis shows that asynchronous oscillators of PUFs generate oscillations of varying frequencies, and the uniqueness for the PUF responses is 49.92%, which is very close to the desired 50% factor.
This thesis is dedicated to my parents, my sisters and my lovely wife.
Acknowledgements

I would like to express my deep sense of gratitude to my thesis supervisor, Dr. Mohammed Niamat, for giving me an opportunity to work with him in this research and providing me a tremendous level of support and cooperation throughout my research work and graduate studies.

I would also like to thank the thesis committee member Dr. Robert C. Green II and Dr. Weiqing Sun for their valuable time in reviewing this thesis.

The research work in this thesis was supported in part by National Science Foundation (NSF) grant award #203687.
Table of Contents

Abstract ........................................................................................................................................ iii
Acknowledgements .................................................................................................................. vi
Table of Contents .................................................................................................................. vii
List of Tables ........................................................................................................................ xi
List of Figures ........................................................................................................................ xii
List of Abbreviations ............................................................................................................. xv
List of Symbols .................................................................................................................... xvii
1 Introduction .......................................................................................................................... 1
  1.1 Context and Motivation ................................................................................................. 1
  1.2 Contributions ............................................................................................................... 3
  1.3 Thesis Outline ............................................................................................................. 4
2 Physical Unclonable Functions ............................................................................................ 5
  2.1 Introduction .................................................................................................................. 5
  2.2 PUF Terminologies ....................................................................................................... 6
    2.2.1 Significance of Process Variations ........................................................................ 6
    2.2.2 Environmental Variations ..................................................................................... 7
    2.2.3 Challenge-Response Pairs ..................................................................................... 8
  2.3 Sources of Noise .......................................................................................................... 8
    2.3.1 Noise due to Manufacturing Process ..................................................................... 8
2.3.2 Local Noise ................................................................. 8
2.3.3 Environmental Noise .................................................... 9
2.4 Measure of Quality .......................................................... 9
  2.4.1 Uniqueness ............................................................... 9
  2.4.2 Reliability ............................................................... 10
  2.4.3 Resiliency ............................................................... 11
2.5 PUF Classifications ......................................................... 11
  2.5.1 Non-electronic PUF, Electronic PUF and Silicon PUF ........... 12
  2.5.2 Strong PUF and Weak PUF ........................................ 13
  2.5.3 Intrinsic PUF and Non-intrinsic PUF ............................. 13
2.6 PUF Circuits ................................................................... 14
  2.6.1 Delay-based PUF ....................................................... 14
    2.6.1.1 Arbiter PUF ..................................................... 14
    2.6.1.2 Ring Oscillator PUF ......................................... 15
    2.6.1.3 Glitch PUF ..................................................... 17
  2.6.2 Memory-based PUF .................................................... 18
    2.6.2.1 SRAM PUF ..................................................... 18
    2.6.2.2 Butterfly PUF ................................................ 19
2.7 PUF Applications ............................................................ 20
3  **Self-Timed Rings** .......................................................... 24
  3.1 Introduction .................................................................. 24
  3.2 Asynchronous Circuits ................................................... 24
  3.3 Asynchronous Logic ..................................................... 25
3.3.1 Muller C-element.................................................................27
3.4 Self-Timed Rings........................................................................28
  3.4.1 Self-Timed Ring Structure......................................................28
  3.4.2 Token and Bubble Propagation..............................................30
  3.4.3 Jitter in Inverter RO and Self-Timed RO.................................31

4 Asynchronous Approach to Ring Oscillator for FPGA-based PUF Design ...33
  4.1 Introduction ..............................................................................33
  4.2 FPGA Architecture ..................................................................34
    4.2.1 Architecture of Spartan-II ..................................................35
  4.3 LUT Implementation of Muller Gate..........................................36
  4.4 Logical Implementation of a Self-Timed Ring Oscillator ...............39
  4.5 Experimental Results ..............................................................42
  4.6 Conclusion ..............................................................................44

5 STRO-PUF: Self-Timed Ring Oscillator based PUF..........................45
  5.1 Introduction ..............................................................................45
  5.2 Architecture of STRO-PUF .......................................................46
  5.3 Implementation of STRO-PUF ....................................................47
  5.4 Experimental Analysis ..............................................................50
    5.4.1 Analysis of Output Frequencies ..........................................51
    5.4.2 Analysis of Uniqueness of STRO-PUF ..................................54
    5.4.3 FPGA Authentication using STRO-PUF ...............................59
    5.4.4 Reliability Enhancement with STRO-PUF ............................60
  5.5 Conclusion ..............................................................................61
6 Conclusion ........................................................................................................................................63
   6.1 Conclusion ..................................................................................................................................63
   6.2 Future Directions .........................................................................................................................64
References ........................................................................................................................................73

A Source Codes ....................................................................................................................................73
   A.1 VHDL Code for a Self-Timed Ring (STR) ..................................................................................73
   A.2 VHDL Code for STRO-PUF .....................................................................................................78
   A.3 UCF File for Mapping STRO-PUF in a Desired Region .........................................................83
   A.4 Uniqueness Analysis of STRO-PUF for 16-bit Response ..........................................................88
   A.5 Uniqueness Analysis of STRO-PUF for 256-bit Response ......................................................92
List of Tables

2.1 Different types of PUFs ..................................................................................11
4.1 LUT mapping of reset Muller gate ..................................................................38
4.2 LUT mapping of set Muller gate ......................................................................38
4.3 Frequency values for implemented asynchronous ring oscillators ...............44
5.1 16-bit STRO-PUF responses ..........................................................................55
5.2 256-bit STRO-PUF responses ..........................................................................57
5.3 Comparing responses with dependent bits and independent bits ...................58
5.4 Uniqueness results for FPGA-based PUFs .....................................................58
List of Figures

2-1 Optical PUF .................................................................................................. 12
2-2 An Arbiter PUF delay circuit ..................................................................... 15
2-3 Ring Oscillator PUF .................................................................................... 16
2-4 RO-PUF generating a single response bit .................................................... 16
2-5 Anderson PUF ............................................................................................. 18
2-6 SRAM Cell ................................................................................................... 18
2-7 Butterfly PUF cell ...................................................................................... 19
2-8 Secret key generation using PUF ................................................................. 21
2-9 HRNG using PUF ....................................................................................... 22
3-1 Synchronous circuit .................................................................................... 26
3-2 Asynchronous circuit .................................................................................. 26
3-3 Abstract data-flow view of an asynchronous circuit.................................... 26
3-4 Standard Muller gate and its truth table .................................................... 28
3-5 Implementations of Muller C-element ....................................................... 29
3-6 Three stage pipeline and ring .................................................................... 29
3-7 An N-stage self-timed ring ......................................................................... 29
3-8 Token-bubble propagation ......................................................................... 31
3-9 Burst mode propagation and evenly-spaced mode propagation ............... 31
4-1 A typical FPGA architecture ....................................................................... 34
   xii
<table>
<thead>
<tr>
<th>Page</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-2</td>
<td>Structure of a typical logic block</td>
</tr>
<tr>
<td>4-3</td>
<td>Spartan-II slice</td>
</tr>
<tr>
<td>4-4</td>
<td>A stage in STR</td>
</tr>
<tr>
<td>4-5</td>
<td>VHDL instantiation of reset Muller gate</td>
</tr>
<tr>
<td>4-6</td>
<td>LUT-based four-stage asynchronous ring oscillator</td>
</tr>
<tr>
<td>4-7</td>
<td>Technology schematic view of 6-stage self-timed ring oscillator</td>
</tr>
<tr>
<td>4-8</td>
<td>Implementation of 6-stage self-timed ring oscillator</td>
</tr>
<tr>
<td>4-9</td>
<td>Placement constraint used to define position of stages of self-timed ring</td>
</tr>
<tr>
<td>4-10</td>
<td>Simulation result of 6-stage STR oscillator with TTBBBB configuration</td>
</tr>
<tr>
<td>4-11</td>
<td>Simulation result of 6-stage STR oscillator with TTTTBB configuration</td>
</tr>
<tr>
<td>4-12</td>
<td>Real output of 6-stage STR oscillator with TTBBBB</td>
</tr>
<tr>
<td>5-1</td>
<td>Architecture of the proposed STRO-PUF</td>
</tr>
<tr>
<td>5-2</td>
<td>Six-stage asynchronous ring oscillator</td>
</tr>
<tr>
<td>5-3</td>
<td>Hard-macro implemented as 6-stage asynchronous ring oscillator</td>
</tr>
<tr>
<td>5-4</td>
<td>Layout view of an STRO-PUF implemented</td>
</tr>
<tr>
<td>5-5</td>
<td>Portion of an STRO-PUF in FPGA Editor</td>
</tr>
<tr>
<td>5-6</td>
<td>PUFs mapped on six different regions</td>
</tr>
<tr>
<td>5-7</td>
<td>PUF outputs in initialization mode and oscillation mode</td>
</tr>
<tr>
<td>5-8</td>
<td>Simulation result of STRO-PUF output frequencies</td>
</tr>
<tr>
<td>5-9</td>
<td>Portion of STRO-PUF output frequencies in a logic analyzer</td>
</tr>
<tr>
<td>5-10</td>
<td>Distribution of frequencies generated by asynchronous ring oscillator</td>
</tr>
<tr>
<td>5-11</td>
<td>Uniqueness Analysis for 16-bit PUF response</td>
</tr>
<tr>
<td>5-12</td>
<td>Uniqueness Analysis for 256-bit PUF response</td>
</tr>
</tbody>
</table>
5-13 FPGA authentication using STRO-PUF ..........................................................59
5-14 Effect of temperature and voltage on oscillator frequencies .............................61
List of Abbreviations

ASIC ...................... Application Specific Integrated Circuits
BPUF ...................... Butterfly PUF
CLB ...................... Configurable Logic Block
CLK ...................... Clock
CRP ...................... Challenge-Response Pair
ECC ...................... Error Correcting Code
EDA ...................... Electronic Design Automation
EMI ...................... Electro-Magnetic Interference
ERAI ..................... Electronics Resellers Association International
FF ...................... Flip-Flop
FPGA ...................... Field Programmable Gate Array
HD ...................... Hamming Distance
HRNG .................... Hardware Random Number Generator
I/O ...................... Input / Output
IC ...................... Integrated Circuit
IP ...................... Intellectual Property
IRO ...................... Inverter Ring Oscillator
ITRS ..................... International Technology Roadmap for Semiconductors
LAB ...................... Logic Array Block
LC ...................... Logic Cell
LE ...................... Logic Element
LUT ...................... Look-Up-Table
MUX ...................... Multiplexer
NIST ...................... National Institute of Standards and Technology
OEM ...................... Original Equipment Manufacturer
PDF                     Probability Density Function
PMF                     Probability Mass Function
PUF                     Physical Unclonable Function

RFID                    Radio Frequency Identification
RO                      Ring Oscillator
RO-PUF                  Ring Oscillator based Physical Unclonable Function
RTL                     Register Transfer Level

SR                      Set / Reset
SRAM                    Static Random Access Memory
STR                     Self-Timed Ring
STRO                    Self-Timed Ring Oscillator
STRO-PUF                Self-Timed Ring Oscillator based Physical Unclonable Function

TRNG                    True Random Number Generator

UCF                     User Constraint File

VHDL                    VHSIC Hardware Description Language
VLSI                    Very Large Scale Integration
List of Symbols

ack.............................. acknowledge signal
B............................... Bubble
C............................... Muller C-element of Muller gate
F............................... Forward input of Muller gate
$f$............................. frequency
MHz............................ Mega-Hertz
$N$............................. Number of stages in a ring oscillator
$N_B$........................... Number of bubbles
ns............................. nano-seconds
$N_T$........................... Number of tokens
$Q$............................. Current output state of Muller gate
$Q'$............................. previous output state of Muller gate
$R$............................. Reverse input of Muller gate
$R'_i$.......................... Response bit from chip $i$ in different environmental conditions
$R'_{i,y}$......................... $y^{th}$ sample of $R'_i$
$R_i$............................ Response bit from chip $i$
SR............................. Set/Reset Signal
T............................... Token
$T_V$........................... Target value
Chapter 1

Introduction

1.1 Context and Motivation

FPGAs are being increasingly used in products and systems of all kinds; FPGAs often form the core of any system. FPGAs are dominating a wide range of application areas including military, defense, space, automotive and consumer electronics. This rise in both the usage and importance of FPGAs in systems makes protecting the IP contained in FPGAs as important as protecting the data processed by the FPGA. There has been a growing concern over the security attributes of FPGAs regarding protecting and securing information processed within them, protecting designs during distribution and protecting intellectual property rights [1]. The design security is often thought of in terms of protecting Intellectual Property (IP); however, potential losses extend beyond just the financial. With the increasing use of programmable logic beyond commercial markets to avionic, space and military applications, design security takes on the additional aspects of safety and national security.

As FPGAs are being used in more applications that require security features, attackers look for vulnerabilities and developers for defenses. Cloning, overbuilding, reverse engineering and tampering are the major security vulnerabilities of FPGAs. These
threats can have far-reaching consequences ranging from counterfeiting to espionage, and are faced by corporations and governments alike [2]. Cloning is making an illegal replica of an original design without understanding the exact details of the design. The attacker simply considers the original design as a black-box to copy the design to resell without making an investment in the initial design effort. Cloning not only harms the revenue of the Original Equipment Manufacturer (OEM) but also affects the OEM’s reputation because of the poor quality of cloned products. Overbuilding is the easiest form of design theft, which occurs when a subcontractor builds more units than have been ordered for fabrication by an OEM. The overbuilt units produced are identical to the originals, which makes identification difficult. Reverse engineering is making functionally equivalent designs from an existing design by probing details of the original design. An adversary can use this information to either develop effective countermeasures or to produce similar equipment. In FPGAs, bitstream reversal can transform the encoded bitstream into a functionally equivalent description of the original design. Tampering is an attempt to gain unauthorized access to an electronic system. Tampering can either be part of a reverse engineering program, or it can have a malicious motive.

Recently, electronic industries have been facing an increased amount of hardware counterfeits. The increased complexity in the supply chain system of electronic components has made counterfeit components easily available in the gray market. These counterfeit components, when assembled into a product or a system, cannot only degrade its performance and reliability but also create safety issues. Increasing incidents have been reported to the Electronics Resellers Association International (ERAI) since 2008. In 2011, there were more than 1,300 counterfeit incidents reported from around the
world. This number is more than double the number reported in 2010 and 2008, and quadruple the number reported in 2009 [3].

Physical Unclonable Function (PUF) [4, 5] provides a means to enhance physical security of Integrated Circuits (ICs) against piracy and unauthorized access. A PUF is used to solve various security issues, such as chip authentication, cryptographic key generation, software licensing, Intellectual Property (IP) protection, and detection and prevention of IC counterfeiting.

Although a Self-Timed Ring (STR) is well studied in many contexts, there has been limited work done in the field of hardware security and hardware cryptography. The work in this thesis is also motivated by the fact that there is no previous work on the FPGA-based implementation of PUFs using asynchronous logic. Self-timed rings are considered robust to environmental variations, [6, 7] and this feature of the self-timed ring oscillator is explored to build robust PUFs that strengthen the PUF responses. The terms ‘asynchronous ring’ and ‘self-timed ring’ are used interchangeably throughout this thesis.

1.2 Contributions

The major contributions of the work described in this thesis are as follows:

- Introduces a Look-Up-Table (LUT) based implementation of asynchronous ring oscillators for PUF design.
- Proposes a novel PUF design approach using self-timed ring oscillators. The proposed PUF is given a name; ‘Self-Timed Ring Oscillator PUF’ (STRO-PUF).
- Experimental analyses are performed on real semiconductor devices. Previous work [8] on an asynchronous PUF was limited to electrical simulations.
1.3 Thesis Outline

This thesis is organized as follows:

Chapter 2 gives an overview of Physical Unclonable Functions (PUFs) including PUF definitions, terminologies related to PUFs, PUF quality measures, different types of PUFs and applications of PUFs.

Chapter 3 gives a brief introduction of asynchronous logic and asynchronous circuits to design a Self-Timed Ring (STR), also called an asynchronous ring. It discusses the structure of a self-timed ring oscillator using Muller C-element and the propagation mode of oscillation in the ring.

Chapter 4 focuses on two major implementations required for the proposed PUF design; LUT-based implementation of Muller C-element and the asynchronous approach to the ring oscillator for implementing the Self-Timed Ring (STR) on FPGAs. This chapter explains the technique for logical implementation of the self-timed ring oscillator using an underlying FPGA architecture.

Chapter 5 discusses the architecture and the detailed implementation of the proposed Self-Timed Ring Oscillator based PUF (STRO-PUF). The experimental analyses are performed to validate the design for calculating PUF uniqueness and analyzing variation in output frequencies of asynchronous ring oscillators.

Finally, Chapter 6 concludes the thesis and presents ideas for future work.
Chapter 2

Physical Unclonable Functions

2.1 Introduction

The security in Integrated Circuits (IC) has become an important issue due to high information security requirements. One of the important aspects of improving the trustworthiness level of semiconductor devices and the semiconductor supply chain is enhancing physical security. These semiconductor devices demand both computational security and physical security. Physical Unclonable Function (PUF) [4, 5] provides a means to enhance physical security of Integrated Circuits (ICs) against piracy and unauthorized access. This chapter discusses PUF definitions, terminologies related to PUFs, PUF quality measures, different types of PUFs and applications of PUFs.

PUFs exploit the inherent delay characteristics of wires and transistors that differ from chip to chip due to manufacturing process variations [9]. These complex physical characteristics of ICs are used to generate unique signatures which are random, unpredictable and difficult to reproduce. A PUF generates a set of responses while stimulated by a set of input challenges. The challenge response relation is defined by complex physical properties of the material, such as process variability of semiconductor devices.
PUFs increase physical security by generating volatile secrets in digital form while the chip is in operation. Secret keys are essential to many security related applications. Storing secrets in a non-volatile memory is not only expensive but can also be an easy target for invasive attacks[1]. A PUF offers an inexpensive and secure approach for generating secret keys. A PUF generates a unique response, or output bits for each challenge, or input bits. This feature of PUF is used to solve various security issues, such as chip authentication, cryptographic key generation, software licensing, Intellectual Property (IP) protection, and detection and prevention of IC counterfeiting.

2.2 PUF Terminologies

2.2.1 Significance of Process Variations

Significant process variations exist in IC fabrication, which makes each IC unique in its delay characteristics [10]. These variations exist die-to-die (inter-die) or within a die (intra-die). Die-to-die parameter fluctuations resulting from lot-to-lot, wafer-to-wafer, and a portion of the within-wafer variations affect every element on a chip equally. Within-die parameter fluctuations consisting of both random and systematic components produce a non-uniformity of electrical characteristics across the chip. These variations occur during various fabrications steps. The lot-to-lot and wafer-to-wafer variations include process temperatures and pressures, equipment properties, wafer polishing, and wafer placement. The within-wafer variations affect both die-to-die and within-die variations. Across a die, device delays vary due to mask variations and placement of dopant atoms in the device channel region. Variability in device parameters, such as effective channel length, threshold voltage and gate oxide thickness results in different characteristics of circuit elements in a chip.
The process variation is becoming more difficult to control in modern Very Large Scale Integration (VLSI) designs due to the continuous reduction in feature size. Process variations in nanometer technologies are becoming more significant for cutting-edge FPGAs. Though FPGA has a regular fabric with replicated layout tiles, the design-dependent systematical variation is significant in advanced technology [11]. A manufacturer resistant PUF can be created by exploiting statistical delay characteristics of the PUF circuit [12].

Most of the PUF designs are based on delay variation of logic and interconnects. The fundamental principle behind the delay based PUF is to compare a pair of identically mapped circuit elements and measure the delay mismatch due to manufacturing process variations. This technique demands identical implementation of two circuit elements being compared. The identical mapping of circuit elements mapping can be achieved by VLSI level placement and routing techniques.

2.2.2 Environmental Variations

The delay of gates and wires depends on junction temperatures which rely on ambient temperatures. The significant variations in the ambient temperatures can result in major variations in delays. Therefore, the ambient temperature is one of the most significant environmental conditions that affect the circuit operating conditions. The impact of varying junction temperatures can be compensated for by using identical components in PUF circuit design. The main problem caused due to environmental variation is the inconsistent result from the same design, which may pose challenges related to robustness. The relative measure of delays can provide robustness against environmental variations including variations in temperatures and voltages. Circuit aging
can also change delay characteristics of a circuit, but its effect is considerably smaller than variations in supply voltage and temperatures.

2.2.3 Challenge-Response Pairs

An input to a PUF is called a challenge and the output a response. An applied challenge and its measured response are generally called a Challenge-Response Pair (CRP). A PUF generates a unique set of output bits, or response, for each secret input set, or challenge. In PUF-based authentication, a CRP database is created from a particular PUF by applying randomly chosen challenges to obtain unpredictable responses. During verification, a challenge from the CRP database is applied to the PUF, and the response produced by the PUF is compared with the corresponding response from the database.

2.3 Sources of Noise

The PUF circuit can have three major sources of randomness from its manufacturing to its usage; noise due to the manufacturing process, local noise and environmental noise [13].

2.3.1 Noise due to Manufacturing Process

Manufacturing process noise is due to variations in silicon layers during various steps in the manufacturing processes. This noise is specific to each IC. An ideal PUF is built to extract the maximum information related to manufacturing process noise to uniquely identify a circuit or device.

2.3.2 Local Noise

Local noise arises when the circuit is in operation. This noise is due to the random thermal motion of charge carriers. Local noise should be minimized to decrease intra-
chip variation for PUF designs. However, local noise can be a good source of randomness for random number generators.

2.3.3 Environmental Noise

Environmental variations such as temperature and power supply voltages variations are the major causes of noise in PUF responses. This environmental noise can disrupt the consistency in PUF responses and increase the intra-chip variations, which reduces the robustness of PUF design.

2.4 Measure of Quality

The metrics to evaluate the basic PUF functions define the trustworthiness of the PUF. The quality factor of a PUF is measured in terms of its uniqueness, reliability and resiliency [9, 14].

2.4.1 Uniqueness

Uniqueness is the estimation of how uniquely a PUF can distinguish different chips based on the generated response. The uniqueness factor is the measure of inter-chip variation, which gives information on the number of PUF output bits that are different between two different PUFs. The uniqueness of a PUF is estimated by the average inter-die Hamming Distance (HD) over a group of chips. It quantifies the Hamming distance of PUF responses that are provided with the same input challenge. It is characterized by the Probability Mass Function (PMF) or Probability Density Function (PDF) of Hamming distances, where PUFs have PDF or PMF curves that are centered at half the number of response bits. For binary strings, a Hamming distance between any two strings of equal length is the number of bits that are different in the two strings.
Let \((i, j)\) be a pair of chips with \(i \neq j\) and \(R_i\) (respectively, \(R_j\)) the \(n\)-bit response of chip \(i\) (respectively, chip \(j\)). The first metric is the average inter-die Hamming distance among a group of \(k\) chips and is defined as [14]:

\[
\frac{2}{k (k - 1)} \sum_{i=j}^{k-1} \sum_{j=i+1}^{k} \frac{HD (R_i, R_j)}{n} \times 100 \%
\]

If the PUF produces uniformly distributed independent random bits, i.e. if each binary response bit of a PUF has an equal probability of producing a ‘0’ or a ‘1’, then the inter-chip variations should be 50% on average. Truly random bits are produced if only the random process variation exists.

2.4.2 Reliability

Reliability indicates the reproducibility of the PUF outputs. Reliability gives information on how many PUF output bits are changed when regenerated from the same PUF with or without environmental variations. The responses for an ideal PUF are expected to be consistent; however, factors such as variation in temperature, supply voltage fluctuations and errors due to thermal noise affect the reproducibility of the PUF responses. Reliability is the measure of consistency or stability of the PUF output responses, when the responses are subjected to varying environmental conditions such as variations in power supply voltages and temperature, and the same input challenge. Since, the responses being compared are from generated from the same chip; this variation is also called as intra-chip or intra-die variations.

An \(n\)-bit reference response \((R_i)\) is extracted from chip \(i\) at normal operating conditions. The same \(n\)-bit response is extracted from the same PUF at a different
operating condition with response bits $R_i$. Let, $R_i, y$ be the $y^{th}$ sample of $R_i$. Then, the average intra-die HD over $x$ samples for the chip $i$ is defined as [14]:

$$\frac{1}{x} \sum_{y=1}^{x} \frac{HD (R_i, R_i', y)}{n} \times 100\%$$

2.2

The lower value of the average intra-die HD factor results in more reliable PUF responses. The intra-chip variations for an ideal PUF should be $0\%$.

2.4.3 Resiliency

Resiliency of a PUF is the ability of the PUF to prevent an adversary from revealing the PUF secrets. This is the measure of resiliency against attack or security.

2.5 PUF Classifications

PUFs can be categorized based on their construction properties, operation principle and from a security point of view. Table 2.1 summarizes various PUFs under different categories.

<table>
<thead>
<tr>
<th>Categories</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-electronic PUF</td>
<td>Optical PUF [15], Acoustical PUF [16]</td>
</tr>
<tr>
<td>Electronic PUF</td>
<td>Coating PUF [17], Power Distribution PUF [18]</td>
</tr>
<tr>
<td>Delay-based PUF</td>
<td>Arbiter PUF [5], Ring Oscillator PUF [9], Glitch PUF [19], Anderson PUF [20]</td>
</tr>
<tr>
<td>Memory-based PUF</td>
<td>SRAM PUF [21], Butterfly PUF [22], Flip-flop PUF [23]</td>
</tr>
</tbody>
</table>
2.5.1 Non-electronic PUF, Electronic PUF and Silicon PUF

On the basis of construction and operation principles, PUFs can be categorized into three categories; non-electronic PUFs, electronic PUFs and silicon PUFs [24].

Non-electronic PUFs refer to those with PUF-like properties whose construction and/or operation is inherently non-electronic. Their PUF-like behavior is based on non-electronic technologies or materials such as the random fiber-structure of a sheet of paper or the random reflection of the scattering characteristics of an optical medium. For example, optical PUFs based on transparent media as proposed in [15] are physical one-way functions. Figure 2-1 shows the basic implementation of the Optical PUF. The CRP, consisting of the laser orientation and the resulting hash, is saved in a public database for later use.

![Figure 2-1: Optical PUF [15]](image)

In electronic PUFs, the basic operation consists of an analog measurement of an electric or electronic quantity such as power, resistance and capacitance. An example of
an electronic PUF is the coating PUF [17], which considers the randomness of capacitance measurements in comb-shaped sensors in the top metal layer of an IC.

Silicon PUFs [4] exhibit PUF behaviors which are embedded on a silicon chip. Silicon PUFs are based on the hidden timing and delay information of ICs. A complex integrated circuit can be represented as silicon based PUF, which helps in identifying and authenticating individual ICs. Silicon PUFs can be implemented as a hardware building block in cryptographic implementations. Silicon PUFs exploit manufacturing process variations in integrated circuits with identical masks to uniquely characterize each IC. Silicon PUFs are of particular interest for security solutions, and they are widely studied as a major type of PUF. Delay-based PUFs and memory-based PUFs are considered silicon PUFs.

2.5.2 Strong PUF and Weak PUF

The distinction between strong PUFs and weak PUFs is explained based on the security properties of their challenge-response behavior [25]. A PUF is considered a strong PUF; if it has a large number of CRPs such that an attack based on exhaustively measuring the CRPs only has a negligible probability of success. For a strong PUF, it is infeasible to build an accurate model of the PUF based on observed CRPs. If the number of CRPs is small, then it is considered a weak PUF.

2.5.3 Intrinsic PUF and Non-intrinsic PUF

Another classification based on PUFs construction properties are intrinsic PUFs and non-intrinsic PUFs. The intrinsic PUF was initially proposed by Guajardo et al. in [21]. In intrinsic PUFs, its evaluations are performed internally by embedded measurement equipment, and its random instance-specific features are implicitly
introduced during the manufacturing process. All silicon PUF based on random process variations occurring during the manufacturing process of silicon chips, are intrinsic PUFs. These silicon PUFs include both delay-based PUFs and memory-based PUFs.

The non-intrinsic PUFs are externally evaluated and their randomness features are explicitly introduced. Optical PUF and Coating PUF are the types of non-intrinsic PUFs.

2.6 PUF Circuits

PUFs have drawn considerable attention over the past couple of years, making them one of the potential areas in the field of hardware security and cryptography. There have been various PUF techniques proposed for on-chip implementations; on both Application Specific Integrated Circuits (ASICs) and FPGAs. Since this thesis is about the FPGA-based PUF implementation, the discussion is limited to those techniques that have been implemented on FPGAs.

2.6.1 Delay-based PUF

2.6.1.1 Arbiter PUF

Arbiter PUF is the first silicon PUF to be proposed [5]. Arbiter PUF is based on a delay-based circuit consisting of a parallel multiplexer chain and an arbiter. Depending on the challenge bits, the skew in propagation delay between the two paths due to process variations is detected by an arbiter which latches out either logic ‘0’ or logic ‘1’. The two delay paths are simultaneously excited and make the transition race against each other. The arbiter block, which is simply a latch or a flip-flop, at the output determines which rising edge arrives first and sets its output to ‘0’ or ‘1’ depending on the winner. If the racing paths are symmetric or identical in layout and the arbiter is not biased to either
path, the response is equally likely to be ‘0’ or ‘1’ regardless of the challenge bits. The output is determined only by the statistical delay variation due to process variations.

Figure 2-2 shows a silicon PUF delay circuit. The circuit has multiple-bit input and computes a one-bit output based on the relative delay difference between two paths with identical layout length. Arbiter PUF demands careful layout and routing for identical mapping of the logic, which is quite difficult, especially in the case of FPGA.

![Figure 2-2: An Arbiter PUF delay circuit [9].](image)

### 2.6.1.2 Ring Oscillator PUF

The Ring Oscillator (RO) PUF consists of several identically mapped delay loops, or ring oscillators, each of which oscillates with unique frequency due to manufacturing process variations [9]. Each input challenge selects a pair of oscillator for comparison in order to generate a response bit. A set of input challenges are given to PUF, which selects a fixed sequence of oscillator pairs to generate a fixed number of response bits. The frequency differences are determined by process variations if all the oscillators are identically laid-out, which results in equal probability of getting ‘1’ or ‘0’ as a response bit if random variation exists. The ease of duplicating a ring oscillator using hard-macros
features has made its implementation more popular in FPGAs. Figure 2-3 and Figure 2-4 illustrate the structure of RO-PUF.

A configurable ring oscillator has been proposed in [26] to improve reliability in an RO-PUF. The authors have shown that an RO-PUF requires careful design decisions to avoid the systematic process variations; and the placement techniques and the selection of ring oscillator pairs significantly improves the PUF uniqueness.
2.6.1.3 Glitch PUF

In a combinational logic, there exists a time difference between output changes from an input change, i.e. it takes some time before the output is settled to its steady-state value. These unintended transitions in signals are called glitches. The occurrence of glitches is determined by the differences in delay of the different logical paths from the inputs to an output signal.

The glitch PUF proposed in [19] exploits glitch waveforms that behave non-linearly from delay variation between gates. It consists of an on-chip high-frequency sampling of the glitch waveform and a quantization circuit which generates a response bit based on the sampled data. The operation sequences of the glitch PUF are as follows:

- Data input to a random logic
- Acquisition of glitch waveforms at the output
- Conversion of the waveforms into response bits

The Anderson PUF proposed in [20] generates a response bit depending on the presence or absence of glitch. This design is targeted especially for FPGA-based implementations. It consists of custom logical circuits implementing shift registers and carry-chain multiplexers. Figure 2-5 shows a basic Anderson PUF. The shift registers are implemented using a Look-Up-Table (LUT) and are initialized with bit strings that are inverses of each other. The two LUTs generate square waves that are 180 degrees out of phase. Due to the process variations in the LUTs and the multiplexers, the propagation delay from the input to the output will vary from LUT to LUT. When an LUT’s outputs are sufficiently out of phase, it produces a glitch at the output, which can be captured by a flip-flop. The presence or absence of the glitch determines the PUFs output bit. Anderson
PUF is also analyzed using the concept of neural network and artificial intelligence [27-29].

2.6.2 Memory-based PUF

2.6.2.1 SRAM PUF

Static Random Access Memory (SRAM) is a volatile digital memory cell, each capable of storing a single bit. SRAM memories are available in almost every computing device including FPGAs, and they can be used as an intrinsic PUF. It is bi-stable and can be realized with two cross-coupled inverters as illustrated in Figure 2-6.
SRAM PUF proposed in [21] is an FPGA intrinsic PUF based on random initial states of SRAM cells. Every cell contains a certain degree of mismatch between the two halves of the cross-coupled circuit. The random physical mismatch in the cell, caused by manufacturing variability, determines the power-up behavior. When the cell is powered on, it tends to attain both the stable stages. The power-on condition forces a cell to ‘0’ or ‘1’ during power-up depending on the sign of the mismatch. But, which power-up state a cell prefers is random and not known in advance, and this random behavior can be used as a PUF response.

2.6.2.2 Butterfly PUF

The Butterfly PUF (BPUF) is proposed in [22] to overcome the drawbacks of an SRAM PUF. The disadvantage of intrinsic SRAM PUFs is that not all FPGAs support uninitialized SRAM memory. In most of the FPGAs, all SRAM cells are enabled hard reset to zero directly after power-up and hence all the randomness is lost. Also, the SRAM PUFs require device power-up to enable the response generation.

Figure 2-7: Butterfly PUF cell
The construction of a butterfly PUF is similar to the SRAM PUF except BPUF consists of a cross-coupled latch instead of an inverter. A butterfly PUF cell is depicted in Figure 2-7. A BPUF cell can be brought to a floating or unstable state before allowing it to settle to one of the two possible stable states. Using the clear/preset functionality of the latches, an unstable state can be introduced after which the circuit converges back to one of the two stable states. The preferred stable state of a butterfly PUF cell is determined by the physical mismatch between the latches and the cross-coupling wires.

2.7 PUF Applications

Some of the major PUF applications proposed so far are as follows:

- Low-cost device authentication [9]

  As the PUF output is unique and unpredictable for each IC, PUF can be used for device identification and authentication. The PUF outputs can be stored in a database and compare that output with a re-generated signature later. The set of challenge-response pairs act as the lock and PUFs act as the key. When a key is presented to a lock, the lock queries the key for the response to a particular challenge. The lock opens only when the correct key from the database responds.

- Cryptographic key generation [9]

  Due to the presence of noise, the PUF outputs are likely to vary slightly on every evaluation. In order to use PUF outputs as cryptographic keys, the outputs are required to undergo error correction process and key generation process. With error correction process, which contains initialization and re-generation, PUF can consistently produce the same result despite significant environmental changes. During initialization step, PUF output is generated and the error correcting syndrome for that output is computed and
saved for later. The syndrome is the information that allows correcting bit-flips in re-generated PUF outputs. In re-generation phase, the PUF uses the syndrome from the initialization step to correct any changes in the PUF output. The key generation process converts the PUF output into cryptographic keys.

*Figure 2-8: Secret key generation using PUF*

- Memoryless secret key storage [9]

In current practice, secret keys are stored in a non-volatile memory for cryptographic primitives. Managing secrets in a memory in a secure way is difficult and expensive. Storing secrets in a non-volatile memory is also vulnerable to invasive attacks. PUF can generate volatile secret keys for cryptographic applications. PUFs increase the physical security by generating volatile secret keys in digital form when the chip is operating.

- Hardware Random Number Generator (HRNG) [30]

Hardware random number generator extracts randomness directly from a complex physical source. HRNG accepts an incoming request for a random output and produces an output using an iterative process for generating a challenge in order to give unpredictable results. An unpredictable challenge is saved in local registers. Once a
suitable challenge is found, a post-processing step is applied to remove bias and extract randomness from the bit ordering. The National Institute of Standards and Technology (NIST) test results carried out indicate that a PUF can be used as a reasonably good hardware random number generator with low area overhead.

![PUF Circuit Diagram]

Figure 2-9: HRNG using PUF

- **Software licensing [12]**

  A piece of code can be made to run only on a chip that has a specific identity defined by a PUF. This prevents the execution of pirated code.

- **Intellectual Property (IP) protection [21]**

  PUFs provide IP protection of FPGAs based on public key cryptography. The major advantage of using public-key based protocol is that it allows the design in which the private key is always stored in a FPGA. As PUFs implemented on FPGAs are intrinsic to the FPGAs, it provides better security.
- PUF-based Radio Frequency IDentification (RFID) tags for anti-counterfeiting [31]

A RFID-tag can be made unclonable by linking it inseparably to a PUF.
Chapter 3

Self-Timed Rings

3.1 Introduction

On-chip digital oscillators are ubiquitous in many IC designs. They are considered a key component in many applications including PLLs, frequency synthesizers and clock recovery systems. Oscillators are also an essential block for many cryptographic applications such as on-chip TRNGs [32, 33] and PUFs [9, 14]. This chapter discusses the Self-Timed Ring (STR), also called as asynchronous ring, as an alternative approach to standard inverter ring oscillator.

3.2 Asynchronous Circuits

Asynchronous circuits, or self-timed circuits, use handshaking between their components in order to perform the necessary synchronization, communication, and sequencing of operations. Asynchronous circuits have shown many interesting potentials including low power consumption, high operating frequency, less EMI (Electro-Magnetic Interference), less noise, robustness towards variation in supply voltage, temperature, and fabrication process parameters, better modularity for easier reuse of components, and no clock skew problems [6]. However, asynchronous circuits are not yet matured enough to
be accepted openly in the industries, especially due to the lack of suitable Electronic Design Automation (EDA) tools for asynchronous designs. The acceptance of asynchronous technology by the semiconductor industries strongly depends on the availability of synthesis tools and the possibility to prototype a design on standard FPGAs.

The development of synchronous circuits currently dominates the semiconductor design industry. However, there are major limiting factors to the synchronous, clocked approach, including the increasing difficulty of clock distribution, increasing clock rates, decreasing feature size, increasing power consumption, timing closure effort, and difficulty with design reuse. Asynchronous circuits can offer a better solution to address these issues. As the demand continues for designs with higher performance, higher complexity, and decreased feature size, asynchronous paradigms will become more widely used in the industry, as evidenced by the 2003 and 2007 International Technology Roadmap for Semiconductors’ (ITRS) prediction of a likely shift from synchronous to asynchronous design styles in order to increase circuit robustness, decrease power, and alleviate many clock-related issues. The 2008 ITRS shows that asynchronous circuits account for 11% of chip area in 2008, compared to 7% in 2007, and estimates they will account for 23% of chip area by 2014, and 35% of chip area by 2019 [34].

3.3 Asynchronous Logic

Logic design, in general, consists of a separate computation part and storage part. Computation takes place in a combinational block or a functional block; whereas storage takes place in flip-flops, or registers, or latches, although they may exist combined or separately. In synchronous logic, a global time reference, or a clock, controls activity to
synchronize the entire functional block in a circuit, or a system. Asynchronous logic uses a local handshaking protocol to communicate among different modules, or functional blocks. Local handshake between combinational blocks is also called asynchronous control. Figure 3-1 and Figure 3-2 shows the synchronous and asynchronous communication to control the events.

An asynchronous circuit can be represented as a static data-flow structure. The static data-flow structure represents a high-level view of asynchronous design that is equivalent to Register Transfer Level (RTL) in synchronous design. The data is copied from one register to the next along the path through the circuit. The handshaking between
the registers controls the data. The data and handshake signals connecting one register to
the next can be viewed as a handshake channel, or a link, as in Figure 3-3. The arrows
represent channels or links consisting of request, acknowledge and data signals. The
handshaking protocol is the basis of following sequencing rules of asynchronous circuits
[6, 35]:

- a module starts the computation, if and only if, all the data required for the
  computation are available,
- as far as the result can be stored, the module releases its input ports,
- it outputs the result in the output port, if and only if, the port is available.

### 3.3.1 Muller C-element

The Muller C-element or Muller gate is a fundamental primitive for building
asynchronous logic and implementing the synchronization required by most handshaking
protocols. Figure 3-4 shows a Muller gate representation and its truth table. ‘F’ and ‘R’
represent forward and reverse input respectively, ‘Q’ and ‘Q’‘ represent current output
state and previous output state respectively. Figure 3-5 shows transistor level and logic
level implementation of Muller gate. Muller gate copies its input values to output if its
inputs are matched, otherwise it will hold the previous state. In the case of Muller gate
with inverted reverse input, it will copy forward input values to output if its inputs differ
in states, otherwise it will hold the previous states.
3.4 Self-Timed Rings

Rings are the backbone structures of circuits that perform iterative computations. One can turn a pipeline into a ring by looping data from its output back around to its input [36]. Figure 3-6 shows a three stage pipeline and the pipeline with its output connected around to its input to form a ring. If the stages in the ring are all self-timed and initialized with input data, then the ring will iterate under self-timed control. Self-timed circuits use handshake protocols to control the sequencing of operations. In a self-timed ring, events propagate between adjacent stages according to a simple request/acknowledge handshake. These handshake signals replace the clocks of synchronous designs.

3.4.1 Self-Timed Ring Structure

Muller C-element or Muller gate is an integral part of self-timed rings. Each stage of a self-timed ring consists of a Muller gate and an inverter [37]. A standard N-stage self-timed ring is depicted in Figure 3-7 [38].
Figure 3-5: Implementations of Muller C-element

Figure 3-6: Three stage pipeline (top) and a ring (bottom)

Figure 3-7: An N-stage self-timed ring
3.4.2 Token and Bubble Propagation

The temporal behavior of the self-timed ring can be explained on the basis of the token-bubble abstraction model. From micro-pipeline point of view, a token usually represents the presence of data in a stage, whereas a bubble represents an empty stage ready to accept new data. A stage is said to have token if its output is not equal to its input. Similarly, a stage is said to have bubble if its output is equal to its input. If $Q_i$ and $Q_{i+1}$ represent output for stage $i$ and stage $i+1$ respectively, then token (T) and bubble (B) may be represented as:

Token: if $Q_i \neq Q_{i+1}$ and Bubble: if $Q_i = Q_{i+1}$.

Token-bubble configuration also represents the output states of each stage in a ring. For example, for a ring having TTBBBB configuration, the stage output is either “101111” or 010000”. A token propagates from stage $i$ to next stage $i+1$ if, and only if, the next stage $i+1$ contains a bubble. Similarly, a bubble propagates from stage $i+1$ to previous stage $i$ if, and only if, the previous stage $i$ contains a token. Figure 3-8 illustrates propagation of tokens and bubbles in a self-timed ring. For example, with initial ring configuration as TTB (101 or 010), propagation occurs as:

TTB (101) → TBT (011) → BTT (110) → TTB (101)

An STR will create an oscillation only if the following conditions are satisfied[7, 39]:

- $N \geq 3$ and $N = N_T + N_B$, where $N$ is the number of stages in an STR with $N_T$ number of tokens and $N_B$ number of bubbles.
- $N_B > 1$
- $N_T$ is a positive even number
The oscillation depends on process variability and the initial stages of the ring defined by $N_T$ and $N_B$. STR provides two different propagation modes; burst mode and evenly-spaced mode, as shown in Figure 3-9. In burst mode, the tokens get together to form a cluster that propagates all around the ring. In evenly-spaced mode, the tokens get distributed evenly around the ring with constant spacing.

![Token-bubble propagation](image)

**Figure 3-8: Token-bubble propagation**

![Burst mode propagation](image)

**Figure 3-9: Burst mode propagation (top) and evenly-spaced mode propagation (bottom)**

### 3.4.3 Jitter in Inverter RO and Self-Timed RO

Inverter Ring Oscillators (IROs) and self-timed ring oscillators exhibit thermal noise [8]. This thermal noise is called jitter in time-domain and phase noise in frequency domain. Self-timed ring oscillators and inverter ring oscillators differ in the way jitter accumulates. There are two major jitter sources in FPGAs; local Gaussian jitter and global deterministic jitter [39, 40].

Local Gaussian jitter is the source of randomness. For FPGA-based implementation, where each stage of ring oscillators is implemented in a single Look-Up-
Table (LUT), each stage of ring oscillators is considered source of the local Gaussian jitter. In inverter ring oscillators, oscillation period is defined by two loops of a single token around the ring and the jitter accumulates from the number of crossed stages. Whereas, in asynchronous ring oscillators, several tokens propagate around the ring and the oscillation period is defined by the elapsed time between successive tokens. Each token crossing a stage experiences varying delay characteristics due to local Gaussian jitter contribution of the stage. So, the period jitter in STRs is mostly composed of the jitter generated locally in the ring stage. This provides better robustness against noise instabilities caused by jitter in inverter ring oscillators in PUF design.

Global deterministic jitter is due to the non-random variations in delay characteristics caused from external environmental variations. The global deterministic jitter accumulates linearly throughout the ring in IROs. In STR oscillators, several events propagate simultaneously, so deterministic jitter affects each event in the same way rather than the whole ring structure. This gives increased robustness in self-timed ring oscillators than inverter ring oscillators.
Chapter 4

Asynchronous Approach to Ring Oscillator for FPGA-based PUF Design

4.1 Introduction

Recent development and advancement in design and process technology has made Field Programmable Gate Array (FPGA) a key component in most of the electronic systems. FPGAs are semiconductor devices consisting of matrix of Configurable Logic Blocks (CLBs), which are interconnected using programmable interconnects. FPGA is dominating a wide range of application area including military, defense, space, automotive and consumer electronics. It is believed that FPGA may emerge as a potential security platform due to their desirable features including flexibility, rapid time-to-market, and post-silicon validation of the functionality. There has been growing concern over the security attributes of FPGAs regarding protecting and securing information processed within it, protecting designs during distribution and protecting intellectual property rights [1].

This chapter mainly discusses two major implementations required for the proposed STRO-PUF design; LUT-based implementation of Muller C-element and the
asynchronous approach to the ring oscillator for implementing Self-Timed Ring (STR) on FPGAs.

4.2 FPGA Architecture

The typical FPGA architecture consists of an array of logic blocks, Input / Output (I/O) pads and routing channels. The array is surrounded by programmable I/O blocks, which provides external interface to the FPGA. The logic block is also called as Combinational Logic Block (CLB) or Logic Array Block (LAB) depending on vendors. Xilinx and Altera are the two major FPGA vendors in the current market. The detail architecture of FPGAs differs from one vendor to another vendor; however, the typical FPGA architecture is shown in Figure 4-1.

Figure 4-1: A typical FPGA architecture
Logic blocks implement logic functions. They form the basic computation and storage element of digital logic functions on FPGA. The logic block consists of Logic Cells (LCs), which is also called as Logic Elements (LEs) or a slice. The typical logic cell consists of Look-Up-Table (LUT) and storage elements such as latches or flip-flops. The input signals consist of inputs to LUTs and a clock input; and can have registered or unregistered output. The basic structure of a logic block is shown in Figure 4-2.

![Figure 4-2: Structure of a typical logic block](image)

4.2.1 Architecture of Spartan-II

The proposed design is implemented using Xilinx XC2S100 FPGA device. This section describes the overview of a Spartan-II family architecture, which helps in implementing the STR on the FPGA. The particular XC2S100 device has 20 rows by 30 columns CLBs, which totals 600 CLBs and has 2700 logic cells [41].

The basic building block of the Spartan-II FPGA CLB is the Logic Cell (LC). An LC includes a 4-input function generator, carry logic, and a storage element. Each Spartan-II FPGA CLB contains four LCs, organized in two identical slices. Each CLB consists of two identical slices. A Spartan-II slice is shown in Figure 4-3. The function generators are implemented as 4-input LUTs.
4.3 LUT Implementation of a Muller Gate

Every Look-Up-Table (LUT) implements a Boolean logic equation, which is defined by an INIT attribute. The INIT attribute defined with an appropriate hexadecimal digits is attached to the LUT inputs to specify its logical function [42]. The INIT
parameter for the LUT primitive defines the logical values of the LUT. This value is zero by default, which drives the output to a zero regardless of the input values. The LUT can be loaded with custom hexadecimal values, defined by INIT attribute, to perform a particular logical function.

A self-timed ring requires its initial states to be loaded with required configuration of tokens and bubbles, which can be defined by assigning the output of each stage with either ‘0’ or ‘1’. A Muller gate with a set/reset feature (as shown on the left side of Figure 4-4) is used to force its output to either set or reset as desired. A Muller gate with set input is called set Muller gate and a Muller gate with reset input is called reset Muller gate. The set Muller gate forces its output to ‘1’ and reset Muller gate forces its output to ‘0’ during the initialization process.

![Muller gate with set/reset option](image)

Figure 4-4: A stage in STR. Muller gate with set/reset option (left). LUT mapped as Muller gate (right) for FPGA implementation

A 4-bit LUT with general output is considered in the implementation to define STR stages. Figure 4-4 shows a single stage of a self-timed ring oscillator for its implementation in LUT. One of the inputs is configured as a Set/Reset (SR) signal, which is responsible for setting stage output value at either ‘0’ or ‘1’. The remaining three inputs are configured as forward input (F), reverse input (R) and feedback (Q’).
A common technique to determine the desired INIT value for realizing a logical function with LUT is using a truth table. The logical function of set Muller gate and reset Muller gate is mapped in the Table 4.1 and Table 4.2. The custom hexadecimal digits to define INIT attribute are obtained by grouping the output bits. The INIT attribute can be obtained by reading the output states in groups of four from the bottom-up fashion and converting them into hexadecimal characters. From the tables below, the INIT attribute obtained for reset Muller gate and set Muller gate are “00B2” and “FF02” respectively. Figure 4-5 shows the VHDL instantiation of reset Muller gate using a 4-input LUT with INIT attribute.

Table 4.1: LUT mapping of reset Muller gate. INIT = > x“00B2”

<table>
<thead>
<tr>
<th>I3 = SR</th>
<th>I2 = F</th>
<th>I1 = R</th>
<th>I0 = Q’</th>
<th>O = Q</th>
<th>INIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>“0010” = 2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>“1011” = B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>“0000” = 0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>“0000” = 0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: LUT mapping of set Muller gate. INIT => x“FFB2”

<table>
<thead>
<tr>
<th>I3 = SR</th>
<th>I2 = F</th>
<th>I1 = R</th>
<th>I0 = Q’</th>
<th>O = Q</th>
<th>INIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>“0010” = 2</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>“1011” = B</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>“1011” = B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

38
Logical Implementation of a Self-Timed Ring Oscillator

The proposed PUF design is a logic-based design, which uses asynchronous ring oscillators instead of basic inverter ring oscillators. The design is especially targeted for LUT-based FPGAs. Each stage in a ring is mapped in an LUT to perform a Muller gate function. An asynchronous ring oscillator can be constructed by replicating each stage of

```vhdl
Library UNISIM;
use UNISIM.vcomponents.all;
-- LUT4: 4-input Look-Up Table with general output
-- Xilinx HDL Libraries Guide, version 10.1.2
-- Defining Reset Muller gate
LUT4_inst : LUT4
generic map (
  INIT => X"00B2")-- FF2B2 for set Muller gate
port map (
  O => Q, -- LUT general output
  I0 => Q_bar, -- LUT feedback input
  I1 => R, -- LUT reverse input
  I2 => F, -- LUT forward input
  I3 => SR -- LUT set/reset input
);
-- End of LUT4_inst instantiation
```

Figure 4-5: VHDL instantiation of reset Muller gate.

4.4 Logical Implementation of a Self-Timed Ring Oscillator

The proposed PUF design is a logic-based design, which uses asynchronous ring oscillators instead of basic inverter ring oscillators. The design is especially targeted for LUT-based FPGAs. Each stage in a ring is mapped in an LUT to perform a Muller gate function. An asynchronous ring oscillator can be constructed by replicating each stage of
the ring described in Figure 4-4 to form a ring structure, as illustrated in Figure 3-7 in Chapter 3. The ring should be designed to meet the oscillation conditions described in Chapter 3. It is necessary to initialize ring stages, satisfying the oscillation conditions, before oscillation occurs. The number and positions of set Muller gates or reset Muller gates, defines the initialization states and the token-bubble states in the ring.

Figure 4-6 depicts a four-stage asynchronous ring oscillator implemented using LUTs. A common signal ‘SR’ is connected to every stages of the ring. SR signal controls the initialization and oscillation of the ring oscillator. In other words, SR switches the self-timed ring oscillator between initialization mode and oscillation mode. For the purpose of this design, initialization occurs when SR = ‘1’ and oscillation occurs when SR = ‘0’.

![Figure 4-6: LUT-based four-stage asynchronous ring oscillator](image)

The placement constraints [43] are used in the coding to ensure each stage of the ring is mapped in a separate LUT. Placement constraints are used to prevent alteration of design mapping, which may be caused by a synthesis tool. Figure 4-7 shows the
schematic view of the implemented 6-stage self-timed ring oscillator with 2T4B configuration and the initial states of “101111”.

Each stage of the ring is mapped in a separate LUT. Since six different LUTs are used for implementing the ring oscillator, three different slices are used, as shown in Figure 4-8. The position of each stage of the self-timed ring oscillator is defined by using placement constraints, as shown in Figure 4-9.

Figure 4-7: Technology schematic view of 6-stage self-timed ring oscillator with 2T4B configuration and the initial states of “101111”. 
4.5 Experimental Results

To observe the oscillatory behavior of a self-timed ring, the design is implemented on XSA board with Xilinx XC2S100 FPGA device. For experimental analysis, the self-timed ring oscillator is implemented with different numbers of stages, and with different spatial configurations. Figure 4-10 through Figure 4-12 below show the oscillation pattern of post-place & route simulation results and the real output tapped.
from a logic analyzer. Table 4.3 shows the frequency observed for different configurations of self-timed ring oscillators.

![Figure 4-10: Simulation result of 6-stage STR oscillator with TTBBBB configuration](image)

![Figure 4-11: Simulation result of 6-stage STR oscillator with TTTTBB configuration](image)

![Figure 4-12: Real output of 6-stage STR oscillator with TTBBBB configuration obtained from a logic analyzer](image)

The oscillation frequency of the ring oscillator depends on the number of events, i.e. number of bubbles or number of tokens; but not on the spatial arrangement or distribution for the same number of tokens and bubbles. From the Table 4.3, it can be observed that the 6-stage ring oscillator with spatial distribution of “TTBBBB” or “TBTBBBB” results in the same frequency. Also, with the different initialization states, the same stage ring oscillator can give different oscillation frequencies. This is one of the
benefits of the self-timed ring to add reconfigurable features within the design. Unlike, conventional inverter oscillator, the oscillator frequency of the asynchronous ring oscillator does not decrease with the number of stages.

<table>
<thead>
<tr>
<th>No. of Stages</th>
<th>N_τ,N_B</th>
<th>Time Period (ns)</th>
<th>Frequency (MHz)</th>
<th>Spatial Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>2T4B</td>
<td>10</td>
<td>100</td>
<td>TTBBBBB, TBTTBBB</td>
</tr>
<tr>
<td>6</td>
<td>4T2B</td>
<td>6.2</td>
<td>169.29</td>
<td>TTTTBB, TTBBBT</td>
</tr>
<tr>
<td>8</td>
<td>2T6B</td>
<td>8.3</td>
<td>120.48</td>
<td>TTBBBBBBB</td>
</tr>
<tr>
<td>8</td>
<td>4T4B</td>
<td>5.9</td>
<td>169.49</td>
<td>TTBBBBBBB</td>
</tr>
<tr>
<td>8</td>
<td>6T2B</td>
<td>8.3</td>
<td>120.48</td>
<td>TTTTTBBB</td>
</tr>
</tbody>
</table>

4.6 Conclusion

The technique for LUT-based implementation of Muller gate to construct a self-timed ring oscillator, or an asynchronous ring oscillator is described in this chapter. The experimental analysis illustrates the oscillation generating from an asynchronous ring oscillator with different configurations.

It is a well known fact that significant process variations exist in IC fabrication, which makes each IC unique in its delay characteristics [11, 44]. The statistical delay variation in transistors and wires across FPGA chips can be exploited through identically laid-out asynchronous ring oscillators. The next chapter discusses the proposed FPGA-based PUF using the self-timed ring oscillator.
Chapter 5

STRO-PUF: Self-Timed Ring Oscillator based PUF

5.1 Introduction

This chapter introduces the implementation of self-timed ring oscillators as a novel PUF approach on FPGAs. The proposed PUF is given a name; ‘Self-Timed Ring Oscillator based Physical Unclonable Function (STRO-PUF)’. Like RO-PUF, the self-timed ring oscillator based PUF generates oscillations of different frequencies when identically mapped on a semiconductor device. These varying frequencies produced by all identically mapped self-timed ring oscillators can be used to generate unique PUF response bits.

Although the self-timed ring is well studied in many contexts, there has been very limited work done in the field of hardware cryptography and the areas of security applications using the concept of asynchronous logic. In [8], the author has initiated PUF implementation using asynchronous ring oscillators to address robustness and entropy. However, the result is limited to electrical stimulation. The work described in this thesis is implemented on real silicon devices. In [39], authors have analyzed a self-timed ring oscillator as the entropy source for the True Random Number Generator (TRNG)
implemented on FPGA. This chapter aims to explore the implementation of asynchronous ring oscillators in PUF design targeting FPGA devices.

5.2 Architecture of STRO-PUF

The proposed PUF architecture is also based on a ring oscillator, but it uses a self-timed ring oscillator instead of a conventional inverter ring oscillator. The architecture of the proposed design for a self-timed ring oscillator based PUF is shown in Figure 5-1. It consists of two groups of identically laid-out self-timed ring oscillators. A Set/Reset (SR) signal is common to all the oscillators present in both groups. The SR signal initializes the states of every ring oscillator in order to create oscillations.

The initialization is done setting SR = ‘1’; SR can be switched back to SR = ‘0’ so that oscillation is created. Each oscillator oscillates with different frequencies due to process variations. Outputs of each oscillator are fed to the multiplexers (MUX) of corresponding groups. Inputs to the PUF are given through a challenge generator, which selects two self-timed ring oscillators from each group. The frequency comparator captures the frequency differences between these two oscillators and generates a single output bit. A frequency comparator consists of two counters counting $T_V$ (target value) periods of two frequencies coming from each MUX. Whichever counter reaches the targeted value of $T_V$ first, the frequency driving that counter is greater than the other. For example, if the frequencies of STROs from group A and group B are $f1$ and $f2$ respectively, then the response bit = 1 if $f1 \geq f2$; otherwise the response bit = 0. A unique set of output responses is generated for each set of input challenges, which is used in identifying a particular device and also used in various cryptographic applications.
5.3 Implementation of STRO-PUF

FPGAs are considered an efficient platform for implementing cryptographic algorithms on hardware. The implementation of PUFs on FPGAs involves significant challenges because it is difficult for a designer to exploit full layout level design techniques, and there is not sufficient information available about the gate level structure of the FPGA fabric. Also, many PUF designs require careful routing symmetry, and this is quite difficult to achieve in FPGA-based design.

A six-stage asynchronous ring oscillator is considered for the purpose of the implemented PUF design. The prototype asynchronous ring oscillator, which is implemented using an LUT-based approach, is shown in Figure 5-2. The details of LUT-based implementation of a self-timed ring oscillator have already been discussed in Chapter 4. The proposed PUF design requires the identical mapping of each self-timed ring oscillator. This includes both the symmetrical routing and the placement of identical
circuit instances. The FPGA Editor in the Xilinx toolset allows the user to create identical instances using hard-macros. Figure 5-3 shows the layout of a six-stage self-timed ring oscillator implemented as a hard-macro. The bull’s eye symbol represents the reference point of the hard-macro.

![Figure 5-2: A 6-stage asynchronous ring oscillator.](image)

![Figure 5-3: Hard-macro implemented as a six-stage asynchronous ring oscillator.](image)

Each group in a PUF circuit can have a number of asynchronous ring oscillators. The number of ring oscillators in the groups determines the possible combination of input challenges, the number of responses and the number of bits in each response. The response generated from the PUF circuit also depends on how the comparisons are made among the oscillators. Depending on the number of oscillators required in each group, the self-timed ring is duplicated using the hard-macro to ensure all the oscillators are identical.
Figure 5-4: Layout view of an STRO-PUF implemented with 16 pairs of identical STR oscillators in each group.

Figure 5-5: Portion of an STRO-PUF in FPGA Editor.
Hard-macros are instantiated in the main program and the locations of the hard-macros are defined in a User Constraint File (UCF) to map the PUF as desired. Figure 5-4 shows the duplication of a self-timed ring oscillator instance, which is created using hard-macros, in order to map 16 pairs of identical oscillators for implementing the STRO-PUF. Figure 5-5 shows a portion of the implemented STRO-PUF mapped in a region defined in the user constraint file.

5.4 Experimental Analysis

The proposed design is implemented on three different Xilinx Spartan-II boards. PUFs are mapped onto six different regions of each device as shown in Figure 5-6. Each PUF is realized using 16 pairs of identically laid-out STROs with 16 STROs in each group. For the purpose of the implemented design, a six-stage self-timed ring oscillator is used with two token and four bubble configurations, which are represented by their initial states of either ‘101111’ or ‘010000’ (TTBBBB).

Figure 5-6: PUFs mapped on six different regions of XC2S100 FPGA (20 X 30 CLBs)
The Set/Reset (SR) signal initializes the PUF states when SR = ‘1’ and generates oscillations when SR = ‘0’. Figure 5-7 illustrates PUF output read from a logic analyzer during initialization mode and oscillation mode.

5.4.1 Analysis of Output Frequencies

Frequencies generated from each of the self-timed ring oscillators of the STRO-PUFs are read through a logic analyzer. The varying oscillatory behavior of STROs is observed in the logic analyzer. In the simulation output, however, the same PUF design gives identical oscillatory behavior with same frequency for all STROs. Figure 5-8 and Figure 5-9 show the simulated waveform, and the real output taken from the logic analyzer. Figure 5-10 shows the frequency variations for 36 groups of asynchronous ring oscillators, which are mapped across six different regions of all three FPGAs. The maximum and the minimum frequencies observed are 125 MHz and 16.2438 MHz, respectively. The average frequency observed is 101.4460 MHz. The simulation result shows the identical frequency of 100 MHz for all the oscillators, which is different from the real responses. The robust responses can be determined by selectively comparing the frequencies of the oscillators, which have larger frequency differences.
Figure 5-8: Simulation result of STRO-PUF output frequencies.
Figure 5-10: Distribution of frequencies generated by asynchronous ring across FPGA devices.
5.4.2 Analysis of Uniqueness of STRO-PUF

For each challenge provided, a pair of oscillators is selected to generate a single bit response. For \( k \) number of ring oscillators, \( k \frac{(k-1)}{2} \) distinct pairs can be selected to generate \( k \frac{(k-1)}{2} \) response bits. But generating response bits from all the possible pairs reduces entropy due to the inclusion of dependent bits [13]. To avoid correlation, a simple approach is to use each oscillator only once in order to generate a single bit. The uniqueness can be calculated by using equation 2.1.

\[
\frac{2}{k(k-1)} \sum_{i=j}^{k-1} \sum_{j=i+1}^{k} \frac{HD(R_i, R_j)}{n} \times 100 \%
\]

The uniqueness analyses are performed for 16-bit PUF response and 256-bit PUF response, which are generated based on how the comparisons are made. Table 5.1 and Table 5.2 show 18 different PUF responses for two different comparisons. If each oscillator is used only once to generate a response bit, the STRO-PUF, having 16 pairs of STROs, can generate a 16-bit response. To analyze the overall signature uniqueness of the implemented design, all the PUF responses are considered. There are six different PUFs mapped on each of three FPGAs, which gives total of \((6 \times 3 = 18)\) 18 PUFs, producing \((18 \times (18-1)/2 = 153)\) 153 data points. The average Hamming distance for 16-bit responses is calculated as 7.99. Figure 5-11 illustrates the probability histogram of responses from the PUFs, indicating an average uniqueness of 49.92%, which is very close to the desired 50% factor.
Table 5.1: 16-bit STRO-PUF responses

<table>
<thead>
<tr>
<th>16-bit STRO-PUF responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>B0A2</td>
</tr>
<tr>
<td>6FFF</td>
</tr>
<tr>
<td>7F2A</td>
</tr>
<tr>
<td>B8DF</td>
</tr>
<tr>
<td>A647</td>
</tr>
<tr>
<td>F49B</td>
</tr>
<tr>
<td>F6E1</td>
</tr>
<tr>
<td>06FF</td>
</tr>
<tr>
<td>4041</td>
</tr>
</tbody>
</table>
If comparisons are made with each oscillator in a group being compared with every oscillator in another group, it can give a 256-bit (16x16 = 256) response. The entropy of these responses is reduced because the bits obtained also include the correlated bits. For example, consider two ring oscillators ‘a’ and ‘b’ in group ‘A’ and two ring oscillators ‘c’ and ‘d’ in group ‘B’. The possible combinations are (a, c), (a, d), (b, c) and (b, d), generating 4-bit response. If a>c, c>b, b>d then it can be easily predicted that a>d. The uniqueness for 256-bit response is obtained as 26.28% and its histogram is shown in Figure 5-12.

![Uniqueness Analysis of 256-bit response](image-url)

Figure 5-12 Uniqueness Analysis for 256-bit PUF response
Table 5.2: 256-bit STRO-PUF responses

<table>
<thead>
<tr>
<th>256-bit STRO-PUF responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>F062300030003000F062F062F062F062F2E7000F062F062F2E2E2F062F2E2</td>
</tr>
<tr>
<td>7040500070400000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF</td>
</tr>
<tr>
<td>700070000700070000FF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7AFF7A</td>
</tr>
</tbody>
</table>

Table 5.3 summarizes the analysis based on two different comparisons; comparing each oscillator only once, which gives the responses without dependent bits
and comparing each oscillator in group A to every oscillator in group B, which gives responses with dependent bits.

Table 5.3: Comparing responses with dependent bits and independent bits

<table>
<thead>
<tr>
<th></th>
<th>Response without dependent bits</th>
<th>Response with dependent bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of output bits</td>
<td>16</td>
<td>256</td>
</tr>
<tr>
<td>Uniqueness</td>
<td>49.92 %</td>
<td>26.28 %</td>
</tr>
<tr>
<td>Average HD</td>
<td>7.99</td>
<td>67.27</td>
</tr>
<tr>
<td>Minimum HD</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>Maximum HD</td>
<td>15</td>
<td>123</td>
</tr>
</tbody>
</table>

The uniqueness (inter-die variation) achieved with the proposed STRO-PUF is the closest to the desired factor of 50% compared to the previous work on FPGA-based PUF. For the comparison, uniqueness analysis with Table 5.4 shows the uniqueness results of the implemented STRO-PUF with 16-bit response versus previous work.

Table 5.4: Uniqueness results for FPGA-based PUFs

<table>
<thead>
<tr>
<th>Different PUFs</th>
<th>Uniqueness</th>
</tr>
</thead>
<tbody>
<tr>
<td>STRO-PUF (proposed)</td>
<td>49.92 %</td>
</tr>
<tr>
<td>Configurable RO-PUF [45]</td>
<td>47 %</td>
</tr>
<tr>
<td>RO-PUF [9]</td>
<td>46.15 %</td>
</tr>
<tr>
<td>RO-PUF [46]</td>
<td>48.4 %</td>
</tr>
<tr>
<td>Configurable RO-PUF[14]</td>
<td>47.31 %</td>
</tr>
<tr>
<td>Anderson PUF [20]</td>
<td>48.28 % (Average HD of 61.8 for 128-bit output)</td>
</tr>
</tbody>
</table>
**RO-PUF based on placement [47]**

<table>
<thead>
<tr>
<th>Random placement</th>
<th>Chain-like placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>43.40 %</td>
<td>48.51 %</td>
</tr>
</tbody>
</table>

### 5.4.3 FPGA Authentication using STRO-PUF

STRO-PUFs can be used to authenticate individual ICs without costly primitives. Figure 5-13 shows a basic PUF-based FPGA authentication process. Trusted parties create a Challenge-Response Pair (CRP) database from an authentic FPGA for future authentication operations. To verify the authenticity, the trusted party selects a challenge from the database and checks whether it matches its corresponding response or not. Each CRP is used only once to increase security. Both the 16-bit responses and the 256-bit responses generated from STRO-PUFs can be applied for this device authentication mechanism.

![FPGA Authentication Diagram](image)

**Figure 5-13: FPGA authentication using STRO-PUF**
5.4.4 Reliability Enhancement with STRO-PUF

Frequencies of ring oscillators can change significantly as environmental effects can cause the oscillators to flip their output bits. The effect of temperature and voltage on frequencies of ring oscillators is shown in Figure 5-14. When temperature increases, frequencies of oscillators slow down at different rate due to different device or physical parameters. In the Figure 5-14, at certain initially temperature, a ring oscillator represented by a dotted line is faster than a ring oscillator represented by a solid line. When temperature changes significantly, these ring oscillators flip. Similarly, with significant changes in voltages, the frequencies of oscillators change at different rate, which gives different result. It shows that ring oscillators with greater frequency differences are much less likely to flip than ring oscillators with narrow frequency differences. The errors caused due to the bit flips can be significantly reduced by comparing ring oscillators, whose frequencies are far apart, to generate response bits.

In STRO-PUF, the number of token and bubble can be configured from its initialization stage. By determining the configuration of the self-timed ring oscillators with the maximum frequencies differences, maximum reliability can be achieved.
Conclusion

This chapter described the implementation of a PUF using self-timed ring oscillators on FPGA. It uses a logic-based design of an underlying FPGA architecture. The approach can be used to implement low-cost authentication of the FPGA device and to generate secret keys for many cryptographic applications. The frequency analysis shows that asynchronous oscillators generate varying frequencies due to process variations. These frequencies can be selectively compared to generate response bits. The uniqueness of the implemented STRO-PUF for 16-bit response is calculated as 49.92 %,
which is close to desired 50% factor. The uniqueness and the strength of PUF responses also depend on how the comparisons are done. With the inclusion of dependent bits, the uniqueness factor reduces.
Chapter 6

Conclusion

6.1 Conclusion

Today’s global marketplace has opened up not only new opportunities but also new threats. In the current global marketplace, commercial products can be obtained easily, either by legitimate means or simply by theft. Counterfeiting has become one of the most significant threats to the free market. Physical Unclonable Functions (PUFs) have emerged as a potential technique to fight against hardware counterfeiting. PUFs are methods of extracting unique identity information from silicon devices or circuits based on their physical properties for device authentication.

Since the inception of a PUF concept, there have been various PUF techniques proposed, each with their own implications. In this thesis, a novel approach towards FPGA-based PUF design using asynchronous ring oscillators has been described. It uses the logic-based design of the underlying FPGA architecture. The frequency analysis shows that asynchronous oscillators generate varying frequencies due to process variations. These frequencies can be selectively compared, based on input challenges provided to the PUF, to generate response bits. The responses generated from the STRO-PUF are used in device authentication and in many cryptographic applications such as
generating secret keys and TRNG. From the experimental analysis, it is observed that the proposed PUF has a uniqueness factor of 49.92 %, which is close to desired factor of 50%. The uniqueness achieved with the STRO-PUF is better than the previous work in FPGA-based PUF designs (Table 5.4). The experimental analyses also show that the uniqueness of PUF responses also depends on how the input challenges are given to the PUF in order to generate response bits. The input challenge decides the selection of oscillators and the number of response bits.

The STRO-PUF can achieve better re-configurability features without significant hardware overhead. The initial stages of asynchronous ring oscillators in an STRO-PUF can be configured by setting different number of tokens and bubbles. Reliability of STRO-PUF can be enhanced by selectively comparing the frequencies of asynchronous oscillators which have wider frequency differences.

6.2 Future Directions

The work in this thesis is an initial step toward PUF design using asynchronous logic. Some possible extensions to this work include the following:

- The proposed design can be extended to have reconfigurable features by adding control logic to load a different token-bubble word during the initialization stage. A self-timed ring oscillator with same number of stages can generate different frequencies with different token-bubble configurations. This feature is not possible in a conventional inverter oscillator.

- Experimental analysis of the robustness of STRO-PUF in varying environmental conditions such as varying temperatures and varying voltages.
• Implementing STRO-PUF in other PUF applications such as a secret key generator and TRNG.

• Power analysis of STRO-PUF compared to other PUF designs.
Reference


[29] S. Pappala, M. Niamat, and W. Sun, "FPGA based key generation technique for anti-counterfeiting methods using Physically Unclonable Functions and artificial


Appendix A

Source Codes

A.1  VHDL Code for a Self-Timed Ring (STR)

-- six-stage self-timed ring oscillator

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;

use UNISIM.VComponents.all;

entity str6 is

    Port ( ring_out : out STD_LOGIC;

            init : in STD_LOGIC);

end str6;

architecture Behavioral of str6 is

    signal qout : std_logic_vector (1 to 6) := (others => '0');
attribute loc: string;
attribute loc of LUT4_inst1: label is "CLB_R1C1.S1";
attribute loc of LUT4_inst2: label is "CLB_R1C1.S1";
attribute loc of LUT4_inst3: label is "CLB_R1C1.S0";
attribute loc of LUT4_inst4: label is "CLB_R1C1.S0";
attribute loc of LUT4_inst5: label is "CLB_R1C2.S1";
attribute loc of LUT4_inst6: label is "CLB_R1C2.S1";

begin
  -- SET cell
  LUT4_inst1 : LUT4
generic map ( 
    INIT => X"FFB2")
port map ( 
  O => qout(1), -- LUT general output 
  I0 => qout(1), -- LUT input, feedback 
  I1 => qout(2), -- LUT input, reverse signal 
  I2 => qout(6), -- LUT input, forward 
  I3 => INIT -- LUT input, set/reset 
);
  -- End of LUT4_inst instantiation
-- RESET cell

LUT4_inst2 : LUT4

generic map (
  INIT => X"00B2")

port map (  
  O => qout(2),
  I0 => qout(2),
  I1 => qout(3),
  I2 => qout(1),
  I3 => INIT
);

-- SET cell

LUT4_inst3 : LUT4

generic map (  
  INIT => X"FFB2")

port map (  
  O => qout(3),
  I0 => qout(3),
  I1 => qout(4),
  I2 => qout(2),
I3 => INIT
);

-- SET cell

LUT4_inst4 : LUT4
generic map (
  INIT => X"FFB2"
) port map (
  O => qout(4),
  I0 => qout(4),
  I1 => qout(5),
  I2 => qout(3),
  I3 => INIT
);

-- SET cell

LUT4_inst5 : LUT4
generic map (
  INIT => X"FFB2"
) port map (
  O => qout(5),
  I0 => qout(5),

76
I1 => qout(3),
I2 => qout(4),
I3 => INIT
);

-- SET cell
LUT4_inst6 : LUT4
generic map (  
INIT => X"FFB2")
port map (  
O => qout(6),
I0 => qout(6),
I1 => qout(1),
I2 => qout(5),
I3 => INIT
);

ring_out <= qout(6);

end Behavioral;
A.2 VHDL Code for STRO-PUF

-- STRO-PUF; 16 STROs per group; 32 STROs per PUF

library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

library UNISIM;

use UNISIM.VComponents.all;

entity str6PUF8 is

  Port ( init : in  STD_LOGIC;
          ringout : out  STD_LOGIC_VECTOR (1 to 32));

end str6PUF8;

architecture Behavioral of str6PUF8 is

component hm_str6 is

  port (hm_init : in std_logic;
          hm_ringout : out std_logic);

end component;

begin

  -- instantiating hard-macros
puf1: hm_str6
    port map( hm_init => init, hm_ringout => ringout(1));
puf2: hm_str6
    port map( hm_init => init, hm_ringout => ringout(2));
puf3: hm_str6
    port map( hm_init => init, hm_ringout => ringout(3));
puf4: hm_str6
    port map( hm_init => init, hm_ringout => ringout(4));
puf5: hm_str6
    port map( hm_init => init, hm_ringout => ringout(5));
puf6: hm_str6
    port map( hm_init => init, hm_ringout => ringout(6));
puf7: hm_str6
    port map( hm_init => init, hm_ringout => ringout(7));
puf8: hm_str6
    port map( hm_init => init, hm_ringout => ringout(8));
puf9: hm_str6
    port map( hm_init => init, hm_ringout => ringout(9));
puf10: hm_str6
    port map( hm_init => init, hm_ringout => ringout(10));
puf11: hm_str6
  port map( hm_init => init, hm_ringout => ringout(11));
puf12: hm_str6
  port map( hm_init => init, hm_ringout => ringout(12));
puf13: hm_str6
  port map( hm_init => init, hm_ringout => ringout(13));
puf14: hm_str6
  port map( hm_init => init, hm_ringout => ringout(14));
puf15: hm_str6
  port map( hm_init => init, hm_ringout => ringout(15));
puf16: hm_str6
  port map( hm_init => init, hm_ringout => ringout(16));
puf17: hm_str6
  port map( hm_init => init, hm_ringout => ringout(17));
puf18: hm_str6
  port map( hm_init => init, hm_ringout => ringout(18));
puf19: hm_str6
  port map( hm_init => init, hm_ringout => ringout(19));
puf20: hm_str6
  port map( hm_init => init, hm_ringout => ringout(20));
puf21: hm_str6
  port map( hm_init => init, hm_ringout => ringout(21));

puf22: hm_str6
  port map( hm_init => init, hm_ringout => ringout(22));

puf23: hm_str6
  port map( hm_init => init, hm_ringout => ringout(23));

puf24: hm_str6
  port map( hm_init => init, hm_ringout => ringout(24));

puf25: hm_str6
  port map( hm_init => init, hm_ringout => ringout(25));

puf26: hm_str6
  port map( hm_init => init, hm_ringout => ringout(26));

puf27: hm_str6
  port map( hm_init => init, hm_ringout => ringout(27));

puf28: hm_str6
  port map( hm_init => init, hm_ringout => ringout(28));

puf29: hm_str6
  port map( hm_init => init, hm_ringout => ringout(29));

puf30: hm_str6
port map( hm_init => init, hm_ringout => ringout(30));

puf31: hm_str6

port map( hm_init => init, hm_ringout => ringout(31));

puf32: hm_str6

port map( hm_init => init, hm_ringout => ringout(32));

end Behavioral;
A.3 UCF File for Mapping STRO-PUF in a Desired Region

#PACE: Start of Constraints generated by PACE

#PACE: Start of PACE I/O Pin Assignments

# Mapping onto Region 1

NET "init" LOC = "p59" ;

# output pins for oscillators in Group A
NET "ringout<1>" LOC = "p43" ;
NET "ringout<2>" LOC = "p48" ;
NET "ringout<3>" LOC = "p47" ;
NET "ringout<4>" LOC = "p42" ;
NET "ringout<5>" LOC = "p40" ;
NET "ringout<6>" LOC = "p29" ;
NET "ringout<7>" LOC = "p28" ;
NET "ringout<8>" LOC = "p27" ;

NET "ringout<9>" LOC = "p68" ;
NET "ringout<10>" LOC = "p44" ;
NET "ringout<11>" LOC = "p46" ;
NET "ringout<12>" LOC = "p49" ;
NET "ringout<13>" LOC = "p26" ;
NET "ringout<14>" LOC = "p23" ;
NET "ringout<15>" LOC = "p57" ;
NET "ringout<16>" LOC = "p22" ;

# output pins for oscillators in Group B
NET "ringout<17>" LOC = "p75" ;
NET "ringout<18>" LOC = "p50" ;
NET "ringout<19>" LOC = "p51" ;
NET "ringout<20>" LOC = "p60" ;
NET "ringout<21>" LOC = "p62" ;
NET "ringout<22>" LOC = "p54" ;
NET "ringout<23>" LOC = "p56" ;
NET "ringout<24>" LOC = "p63" ;

#NET "ringout<25>" LOC = "p64" ;
#NET "ringout<26>" LOC = "p65" ;
#NET "ringout<27>" LOC = "p66" ;
#NET "ringout<28>" LOC = "p76" ;
#NET "ringout<29>" LOC = "p79" ;
#NET "ringout<30>" LOC = "p80" ;
#NET "ringout<31>" LOC = "p77" ;
#NET "ringout<32>" LOC = "p67" ;
#PACE: Start of PACE Area Constraints Region 1

#Region 1 Group A

INST "puf1" LOC=CLB_R1C1.S1;
INST "puf2" LOC=CLB_R1C3.S1;
INST "puf3" LOC=CLB_R1C5.S1;
INST "puf4" LOC=CLB_R1C7.S1;
INST "puf5" LOC=CLB_R1C9.S1;
INST "puf6" LOC=CLB_R1C11.S1;
INST "puf7" LOC=CLB_R1C13.S1;
INST "puf8" LOC=CLB_R1C15.S1;
INST "puf9" LOC=CLB_R2C1.S1;
INST "puf10" LOC=CLB_R2C3.S1;
INST "puf11" LOC=CLB_R2C5.S1;
INST "puf12" LOC=CLB_R2C7.S1;
INST "puf13" LOC=CLB_R2C9.S1;
INST "puf14" LOC=CLB_R2C11.S1;
INST "puf15" LOC=CLB_R2C13.S1;
INST "puf16" LOC=CLB_R2C15.S1;
#Region 1 Group B

INST "puf17" LOC=CLB_R3C1.S1;
INST "puf18" LOC=CLB_R3C3.S1;
INST "puf19" LOC=CLB_R3C5.S1;
INST "puf20" LOC=CLB_R3C7.S1;
INST "puf21" LOC=CLB_R3C9.S1;
INST "puf22" LOC=CLB_R3C11.S1;
INST "puf23" LOC=CLB_R3C13.S1;
INST "puf24" LOC=CLB_R3C15.S1;
INST "puf25" LOC=CLB_R4C1.S1;
INST "puf26" LOC=CLB_R4C3.S1;
INST "puf27" LOC=CLB_R4C5.S1;
INST "puf28" LOC=CLB_R4C7.S1;
INST "puf29" LOC=CLB_R4C9.S1;
INST "puf30" LOC=CLB_R4C11.S1;
INST "puf31" LOC=CLB_R4C13.S1;
INST "puf32" LOC=CLB_R4C15.S1;
# End of UCF
A.4 Uniqueness Analysis of STRO-PUF for 16-bit Response

% 6 PUF /FPGA, 3 devices, 16 STROs/group, 32 STROs / PUF

% independent bits, one on one comparison

% simple comparison, comparing each oscillator only once

\[
\begin{align*}
group\_size &= 16; \\
Npuf &= 36; \\
k &= 1; \\
t &= 0; \\
rbit &= (0); \\
\end{align*}
\]

\[
\begin{align*}
\text{for } p &= 1:2:Npuf \\
k &= 1; \\
t &= t + 1; \\
\text{for } j &= 1:\text{group}\_size \\
\text{if } (\text{data}(j,p) \geq \text{data}(j,p+1)) \\
rbit(t,k) &= 1; \\
\text{else} \\
rbit(t,k) &= 0; \\
\text{end} \\
k &= k + 1; \\
\text{end}
\end{align*}
\]
end
disp(rbit);

% binary to decimal
nbit1=2.^(size(rbit,2)-1:-1:0);
% decimal to hex
hexR1=dec2hex(nbit1*rbit.);
disp(hexR1); % PUF responses

% probability density function (PDF)
% calculating hamming distance between 1-1 pairs
%(generate 16-bit per comparisons from 16 pairs of STROs)

c=0;

int_hd= (0);
for i=1:(Npuf/2 - 1)
    for j = i+1 : Npuf/2
        c =c+1;
        int_hd(c,:)= sum(abs(rbit(i,:)-rbit(j,:)));
    end
end
display combinations:

disp(c);

disp(int_hd);

hd_data= int_hd; %frequency of occurrence of #bits flipped

binWidth = 1;

binCtrs = 1:1:16; %Bin centers, depends on data

n=length(hd_data);

counts = hist(hd_data,binCtrs);

prob = counts / (n * binWidth); %pmf = prob = counts / n

bar(binCtrs,prob,'hist');

min1 = min (int_hd); % minimum HD

max1 = max (int_hd); % maximum HD

fprintf (\n minimum hamming distance = %d', min1);

fprintf (\n maximum hamming distance = %d', max1);

s = sum(int_hd); % sum of HD

avg_hd = s/153;

fprintf (\n average hamming distance = %f', avg_hd);

uniqueness = s/(153*16) *100; % sum_hd /(no. of combination * no. of bits in an output)
fprintf("\n uniqueness = \%f\%n \n", uniqueness);
A.5 Uniqueness Analysis of STRO-PUF for 256-bit Response

% 6 PUF /device, 3 devices, 16STROs/group, 32 STROs

% inclusion of dependent bits

% comparing each oscillator in a group with every oscillator in another group

group_size = 16;
Npuf = 36;% no of oscillator. so no of pufs = Npuf/2
t=0;
rbital = (0);
for p = 1:2: (Npuf-1)
    k=1;
t=t+1;
    for i=1:(group_size)
        for j=1:group_size
            if (data(i,p) >= data(j,p+1))
                rbit(t,k) = 1;
            else
                rbit(t,k) = 0;
            end
            k = k +1;
        end
    end
end
end

disp(rbit);

% converting binary to decimal to hex
% can’t convert more than 52 bit ; breaking 256-bit into set of 32-bit
r1 = rbit(1:(Npuf/2),1:32);
r2 = rbit(1:(Npuf/2),33:64);
r3 = rbit(1:(Npuf/2),65:96);
r4 = rbit(1:(Npuf/2),97:128);
r5 = rbit(1:(Npuf/2),129:160);
r6 = rbit(1:(Npuf/2),161:192);
r7 = rbit(1:(Npuf/2),193:224);
r8 = rbit(1:(Npuf/2),225:256);

%binary to decimal
nbit1=2.^(size(r1,2)-1:-1:0);

%decimal to hex
hexR1=dec2hex(nbit1*r1.');

nbit2=2.^(size(r2,2)-1:-1:0);
hexR2=dec2hex(nbit2*r2.');

nbit3=2.^(size(r3,2)-1:-1:0);
hexR3 = dec2hex(nbit3*r3.');
nbit4 = 2.^(size(r4,2)-1:-1:0);
hexR4 = dec2hex(nbit4*r4.');
nbit5 = 2.^(size(r5,2)-1:-1:0);
hexR5 = dec2hex(nbit5*r5.');
nbit6 = 2.^(size(r6,2)-1:-1:0);
hexR6 = dec2hex(nbit6*r6.');
nbit7 = 2.^(size(r7,2)-1:-1:0);
hexR7 = dec2hex(nbit7*r7.');
nbit8 = 2.^(size(r2,2)-1:-1:0);
hexR8 = dec2hex(nbit8*r8.');

rbitHex = [hexR1 hexR2 hexR3 hexR4 hexR5 hexR6 hexR7 hexR8];

disp(rbitHex); % PUF responses

% probability density function (PDF)
% calculating hamming distance

c = 0;
int_hd = (0);
for i = 1:(Npuf/2 - 1)
    for j = i+1 : Npuf/2
c = c+1;

int_hd(c,:) = sum(abs(rbit(i,:)-rbit(j,:)));

disp(c);
disp(int_hd);

hd_data= int_hd; % frequency of occurrence of #bits flipped

binWidth = 1;

binCtrs = 1:5:250; % Bin centers, depends on data

n=length(hd_data);

counts = hist(hd_data,binCtrs);

prob = counts / (n * binWidth); % pmf = prob = counts / n

bar(binCtrs,prob,'hist');

min1 = min (int_hd); % minimum hd

max1 = max (int_hd); % maximum hd

fprintf('\n minimum hamming distance = %d', min1);

fprintf('\n maximum hamming distance = %d', max1);
s = sum(int_hd); % sum of hd

avg_hd = s/153; % average hamming distance

fprintf ('\n average hamming distance = %f', avg_hd);

uniqueness = s/(153*256) *100; % sum_hd /(no. of combination * no. of bits in an output)

fprintf ('\n uniqueness = %f%% \n', uniqueness);