Reconfigurable Run-Time Hardware Trojan Mitigation for Logic-Locked Circuits

Jordan Maynard
Computer Engineering & Computer Science Department
California State University Long Beach
Long Beach, CA, USA
jordan.maynard@student.csulb.edu

Amin Rezaei
Computer Engineering & Computer Science Department
California State University Long Beach
Long Beach, CA, USA
amin.rezaei@csulb.edu

***Abstract*—Globalized outsourcing of integrated circuit manufacturing has introduced potent security threats such as unauthorized overproduction and hardware Trojan insertion. An approach that is used to protect circuit designs from overproduction is logic locking, which introduces key inputs to a digital circuit such that only the correct key will allow the circuit to work properly and all others will cause unintended functionality. On the other hand, the majority of the existing methods to tackle hardware Trojans are in the realm of proactive prevention or static detection, but a more challenging problem, which is the run-time mitigation of the Trojans inserted in a zero-trust design flow, is yet to be solved. In this work, we look through the lens of logic locking with the goal of introducing online reconfigurability into a design and apply the fundamental principles of fault tolerance and state traversal to create an effective mitigation tactic against hardware Trojans. Redundancy is inserted at low-controllable states to create trap states for the attackers, and key inputs are added to select the active path. The strength of our proposed approach lies in its ability to circumvent Trojan payloads transparently at run-time with only a slight overhead, as demonstrated by experiments run on over 40 benchmarks of varying sizes. We also demonstrate viability when combined with secure logic locking methods to provide multi-objective security.**

***Index Terms*—Hardware Trojan; Reconfigurability; Logic Locking; Fabless Manufacturing; Zero-Trust Environment**

I. Introduction

Although the fabless model frees the semiconductor industry from having to invest in pricey manufacturing facilities and equipment, it poses additional security challenges, such as unauthorized overproduction and malicious insertion of Hardware Trojans (HTs). The situation worsens when integrating third-party Intellectual Property (IP) cores and utilizing different Electronic Design Automation (EDA) tools have become the norm in different design stages. Given the current trend, we have no choice but to consider a "zero-trust" environment in the Integrated Circuit (IC) design cycle.

Logic Locking (a.k.a. logic encryption) techniques [1], [2], which add extra key inputs to a given netlist, are well-studied solutions [3]–[12] to preventing unauthorized ICs from working. However, *not much exploration has been done on the additional key space that logic-locked circuits provide.* Considering a binary key size of n, only 1 out of 2^n possible keys is considered the correct key, and the others are left to be wrong and thus not utilized.

HTs can be inserted at any stage of the design cycle and are comprised of two parts: the trigger and the payload. The trigger consists of a latch-like component that activates only under rare conditions. This means that rare nets which are usually inactive, are likely to be chosen as trigger signals by the attackers to reduce the chance of static detection via testing. The payload is dormant until the trigger is activated, when it will carry out the intended malicious activity, such as information leakage [13], incorrect operation, or inflicting damage on the chip.

Existing approaches to securing ICs against HTs are either proactive [14]–[16], which aim to prevent HT insertion, or static [17]–[24], which attempt to detect inserted HTs by additional testing after the manufactured ICs return from the foundry. However, proactive methods are always imperfect due to the introduction of new and unknown Trojan insertion methods that could limit the capability of existing techniques. In addition, static detection methods rendered the IC useless after detecting possible HTs. A more effective yet challenging direction is to find architectural approaches to mitigate the effects of HTs during run-time. Key-controlled AND/OR gates are proposed in [25] to reduce the number of rare nodes and introduce false ones. However, the attacker can prune out wrong keys and identify the false nodes via SAT-based attacks [3]. Also, the way logic locking defends against HT insertion has been quantified [26] and it is proven that traditional logic locking methods [1], [2] have poor HT resilience.

After years of research, *we believe the most promising direction to achieve both secure logic locking and run-time HT mitigation objectives is to define security in terms of reconfigurability.* If the goal of logic locking is to have one correct key while all other circuit configurations are meaningless, the goal of HT mitigation can be defined as having multiple meaningful configurations (i.e., multiple correct keys) that the IC can switch to at run-time if a Trojan is triggered. Thus, if we carefully design secure logic-locked circuits with multiple but not exponential correct keys in which different correct keys do not overlap on Trojan-susceptible states, these two hardware security goals can converge. Figure 1 depicts the Trojan-aware logic-locked IC design flow in a zero-trust environment.

979-8-3503-4953-5/24/$31.00 ©2024 IEEE

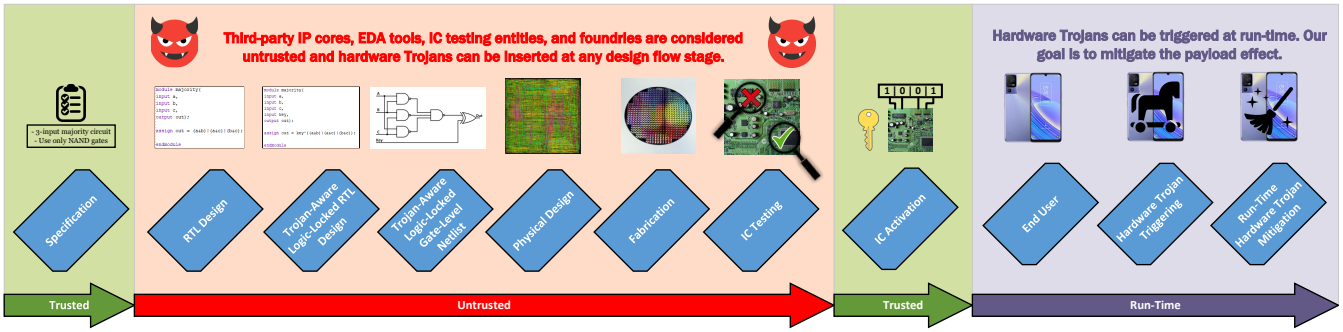


Fig. 1. Trojan-aware logic-locked IC design flow in a zero-trust environment

A. Problem Statement

Consider a sequential circuit $f(X,S) : \{0,1\}^n \times \{0,1\}^m \rightarrow \{0,1\}^t$ in which X is the set of input vectors and S is the set of all states. Supposing the set of key vectors K, the goal of traditional logic locking is to define a locked circuit $g(X,S,K) : \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^r \rightarrow \{0,1\}^t$ that satisfies the following constraint with respect to some security measurements:

$$\exists k^* \in K : g(X,S,k^*) \equiv f(X,S) \tag{1}$$

Then, a Trojan-infected logic-locked sequential circuit $g_{HT}(X,S,K) : \{0,1\}^n \times \{0,1\}^m \times \{0,1\}^r \rightarrow \{0,1\}^t$ can be modeled to satisfy the following constraint with respect to some malicious goals:

$$\begin{aligned} &\exists x_{HT} \in X, \exists s_{HT} \in S, \exists k_{HT} \in K : \\ &g_{HT}(x_{HT}, s_{HT}, k_{HT}) \not\equiv g(x_{HT}, s_{HT}, k_{HT}) \end{aligned} \tag{2}$$

Now, the combined goal of secure logic locking and HT mitigation would be defined as follows:

$$\begin{aligned} &\exists K^* \subset K : \\ &\quad \forall k^* \in K^* : g(X,S,k^*) \equiv f(X,S) \\ &\quad \exists k^*_{HT} \in K^* : g_{HT}(X,S,k^*_{HT}) \equiv f(X,S) \end{aligned} \tag{3}$$

In other words, there must be a subset of correct keys (i.e., K^*) under which the logic-locked circuit is equivalent to the original circuit, and there must be at least one correct key (i.e., k^*) within that subset under which the Trojan-infected logic-locked circuit is still functionally equivalent to the original circuit.

B. Contributions

The basis of our proposed method lies in using equivalent logic in the circuit to prevent the effects of a hardware Trojan from affecting circuit operation during runtime. Our unique contributions are as follows:

- Defining and implementing algorithmic identification and duplication of Trojan-likely states from behavioral circuit specifications.
- Proposing a novel reconfigurability-based approach for mitigating hardware Trojan infections in logic-locked sequential circuits during run-time.
- Demonstrating the security gain and overhead of our proposed approach through implementations on over 40 different FSM benchmark circuits with varying sizes.

C. Threat Model

We assume a zero-trust environment in which third-party IP vendors, EDA tools, manufacturing foundries, and even IC testing entities are all untrusted. The HT is inserted into the design at some point before fabrication in the design flow and can be skipped during the testing phase. We assume that one or more nets are chosen as victims for triggers, as is customary for any HT design. Furthermore, we suppose these triggers are placed on nets that have rare activation conditions. We also suppose that there is a malicious activity detection method embedded into the circuit that is flagged once HT is triggered. The design of such mechanisms can be done via run-time reliability analysis [27] or online machine learning approaches [28]. Once a HT is triggered in the design, we assume that the payload is continuously delivered. The payload is considered to have some malicious intent, but it should not be destructive in any way that damages the chip and renders it unusable.

II. Run-Time Trojan Mitigation

In this section, we propose **LIANA**, a low-overhead reconfigurab**L**e run-t**I**me h**A**rdware Troja**N** mitig**A**tion approach. Stealth is advantageous to attackers inserting HTs, so in a high-level design, we expect a Trojan to be inserted on the least-likely areas of operation. These are referred to as low-probability, low-controllable, or least-likely states. If we insert redundancies in these portions of the circuit, we can gain security benefits to circumvent the effects of possible inserted HTs. Especially, we explore what occurs when one or several least-likely states are cloned and a selective reachability property is added to each clone. Multiple lower likelihood states result from this design addition, and if any are chosen as victim wires for HT triggers, the HT payload effect can be mitigated through reconfigurable key inputs to the circuit. Figure 2 shows an overview of *LIANA*.

We approach this problem from the Finite State Machine (FSM) representation of sequential circuits. We can compute transition probability based on the number of inputs that

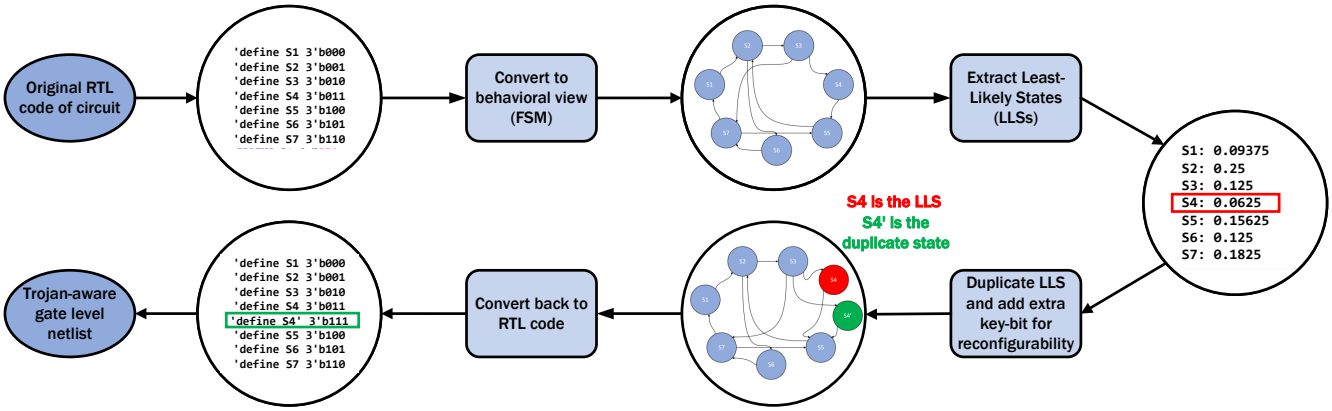
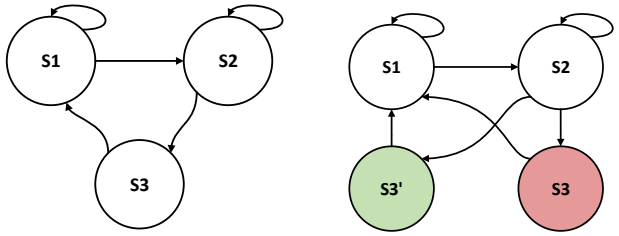


Fig. 2. LIANA overview

correlate to each next state transition. With this, it is possible to find out which states are the least-likely using matrix calculations. Then, we duplicate the least-likely states in order to create clear trap states for attackers who want to insert malicious Trojans. Transitions to and from the duplicate state are added to the parents and children of the original state; this allows for equivalent functionality with separate logic. A key-bit can be introduced to choose between the original and duplicate states. The low-controllable state clones serve to fool attackers into inserting a HT trigger on these rare transitions while retaining fully correct operation when an inserted HT is activated during run-time. Additional key-bits must be added into the cloning states such that in the case of wrong key insertion, these states behave differently and thus cannot be merged once a logic optimization is applied.

Finding the least-likely state is not possible from the deterministic graph representation since it represents transitions as a function of inputs decided by the user. We need to convert the FSM into a probabilistic form that assigns each transition a value representing the likelihood it will be chosen. Considering F the original FSM, four steps are taken toward choosing which states to duplicate. 1 We convert an original FSM into a deterministic transition matrix. 2 We use this transition matrix to create a probabilistic transition matrix to represent the long-term FSM behavior. 3 We compute the steady-state using Markov chains to determine the probability of being in each state at any given time. 4 The least-likely states will be cloned and may be traversed via new key inputs. Figure 3 and Tables I and II show the corresponding visual aids.

The Markov chain steady-state calculation takes two initial inputs: a probability matrix P and a state vector S_0. The probability matrix denotes the transition probability between every state. The state vector begins in the first state in the machine and will reach a steady-state when continuously multiplied by the probability matrix. The first iteration will look like this: $S_1 = S_0 \times P$. This is repeated n times until we reach $S_n \times P = S_{n-1} \times P$. Our final steady-state vector is S_n, and it will contain the generalized probability of being in each state at a given time.



(a) F (b) 4

Fig. 3. (a) Example FSM with least-likely state S3 (b) Example FSM with duplicated least-likely state

TABLE I
STEADY-STATE CALCULATION FOR FIGURE 3A

(a) Deterministic transition matrix 1

Deterministic	S1	S2	S3
S1	1	1	0
S2	0	1	1
S3	1	0	0

(b) Probabilistic transition matrix 2

Probabilistic	S1	S2	S3
S1	0.5	0.5	0
S2	0	0.5	0.5
S3	1	0	0

(c) Markov steady-state probabilities 3

Markov Steady-State	S1	S2	S3
Probability	0.4	0.4	0.2

There are cases in which a steady-state vector may not be reached, but instead a set of repeating state vectors will occur infinitely. Let us consider the case where there are two alternating steady-states S_A and S_B. It follows that $S_A = S_B \times P$ and $S_B = S_A \times P$.To consolidate the steady-state into a single vector, it is sufficient to combine the vectors and take the mean of each state probability value. This will return an estimated probabilistic likelihood for each state to

TABLE II
STEADY-STATE CALCULATION FOR FIGURE 3B

(a) Deterministic transition matrix

Deterministic	S1	S2	S3	S3'
S1	1	1	0	0
S2	0	1	1	1
S3	1	0	0	0
S3'	1	0	0	0

(b) Probabilistic transition matrix

Probabilistic	S1	S2	S3	S3'
S1	0.5	0.5	0	0
S2	0	0.5	0.25	0.25
S3	1	0	0	0
S3'	1	0	0	0

(c) Markov Steady-State Probabilities

Markov Steady-State	S1	S2	S3	S3'
Probability	0.4	0.4	0.1	0.1

infer the lowest-controllable state.

The idea behind duplicating the least-likely state in the FSM design relies on mitigating the effect of the payload of an inserted HT. Transitions in and out of the new state are identical to those in the old state, and the input and output values are also identical. The only difference will be an added key input that allows the user to choose between the original path and the new path. Additional key-bits must be added into the cloning states such that, in the case of the wrong key insertion, the cloned states behave differently and thus cannot be merged if a logic optimization is applied. Once HT is detected in the design, the Trojan-free state should be chosen over the Trojan-infected one. This will allow the IC to function as if no Trojan was inserted, essentially bypassing the payload effect.

The effect of a duplication on the steady-state probability provides a crucial security advantage. The existence of a duplicate state halves the likelihood of both itself and the original state and therefore increases their viability as an insertion point for a Trojan trigger. This heightens the possibility that the malicious modification will take place in either one of the protected states, ensuring the success of our defense tactic. This effect can be seen by comparing Tables I and II, referring to the steady-state probabilities of the example FSMs in Figure 3.

Theorem 1. *When a state in FSM is cloned with an extra input, its steady-state likelihood is reduced by half compared to the original FSM.*

Proof. This is a constructional proof. In a sequential circuit, $f(X, S)$, the set of all states is denoted by S. We define another set P, which contains a steady-state probability value for each state in S: $P \leftarrow steadyState(S)$. Using the smallest value in P, we can map this to a state s in S: $min(P) \mapsto \{s \in S\}$. We then duplicate this state $copy \leftarrow s$ and add it to the original state set $S \leftarrow \{S \cup copy\}$. We also add a single key-bit to the circuit so that the design can choose between equivalent states: the circuit becomes $g(X, S, K)$. The key does not determine any change in functionality aside from which path is chosen for operation. Since the functionality is the same and there are now two states in place of the least-likely one, the original and duplicate states combined will be chosen as often as the original state. Thus, both will have half of the original probability of the least-likely state. □

State duplication must be done in a way that does not add much overhead to the circuit. In any large design, there are always some unreachable states that are not utilized in the regular operation of the circuit. We can utilize these unreachable states in the design by assigning them to our new duplicate state(s), incurring no sequential logic overhead. The level of protection provided by this defense is inherently tunable as well. If we only duplicate the least-likely state, it leaves open the possibility for the attacker to insert Trojans both in the original and duplicated least-likely states. However, it can be simply overcome by creating more copies of the least-likely state with extra key inputs. This approach also maximizes the state space by occupying every possible state of every flip-flop in the design, leaving no room for a malicious sequential functionality change using unreachable states. In this case, an attacker would need to add extra FFs to the design, which provides obvious footprints. Area, power, and utilization overheads can also be considered tunable as an effect of creating multiple state clones. Thus, our method can be applied and modified to meet design constraints.

III. EXPERIMENTAL RESULTS

For the experiments, we used the custom example FSM depicted in Figure 3, "b02" from the ITC'99 benchmark suite [29], and a collection of 41 FSM designs from Synthezza open source benchmarks [30]. The circuits were split into three groups based on size. We applied *LIANA* to the Verilog source of the benchmarks and then converted from Verilog (.V) to .BENCH format using Yosys open synthesis suite [31]. Security results were gathered using Netlist Encryption and Obfuscation Suite (NEOS) [32] in Ubuntu 64-bit 20.04.4 LTS. Area and utilization overhead results were collected in Xilinx Vivado 2016.4 webpack edition on Windows 10.

For each original benchmark, four versions were created to demonstrate the capabilities of *LIANA*, as shown in Table III. The small and medium benchmarks were given a 3-bit reconfigurable key, while the large benchmarks were given a 5-bit reconfigurable key. An additional 10-bit key is used for locking each benchmark. It is worth mentioning that our choice of SAT-secure locking [12] uses two different keys that are applied to one set of key inputs at different times. The key widths from the reconfigurable key and the locking key are concatenated, leaving us with 13-bit and 15-bit key sizes. *LIANA*'s script and the created benchmarks are hosted on GitHub [1].

A. Security Analysis

Table IV depicts NEOS attack [32] results on different versions of the benchmarks. As we anticipated, the correct

[1]https://github.com/cars-lab-repo/LIANA

TABLE III
BENCHMARKS

Benchmark	Acronym	Description
Original	ORG	Used as oracle for security metrics and baseline for power and utilization overhead metrics.
Duplicated	DUPE	Low-controllable states are duplicated with added key input to choose between states. Functionality is equivalent to ORG under all the keys.
Duplicated Trojan-Inserted	DUPE-TI	Same as DUPE, but out of every two duplicated states, one has inverted outputs or wrong state transitions. Functionality is equivalent to ORG only under one correct key.
Duplicated Trojan-Inserted XOR-Locked	DUPE-TI-XOR	Same as DUPE-TI, but an added XOR lock [1] showcases its ability to be used in compound locking. Functionality is equivalent to ORG only under one correct key.
Duplicated Trojan-Inserted Secure-Locked	DUPE-TI-SEC	Same as DUPE-TI, but an added SAT-secure lock [12] showcases its ability to be used as multi-objective security. Functionality is equivalent to ORG only under a pair of keys.

TABLE IV
NEOS ATTACK [32] RESULTS ON LIANA

Benchmark	Key	DUPE	DUPE-TI	DUPE-TI-XOR	DUPE-TI-SEC
Small: ¡ 100 LUTs custom, b02, girl10, robm, sortmax, e191, lightnew, ex6, knot2, cat, e18, e17, lift2, e161, indep, and cpu	Key Size	3 bits	3 bits	13 bits	I: 13 bits, F: 13 bits
	Reported Key	Correct	Correct	Correct	No Result
Medium: ¿= 100 LUTs & ¡ 250 LUTs lift, checker9, as13, e10, robotben, bridge, bec, dmac, bcomp, pilot, e7, lcu, e2, and e16	Key Size	3 bits	3 bits	13 bits	I: 13 bits, F: 13 bits
	Reported Key	Correct	Correct	Correct	No Result
Large: ¿= 250 LUTs e8, e15, pp, v16, e4, sara, proc81616, proc16816, proc1688, max, and micks	Key Size	5 bits	5 bits	15 bits	I: 15 bits, F: 15 bits
	Reported Key	Correct	Correct	Correct	No Result

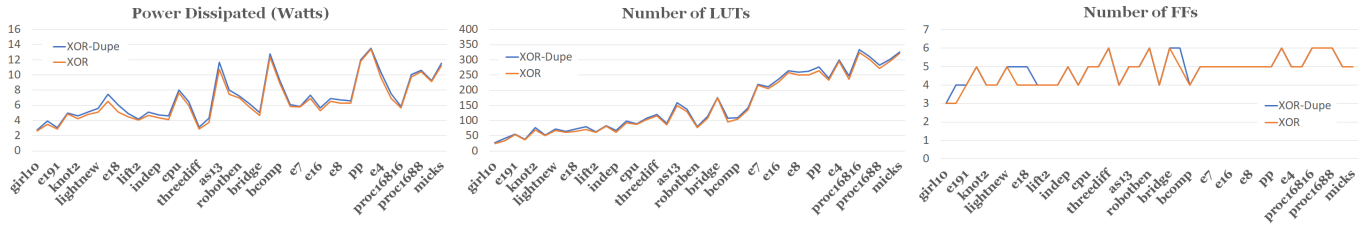
keys were returned for each benchmark type except DUPE-TI-SEC. For the DUPE benchmarks, each key value retains its original functionality, so any key may be returned. The NEOS attack framework checks the key at all 1's first, so the returned key was always comprised of 1's. In the DUPE-TI set, we inserted sequential Trojans in the benchmarks such that only a single correct key would retain original functionality. The results prove that the solver will choose the correct key under the circumstance that a Trojan payload is active in the design. Likewise, for the DUPE-TI-XOR set, the solver returns the correct key under an active Trojan payload. In this case, the solver is also able to report the key values for the added XOR locks, which demonstrates our ability to combine reconfigurability with other locking methods and retain correct circuit functionality. The DUPE-TI-SEC set yields no key when attacked. This verifies our claim that reconfigurable state duplication can contribute to high multi-objective security when combined with secure logic locking methods. Multiple threat types can be mitigated using our proposed approach in combination with another secure method. The correct operation under secure locking is verified by equivalence checking under the correct key.

B. Overhead Analysis

Figure 4 shows the overhead comparison for each benchmark between an XOR-locked version and a Trojan-aware XOR-locked version implemented on the Nexys A7-100T FPGA board. The active and static power dissipations are reported for each benchmark version. The numbers of Look Up Tables (LUTs) and Flip-Flops (FFs) utilized for each design are also reported. In terms of power, the highest power overhead came from "cat" with a 13.8% increase from 6.549 Watts to 7.451 Watts. The smallest power overhead was demonstrated on "sara" with a 0.8% increase from 13.415 Watts to 13.524 Watts. The size of the benchmarks and the number of states in the original FSM have a notable influence on this outcome. Benchmarks with a FF increase tended to have a slightly higher power consumption increase in comparison with their peers. The ratio of added states and key inputs to the original amount of states and allocated I/O correlates closely to the power overhead. However, as can be inferred from the results, our proposed method is scalable as the circuit size increases.

LUTs, or combinational logic, displayed the highest increase in "robm" with a 34.2% utilization increase from 34 LUTs to 41. The lowest increase was 0.5% from "bridge", which is likely because the content of the duplicated state was very minimal. When only unspecified states are utilized for reconfigurability, area overhead is only observable in combinational logic, not in sequential logic. The correlation between XOR-locked and Trojan-aware XOR-locked state space increases is proportional to LUT utilization.

Sequential logic showed change only on the benchmarks "robm", "e18", and "dmac". In each of these cases, the number of utilized FFs increased by one after adding reconfigurability. The reason this happened in only this small sample is that the state space increased from over the threshold limit set by the original number of FFs in the design. For instance, if a circuit has 3 FFs, it is possible to implement 2^3 or 8 states. If we add a state, making the total 9, we will need an additional FF to account for this extra functionality. This can be avoided by adding states up to but not over the limit set by the original design, in addition to utilizing unreachable states as discussed in Section II.



(a) (b) (c)

Fig. 4. Overhead analysis (a) Power consumption (b) Number of LUTs (c) Number of FFs

IV. Conclusion

In this work, we developed a multi-objective security framework called *LIANA* to not only protect circuit designs from overproduction but also enable run-time mitigation of hardware Trojans. We explored the opportunity that the additional key space of logic locking can provide to develop algorithmic identification and duplication of Trojan-likely states from behavioral circuit specifications in order to provide clear trap states for attackers to insert possible hardware Trojans. Then, by changing different correct keys, we allowed the run-time reconfigurability to switch to equivalent non-Trojan-induced states that mitigate the effect of the Trojan payload. Experiments on over 40 benchmarks with varying sizes confirmed that *LIANA* provides high security with low power and resource utilization.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Award No. 2245247.

References

[1] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1069-1074, 2008.

[2] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," In *Design Automation Conference (DAC)*, pp. 83–89, 2012.

[3] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," In *International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 137-143, 2015.

[4] K. Shamsi, M. Li, D. Z. Pan and Y. Jin, "KC2: Key-condition crunching for fast sequential circuit deobfuscation" In *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 534-539, 2019.

[5] Y. Hu, Y. Zhang, K. Yang, D. Chen, P. A. Beerel, and P. Nuzzo, "Fun-SAT: Functional corruptibility-guided SAT-based attack on sequential logic encryption," In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 281-291, 2021.

[6] A. Saha, H. Banerjee, R. S. Chakraborty, and D. Mukhopadhyay, "ORACALL: An oracle-based attack on cellular automata guided logic locking," In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 12, pp. 2445-2454, 2021.

[7] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-Lock: Hard distributions of SAT instances for obfuscating circuits using fully configurable logic and routing blocks," In *ACM/IEEE Design Automation Conference (DAC)*, pp. 1-6, 2019.

[8] K. Nayak, D. Upadhyaya, F. Regazzoni, and I. Polian, "On the limitations of logic locking the approximate circuits," In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 1-6, 2022.

[9] R. Afsharmazayejani, H. Sayadi, and A. Rezaei, "Distributed logic encryption: Essential security requirements and low-overhead implementation," In *Proceedings of Great Lakes Symposium on VLSI (GLSVLSI)*, pp. 127-131, 2022.

[10] M. T. Rahman, M. S. Rahman, H. Wang, S. Tajik, W. Khalil, F. Farahmandi, D. Forte, N. Asadizanjani, and M. Tehranipoor, "Defense-in-depth: A recipe for logic locking to prevail," In Integration, vol. 72, pp. 39-57, 2020.

[11] M. Zuzak, Y. Liu, and A. Srivastava,"Trace logic locking: Improving the parametric space of logic locking," In *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 40, no. 8, pp. 1531-1544, 2021.

[12] J. Maynard and A. Rezaei, "DK lock: Dual key logic locking against oracle-guided attacks," In *International Symposium on Quality Electronic Design (ISQED)*, pp. 1-7, 2023.

[13] A. De, M. Nasim Imtiaz Khan, K. Nagarajan and S. Ghosh, "HarTBleed: Using hardware Trojans for data leakage exploits," In *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 4, pp. 968-979, 2019.

[14] G. Guo, H. You, Z. Tang, B. Li, C. Li, and X. Zhang, "ASSURER: A PPA-friendly security closure framework for physical design," In *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 504-509, 2023.

[15] J. Knechtel, J. Gopinath, J. Bhandari, M. Ashraf, H. Amrouch, S. Borkar, S. -K. Lim, and O. Sinanoglu, and R. Karri, "Security closure of physical layouts." In *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pp. 1-9, 2021.

[16] F. Wang, Q. Wang, B. Fu, S. Jiang, X. Zhang, L. Alrahis, O. Sinanoglu, J. Knechtel, T. -Y. Ho, and E. F. Y. Young, "Security closure of IC layouts against hardware Trojans," In *International Symposium on Physical Design (ISPD)*, pp. 229–237, 2023.

[17] Q. Yu, J. Dofe, and Z. Zhang, "Exploiting hardware obfuscation methods to prevent and detect hardware Trojans," In *IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 819-822, 2017.

[18] S. Bhasin and F. Regazzoni, "A survey on hardware Trojan detection techniques," In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2021-2024, 2015.

[19] H. Salmani, "COTD: Reference-free hardware Trojan detection and recovery based on controllability and observability in gate-level netlist," In *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 2, pp. 338-350, 2017.

[20] A. Nahiyan, M. Sadi, R. Vittal, G. Contreras, D. Forte, and M. Tehranipoor, "Hardware Trojan detection through information flow security verification," In *IEEE International Test Conference (ITC)*, pp. 1-10, 2017.

[21] R. Vishwakarma and A. Rezaei, "Risk-aware and explainable framework for ensuring guaranteed coverage in evolving hardware Trojan detection," In *IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1-9, 2023.

[22] A. Jain, Z. Zhou, and U. Guin, "Survey of recent developments for hardware Trojan detection," In *IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1-5, 2021.

[23] K. I. Gubbi, B. S. Latibari, A. Srikanth, T. Sheaves, S. A. Beheshti-Shirazi, S. Manoj Pd, S. Rafatirad, A. Sasan, H. Homayoun, and S. Salehi, "Hardware Trojan detection using machine learning: A tutorial," In *ACM Transactions on Embedded Computing Systems*, vol. 22, no. 3, article 46, 2023.

[24] A. Vakil, A. Mirzaeian, H. Homayoun, N. Karimi, and A. Sasan, "AVATAR: NN-Assisted variation aware timing analysis and reporting for hardware Trojan detection',' In *IEEE Access*, vol. 9, pp. 92881-92900, 2021.

[25] Y. -L. Zhang, Y. -J. Yan, J. -W. Li, and M. Liu, "A novel hardware Trojan protection technology based on logic encryption," In *IEEE International Conference on Integrated Circuits and Microsystems (ICICM)*, pp. 396-400, 2018.

[26] J. Cruz, P. Gaikwad, and S. Bhunia, "Analysis of hardware Trojan resilience enabled through logic locking," In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pp. 1-6, 2022.

[27] R. S. Chakraborty, S. Pagliarini, J. Mathew, S. R. Rajendran, and M. N. Devi, "A flexible online checking technique to enhance hardware Trojan horse detectability by reliability analysis," In *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 2, pp. 260-270, 2017.

[28] A. Kulkarni, Y. Pino, and T. Mohsenin, "Adaptive real-time Trojan detection framework through machine learning," In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)* pp. 120-123, 2016.

[29] S. Davidson, "ITC'99 Benchmark Circuits - Preliminary Results," In *International Test Conference (ITC)*, pp. 1125-1125, 1999.

[30] S. Baranov, "Benchmarks of FSMs and Logic Circuits," https://www.synthezza.com/fsm-and-logic-circuit-benchmarks.

[31] C. Wolf, "Yosys open synthesis suite," https://github.com/YosysHQ/yosys.

[32] K. Shamsi "NEOS: Netlist encryption and obfuscation suite," http://bitbucket.org/kavehshm/neos/.