Ring Oscillator PUF Design and Results

Michael Patterson
Reconfigurable Computing Laboratory
Iowa State University
mjpatter@iastate.edu

Joseph Zambreno
Reconfigurable Computing Laboratory
Iowa State University
zambreno@iastate.edu

Chris Sabotta
Reconfigurable Computing Laboratory
Iowa State University
csabotta@iastate.edu

Sudhanshu Vyas
Reconfigurable Computing Laboratory
Iowa State University
spvyas@iastate.edu

Aaron Mills
Reconfigurable Computing Laboratory
Iowa State University
ajmills@iastate.edu

1. Motivation and Background

A Physical Unclonable Function (PUF) is a function in some physical device that is easy to evaluate but hard to predict. In many ways it is the hardware equivalent of a mathematical one-way function. Every challenge should map to a specific response, but this mapping should be impossible to predict and different for every physical device.

An on-chip PUF is very useful for several reasons. First, it provides a way to uniquely identify a given device. This could be used for secure chip authentication or for protection of Intellectual Property. Second, since an FPGA PUF is based on intrinsic randomness, it could also serve as a Random Number Generator. Finally, and most importantly, a PUF provides all this functionality without storing any secret information on the chip. Most solutions to the previously listed problems involve storing some type of key or seed value on the chip. This poses many security hazards. However, a PUF relies on the characteristics of the chip itself, which are considered impossible to observe and duplicate.

There are a wide variety of PUF designs, but the only ones relavant to our interests are those that can be implemented on an FPGA. There are basically five designs that meet these requirements.

1)The first PUF designed for an FPGA was introduced in [1]. The proposed ”Arbiter PUF” design can be seen in Figure 1. Two paths are created by connecting a number of ”switch delay elements” in series. Each element uses a two-to-one multiplexer to control which path is taken by the inputs. The challenge bits X are used as the select bits to the multiplexers, and the response of the PUF is based on which path leads to the faster signal propogation.

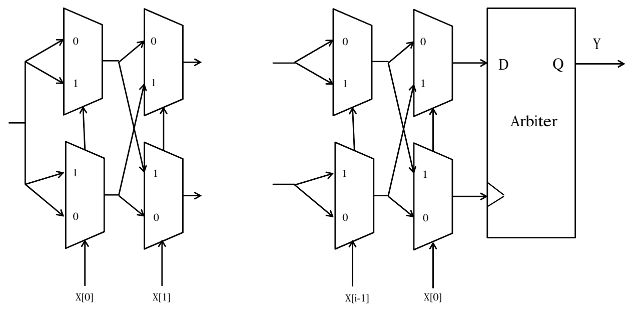

Figure 1. Arbiter PUF Design

However, there are severe problems with using this type of PUF on an FPGA. The delays due to the routing done automatically by the tools completely overshadow the small delays due to the process variability. Since the basis for the PUF is the randomness intrinsic to the process variability, this design will not function well.

2)Another PUF design is described in [5]. This Ring Oscillator PUF is composed of many delay loops that oscillate with a particular frequency. They are laid out identically, but the minor variations in manufacturing lead to loops with slightly different frequencies. The loops drive counters which are used to produce the response bits to a given challenge. Figure 2 shows the structure of this type of PUF.

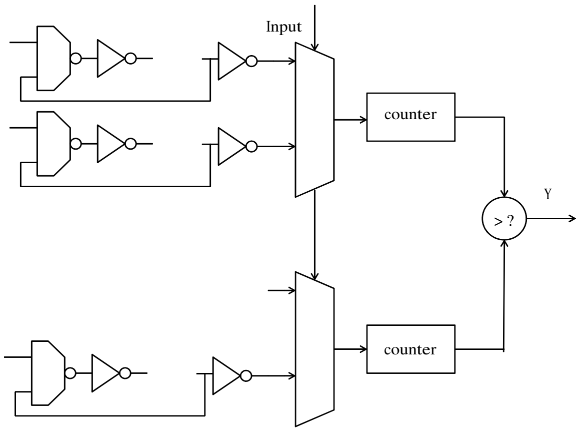

Figure 2. Ring Oscillator PUF Design

This design also suffers from the same drawbacks as the first one. The randomness is dominated by the difference in routing, not by the differences in process variation. However, by using hard macros to make the routing very symmetric, we hope to solve this problem. Our methods for accomplishing this are described further in the following section.

3)A different design option is described in [2]. This design does not depend on laying out a certain circuit and programming it onto an FPGA. Instead, it relies on the SRAM that is present on most modern FPGAs. Since an SRAM bit assumes a ”random” value of 0 or 1 when it is originally given power, these bits can be used to provide a unique response.

However, we decided not to pursue this design for the current CSAW competition for a few reasons. First, we did not find many ways in which this design could be improved or expanded. Also, more and more FPGAs provide some form or option of SRAM initialization. This functionality would prevent the use of an SRAM PUF.

4)Another design very similar to the SRAM PUF is described in [3]. This Butterfly PUF is based on unstable cross-coupled circuits and can be seen in Figure 3. It basically serves the same purpose as the SRAM design, but it can be implemented on any FPGA, not just those with SRAM.

5)Finally, yet another design is discussed in [4]. This design actually uses a bitfile to reprogram the FPGA as the

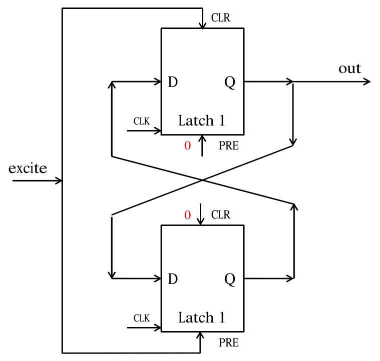

Figure 3. Butterfly PUF Design

challenge. The characteristics of the programmed FPGA then serve as the response. While this approach is rather unique and interesting, it is not within the scope of this competion.

2. Design

As previously mentioned, we chose to implement a Ring Oscillator PUF. We started with a fairly basic design. Figure 4 shows a five-stage ring oscillator that serves as the basis of our PUF. It includes an enable bit so that only the ring oscillators that are currently being used will oscillate. This is needed to prevent our chip from overheating.

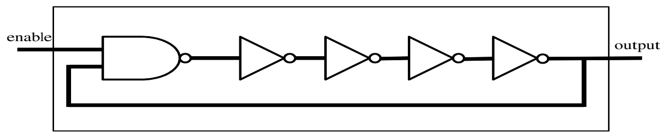

Figure 4. Basic Ring Oscillator

Figure 5 shows how we are using 256 ring oscillators in combination with some other components to generate the output. A decoder is used to generate the enable bits for the oscillators, and a mux is used to choose which ring oscillator gets fed into the counter based on the input bits. The two counters are then compared, and a one is output if the first counter value is larger than the second. A zero is output if this is not the case. This basic block is used eight times to generate the eight required output bits. The testing and results generated by this PUF design are discussed in sections 3 and 4.

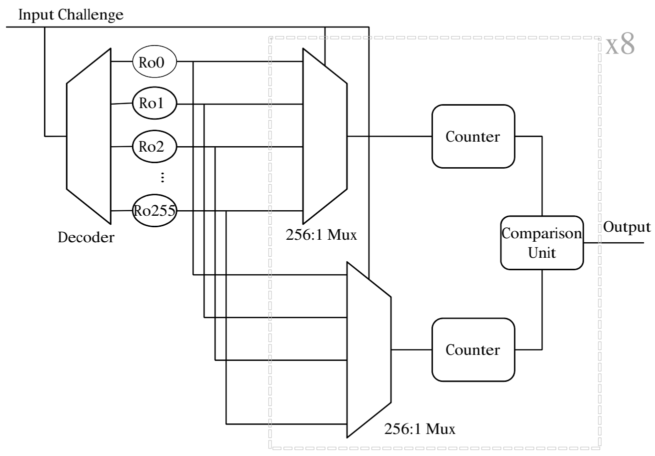

Figure 5. Our PUF Design

As mentioned earlier, this ring oscillator PUF design can be greatly improved by manual routing of the ring oscillators. This guarantees that every instance of a ring oscillator has identical routing, so the delays truly are based on the physical process variations and not the automatic routing process. In addition to improving the characteristics of a PUF, it also packs the ring oscillators into fewer slices resulting in a more efficient use of resources. Figure 6 shows the resources used by a ring oscillator that is not manually placed. In contrast, figure 7 shows the resources used by not one, but eight ring osciallators that are manually placed through hard macros. When not manually routed, a single ring oscillator takes four slices. However, with manual routing eight ring oscillators can be packed into ten slices. This is a drastic improvement. Unfortunately, due to time constraints and bugs in the tools, we were unable to implement and test this design.

3. Experimental Setup

The success of a PUF can be viewed by its performance in three major categories. First, for a challenge that is repeatedly given to a PUF, the response should be consistent. Second, for two challenges that are very similar, (having only one different bit), the two responses should be very unique. This uniqueness is often measured by computing the hamming distance (number of changed bits) between two responses. Finally, for two instances of the PUF, the

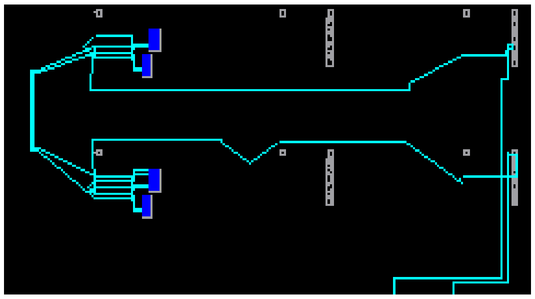

Figure 6. Automatically Routed RO

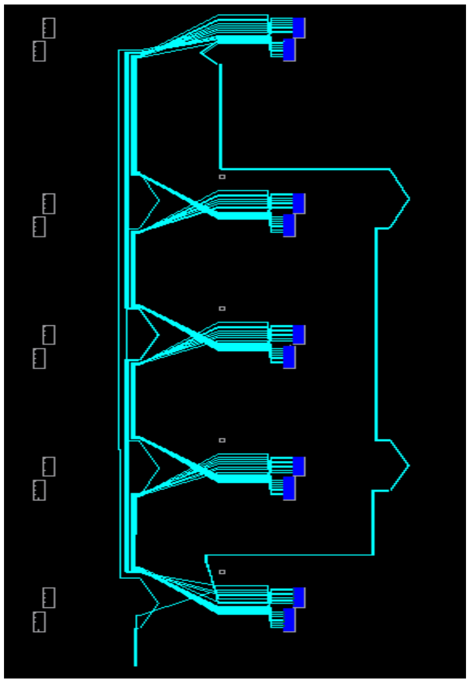

Figure 7. Manually Routed RO

responses to the same challenge should have a significant hamming distance as well. All three of these components are critically important to the PUF, as a substantial short coming in any of the categories can render it useless. In order to evaluate the performance of our PUF design, we designed the following tests. Every test was carried out on a Xilinx Spartan-6 LX45 FPGA.

3.1. One PUF, One Challenge Test

A single PUF is given the same 16-bit challenge 32 times, and this is repeated for 128 randomly generated challenges. Every response to a given challenge should be exactly the same, so the percentage of responses that differ will be calculated. In addition to this, the analysis will also be applied to individual bits. In both cases, the ideal value is a 0 percent change.

3.2. One PUF, Multiple Challenges Test

A single PUF is given 1024 different challenges consisting of a Gray Code pattern (a series of numbers that tours unique data values by changing only one bit at a time). The average hamming distance between adjacent responses will be calculated. Again, this will also be done at the bitwise level. In both cases, the ideal average hamming distance is 50 percent.

3.3. Multiple PUFs, One Challenge Test

Multiple PUFs are given the same series of 1024 challenges. The average hamming distance between responses is calculated. This value is also calculated at the bitwise level. In both cases, the ideal average hamming distance is 50 percent.

4. Analysis

The three tests described in the previous section were performed on our PUF design, and the following results were gathered.

Response Bit	0	1	2	3	4	5	6	7
Ham. Dist. (%)	1.11	1.44	1.10	1.30	1.22	0.91	1.33	1.27

Figure 8. One PUF, One Challenge Test

The whole 8-bit response was correct 94.68 percent of the time.

Response Bit	0	1	2	3	4	5	6	7
Ham. Dist. (%)	20.5	48.4	36.9	43.1	50.7	50.7	27.6	37.1

Figure 9. One PUF, Multiple Challenges Test

On average, 2.88 bits change in the response for a one bit change in input.

Response Bit	0	1	2	3	4	5	6	7
Ham. Dist. (%)	44.0	44.1	46.2	46.9	47.4	48.3	45.2	46.8

Figure 10. Multiple PUFs, One Challenge Test

On average, 46.16 percent of the bits are different between responses of two different PUFs to the same challenge.

As the previous results show, our PUF design is very consistent. The bitwise results of the first test show that any given bit will change only about once in 100 challenges, and the overall response will be correct almost 95 percent of the time. The second group of test results shows that a change in one input bit results in a change on average of almost three output bits. Ideally this value would be closer to four, but three is high enough that our design is still able to function well as a PUF. Finally, the results of the third test show that different instances of our design generate different responses to a given challenge. Over 40 percent of the bits were different in responses from different PUFs, which allows instances of our design to easily be uniquely identified.

5. Temperature Stability

Observing consistency in the face of wide ambient temperature changes is also essential. Being a delay-based PUF, our design relies on the propagation delays inherent in the interconnects and the LUTs within the FPGA. However, since the resistance of the interconnects in particular change with temperature, so to their RC delays. For our PUF we also performed a test to determine how consistent our results are as temperature varies from 10C to 65C. The experiment is designed the same way as the One PUF, one Challenge test, with the exception that the test is repeated every 5C. The number of bits that differed between 10C and 65C are shown below.

Response Bit	0	1	2	3	4	5	6	7
Ham. Dist.(%)	0.3	2	0.1	0.6	0	0	0.2	2

Figure 11. Temperature Stability

The results are quite close to the desired 0% change. In addition, a pattern was observed in the experiment. The majority of the bit observed bit changes occurred for only a small subset of the input challenges. This suggests that some challenges are more robust than others. Given that the delay path through the multiplexors will change based on the challenge, it is assumed that some routes are more susceptible to temperature change than others (for example, if the interconnects are longer). It is proposed then that simply constraining the challenge set to those that are the most robust will drive the temperature-induced error rate down to near-zero.

6. Conclusions and Future Work

We were able to successfully create and implement a design that meets all the criteria for being a PUF. Our testing and analysis demonstrated the functionality of our design.

However, there are definitely areas in which additional work could be done, especially in the area of testing. We would like to be able to test the full input challenge space, and in general increase the repetitions of each test in order to increase the statistical significance of our results.

We would also like to explore using larger bit-widths to increase the security of the PUF. Unfortunately, our design (especially the multiplexors) consumed all the resources of our Xilinx Spartan-6 LX45. Therefore, our design would have to be refactored for size-reduction. For example, the calculation could be divided into multiple phases in which subsets of the challenge vector are shifted in one-by-one. This would allow greater hardware reuse.

Finally, we would like to place our PUF on multiple different FPGAs and analyze some results. It would be interesting to see the different characteristics between multiple PUFs on a single FPGA and multiple PUFs on different FPGAs.

References

[1] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Delay-based circuit authentication and applications. In *Proceedings of the 2003 ACM symposium on Applied computing*, SAC '03, pages 294–301, New York, NY, USA, 2003. ACM.

[2] J. Guajardo, S. Kumar, G.-J. Schrijen, and P. Tuyls. Physical unclonable functions and public-key crypto for fpga ip protection. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 189 –195, aug. 2007.

[3] S. Kumar, J. Guajardo, R. Maes, G.-J. Schrijen, and P. Tuyls. Extended abstract: The butterfly puf protecting ip on every fpga. In *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*, pages 67 –70, june 2008.

[4] M. Majzoobi and F. Koushanfar. Time-bounded authentication of fpgas. *Information Forensics and Security, IEEE Transactions on*, 6(3):1123 –1135, sept. 2011.

[5] G. E. Suh and S. Devadas. Physical unclonable functions for device authentication and secret key generation. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 9–14, New York, NY, USA, 2007. ACM.