

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/325491561>

ATPG-based cost-effective, secure logic locking

Conference Paper · April 2018

DOI: 10.1109/VTS.2018.8368625

CITATIONS

70

READS

327

4 authors:



Abhrajit Sengupta

New York University

28 PUBLICATIONS 957 CITATIONS

[SEE PROFILE](#)



Mohammed Nabeel

New York University Abu Dhabi

35 PUBLICATIONS 717 CITATIONS

[SEE PROFILE](#)



Muhammad Yasin

Texas A&M University

43 PUBLICATIONS 2,687 CITATIONS

[SEE PROFILE](#)



Ozgur Sinanoglu

New York University Abu Dhabi

349 PUBLICATIONS 8,296 CITATIONS

[SEE PROFILE](#)

ATPG-Based Cost-Effective, Secure Logic Locking

Abhrajit Sengupta[†], Mohammed Nabeel[‡], Muhammad Yasin[†], and Ozgur Sinanoglu[‡]

[†] Tandon School of Engineering, New York University, New York, USA

[‡] New York University Abu Dhabi, Abu Dhabi, United Arab Emirates

{as9397, mtn2, yasin, ozgursin}@nyu.edu

Abstract—The globalization of IC supply chain lead to the emergence of hardware security threats such as IP piracy, reverse engineering, overbuilding, and hardware Trojans. Among the techniques developed to mitigate these threats, logic locking offers the most versatile protection and is being actively researched. The most recent locking technique SFLL thwarts with provable and quantifiable security all the state-of-the-art attacks including SAT, AppSAT, and the removal attack. However, the implementation cost of SFLL can sometimes be prohibitive, as it lacks an automated framework that explores cost-effective implementation options. In this paper, we show how VLSI testing principles and tools can be adopted to automate critical steps in SFLL and minimize its cost. We propose “SFLL-fault” that utilizes fault injection driven synthesis to efficiently explore design options and ATPG to assess security levels. Our experimental results confirm the efficacy of our strategy; SFLL-fault can reduce the implementation cost by 35% compared to SFLL without compromising security.

Index Terms—IP piracy, reverse engineering, logic locking, ATPG, VLSI testing

I. INTRODUCTION

In today’s ever-growing market of digital electronics, threats such as IP piracy, hardware Trojans, overbuilding, and reverse-engineering are becoming an increasing concern for military and commercial organizations [1]. With globalization of IC fabrication, many companies send their intellectual property (IP) to off-shore foundries for cost-effective access to advanced technology nodes. This way the companies are forced to share their valuable IP with potentially *untrusted* parties, exposing it to the aforementioned threats. In fact, it was estimated that due to IP piracy the semiconductor industry loses \$4 billion annually [2].

Design-for-Trust (DfTr) techniques. To protect IP from unauthorized access, several techniques such as IC metering [3], watermarking [4], camouflaging [5], split manufacturing [6], physically unclonable function (PUF) [7], and logic locking [8]–[14] have been proposed. Due to its simplicity and effectiveness, logic locking has gained significant popularity in the research community. It can protect against an attacker located anywhere in the supply chain whereas other DfTr techniques protect only against a subset of malicious entities.

Logic locking. Logic locking was first introduced in EPIC [8]. It is effected at the hardware level by inserting extra *key-gates* in the original netlist to lock its functionality. The *secret key* is stored in a *tamper-proof* memory [15] on the chip as shown in Fig. 1. Without the secret key loaded onto its memory, the IC is rendered practically useless as it will produce corrupted outputs. The threat model for

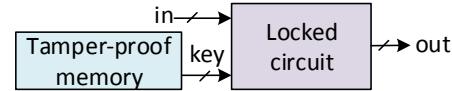


Fig. 1. The output of a logic locked circuit is a function of the key inputs and the primary inputs. The key inputs are driven from a tamper-proof memory.

logic locking is illustrated in Fig. 2. Several other works followed [9], [10] improving the security of logic locking. Nonetheless, a lethal attack based on Boolean satisfiability (SAT) broke all the existing logic locking techniques [11] then. To thwart SAT attack, SARLock [12] and Anti-SAT [16] were proposed. However, both SARLock and Anti-SAT fall short against removal [14] and bypass attack [17]. Later, TTLock was proposed to protect against such attacks [18]. The idea is to make the on-chip hardware implementation different from that of the original design (and thus, hidden/protected from an attacker). The modified design implemented in hardware produces different outputs compared to the original design for a selected set of input patterns/cubes known, accordingly, as *protected input patterns/cubes*¹. This set of protected input patterns constitutes the secret key. Consequently, a restoration logic, driven by this key, is also implemented on-chip to restore the original functionality for the protected cubes. However, TTLock protects only one input cube, which results in low output corruptibility and is vulnerable to approximate attacks such as AppSAT [13].

The baseline: SFLL. Recently, stripped functionality logic locking (SFLL) was proposed to generalize TTLock, protecting a large number of input patterns using a security-aware CAD framework [19]. One variant of SFLL, referred to as SFLL-flex, modifies the design based on the designer-specified protected patterns using a simulated annealing approach [19]. Though this might work well for a certain class of applications, simulated annealing is extremely slow in practice and may run into scalability issues for general applications.

Note that none of the previous work on logic locking accounts for the cost of a tamper-proof memory during the synthesis process, which might have a significant impact on the area footprint of the design leading to high overhead². Moreover, most existing SAT resilient logic locking techniques rely solely on a designer to specify the security-critical parts

¹Input patterns may be compactly represented using input cubes, which contain don’t care entries (x’s). We use the two terms interchangeably.

²As part of standard power up sequence, the content of non-volatile tamper-proof memory is shifted to chain of flops, thereby, not impacting the timing of a design afterwards.

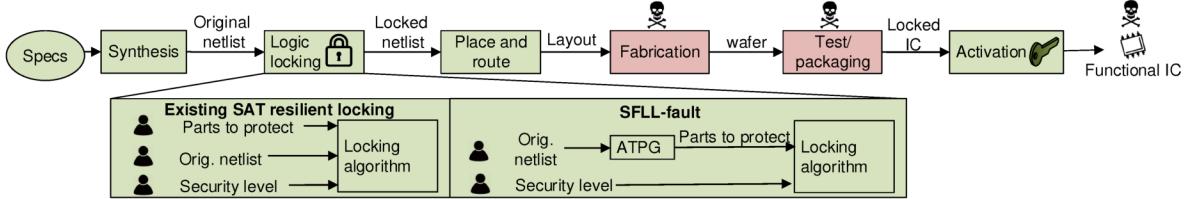


Fig. 2. Threat model for logic locking. The skulls denote the untrusted stages. The inset highlights that the proposed SFLL-fault technique uses ATPG to determine which parts of the netlist are cost-effective candidates for protection; otherwise, the designer is burdened with providing these security specifications.

of a design or the protected input patterns(s) [19]. This leads to additional work for the designers, who are already overburdened with hard deadlines and frequent engineering change orders in design specifications.

Contribution. The discussion above asserts that in order to minimize reliance on the designers, it is imperative to develop design tools that automate the additional security related design decisions. The objective of this work is to provide an automated framework that enables the designers to both 1) identify and modify the parts of a design that constitute cost-effective candidates to be protected using functionality-strip operation, and 2) compute the associated protected patterns. It is crucial that such a framework is built around well-established EDA tools and can be easily integrated with existing IC design flow with minimal changes. Accordingly, our strategy for selecting protected patterns is to leverage the principles of VLSI testing and make use of automatic test pattern generation (ATPG) tools. The contributions of our work are:

- 1) We develop a fault-injection based heuristic to select the protected input patterns minimizing the overhead.
- 2) We present a scalable, comprehensive SFLL-fault optimization framework that also accounts for the cost of tamper-proof memory. We achieve on average 35% improvement in area overhead when compared to the baseline SFLL-flex.
- 3) We conduct a detailed security analysis of the proposed SFLL-fault against the state-of-the-art attacks.

II. PRELIMINARIES

Before delving into further details, we briefly discuss the state-of-the-art attacks against logic locking and elucidate in particular stripped-functionality logic locking (SFLL). A summary of all the existing locking techniques is shown in Table I.

A. Attacks on logic locking

1) *SAT attack:* SAT attack is the most effective attack against logic locking to date. In each iteration, the attack finds an input pattern, called a *distinguishing input pattern* (DIP), which divides the key space into two classes; correct and incorrect. A DIP is an input pattern for which the circuit produces different outputs for two different sets of keys. The attack iteratively divides the key space by finding a DIP in each iteration. It terminates successfully when the incorrect search space is pruned, that is, all incorrect key values have been eliminated. The computational effort of the attack is

TABLE I
COMPARISON OF LOGIC LOCKING TECHNIQUES WRT ATTACK RESILIENCE AND OPTIMAL OVERHEAD. ✓ DENOTES PRESENCE OF A FEATURE.

Feature	RLL [8]	FLL [10]	SLL [9]	AntiSAT [16]	SARLock [18]	SFLL [19]	Proposed SFLL-fault
SAT resilient [11]	x	x	x	✓	✓	✓	✓
Removal resilient [14]	✓	✓	✓	x	x	✓	✓
AppSAT resilient [13]	x	x	x	x	x	✓	✓
Cost-effective	x	x	x	x	x	x	✓

determined by the number of DIPs required by the attack to successfully find the correct key.

2) *Removal attack:* Removal attack exploits structural traces to identify, and ultimately remove the protection logic [14]. Both SARLock and Anti-SAT leave the original circuit unchanged; the protection circuitry is added merely as a wrapper around the original circuit. SARLock hardcodes the secret key in the mask logic, which can be easily extracted by tracing the fan-outs of the key inputs. The symmetric construction of Anti-SAT block allows the identification of the protection logic through signal probability analysis. Even when the symmetry is dissolved using additional obfuscation, Anti-SAT remains vulnerable to slightly more sophisticated AppSAT guided removal attack [14].

3) *Bypass attack:* The principle of Bypass attack is to select a random key as the correct key, find the patterns that lead to an incorrect output for that particular key, and finally construct a bypass circuit to rectify the incorrect circuit output [17]. The complexity of the Bypass attack is dictated by the size of the bypass circuit. Both SARLock and Anti-SAT are vulnerable to the Bypass attack as well.

B. Stripped-functionality logic locking

SFLL comes in two flavors: 1) SFLL-HD that is suited to protect an arbitrarily large number of cubes, and 2) SFLL-flex that is suitable for applications where the cubes-to-be-protected can be compactly represented [19]. SFLL (and also TTLock, which is a special case of SFLL-HD) introduced the notion of *protected input cubes* to achieve resilience against removal and bypass attacks. It “strips” part of the functionality from the design and implements this modified functionality in hardware; protected input cubes are those that differentiate the outputs of the original and the modified designs. A restoration unit is also implemented on chip, canceling the errors produced by the protected input patterns only when the correct key is in place. Removal attack thus fails as it only delivers a netlist with stripped functionality. While TTLock thwarts removal attacks by introducing only one protected cube [18], SFLL allows for a larger number of protected cubes,

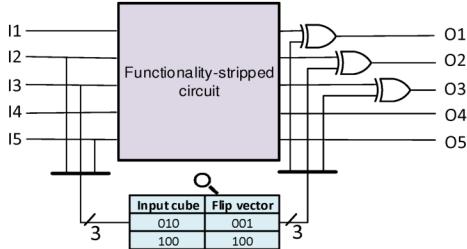


Fig. 3. Architecture for SFLL-flex. The protected input cubes are stored in an LUT along with flip vectors that restore the stripped functionality. Source: [19].

and thus a controllably stronger protection against removal attacks.

As opposed to SARLock and Anti-SAT, SFLL only focuses on the average number of keys eliminated by a DIP and not the worst case number. As such, SFLL allows trade-offs between SAT attack resilience and removal attack resilience. Note that if the DIP corresponds to the protected input cube, then SAT can immediately recover the correct key by eliminating all the incorrect keys in a single iteration. Thus, the resilience against a SAT attack is defined by the probability of finding such a protected input cube by the attacker [18], [19].

SFLL-flex. SFLL-flex strips functionality of a design based on either randomly or designer selected protected input cubes. As demonstrated in Fig. 3, it comprises a functionality-stripped circuit (FSC) and a restore unit, integrated using XOR gates. The restore unit is driven from a tamper-proof look-up table (LUT) that stores the compressed input cubes along with the flip vectors. The stripped functionality is thus restored for the protected input patterns, which act as “keys” of SFLL-flex.

III. PROPOSED SFLL-FAULT

In this section, we illustrate our technique with necessary details. SFLL-flex compactly represents the randomly-generated or designer-selected protected input cubes by first compressing them. It then relies on simulated annealing driven logic synthesis to lower the cost of stripping functionality based on the resulting set of compressed cubes. However, there are two pitfalls of this technique. First, simulated annealing is extremely slow in practice and rarely used for modern, large designs. Second, while SFLL-flex scales well for a small class of applications where protected input cubes are few and easy to extract, it may not scale well for generic applications where cube compression on a poorly selected initial set of protected cubes leads to high overhead in terms of area, power and timing. A question that naturally follows is that can we develop a heuristic to intelligently select the protected patterns such that the overall cost of implementation is minimized without compromising its security? In this paper, we present a new heuristic which is not only suitable for generic applications, but also allows for a cost-effective implementation of the proposed SFLL technique.

To identify the most cost-effective parts to strip and the list of protected input cubes associated with it, we leverage well-understood concepts and effective tools from VLSI testing.

We inject easy-to-enumerate faults³, one at a time, to strip functionality from a design. Fault injections are traditionally used to model the behavior of physical defects in higher levels of abstraction. In this context, we use faults to enumerate our options to strip functionality from a design. We use ATPG tools to identify the list of patterns that detect a fault; these patterns are indeed the protected (or equivalently “failing”) input cubes for the design modified by injecting the fault. This way, we can assess the implementation cost and security levels associated with different design modification options and choose the least-cost implementation with sufficient security.

Example. Let us consider the example of c17 ISCAS benchmark circuit shown in Fig. 4a. A stuck-at-zero fault was injected at node N5 at the output of the NAND gate. This reduces the circuit to that in Fig. 4b. The two circuits differ at the output O23 for the cubes listed in Fig. 4c. It is evident from the figure that the number of gates gets reduced by two for the FSC. Next, a restore circuit is constructed from these failing cubes to recover the original functionality of the circuit as shown in Fig. 4d.

A. Architecture.

The architecture consists of a tamper-proof [15] look-up table (LUT) and a *restore* unit as shown in Fig. 4d.

Cost. The cost⁴ of implementation is dictated by the size of the look-up table that is used to store the protected input cubes, in addition to the comparators and XOR gates inserted at the outputs. Suppose an injected fault causes f outputs to fail where output o_1 fails for k_1 patterns each with n_1 bits, o_2 fails for k_2 patterns each with n_2 bits and so on. This implies that the number of bits to be stored in the LUT is $n_1 \cdot k_1 + n_2 \cdot k_2 + \dots + n_f \cdot k_f$ and its associated cost is:

$$cost_{lut} = (n_1 \cdot k_1 + n_2 \cdot k_2 + \dots + n_f \cdot k_f) \cdot \alpha_{lut} \quad (1)$$

where α_{lut} denotes the per-bit cost of a tamper-proof LUT.

The restore unit consists of f comparators of size n_1, n_2 , and n_f , respectively, along with f XOR gates at the failing outputs. Thus, the cost of the restore circuit is given by:

$$cost_{rest} = (n_1 + n_2 + \dots + n_f) \cdot \alpha_{comp} + f \cdot \alpha_{xor} \quad (2)$$

where α_{comp} and α_{xor} denotes the cost of one bit comparator and a 2-input XOR gate, respectively. Thus, the total cost of the LUT and the restore unit is $cost_{lut} + cost_{rest}$.

B. Optimization framework

Suppose the cost of the FSC is denoted by $cost_{sf}$, then the problem reduces to the following optimization problem:

$$\begin{aligned} &\text{minimize: } cost_{sf} + cost_{lut} + cost_{rest} \\ &\text{such that } \max\{n_i - \log_2 k_i\} \geq s; 1 \leq i \leq f \end{aligned} \quad (3)$$

where s is the desired security level and $\max\{n_i - \log_2 k_i\}$ is the security level attained against SAT attack, explained later

³In this work, we use stuck-at faults, while we note that the fault model can be extended.

⁴Here, we focus on area as our cost metric. Similarly, we can optimize for other metrics such as power or timing of a design.

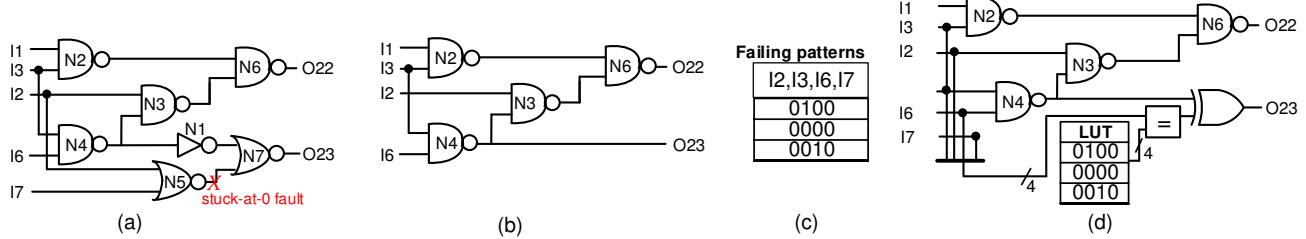


Fig. 4. SFLL-fault applied on c17 circuit. (a) Original circuit. (b) FSC after fault injection at N5. (c) List of failing patterns at O23. (d) Locked circuit including restore unit.

Algorithm 1: Fault injection based synthesis algorithm

```

Input : Original netlist  $N$ , Security level  $s$ 
Output: Optimized netlist  $N_{opt}$ , Set of failing patterns  $V_{opt}$ 
1  $N_{opt} \leftarrow \phi$ 
2  $V_{opt} \leftarrow \phi$ 
3  $X \leftarrow \text{parse\_list\_of\_nodes}(N)$ 
4  $cost_{min} \leftarrow \infty$ 
5 for each  $x \in X$  do
6    $N_{sf} \leftarrow \text{inject\_fault}(N, x)$ 
7    $V_{sf} \leftarrow \text{get\_failing\_patterns}(N_{sf}, N)$ 
8   if  $\text{security\_level}(V_{sf}) < s$  then
9     | select_random_pattern( $N_{sf}$ ,  $s$ )
10    end
11    $cost_{new} \leftarrow cost_{sf} + cost_{lut} + cost_{rest}$ 
12   if  $cost_{new} < cost_{min}$  then
13     |  $cost_{min} \leftarrow cost_{new}$ 
14     |  $N_{opt} \leftarrow N_{sf}$ 
15     |  $V_{opt} \leftarrow V_{sf}$ 
16   end
17 end

```

in this section. Thus, essentially we want to select the FSC which has the least overall implementation cost, accounting also for the restore logic. The synthesis algorithm to find such an optimized FSC is depicted in Algorithm 1.

The input to the algorithm is the original netlist N and a desired security level s . We inject a stuck-at fault at every node in the netlist N and list its corresponding failing patterns. These patterns constitute the protected input patterns for which the circuit produces an incorrect output without the restore operation. Next we compute the security level delivered by the protected input cubes of the fault from Eq. 3. If the desired security level is not achieved, then a random pattern, satisfying the security requirement, is added to the set of protected patterns. The algorithm terminates when all the faults in the design have been explored and produces the overall optimal circuit in terms of area.

C. Security analysis of SFLL-fault

In this section, we analyze the security of our technique against state-of-the-art attacks against logic locking.

SAT attack resilience. We prove the following claim.

Claim 1. SFLL-fault achieves a security of $\max\{n_i - \log_2 k_i\}, 1 \leq i \leq f$ against the SAT attack.

Proof. Recall from Section II-B that the resilience against a SAT attack is defined by the probability of an attacker to identify a protected input cube. However, as the attacker has no information about the protected cubes, he/she can not do any better than a random guess. Note that in our case, finding one

of the protected cubes does not lead to the recovery of the other protected cubes. Thus, in order to recover the full functionality of the circuit, the attacker is forced to find all the input cubes, implying the SAT attack would be successful if and only if it can find all the DIPs corresponding to the protected cubes. Thus, the security is dictated by the DIP having the minimum probability of being detected. Suppose, V_i denotes the set of protected cubes for output o_i and v denotes a randomly selected DIP.

Thus, the probability of success for SAT attack is given by:

$$\begin{aligned} \Pr[\text{SAT success}] &= \min\{\Pr[p \in V_i]\}, 1 \leq i \leq f \\ &= \min\left\{\frac{k_i}{2^{n_i}}\right\}, 1 \leq i \leq f \end{aligned} \quad (4)$$

From [19], a logic locking technique is called λ -secure if the probability of finding a protected pattern is not greater than $\frac{1}{2^\lambda}$. In our case, the probability that p belongs to a set V_i (and leads to the recovery of key) is $\frac{k_i}{2^{n_i}}$; the security level attained for the output o_i is $n_i - \log_2 k_i$. Using Eq. 4, the security level for the locked circuit is:

$$s = \max\{n_i - \log_2 k_i\}, \quad 1 \leq i \leq f$$

□

Removal attack resilience. Similar to SFLL-flex, SFLL-fault thwarts the removal attack by making the on-chip FSC implementation different from the original circuit for k input patterns, where $k = \max\{k_1, k_2, \dots, k_f\}$. While the current SFLL-fault framework optimizes for area, it can also be extended to optimize for removal attack resilience by simply maximizing k .

Bypass attack resilience. Bypass attack terminates by running only one iteration of the SAT attack and outputs an incorrect key. It then constructs a wrapper circuitry around the locked netlist to correct the output for that particular incorrect key [17]. However, for our design, the attacker fails to extract precise information about the protected input cubes, even if he/she creates a bypass circuit around the locked netlist; the circuit would still fail for the protected cubes, delivering resiliency against Bypass attack.

IV. RESULTS

A. Experimental setup

In this section, we present detailed experimental results to establish the effectiveness of the proposed SFLL-fault. All the experiments have been carried out on a 128-core Intel Xeon

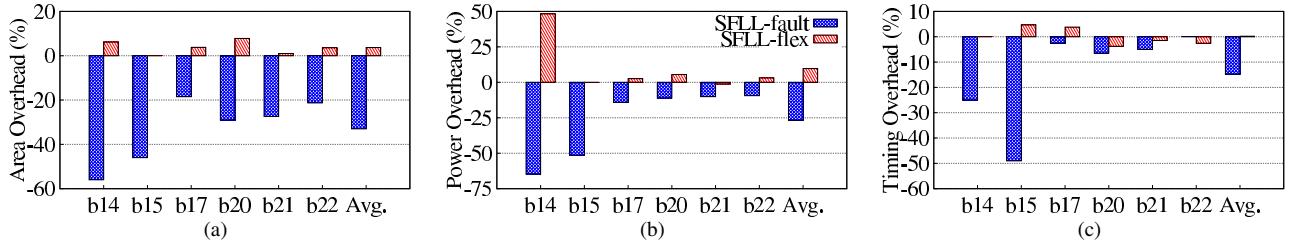


Fig. 5. Overhead comparison of FSC for SFLL-fault and SFLL-flex. (a) Area, (b) Power, and (c) Timing. Note that negative numbers imply gain in terms of cost, underscoring efficacy of SFLL-fault.

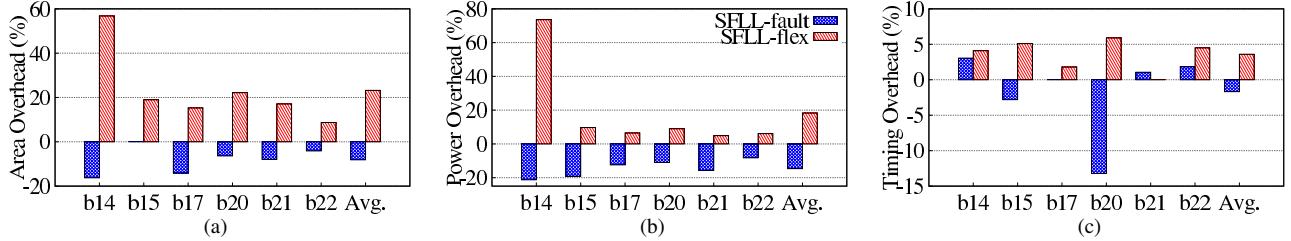


Fig. 6. Overhead comparison of locked circuits for SFLL-fault and SFLL-flex for $s=128$. (a) Area, (b) Power, and (c) Timing.

TABLE II
DETAILS OF ITC'99 BENCHMARK CIRCUITS

Benchmark	# Inputs	# Outputs	# Gates
b14	277	299	9,767
b15	485	519	8,367
b17	1452	1,512	30,777
b20	522	512	19,682
b21	522	512	20,027
b22	767	757	29,162

processor running at 2.2 GHz with 256 GB of RAM. All the codes have been compiled using GCC compiler 4.4.7 on Cent OS 6.9 (Final). We implemented our technique on ITC'99 benchmark circuits whose details are provided in Table II. We used Synopsys Design Compiler along with Global Foundries (GF) 65nm LPe library to compute the area, power, and timing overhead numbers. We assume that the tamper-proof LUT occupies $1.46\mu\text{m}^2$ per bit, the same as the area of a single bit of one-time programmable memory in GF 65nm LPe technology. Atalanta-M ATPG tool is used to compute the failing patterns for the fault-injected circuit⁵ and the results were cross verified using the Cadence Conformal for logical equivalence.

B. Overhead analysis

Functionality-stripped circuit. The basic difference between SFLL-flex and SFLL-fault is the underlying method used to effect the functionality-strip operation; SFLL-flex uses simulated annealing, whereas SFLL-fault relies on fault injection. To demonstrate the cost-effectiveness of the functionality-strip operation in SFLL-fault as compared to that in SFLL-flex, we report the area, power, and timing overhead of the FSC circuit for both techniques. Note that negative numbers indicate savings compared to the original design.

As shown in Fig. 5a, SFLL-fault always achieves a significant reduction in area as compared to that of SFLL-flex. On average, the area overhead incurred by SFLL-fault is

only -33.0% compared to 3.7% of SFLL-flex. Thus, the FSC obtained using fault injection occupies 33.0% lesser chip area than the original one. These savings in area can be attributed to the autonomy given to the ATPG tool in deciding both 1) the part(s) of the design to be protected and 2) the set of protected patterns.

As shown in Fig. 5b, the power overhead is -26.8% for SFLL-fault and 9.7% for SFLL-flex on average. As expected the power overhead correlates well with the area overhead. As illustrated in Fig. 5c, SFLL-fault is also able to achieve an average reduction of -14.8% for timing. Overall, SFLL-fault significantly outperforms SFLL-flex in terms of area, power, and timing savings for the FSC circuit, underscoring the benefits of delegating critical design decisions to carefully crafted EDA algorithms.

Logic-locked circuit. Next, we report the overhead of the overall logic-locked circuits with $s=128$ bits of security in Fig. 6. SFLL-fault is geared towards minimizing the overall cost which comprises three components listed in Eq. 3. In SFLL-flex, the FSC circuit incurs a small area overhead of about 3.7%. Upon addition of the overhead of LUT and restore unit (comparators and XORs), the overall overhead is 23.3%, on average. In contrast, the overall overhead of SFLL-fault is only -8.2% on average. SFLL-fault achieves significant savings in the FSC area which can offset the overhead of the restore unit and LUT, leading to a notable area savings for the locked circuit. Similarly, the average power overhead of the locked circuits is 18.3% and -14.6% for SFLL-flex and SFLL-fault, respectively. Also, SFLL-fault improves timing by -1.7% compared to 3.4% for SFLL-flex. However, for some of the circuits SFLL-fault does incur small timing overhead. This can be attributed to the fact that SFLL-fault protects a larger part of the design compared to SFLL-flex. This inadvertently requires a comparatively larger restore unit. **Runtime.** In SFLL-fault, the fault injections are independent of one another, allowing for a parallelized implementation. For faster, exhaustive processing of faults in large circuits such as

⁵Note that Tetramax ATPG tool could not be used as it does not return all the failing patterns for a fault.

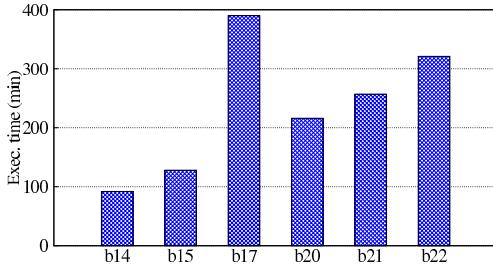


Fig. 7. Execution time for SFLL-fault for different benchmark circuits.

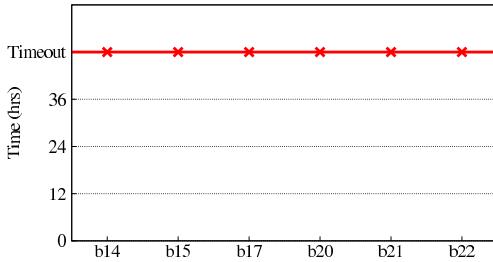


Fig. 8. Timeout for SAT attack for all the locked benchmark circuits.

b17, we simulate up to 100 faults in parallel. As reported in Fig. 7, the execution time of SFLL-fault is in the order of a few hours even for large circuits with >30K gates. The majority of the time is spent on optimizing the FSC using Synopsys DC compiler.

C. Security analysis

Note that the overall architecture of SFLL-fault and the security properties of SFLL-fault are the same as SFLL-flex; SFLL-flex differs mainly in the implementation methodology. As such, the security achieved against the other attacks such as AppSAT would be the same as that achieved by SFLL-flex. Nevertheless, we validate the security of SFLL-fault by launching SAT attack on circuits locked with SFLL-fault based on 128-bit security. Note that each experiment is repeated 10 times to improve the statistical significance of the results. Fig. 8 shows that the attack fails to terminate within the specified time limit of 48 hours for every trial.

D. Discussion

Comparison with SFLL-HD. Although we cannot directly compare SFLL-fault to SFLL-HD that uses a purely combinational logic for restore unit, we present some qualitative differences here. SFLL-HD allows to protect a large number of patterns; however, the functionality-strip operation in SFLL-HD relies on existing logic synthesis tools, which may not be security-aware [19]. Moreover, SFLL-HD has a significantly higher overhead compared to SFLL-fault; the functionality-strip operation itself incurs significant overhead. As an example, for the b14 circuit, SFLL-HD incurs an area overhead of 43% compared to only -16.2% for SFLL-fault.

V. CONCLUSION

We presented an automated synthesis framework by leveraging the principles of VLSI testing towards developing a low cost and secure logic locking technique, seamlessly incorporating security into the IC design flow. SFLL-fault is

able to achieve savings of 33.0%, 26.8%, and 14.8% for area, power, and timing, respectively during the functionality-strip operation. This ultimately results in savings of -8.1%, -14.6%, and -1.7% for area, power, and timing, respectively, for the overall locked circuit. While the platform currently optimizes for area, it can be easily extended to optimize for other metrics such as power, timing or resilience against different attacks. The proposed platform demonstrates the effectiveness of leveraging design automation techniques and advocates their application in emerging fields such as hardware security.

REFERENCES

- [1] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," *IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.
- [2] SEMI, "Innovation is at Risk Losses of up to \$4 Billion Annually due to IP Infringement," 2008. [June 10, 2015].
- [3] Y. Alkabani and F. Koushanfar, "Active Hardware Metering for Intellectual Property Protection and Security," in *USENIX Security Symposium*, pp. 291–306, 2007.
- [4] A. Kahng, J. Lach, W. H. Mangione-Smith, S. Mantik, I. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Watermarking Techniques for Intellectual Property Protection," in *IEEE/ACM Design Automation Conference*, pp. 776–781, 1998.
- [5] J. Baukus, L. Chow, R. Cocchi, and B. Wang, "Method and apparatus for camouflaging a standard cell based integrated circuit with micro circuits and post processing," 2012. US Patent no. 20120139582.
- [6] F. Imeson, A. Emtenan, S. Garg, and M. V. Tripunitara, "Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation," in *USENIX Security Symposium*, pp. 495–510, 2013.
- [7] R. Pappu, B. Recht, J. Taylor, and N. Gershenfeld, "Physical One-Way Functions," *Science*, vol. 297, no. 5589, pp. 2026–2030, 2002.
- [8] J. Roy, F. Koushanfar, and I. L. Markov, "Ending Piracy of Integrated Circuits," *IEEE Computer*, vol. 43, no. 10, pp. 30–38, 2010.
- [9] M. Yasin, J. Rajendran, O. Sinanoglu, and R. Karri, "On Improving the Security of Logic Locking," *IEEE Transactions on CAD of Integrated Circuits and Systems*, vol. 35, no. 9, pp. 1411–1424, 2016.
- [10] J. Rajendran, H. Zhang, C. Zhang, G. Rose, Y. Pino, O. Sinanoglu, and R. Karri, "Fault Analysis-Based Logic Encryption," *IEEE Transactions on Computer*, vol. 64, no. 2, pp. 410–424, 2015.
- [11] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in *IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 137–143, 2015.
- [12] M. Yasin, B. Mazumdar, J. Rajendran, and O. Sinanoglu, "SARLock: SAT Attack Resistant Logic Locking," in *IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 236–241, 2016.
- [13] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z., and Y. Jin, "AppSAT: Approximately Deobfuscating Integrated Circuits," in *IEEE International Symposium on Hardware Oriented Security and Trust*, pp. 95–100, 2017.
- [14] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal Attacks on Logic Locking and Camouflaging Techniques," *IEEE Transactions on Emerging Topics in Computing*, vol. 99, no. 0, p. PP, 2017.
- [15] P. Tuyls, G. Schrijen, B. Škorić, J. van Geloven, N. Verhaegh, and R. Wolters, "Read-Proof Hardware from Protective Coatings," in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 369–383, 2006.
- [16] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 127–146, 2016.
- [17] X. Xu, B. Shakya, M. M. Tehrani, and D. Forte, "Novel Bypass Attack and BDD-based Tradeoff Analysis Against All Known Logic Locking Attacks," in *International Conference on Cryptographic Hardware and Embedded Systems*, pp. 189–210, 2017.
- [18] M. Yasin, A. Sengupta, B. Schafer, Y. Makris, O. Sinanoglu, and J. Rajendran, "What to Lock?: Functional and Parametric Locking," in *Great Lakes Symposium on VLSI*, pp. 351–356, 2017.
- [19] M. Yasin, A. Sengupta, M. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably Secure Logic Locking: From Theory to Practice," in *ACM SIGSAC Conference on Computer and Communications Security*, 2017. To appear.