The Glitch PUF: A New Delay-PUF Architecture Exploiting Glitch Shapes

Daisuke Suzuki[1,2] and Koichi Shimizu[1]

[1] Information Technology R&D Center, Mitsubishi Electric Corporation
{Suzuki.Daisuke@bx,Shimizu.Koichi@ea}
.MitsubishiElectric.co.jp
[2] Graduate School of Environmental and Information Sciences,
Yokohama National University

Abstract. In this paper we propose a new Delay-PUF architecture that is expected to solve the current problem of Delay-PUF that it is easy to predict the relation between delay information and generated information. Our architecture exploits glitches that behave non-linearly from delay variation between gates and the characteristic of pulse propagation of each gate. We call this architecture Glitch PUF. In this paper, we present a concrete structure of Glitch PUF. We then show the evaluation results on the randomness and statistical properties of Glitch PUF. In addition, we present a simple scheme to evaluate Delay-PUFs by simulation at the design stage. We show the consistency of the evaluation results for real chips and those by simulation for Glitch PUF.

1 Introduction

1.1 Background

High-level security needs such as in financial transactions and Digital Rights Management (DRM) have widened the use of security chips as represented by smart cards and Trusted Platform Modules (TPMs). Security chips provide not only a variety of cryptographic functions but tampering countermeasures, which are mechanisms to protect sensitive information stored within the chips from physical attacks. Examples of tampering countermeasures include mounting sensors or mesh shielding within a chip.

Physical Unclonable Function (PUF) [1] is a technique in relation to tampering countermeasures which has been attracting wider attention in recent years. PUFs are designed to return responses to given challenges according to physical characteristics that are innately possessed by each artificial object such as an LSI. It is arguably difficult to clone an artificial object from the fact that its characteristics originate from manufacturing variation.

With the help of Fuzzy Extractor [2], which is a technique to extract stable secret information from noisy characteristics, it is even possible to generate device unique keys that are difficult to copy. The key information is resistant to analysis that directly reads data inside a chip by breaking it open, because the information does not need storing in nonvolatile memory to be reproducible.

S. Mangard and F.-X. Standaert (Eds.): CHES 2010, LNCS 6225, pp. 366–382, 2010.

PUFs are also advantageous in that they are feasible on general-purpose LSI such as FPGA and ASIC. There are many active research works on methods of PUF realization and generation of device unique keys [4–12].

SRAM-PUF is recognized as one of the most feasible and secure PUFs thus far because there have already been implementations of error correcting codes and universal hash functions optimized for it, which are needed for Fuzzy Extractor. It is, however, difficult to evaluate the information entropy and error rate of SRAM-PUF on ASIC chips before production because it, as well as Butterfly-PUF [8], exploits the metastable state of memory cells on power activation, where only a behavior model is available for the characteristics of the cells. The error rate is particularly changeable according to the process. In fact, Ref. [13] reports a much higher rate of error than that reported by the proposers [5], which implies the possibility that the error rate of SRAM-PUF changes on different target devices. On the other hand, the evaluation is possible on devices that are available before production such as FPGA.

As for Delay-PUF, security issues have been reported. It is shown that a machine learning attack can predict challenge-response pairs after a decent number of pairs are collected by self-evaluation [6]. Furthermore, although there have been proposed countermeasures such as Feed-Forward Arbiter PUF, which installs non-linear operations, and XOR-PUF, which is comprised of multiple Arbiter PUFs, it is shown that machine learning attacks are still applicable to those [14]. These issues originate from the simplicity of the circuit structure of Delay-PUF.

At the same time, however, Delay-PUF is advantageous in that delay information utilized by it has affinity with logic simulation, which is performed at the design stage. That enables to evaluate the amount of information of the PUF at an earlier stage of design process. At least, chip vendors must possess information about the delay variation since they need to embed the delay information in a cell library when they develop it.

On the other hand, Statistical Static Timing Analysis (SSTA), which is a design method for variation, has been intensely studied [15] now that increasing manufacturing variation prevents performance improvement as the process miniaturizes. SSTA is adopted by standard Electronic Design Automation (EDA) tools as PrimeTime. It is anticipated from these facts that logic circuit designers will be able to access information about delay variation in a near future.

1.2 Our Contributions

We propose a new Delay-PUF architecture, which is expected to solve the easiness of predicting the relation between delay information and generated information. The proposed architecture exploits glitch waveforms that behave non-linearly from delay variation between gates. We thus call this architecture *Glitch PUF*.

In this paper, we show a concrete structure of Glitch PUF. We also present the results of the evaluation on randomness and statistical properties of Glitch PUF performed on FPGA.

As the second contribution, we present a simple scheme to evaluate the characteristics of Glitch PUF with simulation at the design stage. We show the consistency between the results using the scheme, and those for real devices.

2 Simulating Behavior of Delay-PUFs

In this section, we discuss a concrete methodology to evaluate randomness and statistical properties of Delay-PUFs by simulation. The goal of this simulation is to evaluate the randomness and error rate of a Delay-PUF at its design stage. The reason that Delay-PUF circuits of the same design behave differently on each individual chip is that transistors hold characteristic variation (variation of threshold voltages V_{th}, to be concrete). The occurrence of errors even on the same chip results from the change of operating environment (static/dynamic IR-drop, change of temperature, etc.). By attributing these factors to the variation of gate delays, we attempt to realize the evaluation of the randomness and error rate.

The evaluation flow, shown in Fig. 1, is basically the same as an ordinary circuit design and timing evaluation. It is different in that delay variation is reflected before simulation.

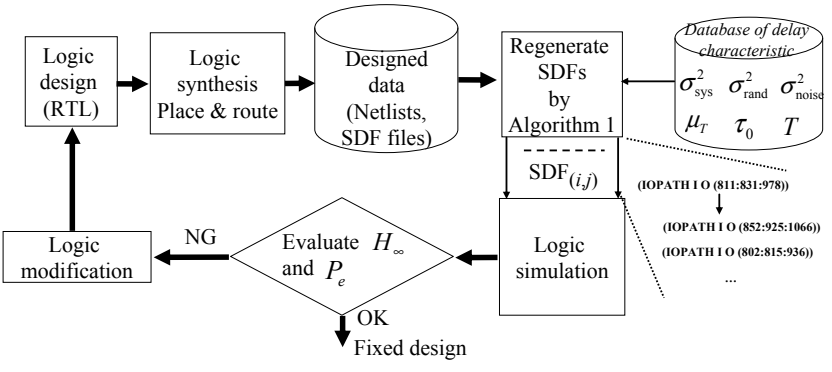

Fig. 1. PUF evaluation flow by simulation

A Standard Delay Format (SDF) [16] in Fig. 1 is a file that defines representative delays for a target device, and used for delay analysis for a circuit. It is thus possible to evaluate the operating delays of the device under corner conditions using the delay values that corresponds to several operating conditions of the circuit. However, it is not possible to evaluate PUFs with SDFs as they are because the delay values are fixed while PUFs assume delay variation.

In order to reflect individual difference and environmental change, we perform simulation with a large number of SDFs varied from the original SDF according to the previously-extracted characteristics and distributions of process, supply voltage and temperature (PVT) of a device.

Process variation is generally classified into systematic variation and random variation [17]. Systematic variation is correlated with location in a wafer or a chip. It is represented by the performance difference between chips such as the speed grade of FPGA. Random variation is not related with spatial location of transistors. It is known to result from the fluctuation of the concentration of impurities. Environmental change is parameterized representatively by voltage and temperature. These parameters are evaluated for TEG chips on a startup of LSI fabrication.

On the contrary, the information about PVT variation on FPGA is not disclosed. We hence try to extract the parameters by observing the delays in a chain of inverters under various layouts and environments as in [18] on 16 FPGAs. The parameters are as follows.

- Systematic delay variation between chips: σ^2_{sys}
- Random delay variation within chips: σ^2_{rand}
- Environmental random delay variation such as from dynamic IR-drop: σ^2_{noise}
- Average fraction of designed delays under 0 °C: τ_0
- Delay temperature coefficient: μ_T

The following assumptions are made to calculate each parameter from measured delays.

(1) Systematic delay variation within chips can be ignored.
(2) The distributions are normal (with variances σ^2_{sys}, σ^2_{rand}, and σ^2_{noise}).
(3) σ^2_{rand}, σ^2_{noise}, and μ_T are constant for all chips.

Note here that all the parameters are represented as fractions of designed delay values in a SDF. It is then possible to simulate individual difference and environmental change based on the delays that EDA tools output reflecting gate depths, numbers of fanouts and layout difference.

Designed delay values $d_1, \cdots, d_{\text{MAX}_{\text{NodeNum}}}$ for each node defined in a SDF are changed by the following formula (Algorithm 1), where sampling r from a distribution $N(0, \sigma^2)$ is denoted as $r \leftarrow N(0, \sigma^2)$.

3 Glitch PUFs

In this section, we explain the architecture of the proposed PUF, which exploits glitch waveforms. It is thus called Glitch PUF.

3.1 Basic Idea

We consider to simulate the behavior of a device at early design stages according to the characteristics of the device. The goal of this simulation is to estimate the amount of information of a PUF, especially its lower limit, without need to evaluate a large volume of real LSIs. As stated earlier, it is delay information that is most compatible with simulation at earlier stages of all characteristics.

Algorithm 1. Regeneration of SDFs with Individual and Environmental Difference

Setting: · $\mathrm{MAX}_{\mathrm{NodeNum}}$ nodes are included in the original SDF file.
· $\mathrm{MAX}_{\mathrm{ChipNum}}$ chips are simulated.
· The response data for each chip is regenerated $\mathrm{MAX}_{\mathrm{RepNum}}$ times to evaluate the error rate.
· T °C is the operating temperature.

Input: $T, (d_1, \cdots, d_{\mathrm{MAX}_{\mathrm{NodeNum}}})$

Output: $\mathrm{SDF}_{(i,j)}$, $0 \leq i \leq \mathrm{MAX}_{\mathrm{ChipNum}}$, $0 \leq j \leq \mathrm{MAX}_{\mathrm{RepNum}}$.

for $i = 1$ to $\mathrm{MAX}_{\mathrm{ChipNum}}$ **do**
  $r_{\mathrm{sys}} \leftarrow N(0, \sigma^2_{\mathrm{sys}})$
  for $j = 1$ to $\mathrm{MAX}_{\mathrm{RecNum}}$ **do**
    for $k = 1$ to $\mathrm{MAX}_{\mathrm{NodeNum}}$ **do**
      $r_{\mathrm{rand}} \leftarrow N(0, \sigma^2_{\mathrm{rand}})$
      $r_{\mathrm{noise}} \leftarrow N(0, \sigma^2_{\mathrm{noise}})$
      $d'_k := ((1+\mu_T \cdot T)(\tau_0 + r_{\mathrm{sys}} + r_{\mathrm{rand}}) + r_{\mathrm{noise}}) \cdot d_k$
    end for
    $\mathrm{WriteSDF}_{(i,j)}(d'_1, \cdots, d'_{\mathrm{MAX}_{\mathrm{NodeNum}}})$
  end for
end for

Then it is probable to be able to evaluate the amount of information of a PUF within the current scheme of logic circuit design, if the delay variation among devices is closely connected to the change of reponse of the PUF.

We consider possible behavioral difference of the same logic circuits with different delays. Examples of such behaviors are shown in Fig. 2. Fig. 2-(a) shows a basic one that there is a time difference between output changes from an input change. The time from an input change to an output change is, however, difficult to be exploited as a device unique key because it depends not only on the variation of gate delays inherent from manufacturing but also largely on the operating temperature and voltage. On reflection, Arbiter PUF by Lin et al. exploits the time difference between two signals to ensure stability against such environmental changes as shown in Fig. 2-(b). But it is known about Arbiter PUF that it is possible to predict challenge-response pairs (CRPs) by machine learning if a sufficient number of CRPs have been collected. Feed-Forward Arbiter PUF, which introduces non-linear operations as a countermeasure, is also possible to be attacked by machine learning [14].

The examples thus far describes behaviors from a delay difference for very simple logic circuits. From here, we discuss more general circuits such as Fig. 2-(c) that perform AND and XOR to multiple inputs. In this kind of circuit there occurs a transient state of an output signal called a glitch from the delay difference between input signals, unless a particular condition holds. In the example in Fig. 2-(c), in case that input signals x_1, x_2, x_3 all change from 0 to 1, there is a convex glitch at the XOR output from the difference of transition time between x_1 and x_2. The glitch propagates to the AND output only if the transition of

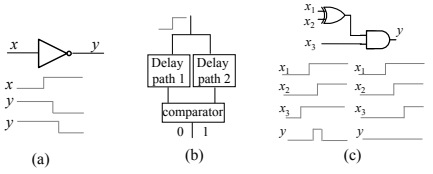

Fig. 2. Circuit behaviors for different delay values

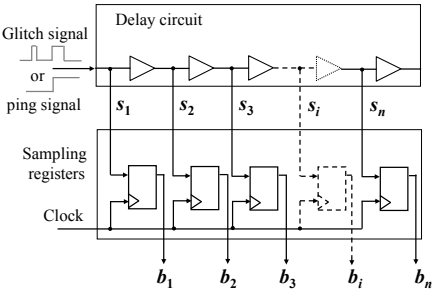

Fig. 3. Sampling circuit

the input signal x_3 reaches the AND gate faster than the glitch. If not on the contrary, the glitch does not propagate to the output, in which case the output remains unchanged. Furthermore, even if x_3 is faster, the PATHPULSE [16] of the AND gate might prevent a narrow glitch from propagating. Notice here that for sufficiently wide glitches, their shapes are determined by the relation of delays, not by the absolute values of the delays. It is then anticipated that shapes of glitches are kept unchanged if the operating environment changes, like Arbiter PUF.

Now our attention is focused on glitches, which can take various shapes according to the order relation of delays between the inputs of each gate that consists in a logic circuit. We discuss a means of applying them to construct a PUF from here.

3.2 Overall Sequence

First of all, we describe a whole sequence of Glitch PUF. Glitch PUF consists largely of the three steps below.

STEP 1 Data input to a random logic
STEP 2 Acquisition of glitch waveforms at the output
STEP 3 Conversion of the waveforms into response bits

In the example of Fig. 2-(c), STEP 1 means causing changes to the inputs x_1, x_2, x_3. The accompanying glitch waveform at the output y is acquired as an n-bit data. The data is then transformed into a one-bit data r according to its shape. Changing the input in STEP 1 and iterating the steps, a bit sequence R is acquired as a response data R.

Each subsection below describes the details of necessary techniques to realize the operations from STEP 1 to STEP 3.

3.3 Acquisition of Glitch Waveforms

As described in Section 2.1, we attempt to construct a PUF using glitches, which can take various shapes according to delay variation. The issue here is how to

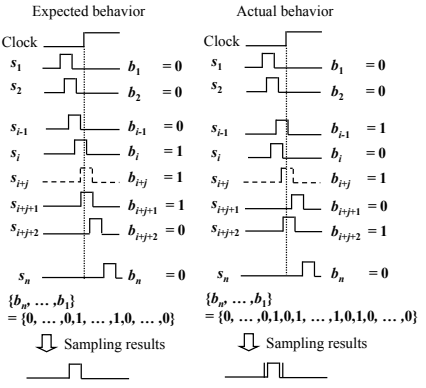

Fig. 4. Error in glitch acquisition

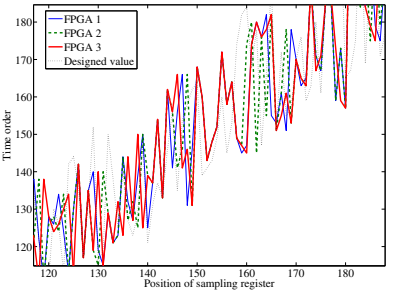

Fig. 5. Time order of sampling registers

accurately acquire the shape of a pulse signal that happens only for a tiny period of time. At the same time, the acquisition process must be realized as a digital circuit for the goal of this paper.

The phase-shift method is one of the general solutions to the issue, where multiple clocks with different phases are prepared to sample a tiny pulse. The sampling accuracy is heightend as the number of different phased clocks is increased. The method is, however, not practical since it needs too many clock lines. Particularly in FPGA, there is a limited number of global clock lines with little jitter, from several to several dozen. Although the number of clock lines can be reduced by introducing time-division, the speed of acquisition decreases then. We hence adopt a method where the target data is shifted by a tiny period of time, and sampled by the same clock. Fig. 3 shows a sampling circuit of this method. We call this operation *glitch acquisition* hereafter. Note here that since the sampling period must be short for acquisition accuracy, it is required to reduce the delay deference between signals loaded into flip-flops (FFs) as much as possible by decreasing the buffer depths between signals, or using other elements with shorter delays in Fig. 3. As the delay difference decreases, though, clock jitter between FFs and wire or gate delay cannot be ignored. In this case, the orders of sampling positions of FFs, and their actual delays do not match as in Fig. 4, thereby permuting the time order of the sampled data. It probably becomes impossible to recover the glitch shape accurately. This problem also occurs for the aforementioned phase-shifting of clock.

We thus introduce a preprocessing shown below before performing glitch acquisition in order to determine the time order of the sampling result. A signal wire is added to generate a simple rising edge, called a *ping signal* hereafter. First, a ping signal is input to the sampling circuit in Fig. 3 and sampled. Then, each FF latches 1 or 0 if the ping signal reaches it before or after the clock, respectively. On the other hand, there is a variable delay circuit in the clock line,

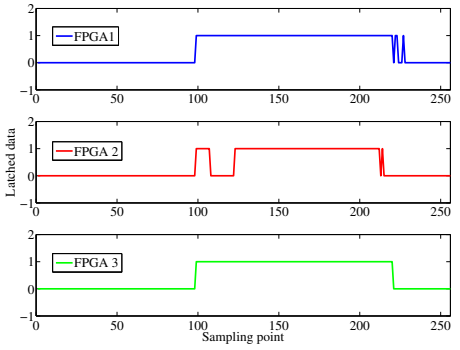

Fig. 6. With jitter correction

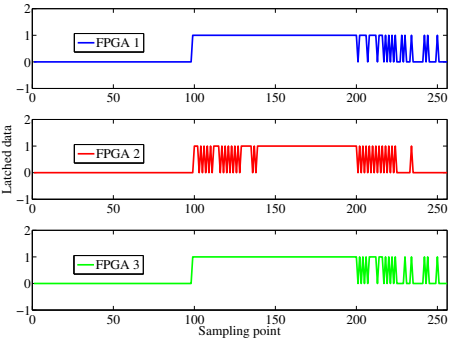

Fig. 7. Without jitter correction

with which the above process is performed multiple times with different clock delays. The number of latching 1 is thereby counted for each FF. Lastly, the time order of the samplings is determined according to the order of the numbers. An example time order is shown in Fig. 5. The glitch shape is recovered after sorting the sampled data according to this order.

From here, the above-mentioned preprocessing is called *jitter correction* and the sorting according to the result of the jitter correction is called *sorting*. The results of glitch acquisition with and without jitter correction are shown in Fig. 6 and Fig. 7 respectively.

3.4 Conversion to Response

One-bit value is converted from the glitch waveforms acquired as digital data by the above sampling method. We describe a means to convert the parity of the number of rising edges in a glitch waveform. The parity can be detected with differential and addition processing implemented in hardware or software. This detection is called *shape judgment*.

It is still difficult to acquire the time order of FFs completely even though the aforementioned preprocessing is performed. When the sorted time order is different from the actual order of the circuit, a glitch waveform like Fig. 4 is acquired, where there are narrow pulses near the edges. This kind of phenomenon is an unstable behavior occurring when the clock delays between FFs are close. As a result, the shape judgment can be different for each trial of glitch acquisition. In addition, an extremely narrow pulse can cause the same phenomenon due to the PATHPULSE mentioned previously.

We decide to perform a processing as shown in Fig. 8 to ignore pulses with widths less than a threshold w. This processing is called *filtering* hereafter.

3.5 Reliability Enhancement

In order to improve the error rate of the shape judgment, we utilize the feature that the same processing can be performed multiple times. That is, the final

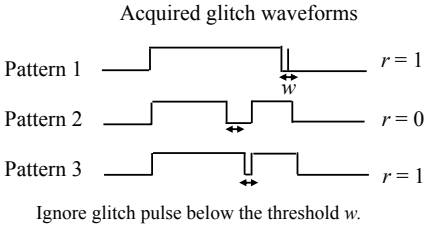

Fig. 8. Conversion from glitch waveform to response with filtering

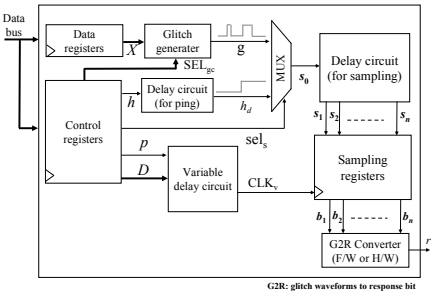

Fig. 9. Whole structure of Glitch PUF

output is determined by majority after multiple trials of shape judgment are performed for the same input transition. In particular, when the initial key is generated such as by Generate (Gen) in Fuzzy Extractor [2], only such inputs as acquire the same outputs for M iterations are used. In this case, performing shape judgment M times for each of N input changes generates an N-bit mask as well as an N-bit response. Value 1 of a mask bit means that the bit position is used for key generation, and vice versa. The mask is output as part of Helper Data.

In methods such as suggested in Ref. [12], a probability distribution of errors is output as Helper Data when performing Gen and soft-decision is performed in the process of Reproduce (Rep) [2]. The amount of information is preserved although the error distribution is made public. On the contrary, the amount of information is reduced by the masking process. In Glitch PUF, bit positions with high error rates are determined for each chip and the number of them is small. We therefore choose to mask them and reduce the size of the necessary error correcting circuit.

3.6 The Architecture

Fig. 9 illustrates the circuit architecture of Glitch PUF. Glitch PUF consists mainly of control registers, data registers, a glitch generator, a sampling circuit, and two kinds of delay circuit. The control registers in Fig. 9 store the control parameters listed below.

- $\mathrm{SEL_{gc}}$: Selection signal to glitch generator ($\log u$ bits)
- $\mathrm{sel_s}$: Input selection signal to sampling circuit (1 bit)
- h: Ping signal (1 bit)
- D: Delay specifier signal to variable delay circuit ($q + q'$ bits)
- p: Trigger signal (1 bit)

The data registers store the data X (u-bit) input into the glitch generator. The glitch generator is comprised of a combinational circuit and a v-1 selector, where the circuit performs $Y = f(X)$ defined for X in Fig. 10 and the selector selects

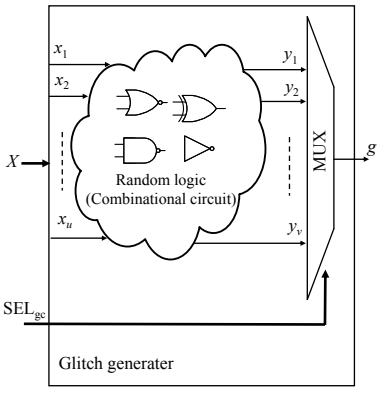

Fig. 10. Glitch generater

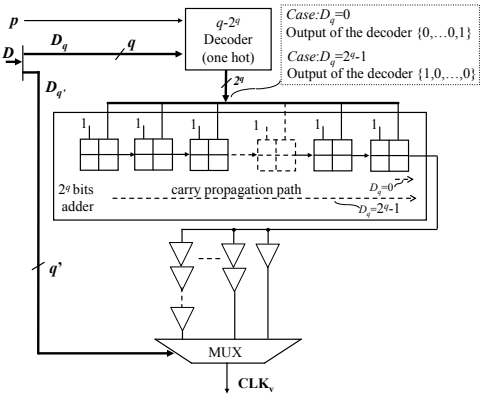

Fig. 11. Variable delay circuit

one bit out of v bits of Y according to a selection signal $\mathrm{SEL_{gc}}$. The delay circuit for the ping signal consists of a buffer chain, thereby outputting h_d, a delayed signal of h. The depth of the chain is determined at the design stage by simulation evaluating the occurrence timing of the glitch signals generated in the glitch generator. We describe the details in the next section. The sampling circuit, as discussed in Section 3.3, is constructed of a buffer chain and FFs shown in Fig. 3. It is noteworthy that when implementing a Glitch PUF on FPGA, the sampling resolution can be heightened by utilizing carry paths of addition circuits as a buffer chain rather than implementing the chain in Look-up tables (LUTs). The variable delay circuit is also implemented with carry paths to minimize the step size of delay by which the delay can be varied. At the same time, however, the range of the variable delay must be wider than that for sampling. Hence, the circuit requires large area if it is all constructed of carry paths. The issue can be avoided by combining delay circuits on carry paths and LUTs as in Fig. 11, thereby keeping a wide variable range and high resolution at the same time.

In this paper, the process until the sampling is implemented on hardware and the rest is on firmware in order to observe the behavior of the generated glitches.

3.7 Adjustment of the Design Parameter

In order to realize efficient glitch acquisition, parameters need to be adjusted for each circuit in Fig. 9 at the design stage. The parameters are as the following.

- n: The number of FFs in the sampling circuit
- $\mathrm{delay_s}$: The delay value of the buffers inserted between the signals of the sampling circuit
- $\mathrm{range_s}$: The sampling range of the sampling circuit
- $\mathrm{range_g}$: The range of glitch occurrence in the glitch generator

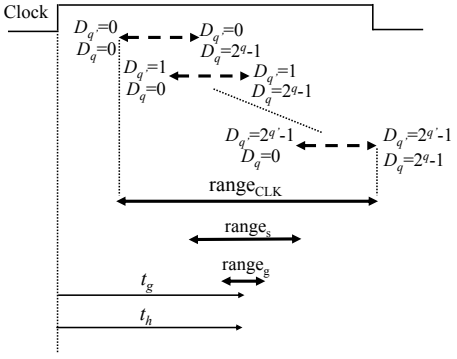

Fig. 12. Timing conditions for the designing parameters

· range$_{CLK}$: The variation range of the variable delay circuit
· t_g: The time of the central value in the range of glitch occurrence
· t_h: The time of rising of the delayed ping signal h_d

Fig. 12 illustrates the relationship between each parameter. n and delay$_s$ are related to the sampling range and resolution. range$_s$ is about $n \cdot$ delay$_s$. To acquire glitch shapes, it must hold that range$_g$ < range$_s$ < range$_{CLK}$. We discuss a design procedure to realize the relation in what follows.

At first, the time range range$_g$ where glitches can occur at the input of the sampling circuit, and the occurrence timing t_g are estimated by logic simulation with delay information. That is performed when the logic of the glitch generator is fixed. Second, n and delay$_s$ are determined such that the equation range$_g$ < range$_s$ holds. Here, the sampling resolution can be heightend by selecting a cell from the target platform as a buffer such that delay$_s$ is as small as possible. As a result, it is n that actually needs determining. The implementation in this paper sets range$_s$ to be more than twice range$_g$ as a design margin. Also, the buffer depth in the variable delay circuit is configured such that range$_{CLK}$ is around twice range$_s$.

Next, the buffer depth in the delay circuit for the ping signal is configured such that $t_g \simeq t_h$ for the previously determined t_g. The configuration is not only used to calculate the time order of the sampling results. It is also to determine the delay specifier signal Dg in the variable delay circuit, which is for acquiring glitch shapes. The details are as follows. Sampling the ping signal is performed with D being incremented from its minimum to maximum values. Dg is set to be the D when a certain FF around the center of the sampling results latches 1 for the first time. Glitch waveforms can thereby be sampled around the center of the sampling range since the glitch occurrence range is configured to be in the same range as the ping delay. However, range$_s$ and range$_{CLK}$ must have margins, like twice/half something as stated earlier, because it is generally difficult to accurately configure the delay between signals.

Table 1. Specification of experimental environment

Implementation environment	
Logic synthesis, P&R	Xilinx Platform Studio 10.1.03i
Simulator	NC-Verilog
Target FPGA	Xilinx XC3S400A-4FTG256C (16 boards)
$\mathrm{MAX}_{\mathrm{ChipNum}}$	1000
$\mathrm{MAX}_{\mathrm{RepNum}}$	1000
Number of bits of generated responses N	2048
Specification of Glitch PUF	
Glitch generator	AES SubByte (composite field)
Design parameter (n, u, v, q, q')	(256, 8, 8, 8, 2)
Filtering parameter w	2
Reliability Enhancement parameter M	10
SLICEs used	891/3584 (Whole SoC 3186/3584)
Operating frequency	50 MHz

Table 2. Delay characteristics

Systematic delay variation $\sigma^2_{\mathrm{sys}}(\%^2)$	2.5037
Random delay variation $\sigma^2_{\mathrm{rand}}(\%^2)$	5.3091
Environmental random delay variation $\sigma^2_{\mathrm{noise}}(\%^2)$	0.0310
Average fraction of designed delays τ_0 (%)	56.9727
Delay temperature coefficient μ_T (%/°C)	0.1401

4 Experimental Results

This section presents the results of evaluating the randomness and statistical properties for an experimental implementation of Glitch PUF on FPGA. The experiment is performed for Spartan-3A evaluation boards by AVNET that mount one XC3S400A-4FT256, a Xilinx FPGA. 16 boards are used. We build a System-on-a-Chip (SoC), mounting on an FPGA a soft-macro CPU (MicroBlaze), UARTs, and memory controllers as well as a PUF circuit. Table 1 shows the specification of the implementation environment. The process after the shape acquisition mentioned above is performed by firmware on a MicroBlaze mounted on the same FPGA as the circuits are implemented on. AES SubBytes is used as the glitch generator since its logic is complex and circuit structure is well studied by designers of cryptographic hardware. The sampling circuit is implemented with 256 FFs. The variable delay circuit consists of a 256-bit addition circuit, and an LUT-based buffer chain whose depth of LUT can be 4, 8, 12, and 16 with a 4-1 selector.

We perform a basic experiment on delay characteristics described in Section 2 in order to extract the parameters for the same FPGA boards needed for the simulation of PUF. The parameters are shown in Table 2. The parameters are calculated as fractions of corresponding worst-case delays defined in SDF

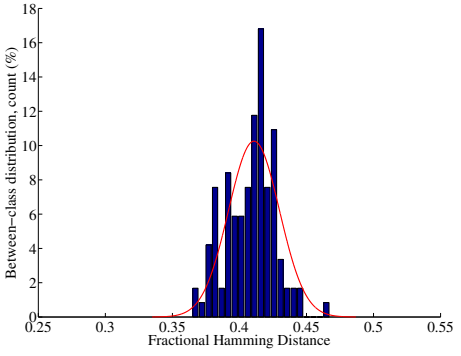

Fig. 13. Hamming distances of response data between FPGAs (actual chips)

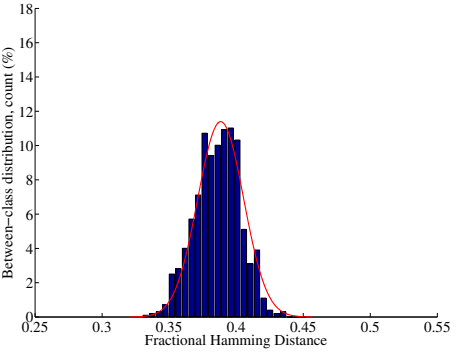

Fig. 14. Hamming distances of response data between FPGAs (simulation)

Table 3. Change of information amount against change of variations

Simulation results (24°C)		
Variation	$H_\infty(R)$	$H_\infty(R\|\text{Mask})$
$(\sigma^2_{\text{sys}}, \sigma^2_{\text{rand}})$	1,043	702
$((2 \cdot \sigma_{\text{sys}})^2, \sigma^2_{\text{rand}})$	1,068	721
$((\frac{1}{2} \cdot \sigma_{\text{sys}})^2, \sigma^2_{\text{rand}})$	1,046	703
$(\sigma^2_{\text{sys}}, (2 \cdot \sigma_{\text{rand}})^2)$	1,167	811
$(\sigma^2_{\text{sys}}, (\frac{1}{2} \cdot \sigma_{\text{rand}})^2)$	828	546
FPGAs	1,156	649

generated by an EDA tool after the layout. Using Table 2, we regenerate a number of SDFs according to Algorithm 1, and evaluate the randomness and statistical properties by simulation. The results are also shown in this section.

4.1 Inter-chip Variation

Fig. 13 is a histogram of Hamming distances between PUF outputs of two different FPGAs out of 16 (i.e. 120 combinations). This evaluation is a general way to show how different responses of chips are. The result shows that about 850 bits out of 2048 bits are different between chips. Fig. 14 shows the result of the same evaluation by simulation. Comparing Figs. 13 and 14, it is seen that the simulation is able to evaluate the randomness of responses generated by real chips.

Table 3 shows the min-entropy of the probability distribution of the response acquired through the experiment, and of the distribution of the masked response described in Section 3.5. Masking reduces the min-entropy of the original distribution since it discards the response bits that are judged to be unstable at Gen. However, the reduction rate is only around 30% for the experimental Glitch

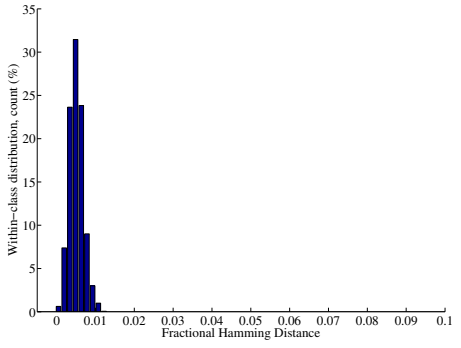

Fig. 15. Hamming distances between response data for the same FPGA (actual chips)

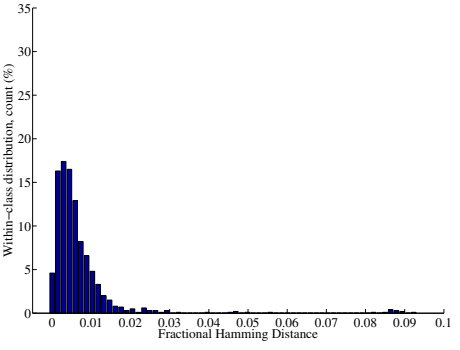

Fig. 16. Hamming distances between response data for the same FPGA (simulation)

PUF, indicating that the amount of information is still sufficient if the unstable bits for each chip are discarded. Table 3 also proves that the min-entropy loss from masking can be evaluated at the design stage by simulation.

It is also noteworthy in Table 3 that there is an interesting relationship between $H_\infty(R)$, $\sigma^2_{\rm sys}$, and $\sigma^2_{\rm rand}$. $H_\infty(R)$ changes sensitively to the change of $\sigma^2_{\rm rand}$ while it does not to the change of $\sigma^2_{\rm sys}$. In other words, Glitch PUF ensures the amount of information of the response data from the random delay variation within chips rather than from the systematic delay variation between chips. The result implies that Glitch PUF can guarantee the randomness for chips on the same wafer as well as for chips on different wafers, or from different lots. In addition, it is arguably possible that the randomness of Glitch PUF further improves for latest devices because random delay variation tends to enlarge as the process miniaturizes.

4.2 Intra-chip Variation

It is desirable for a PUF to stably generate the same response for the same FPGA. Fig. 15 plots the Hamming distances between the initial response and 100 responses measured thereafter, all of which are masked. The measurements are at normal temperature and voltage (24°C,1.20V), and averaged for 16 FPGAs. The mean error rate is around 1.3%. Next, as Fig. 17 shows, the maximum error rate in the rated temperature range is about 6.6 % at 80°C, which is less than the half of 15% assumed in Ref. [11]. In addition it is shown by Figs. 16 and 17 that the change of the error rate with respect to the temperature can be evaluated by simulation with high accuracy.

Next, we discuss the effect of masking. Fig. 18 is a histogram of error rates for each bit of the 2048-bit response data at normal temperature and voltage. It is clear that there are many bits with error rates higher than 0.1 when masking is not performed. At the same time, most of these bits are removed by masking, which correctly treats the response data. Masking is effective for Glitch PUF

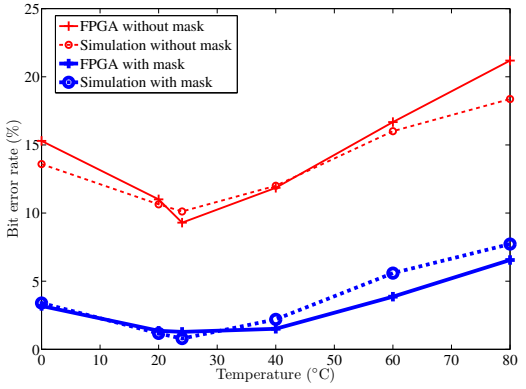

Fig. 17. Change of error rate accompanying temperature change

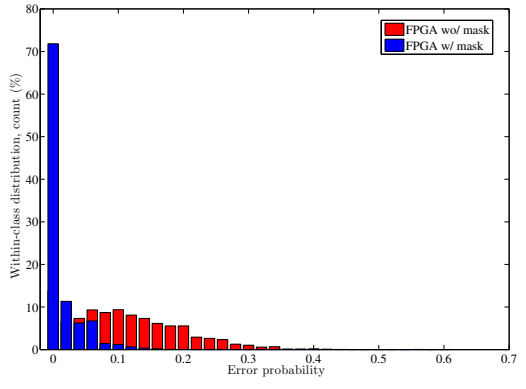

Fig. 18. Reduction of error rate by masking

since it greatly lowers the error rate, although the min-entropy decreases by about 30 % as stated earlier.

4.3 Secrecy Rate

In Ref. [19], the secrecy rate is defined to be $I(R, R')$, the mutual information of the response data at Gen R, and at Rep R'. The average secrecy rate of Glitch PUF is calculated to be 0.26 per bit from the aforementioned experimental results. This is about 1/3 that of SRAM-PUF. At the moment, Glitch PUF is thus inferior to SRAM-PUF in the efficiency to generate secret information. However, delay-PUFs including Glitch PUF have the advantage of being able to evaluate the secrecy rate by logic simulation, the same way as explained in the

previous sections. On the contrary SRAM-PUF requires analog simulation like SPICE to evaluate the same thing.

5 Conclusions

In this paper, we propose Glitch PUF, which is a new Delay-PUF for the purpose of remedying a problem about the previous Delay-PUFs, that is, the easiness to predict the relationship between delay information and generated information. Glitch waveforms occurring at the output of a random logic behave non-linearly from delay variation between gates and the characteristic of pulse propagation of each gate. We present a method to accurately acquire the waveforms and to convert them into response bits. In addition, we prove the feasibility of Glitch PUF by evaluation of the randomness and statistical properties for an FPGA. Furthermore, we show a simple scheme to evaluate the characteristics of Glitch PUF. Using the scheme, we confirm the consistency of the evaluation results for real chips and those by simulation.

Lastly, we list open problems below.

- Construct a glitch generator that brings high amount of information and low error rate .
- Model machine learning attacks to Glitch PUF.
- Construct an error correcting code and universal hash function suitable for Glitch PUF.
- Model logic simulation for voltage change and aging degradation through acceleration test, and evaluate them on real chips

References

1. Pappu, R.S.: Physical One-way Functions. Ph.D. Thesis, M.I.T. (2001), `http://pubs.media.mit.edu/pubs/papers/01.03.pappuphd.powf.pdf`
2. Dodis, Y., Reyzin, M., Smith, A.: Fuzzy Extractors: How to Generate Strong Keys from Biometrics and Other Noisy Data. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 523–540. Springer, Heidelberg (2004)
3. Tuyls, P., Batina, L.: RFID-Tags for Anti-Counterfeiting. In: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 115–131. Springer, Heidelberg (2006)
4. Gassend, B., Clarke, D., van Dijk, M., Devadas, S.: Silicon Physical Random Functions. In: Proc., of the 9th ACM Conference on Computer and Communications Security (CCS 2002), pp. 148–160 (2002)
5. Guajardo, J., Kumar, S.S., S̃chrijen, G.J., Tuyls, P.: FPGA Intrinsic PUFs and Their Use for IP Protection. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 63–80. Springer, Heidelberg (2007)
6. Lee, J.W., Lim, D., Gassend, B., Suh, G.E., van Dijk, M., Devadas, S.: A Technique to Build a Secret Key in Integrated Circuits for Identification and Authentication Applications. In: Proc. of the IEEE VLSI Circuits Symposium, pp. 176–179 (2004)
7. Suh, G.E., Devadas, S.: Physical Unclonable Functions for Device Authentication and Secret Key Generation. In: Proc. of the 44th annual Design Automation Conference (DAC 2007), pp. 9–14 (2007)

8. Kumar, S.S., Guajardo, J., Maes, R., S̃chrijen, G.J., Tuyls, P.: Extended Abstract: The Butterfly PUF: Protecting IP on every FPGA. In: Proc. of the IEEE International Workshop on Hardware-Oriented Security and Trust 2008 (HOST 2008), pp. 67–70 (2008)
9. Majzoobi, M., Koushanfar, F., Potkonjak, M.: Lightweight secure PUFs. In: Proc. of the IEEE/ACM International Conference on Computer-Aided Design (ICCAD 2008), pp. 670–673 (2008)
10. B̈osch, C., Guajardo, J., Sadeghi, A.R., Shokrollahi, J., Tuyls, P.: Efficient Helper Data Key Extractor on FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 181–197. Springer, Heidelberg (2008)
11. Maes, R., Tuyls, P., Verbauwhede, I.: Low-Overhead Implementation of a Soft Decision Helper Data Algorithm for SRAM PUFs. In: Proc. of the 2009 IEEE International Symposium on Information Theory (ISIT 2009), pp. 2101–2105 (2009)
12. Maes, R., Tuyls, P., Verbauwhede, I.: A Soft Decision Helper Data Algorithm for SRAM PUFs. In: Clavier, C., Gaj, K. (eds.) CHES 2009. LNCS, vol. 5747, pp. 332–347. Springer, Heidelberg (2009)
13. Chopra, J., Colopy, R.L.: SRAM Characteristics as Physical Unclonable Functions. Worcester Polytechnic Institute Electric Project Collection (2009), http://www.wpi.edu/Pubs/E-project/Available/E-project-031709-141338/
14. Rührmair, U., Sölter, J., Sehnke, F.: On the Foundations of Physical Unclonable Functions. Cryptology ePrint Archive, 2009/277 (2009)
15. Najm, F.N., Menezes, N.: Statistical Timing Analysis Based on a Timing Yield Model. In: Proc. of the 41st annual Design Automation Conference (DAC 2004), pp. 460–465 (2004)
16. Standard Delay Format Specification version 3.0 (1995), http://www.eda.org/sdf/sdf_3.0.pdf
17. Hiramoto, T., Takeuchi, K., Nisida, A.: Variability of Characterisics in Scaled MOSFETs. J. IEICE 92(6), 416–426 (2009)
18. Berkelaar, M.: Statistical Delay Calculation, a Linear Time Method. In: Proc. of the International Workshop on Timing Analysis (TAU'97), pp. 15–24 (1997)
19. Ignatenko, T., Schrijen, G.J., Skoric, B., Tuyls, P., Willems, F.: Estimating the Secrecy-Rate of Physical Unclonable Functions with the Context-Tree Weighting Method. In: Proc. of the 2006 IEEE International Symposium on Information Theory (ISIT 2006), pp. 499–503 (2006)