An Error-Tolerant Bit Generation Technique For Use With A Hardware-Entangled Path Delay PUF

<authors' names removed per submission guidelines>

Abstract: *Cryptographic and authentication applications in ASICs and FPGAs, as well as codes for the activation of on-chip features, require the use of embedded secret information. The emergence and unconstrained growth of the mobile computing and communications spaces is placing an increasing demand on existing methods of generating and safeguarding this secret data. The generation of secret bitstrings using physical unclonable functions, or PUFs, holds the promise of replacing older, conventional, e.g., EPROM-based methods, and offers several distrinct advantages, including the elimination of the need to store the bitstring in costly, specialized non-volatile memory, and a measurable increase in the number of random bits that can be generated. This paper presents details of an on-chip PUF engine called the Hardware-Entangled Delay PUF, or HELP, and introduces a new bit generation technique using this PUF. HELP leverages the natural variations that occur in the path delays of a core macro on a chip to create a unique, stable, and random bitstring. We evaluate several statistical quality metrics of the bitstrings generated with this method on a set of 30 FPGA boards across a temperature range of 0 to 70°C and an operating voltage range of ±10% of nominal, and propose an error-avoiding scheme that offers measurably improved protection against errors in the resulting bitstring.*

1. Introduction

Physical unclonable functions (PUFs) are becoming increasingly commonly-used mechanisms for generating random numbers for a wide range of security-related applications. PUFs are designed to be able to reliably differentiate one chip from another by leveraging the random variations in physical properties of these chips, and are intended to be difficult or impossible to duplicate or clone, even for the manufacturer. Process variations are effectively impossible to control or eliminate; however, they can be measured. PUFs can differ in the specific properties that they seek to exploit. However, physical properties commonly targeted include propagation delay, metal resistance, transistor drive strength, and mismatches between complementary transistors. A commonality among most PUFs is that they generate bitstrings by comparing measured quantities, in which variations occur between chips, and produce bitstrings based on the results of those comparisons.

The quality of the bitstrings produced by a PUF is an important measure of its usability. Generally, however, three criteria are considered to be essential for a PUF to be used for such applications as encryption: 1) the bitstrings produced for each chip must be sufficiently *unique* to distinguish each chip from every other, 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 paper, we present a detailed examination of a PUF, called HELP, that is based upon path delay variations. The novel features that differentiate HELP from other delay-based PUFs include: 1) the capability of comparing paths of different lengths, 2) eliminating the need for specially designed, layout-dependent delay elements that impose a high area cost while providing a relatively small amount of entropy, 3) a minimally invasive design with low area and performance impact, and 4) a hardware-entangled PUF engine requiring no external testing resources. HELP enjoys the added advantage of the large set of paths typically found in logic macros such as the Advanced Encryption Standard (AES). This large source of entropy allows HELP to generate reasonably long bitstrings, while being extremely conservative in the paths selected for bit generation. The large availability of paths also enables unique opportunities for achieving bit stability and avoiding errors.

Unique Contributions of this Paper: The following are the unique and novel contributions that are proposed and detailed in this paper:

- A novel modulus-based technique that permits the direct comparison of delay measurements from logic paths of widely varying lengths
- A path delay measurement binning scheme that improves tolerance of noise, uncertainty, and small environmental changes
- A fault-tolerant bit generation technique that offers robustness and resilience to errors caused by environmental conditions and measurement uncertainty

To prove the bit generation technique put forward in this paper, and to demonstrate its usefulness and effectiveness, we make use of a complete, functional FPGA-based implementation of the underlying PUF on a set of 30 V2Pro FPGA boards. We present the results of that experimental work, and evaluate the statistical and performance characteristics of the resulting bitstrings.

2. Background

The PUF first appeared as a mechanism for generating secure bitstrings in [1] and [2]. The PUF as a chip identifier, however, was introduced earlier in [3]. Proposed PUF designs generally fall into one of the following classifications: SRAM PUFS [4], ring oscillators [5,6], MOS drive-current PUFs [7], delay line and arbiter PUFs [8], and PUFs based upon variations in a chip's metal wires [9]. Delay-based PUFs also include such designs as the Glitch PUF, which leverages variation in glitch behavior and is presented in [10]. 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 paper 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 when that AES implementation is run 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 learn or compromise the key.

- The bit flip avoidance scheme proposed in this paper is intended to render the probability of failure in regenerating the bitstring negligibly small.
- The physical implementation of HELP uses the standard hardware resources commonly available in the fabric of an FPGA or in a standard cell library, and an on-chip clock generation scheme, i.e., a digital clock manager (DCM). The use of the DCM for performing path timing tests is similar to that proposed in [11] for Trojan detection and IC authentication.
- By using the core logic of AES itself, a large source of entropy is leveraged.

In this paper, we analyze the production of bitstrings that are 256 bits in length. The HELP PUF is not limited to bitstrings of this length, however, and is capable of generating bitstrings of virtually any length, depending upon available resources in a given implementation. The specific bitstring generation technique proposed in this paper does not, to our knowledge, suffer from any known or apparent security weaknesses, or vulnerabilities to direct or model-building attacks.

3. HELP PUF Overview

Similar to other PUFs, the HELP PUF functions by applying a set of challenges and measuring the corresponding responses, called challenge-response pairs. The challenge component for HELP consists of a randomly selected, two-vector test sequence applied to the inputs of the macro-under-test (MUT), which introduces a set of transitions that propagate through the core logic of the MUT and appear on its outputs. The responses are the measured path delays on each of the outputs, and are expressed as 8-bit numbers that correspond to path delay. A single MUT output is isolated and measured individually, as explained in this section.

The delay measurement precision has an impact on the stability of HELP. We use an embedded test structure called REBEL to obtain a high-precision, digitized representation of the path delays [12]. REBEL is integrated directly with the scan chain logic and uses the on-chip clock tree network for launch-capture (LC) timing events.

Fig. 1 depicts a overview of the REBEL test structure, consisting of two rows of flip-flops (FFs) connected together into a scan chain. Small logic blocks on the left of each row, labeled RCL for Row Control Logic, allow the scan elements on each row to be configured as follows:

- The top row is the launch row, and is configured to operate in functional mode.
- The second row is the capture row, and is configured in a *mixed* mode, in which a specific FF, called the **insertion point** (IP), is chosen. This scan-FF and each scan-FF to the right of it in the row are placed in a mode called **flush delay** (described below), and form a combinational delay chain, effectively extending the path at the IP.

Flush-delay mode (FD) is a special mode in which a scan chain can be configured as a combinational delay chain. This is depicted in the callout in Fig. 1, which shows two master/slave FFs in which the output of the first master feeds into the scan input of the second FF. Any transition that occurs on the IP propagates through the *functional input* and into the first master using logic that selects that path (not shown). In contrast, the logic controlling the scan mux for the second FF (and all FFs to its right) selects the *scan input*, effectively allowing the transition to propagate unimpeded through the masters of these

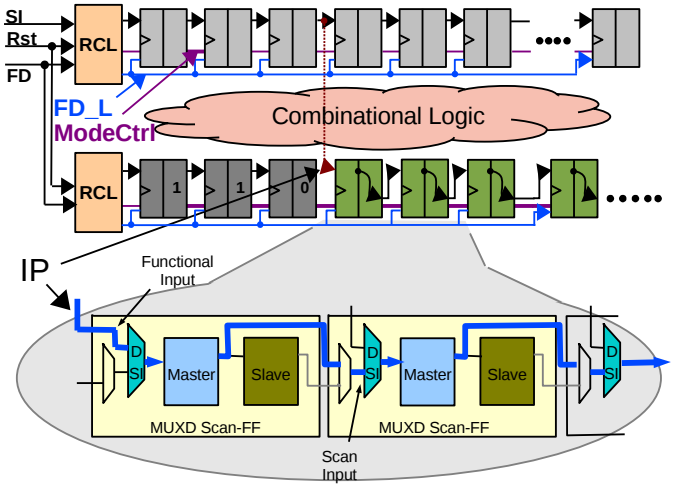

Fig. 1: REBEL embedded test structure.

FFs. Details concerning the control logic for the scan chain muxes can be found in [12].

A REBEL path delay test is carried out by scanning in configuration information, which selects the IP and configures the delay chain as shown in Fig. 1. A clock transition is then applied to the launch row FFs which generates transitions that propagate into the MUT. Any transition that occurs on the MUT output at the IP will propagate into the delay chain. By asserting the clock input on the capture row FFs, the master latches revert to storage mode and digitize the time behavior of the transition(s) as a sequence of 1's and 0's. The combined delay of the MUT path and the delay chain can be derived by searching, from right to left, in the binary sequence for the FF that contains the first transition.

4. Experimental Setup

We've created a complete HELP implementation on an FPGA and carried out experiments on a set of 30 V2Pro FPGA boards. Fig. 2 shows a top-level structural diagram of our HELP implementation.

The MUT used in our implementation is the logic defining a single round of a pipelined AES implementation (space limitations prevented inclusion of all 10 rounds of the logic) from OpenCores [13]. 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 block labeled “REBEL (Capture) Row” in Fig. 2 also represents the pipeline FFs between the logic blocks defining the rounds in AES. We modified this row to incorporate REBEL, and designed it to implement the “mixed mode” functionality described previously in relation to Fig. 1. The number of FFs in this row is expanded from 256 to 264 to extend the delay chain for the IPs in the rightmost side of the MUT.

The remaining components in Fig. 2 define the HELP PUF engine, and can be divided into the **Data Collection Engine** (DCE), and the **BitGen Engine** (BGE). One iteration of the whole process produces the bitstring. The engine behaves differently depending on whether a new bitstring is requested (a process called **enrollment**) or whether the bitstring needs to be reproduced (a process called **regeneration**). We distinguish between these scenarios in the following description where

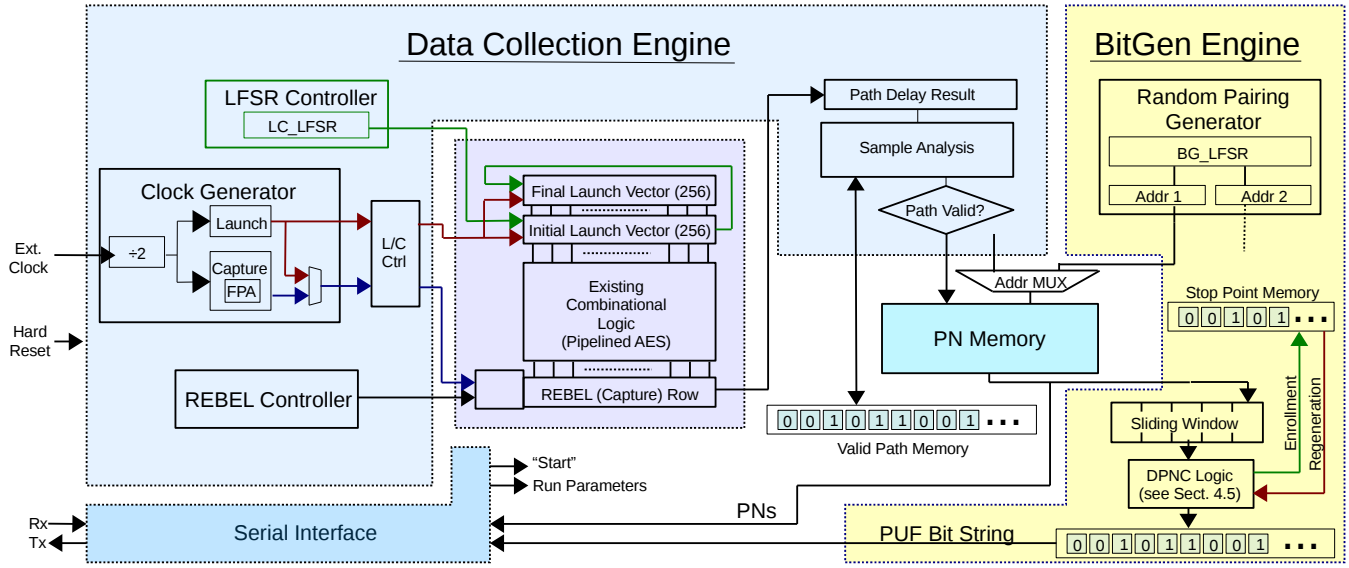

Fig. 2: Top-Level HELP System Diagram

needed.

The overhead of HELP is given by the following table. The resources under the column "AES Macro" corresponds to a single round of AES. A full pipelined implementation of AES would therefore be 10X larger. Factoring this in reduces the overhead of HELP from 200% as shown in the last column to approx. 20%.

TABLE I. FPGA RESOURCE UTILIZATION

	AES Macro	Full PUF	Pct. Used
Flip-flops	1297	1904	60%
LUTs	3122	7174	26%
Slices	2146	4046	29%
RAMB16	0	58	42%
BUFGMUX	1	5	31%
DCMs	0	3	37%

4.1 HELP Components

The DCE in Fig. 2 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.

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. 2. 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 of resolution in the phase of the Capture clock on the V2Pro FPGA 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 set to 0. Just 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 1 summarizes the characteristics of the Capture clock, and Fig. 3 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 PNs) are used as a digital representation of the path delays.

TABLE II. CAPTURE CLOCK PHASE ADJUSTMENT

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

The remaining components of the DCE are as follows:

PN Memory: A block RAM used to store the PNs.

LC LFSR Controller: A 32-bit linear feedback shift register (LFSR) used to produce the randomized launch vectors.

REBEL Controller: Configures the IP in the REBEL row attached to the output of the AES logic block.

Sample Analysis Engine (SAE): Analyzes the digitized

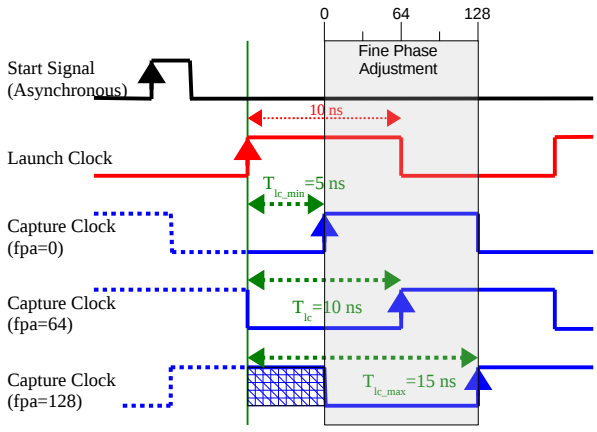

Fig. 3: Launch/Capture Timing Diagram

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**.

Valid Path Memory: A block RAM used to record a pass/fail flag for each tested path that reflects its validity (as defined under SAE). This memory uses public storage because it represents the *helper data* needed in the regeneration process.

Random Pairing Generator: Uses a 28-bit LFSR to generate randomized pairings of PNs for bit generation.

Stop Point Memory: A block RAM used by the Bit Generation Engine to *record* flags, or "stop points", during enrollment that are *read* during regeneration. This memory, like the Valid Path Memory, uses public storage and represents *helper data* needed in the regeneration process.

The Serial Interface component is used to interact with the HELP engine, and to transfer the results of the path testing and bit generation processes.

4.2 Path Delay Measurement

A sequence of paths are tested by the DCE process to produce the PNs used later in bit generation. The starting point and order in which the paths are tested is completely determined by the LC LFSR. The DCE process begins by loading the LC LFSR with a seed (provided by the user), and instructs the LC LFSR controller to load a random pair of vectors into the launch rows. Simultaneously, the REBEL controller configures the REBEL row with a specific IP and places the REBEL row in FD mode. The same random vector pair is reloaded to test each of the 256 IPs, one at a time, before the LC LFSR generates and loads the next random vector pair.

A key contribution of our technique is the discovery that **path stability** can be used as the basis for random number generation. Path stability is defined as those paths which have a rising or falling transition, do not have temporary transitions or glitches, and that produce a small range of PNs (ideally only one) over multiple repeated sampling. As shown below, the paths that pass the stability test are different for each chip in the population. For the bit generation method described in this paper, this unpredictability in path stability results in a random variation in the number of paths that must be measured before another bit is generated. A detailed explanation of this process appears in the next section of this paper.

A state machine within the DCE is responsible for measuring path delays and for determining the stability of the paths. Our algorithm begins testing a path by setting the FPA to 128, which configures the Capture clock phase to 270°. It then iteratively reduces the phase shift in a series of LC tests, called a **sweep**. For paths that have transitions, the process of 'tuning' the FPA toward smaller values over the sweep effectively 'pushes' the transition backwards in the delay chain, since each successive iteration reduces the amount of time available for the transition to propagate. When the edge is 'pushed back' to a point just before a *target FF* in the delay chain, the process stops (the goal has been achieved). The target FF is an element in the delay chain that is a specific distance (in scan-FFs) from the IP. The value of the FPA at the stop point is saved as the PN for this path, i.e., the PN represents the 'response' to the 'challenge' defined by the launch vector and IP.

Evaluating path stability is accomplished by counting the number of transitions that occurred in the REBEL row by 'XOR'ing' neighboring FFs in the delay chain. The path is immediately classified as unstable (and the sweep is halted) if the number of transitions exceeds 1 at any point during the sweep. Once the sweep is complete, the whole process is repeated multiple times. If the range of PNs measured across multiple samples varies by more than a user-specified threshold, the path is classified as unstable and is discarded.

Note that path stability evaluation occurs ONLY during enrollment. In order to make it possible for regeneration to replay the valid path sequence discovered during enrollment, the 'valid path' bitstring is updated after testing each path. For paths considered valid, a '1' is stored and for those classified as unstable, a '0' is stored. During regeneration, the exact same sequence of tests can be carried out by loading the LC LFSR with the same seed and using the 'valid path' bitstring to determine which paths are to be tested (a '1' forces the path to be tested, and a '0' forces the path to be skipped).

4.3 Temperature Compensation and Jumps

Environmental conditions (i.e., temperature and voltage) that are markedly different during regenerations than they were during the enrollment process can cause the path delay measurements to shift. The modulus technique that we discuss in the next section requires the PNs to remain as constant as possible during regeneration at different TV corners, and therefore it would be beneficial to calibrate out these types of shifts, if possible. We developed a technique called Temperature Compensation that is designed to accomplish this goal. The general idea is to derive a constant during regeneration that, when added to all PNs, shifts the distribution back to that obtained during enrollment. This can be accomplished by computing a 'mean PN' from a small subset of tests (we found 64 to be sufficient) during enrollment which is then recorded in public storage. Later, during regeneration, the mean is again computed using the same set of tests and the difference between the enrollment and regeneration means is added to the PNs during the subsequent bit generation phase. In the experimental results that follow, we applied Temperature Compensation to the PNs and found that the worst-case shifts on order of -8 to +14 were typical. We are currently investigating a more sophisticated technique that also 'scales' (multiplies) the PNs as a means of dealing with the expansion and contraction of the distribution that also occurs.

In our experiments, we found that a small portion of the PNs also tend to "jump" to new values well beyond that calibrated for by temperature compensation. The underlying mechanism for the jump behavior, although caused by TV variations, is different

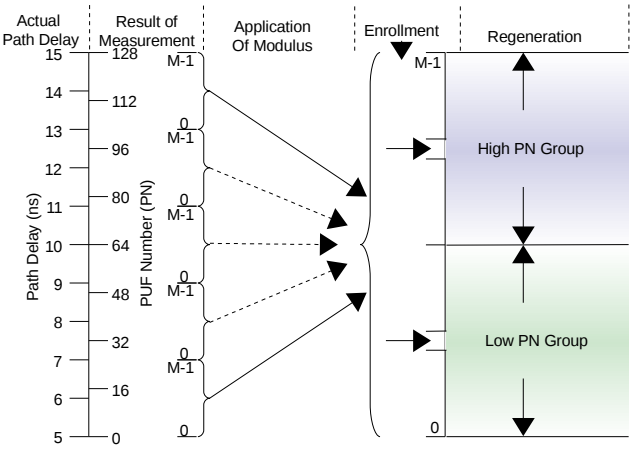

Fig. 4: Dual-PN Path Delay Binning Technique

and deals with the appearance and disappearance of 'hazards' on off-path inputs to gates along the path-under-test. In rare cases, it is possible that an off-path input (which is supposed to remain at its non-dominate value, e.g., a '1' on an input to a AND gate) changes momentarily to a dominate value at a particular TV corner but not at enrollment. Depending on the relative timing of the appearance of the hazard and the signal transition along the tested path, it is further possible that the signal transition along the path-under-test is momentarily delayed by the hazard. When this occurs, a fundamental change occurs in the path timing. Unfortunately, there is no way to predict or compensate for these situations short of running fault simulations and enforcing constraints during test vector generation. This jump behavior is the principle reason for the bit flips that occur in the reported results of the following sections.

4.4 The "Dual-PN" Path Delay Binning Technique

Most PUF primitives are identically designed to avoid bias issues. This is not the case for HELP, because the paths-under-test vary widely in length. We developed a technique called "Dual-PN" which post-processes the PNs to eliminate this bias. The technique first applies a modulus operation to the PNs which trims off the higher order bits which capture the length component and therefore represent the bias. The trimmed PNs, called Mod-PNs, are then partitioned into two groups for bit generation purposes.

The diagram in Fig. 4 provides a graphical depiction of this two-step process. The process begins on the left, as a path with a propagation delay of between 5 ns and 15 ns is measured using REBEL (Section 4.2). This delay is represented by a PN in the range of 0 to 128. The modulus operation reduces this PN 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. 4 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 PNs in the range of ***M/2 to M-1*** comprise the high PN group. As indicated above, temperature shifts are not completely compensated for by TCOMP. This issue is dealt with by implementing a second filtering operation (beyond that described in Section 4.2 in relation to path stability) that further restricts which mod-PNs are considered valid during enrollment. The regions delineated in the center portions of the two groups of Fig. 4 identify the smaller ranges of mod-PNs that are considered valid, i.e., mod-PNs produced during enrollment that fall outside these regions are discarded. During regeneration, small shifts in the mod-PNs of these paths of up to *M/4* values in either direction can occur without introducing a bit flip. Therefore, this scheme adds bit flip resilience to HELP.

4.5 The "Dual-PN Count" (DPNC) Bit Generation Method

The filtering operations described above are sufficient for dealing with shifts introduced by TV variations. However, the larger changes in the mod-PNs introduced by "jumps" (see Section 4.3) require a stronger technique. The rare nature of "jumps" makes it possible to develop a detection and correction method that imposes a low area and time overhead. We propose a technique called "Dual-PN Count" that is based on a popular fault tolerance technique called triple modular redundancy (TMR). Fig. 5 provides an illustration of how it works. During enrollment, the algorithm conducts a forward search through the PUF number memory 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 is to be generated at this point during subsequent regenerations. 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 points indicate where each bit is to be generated, and therefore add to the public storage requirements.

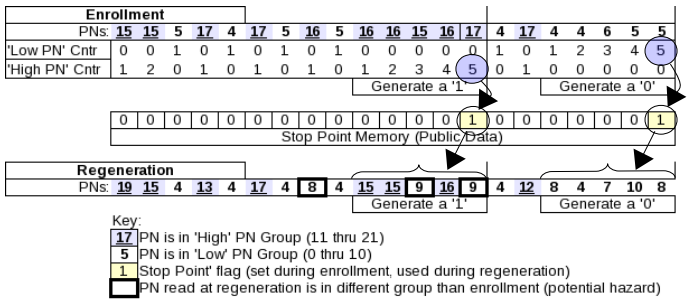

Fig. 5: Dual-PN Count Method - Example

Example: In the example shown in Fig. 5, we use a modulus ***M*=22,** so that the accepted values (during enrollment) for the low PN group are **{4,5,6}**, while the accepted values for the high PN group are **{15,16,17}**. The value of ***k*** is set to 5.

This example first depicts the enrollment process, in which PNs are read from memory, left to right, as shown. Also shown are the states of the counters after each PN is read. When the high PN counter reaches 5 (as shown in the blue 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. Again, a '1' is written to the Stop Point Memory. This process continues until a user-specified number of bits has been generated and added to the bitstring.

The bottom portion of Fig. 5 illustrates the process carried out during regeneration. Here, the '1' bits in the Valid Path Memory determine which paths are to be measured, so that the PN in a given memory location during regeneration always corresponds to the same physical path as it did during enrollment. Similarly, during regeneration, the locations in Stop Point Memory that contain a '1' are used to signal that a bit is to be generated at that point. The value of the generated bit is determined by the group to which the majority of the ***k*** most recent PNs belong. In the regeneration example in Fig. 5, two of the five values that were high PNs during enrollment have changed and now appear as low PNs (note the two '9's in the heavy borders). However, because the majority (3 out of 5) of the values are high PNs, the algorithm correctly generates a '1' bit despite the presence of the two errant measurements. Also note that the first erroneous measurement (the '8' in the heavy border) poses no hazard, since it is not used for bit generation. The five PNs which determine the second bit are free of errors, so that the '0' is also regenerated correctly.

5. Experimental Results And Analysis

We collected data on a set of 30 V2Pro boards using a

thermoelectric cooler (TEC) apparatus and a programmable power supply. Experiments were carried out at three different temperatures (0C, 25C, 70C) and three different voltages (1.35V, 1.50V, 1.65V). Enrollment data was collected at 25°C and 1.5V (nominal internal operating voltage, Vcc_{INT}, for the V2Pro). The experiments carried out at the remaining 8 temperature/voltage (TV) corners represent subsequent regenerations.

5.1 Hamming Distance (HD)

Hamming Distance (HD) is computed by counting the number of bits that are different between any arbitrary pairing of bitstrings from different chips. The ideal result occurs when the number of bits that are different is 50% of the bitstring length. With 30 FPGA boards, the number of computed HDs is 30*29/2 = 435. Fig. 6 plots HD along the x-axis against the number of instances. The average inter-chip HD is given in the figure as 49.923%, a value very close to the ideal. The histogram is also shown fitted with a Guassian curve to illustrate how well it conforms to the expected normal distribution. The standard deviation of the normal curve is **8.192** of **256**, consistent with the expected standard deviation of a normally distributed set of random values.

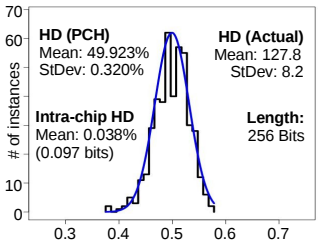

Fig. 6: HD Analysis

The intra-chip HD is the number of bits that differ between two bitstrings from the *same* chip, and a non-zero value indicates the presence of one or more bit flips. Intra-chip HD is not zero, in spite of the bit flip avoidance techniques discussed in the previous sections. The proposed techniques correct for most of the "jumps" that occur, but a few escape. Interestingly, all of the escapes can be caught by carrying out enrollment multiple times, each time at a different power supply voltage. The public data is updated incrementally in the repeated enrollments by marking those PNs associated with paths that "jump" as invalid. In other words, all "jumps" in our experiments are independent of temperature. Given that many modern architectures provide voltage scaling as a feature to deal with power consumption, it may be possible to leverage this feature here to provide error-free bitstrings. The average intra-chip HD is **0.038%**, or an average of **0.097** bits per 256-bit string.

5.2 NIST Statistical Analysis of Randomness

To test the randomness of the bitstrings produced by DPNC, we used a statistical test suite provided by the National Institute of Science and Technology [14]. NIST mainly focuses on evaluating randomness, while inter-chip HD measures uniqueness. These tests were applied to the bitstrings from each of the 30 boards. All of the bitstrings from passed all of the tests that are applicable to 256-bit strings, indicating that the HELP PUF is capable of generating high-quality bitstrings.

5.3 Analysis of Running Time and Key Ratios

We report bitstring generation times for HELP in this section as the average number of bits generated per minute. During enrollment, the time required to generate a single bit depends on several factors, including 1) the number of PNs that are read from memory before encountering the required number of consecutive similar values, 2) the value of k, and 3) the ratio between valid and invalid PNs. The average number of paths tested per bit during enrollment is **1,261** and the average rate of bit generation is **36.4** bits per minute. During regeneration, only those paths that were marked valid during enrollment are actually measured, so the average bit generation rate increases to **167** bits per minute.

5.4 Probability of Failure

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 probability of a single bit being generated incorrectly is $\mathbf{3.09x10^{-4}}$ (19 errors per (30 boards * 8 regenerations per board * 256 bits per regeneration)).

6. Conclusions

In this paper, we have presented details of a practical, realizable PUF, called HELP, and have proposed and demonstrated a novel bit generation technique called DPNC. The HELP PUF is based on measuring variations in path delays in the core logic macro(s) of the host chip. The results of Hamming Distance and NIST statistical test analyses show the bitstrings are genuinely random, unique, and highly reproducible.

7. REFERENCES

[1] R.S.Pappu, *et. al*; "Physical One Way Functions", *Science*, 297(6), 2002, pp. 2026-2030.

[2] B. Gassend, *et.al.;* "Controlled Physical Random Functions", *Conf. on Computer Security Applications*, 2002, pp. 149-160.

[3] K. Lofstrom, *et. al.;* "IC Identification Circuits using Device Mismatch", *SSCC*, 2000, pp. 372-373.

[4] P. Simons, *et. al.*; "Buskeeper PUFs, a Promising Alternative to D Flip-Flop PUFs", *HOST*, 2012, pp. 7-12.

[5] G.E. Suh, S. Devadas; "Physical Unclonable Functions for Device Authentication and Secret Key Generation", *DAC*, 2007, pp. 9-14.

[6] A. Maiti, P. Schaumont; "Improving the Quality of a Physical Unclonable Function using Configurable Ring Oscillators", *Conf. on Field-Programmable Logic and Applications*, 2009, pp. 703-707.

[7] Y. Su, *et. al.*; "A 1.6pj/bit 96% Stable Chip ID Generating Circuit Using Process Variations", *SSCC*, 2007, pp. 406-407.

[8] M. Majzoobi, *et. al.*; "Lightweight Secure PUFs", *ICCAD*, 2008, pp. 670-673.

[9] J. Ju, *et. al.*; "bitstring Analysis of Physical Unclonable Functions based on Resistance Variations in Metals and Transistors", *HOST*, 2012, pp. 13-20.

[10] D. Suzuki, K. Shimizu; "The Glitch PUF: A New Delay-PUF Architecture Exploiting Glitch Shapes", *CHES*, 2010, pp. 366-382.

[11] J. Li, J. Lach; "At-Speed Delay Characterization for IC Authentication and Trojan Horse Detection", *HOST*, 2008, pp. 8-14.

[12] C. Lamech, *et. al.*; "REBEL and TDC: Two Embedded Test Structures for On-Chip Measurements of Within-Die Path Delay Variations", *ICCAD*, 2011, pp. 170-177.

[13] OpenCores (website): *http:/www.opencores.org*

[14] NIST (website): *http://csrc.nist.gov/*