An ASIC Implementation of a Hardware-Embedded Physical Unclonable Function

F. Saqib, M. Areno, J. Aarestad and J. Plusquellic
fareenasaqib1@gmail.com, jaarestad3@comcast.net, mareno@unm.edu and jimp@ece.unm.edu
Univ. of New Mexico

Abstract -- *Within-die variations in path delays are increasing with scaling. Although higher levels of within-die delay variations are undesirable from a design perspective, they represent a rich source of entropy for applications that make use of 'secrets', such as authentication, hardware metering and encryption. Physical Unclonable Functions or PUFs are a class of circuit primitives that leverage within-die variations as a means of generating random bitstrings for these types of applications. In this paper, we present test chip results of a hardware-embedded delay PUF (HELP) that extracts entropy from the stability characteristics and within-die variations in path delays. HELP obtains accurate measurements of path delays within core logic macros using an embedded test structure called REBEL. REBEL provides capabilities similar to an off-chip logic analyzer, and allows very fast analysis of the temporal behavior of signals emerging from paths in a core logic macro. Statistical characteristics related to the randomness, reproducibility and uniqueness of the bitstrings produced by HELP are evaluated across industrial-level temperature and supply voltage variations.*

1 Introduction

Physical Unclonable Functions or PUFs are circuit primitives that leverage within-die variations in ICs as a means of producing random bitstrings for applications such as authentication and encryption [1]. Each IC is uniquely characterized by random manufacturing variations, and therefore, the bitstrings produced by PUFs are unique from one chip to the next. Cloning a PUF, i.e., making an exact copy, is nearly impossible because it would require control over the fabrication process that is well beyond our current capabilities. A PUF maps a set of digital "challenges" to a set of digital "responses" by exploiting these physical variations in the IC. The entropy in the responses is stored in the physical structures on the IC and can only be retrieved when the IC is powered up. The analog nature of the entropy source makes PUFs 'tamper-evident', whereby invasive attacks by adversaries will, with high probability, change its characteristics.

The main distinguishing characteristic of PUF systems is the source of entropy that they leverage. Proposed entropy sources include variations in transistor threshold voltages [2], in propagation delays in inverter chains and ROs [1][3-11], in power up patterns in SRAMs [12], in leakage current [13], in metal resistance [14], and many others. The hardware-embedded delay PUF (HELP) that we investigate in this paper leverages path stability characteristics and within-die delay variations in core logic macros. HELP is unique because it does not measure and analyze within-die variations between identically designed structures, as is true for PUFs based on SRAM, ROs, delay chains, etc. Instead, it derives entropy from a tool-synthesized circuit macro, where paths of widely varying lengths are present. Consequently, the source of entropy for HELP is not based on raw path delays, because doing so would result in significant levels of undesirable bias. For example, comparing a short path with a long path would always yield the same result in every chip. Instead, randomness is distilled from the stability characteristics of the paths and the high frequency behavior of within-die variations, as we will illustrate in this paper.

The basic concepts of HELP are described in [10-11], where experimental results are presented using an instance of an Advanced Encryption Standard core logic macro implemented on a set of FPGA boards. In this paper, we implement HELP in a 90 nm ASIC using a IEEE-754 compliant floating point unit (FPU) as the core logic macro, and present results using data collected from multiple instances of the test chips. Differences in the path stability and within-die variation characteristics of FPGA and ASIC implementations, as well as differences in the internal connectivity of core logic macros themselves, impact the effectiveness of our proposed bitstring generation methods. In particular, techniques that rely entirely on path stability as the source of entropy break down in the FPU ASIC implementation, while methods based on the high frequency behavior of within-die variations improve, particularly w.r.t. reliability, over the FPGA implementation.

Although core logic macros provide a rich source of entropy, reconvergent-fanout in their connectivity structure causes significant amounts of glitching on the path outputs, which increases the probability that bit flips will occur during the bitstring regeneration process. Moreover, the glitching behavior is affected by supply voltage conditions (and temperature to a lesser degree). Therefore, conventional techniques for measuring path delays, e.g., those that vary the launch-capture clock interval in a sequence of tests, are not effective for quickly identifying and eliminating glitchy paths and obtaining accurate path delay measurements for stable paths. We proposed an embedded test structure (ETS) called REBEL in [16] that is designed to deal with these challenges.

REBEL is integrated directly into the scan-chain logic already present in the core logic macro and allows the temporal behavior of a signal to be captured as a digital snapshot using a single launch-capture event[1]. The digital snapshot is similar to that produced by a bench-top logic analyzer, which digitizes the voltage behavior but preserves the analog delay characteristics of a signal over time. The digital snapshots produced by REBEL significantly improve the ability of HELP to make good decisions about which path delays to use in the bitstring generation process. REBEL also allows timing information to be obtained for very short paths in the core logic macro and, by extending the physical length of the path, REBEL improves the robustness of the delay measurement process by allowing narrow glitches to die out.

In the ASIC implementation, HELP applies random test vectors to each of the 5 pipeline stages of the FPU while REBEL is used to time the combinational logic paths between each of the pipeline stages. We evaluate the HELP PUF at 9 temperature/voltage (TV) corners defined using all combinations of temperatures -40°C, 25°C and 85°C and supply voltages of nominal, +10% of nominal and -10% of nominal. Inter-chip hamming distance (HD), intra-chip HD and the NIST statistical tests are used to evaluate the quality of the bitstrings. The resilience of the HELP bitstring generation algorithms to reverse engineering and model building attacks are also discussed as appropriate.

2 REBEL ETS

In this section, we describe the modifications needed to integrate REBEL into a clocked-LSSD-style (CLSSD) scan architecture. The macro-under-test (MUT) in Fig. 1 is the combinational logic from a core logic macro. A row of scan flip-flops (FFs) is shown along the top which serve to launch transitions into the MUT. The bottom row is used to capture transitions that propagate through the MUT. REBEL ETS components are integrated into the bottom row and are labeled 'Row Control Logic' and 'front-end-logic' in the figure.

---

1. The implementation described in [10] uses a MUX-D-style scan chain, while the ASIC chip described in this paper uses a CLSSD-style scan chain.

Transitions can be launched into the MUT using standard manufacturing delay test strategies such as *launch-off-capture* and *launch-off-shift* [18]. In either of these two scenarios, the scan chain is loaded with the first pattern of the 2-pattern delay test and the system clock (Clk) is asserted to generate transitions in the MUT by capturing the output of a previous block or by doing a 1-bit shift of the scan chain. The transitions that propagate through the MUT emerge on some of its outputs. REBEL allows only one of these transitions to be measured at a time in a specific **region** of the MUT, as indicated by the label **PUT** for path-under-test in the figure. The PUT's transition normally drives only the D input on the capture FF. However, the REBEL component labeled 'front-end' logic (to be described) allows this transition to be diverted to the scan input (SI) on the FF. This special logic also converts all scan FFs to the right of this **insertion point** FF into a delay chain. A digital snapshot of the signal as it propagates along the delay chain can be obtained by de-asserting Clk. The digital snapshot can be used to determine the timing of the PUT, and because it captures the temporal behavior of the PUT, it can also be used to determine if any glitching occurred. This is a unique and powerful feature of REBEL that will be fully exploited in this work.

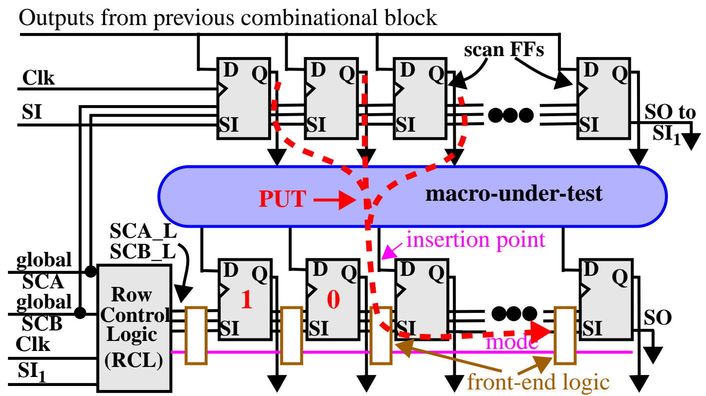

Fig. 1. REBEL Integration Strategy

A special mode called **flush-delay** (FD) can be used to implement the delay chain in CLSSD-based scan architectures. FD mode is enabled by asserting both the scan A and B clock signals simultaneously. These signals are labeled *global SCA* and *global SCB* in Fig. 1. With both signals asserted, both the master and slave of a scan FF are transparent, allowing any transitions on SI to propagate through both latches after a time Δt that represents the delay.

In addition to the designer-specified functional and scan modes, REBEL is required to implement two additional modes in the capture scan FFs shown along the bottom of Fig. 1. In particular, the scan FFs to the left of the insertion point need to preserve their contents during the Clk **launch-capture** (LC) event, while the FFs to the right of the insertion point need to implement the delay chain. These two modes are realized using the RCL block, a special scan chain encoding and the front-end logic shown in Fig. 1. The mode is controlled by configuring two FFs in the RCL block (to be discussed) while the scan chain encoding serves to specify the insertion point of the PUT.

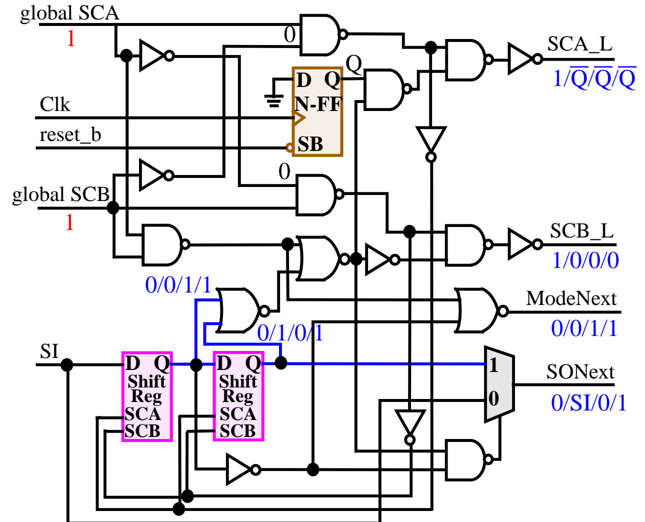

Fig. 2. REBEL Row Control Logic.

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 dedicated to allow the scan FFs in the capture row, hereafter referred to as **row-FFs**, to operate in functional or scan modes. The chip-wide scan signals, *global SCA* and *global SCB*, are used to control the operational mode of the chip. When both are low, functional mode is in effect. Scan mode is implemented when these signals are asserted in a non-overlapping fashion. The timing mode used by REBEL, called **REBEL mode**, is in effect when both of these signals are asserted, as illustrated by the annotations in Fig. 2.

When REBEL mode is in effect, the specific *sub-mode* of operation of the associated row-FFs is determined by the two shift registers. Table 1 identifies the sub-modes for each of the four configurations. Bit configuration "00" places all FFs in functional mode and is used for rows that serve to launch transitions into the MUT. Bit configurations "10" and "11" specify the *mixed mode* described above, where FFs to the left of the insertion point are in preserve-content mode while those to the right are in FD mode. The bit configuration "01" (FD continuation mode) puts all FFs in the REBEL row in FD mode. This allows the delay chain to be extended, which will be necessary in cases where we test multiple regions simultaneously, as is true for the FPU as described below. The outputs from the RCL block shown in Fig. 2 are annotated to show the values under each of these four bit configurations. Further operational details of the RCL block can be found in [16].

Shift Reg	Functionality
00	All scan FFs in row are in **functional mode**
11/10	Left scan FFs in preserve-contents mode, right scan FFs in FD mode, referred to as **mixed mode**
01	All scan FFs in row are in **FD continuation mode**

Table 1: Configuration modes for REBEL rows.

Fig. 3(a) shows a CLSSD FF used in the FPU macro. It consists of three latches. The functional path master-slave (MS) pair shown on the left is driven by Clk. The slave latch is dual ported and also serves as the master in the scan path MS pair on the right. Fig. 3(b) shows the additional 'front-end' logic for REBEL. The functional path's D-input is fanned out to a 2-to-1 MUX, which allows for the insertion of a macro's PUT into the delay chain during the REBEL test. This is accomplished with the *mode select logic* shown along the bottom of the fig-

ure. A specific insertion point is selected by pre-loading the row-FFs with a pattern of all ‘1’s followed by a ‘0’ from left to right along the row-FFs (see Fig. 1). Reference [16] elaborates on the operational details of the REBEL logic.

Note that the front-end logic adds only a small capacitive load to the functional path and therefore the impact of REBEL on performance is very small. The area overhead of REBEL within the FPU is 11.45%. All of the HELP bitstring generation components are implemented off-chip in the ASIC experiments. However, in previous work on FPGAs, we found the HELP engine area overhead to be approx. 100% of the area occupied by one stage of an Advanced Encryption Standard (AES) implementation and we expect the overhead to be similar for an ASIC implementation [11].

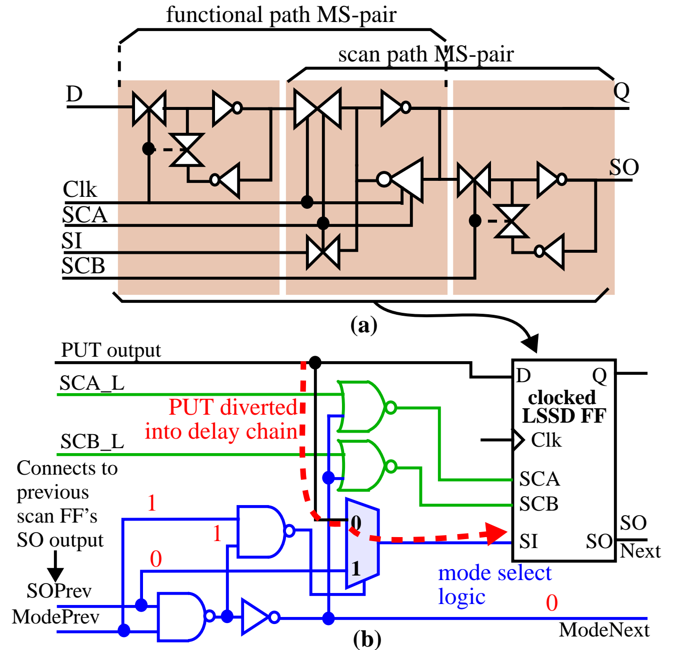

Fig. 3. (a) Modified clocked-LSSD scan FF and (b) additional ‘front-end’ logic.

3 Floating Point Unit (FPU) Macro

Fig. 4 shows a block level diagram of a floating point unit (FPU) incorporated on the chips, as well as the inserted **REBEL rows**, labeled RR*x* from 1 to 28. The design includes 817 FFs, which are wired together into a single scan chain with input SI_1 and output SO_1. A separate set of 70 shift registers are inserted on the inputs (top-most row in figure) which serve to enable a launch-off-capture testing strategy for the combinational logic in the first stage of the pipeline.

The FPU is designed as a 5-stage pipeline, labeled P_1 through P_5, with MUXes, decoders, adder/subtractors, a multiplier, etc. inserted between the pipeline registers. Four separate configurations are needed to test all the combinational logic between the pipeline stages. Two of the configurations place the FFs in P_1 and P_3 into functional mode while the FFs in P_2 and P_4 are configured into the REBEL modes. The other two configurations place FFs in P_0, P_2 and P_4 into functional mode while those in P_1, P_3 and P_5 are configured in the REBEL modes. Within each configuration, pairs of RRs are created to define a set of **regions**, e.g., see RR_{10}-RR_{11} and RR_{12}-RR_{13} in the figure. Within each region, the right-most RR block of the pair, e.g., RR_{13}, is configured into FD continuation mode to allow FFs on the right side of the mixed-mode row, e.g., RR_{12}, to be used as insertion points with delay chains that extend into RR_{13}. Otherwise, the insertion points on the right side of, e.g., RR_{12} would have very small or non-existent delay chains. Each of the 4 configurations allows up to 8 paths to be timed simultaneously.

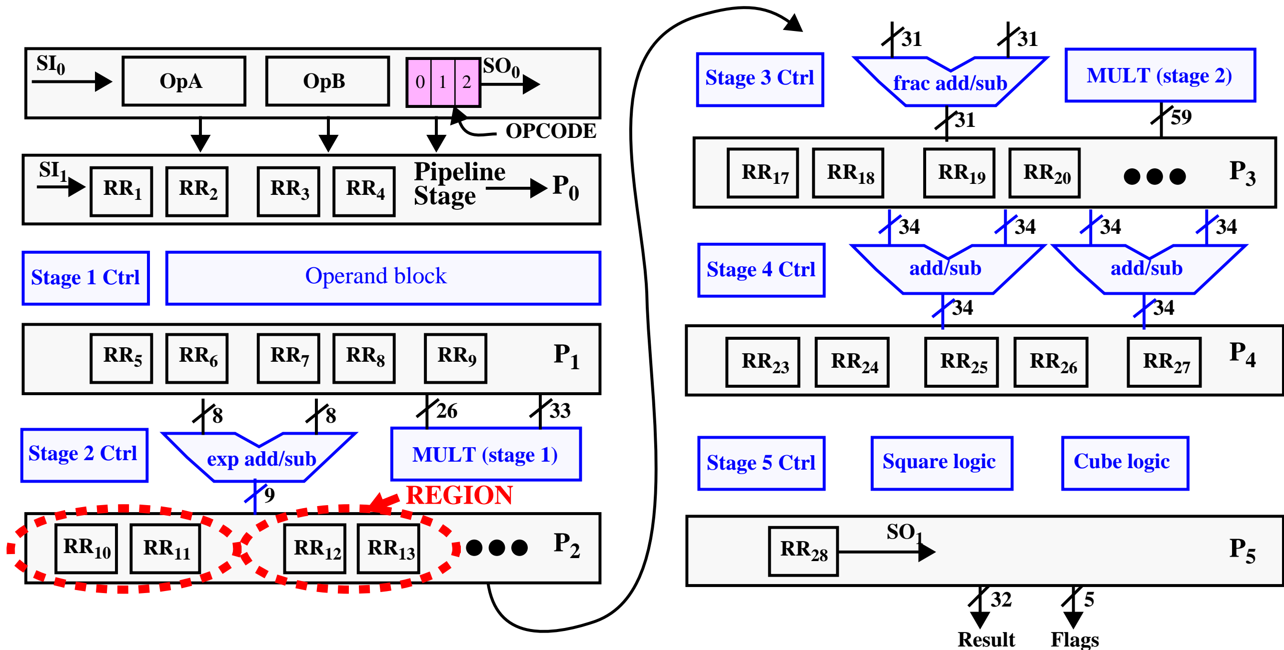

Fig. 4. Floating Point Unit Block Diagram

We apply a random testing strategy to the FPU where the values placed in the functional rows are generated by an pseudo-random number generator. For each random pattern, a sequence of configurations are placed into the REBEL rows, each of which changes the position of the insertion point incrementally from left-to-right across each of the mixed-mode rows. This is necessary because REBEL allows only one

path output per region to be timed during each test. Therefore, in order to test all PUT outputs that connect as inputs to a REBEL mixed-mode row, a sequence of tests are applied using the same random input pattern but with a different insertion point.

3.1 Clock Strobing

The timing relationship of several control signals and the **launch-capture interval** (LCI) are illustrated in Fig. 5. Prior to the LCI test, the SCA and SCB signals are asserted, which effectively activates the delay chains in the REBEL rows. The LCI test is defined by asserting and de-asserting the Clk signal, which launches transitions in the combinational logic. De-assertion halts all signals propagating along the delay chains and creates the digital snapshots.

The resolution of the measured delays is limited by the delay through each of the master-slave FFs that implement the delay chain. This delay is typically larger than the desired resolution, e.g., in our chips, it is approx. 500 ps. A technique called clock strobing is employed to increase the resolution. Clock strobing involves repeating the test sequence at incrementally longer LCIs.

0 (ns) 200 400 600 800 1000 1200 1400
SCA
prop. mode value
SCB
clock strobing
Clk
launch-capture-interval (LCI)

Fig. 5. Launch-Capture (LC) Test Sequence. Clock strobing applies a sequence of LCIs of different widths.

The launch-capture clock sequence is generated using a digital clock manager (DCM) on a Xilinx Zynq FPGA. The **fine phase adjust (FPA)** feature on the DCM allows the LCI to be set with a resolution of **17.857 ps**. A specific FPA is configured into the DCM by a state machine running on the FPGA which accepts an integer input parameter from a controlling C application. Valid values of the FPA are between 0 and 1120, which corresponds to a programmed LCI between 0 and 20.000 ns. The pulse-creation logic within the FPGA as well as the response characteristics of the FPGA pads prevents clock pulses narrower than approx. 2 ns from being produced on the clock output pin of the FPGA. In our experiments, we apply a sequence of LCI tests over the range of FPAs between 120 to 681 in FPA increments of 3. This results in the application of (681-120)/3 + 1 = 188 LCI tests with a Δt of 53.6 ps between consecutive LCIs.

3.2 Digital Snapshots

The raw data captured in the delay chain is a string of binary bits, one string for each of the 188 LCI tests applied to test a path[1]. Fig. 1 shows the digital snapshots for the first 21 LCI tests of a path in a vertical sequence. The insertion point in this example is FF_{15} of the mixed-mode row RR_{12} from Fig. 4. The programmed FPA for each snapshot is displayed on the left side of the figure. The first FPA (120) shows a sequence of 6 0's in the left portion of the snapshot. This indicates that a falling edge propagated along 6 elements of delay chain, i.e., through FF_{15} through FF_{20}, before being halted by the capture event. In each subsequent snapshot up through FPA 126, the edge continues to propagate through FF_{20} but fails to reach FF_{21} until FPA 129 is applied. The falling edge requires 9 more FPAs, i.e., 129 through 153, to propagate completely through FF_{21}.

Combinational logic in pipeline
insertion point
FF_{15}
FF_{21}
RR_{12}
RR_{13}

```
FPA
120 xxxxxxxxxxxxxxx00000011111111111
123 xxxxxxxxxxxxxxx00000011111111111
126 xxxxxxxxxxxxxxx00000011111111111   *
129 xxxxxxxxxxxxxxx00000001111111111
132 xxxxxxxxxxxxxxx00000001111111111
135 xxxxxxxxxxxxxxx00000001111111111
138 xxxxxxxxxxxxxxx00000001111111111
141 xxxxxxxxxxxxxxx00000001111111111
144 xxxxxxxxxxxxxxx00000001111111111
147 xxxxxxxxxxxxxxx00000001111111111
150 xxxxxxxxxxxxxxx00000001111111111
153 xxxxxxxxxxxxxxx00000001111111111
156 xxxxxxxxxxxxxxx00000000111111111
159 xxxxxxxxxxxxxxx00000000111111111
162 xxxxxxxxxxxxxxx00000000111111111
165 xxxxxxxxxxxxxxx00000000111111111
168 xxxxxxxxxxxxxxx00000000111111111
171 xxxxxxxxxxxxxxx00000000111111111
174 xxxxxxxxxxxxxxx00000000111111111
177 xxxxxxxxxxxxxxx00000000111111111
180 xxxxxxxxxxxxxxx00000000111111111
```

Fig. 6. Illustration of a partial sequence of digital snapshots produced from clock strobing a path.

3.3 Path Selection Criteria and Path Timing

All bitstring generation algorithms described below use timing information from paths that are deemed 'stable'. A stable path requires that all digital snapshots contain exactly 1 transition, with the edge proceeding in an orderly fashion from one delay chain FF to the next. Note that the testing of paths that glitch, i.e., those that produce more than 1 transition, is immediately aborted. The subset of digital snapshots obtained for the path shown in Fig. 6 qualifies the path as stable (assuming the remaining snapshots exhibit similar behavior).

On the other hand, Fig. 7 shows a sequence of snapshots that exhibits 'wobble' where the '0' reverts back to FF_{20} at FPA 132 after progressing to FF_{21} at FPA 129. We found that including paths that exhibit wobble, particularly those that wobble multiple times in the sequence of snapshots, are more likely to exhibit glitchy behavior at other TV corners. The timing associated with paths that glitch, in turn, can vary significantly and introduce bit flips. HELP also classifies paths as unstable if the *range* of FPAs obtained from multiple samples is larger than a threshold or if the number of *replicated snapshots* for each FF is below a threshold, i.e., the transition propagated too quickly through a FF. All of these criteria together significantly improve the ability of HELP to identify paths that are stable at enrollment and remain stable at other TV corners. Doing so is critically important to reducing the number of bit flips caused by *jumps*, which are discussed below.

Combinational logic in pipeline
insertion point
FF_{15}
FF_{21}
RR_{12}
RR_{13}

```
FPA
120 xxxxxxxxxxxxxxx00000011111111111
123 xxxxxxxxxxxxxxx00000011111111111
126 xxxxxxxxxxxxxxx00000011111111111
129 xxxxxxxxxxxxxxx00000001111111111
132 xxxxxxxxxxxxxxx00000011111111111
135 xxxxxxxxxxxxxxx00000001111111111
138 xxxxxxxxxxxxxxx00000001111111111
```

Fig. 7. Digital snapshots illustrating 'wobble', a condition that can disqualify the path as being stable.

A user-defined **target FF** is used to determine which FPA is selected as

1. Note that paths with no transitions are only tested with the first LCI and further testing is aborted to save time.

the timing for the path. The target FF is the FF at a fixed position from the insertion point, for example, if the target FF is 6 than FF_{21} in Fig. 6 is the target FF for the insertion point at FF_{15}. The algorithm that we use to determine the timing for a path collects and parses the snapshots in reverse order, i.e., starting with the snapshot produced under the longest LCI and progressing to shorter LCIs. The search for the FPA that represents the timing ends when the transition value for the path, which is '0' for the snapshots in Fig. 6, is 'pushed back' to the FF to the immediate left of the target FF (FF_{20} in Fig. 6). This happens at FPA 126 and represents the FPA where the propagating 1-to-0 transition is just about to enter the target FF.

The only remaining issue for path selection is dealing with paths that are timed by more than one test vector. We found that approx. 40% of the paths that are found to be stable in the FPU macro are timed more than once. This occurs because the random sequence of tests applied to the macro provide no guarantee that each test pattern sequence tests only unique paths. Fortunately, it is relatively easy to identify re-tested paths. Our algorithm stores the FPAs and insertion points for stable paths in a memory, which is searched when a new path is tested and found stable. If a match to the FPA and insertion point is found, the path is discarded. The match criteria to the FPA includes a small tolerance to account for measurement noise.

4 Test Chip Results

We applied 38 random vectors to the FPU in 50 copies of the test chip and tested a total of 31,236 paths using the 4 configurations described earlier. The average number of stable paths per chip is approx. 2,700, which represents approx. 8.6% of all paths tested. We refer to the FPA timing value that we obtain for these stable paths as PUF Numbers or **PNs**. The PNs are distilled from the FPAs described above using the formula given by Eq. 1. Therefore, the range of FPAs from 120 to 681, in steps of size 3 translates into PNs from 0 to 187 in steps of size 1. The actual delays associated with PN = 0 is 2.143 ns and PN = 187 is 12.161 ns.

$$PN = \frac{(FPA - 120)}{3} \qquad \textbf{Eq. 1.}$$

We evaluate the bitstrings produced from our bitstring generation algorithms using the standard statistical criteria that has emerged for judging the quality of a PUF. **Inter-chip hamming distance (HD)** is used to determine the *uniqueness* of the bitstrings among the population of chips. Inter-chip HD computes the average number of bits that are different across all pairings of chip bitstrings and expresses the average as a percentage. The best possible result is 50%, i.e., on average, half of the bits in the bitstrings of any two arbitrary chips are different. The NIST statistical test suite is used to evaluate the *randomness* of the bitstrings produced by each chip [17]. In general, the NIST tests look for *patterns* in the bit strings that are not likely to be found at all or above a given frequency in a 'truly random' bitstring. For example, long or short strings of 0's and 1's, or specific patterns repeated in many places in the bit string work against randomness. The output of the NIST statistical evaluation engine is the *number of chips* that pass the *null hypothesis* for a given test. The null hypothesis is specified as the condition in which the bitstring-under-test is random. Therefore, a good result is obtained when the number of chips that pass the null hypothesis is large. Third, we use **Intra-chip HD** to evaluate *stability* of the bitstrings, i.e., the ability of each chip to reproduce the same bitstring time-after-time, under varying temperature and voltage conditions. It is carried out on the bitstrings produced by each chip across the 9 TV corners. Ideally, all 9 bitstrings are identical and the Intra-chip HD is 0%. An average Intra-chip HD is computed using the individual chip results. In addition to these statistical tests, we also evaluate other security related metrics as appropriate, including, for example, how difficult it is for an adversary to reverse engineer the bitstring and/or to model build the PUF.

4.1 Path Delay Distributions

The delay distributions for $CHIP_1$ for each of the TV corners are shown in Fig. 8. The PNs are obtained with the target FF set to 8. The PN number (delay) is plotted along the x-axis against the number of instances on the y-axis. The graph on the left superimposes the distributions obtained when the supply voltage (V_{DD}) is set to 1.20V (nominal) and the temperature is set to each of 25°C, -40°C and 85°C, while the graph on the right superimposes the distributions at these temperatures but with V_{DD} set to +/- 10 of nominal. It is clear that supply voltage has a larger impact on delay than temperature. All graphs show that the path distribution is skewed, with larger numbers of shorts paths than longer paths.

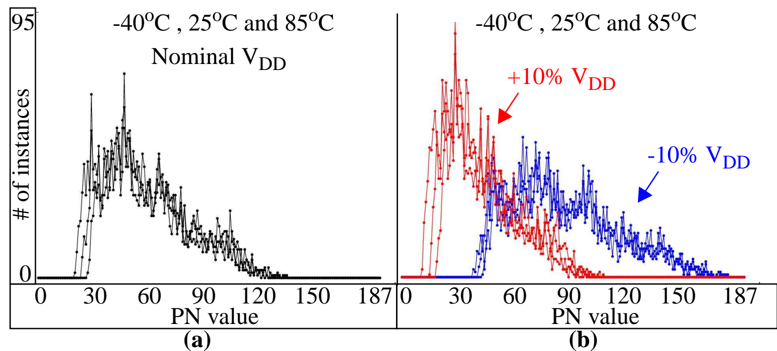

Fig. 8. Path delay distributions at (a) nominal voltage and (b) +/- 10% of nominal. Temperature distributions are superimposed.

Although not shown, the distributions from other chips are similar in shape but vary in width and position along the x-axis, which is caused by chip-to-chip process variations. More importantly, the ordering of the tested paths in each chip's distribution is unique and is determined primarily by within-die process variations, which is an important source of entropy for HELP. Two of the bitstring generation techniques described below generate and then extract information from the delay distribution at 1.20V, 25°C (called the **enrollment** distribution) as a means of improving the robustness of the bitstring regeneration process.

4.2 Path Stability and Within-Die Path Delay Variations

As mentioned in the Introduction, the entropy source for HELP is defined from two components; path stability and within-die variations in delay. This dual source of entropy is a significant benefit to improving the randomness of the generated bitstrings, as well as increasing the difficulty of reverse engineering attacks. In previous work, we used an FPGA implementation of the Advanced Encryption Standard (AES) as the core macro and found that both sources of entropy provided a high degree of randomness in the generated bitstrings [10-11]. We then developed several bitstring generation methods, one of which leverages only the path stability entropy source. We call this method 'Universal, No Modulus' or **UNM**.

Although UNM performed well in the AES implementation in previous work, it does poorly when applied to the data from the FPU. In particular, the computed inter-chip hamming distance is only 38%. The reason it does poorly is shown in Fig. 9. The x-axis assigns a path ID (**PID**), from 1 to 4650, to each path that is identified as stable in at least one chip while the y-axis gives the number of chips that each PID is found to be stable in. Therefore, path IDs with a value of 1 are unique to one chip and are not found to be stable in any other chip. Con-

versely, path IDs with a value of 50 are stable in every chip. The top portion of the graph, and in particular the top line, is densely populated with points, and indicates that there is a high-level of commonality in a large fraction of the stable paths. As we show below, the dependency of the UNM technique on requiring the opposite condition, that a large number of the stable paths for a given chip do not occur in every chip, causes it to score poorly on the uniqueness criteria.

On the other hand, the level of within-die variations within the FPU are sufficient to enable the generation of high quality bitstrings using bitstring generation techniques that are designed to leverage them. We refer to these techniques as 'Universal, No Modulus, Difference' or **UNMD** and 'Dual-PN Count' or **DPNC**. The graphs in Fig. 10 illustrate the process we use, called regression analysis, to quantitate the level of within-die variations across our set of chips. Linear regression is applied to scatter plots which are constructed using the delays from pairings of paths across the chips [19]. Fig. 10(a) depicts the scatterplots for 4 path pairings. We selected paths that are stable in all 50 chips as a means of capturing the full extent of within-die variations in our sample, i.e., we used PIDs from the top line of Fig. 9. Linear regression analysis first computes a least squares estimate (LSE) of a best fit line through the data points of each scatter plot. Several of the LSE lines are labeled in Fig. 10(a). The LSE line tracks chip-to-chip process variations.

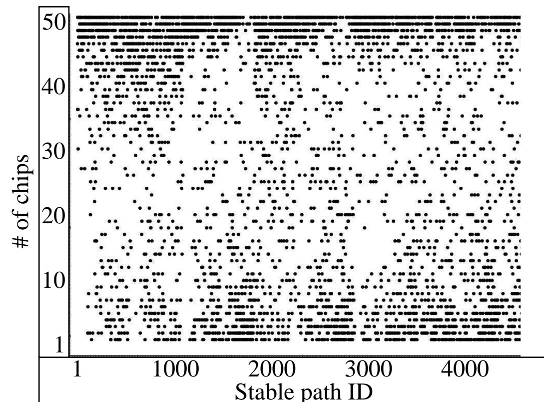

Fig. 9. Commonality in the paths that are stable across all chips.

Within-die variations (and random noise) are represented by the vertical offsets of the data points from the LSE line. The vertical offsets are called residuals (several are labeled in the figure). We compute the **range** of the residuals in each scatterplot, which is given by the sum of the distances from the regression line of the most negative and positive data points, as illustrated in Fig 10(a). Fig. 10(b) plots the within-die variation results for the 716 path pairings that are stable in all chips. The average delay of the first path ($path_a$) is plotted along the x-axis against the range of the residuals along the y-axis. Within-die variation varies from approx. 60 ps to almost 1.25 ns. This information will be used to tune parameters of the bitstring generation methods described below.

4.3 Jumps, Measurement Noise and Temperature/Voltage (TV) Noise

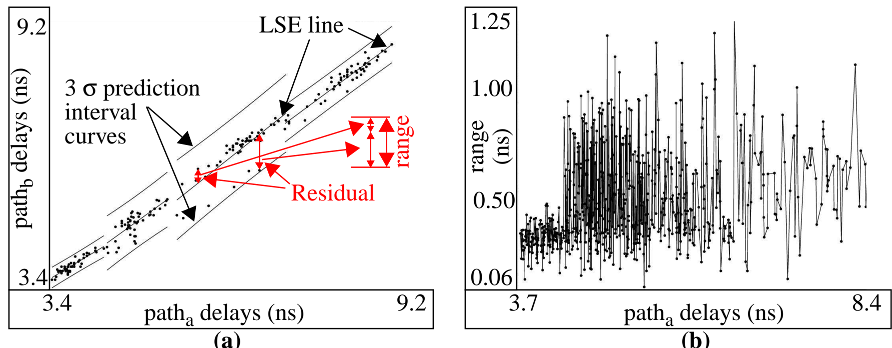

Fig. 10. Within-die Variation Analysis on common paths across chips. (a) Scatterplots for 4 path pairings, (b) average delay against regression 3 σ range.

The terms **enrollment** and **regeneration** are used in reference to bitstring generation processes associated with PUFs. Enrollment is carried out when a new bitstring is required, while regeneration refers to the process of reproducing the bitstring. The application determines whether exact reproduction is required, e.g., encryption requires exact reproduction while authentication typically does not. However, for any application, the closer the regenerated bitstring is to the enrollment bitstring, the better. The main challenge associated with reproducing the bitstring exactly is dealing with measurement and TV noise. These noise sources change the measured values of the entropy source, possibly causing bits in the bitstring to **flip** or change value from '0' to '1' and vise versa.

$$PN_{TVCOMP} = (PN_{regen} - \text{mean}(R)) \times \frac{\text{range}(E)}{\text{range}(R)} - \text{mean}(E) \qquad \textbf{Eq. 2.}$$

Error correction is commonly used to fix errors in regenerated bitstrings in cases where exact reproduction is needed, e.g., encryption [1]. Our approach uses thresholding to avoid errors and redundancy to fix errors introduced by random measurement noise (methods that we discuss in the following sections). We apply a calibration method, called Temperature/Voltage Compensation or TVCOMP, to deal with TV noise. The principle behind TVCOMP is to derive scaling constants during enrollment and regeneration that allow a linear transformation to be applied to the PNs obtained during regeneration. The linear transformation shifts and scales the regenerated PN distribution and makes it similar to the distribution obtained during enrollment. Calibration is carried out by computing a *mean PN* and a *PN range* during enrollment which are recorded as ***Helper Data***. Helper Data is stored in a non-volatile location for use during regeneration. During regeneration, the mean and range are again computed and the PNs are transformed as given by Eq. 2. The transformation works well to compensate for non-random changes in the PNs such as those introduced by TV noise.

Besides measurement and TV noise, HELP has an additional source of noise called **jumps**. Jumps are dramatic changes in the PN values of certain paths when the TV conditions change. The underlying cause for the jump behavior is the appearance and disappearance of 'hazards' on off-path inputs of gates which are components of the tested path. Under some 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 delay. Unfortunately, there is no way to predict which path delays will be effected by jumps. We describe redundancy techniques that are effective in dealing with this problem in the next section.

4.4 Difference Thresholding and Spatial Redundancy Techniques

HELP makes use of thresholding and spatial redundancy techniques as a means of allowing the user to trade-off reliability with bitstring

generation time and Helper Data size for applications that require exact regeneration of the bitstring. The term *spatial* refers to the multiple, redundant copies of the bitstring that are produced across different regions of the entropy space. Fig. 11(a) illustrates one form of thresholding that we propose, called **difference thresholding**, and the spatial redundancy technique. Difference thresholding is used in the UNMD bitstring generation method. The illustration portrays the sequence of operations that are performed to generate a bitstring and the corresponding set of Helper Data that is produced during enrollment.

First, **TV noise thresholds** labeled +Tr and -Tr are derived from the width of the delay distribution profile for the chip, similar to the one shown for enrollment in Fig. 8. We found that making the threshold dependent on the chip's distribution profile works well in keeping the size of the bitstrings across all chips approximately equal. A bit is generated under difference thresholding by computing the signed difference between a pairing of PNs, and then comparing the magnitude of the difference with the +Tr and -Tr thresholds. If the difference falls within the +Tr and -Tr region, the PN pairing is considered **invalid** and is not allowed to generate a bit. When this occurs, the Helper Data, labeled as *valid bitstring* in the figure, is updated with a '0' as a means of instructing subsequent regeneration processes that this PN pairing is to be skipped. In contrast, we use the term **strong bit** to refer to cases where the PN difference falls above +Tr or below -Tr. In this case, the sign of the difference is used to generate the secret bit and the valid bitstring is updated with a '1'.

A set of example PN differences are plotted along the x-axis in Fig. 11(a) against their differences along the y-axis. The left-most 6 PN pairings shows the process of generating a secret bitstring of length 4. The remaining sequence of PN pairings illustrate the process associated with spatial redundancy, in particular, a redundancy scheme that uses 3 copies of the bitstring (the spatial redundancy scheme extends to any odd number of copies). The left-most bitstring labeled *Secret BS* is generated from the first 4 strong bits encountered in the PN pairing sequence as discussed above. The second bitstring labeled *Redundant BS1* is produced from the next sequence of PN pairings but has the additional constraint that each of its bits must match those in the first bitstring. During its construction, it may happen in the left-to-right parsing of the PN pairings that a strong bit is encountered that does not match the corresponding position in the Secret BS. In the example, this occurs at the position indicated by the left-most bold '0' in the valid bitstring. Here, we encountered a strong bit with a value of '0'. But the Secret BS requires the first bit to be a '1', so this strong bit is skipped. This process continues until redundant bitstrings BS1 and BS2 bitstrings are constructed.

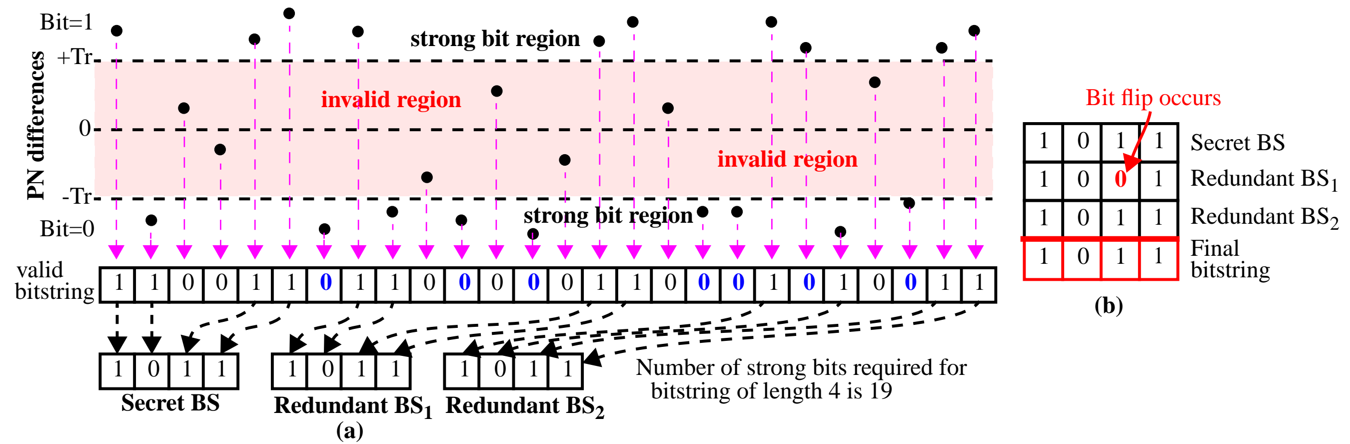

Fig. 11. (a)Illustration of spatial redundancy during enrollment and (b) bit flip resiliency during regeneration.

The number of strong bits required to generate a secret bitstring of length 4 is approx 5x or 20. From the example, this is evaluated by counting the number of '1's and bolded '0's in the valid bitstring, which is given as 19. The benefit of creating these redundant bitstrings is the improved tolerance that they provide to bit flips. For example, during regeneration, the three bitstrings are again produced, but this time using the valid bitstring to determine which PN pairings to consider. In cases where a *jump* occurs, it is possible that the difference from a PN pairing for a strong bit becomes displaced from the strong bit region across the '0' line in Fig. 11(a). The change in the sign of the difference would normally introduce a bit flip. However, with spatial redundancy, a bit flip can be avoided if no more than 1 bit flip occurs in a single column of the matrix of bits created from the 3 bitstrings. For example, the first 3 rows of the matrix of bits in Fig. 11(b) are constructed during regeneration in a similar way to those shown in Fig. 11(a) for enrollment. The bottom row represents the final secret bitstring and is constructed by using a majority vote scheme. The bit flip shown in the third column has no effect on the final bitstring because the other two bits in that column are '1', and under the rule of majority voting, the final secret bit is therefore defined as '1'.

4.5 Bitstring Generation Methods

As indicated earlier, we test 31,236 paths on each chip and found 8.6%, i.e., 2,700 paths on average, pass our stability criteria per chip. Our statistical analysis requires the number of bits to be equal in the bitstrings of all chips so we reduce the number of PNs to the smallest number produced by one of our chips, which is **2,519**. The actual bitstring size is dependent on the number of PNs, the level of redundancy and on the bitstring generation method, as described in the following three subsections.

4.5.1 Universal, No Modulus (UNM) Bitstring Generation Method

The UNM technique is the simplest of the 3 methods from an implementation point of view, and also produces the smallest amount of Helper Data. Unfortunately, it is also the easiest to reverse engineer and, as we discussed above, the statistical quality of the generated bitstrings is dependent on the macro-under-test. The enrollment distribution (25°C, 1.20V) in Fig. 12(a) is annotated to illustrate components of the analysis of the UNM bitstring generation algorithm. A region in the center of the distribution labeled *UNM margin* identifies PNs that are ***excluded*** from the bitstring generation process. In contrast, the region below the left-most threshold labeled **low bin** and the region above the right-most threshold labeled **high bin** identify PNs that are valid. The algorithm creates this distribution and sweeps the two thresholds from left to right across the distribution in discrete steps, while maintaining the *UNM margin* between them. At each step, the number of elements in the low and high bins are counted. The position where the number of PNs in each bin is closest to being equal is saved and used in the bit-

string generation process. The goal to equalize the cardinality of the two bins maximizes the chance that the generated bitstrings will have equal numbers of ‘0’ and ‘1’s, which is an important statistical quality metric.

The distance between the vertical thresholds is fixed at a value that ensures that changes in any of the PN values at different TV corners do not exceed half of this margin. Our experimental results show that *jumps* are the worst case condition to deal with and these force the *UNM margin* to be set to approx. 3.8 ns (this margin is sufficient to prevent bit flips from occurring in any of the chips). During regeneration, the positions of the thresholds are again determined but, in this case, the midpoint between them is used to decide whether a given PN is in the *low* or *high* bin as shown in Fig. 12(b). The Helper Data is consulted to ensure the same PNs used in the bitstring generation process carried out during enrollment are used here during regeneration. The value of these PNs in the regeneration data is likely to be different than their value during enrollment because of TV noise. As long as none move from their original bin across the *midpoint* line to the other bin, then no bit flips occur.

The bitstring is generated by pseudo-randomly selecting pairs of PNs from the low and high bins to compare. A linear-feedback-shift-register or LFSR implemented with a primitive polynomial is used to generate the pseudo-random pairings as a means of ensuring that all possible pairings can be generated. As an example, if the sum of cardinalities of the two bins is 1,000, then it is possible to generate up to 1,000*999/2 = 499,500 bits. An XOR-style of comparison if used to generate each bit, i.e., if both PNs being compared are in the low or both are in the high bin, then a ‘0’ is generated, otherwise a ‘1’. We recognize that re-using each of the *n* PNs in *n-1* comparisons (all combinations) subjects the PUF to model-building attacks for applications such as authentication, which, by definition, reveal the responses to the external world. In this paper, we use all combinations only as a means of providing a more significant sample to the evaluation processes designed to determine statistical quality.

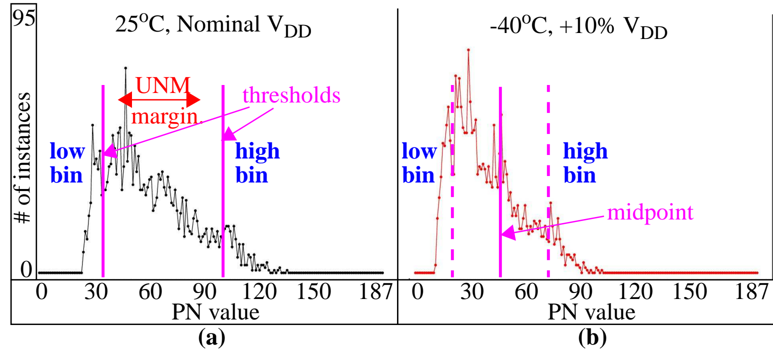

Fig. 12. Illustration of UNM algorithm on (a) enrollment delay distribution and (b) regeneration delay distribution.

From this algorithm, it is clear that the bitstring for one chip would be different from the bitstring for another chip if a significant number of the PNs for each chip are associated with distinct path IDs (PIDs). Given the wide margin between the low and high bins, within-die variations in path delays cannot effect the outcome of the bitstring generation process. In other words, if two chips select the same set of stable paths, the LFSR will select PNs in the same order and produce the same bitstring for both chips because within-die variations are not large enough to move PIDs from the low bin to the high bin and vise versa in different instances of the chips. Therefore, UNM depends entirely on the randomness of path stability. As indicated above, the inter-chip HD for UNM is only 38%, which is considerably lower than the ideal of 50%. Therefore, when using the FPU as the macro-under-test, the entropy content associated with path stability is not sufficient to produce a quality PUF.

The partitioning of the distribution into low and high bins in combination with the public Helper Data, that identifies which paths are stable and participate in bitstring generation on each chip, increases the ease of carrying out reverse engineering attacks on UNM. If the seed and LFSR are known, then an adversary can simulate the tested paths to determine whether the PN is in the low or high bin for a given chip, thereby enabling the secret bitstring to be reconstructed. Obscuring the Helper Data prevents this attack but is difficult to implement. The UNMD and DPNC bitstring generation methods described below leverage both path stability and within-die variations, making this type of attack more difficult or impossible to carry out.

The Helper Data for UNM is a ***path bitstring*** with one bit allocated for each tested path. If a path is stable and the PN falls in either of the low or high bins, then a ‘1’ is recorded, otherwise a ‘0’. The size of the path bitstring is related to the fraction of stable PNs and the UNM margin. For example, we indicated that 8.5% of the paths are stable, so a bitstring of length 256 would require approx. 6 Kbits of Helper Data, computed as 256/0.085 * 2. The factor of 2 assumes that the thresholding preserves half of the distribution in the sum of cardinalities of the 2 bins.

4.5.2 Universal, No Modulus, Difference (UNMD) Bitstring Generation Method

UNMD uses the difference thresholding technique described in Section 4.4. Therefore, the PNs that are valid for comparisons can appear anywhere in the distribution shown in Fig. 12(a), not just in the tails. More importantly, both path stability and within-die variations play a key role in deciding which comparisons generate secret bits. This seemingly small change adds significantly to the entropy of the system and dramatically improves the randomness and uniqueness quality metrics of the secret bitstring.

The drawback of UNMD over UNM is in the size of the public Helper Data. Both methods require a *path bitstring* that records which paths are stable in the sequence of applied random tests during enrollment. However, UNM can then update the path bitstring to exclude stable paths with PNs that are not in the tails of the distribution shown in Fig. 12(a), while UNMD requires a second public data bitstring, called the *valid bitstring*, to record the thresholding result as discussed in Section 4.4. The size of the valid bitstring for UNMD is dependent on the difference threshold and the level of spatial redundancy, e.g., 3, 5, 7, etc.

Difference thresholding was described in Section 4.4 and in reference to Fig. 11. Within-die variations can change the sign of the PN differences in different chips, thereby adding entropy to the bitstrings. However, this only occurs when the two paths of a pairing have delays that are within the range of within-die variation[1]. In Section 4.2, we used regression analysis to determine that within-die process variations introduce delay variations of less than approx. 1.25 ns, which is a range from 0 to 23 PNs (53.6 ps/PN * 23 PNs ~ 1.25 ns). Therefore, path pairing which produce PN differences that are larger than 23 PNs are likely to have the same sign, and corresponding bit value, across chips, which, in turn, acts to reduce the entropy in HELP.

1. The PN difference must also be larger than the TV noise threshold as we discussed in Section 4.4.

We propose a modification to difference thresholding, called **modulo thresholding,** that addresses this problem. The scheme is illustrated in Fig. 13. The enrollment distribution from Fig. 12 is annotated in Fig. 13 with PN data points from 2 path pairing and their corresponding PN differences labeled $PNDiff_a$ and $PNDiff_b$. While $PNDiff_a$ is within the range of within-die variation, i.e., has value of 15, $PNDiff_b$ is larger with a value of 30. Therefore, the bit value derived from $PNDiff_b$, in cases where this pairing is identified as stable, is likely to remain the same in these chips, and will adversely impact inter-chip HD. Modulo thresholding adds two more thresholds to create *stripes* in which strong bits are required to fall as shown in Fig. 13(b). Striping ensures that the delay difference between 2 paths of an arbitrary pairing does not exceed the within-die variation levels. We found a stripe height between 10 and 15 is effective at filtering out these biased path pairings.

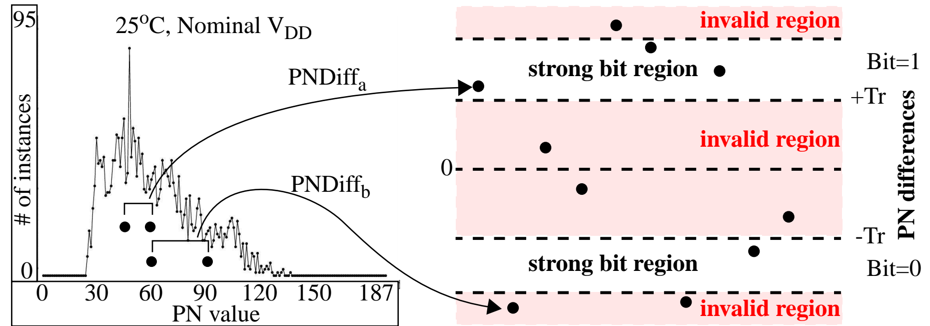

Fig. 13. (a) Enrollment delay distribution and (b) Modulo Enrollment illustration.

In our experiments, we experimented with a variety of TV noise thresholds between 0 and 15 as a means of evaluating the best choices for stripe height and spatial redundancy. We required that 1) no bit flips occurred for any chip at any TV corner, i.e., the intra-chip HD is 0%, 2) the inter-chip HD is close to the ideal of 50% and 3) the bitstrings score well on the NIST statistical tests. We found these conditions could be met using any one of several different combinations of the TV noise threshold, stripe height and spatial redundancy parameter values. Moreover, we found that the size of the bitstrings produced by the chips remained relatively constant under each of the parameter combinations that met the 3 requirements above. In particular, bitstring size varied between 55 K and 65 K bits. This yields an overhead per bit of approx. 50, i.e., we need to parse 50 path pairings for every valid bit generated. As mentioned above, this *valid bitstring* overhead adds to the *path bitstring* overhead. So the Helper Data for a 256-bit bitstring would be approx. 3 Kbits (assumes 8.5% of the paths are stable) + 13 Kbits (256 * 50) = 16 Kbits or 2.0 KBytes. Another interesting result is that at a spatial redundancy setting of 17, we were able to make the threshold 0 and meet the 3 requirements.

Fig. 14 gives the inter-chip hamming distance (HD) distribution for UNMD. The individual HDs are computed pair-wise using the bitstrings of length 64,948 bits from 50 chips, yielding 1,225 HDs. The ideal (50%) average HD value is 32,474 bits. The value computed is 32,478, which expressed as a percentage is 50.004%. The TV noise threshold was set such that no bit flips occurred in any of the 50 chips at any of the 8 regeneration TV corners, so the intra-chip HD is 0%. However, with thresholding turned off, the actual underlying intra-chip HD is computed to be 2.6%. The size of the bitstrings before thresholding is 3,171,421. The average number of bits that survive the thresholding is approx. 2.5% (the bitstrings used for Fig. 14 are only 2% because we truncated the bitstrings for all chips to the smallest bitstring produced by one of the chips). These bitstrings were also subjected to the NIST statistical test suite using the default level of significance, i.e., $\alpha = 0.01$ [17]. The bitstrings passed the all tests applicable to bitstrings of this size, including frequency, block frequency, cumulative sums, runs, longest run, rank, FFT, nonoverlapping template, approximate entropy and serial.

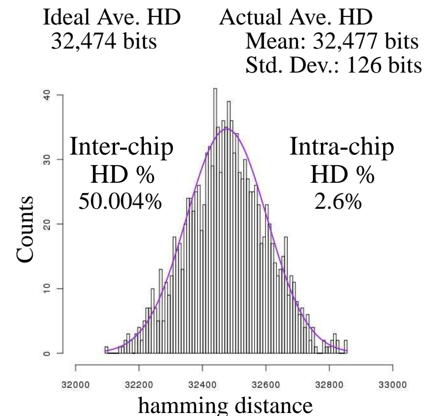

Fig. 14. Inter-chip hamming distance histogram obtained from the UNMD method.

4.5.3 Dual-PN Count (DPNC) Bitstring Generation Method

A third technique called the DPNC method provides several trade-offs when compared to UNM and UNMD techniques. DPNC is the most expensive method in terms of Helper Data bits required per secret bit but is likely to be the most secure with respect to reverse-engineering attacks. The large number of Helper Data bits effectively restricts the size of the secret bitstring to 256 bits or smaller from a practical perspective.

The illustration in Fig. 15 show the basic concepts of the DPNC method. Note that the version described here differs from our previous work [10-11] and is more efficient in terms of entropy extraction. Fig. 15(a) shows the binning process used by DPNC. A segment of the range of PNs between 0 and 41 are listed along the top and are partitioned into 5 groups. The PNs in the groups labeled ‘low bin’ and ‘high bin’ represent valid PNs. PNs in the remaining 3 regions are invalid and are discarded when they appear in the sequence. These regions represent a safety margin between the low and high bins and account for uncompensated shifts in the PNs that occur during regeneration because of TV noise. As indicated earlier, consecutive PNs represent a difference in path delay of approx 53.5 ps so the entire span from 0 to 41 represents approx 2.2 ns of delay variation. The difference between the two valid bins, however, is between 0.27 and 1.12 ns (in PNs, between 5 and 21), which is within the within-die variation range from Fig.10. Fig. 15(b) shows the modulus operation. Using a modulus of 42 causes PNs larger than 41 to wrap back around into the 0 to 41 region. This effectively maps the delays of longer paths into the short path region, while preserving the within-die variations in these longer paths.

Fig. 15(c) shows the DPNC method applied to an example sequence of PNs during enrollment. DPNC requires and odd number of copies, 1, 3, 5, 7, etc. copies of low bin or high bin PN to be found in the sequence before actually generating a secret bit. Similar to spatial redundancy, this scheme adds resiliency to bit flip errors which can occur when PNs move into the opposite bin during regeneration because of TV noise or jumps. The bin of the first valid PN in the sequence determines which bit value, ‘0’ or ‘1’, will be generated.

For example, the modPN given in column 1 in Fig. 15(c) is 13 and is therefore valid and in the low bin so the algorithm searches for 2 more copies of valid modPNs that also fall in the low bin. The modPN in column 2 is valid but falls in the high bin according to Fig. 15(a) and therefore is marked invalid and skipped (header given as ‘FM’ means ‘failed to match’). The modPN in column 3 is 11, a valid low bin

value and therefore represents the second copy. Columns 4, 5 and 6 are valid but fail to match. Columns 7, 8 and 9 fall in the safety regions of Fig. 15(a) and are marked invalid (header given as 'OB' means 'outside the bins'). Column 10 contains a valid low bin PN and represents the third copy. Therefore, a secret bit with value 0 is generated as shown along the bottom of the figure. Regeneration is carried out by reading the valid bitstring to determine which PNs to inspect. Two counters count the number of low bin and high bin values and a bit is generated based on majority vote every time an odd number of valid PNs are parsed.

We applied the DPNC method to our chip data using the parameters given in the example, i.e., modulus of 42 and a bin width of 16 but increased the redundancy from 3 to 7 as a means of meeting the 3 requirements mentioned for UNMD, namely, 1) no bit flips, 2) near 50% inter-chip HD and 3) good NIST results. The average size of the secret bitstrings is 157 bits. The smallest size used in the following statistical results is 148 bits. Intra-chip HD is 0%, inter-chip HD is 49.96% and the bitstrings passed all applicable NIST tests including frequency, block frequency, cumulative sums, runs, longest run and serial.

The public data size for DPNC is 31,236 bits, of which we obtain 2,519 stable PNs and can generate bitstrings of length 157 on average. Therefore, we must test approx. 200 paths to generate each secret bit under DPNC. A bitstring of size 256 requires 51 Kbits or 6.4 KBytes. Clearly, DPNC is the most expensive method with respect to Helper Data. However, the modulus operation makes a simulation-based attack, as we described earlier for UNM, useless because only the high frequency behavior of the path delays are preserved in the modPNs.

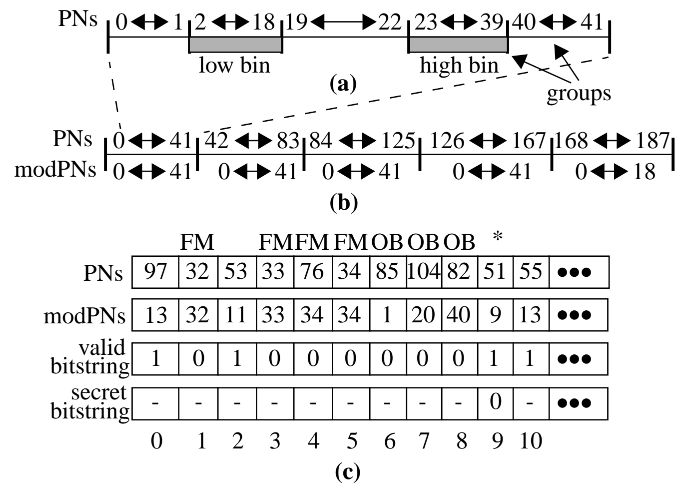

Fig. 15. Illustration of DPNC Method.

5 Conclusions

In this paper, we describe a hardware-embedded delay PUF called HELP which leverages path stability characteristics and within-die delay variations as sources of entropy in core logic macros. We implemented HELP in a floating point unit and fabricated multiple copies in a 90 nm test chip. Three bitstring generation methods are investigated called UNM, UNMD and DPNC. While the statistical characteristics of UNM were substandard, we obtained excellent results from UNMD and DPNC. The poor results from UNM demonstrate that it is not possible in some macros to base the entropy source entirely on path stability. Although aging is not studied in this work, we expect NBTI and HCI to work against the long term stability of the HELP PUF, as is true for all delay-based PUFs.

References

[1]B. Gassend, D. Clarke, M. van Dijk, S. Devadas, "Controlled Physical Random Functions", Conference on Computer Security Applications, 2002, pp. 149-160.

[2]K. Lofstrom, W. R. Daasch, D. Taylor, "IC Identification Circuits using Device Mismatch", ISSCC, 2000, pp. 372-373.

[3]E. Simpson and P. Schaumont, "Offline Hardware/Software Authentication for Reconfigurable Platforms", *CHES*, Volume 4249, Oct., 2006, pp. 10-13.

[4]E. Ozturk, G. Hammouri, B. Sunar, "Physical Unclonable Function with Tristate Buffers", *Symposium on Circuits and Systems*, 2008, pp. 3194-3197.

[5]G. Qu and C. Yin, "Temperature-Aware Cooperative Ring Oscillator PUF", *HOST*, 2009, pp. 36-42.

[6]A. Maiti, J. Casarona, L. McHale, R. Schaumont, "A Large Scale Characterization of RO-PUF", *HOST*, 2010, pp. 94-99.

[7]C. Costea, F. Bernard, V. Fischer, R. Fouquet, "Analysis and Enhancement of Ring Oscillators Based Physical Unclonable Functions in FPGAs", *Conference on Reconfigurable Computing and FPGAs,* 2010, pp. 262-267.

[8]M. Majzoobi, F. Koushanfar, S. Devadas, "FPGA PUF using Programmable Delay Lines", *Workshop on Information Forensics and Security*, 2010, pp. 1-6.

[9]C. Qingqing, G. Csaba, P. Lugli, U. Schlichtmann, U. Ruhrmair, "The Bistable Ring PUF: A New Architecture for Strong Physical Unclonable Functions", *HOST*, 2011, pp. 134-141.

[10]J. Aarestad, P. Ortiz, D. Acharyya and J. Plusquellic, "HELP: A Hardware-Embedded Delay-Based PUF", *IEEE Design and Test of Computers*, Vol. 30, Issue: 2, Mar., 2013. pp. 17-25.

[11]J. Aarestad, J. Plusquellic, D. Acharyya, Error-Tolerant Bit Generation Techniques for Use with a Hardware-Embedded Path Delay PUF, *HOST*, 2013, pp. 151-158.

[12]J. Guajardo, S. S. Kumar, G.-J. Schrijen, P. Tuyls, "Physical Unclonable Functions and Public Key Crypto for FPGA IP Protection", *Conference on Field Programmable Logic and Applications*, 2007, pp. 189-195.

[13]D. Ganta, V. Vivekraja, K. Priya, L. Nazhandali, "A Highly Stable Leakage-Based Silicon Physical Unclonable Functions", *Conference on VLSI Design*, 2011, pp. 135-140.

[14]J. Ju, R. Chakraborty, C. Lamech and J. Plusquellic, "Stability Analysis of a Physical Unclonable Function based on Metal Resistance Variations, *HOST*, 2013, pp. 143-150.

[15]W. Che, D. Ismari, J. Plusquellic and S. Bhunia, "A Non-Volatile Memory-based Physically Unclonable Function without Helper Data", *ICCAD*, 2014.

[16]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.

[17]NIST: Computer Security Division, Statistical Tests, http://csrc.nist.gov/groups/ST/toolkit/rng/stats_tests.html

[18]M. L. Baseline and V. D. Agrawal, "Essentials of Electronic Testing, for Digital, Memory and Mixed-Signal VLSI Circuits," Kluwer Academic Publishers, 2000, ISBN: 0-7923-799-1-8.

[19]http://en.wikipedia.org/wiki/Regression_analysis