9-4-2013

A hardware-embedded, delay-based PUF engine designed for use in cryptographic and authentication applications

James C. Aarestad

James C. Aarestad

Candidate

Electrical and Computer Engineering

Department

This dissertation is approved, and it is acceptable in quality and form for publication:

Approved by the Dissertation Committee:

Dr. James Plusquellic, Chairperson

Dr. Payman Zarkesh-Ha

Dr. Jedidiah Crandall

Dr. Todd Bauer

A Hardware-Embedded, Delay-based PUF Engine Designed for Use in Cryptographic and Authentication Applications

by

James C. Aarestad

B.S., Computer Engineering, University of New Mexico, 2009

M.S., Computer Engineering, University of New Mexico, 2011

DISSERTATION

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Engineering

The University of New Mexico

Albuquerque, New Mexico

July, 2013

Dedication

This work is dedicated to my loving and devoted parents, Wilfred ("W.C.") and Thelma Aarestad, who taught and demonstrated the true meaning of unconditional love, and showed me through example the value of living a worthy life. Dad, you have my unwavering respect, and Mom, you have my undying devotion.

Acknowledgments

As I culminate my years of education with the writing of this dissertation, I am acutely conscious of the many individuals who have helped to make this possible.

I am forever grateful and beholden to my advisor and mentor, Dr. Jim Plusquellic, whose academic support and guidance, research direction, and financial backing have enabled me to achieve this remarkable milestone. His exemplary integrity, research acumen, and industry insight have been powerful influences to me during my graduate education. Also, to the remaining members of my dissertation committee, Dr. Payman Zarkesh-Ha and Dr. Jedidiah Crandall of the University of New Mexico, and Dr. Todd Bauer of Sandia National Labs, I extend my heartfelt appreciation for contributing so graciously to my educational career.

In addition, my good friend, Dr. Charles Lamech, has been a tireless source of learning, inspiration, and excellence in research to me throughout the last four years, as have Dr. Ryan Helinki, Mitch Martin and Raj Chakraborty. Current and former members of Dr. Plusquellic's research team, including Greg Feucht, Jing Ju, Fareena Saqib, and Dylan Ismari, have provided a sense of camaraderie that has fueled my passion for research.

Dr. Howard Pollard, I thank you for the many hours of pertinent classroom instruction and for all the knowledge that you shared when we worked together during the summer of 2010. And I, and everyone else who has walked the halls of ECE, have also benefited greatly from the fastidiously detailed work of ECE graduate coordinator, Elmyra Grelle.

A Hardware-Embedded, Delay-Based PUF Engine Designed for Use in Cryptographic and Authentication Applications

By

James C. Aarestad

ABSTRACT OF DISSERTATION

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

Engineering

The University of New Mexico

Albuquerque, New Mexico

July, 2013

A Hardware-Embedded, Delay-Based PUF Engine Designed for Use in Cryptographic and Authentication Applications

by

James Charles Aarestad

B.S., Computer Engineering, University of New Mexico, 2009

M.S., Computer Engineering, University of New Mexico, 2011

Ph.D., Engineering, University of New Mexico, 2013

Abstract

Cryptographic and authentication applications in application-specific integrated circuits (ASICs) and field-programmable gate arrays (FPGAs), as well as codes for the activation of on-chip features, require the use of embedded secret information. The generation of secret bitstrings using physical unclonable functions, or PUFs, provides several distinct advantages over conventional methods, including the elimination of costly non-volatile memory, and the potential to increase the random bits available to applications.

In this dissertation, a Hardware-Embedded Delay PUF (HELP) is proposed that is designed to leverage path delay variations that occur in the core logic macros of a chip to create random bitstrings. A thorough discussion is provided of the operational details of an embedded path timing structure called REBEL that is used by HELP to provide the timing functionality upon which HELP relies for the entropy source for the cryptographic quality of the bitstrings. Further details of the FPGA-based implementation used to prove the viability of the HELP PUF concept are included, along with a discussion of the evolution of the techniques employed in realizing the final PUF engine design.

The bitstrings produced by a set of 30 FPGA boards are evaluated with regard to several statistical quality metrics including uniqueness, randomness, and stability. The stability characteristics of the bitstrings are evaluated by subjecting the FPGAs to commercial-grade temperature and power supply voltage variations. In particular, this work evaluates the reproducibility of the bitstrings generated at 0C, 25C, and 70C, and 10% of the rated supply voltage.

A pair of error avoidance schemes are proposed and presented that provide significant improvements to the HELP PUF's resiliency against bit-flip errors in the bitstrings.

Table of Contents

List of Figures

Index of Tables

Chapter 1: Introduction

Recent years have seen a decline in the vertically-integrated nature of the semiconductor industry. One effect of this change has been a heightened concern over the issues of hardware security and trust in the VLSI design/manufacturing/distribution cycle. Threats of such activities as the unauthorized cloning of integrated circuits and the placement of key components in sensitive electronic systems that may have been compromised or altered are serious, real, and difficult to thwart using only procedural and policy methods. Additionally, the security weaknesses imposed by the storage of secret keys and identification in non-volatile storage in an IC make these techniques unattractive as well. As a result, the emergence of these threats is causing designers to search for effective measures to counter the security risks that they present. One technique that has shown promise is the physical unclonable function (PUF); the exploitation of repeatedly random device properties to distinguish one instance of a device from another.

PUFs enjoy a key advantage over other techniques for establishing unique identification for individual integrated circuits, particularly over storage-based solutions such as eFUSE and EEPROM technologies; because their uniquely identifying characteristics derive from naturally-varying physical properties within the die, they do not require, nor do they permit, the impression of an identification mark from the outside world.

Physical unclonable functions (PUFs) are becoming increasingly attractive for generating random bitstrings for a wide range of security-related applications. PUFs are designed to reliably differentiate one chip from another by leveraging the naturally-occurring random process variations which occur when the chips are fabricated. Process variations are increasing as layout geometries shrink across technology generations. Although undesirable from a design perspective, the electrical variations introduced by process variations define the entropy source on which PUFs are based. PUFs are designed to measure and 'digitize' these electrical variations to create random bitstrings. The most common sources of variations that PUFs leverage include path delay, metal resistance and SRAM power-up patterns.

The quality of the bitstrings produced by a PUF are typically evaluated using a suite of statistical tests. Generally, three criteria are considered essential for a PUF to be used for applications such as encryption: 1) the bitstrings produced for each chip must be sufficiently *unique* to distinguish each chip in the population, 2) the bitstrings must be *random*, making them difficult for an adversary to model and predict, and 3) the bitstring for any one chip must be *stable* over time and across varying environmental conditions.

In this thesis, we present a detailed examination of a PUF, called HELP, that is based on path delay variations. The novel features that differentiate HELP from other delay-based PUFs include: 1) the capability of comparing paths of different lengths without adding bias, 2) elimination of specialized test structures, 3) a minimally invasive design with low per-bit area and performance impact, and 4) a PUF engine that is integrated into the existing functional units of the chips and requires no external testing resources. The integration of HELP into an existing functional unit, such as an implementation of the Advanced Encryption Standard (AES), allows it to leverage a large source of entropy while minimizing its overall footprint. This large source of entropy allows HELP to generate long bitstrings, while being conservative in the paths selected for bit generation. The large availability of paths also enables unique opportunities for avoiding bit-flip errors.

The HELP PUF introduced in this thesis presents a new concept in path delay PUFs and includes experimental results, performance analysis against a set of established PUF evaluation criteria, and a complete PUF primitive and an associated on-chip PUF engine. The term "hardware-embedded" refers, in this context, to PUFs that generate repeatedly random digital bit strings which can subsequently be supplied to encryption circuitry for use in its functional operation. Because the chip is able, internally, both to generate and consume the PUF bit string (e.g., as a private key for an encryption engine), the bit string does not need to be transmitted off-chip.

HELP is based upon path delay, and takes advantage of manufacturing variations which alter the propagation delays in a target design. By eliminating the magnitudes of path lengths and accounting for timing and measurement uncertainties, this PUF enables the comparison of logic paths whose actual path delays might vary widely. Because I am not constrained to only comparing paths with similar lengths, this approach permits an exponential number of delay comparisons, and consequently, the generation of PUF bit strings the lengths of which grow exponentially with increasingly long challenge vectors. Using data that was generated during prior experimentation using an on-chip path delay measurement structure, we examine the resulting comparisons using several standard statistical metrics to determine the effectiveness of our PUF strategy, including analyses of numeric distribution, Hamming distance, Euclidean distance, and tests of the randomness of the resulting bit string.

As with all PUFs, we have found that uncertainty in our delay measurements has a direct bearing on the amount of the variability that can be used to create separation between devices. As a result, we have performed an analysis of the uncertainty that is present in the data, and that information is presented here as well.

There is a growing set of advanced statistical analysis tools and techniques that are being commonly applied to the data generated by any random or pseudo-random number generator (RNG/PRNG); these include a battery of statistical tests from the National Institute of Standards and Technology (NIST) [2]. We have included a discussion of our ongoing work with these and other tools, such as the use of helper data and error correction to improve the utility of PUF bit strings.

Chapter 2: Background

The PUF first appeared as a mechanism for generating secure bitstrings in [3] and [4]. The PUF as a chip identifier, however, was introduced earlier in [5]. Proposed PUF designs generally fall into one of the following classifications: SRAM PUFS [6], ring oscillators [7][8], MOS drive-current PUFs [9], delay line and arbiter PUFs [10], and PUFs based upon variations in a chip's metal wires [11]. Delay-based PUFs also include such designs as the Glitch PUF, which leverages variation in glitch behavior and is presented in [12]. Each of these PUFs takes advantage of one or more naturally-varying properties, and nearly all PUFs share a common set of challenges such as measurement error and uncertainty, and fluctuations in voltage or temperature. The degree to which a given PUF can tolerate or mitigate these challenges is an important indicator of its utility for generating secret data.

The HELP PUF proposed in this dissertation, is to the best of our knowledge the only delay-based PUF that combines the following features:

- The HELP PUF is entangled with the hardware in which it is embedded, in the sense that the path delays measured in, e.g., an AES core logic macro, can be used to generate a bitstring that is subsequently used as the key for use by AES in functional mode. The proximity of the bit generation to the hardware that uses the bitstring improves robustness against invasive or probing attacks designed to steal the key.
- The bit flip avoidance scheme proposed in this thesis significantly reduces the probability of bit-flip errors during regeneration.

- The physical implementation of HELP uses standard hardware resources commonly available in the fabric of an FPGA, including an on-chip digital clock manager (DCM). (Note: the authors of [13] also leverage the high timing resolution provided by a DCM for Trojan detection and IC authentication, although they did so using an earlier version of the same FPGA. They used a Virtex-II device from Xilinx [14], with a DCM capable of timing increments of 160 ps, while our experiments utilize a Virtex-II Pro, which has the capability of phase adjustments of just under 80 ps when used with a 50-MHz clock like that used in our experiments. Additionally, the applications most directly targeted by their research were authentication and hardware Trojan Horse (HTH) detection.)
- By using the core logic of AES itself, a large source of existing entropy is leveraged.

The area of hardware security research that is concerned with developing methods of security based upon a device's unclonable properties can be coarsely divided into two categories: 1) methods that make use of those properties which are purely physical in nature, and 2) methods which rely upon those physical properties which in some way impinge on the electrical behavior or performance of an individual component or device.

The first category, referred to in Chapter 4 of [15] as Unique Objects (UNOs), encompasses a number of interesting and potentially viable identification methods, including, for example, the unique scattering patterns created by light passing through, or reflecting from, the surface of an object under consideration. However, these UNOs typically require specialized external measurement equipment, are fundamentally different in kind than those involving the electrical properties of electronic circuits and, as such, are not considered further in this work.

2.1 PUF Classification and Types

In recent years, as PUFs have gained attention in hardware security research, developments in the field have crystallized into a handful of architectural classes. Most proposed PUFs fall into one of several categories: so-called “bistable element PUFs”, ring oscillators (ROs), MOS drive-current differences, and delay line and arbiter PUFs, as well as PUFs based upon variations in back-end-of-line (BEOL) properties, such as metal thickness and via resistance.

2.1.1 Bistable Element PUFs

2.1.1.1. SRAM/Butterfly PUFs

SRAM PUFs are based upon a traditional six-transistor static RAM cell, and make use of the mismatched threshold voltages in small inverter transistors caused by variations in the manufacturing process [16]. Proponents of the SRAM PUF claim that the combined effects of common-mode process variations and common-mode temperature and power supply fluctuations are effectively eliminated by the differential nature of the SRAM PUF. These PUFs have less of an advantage in those devices without SRAM cells, as an important design feature of this PUF type is the regularity and symmetry of the layout which must be present in the cells to prevent adverse biasing effects from degrading the performance of the PUF. A similar PUF concept, known as the Butterfly PUF [17], is based upon two opposing NAND gates that interact in much the same way as an SRAM cell and has been proposed for use in those designs which do not use SRAM memory cells as a standard cell library component. As in the SRAM PUF, the stable state of each Butterfly PUF cell is determined randomly by mismatches between the threshold voltages and drive strengths of the component transistors.

This PUF is also heavily dependent upon consistent layout to eliminate bias. SRAM PUFs and Butterfly PUFs are examples of a class known as “weak PUFs”, characterized by a relatively small number of challenge-response pairs (CRPs). Strong PUFs, such as those described in the upcoming paragraphs, generally are marked by a large, often exponential, number of CRPs.

2.1.1.2. The BusKeeper PUF.

This PUF concept was proposed as an area-efficient alternative to the use of D-Flipflop (DFF)-styled and traditional 6T-style SRAM PUFs.

The buskeeper, also known as a *weak keeper*, is a small, inputless cross-coupled latch made from minimum- (or less-than-minimum-) sized transistors that is used to help to preserve a bus-attached (tri-stated) register's state. The small size of the device is designed to be easily overwritten by changes in the state of the register itself.

Beyond the savings in area and power, there is little difference between the fundamental operation of the PUF itself and that of other bistable element PUFs. The authors of [6] present the details of a PUF implementation using the buskeeper in a 65-nm TSMC bulk technology. The analysis includes a discussion of the PUF's performance across an industrial range of temperatures. No voltage analysis is provided, and while this could be considered a weakness of the research, it was shown in [18] that SRAM PUFs are, in relative terms, not significantly subject to changes in supply voltage. Given that the final state of the Buskeeper PUF is also arbitrated during the power supply ramp to its final value, it is reasonable to conclude that the Buskeeper PUF is not adversely affected by changes in supply voltage, it would have strengthened the research to demonstrate this immunity in the paper.

2.1.2. Delay-Based PUFs

Two examples of strong PUFs are the arbiter PUF [19] and the ring oscillator PUF [20]. These delay-based PUFs make use of variations in device characteristics which alter the path delays through these PUFs. These PUFs are of particular interest to the work that I have done, as HELP is also a type of delay-based PUF.

2.1.2.1. The Arbiter PUF

The Arbiter PUF makes use of a pair of configurable, identically laid-out paths through a set of path-switching elements, terminating at an arbitration element. In many such designs, the arbiter is a 'D' flip-flop with one path connected to the 'D' input and the other path driving the D-FF's clock input. A positive-going edge is asserted onto both paths simultaneously, setting up a race condition between the propagating edges which is determined by the random variations in the path design. In some devices, the path which is connected to the clock input of the arbiter will be faster, and a transition will occur at the clock input ahead of the 'D' input, and a '0' will be stored in the arbiter. In other devices, the 'D' transition will occur ahead of the clock, and a '1' will be stored. Multiple arbiters PUF structures, with large numbers of switching elements in each, can be implemented to increase the number of resulting bits in the PUF bit string, and the size of the challenge vector space is determined by the number of switching elements in each

PUF.

While arbiter PUFs have been demonstrated to be effective and realizable, there are several significant drawbacks, as well: they impose heavy area and power penalties, and they require a high degree of regularity to prevent bias. Additionally, as the length of an arbiter PUF increases, the Central Limit Theorem dictates that the aggregate path length tends to converge to a mean value, and there is often not sufficient difference in the path delays to overcome the setup and hold time of the arbiter, resulting in many unstable, unusable CRPs.

2.1.2.2. The Ring Oscillator (RO) PUF

Another delay-based PUF is the RO PUF which, as the name implies, uses a large number of ring oscillators and pulse counters. The ROs are enabled and used to increment counter values for a fixed period; the values in the counters are then compared and the results of the comparisons are used to generate binary digits in the PUF bit string. This PUF also imposes a high area cost, and in addition, the large number of oscillating devices incurs large dynamic power consumption.

RO PUFs are used heavily in the research of PUFs because they are easily understood and can be easily implemented in FPGA-based hardware experiments. Their widespread adoption in commercial PUF applications could be unlikely, however, as their high area cost and large dynamic power footprint will remain a liability throughout the lifetime of the chip.

2.1.3. The Power Grid PUF (PG-PUF)

The PG-PUF is an example of a PUF based upon the electrical characteristics of a power distribution system (PDS) such as that found in a traditional IC design. This PUF belongs to a class of PUFs that leverage physical properties in an existing chip (as contrasted with front-end-of-line (FEOL) variations in transistors); as such, the performance of this PUF will depend more upon the types and magnitudes of variations that occur in post-metal manufacturing operations. The PG- PUF relies on comparisons between the voltage drops that occur at different points in the multi-level metal grid of a PDS when current flow is induced at a point in the network. Process variations result in localized changes in the resistance of the metal lines and the vias between the metal layers. An alternate method is to combine the voltage and current measurements to compute an equivalent resistance (R_{EQ}). Regardless of the method used, the binary evaluations that result from these comparisons are used to create a unique digital bit string for a given device.

Although no formal study on the impact of aging has been completed at the time of this writing, the PG-PUF may offer a significantly reduced sensitivity to changes in PUF response over time. This is because, unlike the classification of PUFs that draw on variations in the front-end-of-line (FEOL) processes that produce the transistor devices, the only significant aging effect related to the entropy source upon which the PUF draws is *electromigration*, the physical displacement of metal molecules caused by extremely localized areas of high current (such as in corners and jogs in the metal lines). The temperature effects related the FEOL, in contrast, will be related more to the readout circuitry, and therefore appear as common-mode changes that affect the individual measurements equally. (Note: the author concedes that this is assertion amounts to little more than informed speculation.)

2.2. PUF Design – Challenges

Each of the PUFs that have been proposed or implemented attempts to leverage one or more naturally varying properties. These PUFs share, to varying degrees, the challenges that arise from a number of sensitivities, including measurement error and uncertainty, fluctuations in process, voltage, or temperature (PVT) conditions, and instability over time [21], among other factors. Devising methods and techniques for the mitigation of these sensitivities is equally as important to the development of robust and reliable PUFs as the underlying PUF design itself. Regardless of the type of PUF that is considered, the ability to attain perfection in terms of robustness and randomness remains an elusive goal. As a result, some PUF designers have come to rely on the use of “helper data”, which is information concerning the PUF bit string that is made externally available to help to identify those bits that are not stable enough to be included in a PUF bit string. This information, however, “leaks” or reduces the amount of entropy that a PUF can leverage. As such, one desirable characteristic of a robust PUF is the avoidance of the need to rely on the use of helper data.

One technique for improving PUF performance without the need for helper data is the use of hardware redundancy to create a voting scheme, which reduces the probability of single-bit instability in a PUF bit string. This technique necessarily incurs a penalty in terms of area and power consumption, while maximizing the use of available entropy in the PUF design.

The HELP PUF is, to the best of my knowledge, the only delay-based PUF that offers the following combination of characteristics; it is hardware-entangled, which means that 'secret' information is derived from the core-logic components themselves and therefore secrets remain localized on-chip to those components that use that information, it takes advantage of an existing REBEL path-delay measurement structure (discussed thoroughly in the following chapter), and it is 'all-digital', which means that it can be implemented using standard library components, i.e., there is no need for a specialized on-chip PUF structure outside of the PUF engine that controls its operation.

Chapter 3: The Hardware-Embedded Delay PUF

The Hardware-Embedded Delay PUF, or HELP, is a physical unclonable function that draws on the variations in path delay found in an existing combinational logic block. The actual path delay measurements are carried out by an embedded test structure called REBEL [22]. This test structure was developed primarily for adding the ability to do on-chip path delay timing measurements in Design-For-Testability (DFT) applications, but which offers a convenient mechanism for providing path timing functionality for the HELP PUF as well. The REBEL embedded timing structure is discussed in greater detail in the following chapter.

Chapter 4: REBEL Test Structure

In [22], we proposed an embedded test structure (ETS), called REBEL, which is designed to measure regional path delays in macros while minimizing these types of adverse effects. This test structure is designed to serve applications such as model-to-

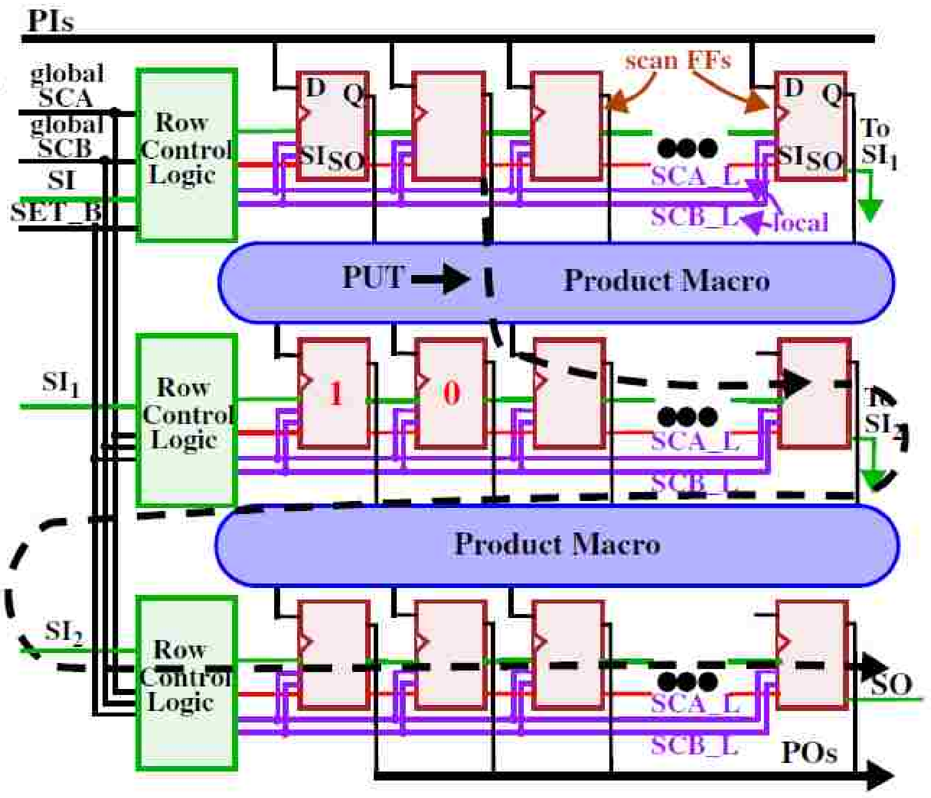

Fig. 1: REBEL Implementation Strategy

hardware correlation, detection of hardware Trojans [13], design debug processes, detection of small delay defects [23], and, especially relevant for this work, physical unclonable functions. Each of these areas requires accurate measurements of path delays and/or the ability to differentiate at high resolutions between delays of neighboring paths.

The REBEL ETS leverages the scan chain architecture to measure delay variations; in particular, it uses a special configuration of flush delay mode that is available in level-sensitive scan design (LSSD) scan chains and in modified Mux-D scan designs. We demonstrated in a previous work [24] the promise of capturing regional delay variations using a special launch-capture timing sequence applied while in flush delay mode. We extend this technique here by allowing output signals from design macros to be inserted into the flush delay chain for path delay measurements.

4.1. REBEL for LSSD Scan

As indicated previously, REBEL is integrated into the scan chain directly, as shown in Fig. 2 for a clocked-LSSD-style scan architecture. Here, the regions labeled ‘Product

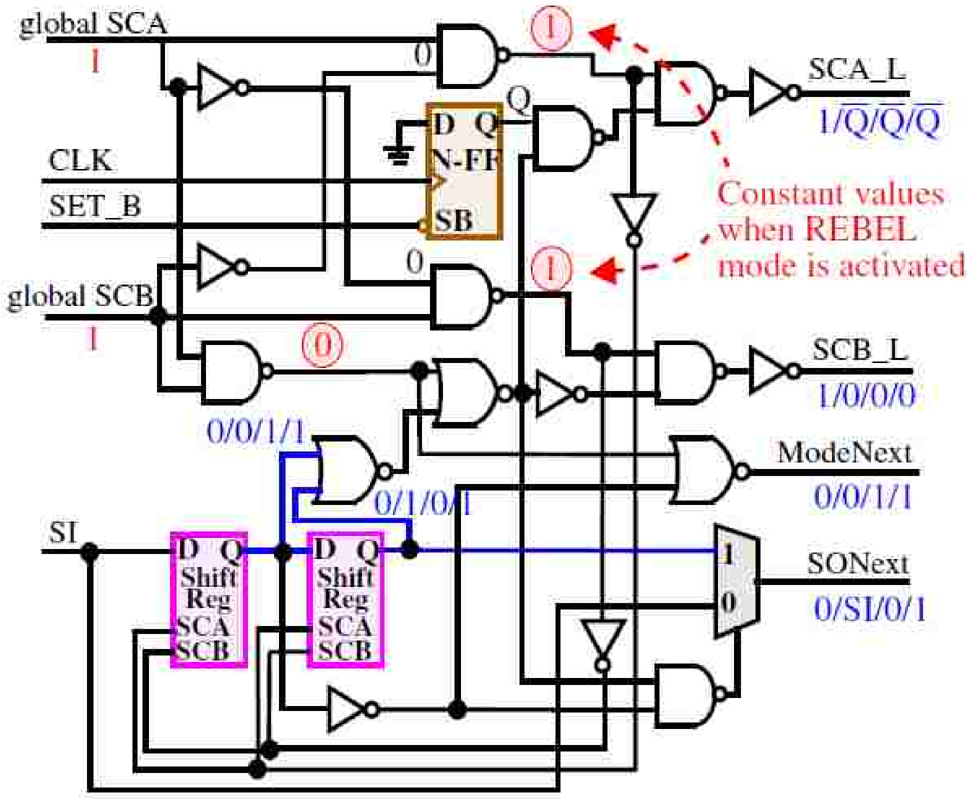

Fig. 2: REBEL Row Control Logic (RCL)

Macro' are functional units composed of combinational logic. Three scan chain segments are shown that serve to deliver input and capture output from these macros. The three blocks labeled Row Control Logic identify components of the REBEL ETS, and are described below. Beyond these three 'header' blocks, smaller blocks are also needed for local scan signal control for each of the scan FFs. The basic idea is to generate a transition on the inputs to the macro using a standard launch-off-capture transition fault test. In this scenario, the scan chain is loaded with the initial pattern of the 2-pattern test and the system clock (CLK) is used to generate a transition in the core logic by capturing the output of a previous block, or by capturing the PI values, as shown in the figure. One or more transitions are propagated through the macro, as shown by the dotted line labeled "PUT" (for "path-under-test"). The PUT's transition emerges on an output of the macro, and drives the D input of a scan FF in the second row. Special control logic associated with the scan FF (to be described) allows the transition to propagate along the scan chain, as shown by the dotted line in Fig. 2. CLK is then reasserted to halt the propagation, which effectively 'takes a digital snapshot' of the signal propagation behavior along the scan chain, including any glitching that may have occurred. This digital snapshot is then scanned out for analysis.

For designs that make use of LSSD-style scan, propagation along the scan chain is relatively straightforward to implement. This is true because LSSD inherently supports a flush-delay (FD) mode of operation. In FD mode, both the 'Scan A' clock (SCA) and 'Scan B' clock (SCB) are held high, effectively making both latches of the FF transparent, i.e., any transition generated on D propagates to Q after a Δt that represents the delay through the FF. FD mode effectively makes the scan chain a combinational inverter chain.

However, the configuration in Fig. 2 differs from the traditional definition of FD mode because only a portion of the scan chain is configured in FD mode. In particular, the scan FFs along the top row and those along the middle row to the point of insertion of the PUT operate in functional mode, and only those to the right (and below) of this point operate in FD mode. In order to realize this configuration, several changes are required to the logic implementing the scan operation.

One of the components to support this dual mode of operation is labeled Row Control Logic (RCL) on the left side of Fig. 2. These blocks, in combination with a scan chain encoding scheme and localized scan FF logic, enable this dual mode of operation and provide a mechanism to specify a PUT's output to direct into the scan chain. This is accomplished by configuring several state bits in the RCL, and by loading a specific pattern into the scan chain before the launch-capture (LC) timing sequence (REBEL test) is applied, as described below.

Each RCL block controls a 'row' of scan FFs, called row-FFs, in the following description. Fig. 2 shows a schematic diagram of the RCL. The top portion of the diagram controls local (row-specific) scan clock signals, labeled SCA_L and SCB_L (_L for local) while the bottom portion contains two shift registers (Shift Reg) and mode select logic. A large portion of the RCL logic is in place to allow different sections of the row FFs to operate in either of the traditional functional or scan modes of operation. The global (chip-wide) scan signals (labeled 'global SCA' and 'global SCB') are used to specify one of the three possible global operational states. When both are low, the entire row is set to operate in functional mode, with CLK controlling the launch-capture activity in the row FFs. Non-overlapping assertion of these signals causes all scan FFs to act as a

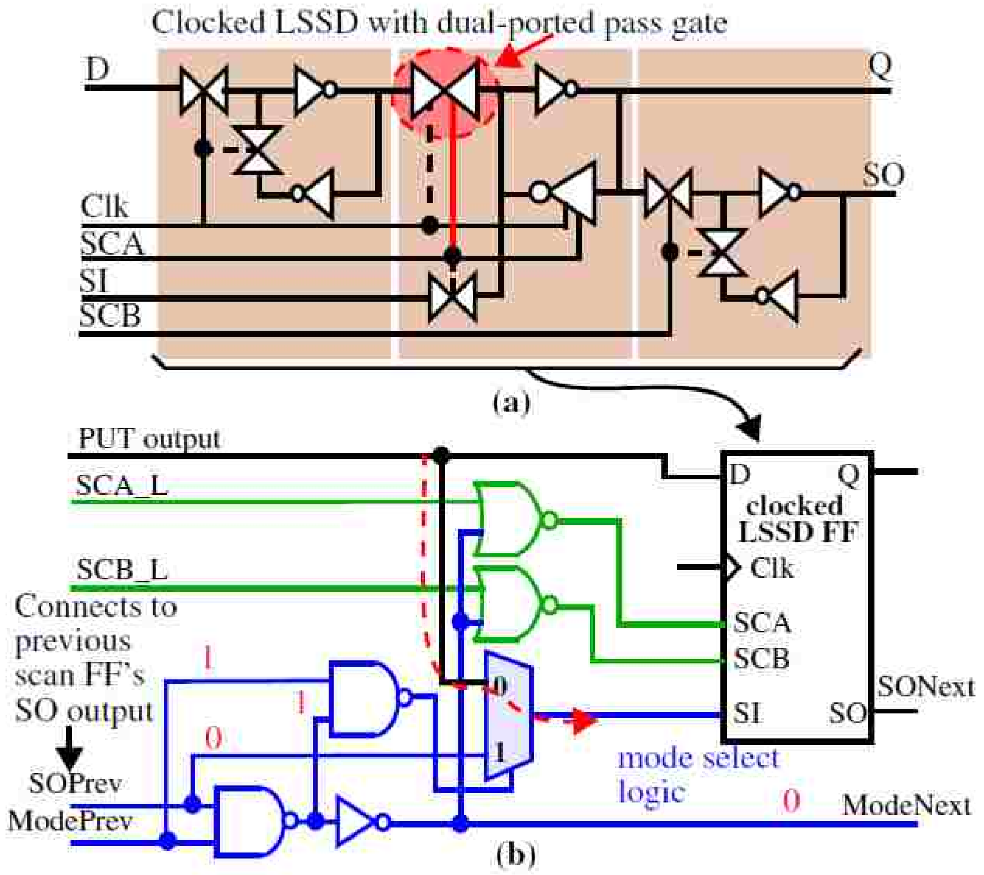

Fig. 3: REBEL Cell Modification Logic (LSSD)

shift register, implementing scan mode. The timing mode used by REBEL is specified when both of these signals are asserted. This is illustrated by the '1's on global SCA and SCB in Fig. 2.

Note that the two shift registers in the RCL block are conditionally inserted into the scan chain during a scan operation and can therefore be configured prior to a REBEL test. The shift registers' scan clock inputs (SCA/SCB) are also gated to prevent them from entering FD mode, thereby destroying the state information, when both global SCA and SCB signals are asserted. The state of the two shift registers defines the mode of operation for the row when REBEL mode is activated. Two control bits (as opposed to one) are needed to implement the simultaneous functional and FD modes discussed above because there are actually four possible conditions that need to be handled. The three rows of scan FFs in Fig. 1 illustrate three of the four conditions. For example, the scan FFs in the top row need to be in functional mode throughout the REBEL test. In contrast, the scan FFs in the bottom row need to be in FD mode to extend the propagation path of the PUT signal captured in the middle row. Finally, the middle row contains scan FFs in both of these modes, i.e., the scan FFs to the left of the PUT insertion point are in function mode while those to the right are in FD mode. The fourth condition is just a special case of this third condition where the insertion point is the left-most scan FF in the row. Table 1 identifies the bit configurations that handle these four conditions.

RCL Shift Reg	**Functionality**
00	All scan FFs in Row are in functional mode
01	All scan FFs in row are in FD mode
11/10	Left scan FFs in functional mode, right scan FFs in FD mode (10: Insertion Point is leftmost scan FF)

Table 1: Configuration States for Row Control Logic

Before describing the annotations in Fig. 4, we turn to the configuration of the scan FFs. Fig. 4(a) shows a clocked LSSD FF (CLSSD), which consists of three latches. The two latches on the left implement the functional path, and are controlled by the system clock (Clk). The center latch is dually ported and serves both as the slave for the functional path and as the master in the LSSD pair. The right-most latch is the slave latch of the LSSD pair. The top pass-gate of the dual port latch is highlighted to indicate that it has been modified. In the following paragraphs, it will become apparent that during the REBEL test, both CLK and SCA will be asserted simultaneously during a portion of the test. This creates a potential shorting condition in the dual port latch, i.e., both the master of the functional path and the SI input paths are enabled. To prevent this from happening, we modified the single input pass gate connected to the master's output to include a second input. The second input contains a "wired-AND" configuration, and prevents the master's output from driving the dual port latch when both CLK and SCA are asserted simultaneously.

Fig. 4(b) shows the additional logic required to integrate REBEL into a design with CLSSD-style scan. The functional path's D-input is fanned out to a 2-to-1 MUX. This will allow for the insertion of a macro's PUT into the scan chain during the REBEL test. The local scan signals (SCA_L and SCB_L) are gated by mode select logic shown along the bottom of the figure. The mode select logic incorporates the normal scan path (SOPrev to the SI input), as well as a propagating mode bit (ModePrev to ModeNext). The mode select logic is responsible for selecting the insertion point. This is accomplished by preloading the row-FFs with a pattern of all '1's followed by a '0' from left to right along the row-FFs. The '0' in this sequence causes the next scan FF to be configured in a special way, i.e., it allows the PUF output signal to drive the SI pin. The annotation and dotted line in the figure illustrates this case, and assumes the scan FF on the left (not shown) is configured with a '0' bit. Given the scan chain connects the SO output of each scan FF to the SOPrev of the next scan FF, this arrangement allows the scan chain encoding to specify the PUT insertion point. Moreover, the split mode of operation required for this row is implemented using a propagating mode bit (ModePrev and ModeNext), which is '1' for all scan FFs to the left of the insertion point and '0' to the right. The left-most scan FFs in the middle row of Fig. 1 are annotated with a bit configuration that enables the insertion of the PUT at the position shown.

The mode select logic also participates in controlling the local scan signals (SCA_L and SCB_L), and completes the implementing of the four conditions described above in reference to the RCL. The shift registers in Fig. 2 are annotated with four states (for the four conditions). The '00' state, which forces functional mode for the row-FFs (row 1 in Fig. 1), sets both SCA_L and SCB_L to '1'. Given that these signal connect to the inputs of the two NOR gates in instances of the scan FFs (as shown in Fig. 4(b)), and '1' is the dominate value for a NOR gate, this condition effectively disables FD mode for the entire row. In this case, the ModeNext and SONext output signals of the RCL, which connect to the left-most scan FF's ModePrev and SOPrev signals, are irrelevant.

The '01' state, as discussed earlier, forces the row-FFs into FD mode (row 3 in Fig. 1). This requires both of the SCA_L and SCB_L signals to be set to '0'. However, the annotation in Fig. 2 indicates that the value of SCA_L is 'Q', which is the inverted output value of the negative-edge-triggered FF (N-FF) in the RCL. In the implementation flow for a REBEL test, the initial value of the N-FF is set to '1' by virtue of strobing the SET_B signal low prior to the REBEL test. The REBEL test is defined as a rising edge on

CLK (which effectively launches a transition(s) into the macro-under- test), followed by a falling edge on CLK that acts to capture a snapshot of the PUT's behavior in the scan chain. The snapshot is realized by deasserting the Q output of the N-FF, which occurs when CLK goes low. This in turn causes the SCA_L output signal from the RCL to transition from '0' (initial value) to '1'. From Fig. 4(b), the arrival of the '1' on SCA_L signals of the scan FFs deasserts the SCA input and turns off FD mode. This action captures the snapshot of the PUT's voltage behavior in the scan chain.

The ModeNext output signal of the RCL configured in the '01' state is '0'. The '0' propagates along the mode select logic of the row and forces all row FFs to operate in scan mode, i.e., SO to SI to SO, and so on. This condition allows for the propagation of the PUT's signal along the scan chain. The SONext signal's value for state '01' is given as 'SI' to indicate that this signal is driven from the SI input of the RCL. Therefore, the scan chain bypasses (and preserves the contents of) the state elements in the RCL. The SI input in turn connects to the SO signal from the right-most scan FF of the previous row, effectively extending the scan path across rows (see Fig. 1).

Finally, the '11' state in the RCL configures a split mode of operation in the row FFs and connects a specific PUT output into the scan chain (row 2 in Fig. 1). The mode select logic in the scan FFs work together with the RCL block to implement this split mode of operation. The behavior of the SCA_L and SCB_L outputs are identical to those described above for state '01'. The difference lies in the state of the ModeNext and SONext output signals in Fig. 2. As noted above, a string of '1's followed by a '0' are preloaded into the scan chain to specify the PUT insertion point. The '1' on the

ModeNext output propagates along the mode select logic, described earlier in reference to Fig. 4(b) until a '0' is encountered in scan FFs of the row. This causes the next scan FF to be configured as the insertion point. The remaining scan FFs in the row are configured in FD mode because the mode bit is inverted to a '0' after the insertion point. RCL state '10' behaves identically but allows the insertion point to be the left-most scan FF in the row.

We have designed the REBEL support logic such that it minimizes the impact on the functional behavior of the design. There are two components of REBEL that impact the functional operation. The first is the change of the CLSSD as shown in Fig. 4(a), and the second is the fanout of the D input to the 2-to-1 MUX as shown in Fig. 4(b). Each of these changes adds a small delay (Δt) to the functional path.

4.2. REBEL For Mux-D-Style Scan

Although we integrate and demonstrate REBEL in a CLSSD- style scan chain (which is the style used in the design flow of our 90-nm test chip), MUX scan is the industry standard and is the design generally required for FPGA-based implementations such as has been used for the majority of this research. Integration of REBEL into MUX scan is easy and even less invasive than it is for CLSSD. The overall operation of REBEL for MUX scan is very similar to that described for CLSSD. The main difference is that the launch and capture is accomplished using rising edges of CLK (as opposed to a rising and falling edge for CLSSD). Also, an additional primary input is required to specify FD mode. This global signal is routed to the RCL blocks (not shown). A RCL block for MUX scan is similar in function to the CLSSD version except that all logic in reference to

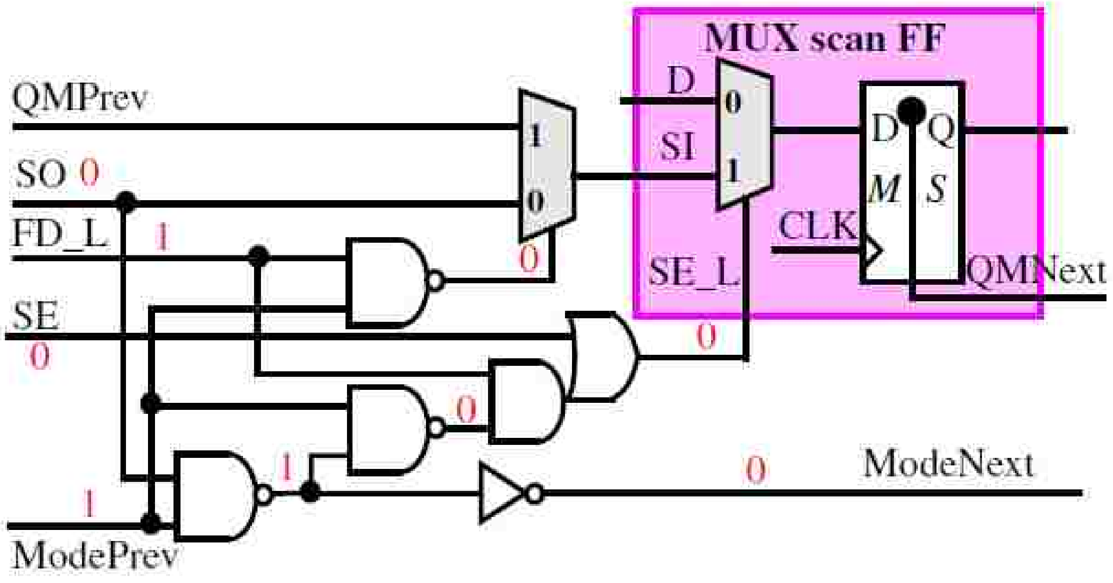

Fig. 4: REBEL Support Logic for MUX Scan

SCA_L and SCB_L of Fig. 2 can be eliminated. The key objective in the MUX scan implementation is to implement a FD mode, i.e., a combinational path, using the latches within the MUX scan FFs. This can be achieved by adding a 'tappoint' to the master latch, called QMNext in Fig. 4, and routing this signal to a 2-to-1 MUX in the next scan FF of the scan path (labeled QMPrev in Fig. 4). The SE input in Fig. 4 refers to the globally routed scan enable signal (already required for MUX scan). SE is set to '1' when we are in scan mode, and '0' when in functional or FD (REBEL test) mode. The remaining logic gates are inserted to implement the four conditions described earlier.

For example, to configure a row in functional mode (row 1 in Fig. 1), the RCL block places a '0' on the FD_L wire. To configure a row in FD mode (row 3 in Fig. 1), the RCL block sets FD_L to '1' and ModePrev to '0'. For a split mode row (row 2 in Fig. 1), the same scan FF encoding method described for CLSSD is used. In addition, the RCL block forces a '1' onto FD_L and sets ModePrev to a '1' for insertion points other than the left-most scan FF in the row or ‘0’ otherwise. The annotation in Fig. 4 shows the values of the scan FF at the point of PUT insertion for a split mode row. The REBEL implementation using MUX scan is actually smaller in overhead and is less invasive to the functional path (only one capacitive load is added at the tap-point in the master latch) than it is for CLSSD.

4.3. PUT Delay Analysis Process

The Launch/Capture delay in REBEL is controlled by CLK, as described earlier, and therefore REBEL leverages the CLK tree for critical timing events.

A REBEL test is carried out as follows:

1. Configuration data is scanned in.
2. The global SCA and SCB signals are asserted.
3. CLK is asserted to launch a transition into the PUT.
4. CLK is deasserted after a specific Δt, sufficiently long to allow the transitions on the PUT to propagate along the scan chain.
5. The global SCA/SCB signals are deasserted, and the values in the scan chain are scanned out.

The delay in the combinational path is computed using **Eq. 1.**

$$\boldsymbol{T_{PATH} = T_{LC} - T_{SC}} \qquad \textbf{Eq. 1}$$

where:

$\boldsymbol{T_{PATH}}$ = Delay in the combinational path

$\boldsymbol{T_{LC}}$ = Launch/Capture Delay

$\boldsymbol{T_{SC}}$ = Delay in the Scan Chain

The scan chain delay, $\boldsymbol{T_{SC}}$, can be calculated from the number of scan cells that are set by the propagating edge(s), and the data obtained from a set of calibration tests[1].

1 The calibration of the scan chain propagation delay is straightforward and performed by performing repeated timing tests with successively longer Launch/Capture intervals, recording the intervals in which a transition reaches each FF in turn. However, for the purposes of the HELP PUF, this calibration is not required; in this case, we are only concerned with the interval required to propagate to a known point in the scan chain, and not with the actual delay of a path in the MUT (as we would be for a DFT application).

Chapter 5: Preliminary HELP PUF Research

As noted in Chapter 4, two of the potential applications envisioned for the REBEL ETS at the time of its design were for measuring process variability and as the basis for a delay-based PUF design. If the REBEL embedded test structure could be used to determine the amount of process variations in a semiconductor manufacturing process, then it could also be used to measure those variations, store them temporarily in memory, and use those measurements as the basis of comparisons for generating a unique bitstring for an individual chip.

The initial investigation of the REBEL structure as a the basis for a PUF was conducted as a series of tests on two sets of path delay data collected during REBEL testing. The first testing of REBEL was done on an FPGA implementation of a stack of three SBOX macros taken from an AES implementation. The design, including the REBEL functionality, was configured in a Xilinx Virtex2-Pro FPGA, while the testing resources, including the scan pattern generation, the clocking for the launch and capture edges, and all of the control logic was provided from off-board by an Inovys Ocelot structural DFT tester. A high-performance differential clock signal was provided by the tester; however, the launch and capture clock signals consisted of consecutive rising edges on the same clocking resource. This contrasts with the separate DCM outputs that are used on the FPGA version of the HELP PUF engine discussed in the following chapter.

The early experimentation with the HELP PUF concept was all done using software manipulations of the scan patterns acquired during the REBEL testing of the design. Nonetheless, the concept appeared to be successful in its primary mission of extracting a meaningful amount of entropy from the same path timing measurements across a set of the FPGA boards. The performance of the PUF was somewhat limited by the excessive jitter introduced by the differential clock on the tester. Also, the limitations of the off-chip analysis, in terms of the amount of data required to prove even a small amount of entropy, soon made it apparent that a specially designed PUF engine on an FPGA was needed to further advance the study.

Chapter 6: FPGA-based HELP PUF Engine

To prove the concept of the HELP PUF introduced in Chapter 3, and to conduct research studies on its performance, we have designed and implemented a complete, self-contained, fully functional FPGA-based HELP engine. The design for the PUF engine was written in VHDL and the engine was created using the ISE design suite from Xilinx Corporation[14].

The target FPGA platform was the Xilinx Virtex-II Pro device (P/N: XC2VP30). The development board that we used for our experimentation was an XUP V2-Pro board from Digilent, Inc. [25]. We chose this development board because we had access to 30 of these boards, which provided a sufficient population size to allow us to conduct meaningful statistical evaluations of the PUF's performance. In addition, this board is equipped with jumpers that can be removed to disable the on-board voltage regulator that supplies the 1.5V internal logic voltage (Vcc_{INT}) for the FPGA, and screw terminals that facilitate the use of a programmable power supply for controlling Vcc_{INT} programmatically. Finally, the layout of the XUP V2-Pro board allows for placement of a heat pump directly on the surface of the Virtex-II Pro for automated control of the chip temperature. These features make this platform ideal for conducting environmental testing across a range of temperatures and voltages. We provide additional details of our environmental testing regimen later in this chapter.

The overarching design goals of the HELP PUF engine were to create and demonstrate a hardware security primitive that: 1) used the REBEL path delay

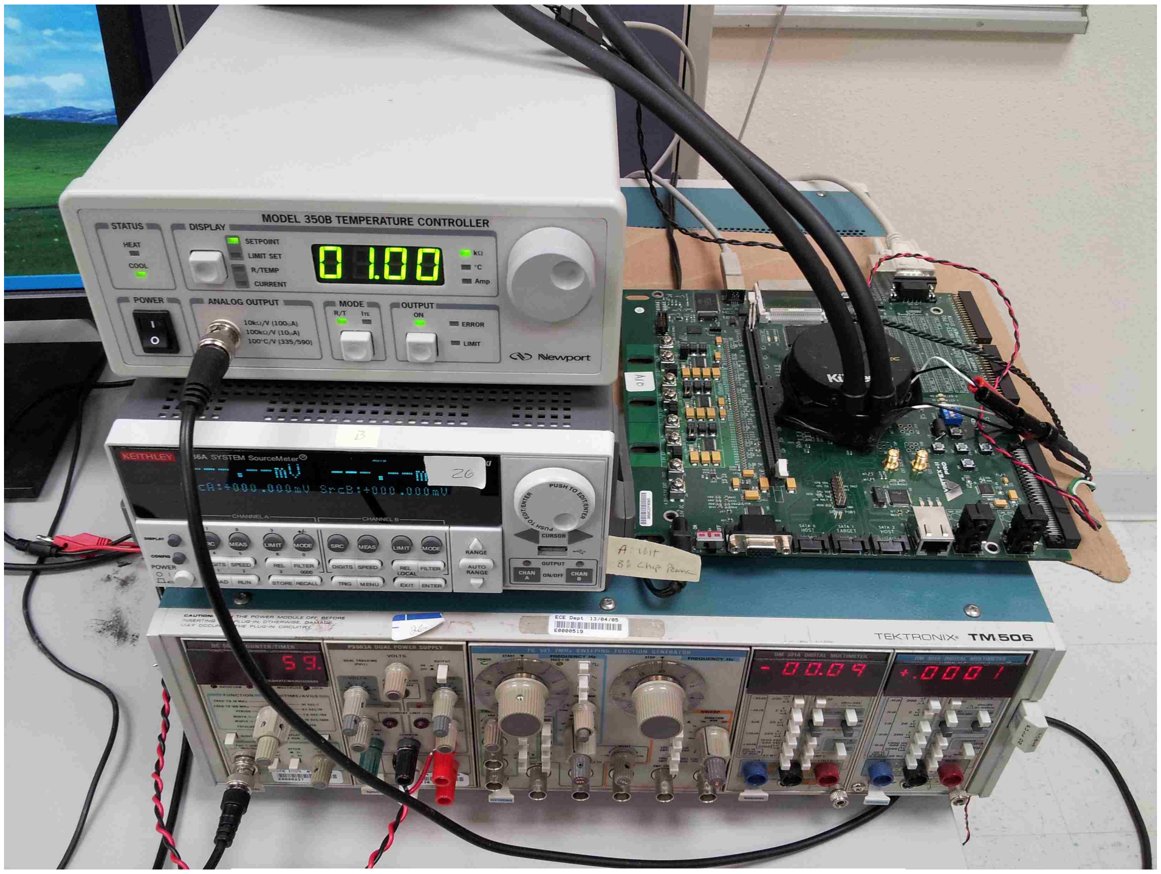

Fig. 5: HELP Experimental Test Setup

measurement structure presented in Chapter 4, 2) could be added to an existing block of combinational logic, and 3) was capable of generating cryptographic-quality bitstrings the length of which is subject only to available on-chip block RAM (BRAM) memory. Based upon our experimental results and the results of our randomness testing, we believe that we have successfully met and exceeded each of these design goals with this implementation. A photo of the XUP board and the test apparatus is shown in Fig. 5.

The HELP PUF design is comprised of several component parts. We start by augmenting the existing logic design with the REBEL test structure, and adding several memories, clocking circuitry, random sequence generators, stability analysis components,

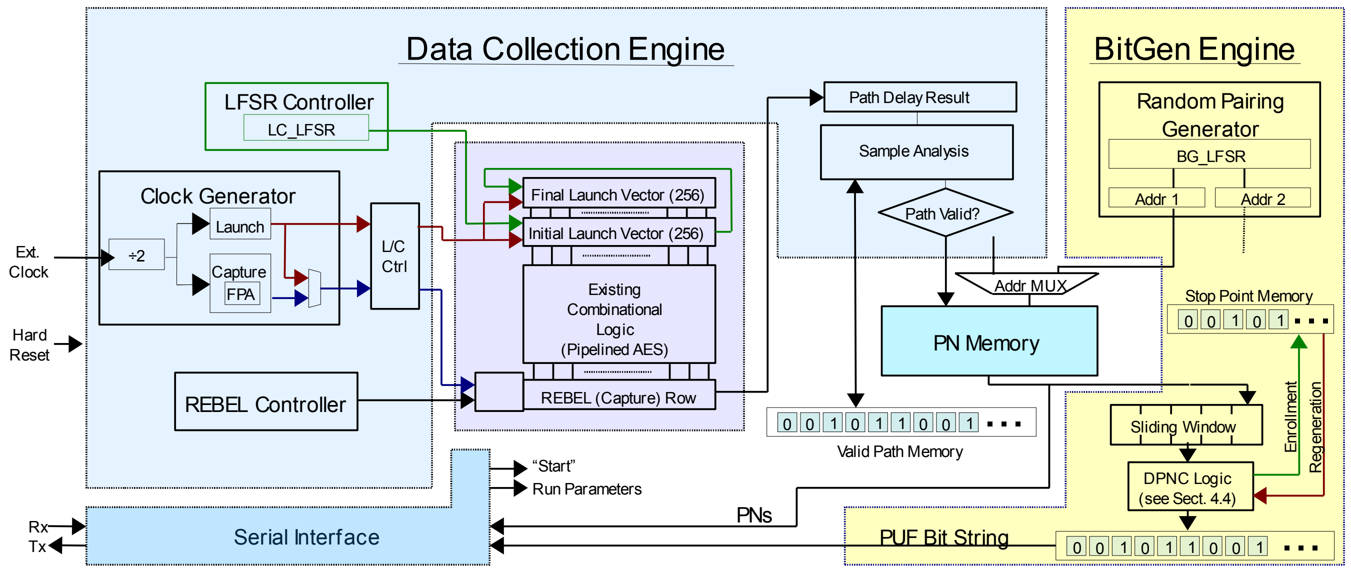

Fig. 6: HELP Structural Block Diagram

and functionality for reliably generating or regenerating the bitstring.

The operation of the HELP PUF (shown in block diagram form in Fig. 6) is divided into two distinct phases. In the first phase, a Data Collection Engine phase, the propagation delays of paths in the MUT are measured, digitized into integer values that represent the path delay, analyzed for stability, and recorded in memory. The second phase of operation is the Bit Generation phase, in which the path delays stored in memory are compared with each other to generate a sequence of binary comparison results in the form of a bitstring. The user requests a bitstring of a specific length, and the data collection process continues until a sufficient number of paths have been tested to allow the Bit Generation process to create a bitstring of the desired length.

The remaining sections of this chapter describe the subparts of the HELP PUF engine and their operation.

6.1. Macro-Under-Test

The macro-under-test, or MUT, used in our implementation is the logic defining a single round of a pipelined Advanced Encryption Standard, or AES [1], implementation from OpenCores.org [26]. Other AES implementation choices in the public domain included non-pipelined, recirculatory designs in which the same hardware is reused for each round[2]. We chose this pipelined implementation because its non-recursive datapath design provided a more stable platform on which to test our PUF concept, and avoids the complicating effects on path timing that are introduced by routing outputs of the macro back into the inputs.

Space limitations on the Virtex-II Pro prevented inclusion of all 10 rounds[2] of a full AES implementation. The block labeled 'Initial Launch Vector (256)' represents the pipeline FFs in the full-blown AES implementation, converted here to MUX-D scan-FFs. A second copy of this block, labeled 'Final Launch Vector (256)', is added to emulate the logic from the omitted previous round. In our implementation, two randomly generated vectors that represent the challenge are scan-loaded into the two blocks.

The original design of the AES core included the logic for ten pipelined rounds, and each round was itself divided into three pipelined stages (addkey, sbox, and colmix). This design is intended as a high-capacity, high-throughput standalone AES engine, capable of encrypting/decrypting as many as 30 128-bit datablocks at a time. A full implementation of this design requires an FPGA with a rather large number of sequential resources to

2 The encryption algorithm defined in the AES standard [1] operates on 128-bit data blocks, and allows for three different key lengths (128-bit, 192-bit, and 256-bit). The number of rounds corresponding to these key lengths is 10, 12, and 14, respectively. The underlying algorithm is identical for all key lengths.

support this level of pipelining, and was not suitable for our purposes. For our HELP PUF engine, we chose to isolate a single round, and to eliminate the sequential elements (registers) within the round, resulting in only combinational logic with a row of registers at the output of the block. We replaced the FFs in these registers with REBEL FFs, which are described in Chapter 4. In addition, we added two 256-element rows of FFs on the input of the MUT which contain the initial and final vectors in a launch-capture test.

6.2. Data Collection Engine (DCE)

The DCE in Fig. 6 carries out a sequence of LC tests, measures the path delays, and records the digitized representation of them, called PUF numbers or **PNs**, in block RAM on the FPGA. In our current implementation, the DCE runs to completion before the BGE component is started[3].

6.2.1. Clock Generator

The clock generator module generates two clock signals: a Launch clock and a Capture clock, and is shown on the left in Fig. 6. In our design, this module contains three digital clock managers, or DCMs. A 'master' DCM is used to reduce the off-chip oscillator-generated 100 MHz clock to 50 MHz. The output of the master DCM drives the Launch and Capture DCMs. We utilize the fine phase adjustment (FPA) feature of the Capture DCM to 'tune' the phase relationship between the Launch and Capture clocks. At 50 MHz, the FPA allows 80 ps increments/decrements in the phase shift of the Capture

3 While not addressed in this dissertation, the potential modification of the HELP engine to allow the data collection and the bit generation engines to operate simultaneously would offer two advantages: a reduction in the time required to perform a HELP PUF operation, and a means by which to obfuscate the individual operations that would be subject to targeting in a differential power analysis attack by an adversary.

clock on the Virtex-II Pro chips.

When the DCE is configuring the scan chains in preparation for the LC test, the phase relationship between the Launch and Capture clocks is eliminated without an adjustment to the phase of the Capture clock by using an FPGA primitive called a BUFGMUX, which allows the Launch clock to be routed to the clock inputs of the REBEL row for the shift operation. Prior to the launch event, the controlling state machine selects the 180° phase-shifted output of the Capture DCM, and the FPA feature is used to tune the phase in an iterative process designed to meet a specific goal (to be discussed).

TABLE I. CAPTURE CLOCK PHASE ADJUSTMENT

Phase Adj.	**Phase Angle**	**LC Interval**
0	90°	5 ns
64	180°	10 ns
128	270°	15 ns

TABLE I. summarizes the characteristics of the Capture clock, and Fig. 7 illustrates the timing relationship between the Launch and Capture clocks for different values of the 'Phase Adj.' control counter in the DCM. The launch and capture events occur on the rising edge of the corresponding clocks. From the timing diagram, this allows path delays from 5 ns to 15 ns in length to be measured. The 0 to 128 range of values (called PUF numbers, or PNs) act as a

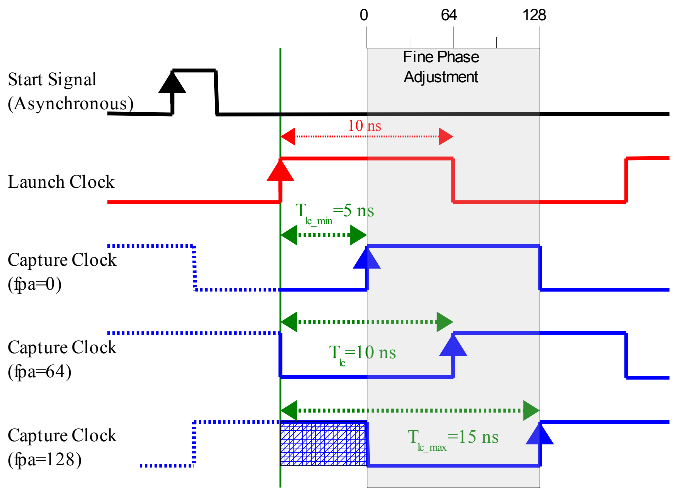

Fig. 7: Launch/Capture Timing Diagram

digital representation of the path delays.

6.2.2. PN Memory

The PN Memory is a block RAM used to store the PNs. The size of this memory block can be made as large or as small as required to efficiently store the PNs required to generate a bitstring of a given size; however, the data width must be 8 bits, as the possible Launch-Capture interval values spans the range of 0 to 128, inclusive.

6.2.3. LC LFSR Controller

The bit sequences that represent the *challenges* in the traditional language of PUF research, and that are shifted into the Initial Launch Vector and Final Launch Vector scan rows of the MUT, are generated by a 32-bit linear feedback shift register (LFSR) contained in the LC LFSR controller. An additional register and control signals are manipulated by a finite state machine (FSM) to cause the LC LFSR to be operated in one of three modes. The first mode initializes the LC LFSR to a seed value provided as a parameter by the HELP PUF user. Another mode stores the current value in the temporary seed register and permits the LC LFSR controller to repeatedly issue the same 512-bit launch vectors for repeated testing across multiple samples and across multiple insertion points (see Chapter 4). Finally, a third mode retains the present LC LFSR contents, effectively advancing the LC LFSR to the next 512-bit subsequence, based upon the LFSR configuration.

6.2.4. REBEL Controller

The REBEL controller, while not part of the REBEL design, is used by the DCE to configure the IP in the REBEL row attached to the output of the AES logic block in advance of each launch-capture interval of each sample of each LC Test (path timing) sequence.

6.2.5. Sample Analysis Engine (SAE)

This FSM analyzes the digitized results in the delay chain after each LC test for a given path and determines whether the path is 'valid'. **A valid path is defined as one that has a real transition, is glitch-free, and produces consistent results across multiple samples**. If a PN is deemed to be stable, the PN is stored in memory, and a "valid path" flag is set in the Valid Path Memory (see below) corresponding to that path name.

6.2.6. Valid Path Memory

A block RAM is used to record a pass/fail flag for each tested path that reflects its validity (as defined by the criteria for the SAE). These flags are set/cleared during enrollment and then *played back* from non-volatile or off-chip memory (public storage) during regeneration, and represent the helper data needed in the regeneration process.

6.3. Bit Generation Engine

The Bit Generation Engine is comprised of a collection of hardware components and state machines that support the use of the PNs stored in PN memory to create the bitstrings. Included in the Bit Generation Engine are an LFSR-based pseudorandom pairings generator that randomizes the order in which PNs are read from memory, a block of BRAM that is used to store public data (in addition to the Valid Path Memory in the DCE) for properly recreating the bitstring during regeneration, and finally, the BRAM which contains the final bitstring.

Chapter 7: Bit Generation Techniques

The ultimate objective of the HELP PUF is the creation of bitstrings that can be used for any of a number of purposes, ranging from cryptographic keys to chip identification and authentication to hardware support for random number generation. As stated in the introduction of this dissertation, the quality of the bitstring that the HELP PUF produces is dependent upon several performance metrics, such as uniqueness, repeatability, randomness, and efficiency.

To accomplish this aim, several techniques for generating bitstrings were created and implemented. This section of the thesis provides a detailed examination of those techniques, and a comparative analysis of the strengths and challenges of each.

7.1. Modulus

The genesis of the HELP PUF began with the germ of an idea – that the variation in path delays present in an existing functional logic macro could provide the foundation for a PUF. However, attempts to capitalize on this phenomenon soon presented a number of challenges. The first issue that was clearly evident was that, unlike other delay-based PUFs such as the RO PUF or the Arbiter PUF, path delays in a macro are not specially designed to be all the same, and in any such logic macro, the presence of a large gross difference between any two path lengths would result in a bias that would overshadow any amount of process variation for that comparison, rendering the comparison unusable since all instances would result in the same evaluation. At the other extreme, environmental and measurement noise effects would necessarily result in some variation being non-repeatable, unstable, and therefore unusable.

To address these issues, the path delay measurements came to be characterized as being comprised of three distinct components. First, the bulk of the propagation delay in a given path is not subject to vary from one chip instance to another. A path in one chip that is, for example, 10 ns is not likely to be measured at, say, 15 ns in another chip. As a result, this path delay component represents a *constant* term, and in a digitized, quantized representation of this delay, this component will appear in the most significant bits (MSBs) of the digitized value. The second component of any path delay measurement will include the *noise*, or uncertainty (resulting from measurement noise, thermal or voltage fluctuations, quantization errors, etc.), and will be found in the least significant bits (LSBs) of a digitized delay value. The third component of delay measurement is the *variation* in the signal (resulting from process variations that occur during the chip's manufacture). This variation is found in the bits of a quantized value that are constant across repeated measurements and changing environmental conditions for the same device, but that are found to be different between the same paths on different chips.

It was theorized that it may be possible to make use of the digitized representation of a pair of path delays to allow the direct comparison of the variability of any two paths, regardless of their path length. To do this, we examined a large number of paths and determined, empirically, the magnitude of the uncertainty across a series of repetitive measurements of the same path on the same chip. We then compared the delay values of the same path on several chips to acquire a sense of the magnitude of the variability

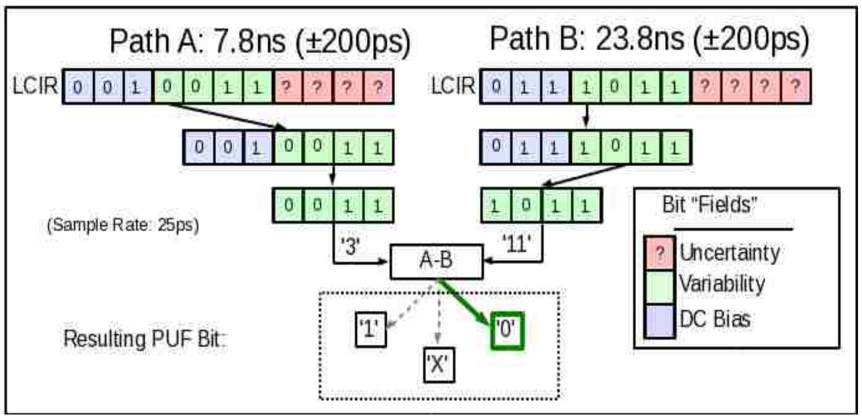

Fig. 8: Modulus-Based Path Delay Comparison (deprecated)

found in the population. The example in Fig. 8 illustrates this technique, and shows how the digitized value can be apportioned into a single set of bits (shown in green in the digitized values of the figure) that represent the variations between chips. In the example, the uncertainty was found to be < 200 ps, and the variations as high as 3.2 ns (2^4, or 16 200-ps intervals). The upper bits in the values represented the gross path length, and were discarded, along with the 4 LSBs that contained the uncertainty (noise).

Ultimately, this technique, while initially promising, was found to be unworkable due to non-uniformity in the distribution of the variations among the 16 values represented by the 4 bits. Additionally, as shown in Fig. 9, the noise was largely determined to be less than 200 ps; however, the outliers that are evident show occasional uncertainty of as much as 1.8 ns. As a result, the Modulus technique was ultimately set aside in favor of more robust methods described below.

7.2. Dual P/N (DPN) Path Delay Binning Method.

With the failure of the Modulus method to reliably generate unique bitstrings, another similar method was explored in which the modulus was not specifically required to be a power of two (so as to be described by a specific number of bits), but rather an integer value, derived empirically, that more closely approximates the amount of true process variation found across the population of chips. The drawing in Fig. 10 on page 42 illustrates this concept. Using this technique, a modulus operation is applied to the PUF numbers returned by the Data Collection Engine. The resulting values, referred to as Mod-PNs, fall into the range of [0..M-1], where M is the empirically derived modulus number, which includes both the *variation* and the *noise* components described earlier, while removing the *bulk* of the path length. This results in the ability to directly compare paths whose lengths may be vastly different (such as the 7.8 ns path and the 23.8 ns path in Fig. 8) without the bias that would normally preclude such a measurement.

The original intent of developing this technique was to then evaluate pairs of Mod-

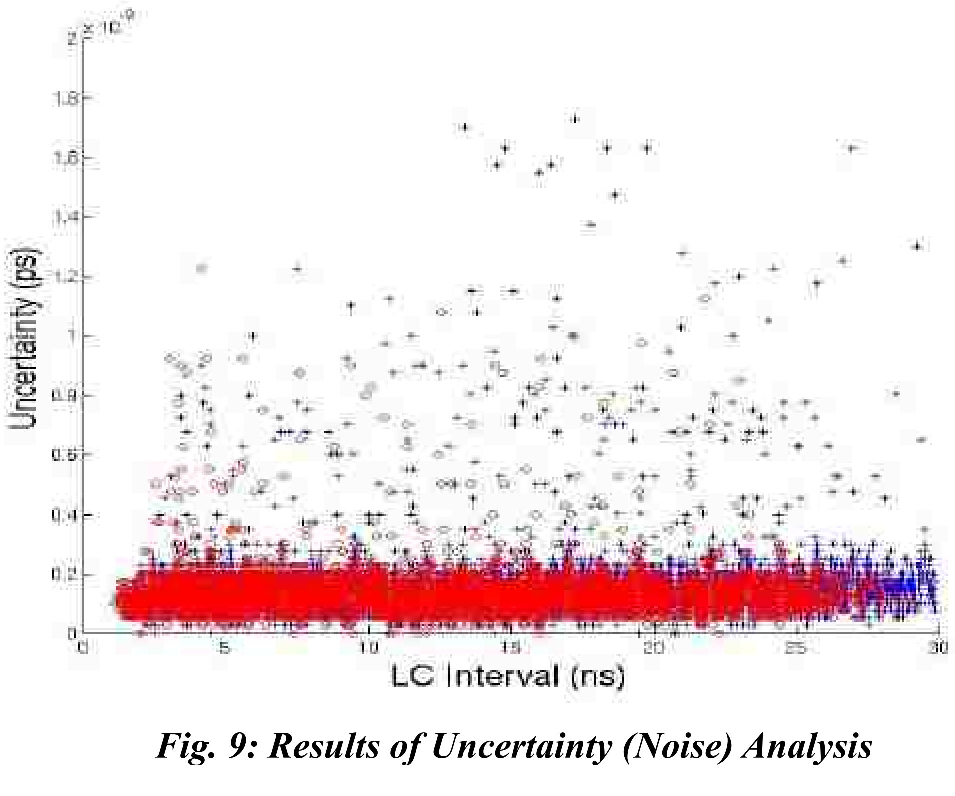

Fig. 9: Results of Uncertainty (Noise) Analysis

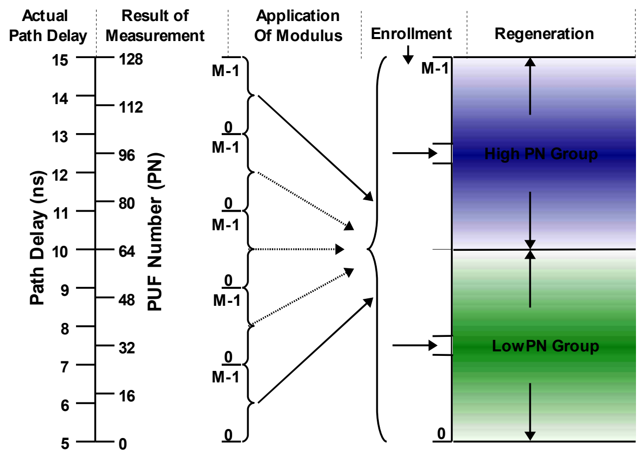

Fig. 10: Dual-PN Path Binning Method

PNs, using an XOR-style comparison, and generate a '0' if the Mod-PNs were from the same "group", and generate a '1' if they differed. However, this technique alone was found to be insufficiently robust to accommodate the uncertainty present in the data. Additionally, the occurrence of jumps in path delay presents an intractable hurdle for making this technique sufficiently reliable to be the basis of a bit generation technique.

However, the DPN method has proved to be an effective means of binning the distribution of path delays for the MUT into a binary pair of groups that subsequently form the basis of a more robust bit generation technique. What is required is a means of providing hardware-based error tolerance for those relatively infrequent cases where a PN measured during regeneration is incorrectly partitioned, due to jumps or other factors.

7.3. The Dual-PN Count (DPNC) Bit Generation Method

Most PUF are designed using identical circuit primitives as a means of avoiding bias. This is not the case for HELP, because the PUTs vary widely in length. We developed a technique called 'Dual-PN Count' which post-processes the PNs to eliminate this bias. The technique applies a modulus operation to the PNs, which 'trims off' the higher order bits of the path delay measurement. The truncation of the PNs effectively reduces all path delays to a range upper-bounded by the modulus, i.e., it reduces the overall path length to a range more closely consistent with the degree of variations found in the data and allows unbiased comparisons to be made among all paths. The trimmed Mod-PNs are then partitioned into two groups for bit generation purposes.

The diagram in Fig. 10 provides a graphical depiction of this two-step process. The process begins on the left using a PUT with a delay between 5 ns and 15 ns. The measured PN for this PUT is originally in the range 0 to 128, but the modulus operation reduces it to a number in the range of ***0 to M-1*** (where ***M*** is a user-specified modulus). The right-most portion of the diagram in Fig. 10 shows the partitioning of the Mod-PNs into two groups, where values in the range of ***0 to M/2-1*** are placed in the low PN group, while Mod-PNs in the range of ***M/2 to M-1*** comprise the high PN group. Noise in the measurements is dealt with by discarding additional PNs (beyond those discarded because of path stability problems as described in a previous section of this thesis). In particular, Mod-PNs that fall into regions outside those delineated in the center portion of Fig. 10 are considered invalid during enrollment. This allows valid PNs, i.e., those that fall within the center portions, to 'shift' during regeneration by up to *M/4* in either direction before causing a bit flip. Therefore, this scheme both eliminates bias and adds bit flip resilience to HELP.

7.3.1. Bit Generation using DPNC

The filtering operations described above are sufficient to eliminate the adverse effects on delay introduced by noise and TV variations. However, large changes in the Mod-PNs introduced by “jumps”, as described in Chapter 8, require a more resilient technique. The rare nature of “jumps” makes it possible to develop a bit-flip avoidance method that imposes a low area and time overhead. The 'Count' term in DPNC refers to this feature of the method, and characterizes the process used to generate bits, which is described as follows. During enrollment, DPNC parses the valid PNs until it encounters a sequence of ***k*** consecutive values from the same group, where ***k*** is an odd-numbered, user-specified threshold. Two counters track the length of a sequence of PNs from the same group. As each PN is read, the counter for the corresponding group is incremented, while the other group's counter is reset to 0. When either of the counters reaches ***k*** (indicating that the ***k*** most recent PNs belong to the same group), a new bit is generated and added to the bitstring, and a 'stop point' flag is set in the Stop Point Memory to indicate that a bit was generated at this point. The value of the generated bit is a '1' if the PNs are from the high PN group, and a '0' if the PNs are from the low PN group. During regeneration, the stop point flags (represented as a bitstring) are consulted to determine when bit generation occurs. Therefore, the bitstring of stop point flags represents additional helper data.

7.3.2. DPNC Example

An example of the DPNC process is shown in Fig. 11. The modulus is set to ***M*=22,** and the range of valid PNs accepted in the low PN bin are given by **{4,5,6}**, while the valid PNs for the high PN bin are defined as **{15,16,17}**. The value of counter ***k*** is set to 5. This example first depicts the enrollment process, in which PNs are read from the on-

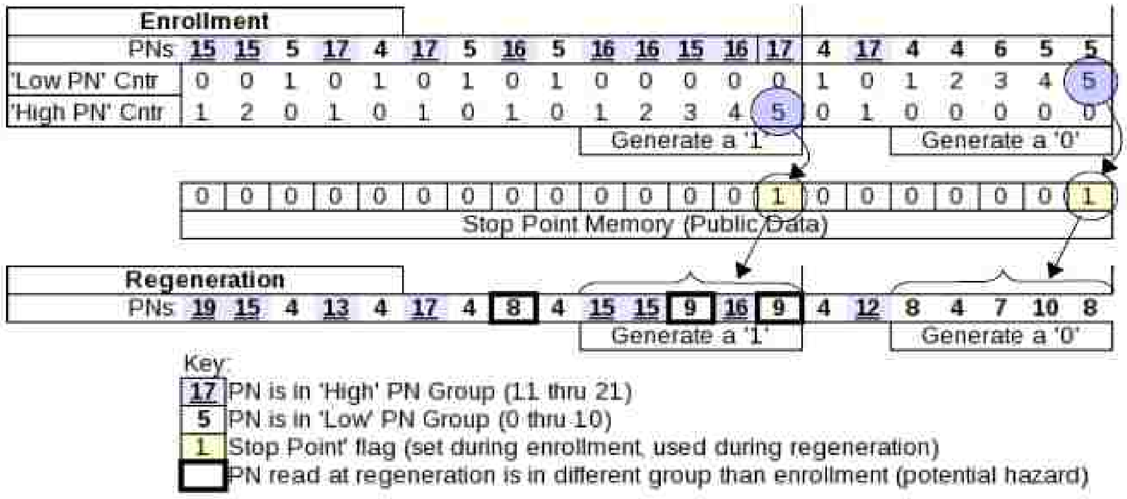
Enrollment
PNs 15 15 5 17 4 17 5 16 5 16 16 15 16 17 4 17 4 4 6 5 5
'Low PN' Cntr 0 0 1 0 1 0 1 0 1 0 0 0 0 0 1 0 1 2 3 4 5
'High PN' Cntr 1 2 0 1 0 1 0 1 0 1 2 3 4 5 0 1 0 0 0 0 0
Generate a '1'
Generate a '0'
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1
Stop Point Memory (Public Data)
Regeneration
PNs 19 15 4 13 4 17 4 8 4 15 15 9 16 9 4 12 8 4 7 10 8
Generate a '1'
Generate a '0'
Key:
17 PN is in 'High' PN Group (11 thru 21)
5 PN is in 'Low' PN Group (0 thru 10)
1 Stop Point' flag (set during enrollment, used during regeneration)
PN read at regeneration is in different group than enrollment (potential hazard)

Fig. 11: Dual-PN/Count Example

chip memory, left to right, as shown in the top of the figure. Also shown are the states of the counters after each PN is read. When the high PN counter reaches 5 (as shown in the circle), a '1' bit is generated and added to the bitstring (not shown), and a '1' is written to the current location in the Stop Point Memory. At this point, both counters are cleared and the process continues until a second bit (a '0' in this case) is generated. The bitstring is built up in this fashion one bit at a time, until a user-specified number is reached.

The bottom portion of Fig. 11 illustrates the process carried out during regeneration. Here, the '1' bits in the Valid Path Memory (not shown) indicate which paths were used for bit generation during enrollment, and dictate now those paths that must be re-tested for proper regeneration. Similarly, the '1' bits in Stop Point Memory (in Fig. 11) force bits to be generated at these points (the counters are not consulted). The counters, however, *are* consulted to determine the value of the generated bit, which is determined by the larger of the two counter values. In the example, two of the five values that were in the high PN bin during enrollment have 'flipped' and now appear in the low

PN bin (see elements highlighted with the heavy borders).

However, because the majority, 3 out of 5, are high PNs, the algorithm correctly regenerates a '1' bit despite the presence of the erroneous measurements. Also note that the first erroneous measurement (the '8' in the heavy border) is of no consequence since it is not part of the consecutive sequence of 5 PNs that are consulted to determine the value of the bit (these 5 PNs are identified in the figure with a curly bracket).

This seemingly innocuous observation constitutes a major strength of this technique. The initial intent of the DPNC algorithm called for simply incrementing the counter associated with the current PN and monitoring the difference between the two counters, generating a bit as soon as a *difference* of ***k*** was reached. The logic behind this approach was that a *difference* of ***k*** would occur much more frequently than a *sequence* of ***k*** would in a given sequence of PNs, resulting in less unused PNs and, as a result, greater efficiency and reduced running time for the algorithm. However, an analysis of the two approaches showed that this method resulted in only a **33%** increase in efficiency, while incurring **9.95 times** as many errors as the technique which requires ***k*** common-group values in a sequence. This analysis was the basis for the decision to implement the latter technique for DPNC.

7.4. Universal/No-Modulus Bit Generation Technique

In [27], we developed a method called “Universal-No Modulus” (UNM) that is capable of generating $O(n^2)$ bits from n PNs. UNM avoids bit flips by using only the longest and shortest paths in the MUT for comparisons, discarding paths of medium length. It avoids the bias that would normally result under these conditions by exploiting the property that **path stability** is random across chips. In other words, even though the result of comparing a short path with a long path is predictable from the design, the stability, and therefore selection, of short and long paths is random from chip to chip.

Figs. 12(a) and 12(b) show the path distribution from a typical chip, with the PN range plotted along the x-axis against 'number of instances' along the y-axis. During enrollment (Fig. 12(a)), UNM uses two thresholds to partition the distribution into 3 regions. The tail regions on the left and right are considered valid PN regions. PNs in the tails represent short (Low PNs) and long (High PNs) paths respectively. The large 'invalid' region between the thresholds, given as 32 and 90 in Fig. 12(a), is a safety zone between the groups designed to prevent 'jumps', and bit flips, between the Low and High PN regions. The placement of the thresholds determines the balance between the number of paths in each tail region, and are established using a process that characterizes the path-length distribution at the start of each enrollment. Jumps, although infrequent, can occur because of the appearance and disappearance of hazards (glitches) on side-inputs of gates along the tested paths. Small temperature variations or power supply noise influence the behavior of these hazards, in which changes in the delays of difference branch paths are affected nonuniformly. Examples of tolerable (green line) and intolerable (red line) jumps are shown Fig. 12, wherein the lines indicate PNs that were significantly higher during regeneration than they were during enrollment.

The safety zone is only enforced during enrollment, and is redefined as the midpoint between the margins during regeneration as shown in Fig. 12(b). The DCE process creates a valid path bitstring during enrollment so the same sequence of path tests can be carried out during regeneration. In our experiments, we found that UNM generates a valid PN after approx. every 20 tested paths, depending on the user-specified width of the 'invalid' region. The “XOR-style” bit generation process is carried out by comparing pairs of PNs, where PNs from the same region generate a '0', while those from opposite regions generate a ‘1’. With *n* PNs, up to *n*(n-1)/2* bits can be generated by considering all combinations.

7.4.1. Weakness of the Universal/No-Modulus Method

The Universal/No-Modulus bit generation method is robust in the presence of jumps that do not exceed at least one-half the size of the safety gap. However, the net

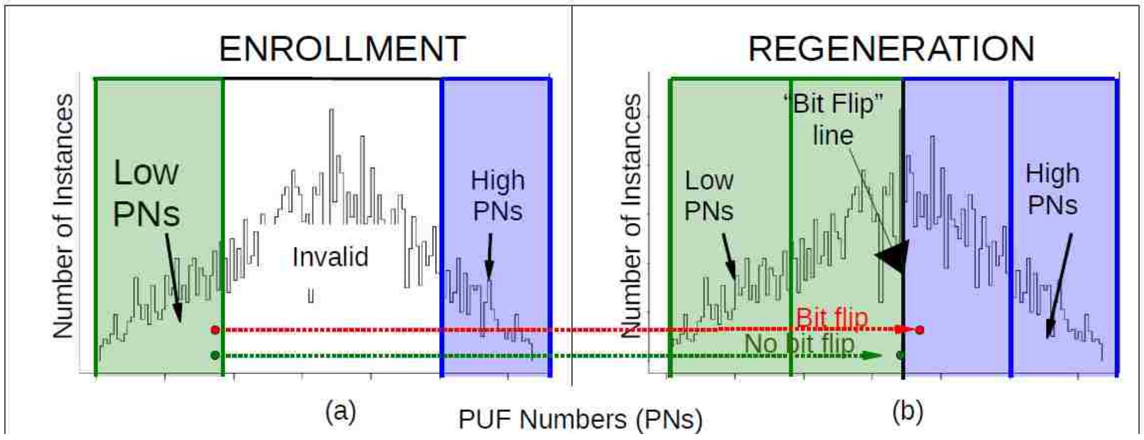

Fig. 12: Universal/No-Modulus Bit Generation Technique

effect of imposing this large gap in the allowable PNs during enrollment is a dramatic reduction in the number of stable paths that can be used for bit generation. This creates three effects, all of which adversely affect the utility of this technique. First, the efficiency, in terms of the number of paths that must be tested for a given bitstring length and the running time of the algorithm, is severely limited. Additionally, the size of the public storage requirement increases in direct proportion to the increased demand for paths tested. Finally, the amount of *excess entropy* for a given macro-under-test is reduced, increasing the probability that a reduction in performance against standardized randomness testing will occur.

7.5. Universal/No-Modulus Difference (UNMD) Bit Generation

In response to these weaknesses in the UNM bit generation technique, in [28], we presented the HELP PUF and a bit generation technique called Universal/No-Modulus (UNM). We investigate a variant of this UNM technique in this section. Unlike the DPNC described above, UNM leverages the randomness associated with the stability of paths across chips and, as a result, there is no need to consider bias, i.e., UNM can compare short paths with long paths directly without first reducing the overall path length of the PNs, as is true for DPNC. The technique described in [27] defines a low and a high PN bin (similar to DPNC), but with the bins defined in this case over the entire path distribution range from 0 to 128. A large margin of approx. 100 is created between the bins to allow for shifts and jumps in the PNs during regeneration. The original technique therefore discards a large fraction of PNs that fall within this margin during enrollment (beyond those discarded because of path stability problems as described in Chapter 4).

We refer to the variant described here as 'UNM Difference' or UNMD. In UNMD, we replace the fixed margin with the concept of a noise threshold, discussed below. By doing so, UNMD does not discard stable PNs as is true of UNM, but rather preserves and makes use of all PNs generated by the DCE. This feature reduces the workload imposed on the DCE to find a suitable set of PNs that meet a bitstring target by **95.8%** when compared with the original fixed threshold technique. As we will show, UNMD offers significant advantages in both running time and memory requirements.

7.5.1. Bit Generation Process and Procedure

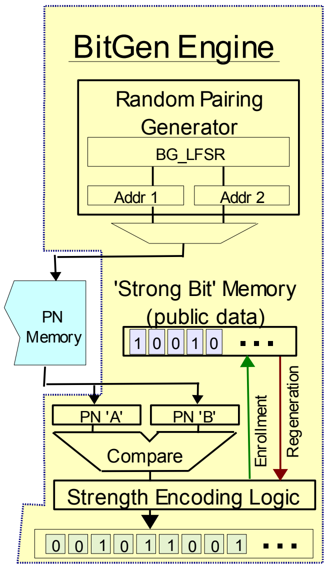

Fig. 13: Bit Gen Engine (UNMD)

All components except for the BitGen Engine in Fig. 6 on page 32 are identical for both the DPNC and UNMD techniques. The BitGen Engine for UNMD, shown in Fig. 13, randomly selects two PNs to compare (unlike DPNC which parses the PNs one at a time as shown in Fig. 6). The Random Pairing Generator produces the two addresses of the PNs to compare and the values are read from on-chip memory into a pair of registers (PN 'A' and PN 'B'). PN 'B' is then subtracted from PN 'A' to produce a PN difference. The magnitude of the difference determines the *strength* of that pairing, as discussed in the next section. If that difference is sufficiently large, then the *sign* of the comparison determines the value of the generated bit. A negative sign produces a '0', and a positive sign produces a '1'.

7.6. Thresholding Technique

A thresholding technique similar to that proposed in [29] is used to decide if a given comparison generates a strong bit (which is kept) or a weak bit (which is discarded). Thresholding works as follows. During enrollment, a *noise threshold* is defined using the path distribution histogram for the chip. The histogram is constructed using all *n* PNs collected by the DC engine. The noise threshold is then computed as a constant that is proportional to the difference between the PNs at the 5 and 95 percentiles in this distribution. Therefore, each chip uses a different threshold that is 'tuned' to that chip's overall (chip-to-chip) delay variation profile.

For each comparison, the difference between the two PNs is compared against the noise threshold. A strong bit is generated if the magnitude of the difference exceeds the threshold, otherwise the bit is discarded. Simultaneously, a bit is added to the 'Strong Bit Memory' shown in Fig. 13 that reflects the status of the comparison, with a '1' indicating a strong bit and a '0' indicating a weak bit. During regeneration, the Strong Bit Memory is consulted to determine which comparisons are used to regenerate the bitstring.

Fig. 14 shows the path distribution for a typical chip. The dashed lines indicate the 5 and 95 percentiles, with PNs of 23 and 117 respectively. The difference between these PNs is multiplied by a noise margin (0.90 in this example) to compute a noise threshold of 84.6. Pairings which differ by more than this threshold form 'strong' bits, while pairings that differ by less than this threshold are deemed to be 'weak' and will be discarded. The 'pairings' in Fig. 14 illustrate this concept.

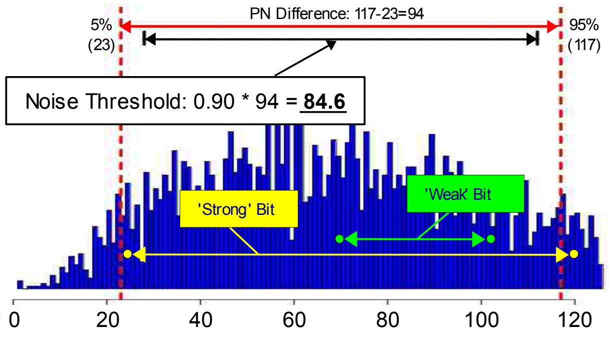

Fig. 14: Illustration – UNMD Technique

7.7. TMR-Based Error Correction Scheme

In Chapter 8, a detrimental behavior referred to as “jumping” is described as a worst-case condition and these discontinuous jumps represent our biggest challenge in dealing with bit flips. Both DPNC and UNMD are adversely impacted by jumps. In our experiments, some path delays changed because of jumps by as much as **4.5 ns**, or **58 PNs**, at different TV corners. Moreover, the PN differences computed by UNMD exacerbate the problem, where jumps in two path delays can combine in a worse-than-worst-case fashion.

This is illustrated in the graphs of Fig. 15, which depict data from one of the Virtex-II boards. The graphs plot the 'strong bit' number along the x-axis against the PN differences on the y-axis, with the noise thresholds (as described above) set to ±77.4 for this Virtex-II board. The data points from enrollment on the left all fall above or below these thresholds (by definition), but data points from measurements taken at different TV

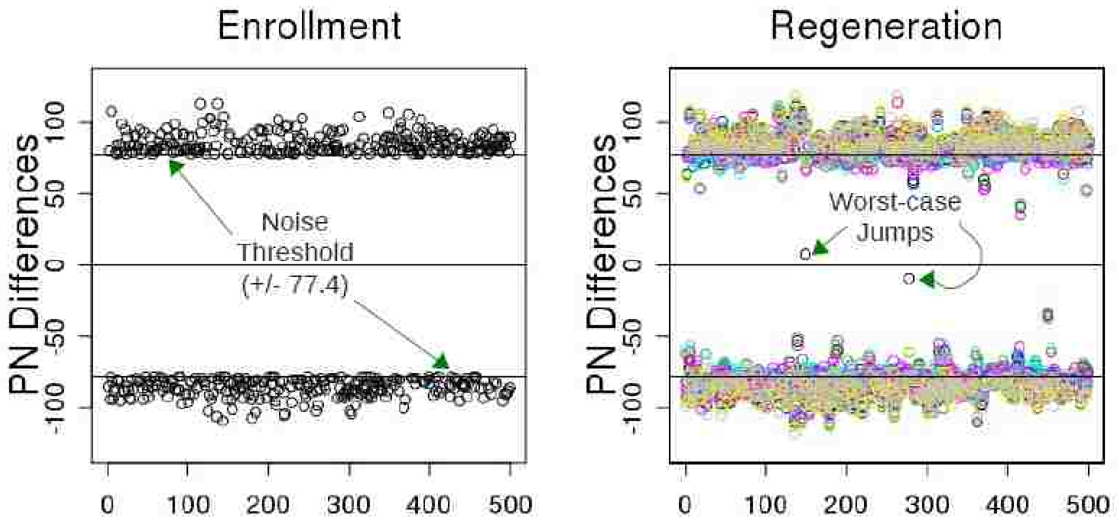

Fig. 15: Effect of Thresholding Technique on Reliability

corners in the graph on the right 'infringe' into the space between the thresholds. Most data points remain close to the thresholds, but some move significantly (because of jumps), as highlighted, by as much as **5.6 ns** or **71 PNs**.

By choosing a conservative noise threshold, bit flips caused by jumps such as those shown in Fig. 15(b) can be avoided. However, a different strategy is needed in cases where the application requires the probability of a bit-flip to be negligibly small (e.g., encryption). We proposed a technique in [28] that is based on a popular fault tolerant technique called triple modular redundancy (TMR), which is capable of reducing the probability of failure to values below **1e-11**. The method constructs 3 copies of the bitstring (using the abundance of bits provided by the PUF) and uses majority voting to construct the final bitstring. The probability of a bit-flip error is significantly reduced because any single bit-flip that occurs in any column of bits defined by the 3 copies can be tolerated. Probability of failure is investigated in Chapter 9.

Chapter 8: Reliability Enhancement Techniques

Throughout the course of the experimental work with the HELP PUF, two persistently troublesome issues have emerged as the primary sources of the errors in our bitstring generation techniques.

The first is the change in path delay that is caused by one or more changing environmental conditions, primarily in supply voltage and operating temperature. The impacts of these changes on the propagation delay through a given path are consistent with the drain current equations for the complementary MOSFET devices that comprise the gates along the path, and are therefore somewhat predictable and consistent.

The second source of errors, and the least easily predicted or mitigated, are characterized by discontinuous "jumps" that can cause the delay for a given path to change in abrupt, often large amounts of as much as 4 ns to 5 ns.

A third, less problematic error mechanism consists of the subtle differences in the distributions of the path lengths across the range of paths measured for each individual chip. These differences manifest themselves in two distinct modes: the first mode is a apparently random, normal distribution of mean path values throughout the population of boards, and the second mode is marked by a distinct shift in the means of two speed grades (-6, -7) for the devices mounted on the two boards.

To correct for these sources of errors in the PN measurements, the following reliability enhancement techniques were devised and implemented in hardware: a temperature/voltage compensation technique (TVCOMP) for addressing the changes in

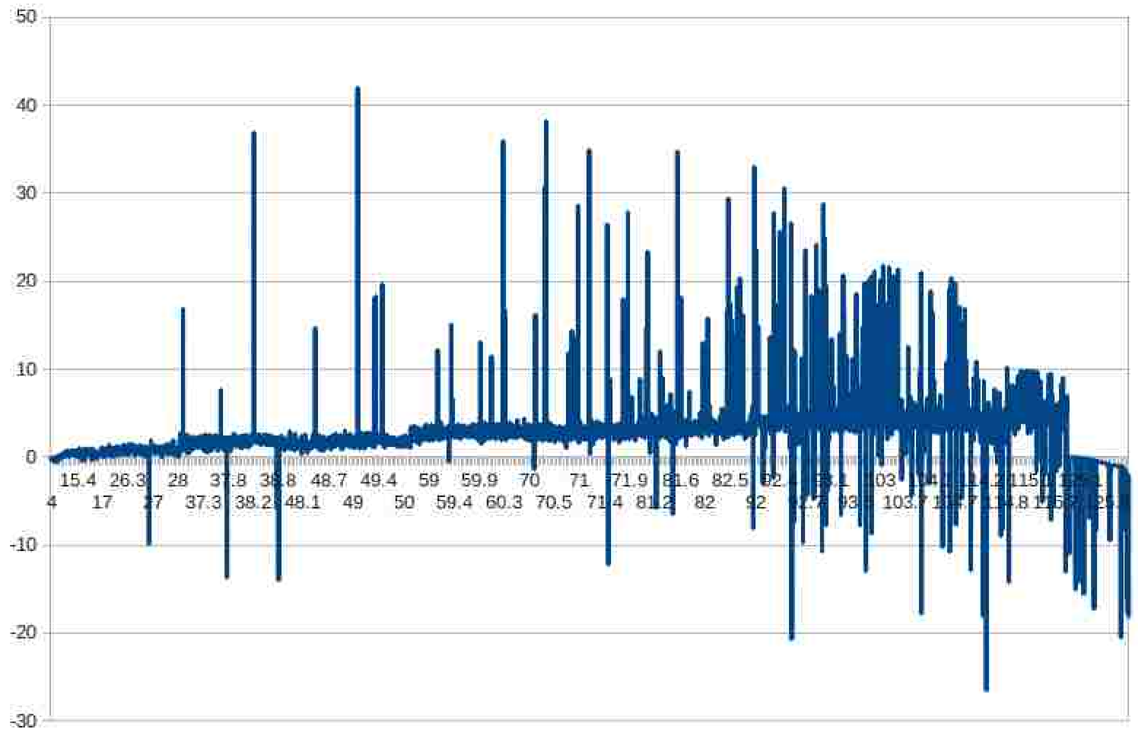

Fig. 16: Jump Behavior vs. Path Length

environmental conditions and a path distribution characterization technique (PDIST) for addressing the differences in path length distribution between devices. Unfortunately, the "jump" behavior described earlier, and depicted in Fig. 16, has been found to be too inconsistent and too unpredictable to be solved using the types of broadly applied corrective measures described here. The only techniques that have been shown to be effective against these jumps are the error-tolerance features included in the bit generation techniques in the previous chapter.

8.1. Temperature/Voltage Compensation (TVCOMP)

To adjust for the measurable changes in path length that are caused by changing voltages and temperatures occurring between enrollment and regeneration, a technique was developed that establishes a mean path length for a small subset of paths that are deemed to be stable during an enrollment. We then compute the total path lengths of the same set of paths during regeneration, and adjust each of the subsequent path measurements to accommodate this shift in mean path delay.

To implement this feature, we maintain two registers during enrollment (a path counter and a path delay accumulator). For each of the first 64 valid, stable paths, that path length is added to the accumulator and the counter is incremented. After 64 paths have been added to the counter, the counter is divided by 64, using a rightward shift of 6 bits (2^6=64). This value is transferred into a register to use as a "mean enrollment reference delay", and the enrollment process continues normally.

During regeneration, the same technique is performed: the first 64 path delays are accumulated and the result is divided by 64 to determine the "mean regeneration delay". At this point, the difference between the mean delay values at enrollment and regeneration is computed, and the Data Collection Engine *restarts the regeneration*, applying the difference to the regenerated path delay values. Following this adjustment, each of the path delays measured during regeneration are shifted back into alignment with the path delays measured during enrollment.

In implementing this technique, two approximations were made and merit discussion here. The first approximation is that all paths are equally affected by the changing environmental conditions. In fact, this is not the case; consistent with the drain current equations for MOSFET devices, the path length change, $\boldsymbol{\Delta t}$, is a function of the overall path delay, so that $\boldsymbol{\Delta t}$ tends to be greater for long paths and less for short paths. However, because the 64 paths that are sampled are drawn without bias from the chip's path distribution, this approximation has some merit and represents a reasonable trade-off between accuracy and hardware efficiency and simplicity.

The second approximation that is made concerns the stability of the environmental conditions during the Data Collection Engine operation. This algorithm is implemented only at the beginning of the data collection process. As a result, additional changes in the operating conditions the occur during the data collection process will degrade the effectiveness of the TVCOMP process. Experiments performed with different lengths of temperature soak times (the time after a new temperature point was reached and held before a new operation was started) clearly showed that thermal changes propagating through the device during the data collection process, whether during enrollment or regeneration, adversely impacted the effectiveness of the technique.

8.2. Path Distribution (PDIST) Characterization

The bit generation techniques presented in the previous chapter rely on some degree of freedom from structural bias in the underlying data (the PNs stored in PN memory). One potential source of bias is an incorrect assumption about the likelihood of some numbers occurring more frequently than others. This condition can occur for several reasons, but the primary cause is a difference in the distribution of paths lengths being measured from one chip to the next.

Path length distribution (hereafter referred to as "path distribution") in the MUT for a given chip was found to follow a generally Gaussian distribution around a mean path length of 11 ns to 12 ns. Furthermore, the distributions of the mean path lengths of each of the chips for each of two speed grades also was normally distributed around a population mean for that speed grade. Slower chips (identified on the package with a speed grade of -6) tend to have a mean path length of **10.70 ns** while faster chips (speed grade -7) have a mean path length of **9.93 ns**. The graphic in Fig.17 provides a succinct depiction of the path length distributions in our set of boards through the use of box plots which capture the mean, minimum, maximum, and 25th and 75th percentiles of the distribution. As shown, the boards are divided into two sets ("Slow"/"Fast" indicates a speed grade of (-6/-7), respectively).

The PDIST algorithm is a process that, when requested via a user parameter, is invoked at the start of an enrollment operation that uses either UNM or UNMD as the bit generation algorithm. It runs immediately following the TVCOMP algorithm, and collects the path lengths of 1,024 random paths in the distribution. A finite state machine then sorts the results using a histogram-based sorting algorithm and identifies the PUF

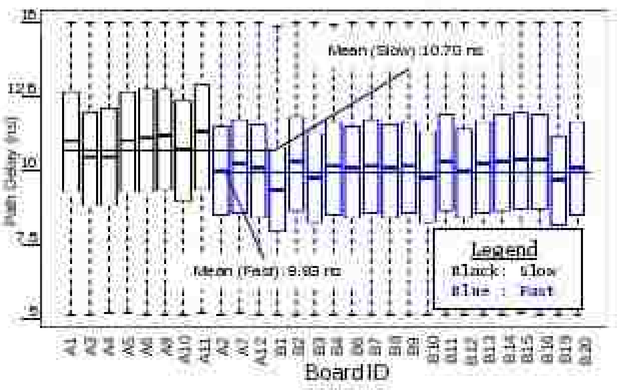

Fig. 17: Distributions of Path Lengths across FPGA Boards

numbers associated with the 5th and the 95th percentiles of the distribution. This mechanism allows a finite state machine within the data collection engine, called the LC_Test, to dynamically adjust the parameters of the data collection engine so that the same number of paths exist at the low and high ends of the range, resulting in a more balanced distribution of PUF numbers from each end of the path length spectrum. This dynamic balancing is required to prevent bias in the data from adversely impacting the randomness performance of the HELP PUF.

Chapter 9: Experimental Results and Analysis

To test the effectiveness of the HELP PUF, environmental experiments were carried out on 30 Virtex-II Pro boards using a thermoelectric cooler (TEC) apparatus and a programmable power supply. As indicated earlier, each board was tested at 9 TV corners, defined by all combinations of three temperatures (0°C, 25°C and 70°C) and three voltages (1.35V, 1.50V and 1.65V). Data collected at 25°C and 1.50V is treated as enrollment data while the data collected at the remaining 8 TV corners is treated as regeneration data.

9.1. Hamming Distance (HD)

Inter-chip Hamming Distance (HD) measures uniqueness of the bitstrings across boards, and is computed by counting the number of bits that are different in the bitstrings from each pairing of boards. An average inter-chip HD is computed using the results from all possible pairings, which in our experiments is 30*29/2 = 435. The inter-chip HDs are typically converted into percentages by dividing each of them by the length of the bitstrings. The best achievable average HD under these conditions is 50%. *Intra-chip* HD, on the other hand, is the number of bits that differ in two bitstrings obtained from the *same* chip but tested under different environmental conditions. The ideal intra-chip HD is zero, and a non-zero value indicates that one or more bit flips occurred during regeneration. In our experiments, intra-chip HDs are computed across the 9 TV corners for each board and then an average is computed using the 9*8/2 = 36 individual HDs.

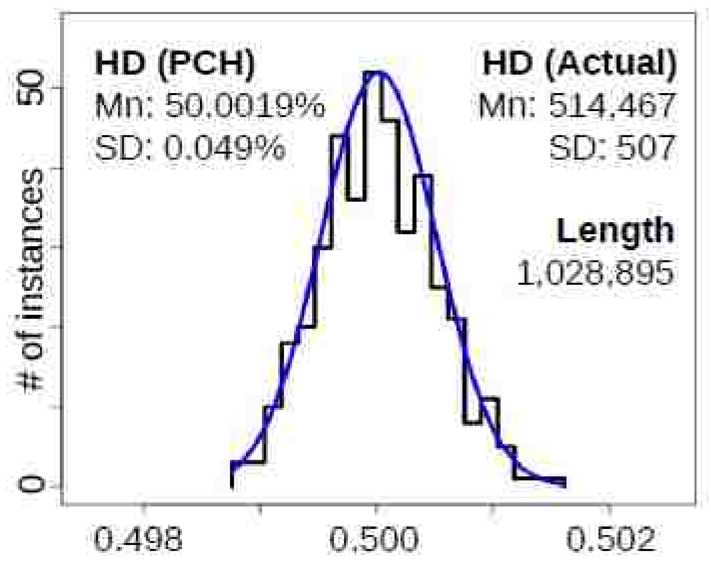

Fig. 18: Hamming Distance (HD) for UNM Method

The 'average-of-the-averages' is then computed using the average HDs from all boards. Fig. 19 shows histograms for the inter-chip HDs and other statistical results obtained for the DPNC and UNMD techniques.

9.1.1. UNM

Fig. 18 shows the HD distribution as well as the mean (Mn) and standard deviation

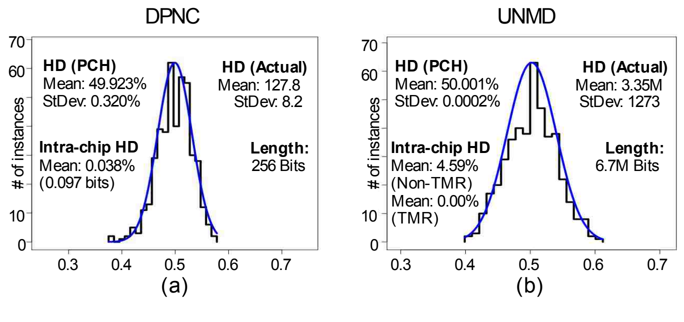

Fig. 19: Hamming Distance (HD)

(SD) of the Gaussian curve fitted to, and superimposed on, the distribution. The inter-chip HD is 50.0019% for bitstrings with a length of 1,028,890. This is very close to the ideal value of 50%. The standard deviation is also very small. Combined with a mean intra-chip HD of **2.74e-7**, these results indicate the bitstrings are highly reliable and unique.

9.1.2. DPNC

The length of the bitstrings using the DPNC technique is 256 bits. The average inter-chip HD in Fig. 19(a) is **49.923%**. A Gaussian curve is shown fitted on top of the inter-chip HD distribution as a means of illustrating expected behavior. The standard deviation of the normal curve is 8.192 (where smaller is better). This value is consistent with the expected standard deviation of a normally distributed set of random values.

The average intra-chip HD is 0.038%. The non-zero value indicates that bit-flips occurred with a frequency of 0.097 bit-flips per 256-bit string.

9.1.3. UNMD

The length of the bitstrings for the UNMD technique is 6,698,512. Fig. 19(b) plots the inter-chip HD distribution. The average inter-chip HD is **50.001%**. The intra-chip HD using the bitstrings prior to applying is **4.59%**, which became **0%** after applying TMR.

9.2. NIST Statistical Analysis of Randomness

To test the randomness of the bitstrings produced by the HELP PUF, we used a statistical test suite provided by the National Institute of Science and Technology, or NIST [14]. These tests were applied to the bitstrings from the 30 boards.

9.2.1. UNM

The NIST statistical test suite is also applied to the bitstrings from the 29 boards. The bitstrings pass all NIST statistical tests, with no more than 2 boards failing any of the

15 tests. In addition, these bitstrings pass all of the P-value-of-the-P-values tests, even in spite of the fact that the NIST documentation indicates that a minimum of 55 boards is required before this metric can be considered valid.

9.2.2. DPNC

All of the bitstrings generated by this method passed each of the tests in the subset of NIST tests that are applicable to 256-bit strings.

UNMD. The bit sequences generated by the UNMD method were sufficiently long that all 15 NIST tests are applicable. All 15 tests passing, with no fewer than 28 boards passing any one test (the number required by NIST for a test to be considered 'passed').

9.3. Analysis of Running Time

9.3.1. UNM

On average, the number of valid paths tested per second is **30.20** for enrollment and **88.03** for regeneration. This includes the time required to test and discard invalid paths, and the time required to generate the *n(n-1)/2* bitstrings from the *n* PNs stored in block RAM.

9.3.2. DPNC

Bitstring generation times for HELP are reported here as the average number of bits generated per minute, excluding serial data transfer time. During enrollment, the time required to generate each bit depends on several factors, including the percentage of tested paths that are stable, the value of ***k*** (the number of consecutive copies of a value required to produce a bit), and the number of PNs that are read from memory before encountering ***k*** consecutive copies.

With ***k*=5**, the average number of paths tested for each generated bit during enrollment is **1,261**, due to the highly selective nature of the DPN binning algorithm described previously. Bits are generated at an average rate of **36.4** bits per minute. During regeneration, since only valid paths are measured, the average bit generation rate increases to **167** bits per minute.

9.3.3. UNMD

On average, the data collection engine tested 3.92 paths for each of the 4,096 valid PNs that we collected across 30 boards. On average, 22.35 paths were tested, at up to 12 samples per path, for stability every second. For the UNMD analysis, the PNs were collected by the HELP PUF engine, while the bit generation process was completed off-chip using a software program. This was done to allow us to evaluate a range of noise thresholds without needing to re-collect the PNs each time. As a result, the FPGA running time of the bit generation process for UNMD is not known.

9.4. Probability of Failure: Results and Analysis

9.4.1. UNM

As discussed above, a bit flip occurs when a PN measured during regeneration jumps across the “bit flip line” as shown by the example in Fig. 4(b). The number of bit flips that occurred across the 8 regenerations for the 29 boards is **10**. This yields a probability of failure of **8e-5**, computed as 10 / (29 boards x 4096 PNs per board). Although beyond the scope of this work, we have developed a simple, very low-overhead technique that eliminates all bit flips in our results and improves the probability of failure to **7.25e-11**.

9.4.2. DPNC

There were a total of 9 unique errors that resulted in 19 bit flips during the 240 regenerations that were performed during our experimentation. The overall single-bit probability of failure (POF) is **3.09x10^{-4}** (19 errors per (30 boards * 8 regenerations per board * 256 bits per regeneration)). 16 of these 19 bit flips occur when the core logic voltage of the FPGA is 10% lower than nominal.

9.4.3. UNMD

The probability-of-failure analysis for the UNMD method is performed as two analyses: the POF for the initial bitstring and the POF for the TMR-based bitstring described in Chapter 7. Both of these analyses involve generating bitstrings at all 9 TV corners across a range of noise thresholds. In each case, we record the number of bit flips that occur at each noise threshold, and then fit an exponential curve to this data. The exponential fit allows us to model expected error rates for noise thresholds far higher than those at which bit flips actually occur in our empirical results.

For the initial bitstrings, we computed a theoretical error rate of 1.54×10^{-6}, or 1 bit flip in approx. 650,000 bits generated. Fig. 20(a) illustrates the actual and theoretical error rates for each of the TMR-based bitstrings. Fig. 20(b) shows an enlarged view of the theoretical error rate at a noise margin of 0.90. At this noise threshold, our POF is 1.096×10^{-11}, or 1 bit flip in approx. 91 billion bits generated.

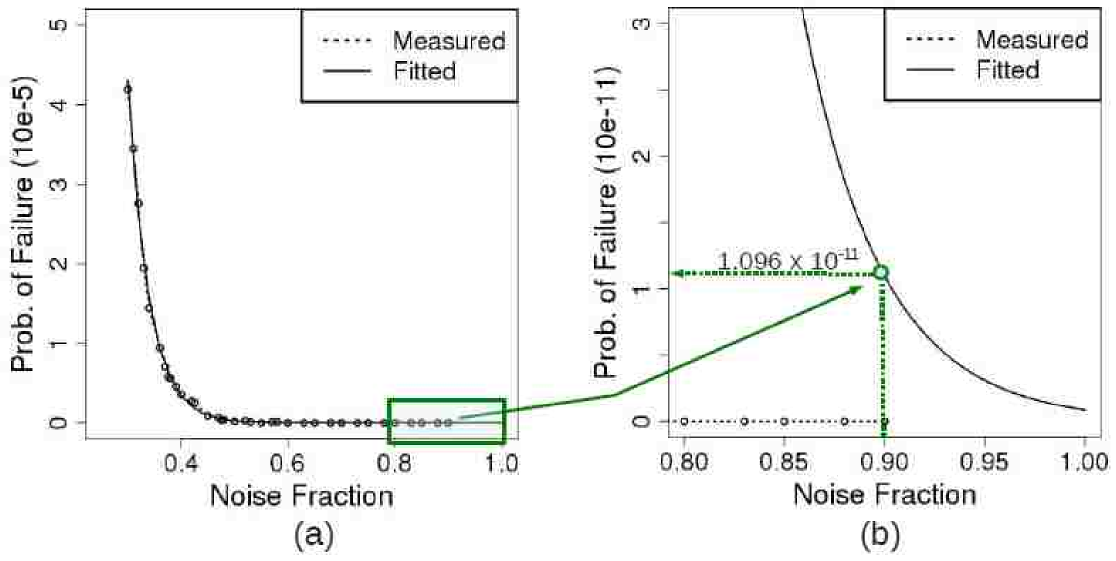

Fig. 20: Probability of Failure (UNMD)

9.5. UNMD Security Vulnerability and Mitigation.

The HELP PUF, when using the UNMD method, is capable of generating reliable, cryptographic-strength bitstrings of up to several million bits in length. However, an adversary with access to the simulation model of the target system may be able to "reverse engineer" the secret bitstring. While this vulnerability would be difficult to exploit, the only way to completely eliminate the threat is to obfuscate the Valid Path Memory component of the public data.

Since the DPNC method is not subject to this vulnerability, we propose to use DPNC to generate a small (32- to 64-bit) bitstring that can be used to obscure the public data produced by the UNMD technique using the same set of PNs during the enrollment process. The public data for this short bit string is added to the obfuscated UNMD public data. At the start of regeneration, the unobscured public data for the DPNC method is used to regenerate the short bitstring, which is then used to unveil the public data for the

UNMD regeneration process.

Chapter 10: Future Work

The FPGA implementation of the HELP PUF has been highly successful in demonstrating the concept of the PUF and studying its performance. The PUF has been shown to perform well against the three fundamental criteria for PUF performance that were set out in the introduction: *uniqueness (inter-chip HD), robustness (intra-chip HD),* and *randomness (NIST)*. Additionally, through the use of the TMR-styled bitstring replication method explained in Sect. 7.7, we have been able to reduce the probability of an error in a 256-bit bitstring to approximately 3 parts per billion.

In this chapter, I present a discussion of additional work that will be of value to increasing the attractiveness of, and/or enhancing our understanding of, the HELP PUF.

10.1. ASIC Implementation

Perhaps the most obvious extension of this work is to port an improved version of the HELP PUF's design into an application-specific integrated circuit (ASIC) platform. At the time of this writing, this work is already underway, and a design is being prepared for inclusion on a test chip that will be fabricated in a 130-nm bulk technology. This implementation is expected to be completed in the Fall of 2013.

Challenges posed by an ASIC implementation will include the necessity of designing the required clocking resources, replacing the proven digital clock managers (DCMs) that are available on the Xilinx Virtex2-Pro FPGA. In particular, the ability to control the phase relationship between a DCM output and a reference clock must be preserved. The critical role that this adjustment plays in both the operation of the PUF and its performance means that this design goal will require careful analog-level simulation of phase controllability using a post-layout netlist.

10.2. Aging: A Study of the PUF Performance Over Time

The reliability of the HELP PUF is directly related to the relationships between path delays of different paths in the MUT. Therefore, anything that causes those path delays to change presents a risk to the PUF's ability to correctly regenerate a bitstring. And, while we have thus far treated the variations upon which the PUF relies for its operation as being static and unchanging after the point of manufacture, this assumption may not be valid when the effects of aging are considered.

Aging effects for path delay include anything that affects the rate of charging and discharging the parasitic on-path and off-path capacitances. For transistors, changes to threshold voltages, effective resistance, and charge carrier mobility caused by hot carrier injection (HCI) and negative bias temperature instability (NBTI) are a primary concern, while electromigration in the metal lines of the interconnections can also impact the resistances through which these currents are supplied.

The specific concerns surrounding aging effects on the HELP PUF differ slightly depending upon the bit generation method that is invoked. For UNM/UNMD, the bitstring that is generated depends on the relative differences in delay between paths, so that the sensitivity of these methods to common-mode changes (changes that affect all paths equally) will have a less detrimental effect than localized changes in individual path delays. DPNC, however, considers each of the PN values individually, and is therefore more likely to suffer from *any* changes to path delay that cause mod-PNs to change from one group to another.

At present, we do not have any specific data that directly correlates the same path delay measurements separated only by significant amounts of time. The evolving nature of the HELP PUF's structure and operation means that the data that was collected in the early experiments is not directly comparable to the data collected more recently. This is due to the lack of control over the actual placement and routing of each of the different versions of the HELP PUF engine bitstream. As new features were added and changes made to the implementation, each generated bitstream resulted in the use of different primitives and routing resources on the FPGA, and therefore path delays for the same path (comprised of an initial and final launch vector and a target insertion point) no longer bore any relevance.

An accelerated aging test regimen on all or part of the population of test boards is not a desirable option. The XUP boards that we have used for this experimentation represent a sizable investment and are a valuable resource for other experimentation. As a result, any testing that risks permanent damage to the boards is not suitable under the circumstances. However, specific repetition of tests using the same bitstreams that were used in the early phase of HELP PUF experimentation could be performed solely for the purpose of determining the impact of aging on the HELP PUF's performance.

10.3. The Use of Voltage Scaling to Improve Reliability

Our environmental testing revealed a strong correlation between the worst reliability performance and low supply voltage. Sixteen out of nineteen bit flips that occurred during testing of the DPNC method occurred when the supply voltage was set to 1.35V (10% below the normal V_{CC}).

One idea that offers a measure of protection against this error-producing condition is to perform an additional filtering step during enrollment to eliminate paths that are susceptible to jumping at lower-than-normal supply voltages. The use of voltage scaling circuits to reduce the supply voltage on an ASIC could allow paths to be measured at both nominal and low voltages, and if significant changes occur between the path lengths under these conditions, the path could be marked as invalid in PN memory and the possible error averted.

Voltage scaling circuits were originally introduced to match power consumption and applications' performance requirements in computing hardware designed for mobile applications. Static voltage scaling technologies such as Intel's SpeedStep are used to match power and performance requirements on a per-application level, while true dynamic voltage scaling techniques (such as Marvell's Xscale processor technology) involves “on-the-fly” changes to operating voltage and processor frequency based upon continuously changing operating environments[30]. It is anticipated that either approach would be appropriate for this application.

Much of the research work required to explore this option could be done using existing hardware, and a sense of the effectiveness of voltage scaling on the reliability of HELP could possibly be obtained from existing data from our TV experiments. The same is true for the ASIC version; it is not strictly necessary to incorporate voltage scaling hardware into the on-chip design, since the entire chip's supply voltage may be adjusted externally. Regarding the inclusion of voltage scaling circuitry on the test chip currently being design, several questions emerge. These questions include whether the scaling circuitry could be built simply from the standard cell library currently being developed for the next test chip, how costly the circuit would be in terms of area, power, and design complexity, and whether its inclusion in the design would adversely impact the other primitives on the ASIC.

10.4. Attack Threat Analysis and Mitigation

An important topic in any PUF research is the threat of adversarial attacks designed to steal secret data from a PUF. Side-channel attacks are aimed at extracting information from one of a number of secondary sources, such as global chip current, power consumption, electromagnetic emissions, or optically observing the circuit's behavior through the use of such techniques as backside thinning. More invasive attacks, such as probing attacks, attempt to learn the key through more direct physical access.

Another type of side-channel attack is the *fault injection* attack. This is considered an *active* attack because it involves the creation of one or more fault conditions, which might include abnormal power supply levels, out-of-spec clock timing, or invalid timing conditions (i.e., setup-hold violations). By recording the resulting output of the PUF, it is possible to gain critical information about the internal state of the device.[31]

Another form of attack is the *modeling attack,* wherein an attacker repeatedly applies challenges and records the corresponding responses in order to build a linear model of the variations in the system, and in so doing can learn the secret data or significantly reduce the size of the problem to the point at which a brute force attack is feasible.

Of these, perhaps the greatest threat of attacks on the HELP PUF are power analysis attacks. In particular, it would be prudent to study the effectiveness of differential power analysis (DPA) attacks, in which repeated power traces are captured on an oscilloscope and analyzed using statistical methods to identify power signatures associated with certain states, values, or conditions within the chip. No specific attempts have been made thus far to assess the vulnerability of HELP to DPA attacks. Additionally, no hardware features have been added to HELP to obfuscate the power behavior of critical operations that may leak the internal states of the engine. Both of these represent opportunities for improving the viability of the HELP PUF.

The HELP PUF may prove to be more resilient to model-building attacks, since the specific MUT does not present an adversary with a simple, repeating set of gates to model, in the way that Arbiter PUFs and RO-PUFs do. However, the work that we have been doing has heretofore been confined to ensuring that the PUF performs well in terms of the specific PUF metrics, and a complete attack threat analysis has been outside of the scope of that work.

Chapter 11: Conclusion

In this dissertation, details have been presented regarding the design and experimental testing of HELP, a practical, realizable PUF, and several bit generation techniques have been proposed and their strengths and weaknesses analyzed. In particular, two of these bit generation techniques, called DPNC and UNMD, have been put forward as showing the most robustness, reliability, and utility. The HELP PUF is based on variations in path delays and on the stability of those paths, each measured from a core logic macro embedded within the chip. The results of the HD, NIST, and POF analyses show the bitstrings to be genuinely random, unique, and highly reproducible under changing environmental conditions, all of which are critical requirements for the potential use of HELP in applications such as mobile computing or smartcards.

Finally, a list of additional research work was included that, if completed, will strengthen the viability of the HELP PUF as a cryptographic-strength hardware security primitive.

References

[1] "Federal Information Processing Standard 197 (FIPS-197), Advanced Encryption Standard." [Online]. Available: csrc.nist.gov/publications/fips/fips197/fips-197.pdf.

[2] "National Institute of Science & Technology." [Online]. Available: http://csrc.nist.gov/.

[3] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical one-way functions.," *Science (New York, N.Y.)*, vol. 297, no. 5589, pp. 2026–30, Oct. 2002.

[4] B. Gassend, D. Clarke, and M. van Dijk, "Controlled Physical Random Functions," in *Conf. on Computer Security Applications*, 2002, pp. 149–160.

[5] K. Lofstrom, W. R. Daasch, and D. Taylor, "IC Identification Circuit using Device Mismatch," in *IEEE International Solid-State Circuits Conference*, 2000, vol. 46, no. 8, pp. 372–373.

[6] P. Simons, E. van der Sluis, and V. van der Leest, "Buskeeper PUFs, a promising alternative to D Flip-Flop PUFs," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, 2012, pp. 7–12.

[7] G. E. Suh and S. Devadas, "Physical Unclonable Functions for Device Authentication and Secret Key Generation," in *2007 44th ACM/IEEE Design Automation*

Conference, 2007, pp. 9–14.

[8] P. S. Abhranil Maiti, "Improving the Quality of a Physical Unclonable Function Using Configurable Ring Oscillators," in *International Conference on Field Programmable Logic and Applications*, 2009, pp. 703–707.

[9] Y. Su, J. Holleman, and B. P. Otis, "A Digital 1 . 6 pJ / bit Chip Identification Circuit Using Process Variations," *IEEE Journal of Solid State Circuits*, vol. 43, no. 1, pp. 406–407, 2008.

[10] M. Majzoobi, F. Koushanfar, and M. Potkonjak, "Lightweight secure PUFs," in *2008 IEEE/ACM International Conference on Computer-Aided Design*, 2008, vol. 0, no. 1, pp. 670–673.

[11] J. Ju, J. Plusquellic, R. Chakraborty, and R. Rad, "Bit string analysis of Physical Unclonable Functions based on resistance variations in metals and transistors," in *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, 2012, pp. 13–20.

[12] D. Suzuki and K. Shimizu, "The Glitch PUF : A New Delay-PUF," in *Conf. on Hardware Embedded Security (CHES)*, 2010, pp. 366–382.

[13] J. Lie and J. Lach, "At-speed delay characterization for IC authentication and Trojan Horse detection," in *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*, 2008, pp. 8–14.

[14] "Xilinx Corporation." [Online]. Available: http://www.xilinx.com.

[15] M. Tehranipoor and C. Wang, *Introduction to Hardware Security and Trust*. Springer, 2012, p. 427.

[16] M. Bhargava, C. Cakir, and K. Mai, "Comparison of Bi-stable and Delay-based Physical Unclonable Functions from Measurements in 65nm bulk CMOS," in *Custom Integrated Circuits Conference (CICC)*, 2012, pp. 1–4.

[17] S. S. Kumar, J. Guajardo, R. Maes, G. Schrijen, and P. Tuyls, "The Butterfly PUF Protecting IP on every FPGA (Extended Abstract)," in *Hardware-Oriented Security & Trust (HOST*, 2008, no. 71369, pp. 67–70.

[18] G.-J. Schrijen and V. van der Leest, "Comparative analysis of SRAM memories used as PUF primitives," *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1319–1324, Mar. 2012.

[19] Y. Hori, T. Yoshida, T. Katashita, and A. Satoh, "Quantitative and Statistical Performance Evaluation of Arbiter Physical Unclonable Functions on FPGAs," *2010 International Conference on Reconfigurable Computing and FPGAs*, pp. 298–303, Dec. 2010.

[20] C. Yin and G. Qu, "Temperature-aware cooperative ring oscillator PUF," *2009 IEEE International Workshop on Hardware-Oriented Security and Trust*, pp. 36–42, 2009.

[21] A. Maiti, L. McDougall, and P. Schaumont, “The Impact of Aging on an FPGA-Based Physical Unclonable Function,” in *International Conference on Field Programmable Logic and Applications*, 2011, pp. 151–156.

[22] C. Lamech, J. Aarestad, J. Plusquellic, R. Rad, and K. Agarwal, “REBEL and TDC: Two embedded test structures for on-chip measurements of within-die path delay variations,” *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 170–177, Nov. 2011.

[23] Y. Haihua and A. D. Singh, “Experiments in detecting delay faults using multiple higher frequency clocks and results from neighboring die,” *International Test Conference, 2003. Proceedings. ITC 2003.*, vol. 1, pp. 105–111, 2003.

[24] J. Aarestad, C. Lamech, J. Plusquellic, D. Acharyya, and K. Agarwal, “Characterizing within-die and die-to-die delay variations introduced by process variations and SOI history effect,” *Proceedings of the 48th Design Automation Conference on - DAC ’11*, p. 534, 2011.

[25] “Digilent, Inc.” [Online]. Available: http://www.digilentinc.com.

[26] “OpenCores.com.” [Online]. Available: http://www.opencores.org.

[27] J. Aarestad, P. Ortiz, D. Acharyya, and J. Plusquellic, “HELP: A Hardware-Embedded Delay PUF,” *IEEE Design & Test*, pp. 1–8, 2013.

[28] J. Aarestad, J. Plusquellic, and D. Acharyya, “Error-Tolerant Bit Generation

Techniques For Use With A Hardware-Embedded Path Delay PUF," in *Hardware-Oriented Security & Trust (HOST)*, 2013, pp. 1–8.

[29] J. Ju, R. Chakraborty, C. Lamech, and J. Plusquellic, "Stability Analysis of a Physical Unclonable Function based on Metal Resistance Variations," in *Hardware-Oriented Security & Trust (HOST)*, 2013, pp. 1–8.

[30] L. T. Clark, F. Ricci, and W. E. Brown, "Dynamic Voltage Scaling with the XScale Embedded Microprocessor," in *Adaptive Techniques for Dynamic Processor Optimization*, Springer`, 2008, pp. 123–143.

[31] B. A. Barenghi, L. Breveglieri, I. Koren, F. Ieee, and D. Naccache, "Fault Injection Attacks on Cryptographic Devices : Theory ," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.