Error-Tolerant Bit Generation Techniques For Use With A Hardware-Embedded Path Delay PUF

J. Aarestad, J. Plusquellic
University of New Mexico
jaarestad@ece.unm.edu
jimp@ece.unm.edu

D. Acharyya
AdvanTest, Inc.
Dhruva.Acharyya@advantest.com

Abstract: *Cryptographic and authentication applications in application-specific integrated circuits (ASICs) and 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 number of random bits available to applications. In this paper, we propose a Hardware-Embedded Delay PUF (HELP) that is designed to leverage path delay variations that occur in the core logic macros of a chip to create random bitstrings. 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-level temperature and supply voltage variations. In particular, we evaluate the reproducibility of the bitstrings generated at 0°C, 25°C, and 70°C, and at nominal and ±10% of the supply voltage. An error avoidance scheme is proposed that provides significant improvement against bit-flip errors in the bitstrings.*

Keywords: Physical unclonable function, PUF, hardware security, cryptography, path delay variation

1 Introduction

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 paper, 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 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.

Unique Contributions of this Paper: The following are the novel contributions of 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 to environmental, measurement and meta-stability noise sources
- Fault-tolerant bit generation techniques that provide resilience against bit-flip errors caused by these noise sources

These characteristics of the HELP PUF are demonstrated on a set of 30 Virtex-II Pro FPGA boards. HELP is integrated into an AES functional unit and is evaluated across a set of 9 temperature-voltage (TV) corners, which represent commercial-grade standards. The bitstrings produced by each board are evaluated using statistical tests, which are designed to measure their uniqueness, reliability, and randomness.

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, and introduced in [15], 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 paper 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). The authors of [11] also leverage the high timing resolution provided by a DCM for Trojan detection and IC authentication.
- By using the core logic of AES itself, a large source of existing entropy is leveraged.

3 HELP PUF Overview

The HELP PUF produces a bitstring using a challenge-response mechanism. The challenge component for HELP consists of a randomly selected, two-vector test sequence applied to the inputs of the macro-under-test (MUT). The test sequence introduces a set of transitions that propagate through the core logic of the MUT and appear on its outputs. The responses are defined as the measured path delays, represented as 8-bit numbers as explained below, for each of the outputs. The delays on each MUT output are measured one-at-a-time.

The precision of the delay measurement impacts the stability of HELP. We use an embedded test structure called REBEL to obtain high-precision, digitized representations 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 an overview of the REBEL test structure, which consists 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 '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 'flush delay' mode (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 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

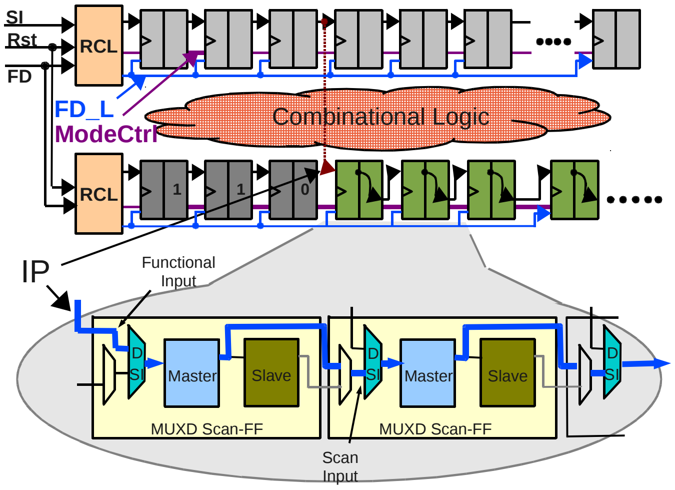


Fig. 1: REBEL embedded test structure.

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 Virtex-II Pro XUP FPGA boards [16]. The Virtex-II Pro board incorporates a 130-nm Virtex-II Pro device and permits power for the core logic to be supplied by an external power supply, which proved to be convenient for the TV corner testing carried out in our experiments[1]. 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 from OpenCores [13]. Space limitations on the Virtex-II Pro prevented inclusion of all 10 rounds 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 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 reference to Fig. 1. The number of FFs in this row is expanded from 256 to 264 to extend the delay chain for the IPs on the right end of the MUT.

The remaining components in Fig. 2 define the HELP PUF engine, and can be divided into the **Data Collection Engine**

1 Although the Virtex-II Pro chips are fabricated in an older technology, we expect similar (or better) results to those presented using chips fabricated in more advanced technologies.

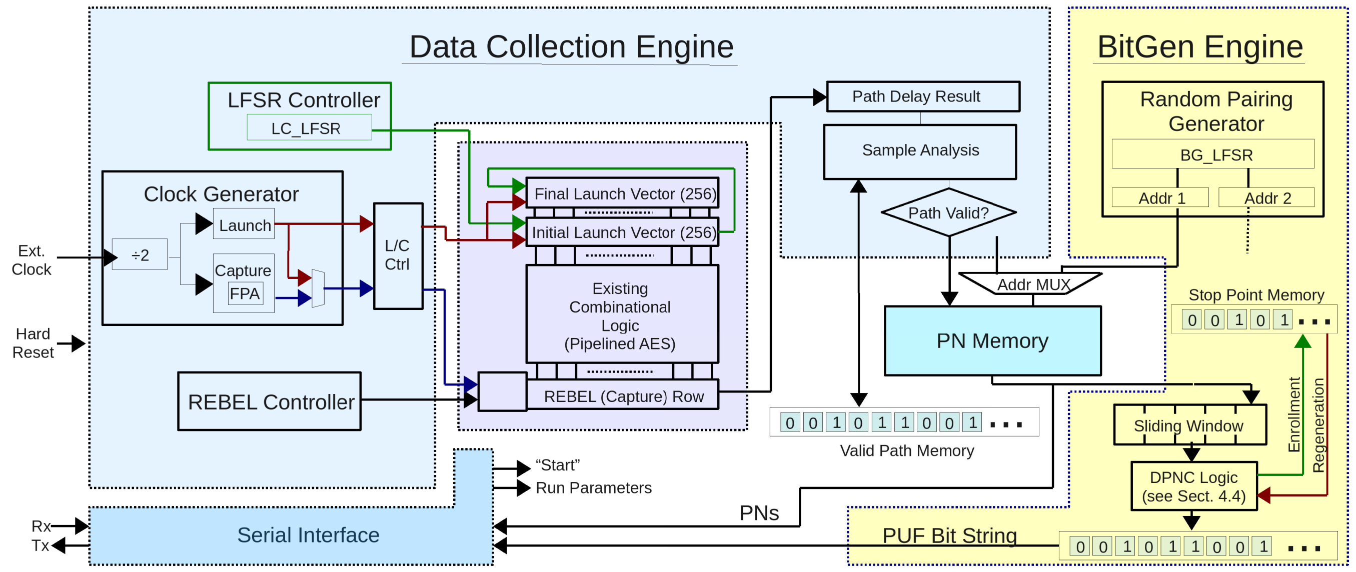


Fig. 2: Top-Level HELP System Diagram

(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 as needed.

The overhead of HELP is given in TABLE I. The resources under the column 'Single AES Stage' correspond to a single stage of the pipelined AES macro. The fully pipelined version is 10X larger, and therefore, the reported overhead for HELP in the first 3 rows would reduce by a factor of 10 in a full implementation, e.g., the values in the 'LUTs' row of TABLE I. would become '31220, 3931, 12.6%'.

TABLE I. HELP PUF Resource Overhead

	Single AES Stage	PUF w/o AES	% Overhead
Flip-flops	1297	456	35%
LUTs	3122	3931	126%
Slices	2146	1831	85%
RAMB16	0	58	---
BUFGMUX	1	4	400%
DCMs	0	3	---

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 increments/decrements in the phase shift of the Capture 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 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 II. Capture Clock Phase Adjustment

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

TABLE II. 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

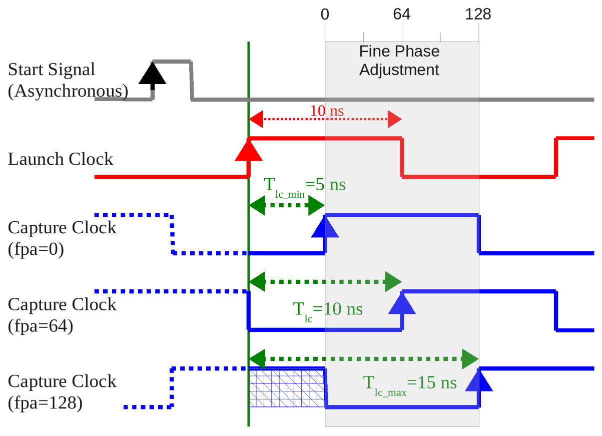


Fig. 3: Launch/Capture Timing Diagram

delays.

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 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). These values are technically stored during enrollment and then read back in from non-volatile or off-chip memory (public storage) during regeneration, and represent 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/Strong Bit Memory: A block RAM used by the Bit Generation Engine to *record* 'stop points' or 'strong bits' (depending on the bit generation method in use – see Sects. 4.4 and 4.5) during enrollment. The values stored in this memory, like the Valid Path Memory, are also components of the helper data.

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 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 bitstring 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 we show in Sect. 5 below, the paths that pass the stability test are different for each chip in the population.

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 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 (or have no transition), 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 TV Compensation and Jumps

Temperature and voltage can vary between enrollment and regeneration, which will introduce variations in path delays. 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 is necessary to calibrate for these types of environmental effects. We developed a calibration technique called Temperature/Voltage Compensation or TVCOMP to deal with TV variations. The principle behind TVCOMP is to derive a constant during regeneration that, when added to all PNs, shifts the PN distribution so that it matches the distribution obtained during enrollment. Calibration is carried out by computing a 'mean PN' during enrollment from a small subset of tests (we found 64 to be sufficient) which is then recorded as helper data. Later, during regeneration, the mean is again computed using the same set of tests and the difference between the two mean values is added as a 'correction factor' to the PNs obtained during regeneration. In our experiments, we found these correction factors to be in the range from -8 to +14 PNs, depending on the TV corner.

Unfortunately, not all types of path delay variations can be compensated for using TVCOMP. In particular, we found that a small number of the PNs tend to "jump" to new values well beyond that predicted by the correction factor. Although these jumps are exacerbated by TV variations, the underlying cause for the jump behavior is the appearance and disappearance of 'hazards' on off-path inputs to gates along the PUT. Under certain TV conditions, it is possible that an off-path input (which normally remains at its non-dominant value, e.g., a '1' on an input to an AND gate) changes momentarily to a dominant value. Depending on the relative timing between the appearance of the hazard and the actual signal transition along the tested path, it is further possible that the actual signal transition 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 given in the following sections.

4.4 The “Dual-PN Count” (DPNC) 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 makes short paths out of long paths and allows unbiased comparisons to be made along all paths. The trimmed PNs, called Mod-PNs, are then partitioned into two groups for bit generation purposes.

The diagram in Fig. 5 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. 5 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, TV variations are not completely compensated for by TVCOMP. This issue is dealt with by discarding additional PNs (beyond those discarded because of path stability problems as described in Sect. 4.2). In particular, Mod-PNs that fall into regions outside those delineated in the center portion of Fig. 5 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.

4.4.1 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 Sect. 4.3, 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

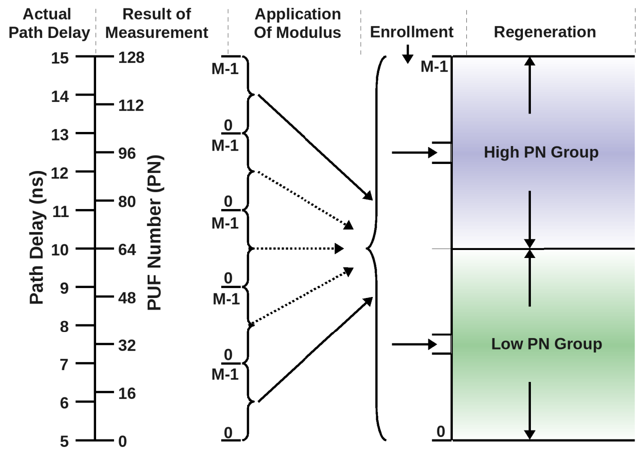


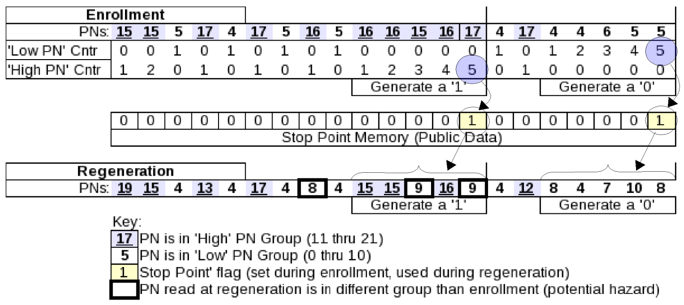
Fig. 5: Dual-PN Path Delay Binning Technique

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.

4.4.2 DPNC Example

An example of the DPNC process is shown in Fig. 4. 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-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. 4 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 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



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. 4: Dual-PN Count Method - Example

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).

4.5 The "UNM Difference" (UNMD) Technique

In [15], 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 (see Sect. 4.2) and therefore it does not need to calibrate for bias, i.e., UNM can compare short paths with long paths directly without truncating the high order bits of the PNs as is true for DPNC. The technique described in [15] 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 Sect. 4.2).

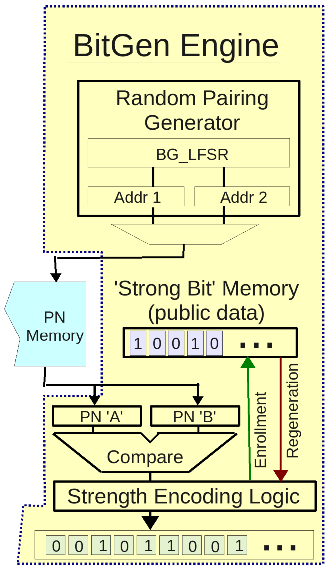


Fig. 6: BitGen Engine – UNMD Configuration

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.

4.5.1 Bit Generation Process and Procedure.

All components except for the BitGen Engine in Fig. 2 are identical for both the DPNC and UNMD techniques. The BitGen Engine for UNMD, shown in Fig. 6, randomly selects two PNs to compare (unlike DPNC which parses the PNs one at a time as shown in Fig. 4). 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'.

4.5.2 Thresholding Technique.

A thresholding technique similar to that proposed in [17] 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. 6 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. 7 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. 7 illustrate this concept.

4.5.3 TMR-Based Error Correction Scheme.

In Sect. 4.3, we described “jumps” as a worst-case condition and they 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. 8, 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 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

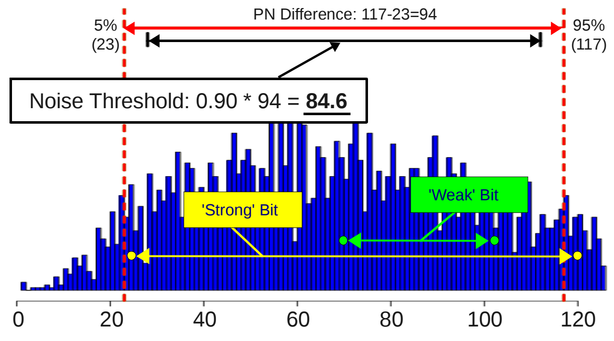


Fig. 7: Thresholding Technique

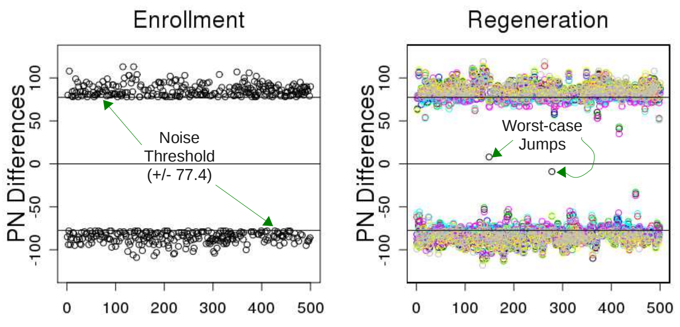


Fig. 8: Thresholding Technique

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. 8 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 [17] 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 Sect. 5.4.

5 Experimental Results And Analysis

We conducted environmental experiments 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, 0C, 25C and 70C, and three voltages, 1.35V, 1.50V and 1.65V. Data collected at 25°C and 1.5 is treated as enrollment data while the data collected at the remaining 8 TV corners is treated as regeneration data.

5.1 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. The 'average-of-the-averages' is then computed using the average HDs from all boards. Fig. 9 shows histograms for the inter-chip HDs and other statistical results obtained for the DPNC and UNMD techniques.

DPNC. The length of the bitstrings using the DPNC technique is 256 bits. The average inter-chip HD in Fig. 9(a) is **49.923%**. A Gaussian curve is shown fitted on top of the inter-

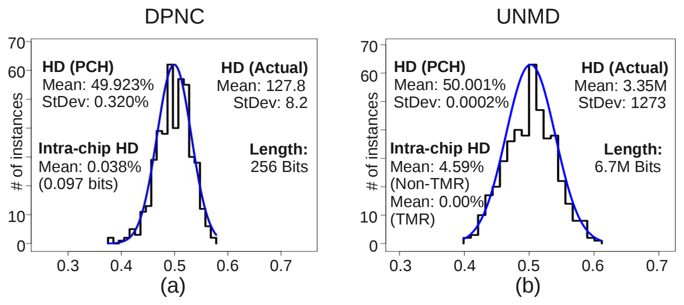


Fig. 9: HD Analysis

chip HD distribution as a means of illustrating its 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.

UNMD. The length of the bitstrings for the UNMD technique is 6,698,512. Fig. 9(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.

5.2 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.

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').

5.3 Analysis of Running Time

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.

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.

5.4 Probability of Failure: Results and Analysis

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.

UNMD. The POF 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 Sect. 4.5.3. 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. 10(a) illustrates the actual and theoretical error rates for each of the TMR-based bitstrings. Fig. 10(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.

6 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 un-obscured 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.

7 Conclusions

In this paper, we have presented details of HELP, a practical, realizable PUF, and have proposed and demonstrated two bit generation techniques called DPNC and UNMD. 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.

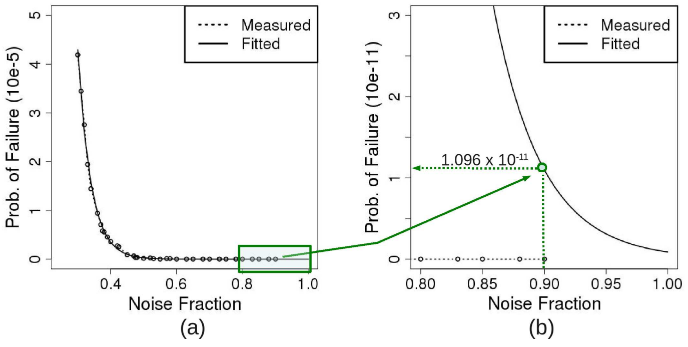


Fig. 10: Probability of Single-Bit Failure

8 References

[1] R.S.Pappu, B. Recht, J. Taylor, N. Gershenfeld; "Physical One Way Functions", *Science*, 297(6), 2002, pp. 2026-2030.
[2] B. Gassend, D. Clarke, M. van Dijk, S. Devadas; "Controlled Physical Random Functions", *Conf. on Computer Security Applications*, 2002, pp. 149-160.
[3] K. Lofstrom, W.R. Daasch, D. Taylor; "IC Identification Circuits using Device Mismatch", *SSCC*, 2000, pp. 372-373.
[4] P. Simons, E. van der Sluis, V. van der Leest; “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, J. Holleman, B. Otis; "A 1.6pJ/Bit 96% Stable Chip ID Generating Circuit Using Process Variations", *SSCC*, 2007, pp. 406-407.
[8] M. Majzoobi, F. Koushanfar, M. Potkonjak; "Lightweight Secure PUFs", *ICCAD*, 2008, pp. 670-673.
[9] J. Ju, J. Plusquellic, R. Chakraborty, R. Rad; “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, J. Aarestad, J. Plusquellic, R. Rad, K. Agarwal; "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] Nat'l Institute of Science & Technology (NIST) Computer Security Division, Statistical Tests: h*ttp://csrc.nist.gov/*
[15] J. Aarestad, P. Ortiz, D. Acharyya, J. Plusquellic; “HELP: A Hardware-Embedded Delay PUF”, *IEEE Design & Test*, Volume: PP, Issue: 99, March/April, 2013, pp. 1-8.
[16] Digilent, Inc. (website): *http://www.digilentinc.com*
[17] R. Chakraborty, C. Lamech, D. Acharyya, J. Plusquellic; “A Transmission Gate Physical Unclonable Function and On-Chip Voltage to Digital Conversion Technique”, *DAC*, 2013