# Securing Netlist-Level FPGA Design through Exploiting Process Variation and Degradation

Jason Xin Zheng, Miodrag Potkonjak
Computer Science Department
University of California, Los Angeles (UCLA)
Los Angeles, USA
{jxzheng,miodrag}@cs.ucla.edu

## ABSTRACT

The continuously widening gap between the Non-Recurring Engineering (NRE) and Recurring Engineering (RE) costs of producing Integrated Circuit (IC) products in the past few decades gives high incentives to unauthorized cloning and reverse-engineering of ICs. Existing IC Digital Rights Management (DRM) schemes often demands high overhead in area, power, and performance, or require non-volatile storage. Our goal is to develop a novel Intellectual Property (IP) protection technique that offers universal protection to both Application-Specific Integrated Circuits (ASIC) and Field-Programmable Gate-Arrays (FPGAs) from unauthorized manufacturing and reverse engineering. In this paper we show a proof-of-concept implementation of the basic elements of the technique, as well as a case study of applying the anti-cloning technique to a nontrivial FPGA design.

## Categories and Subject Descriptors

B.7.m [**Hardware**]: Integrated Circuits—*Miscellaneous*

## General Terms

Design, Experimentation, Measurement, Security

## Keywords

IP protection, active hardware metering, unclonable

## 1. INTRODUCTION

There exist several high-impact gaps in the design, implementation, and manufacturing of integrated circuits (ICs), including silicon capacity vs. design productivity, number of gates vs. number of pins, and the disparity between gate delays and wire delays. These gaps have been having deep and profound impacts on both IC design and manufacturing processes. In the last two decades another gap emerged that may have a far-reaching economic impact on the semiconductor industry. The gap between Non-Recurring Engineering (NRE) costs and Recurring Engineering (RE) costs has been growing exponentially as the manufacturing process continues to scale down. The numbers are truly fantastic: while in the sixties the cost of manufacturing one gate was $1, it is expected that by the end of this decade 1 trillion gates will cost only $1. Meanwhile, owing mainly to the increasing size and verification complexity of the designs that are executed, the cost of designing a modern IC product has skyrocketed. On the other hand, the cost of building a state-of-the-art semiconductor foundry has also rapidly grown to well over $1 billion. The NRE-RE cost gap provides high incentives for independent silicon foundries to recuperate setup costs by manufacturing non-authorized ICs, and for fab-less design houses to prevent manufacturing piracy.

This situation provided impetus for the initiation of active IC digital rights management (DRM) research. Several techniques have been proposed and implemented. They share at least one of two common denominators: the use of physically unclonable functions (PUFs) [2][4] or conditionally enabling through classical cryptographical one-way functions [39][7]. While these techniques address several aspects of IC intellectual property (IP) protection such as prevention of use of non-authorized ICs, they have significant limitations including rather high area, power, and frequency overheads, additional storage requirements for enabling keys, and susceptibility to operational and environmental conditions.

Most importantly, they do not offer protection of the design know-how that is often strategically important, i.e. although the attacker may not be able to produce non-authorized ICs, he can gain insight on how significant parts of the design are created.

Furthermore, the arsenal for IP protection is even more sparse for volatile SRAM-based Field-Programmable Gate-Array (FPGA) designs [20][11][15], where a locally-stored configuration bitstream is usually required. Although the stored bitstreams can be encrypted with existing cryptographical mechanisms, there exists similar overhead and storage concerns over this technique as crytptographically enabled IC designs.

In this paper, we present a novel technique for comprehensive IP protection and active device metering with very low resource and energy overhead. The technique is universally applicable to both Application-Specific Integrated Circuit (ASIC) and FPGA designs. In addition to protecting against non-authorized manufacturing, we show that reverse-engineering of the design can be prevented at user-

*FPGA'12,* February 22–24, 2012, Monterey, California, USA.

specified levels. Key to this IP protection approach is to implement sensitive logic paths in such a way that their functionality can be altered, post-silicon, using targeted device aging. In the context of FPGA IP protection, this means that the chosen sensitive logic paths can be altered, through targeted aging, independent of the bitstream design.

The notion of sensitive logic paths reflects the understanding that not the entire silicon product requires protection against cloning or reverse engineering. For instance, it is typically not necessary to protect a generic multiplier, as such designs are easily obtainable or trivial. Sensitive logic paths refer to the portions of the design where critical know-how or functionality is embedded, such as the Finite State Machine (FSM) of a video compression engine.

## 1.1 Key Concepts

To understand how logic components can be configured post-silicon or post-bitstream, we introduce the concept of delay logic. Traditional binary combinational logic produces outputs that can be determined statically once the functions and the connectivities of each element, e.g. the netlist of the design, are known. Delay logic is a type of logic whose outputs depend on a third runtime factor, which is the delays of the gates in the circuit.

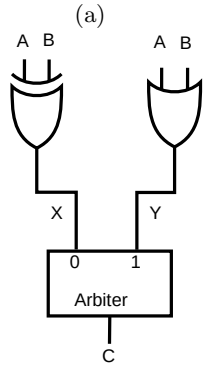
A small example is presented in Figure 1(a), where the outputs of an XOR gate and an OR gate are combined by an arbiter element. The output C of the arbiter is determined in the following fashion:

- If a rising edge arrives on input port 0 before input port 1, then the output is 0.
- If a rising edge arrives on input port 1 before input port 0, then the output is 1.

If the initial state of the circuit is that A and B are both 0, and A and B are changed at the same time, then it is clear that the output port C depends on the knowledge of the delay of the XOR and OR gates in the circuit. The tables in Figure 1(b) and (c) show the values of output $c$ as a function of inputs $a$ and $b$ for the two possible relative speeds of the XOR and OR gates. Evidently the whole circuit behaves like an AND gate when the XOR is faster than the OR gate, and an OR gate when the OR is faster than the XOR.

The delay logic has a clear advantage to the traditional combinational logic for protection against cloning and reverse engineering, due to the fact that its output depends on a dynamic delay factor. In the context of this paper, we assume that obtaining such information from arbitrary gates and designs is difficult. However, there are challenges to using delay logic in implementation. Many factors, such as die temperature, $V_{dd}$, and manufacturing variations, affect the logic delay, thus making it difficult to predict. The use of the arbiter elements relaxes the design constraints by measuring only the relative delays between two paths. Furthermore, the reliability of the delay logic can be improved through post-silicon configuration.

Negative Bias Temperature Instability, or NBTI [1][28], is an




(b)

| A | B | X | Y | C |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 |

(c)

| A | B | X | Y | C |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 |

**Figure 1: (a) Delay logic with XOR and OR gates. (b) Truth table when XOR is faster than OR. (c) Truth table when OR is faster than XOR.**

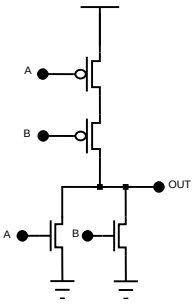



**Figure 2: CMOS NOR gate.**

aging process that occurs to CMOS transistors when a negative bias is applied to the gate. When stressed, the breakage of hydrogen-silicon bonds creates interface traps which lead to increases in the effective threshold voltage $V_{th}$ of the gate. Figure 2 shows a typical CMOS representation of a NOR gate. The top transistors are of PMOS type, and the bottom ones are of NMOS type. Although NBTI acts upon both the PMOS and NMOS transistors, PMOS transistors are impacted more significantly than NMOS transistors, as they are always negatively biased when turned on. Nevertheless, the overall effect of the NBTI aging is that the logic propagation delay through the gate is increased, i.e. the gate is slower.

Combining selective NBTI aging with arbiter-based delay logic yields a new approach to protecting the sensitive logic paths of an IC design. Without the knowledge of relative logic speeds, an attacker cannot understand the functionality of the circuit, even if he was equipped with the full gate-level netlist. Without proper post-silicon configuration, a non-authorized copy of the IC will not function as intended.

## 1.2 Related Efforts and State-of-the-Art

Security techniques for FPGA designs and implementation are a broad research area that covers a variety of issues ranging from digital rights management (DRM) and detection of malicious circuitry to reverse engineering and trusted synthesis. There are several recent surveys on FPGA security [45][30]. In our brief survey of the related work we mainly

focus on directly related IC DRM and FPGA security techniques.

The first set of FPGA DRM techniques was created by John Lach and his coauthors [25][26][27]. Champagne et al. [13] discussed secure techniques for distribution of FPGA configurations.

The largest impetus for IC reverse engineering, its surprising easiness and effectiveness was created by Cambridge University researchers [5]. Consequently, several groups demonstrated that even highly security hardware security primitives such as PUFs is surprisingly easy to reverse engineer [29][40].

The IC IP protection efforts emphasized techniques that enable zero knowledge proofs that a particular hardware is designed by a specific entity. Almost all of them used some form of design watermarking and/or fingerprinting [24][19][36]. Consequently these IC fingerprinting techniques were combined with data mining techniques to form first passive metering approaches [22][3][46][47]. Passive IC metering enables counting of the number of non-authorized ICs. Since 2007, several active IC metering techniques have been developed [4][39]. These technique enable remote activation and deactivation of ICs. In many of these techniques PUF and PPUF [17][8] play a crucial role in the creation and employment of unique IC IDs. They are creative and effective solutions and advance active metering research frontier. Nevertheless, they are subject to several significant limitations such as high hardware and energy overheads, limited security protocols flexibility (e.g. no mechanisms for specifying active time intervals and only single user control), and the requirement of key storage that may be the source of security vulnerabilities. In addition, they are not amenable to quantitative security analysis and do not guarantee prevention of the IC reverse engineering.

While recovering netlists by reverse engineering actual ICs is a well-established research and business endeavor [9][33][23], there is surprisingly little reported work on IC reverse engineering to higher levels of abstraction. Notable exceptions include efforts at the University of Michigan [18], Michigan State [48][49][14], and recently the Air Force Institute of Technology [35][32][31]. However, there have been numerous reverse engineering efforts at lower levels of abstraction mainly with the goal of verifying actual implementations [10][42][12].

Recently, in a series of papers Torrance and James provided detailed description about capabilities, and limitation of state-of-the-art industrial IC reverse engineering procedures [43][44]. Even more recently, reverse engineering started to attract rapidly growing interest from academic community [41].

In addition to directly related IC structure extraction and reverse engineering techniques, there are several other related areas. They are related either because we use their techniques and tools or due to conceptual similarity. The most difficult task of IC reverse engineering is probably FSM extraction and traversal. The problem has been addressed in several communities [38][21][6]. Furthermore, identification and coverage of regular patterns has been a popular and important problem in behavioral and system synthesis [34][37].

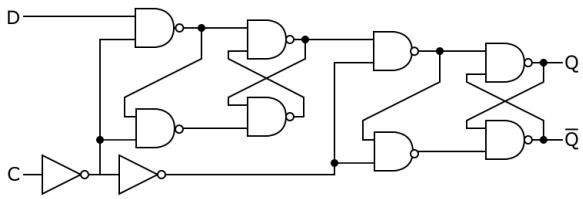



**Figure 3: An example DFF design[50].**

## 1.3 Overview

In the rest of this paper, we present a proof-of-concept of the delay logic implemented on an FPGA platform and show that not only the delay logic responds to the slight differences in delays caused by process variation, but also to the controlled aging effects of NBTI. We introduce this delay logic element in Section 2 and explain in detail its implementation.

To show that manufacturing variation introduces observable relative delay differences that are unique for each FPGA, in Section 3 we present the experimental results of the delay logic by comparing the outputs from an array of 64 arbiters implemented on two FPGAs using identical configuration bitstreams.

We also present an preliminary aging study in Section 3 to show that short-term NBTI aging is observable on FPGA devices, and this serves as a proof-of-concept for post-silicon configuration.

In Section 4, we present a case study of incorporating the delay logic in a non-trivial logic design and proving that it works as expected.

# 2. THE SR ARBITER

In this section, we will introduce a delay logic element with the combined functions of an arbiter and two NAND gates that compete for the control of the arbiter output, and show how it can be implemented on an FPGA platform with repeatable results.

D-type flip-flops (DFFs) have been used as arbiters in PUF designs in the past [17], where the path differences are designed to be offset by the preceding tuning circuits. Though easy to implement, arbiters implemented using DFFs have a built-in bias due to the fact that the Clock-to-Q and D-to-Q paths are different by design. Figure 3 shows an example of an asymmetrical DFF design. In contrast, SR latches, such as the one depicted in Figure 4 are intrinsically symmetrical, making it more suitable to function as an unbiased arbiter.

Furthermore, if a trigger signal arrives at both the S and the R input ports at exactly the same time, then the output of the arbiter solely depends on the relative speeds of the two NAND gates. This is an important aspect for the FPGA platforms, as identical logic paths are almost impossible to come by. Often the minor speed differences at the transistor

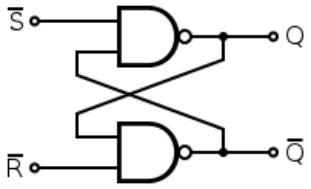



**Figure 4: An SR latch[50].**

level are trumped by the enormous routing delay differences. Therefore an SR latch can serve both as a good arbiter and delay racing element. We will refer to this design as the SR arbiter from here on.

## 2.1 Implementation

We would like to highlight the necessity of nearly identical logic paths in the implementation of arbiters. Many obstacles, such as unpredictable cell placement and routing, are to be tackled to ensure that the competing paths to the arbiters are as symmetrical as possible.

Unfortunately, the target platform (Xilinx Virtex5) does not offer any native SR latch logic cells. Therefore, an SR latch must be meticulously implemented by instantiating two lookup table (LUT) cells occupying the same logic slice (as NAND gates) and connecting them in combinational loops. By constraining the two LUT cells to the same logic slice, the combinational loop routing between the two LUTs are kept minimum and as close as possible.

Though the two NAND gates are identical in logic design, within each NAND gate, the two input-to-output paths are purposely designed to be different (as a matter of fact, the two paths cannot be designed to be identical due to the nature of SRAM-based LUT cells). While the Q-to-Qn and Qn-to-Q paths utilize the fastest path in each LUT (the highest address bit of the LUT), the S-to-Q and R-to-Qn paths utilize the slowest path in each LUT (the lowest address bit of the LUT). On the Virtex 5 FPGA where 6-input LUTs are available, the Q-Qn path only has one multiplexer, while the S/R-to-Q/Qn path has 6 layers of multiplexers. This arrangement echoes the desire to use the SR latch to measure the relative speed of the its NAND gates.

To further ensure that the signal transitions arrive as close as possible at the S and R input ports, strict relative placement constraints are used to enforce an in-slice floorplan as illustrated in Figure 5. The two DFFs in the middle of the slice (B-DFF and C-DFF) stores the Q and Qn results from the immediate NAND gates at B-LUT and C-LUT. The outer two DFFs (A-DFF and D-DFF) have very little clockskew between them, so they serve as precision triggers for the SR arbiter. To preserve local routing channels, the spare LUTs (A-LUT and D-LUT) are not allowed to be occupied by other functions. As a result, each SR arbiter occupies precisely one slice.

# 3. EXPERIMENT AND RESULTS

The experiment setup and results of the SR-latch based arbiters are described in this section.

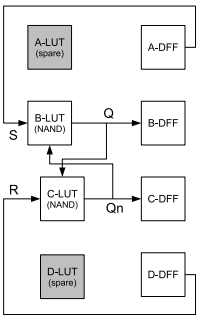



**Figure 5: SR latch mapped to Virtex5 CLB Slice.**

## 3.1 Experiment Setup

The main experiment of this paper carries two objectives. The first objective is to determine whether the process variations, manifested as differences in propagation delay between two FPGA chips, can be effectively detected by the SR arbiters. To achieve this, two SR arbiters with identical placement and routing are configured on two separate FPGA chips, and a trigger pulse is sent to the S and R ports of the arbiter. The output of the arbiter indicates whether the S path is faster than the R path, or the contrary. If the S path is consistently faster than the R path on both FPGAs, then the results will agree. However, if S path is faster than the R path on one FPGA, but slower on the other, then the arbiter results will disagree. Thus by comparing the arbiter results of two FPGAs, the propagation delay differences can be detected.

The second objective is to determine whether the effects of NBTI aging and recovery can be detected by the SR arbiters. We will exploit the frequency dependency of the NBTI aging effect by maintaining the S and R inputs of the arbiter at either logic one or zero for a prolonged period of time in the hope to slow down S or R path enough to change the outcome of the race between S and R paths. Following the aging process, the static S and R inputs are changed to toggling between one and zero in order to recover from the aging effect. The output of the SR arbiter is measured and compared after each aging and recovery cycle.

### *3.1.1 Environment*

The target platform used in this experiment is the Xilinx ML505 reference design board. The ML505 board is equipped with a Virtex-5 FPGA (v5lx50) that can be configured via a JTAG port. For the process variation objective, two such ML505 boards are used. The two FPGAs installed on the ML505 boards will be referred to as "FPGA-A" and "FPGA-B" from here on.

A desktop PC is used to configure and collect the results from the ML505 boards via RS-232 serial ports. All tests are conducted at ambient room temperature.

### *3.1.2 Baseline 64-Arbiter Array Design*

To facilitate the test objectives of the experiment, an FPGA design populated with an array of 64 SR arbiters is implemented. Figure 6 shows a functional diagram of the arbiter array design. Besides the array of the SR arbiters, a timing unit is used to generate the trigger pulses to the array of ar-

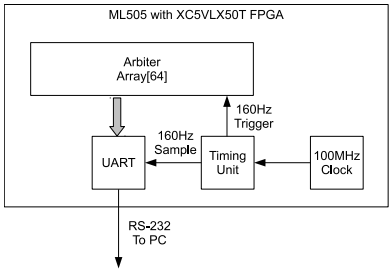



**Figure 6: 64-Arbiter array test setup.**

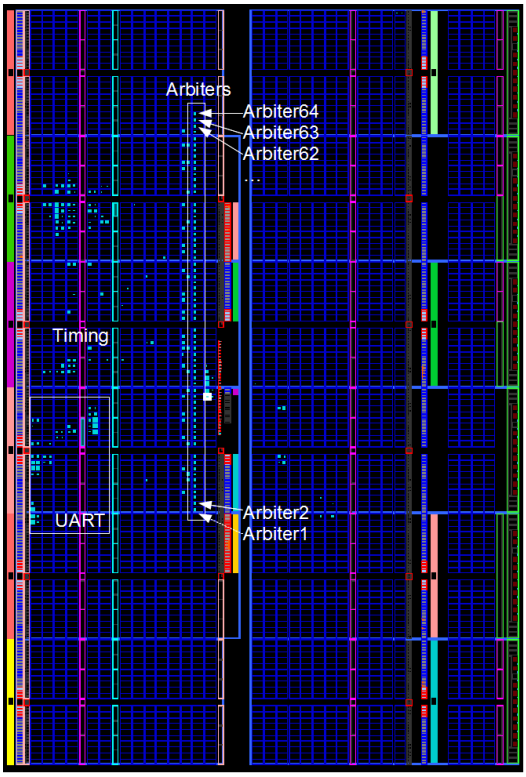



**Figure 7: 64-Arbiter array floorplan.**

biters at a rate of 160Hz. The results of the arbiters are then captured and transmitted by a UART encoder. A chip-level floorplan of the FPGA design is shown in Figure 7.

### 3.1.3 *Arbiter Result Scoring*

An arbiter race result is represented in a binary format. A one indicates that the S path of the arbiter has won the previous race, while a zero indicates that the R path of the arbiter has won. Each result set contains 64 such binary values. A total of 1000 such result sets are collected by the host PC to compute an average score for each of the arbiters, i.e. a score of 0.0 means that the S path of the arbiter has won the race 1000 times, and a score of 0.6 means that the S path has won 600 times while the R path has won 400 times, etc. This average score is a reflection of the expected output of the arbiter under the same condition, and will be referred to as the "arbiter score" from here on.

## 3.2 Results

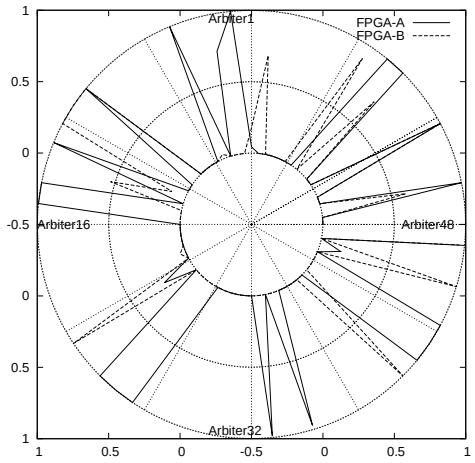



**Figure 8: Arbiter score comparison.**

### 3.2.1 *Process Variation*

The first objective of the experiment is to determine whether process variation between two FPGAs can be detected by the SR arbiters. The results are collected before the first NBTI aging cycle and reflect the starting state of the FPGAs. Figure 8 plots the arbiter scores of the two FPGAs in a polar form, with the solid lines corresponding to FPGA-A, and the dashed lines corresponding to FPGA-B. The radius of each point on the plot reflects the arbiter score. For aesthetic reasons the inner circle represents a score of 0.0, and the maximum score (the tips of the longest spikes) is 1.0.

It is evident from the lack of overlaps between the solid and dashed lines in Figure 8 that the arbiter scores of the two FPGAs are significantly different. Since the two FPGAs are configured from an identical bitstream file, and the two FPGAs are exposed to the same ambient environment, we conclude that the differences in the arbiter scores observed at the same arbiter site are due to process variations at the transistor level.

Though it is possible that the minor differences in voltage and junction temperatures may contribute to the differences in arbiter scores, we believe that such differences will only cause a systematic shift of the scores. The seemingly random nature of the arbiter scores is more consistent with the effects of process variation.

### 3.2.2 *NBTI Aging and Recovery*

The second objective of the experiment is to determine whether NBTI aging and recovery effects can be detected by the SR arbiters. For this purpose, two variations of the baseline design are created. The first variation of the design applies a static logic pattern at the S and R ports of all arbiters to stress the the S paths of the arbiters, while the second variation stresses the R paths. During an aging session, the FPGA-A is treated with the first variation (stressing S paths), and the FPGA-B is treated with the second variation (stressing R paths).

Each aging session lasts approximately 14 hours (overnight), followed by one or two days of recovery session where the

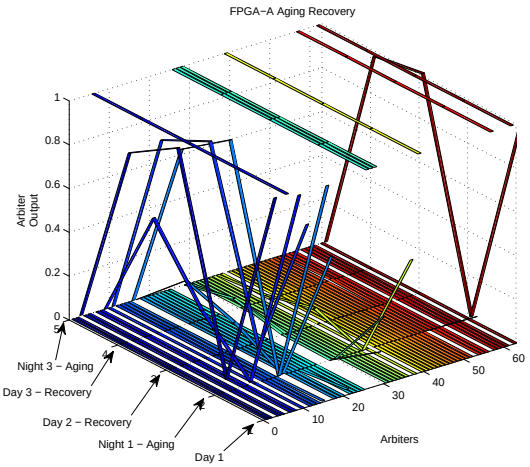



**Figure 9: Aging and Recovery of FPGA-A**

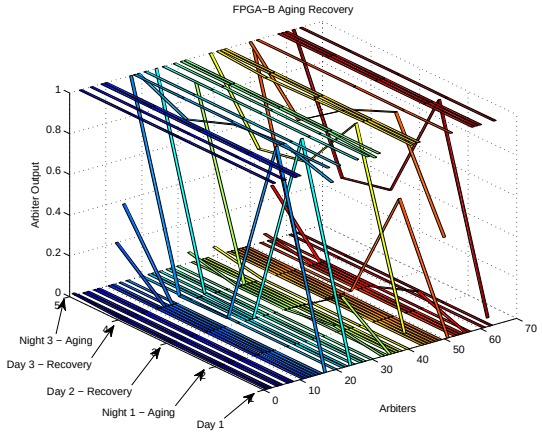



**Figure 10: Aging and Recovery of FPGA-B**

baseline design is loaded and the S and R ports are constantly toggling. The arbiter scores are recorded between aging and recovery sessions.

In Figure 9 and Figure 10, results from two aging and one recovery sessions are presented. The y (vertical) axis reflects the arbiter scores for each arbiter (x-axis). The z-axis is a series of sample times in chronological order. In Figure 9, the arbiter scores for some arbiters in FPGA-A are pushed towards 0.0 after each aging session. This is consistent with the fact that the S paths are being stressed during the aging session, and that an arbiter score closer to 0.0 reflects the higher likelihood of the R paths winning a race. After a recovery session, the arbiter scores move back towards 1.0, indicating that the NBTI stress on the S paths has receded, and that the S paths are more becoming likely to beat the R paths in a race.

The complete opposite takes place in Figure 10, where the R paths of the FPGA-B are stressed during the aging session. Note that on neither FPGA-A or FPGA-B did all arbiter scores change; in other words, the effect of a 14-hour continuous NBTI stress is not enough to change the outcome of the race between the S and R paths.

# 4. CASE STUDY: LEON3 PROCESSOR

The LEON3 [16] is a 32-bit general-purpose processor based on the SPARC V8 architecture. The complete VHDL source code of the LEON3 is released under the GNU Public License (GPL) for academic use. Due to its moderate size and design sophistication, we have chosen to use the LEON3 as a platform to demonstrate a simple anti-cloning mechanism using the SR arbiters.

## 4.1 Control Logic Shuffling

A simple way to implement anti-cloning is to use physical signatures of the device to scramble control signals in a controlled way. As an analogy to cryptography, the control signals are like messages that can be encrypted, transmitted, and decrypted to be used. The encryption key is the physical signature of the chip, in this case, the arbiter scores, and the decryption key is a function of the encryption key.

One method to scramble control signals is to shuffle logic. A two-input shuffler has two data inputs, two data outputs, and a control input. When the control input is set to logic zero, the two data outputs are exact copies of the two data inputs: data input A drives data output A, and data input B drives data output B. When the control input is set to logic one, the output orders are swapped: data input A drives data output B, and data input B drives data output A. A tree of such shuffling logic controlled by arbiter outputs can ensure that the design will only function properly on FPGA devices with a matching arbiter output signature.

## 4.2 Selection of Control Logic

At the core of the LEON3 processor is a seven-stage (fetch, decode, register read, execute, memory, exception, and register write) integer unit. Since all instructions must go through the integer unit, the control logic in the pipeline stages is an ideal place to employ the shuffler technique to prevent unauthorized design cloning.

To minimize the performance impact on the LEON3, the selection of the control logic for shuffling is made after first sorting the control logic paths by timing slack. The control logic with the largest amount of setup slack is least likely to become the critical path after shuffling logic is inserted. Evidently the ALU control signals (ALUOP) have the most slack among all control signals in the integer unit after examining the static timing analysis. What is also interesting about the ALU control signals is that they are highly critical to the normal operation of a processor. Therefore the ALU control signals are chosen as the target of the shuffling logic.

## 4.3 Static Shuffling Strategy

The basic strategy to organize static shuffling is described in this section. The strategy can be summarized in three words: shuffle, rotate, and reorder. These three words represent the three types of operations that can be performed on a group of control signals.

The shuffle operation mainly makes use of two-input shuf-

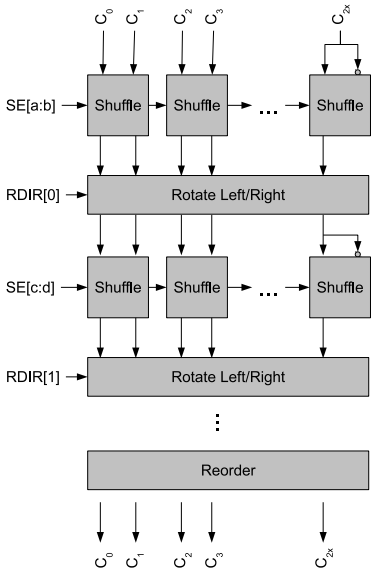



**Figure 11: Shuffle, Rotate, and Reorder**

flers. Each shuffler, depending on the control port ShuffleEn (SE), can either swap the the two input signals or do nothing to them. Shown as the first and third stages in Figure 11, two adjacent control signals are sent to a shuffler. The ShuffleEn signal is controlled by the output of an SR arbiter. If there is an odd number of control signals to shuffle, the last signal is sent to a shuffler with its inverted version, i.e. the shuffler can either send out the original or the inverted version of the signal, depending on the state of the ShuffleEn port. This type of shuffler is also referred to as an Inv-Shuffler.

The rotate operation always perform a single-bit rotation on the group of input signals. The direction of the rotation is decided by the RDIR input port, which is also driven by an SR arbiter.

The shuffle and rotation operations can be repeated as many times as possible, so long as the timing slack is sufficient. Figure 11 shows the shuffle and rotate interleaving each other to incrementally introduce entropy into the system.

The reorder operation is always the last stage before the control signals leave the shuffling tree. The purpose of having the reorder operation is to maintain the original order and polarity of the signals the same as they enter the shuffling tree. For example, if signal A and signal B are to be swapped position by a shuffler, then reorder stage must swap them back.

In a fully static shuffling tree, all the SE and RDIR inputs are controlled directly by the outputs of SR arbiters. At the compile/synthesis time, the reorder stage must know ahead of time what the SR arbiters will output to properly reorder the signals.

## 4.4 Metrics

The implementation result of the LEON3 with static shuffling will be judged on various metrics. The first metric is the functional correctness of the processor. The modified LEON3 processor must remain functional as before on the intended FPGA target, and any unauthorized copy must render the processor unusable. This is tested by loading the same LEON3 design on two ML505 boards. Only one of the ML505 board has the FPGA that matches the SR arbiter outputs expected by the reorder stage, i.e. this is the intended FPGA target. The other ML505 board, when loaded with the same design, should not function properly as a processor.

The second metric is the amount of area and performance overhead incurred by the modification. The area overhead is measured in number of LUTs used, and the performance overhead is measured by the minimum clock cycle, or the maximum clock speed.

The last metric is the most difficult to quantify: how easy can the anti-cloning scheme be attacked? To answer this question, we must make several assumptions about the attacks:

- The bitstream of the design can be recovered from the EEPROM storage.
- With the proper know-how, the bitstream can be converted into a netlist.
- The SR arbiters can be readily identified from the netlist. However, the outputs of the SR arbiters cannot be determined statically, nor can they be measured dynamically.
- The attacker therefore must resort to a brute force attack by guessing the outputs of the arbiters and verify the correctness of their guesses by running a netlist-level simulation.

Using the above assumptions, the difficulty of the attack then depends exponentially on the number of arbiters used, and linearly on the number of simulation cycles required to verify the correctness of the guess. Since the number of arbiters is limited and trivial to determine, we decided to use the number of simulation cycles to verify correctness as the metric against attacks. Finally, since the netlist simulation is typically simulated with a time resolution of 1ps ($10^{-12}$ seconds, the number of simulation cycles is essentially how many picoseconds the simulation must run before an attacker realizes that the guess is wrong.

## 4.5 Results

Figure 12 shows the floorplan of the modified LEON3 processor implemented on the v5lx50 FPGA. In the picture, DIV refers to the radix-2 integer divider, MUL refers to the dedicated multiplier. The integer unit, shown in orange, occupies most of the north side of the FPGA. The ALU control shufflers occupies a very small area (inside the white circle). A magnified view of the shufflers can be see in Figure 13, where the LUTs implementing the shuffle and rotate stages are pointed out. As a proof of concept, only three arbiter

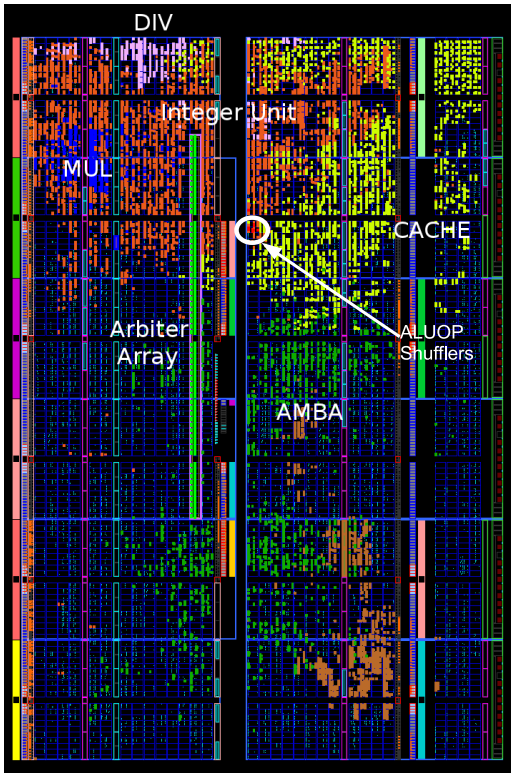



**Figure 12: LEON3 Floorplan**

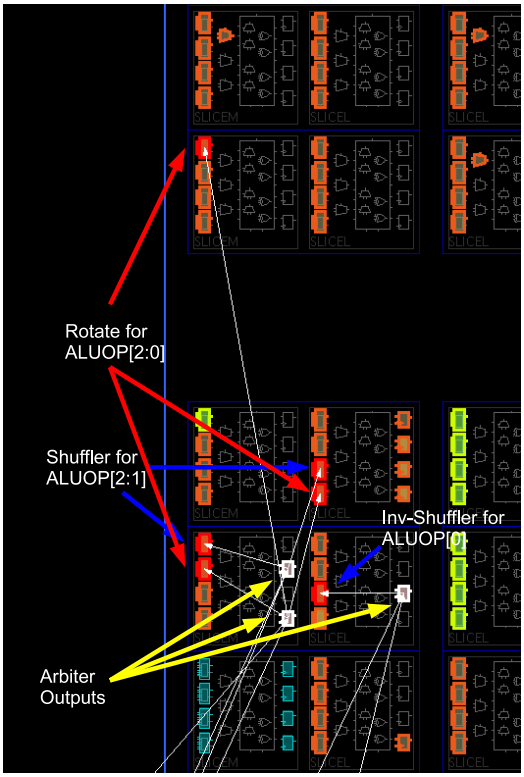



**Figure 13: ALUOP Shufflers**

are used to statically control the shuffle and rotation stages. The three DFFs (highlighted in white in Figure 13) store the arbiter outputs from the selected arbiters. The two DFFs on the left are expected to hold a value of one, and the DFF on the right is expected to hold a value of zero in order for the processor to function properly.

To answer the metric of functional correctness, the modified LEON3 design is loaded on two ML505 boards. A test program that finds all prime numbers less than 1000 is then loaded to the processor to run. On the FPGA with the correct arbiter outputs, the results are returned in exactly the same way as an unmodified LEON3 processor design would. On the FPGA with the incorrect arbiter outputs, the computation did not complete. Instead, the processor quickly traps to an error state. This proves that the modified LEON3 design passes the functional correctness metric.

To evaluate the area and performance overhead of the static shuffling and arbiter array, the FPGA mapping and static timing reports are examined and compared against the unmodified version of the LEON3 design. As shown in Table 1, the static shufflers and the arbiter array has a very small resource overhead, using only 0.4% additional LUTs and 6.3% additional DFFs. There is no timing overhead. In fact the modified LEON3 runs slightly faster than the unmodified LEON3.

| | Unmodified LEON3 | Modified LEON3 | Overhead |
|---|---|---|---|
| LUT Usage | 15,271 | 15,334 | 0.4% |
| DFF Usage | 7,649 | 8,132 | 6.3% |
| Clock Speed | 80.18MHz | 80.48MHz | -0.4% |

**Table 1: Area and Performance Overhead**

The last metric is the number simulation cycles the design must be simulated before an attacker can determine that the guessed arbiter outputs are incorrect. By simulating an instance of the modified LEON3 design with incorrect

arbiter outputs until the process traps to an error state, the answer is determined to be 6.2585 milliseconds, or 6.2585 billion simulation cycles at 1 picosecond per cycle.

# 5. CONCLUSIONS

We have introduced a new approach to IC Digital Rights Management by combining directed NBTI aging and delay logic. As a proof of concept, we have implemented a basic form of the delay logic on an FPGA platform, and shown that the delay logic measures the relative speed differences in competing logic paths due to process variation. Furthermore, we have shown that the delay logic responds correctly to NBTI aging and recovery cycles. We also presented a case study to use static arbiter outputs and shuffling logic to add anti-cloning protection to a non-trivial FPGA design. We showed that with very little resource and no clock speed overhead, the LEON3 processor design can be made clone-proof.

# 6. REFERENCES


[1] M. A. Alam. A critical examination of the mechanics of dynamic NBTI for PMOSFETs. In *IEDM*, page 346. IEEE, 2003.

[2] Y. Alkabani and F. Koushanfar. Active hardware metering for intellectual property protection and security. In *USENIX Security*, pages 291–306, 2007.

[3] Y. Alkabani, F. Koushanfar, N. Kiyavash, and M. Potkonjak. Trusted integrated circuits: A nondestructive hidden characteristics extraction approach. In *Information Hiding Workshop*, pages 102–117, 2008.

[4] Y. Alkabani, F. Koushanfar, and M. Potkonjak. Remote activation of ICs for piracy prevention and digital right management. In *ICCAD*, pages 674–677, 2007.

[5] R. Anderson and M. Kuhn. Tamper resistance: A cautionary note. In *Proceedings of the 2nd USENIX Workshop Electronic Commerce*, pages 1–11, 1996.

[6] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.

[7] A. Baumgarten, A. Tyagi, and J. Zambreno. Preventing IC piracy using reconfigurable logic barriers. *IEEE Design & Test of Computers*, 27(1):66–75, 2010.

[8] N. Beckmann and M. Potkonjak. Hardware-based public-key cryptography with public physically unclonable functions. In *Information Hiding Workshop*, pages 206–220, 2009.

[9] S. Blythe, B. Fraboni, S. Lall, H. Ahmed, and U. de Riu. Layout reconstruction of complex silicon chips. *IEEE Journal of Solid-State Circuits*, 28(2):138–145, 1993.

[10] M. Boehner. LOGEX - an automatic logic extractor form transistor to gate level for CMOS technology. In *Proceedings of the Design Automation Conference*, DAC '88, pages 517–522, 1988.

[11] L. Bossuet, G. Gogniat, and W. Burleson. Dynamically configurable security for SRAM FPGA bitstreams. In *Proceedings of the Parallel and Distributed Processing Symposium*, pages 146–153, April 2004.

[12] N. G. Bourbakis, A. Mogzadeh, and S. A. Mertoguno. Knowledge-based expert system for automatic visual VLSI reverse-engineering: VLSI layout version. *IEEE Transactions on Systems, Man and Cybernetics*, 32(3):428–436, 2002.

[13] D. Champagne, R. Elbaz, C. Gebotys, L. Torres, and R. B. Lee. Forward-secure content distribution to reconfigurable hardware. In *Reconfigurable Computing and FPGAs*, pages 450–455, 2008.

[14] T. Doom, J. White, A. Wojcik, and G. Chisholm. Identifying high-level components in combinational circuits. In *Great Lakes symposium on VLSI*, pages 313–318, 1998.

[15] S. Drimer. Volatile FPGA design security – a survey (v0.96), April 2008.

[16] J. Gaisler, E. Catovic, M. Isomaki, K. Glembo, and S. Habinc. GRLIB IP core user's manual. *Gaisler Research*, 2007.

[17] B. Gassend, D. Clarke, M. van Dijk, and S. Devadas. Silicon physical random functions. In *ACM Conference on Computer and Communications Security*, pages 148–160, 2002.

[18] M. C. Hansen, H. Yalcin, and J. P. Hayes. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *IEEE Design & Test*, 16(3):72–80, 1999.

[19] I. Hong and M. Potkonjak. Technique for intellectual property protection of DSP designs. In *International Conference on Acoustic, Speech, and Signal Processing*, pages 3133–3136, 1998.

[20] T. Kean. Method of using a mask programmed key to securely configure a field programmable gate array. United States Patent 7,240,218, 2001.

[21] M. Kearns and L. G. Valiant. Cryptographic limitations on learning boolean formulae and finite automata. In *ACM Symposium on Theory of computing*, pages 433–444, 1989.

[22] F. Koushanfar, G. Qu, and M. Potkonjak. Intellectual property metering. In *Information Hiding Workshop*, pages 81–95, 2001.

[23] J. Kumagai. Chip detectives. *IEEE Spectrum*, 37(11):43–48, 2000.

[24] J. Lach, W. Mangione-Smith, and M. Potkonjak. Fingerprinting digital circuits on programmable hardware. In *Information Hiding Workshop*, pages 16–31, 1998.

[25] J. Lach, W. Mangione-Smith, and M. Potkonjak. FPGA fingerprinting techniques for protecting intellectual property. In *Customom Integrated Circuits Conference*, pages 299–302, 1998.

[26] J. Lach, W. Mangione-Smith, and M. Potkonjak. Robust FPGA intellectual property protection through multiple small watermarks. In *Proceedings of the Design Automation Conference*, DAC '98, pages 831–836, 1999.

[27] J. Lach, W. Mangione-Smith, and M. Potkonjak. Fingerprinting techniques for field programmable gate array intellectual property protection. *IEEE Transactions on CAD*, 20(10):1253–1261, 2001.

[28] S. Mahapatra, M. A. Alam, P. B. Kumar, T. R. Dalei, D. Varghese, and D. Saha. Negative bias temperature

instability in CMOS devices. *Microelectronic engineering*, 80:114–121, 2005.

[29] M. Majzoobi, F. Koushanfar, and M. Potkonjak. Techniques for design and implementation of secure reconfigurable PUFs. *ACM Trans. Reconfigurable Technol. Syst.*, 2:5:1–5:33, March 2009.

[30] M. Majzoobi, F. Koushanfar, and M. Potkonjak. Trusted design in FPGAs. In *Introduction to Hardware Security and Trust*, pages 195–230. Springer, 2011.

[31] J. T. McDonald, Y. C. Kim, and M. R. Grimaila. Protecting reprogrammable hardware with polymorphic circuit variation. In *Cyberspace Research Workshop*, pages 63–78, 2009.

[32] J. T. McDonald, E. D. Trias, Y. C. Kim, and M. R. Grimaila. Using logic-based reduction for adversarial component recovery. In *ACM Symposium on Applied Computing*, pages 1993–2000, 2010.

[33] R. Nakagaki, Y. Takagi, and K. Nakamae. Automatic recognition of circuit patterns on semiconductor wafers from multiple scanning electron microscope images. *Measurement Science and Technology*, 21(8):085501, 2010.

[34] R. Nijssen and J. Jess. Two-dimensional datapath regularity extraction. In *ACM/SIGDA Internation symposium on Physical Design*, pages 42–47, 1997.

[35] J. D. Parham, J. T. McDonald, M. R. Grimaila, and Y. C. Kim. A java based component identification tool for measuring the strength of circuit protections. In *Proceedings of the Sixth Annual Workshop on Cyber Security and Information Intelligence Research*, pages 26:1–26:4, 2010.

[36] G. Qu and M. Potkonjak. *Intellectual Property Protection in VLSI Design Theory and Practice*. Springer, 2003.

[37] D. S. Rao and F. J. Kurdahi. On clustering for maximal regularity extraction. *IEEE Transactions on CAD*, 12(8):1198–1208, 1993.

[38] R. L. Rivest and R. E. Schapire. Inference of finite automata using homing sequences. In *ACM Symposium on Theory of computing*, pages 411–420, 1989.

[39] J. Roy, F. Koushanfar, and I. Markov. EPIC: Ending piracy of integrated circuits. In *DATE*, pages 1069–1074, 2008.

[40] U. Rührmair, F. Sehnke, J. Sölter, G. Dror, S. Devadas, and J. Schmidhuber. Modeling attacks on physical unclonable functions. In *ACM Conference on Computer and Communications Security*, pages 237–249, 2010.

[41] D. Saab, V. Nagubadi, F. Kocan, and J. Abraham. Extraction base verification method for off the shelf integrated circuits. In *Quality Electronic Design*, pages 396–400. ASQED 1st Asia Symposium, 2009.

[42] S. Sirowy, G. Stitt, and F. Vahid. C is for circuits: capturing FPGA circuits as sequential code for portability,. In *ACM/SIGDA Symposium on Field programmable gate arrays*, pages 117–126, 2008.

[43] R. Torrance and D. James. Reverse engineering in the semiconductor industry. In *Customom Integrated Circuits Conference*, pages 429–436. IEEE, 2007.

[44] R. Torrance and D. James. The state-of-the-art in ic reverse engineering. *CHES, ser. Lecture Notes in Computer Science*, 5747:363–381, 2009.

[45] S. Trimberger. Trusted design in FPGAs. In *Proceedings of the Design Automation Conference*, DAC '07, pages 5–8, 2007.

[46] S. Wei, F. Koushanfar, and M. Potkonjak. Integrated circuit digital rights management techniques using physical level characterization. In *ACM Workshop on Digital Rights Management*, pages 3–14, 2011.

[47] S. Wei, A. Nahapetian, and M. Potkonjak. Robust passive hardware metering. In *ICCAD*, 2011.

[48] J. L. White. *Candidate subcircuit enumeration for module identification in digital circuits*. PhD thesis, Michigan State University, East Lansing, MI, 2000.

[49] J. L. White, A. S. Wojcik, M.-J. Chung, and T. E. Doom. Candidate subcircuits for functional module identification in logic circuits. In *Great Lakes Symposium on VLSI*, pages 34–38, 2000.

[50] Wikipedia. Flip-flop (electronics) — wikipedia, the free encyclopedia, 2011. [Online; accessed 26-September-2011].