A Delay-based PUF Design Using Multiplexer Chains

Miaoqing Huang and Shiming Li
Department of Computer Science and Computer Engineering
University of Arkansas
Fayetteville, AR 72701, USA
Email: {mqhuang, sl003}@uark.edu

Abstract—Physically unclonable functions (PUFs) have been a hot research topic in hardware-oriented security for many years. Given a challenge as an input to the PUF, it generates a corresponding response, which can be treated as a unique fingerprint or signature for authentication purpose. In this paper, a delay-based PUF design involving multiplexers on FPGA is presented. Due to the intrinsic difference of the switching latencies of two chained multiplexers, a positive pulse may be produced at the output of the downstream multiplexer. This pulse can be used to set the output of a D flip-flop to ‘1’. Further, it is proposed to directly incorporate challenge bits into the primitive PUF design to bring another layer of randomness for the response. Evaluation results on various devices and under different operating temperatures demonstrate the applicability of the proposed PUF design.

I. INTRODUCTION

Although manufactured in high volume, each hardware device is unique due to the physical randomness even with the same manufacturing process and the same material. This inherent variation can be extracted and used as its unique identification, as DNA to human beings. Hardware DNAs are not clonable, therefore providing a higher level of security. Hardware DNAs can be created from physically unclonable functions (PUFs) [1]–[6]. A Physically Unclonable Function (PUF) is a function that (1) is embodied in a physical structure, (2) is easy to evaluate but hard to predict, and (3) is easy to make but practically impossible to duplicate, even given the exact manufacturing process. In this respect it is the hardware analogue of a one-way function [7].

Formally a PUF is a function that maps a set of challenges to a set of responses based on an intractably complex physical system. Ideally, the function can only be evaluated with the physical system, and is unique for each physical instance. PUFs can be implemented with various physical systems [8], including non-electronic PUFs, analog electronic PUFs, delay-based intrinsic PUFs, and memory-based intrinsic PUFs. The latter two types of PUFs are typical ones used in integrated circuits. This paper presents a delay-based PUF design using the multiplexers.

PUFs can be used in many scenarios. One usage of PUF is to provide the keys for cryptographic algorithms. Public-key cryptographic algorithms [9] are typically used for authentication in which a private key needs to be stored inside the device using non-volatile memory, such as Flash memory or one-time programmable (OTP) ROM [10]. A capable adversary is able to access the key without damaging the system, therefore comprising the security. A countermeasure to this security threat is not to save the key at all. Instead, a PUF (e.g., a special hardware circuit) is deployed to generate a response (equivalent to a private key) given a challenge (equivalent to a public key) on the fly. Another usage is in the supply chain management. A PUF embedded in a hardware device will generate the unique response given a challenge, therefore proving its identity.

In this work, FPGA device is used as the platform to implement and evaluate the PUF prototype. Modern FPGA device consists of thousands of Configurable Logic Blocks (CLBs), each of which is comprised of multiple lookup tables (LUTs), flip-flops, and multiplexers. In [11] Anderson proposed a PUF design relying on the difference of the switching latencies of two multiplexers that are chained together. This work further enhances Anderson’s design by improving its randomness. A new pair of driving sequences to the select pins of multiplexers is presented. Additionally, we propose an approach to effectively integrating challenge bits to the primitive 1-bit PUF design. This new design not only further improves the randomness of the design, but also demonstrates a good scalability through experimental evaluation.

The remainder of the paper is organized as follows. The related work is introduced in Section II. The detailed design of the primitive PUF is presented in Section III. The experimental work on Xilinx Virtex-6 FPGAs and the results are discussed in Section IV, which is followed by the conclusions in Section V.

II. RELATED WORK

Physically unclonable functions (PUFs) have been a hot research topic in hardware-oriented security for many years. PUFs can be categorized roughly into two types, i.e., delay-based PUFs [1], [4], [11]–[15] and memory-based PUFs [16]–[20]. The delay-based PUFs requires hardware resources to implement. The advantage of the delay-based PUFs is that various challenges (inputs) can be applied on the PUF to produce the corresponding responses (outputs). On the other hand, the memory-based PUFs use the random initial values of SRAM cells or D flip-flops at power up as the fingerprint. Memory-based PUFs are not the focus of this work, therefore, will not be discussed in the remainder of this paper.

Figure 1 illustrates a PUF delay circuit based on multiplexers and an arbiter [4]. The circuit has a multi-bit input...
X and computes a 1-bit output \( Y \) based on the relative delay difference between two paths with the same layout length. The input bits \( X[i] \)'s determine the delay paths by controlling the multiplexers. A pair of multiplexers controlled by the same input bit \( X[i] \) works as a switching box. They pass through the two delay signals from the left side if the input control bit \( X[i] \) is zero. Otherwise, the top and bottom signals are switched. In this way, the circuit can create a distinct pair of delay paths for each unique input \( X \). To evaluate the output for a particular input, a rising signal is given to both paths at the same time. The same signal races through the two delay paths, and the arbiter at the end decides which path is faster. The output is ‘1’ if the signal to the data input \((D)\) is faster, and ‘0’ otherwise. In order to generate a response of \( n \) bits, \( n \) copies of the same circuit can be implemented, each of which produces a single output bit.

Kumar et al. presented a Butterfly PUF using cross-coupled latches [12]. The random initial status of a cross-coupled latch was used to generate a single PUF output bit. Mills et al. adopted a very similar approach in [13]. Instead of using latches, cross-coupled inverters implemented by LUTs were used in Mills’ design. Suzuki and Shimizu explored the glitches that behave non-linearly from delay variation between gates and the characteristic of pulse propagation of each gate [14]. A simulation model for the Glitch PUF was also presented.

Anderson presented a delay-based PUF design as shown in Figure 2(a) [11]. The contents of these two shift registers are initialized to 0x5555 and 0xAAAA, respectively. The shift register in LUT A will produce the sequence 0101 \( \cdots \). Correspondingly, the shift register in LUT B will generate the sequence 1010 \( \cdots \). The inputs of both shift registers are assigned such that the same sequences can continue infinitely beyond the initial 16 bits. In this way, the output from LUT A is always the complement of the output from LUT B. The outputs of the shift registers drive the select pins of the carry chain multiplexers, respectively. The 0-inputs of both multiplexers are tied to logic ‘0’. The 1-input of the bottom multiplexer is tied to logic ‘1’. The output of the bottom multiplexer, \( N_B \), is connected to the 1-input of the top multiplexer. The output of the top multiplexer, \( N_A \), is connected to the asynchronous “Set” input pin of a D flip-flop, which is initialized to ‘0’.

Since the outputs of both shift registers are complementary to each other, the stable output of the two chained multiplexers is ‘0’. When \{\( OUT_A, OUT_B \)\} is \{0, 1\}, the top multiplexer chooses the 1-input, i.e., \( N_A = N_B \). Since \( OUT_B = 0 \), the bottom multiplexer will choose 0-input, i.e., \( N_B = 0 \). Therefore, \( N_A \) is equal to ‘0’ for both cases. However, due to the dynamic behavior and signal transition of the circuit, a positive spike on \( N_A \) may occur while these two pieces of circuitry experience different delays because of random manufacturing variations. We will examine two cases, i.e., (1) LUT A and the multiplexer it drives are faster than LUT B and its multiplexer, and (2) LUT A and the multiplexer it drives are slower than LUT B and its multiplexer. In the first case, the transition of the top multiplexer between the two inputs occurs faster than the transition of the bottom multiplexer. Therefore, the output of the top multiplexer, \( N_A \), transitions from 0-input to 1-input, i.e., \( 0 \rightarrow N_B \), before \( N_B \) finishes the transition from 1-input to 0-input, i.e., \( 1 \rightarrow 0 \). A short positive spike will appear on \( N_A \) for the period before \( N_B \) transitions to ‘0’. In the second case, the bottom multiplexer is faster than the top multiplexer. Therefore, \( N_B \) transitions from 0-input to 1-input, i.e., \( 0 \rightarrow 1 \), before \( N_A \) finishes the transition from 1-input to 0-input, i.e., \( N_B \rightarrow 0 \). In this case, a short positive spike will also appear on \( N_A \) for the period before \( N_A \) transitions to ‘0’. This spike is illustrated as the glitch next to wire \( N_A \) in Figure 2(a).

Anderson [11] mentioned that the short glitch may be filtered out due to the electrical capacity of wires. However, our experimental results show that the probability for the glitch to reach the “Set” pin of the D flip-flop is very high. Therefore, a new set of driving sequences is proposed in our design.
III. PUF DESIGN USING MULTIPLEXERS

The FPGA device used in this work is Xilinx Virtex-6LX240T-1FFG1156 on the ML605 FPGA board. Modern FPGA device consists of thousands of Configurable Logic Blocks (CLBs). On the Virtex-6 FPGA devices, a CLB comprises two slices, each of which contains four 6-input LUTs, eight flip-flops, and other arithmetic circuitry (Figure 3). Six-input LUTs are small memories that are capable of implementing any logic function of up to 6 variables. For the LUTs on SliceMs, they can also be configured to distributed RAM or shift registers. Each LUT output connects to the select input of 2-to-1 multiplexer. Each multiplexer receives one of its data inputs from the multiplexer below it, and the other data input can be received from outside the slice. The vertical chain of four multiplexers is called the carry chain and it is intended for implementing fast arithmetic operations.

The PUF primitive in this work is similar to the one in Figure 2(a). However, it is found that the original input pairs for the two shift registers used in [11], 0x5555 and 0xAAAA, are not suitable for generating the output. No matter which multiplexer is faster, there is a good chance that a positive glitch will appear on $N_A$ due to the explanation provided in the related work. Therefore, a new pair of inputs, i.e.,

Input A: 0x8888···(1000100010001000···) and
Input B: 0x4444···(0100010001000100···),

is used such that the output would be ‘1’ if the bottom multiplexer is faster than the top one, and vice versa, as elaborated as follows. A positive glitch will appear on $N_A$ only when both multiplexers choose 1-input. When the bottom multiplexer is switching from 0-input to 1-input, the top multiplexer is switching from 1-input to 0-input. If the bottom multiplexer is faster than the top one, it finishes the switching from 0-input to 1-input before the top multiplexer switches from 1-input to 0-input, allowing the signal ‘1’ to reach $N_A$. Otherwise, $N_A$ will stay ‘0’ if the top multiplexer has a shorter switching latency than the bottom one.

By duplicating the PUF primitive, a multi-bit output can be generated as a hardware fingerprint for each FPGA device. However, a static fingerprint is vulnerable to security attack. A challenge-response framework has to be applied such that each multi-bit challenge will be used to generate a unique response. A simple approach to applying the challenge-response framework as mentioned in [11] is to have the challenge input word drive the select inputs of wide multiplexers. The challenge word selects two different PUF bits whose values are exclusive-OR’ed to produce an output bit depending on the challenge word. Additional multiplexers can be added, each wired differently, to produce a multi-bit output. Adopting this approach requires to use $2N$ wide multiplexers to generate an $N$-bit response. It may take a lot of hardware resources for large $N$’s. Since the outputs of these PUF primitives are static, it is still vulnerable to modeling attack.

Figure 2(b) illustrates the proposed PUF design that directly incorporates a challenge bit $x$. There are three multiplexers in the design. The output from the top shift register is connected to the select pin of the top multiplexer. The challenge bit $x$ is used to choose one multiplexer from the other two. When $x = 1$, the output from the bottom shift register is directed to the select pin of middle multiplexer. Otherwise, the bottom multiplexer is used to compare with the top one.

The proposed design has a good scalability. The randomness of the PUF design can be enhanced by increasing the number of multiplexer pairs for generating the positive glitch. For example, if two challenge bits are incorporated in the design, 4 multiplexers can be adopted to provide 4 pairs of multiplexers. Each multiplexer pair corresponds to a specific value of two challenge bits. A prototype implementation of a PUF primitive incorporating 2 challenge bits is shown in Figure 3(b). The evaluation results are presented in Section IV-B.

IV. EXPERIMENTS AND RESULTS

As mentioned before the experiments are carried out on Xilinx Virtex-6LX240T-1FFG1156 FPGA device, which uses 40 nm CMOS technology. The primitive one-bit PUF design requires to use the output pin of the shift register as the control pin for the multiplexer, as shown in Figure 3. Shift register has to be implemented using LUT in SliceM, which is variant of general slice and supports two additional functions: storing data using distributed RAM and shifting data with 32-bit registers. On Virtex-6 each CLB contains zero or one SliceM. We use two CLBs to implement a primitive 1-bit PUF, as shown in the prototype implementation in Figure 3(a). Eight multiplexers belonging to two SliceMs form a chain. The three multiplexers used to implement the primitive 1-bit PUF are within the dashed circles in the diagram. The LUT next to each corresponding multiplexer is configured to shift register to provide the proper drive signal for the select pin of the multiplexer. In our experiments, it is found that it is very important to keep proper distance between the two multiplexers in order to generate the positive glitch at an average probability of 50%. Therefore, we choose the top multiplexer and the bottom one in the same SliceM to implement Multiplexer A and Multiplexer B, respectively. The top multiplexer in the other SliceM is used to implement Multiplexer C.

How to physically incorporate a challenge bit into the implementation is up to the circuit designer. In this work, we use a separate component called “Driving Sequence Generator”, which takes the challenge bit $x$ as input, to generate the input sequences for the three shift registers. The continuous sequence
when the temperature of the device is around 30
distinct locations near the center. The results are collected
locations on the same FPGA device, i.e., 4 corners and 2
A. Four Experiments of the Primitive PUF Incorporating 1
route step of the design process. In the following, we carry
particular LUTs and multiplexers in the PUF implementation
number of LUTs and multiplexers inside a slice to specify the
location constrain to define the relative locations of all PUF
work is 13.1. The PUF is coded in VHDL. We use the relative
only one CLB on Virtex-6 FPGA. The operating frequency
plementation of the “Driving Sequence Generator” occupies
Virtex-6LX240T FPGA. It is worth mentioning that the im-
a 256-bit PUF, which occupies
<br>

\[
\text{Input B} = \begin{cases} 
0x4444 \cdots & \text{if } x=’1’, \\
0xFFFF \cdots & \text{if } x=’0’. 
\end{cases}
\]

\[
\text{Input C} = \begin{cases} 
0xFFFF \cdots & \text{if } x=’1’, \\
0x4444 \cdots & \text{if } x=’0’. 
\end{cases}
\]

By duplicating the implementation in Figure 3(a), we build
a 256-bit PUF, which occupies <2% of resources on Xilinx
Virtex-6LX240T FPGA. It is worth mentioning that the im-
plementation of the “Driving Sequence Generator” occupies
only one CLB on Virtex-6 FPGA. The operating frequency
of the circuit is 100 MHz. The version of ISE used in this
work is 13.1. The PUF is coded in VHDL. We use the relative
location constrain to define the relative locations of all PUF
elements on the FPGA devices. We also use the internal
number of LUTs and multiplexers inside a slice to specify the
particular LUTs and multiplexers in the PUF implementation
on Xilinx platform. Default setting is used in the place and
route step of the design process. In the following, we carry
out 4 experiments to evaluate the proposed PUF design.

A. Four Experiments of the Primitive PUF Incorporating 1
Challenge Bit

1) PUFs on the Same FPGA Device: In the first exper-
iment, the same 256-bit PUF is replicated at 6 different
locations on the same FPGA device, i.e., 4 corners and 2
distinct locations near the center. The results are collected
when the temperature of the device is around 30°C. For all
the 256 1-bit PUF primitives, the challenge bit \( x \) is set to ‘1’,
i.e., Multiplexer B is chosen to pair with Multiplexer A for
generating the glitch.

The average number of bit ‘1’ in the 6 256-bit PUF outputs
is 123.3. The 48.2% probability for the primitive 1-bit PUF
design to generate the positive glitch shows a good randomness
of the approach. Further we perform a pair-wise Hamming
distance calculation for the 6 PUF outputs. The average pair-
wise distance is 125.2. The distribution of these 15 Hamming
distances is illustrated in Figure 4(a). Both the average distance
and the distance distribution demonstrate the randomness of
the PUF design regardless of the location. As a comparison,
we apply the driving sequences suggested in [11] to the 256-
bit PUF, i.e., 0x5555 and 0xAAAA. It is found the average
number of bit ‘1’ in the 6 256-bit PUF outputs is 238.3, which
means that there is a probability of 93.1% to generate
the positive glitch. We also compute the pair-wise Hamming
distances among the 6 PUF outputs. The distribution is shown
in Figure 4(b) with an average distance of 31.7.

2) PUFs on Multiple FPGA Devices: In order to further
demonstrate the randomness of the PUF design, the experiment
in Section IV-A1 is replicated on 5 different FPGA boards.
The experimental conditions are kept unchanged. 30 256-bit
PUF outputs are collected when the challenge bits are set
to all 1’s. Similar to the experiment on a single board, 435
pair-wise Hamming distances are calculated. The distribution
is shown in Figure 5. The distribution of Hamming distance
across 5 FPGA boards is wider than the distribution on a single
board. This wider distribution is expected because the number
of PUF outputs increases 5 times. The majority (>85%) of
the Hamming distances is within the range of [101:140],
demonstrating a good randomness of the PUF design across
different FPGA devices.

3) PUFs under Different Operating Temperatures: The
realibility of the PUF design is verified by measuring the PUF
output under different temperatures. The PUF under test is
at the bottom left corner of the FPGA device. Similar to the
previous experiments, Multiplexers A and B are used as a pair
for producing the glitch. 9 operating temperatures are used, i.e., 30°C-70°C with 5°C gap. The distribution of Hamming distances among the 9 PUF outputs is illustrated in Figure 6. The smaller the Hamming distance between outputs at various temperatures, the better. The average Hamming distance of the 36 pairs of PUF outputs is 18. There are two approaches to further improving the reliability of the proposed PUF design. The first approach is to add circuit to perform ECC on the raw output of the PUF [4], [15]. The second approach specific for the PUF design in this work is to increase the distance between the multiplexer pair. The longer the distance between these two multiplexers, the more time the signal takes to travel from the bottom multiplexer to the top multiplexer, giving more margin for the top multiplexer to finish the switching between its two inputs. However, the probability to generate the positive glitch becomes smaller when the physical distance between the two multiplexers increases. Therefore, a more detailed tradeoff analysis is worthwhile and can be part of the future work.

4) PUFs with Different Challenges: In the previous three experiments, the value of all 256 challenge bits is set to ‘1’, i.e., Multiplexers A and B are used for generating the PUF output. In this experiment, we flip the value of all challenge bits to ‘0’ so that Multiplexer C is chosen to pair with Multiplexer A. The output of the 256-bit PUF using Multiplexers A and C is compared with the PUF using Multiplexers A and B at the same location. The Hamming distance of the two outputs with different values of the challenge bit is illustrated in Figure 7, which displaces all the 6 locations on an FPGA board. From the results, we can find that choosing a different pair of multiplexers does not necessarily generate a different output bit. Otherwise, the Hamming distance between the two outputs with complementary challenge bits will be 256. Therefore, the proposed PUF design provides two advantages compared with previous work. (1) It adds more randomness and security. (2) It provides a novel and efficient approach to incorporating challenge bits to the PUF design.

B. Design and Experiments of the PUF with 2 Challenge Bits

In this experiment, we want to demonstrate the scalability of the primitive PUF design while keeping randomness. Two bits of challenge are incorporated into the design so that one pair of multiplexers can be chosen from 4 combinations.

We use the 4 multiplexers indicated in Figure 3(b), i.e., Multiplexers A, B, C and D, to provide the 4 combinations. As discussed before, the two multiplexers chosen as a pair should keep a proper distance. If they are too close, it is highly likely to generate a positive pulse. On the other hand, if they are too far, it is very difficult to generate a positive glitch. Therefore, two multiplexer pairs, i.e., \{A, D\} (too far) and \{B, C\} (too close), are not used in the prototype implementation. The complete list of multiplexer pairs, the corresponding value of the 2 challenge bits and the driving sequences are given in Table I.

We perform the experiment on the PUF design with 2 challenge bits similar to the experiment in Section IV-A4. For the 256-bit PUF, the same value of the 2 challenge bits is given to all 256 PUF primitives, each of which generates 1 bit of output. All 4 different values are given in sequence and the corresponding PUF outputs are recorded. Figure 8(a) shows the average number of 1’s in the outputs from the same PUF under
4 different challenges. All the 6 PUFs at different locations are capable of producing outputs with good randomness, i.e., the probability to produce ‘1’ for each primitive PUF is within the range from 43.0% to 55.5%. In addition, we perform the pairwise calculation of Hamming distance among the 4 outputs from the same 256-bit PUF. The average Hamming distance of 6 output pairs from the same PUF is shown in Figure 8(b). 6 PUFs are implemented at 6 different locations. The range of the average Hamming distances is [112, 124], demonstrating the good randomness of the proposed design.

Finally, we perform the pair-wise calculation of Hamming distance among all 24 PUF outputs from 6 different locations. The Hamming distance distribution is illustrated in Figure 9, which resembles a Gaussian distribution. Comparing with the distribution in Figure 5, the Hamming distance here has a wider distribution. This wider distribution is due to the fact that 4 different multiplexer pairs are used to generate the 24 PUF outputs instead of only one pair of multiplexers used in the experiment in Section IV-A2. Further it is noted that the resource requirement of the PUF design incorporating 2 challenge bits is almost same to the resource requirement of the PUF design that incorporates only 1 challenge bit. The Multiplexer D in Figure 3(b) is used as a bypassing component in Figure 3(a). This change does not introduce extra hardware cost except the slight modification of the function of the “Driving Sequence Generator” unit.

V. CONCLUSIONS

This work presents a delay-based PUF design involving multiplexer chains. Due to the intrinsic difference of the switching latencies of two chained multiplexers, a positive pulse may be produced at the output of the downstream multiplexer. This pulse can be used to set the output of a D flip-flop to ‘1’. The original design by Anderson in [11] is optimized in this work by improving its randomness and scalability. A new pair of driving sequence to the select pins of multiplexers is presented. Additionally, we propose an approach to effectively integrating challenge bits to the primitive 1-bit PUF design. Evaluation results on various FPGA devices and under different operating temperatures demonstrate the applicability of the proposed PUF design. This new design that incorporates challenge bits not only further improves the randomness of the design, but also demonstrates a good scalability through experimental evaluation.

A more detailed tradeoff analysis between the randomness and the reliability of the PUF outputs is planned as part of the future work. In order to incorporate more challenge bits in the primitive 1-bit PUF design, a more accurate theoretical model needs to be built by including the wire delay along the multiplexer chain and other factors.

ACKNOWLEDGMENT

The authors would like to thank Xilinx Inc. for the donation of Virtex-6 FPGA ML605 Evaluation Kits. The authors would also thank Dr. Jason Anderson at the University of Toronto for sharing the source code of his PUF design on Virtex-5 FPGA.

REFERENCES