PoTeNt: Post-Synthesis Obfuscation for Secure Network-on-Chip Architectures

Dipal Halder*, Yuntao Liu†, Kostas Amberiadis‡, Ankur Srivastava†, Sandip Ray*
*Electrical and Computer Engineering, University of Florida, Gainesville, FL, U.S.A
†Electrical and Computer Engineering, University of Maryland, College Park, MD, U.S.A
‡ National Institute of Standards and Technology, Gaithersburg, MD, U.S.A

***Abstract*—We develop a novel post-synthesis obfuscation technique, PoTeNt, to protect NoC fabrics against reverse-engineering attacks. PoTeNt integrates programmable switches at NoC routers, concealing topology and communication paths under a dynamically controlled key to make the design resilient to reverse-engineering attempts. By targeting post-synthesis gate-level netlist, PoTeNt overcomes challenges induced by logic optimization that can render pre-synthesis (RTL-level) NoC obfuscations vulnerable to SAT attacks. We extensively evaluate PoTeNt to demonstrate its robustness to SAT attacks. Finally, our experiments show that PoTeNt incurs minimal overhead on area, power, and performance.**

***Index Terms*—Network-on-Chip, System-on-Chip, Post-synthesis obfuscation, Reverse-engineering attacks**

I. Introduction

In modern System-on-Chip (SoC) designs, Network-on-Chip (NoC) fabrics have emerged as the primary communication technique, enabling efficient data transfer between the various integrated hardware blocks or intellectual property (IP) cores shown in Fig. 1. NoCs offer scalability, flexibility, and high-bandwidth communication, making them crucial for meeting the performance demands of complex SoCs. However, with the globalization of the semiconductor supply chain, SoCs and their NoC fabrics have become increasingly vulnerable to various security threats, particularly reverse-engineering attacks. An adversary with access to any form of the design netlist or fabricated chip can extract sensitive information about the NoC topology, communication protocols, and connected IP cores. This information can be exploited to understand the chip's functionality, architecture, and specific IP implementations, enabling counterfeit chip production, unauthorized overbuilding, insertion of hardware Trojans, and other malicious activities [1]. Therefore, protecting the NoC fabric from reverse-engineering attacks is crucial for ensuring the security of the communication infrastructure in SoC designs.

To address these security concerns, previous work [2] proposed a system-level obfuscation technique that modifies the design representation to conceal its functionality. However, this technique demonstrated effective protection against reverse engineering, pre-synthesis obfuscation technique faces several limitations. In particular, when such an obfuscated design is subjected to logic synthesis, many signals are discarded due to logic optimization. This results in a synthesized gate-level netlist with only partial obfuscation, making the synthesized design more vulnerable to Boolean satisfiability (SAT)-based attacks. In many cases, SAT finds keys that correspond to different topologies by appearance but are equivalent to the correct topology due to the reduction of signals during the synthesis.

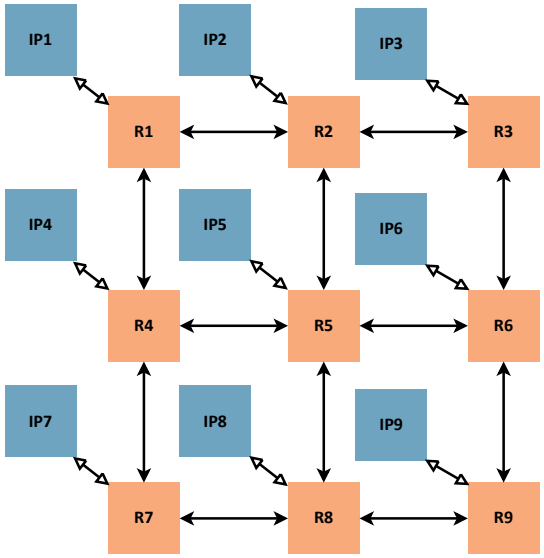

Fig. 1: NoC-based Multi Core Topology

In this paper, we present a comprehensive solution PoTeNt, a novel post-synthesis obfuscation methodology for NoC fabrics. PoTeNt operates directly on the gate-level netlist representation of the NoC, making the technique robust against vulnerabilities due to logic optimization. On the other hand, PoTeNt exploits guidance from the register-transfer-level (RTL) to find appropriate nets from the synthesized NoC that enable the obfuscation to preserve the functional semantics desired from pre-synthesis obfuscation.

This paper makes the following important contributions.

- Our framework PoTeNt is, to our knowledge, the first post-synthesis framework for NoC fabrics.
- We conduct an in-depth security analysis to demonstrate the robustness of PoTeNt against SAT-based attacks
- Our detailed performance analysis on a representative SoC design (with an NoC fabric) demonstrates that the area, power, and performance overheads incurred by PoTeNt are not significant.

The remainder of the paper is organized as follows. Section

II reviews related work on hardware obfuscation and NoC security. Section III discusses the research challenges on NoC obfuscation. Section IV presents our developed methodology, while Section V details our experimental results and security analysis. Finally, Section VI concludes the paper and outlines future research directions.

II. Related Work

Hardware obfuscation is a highly mature research area with a number of interesting research approaches [3], [4], [5]. However, these works target individual IPs. To our knowledge, there is no previous research on NoC obfuscation that targets gate-level netlist. The research closest to ours is from Halder *et al.* [2], which shows how to obfuscate NoC topologies. The underlying foundation of PoTeNt is inspired by this approach, in particular the idea of using programmable switches to redact topology information. However, unlike PoTeNt, Halder *et al.*'s approach targeted pre-synthesis (RTL) designs, hence, netlist created from the obfuscated RTL could be transformed sufficiently by logic optimization and vulnerable to SAT attacks (see Section III-A).

Although there has not been much work on NoC obfuscation, other aspects of NoC security have been explored in previous work. Wassel *et al.* [6] introduced SurfNoC, a low-latency and non-interfering approach that employs time-division multiplexing (TDM) to create isolated communication channels. Sepúlveda *et al.* [7] developed a security-enhanced NoC design incorporating encryption, authentication, and access control mechanisms. There are also several works that have explored interconnect obfuscation. Patnaik *et al.* [8] proposed a layout camouflaging scheme that obfuscates interconnects (BEOL) without modifying the device layer (FEOL), enabling low-cost, generic, and full-chip protection. Chakraborty *et al.* [9] investigated logic obfuscation at the RTL interconnect using polymorphic switch boxes (SBs) to increase the difficulty of key-bit identification. Yu *et al.* [10] proposed a hardware encryption technique based on ferroelectric field-effect transistor (FeFET) by utilizing run-time reconfigurable inverter-buffer logic, enabling output encoding and decoding without affecting critical path delay. Shalaby *et al.* [11] presented Sentry-NoC, a NoC architecture that combines static scheduling with temporal and data obfuscation schemes to mitigate side-channel attacks while reducing security overheads.

Beyond gate-level obfuscation, researchers have investigated post-fabrication and high-level obfuscation techniques. Shihab *et al.* [12] proposed TRAP, a post-fabrication method that programmatically defines parts of a design after manufacturing. Islam *et al.* [13] and Pilato *et al.* [14] explored RTL IP protection techniques during High-Level Synthesis, employing key-based obfuscation, design lockout, camouflaging, and algorithmic transformations. There are several works on protecting NoC fabric from hardware Trojan. Frey *et al.* [15] developed a method to detect hardware Trojans in network interfaces (NIs) by adding key bits and dummy states to the NI control unit's finite state machine. Patooghy *et al.* [16] and Sarihi *et al.* [17] proposed a secure and anonymous routing with minimal hardware overhead by encrypting the entire packet while exchanging information over the network.

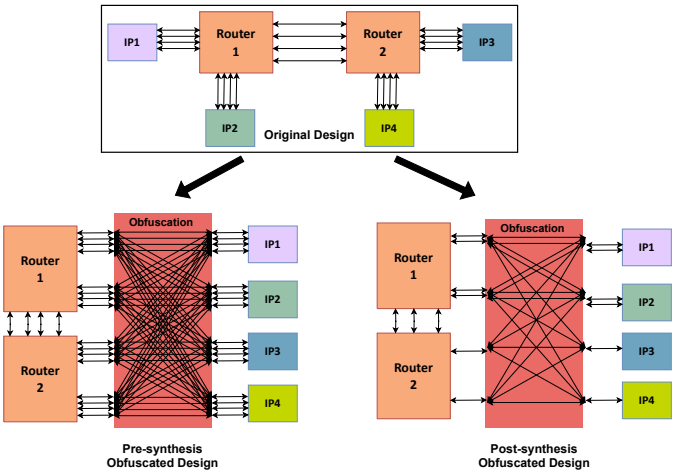

Fig. 2: Pre-Synthesis vs Post-Synthesis Obfuscation Analysis

III. Challenges in NoC Obfuscation: Pre-Synthesis vs. Post-Synthesis

A. Limitations of Pre-Synthesis Obfuscation

Pre-synthesis obfuscation techniques operate on the RTL description. However, this technique has several limitations that can impact its effectiveness and practicality. The primary limitation is that the synthesis modifies the structure and connectivity, making the obfuscation technique unusable. In Fig. 2 we show a design and their corresponding obfuscated design for both pre-synthesis and post-synthesis. All the router IOs with associated IPs are considered for applying the obfuscation in pre-synthesis. In the post-synthesis obfuscated design, we see many signals are missing compared to the pre-synthesis obfuscated design. Fig. 3(a) and (b) show a practical router connection with the associated IPs inside an NoC fabric for both pre-synthesis and post-synthesis. There are various signals among the *router_000* and connected IPs such as *sink_data*, *sink_endofpacket*, *sink_startofpacket*, *sink_valid*, and *src_ready*. This post-synthesis alters the structure and connectivity of this design. The synthesized gate-level netlist only has the *sink_data* signal and *src_valid* signal available to the *router_000*. It also changes the data width for various signals. For example, the *sink_data* signal has a width of 130 bits in the pre-synthesis, whereas this signal only contains 11 bits in the post-synthesis. The logic optimization also affects the design signals introduced by the obfuscation algorithm. Indeed, we implemented the obfuscation algorithm of the experimental analysis of Halder *et al.* on the same SoC that we use to validate PoTeNt in Section V. Note that the obfuscation architecture itself was shown by previous work to be provably secure against reverse-engineering attacks, including SAT attacks. *However, we could successfully apply the SAT attack on the synthesized gate-level netlist after logic optimization and find the correct key within 10 seconds.* On the

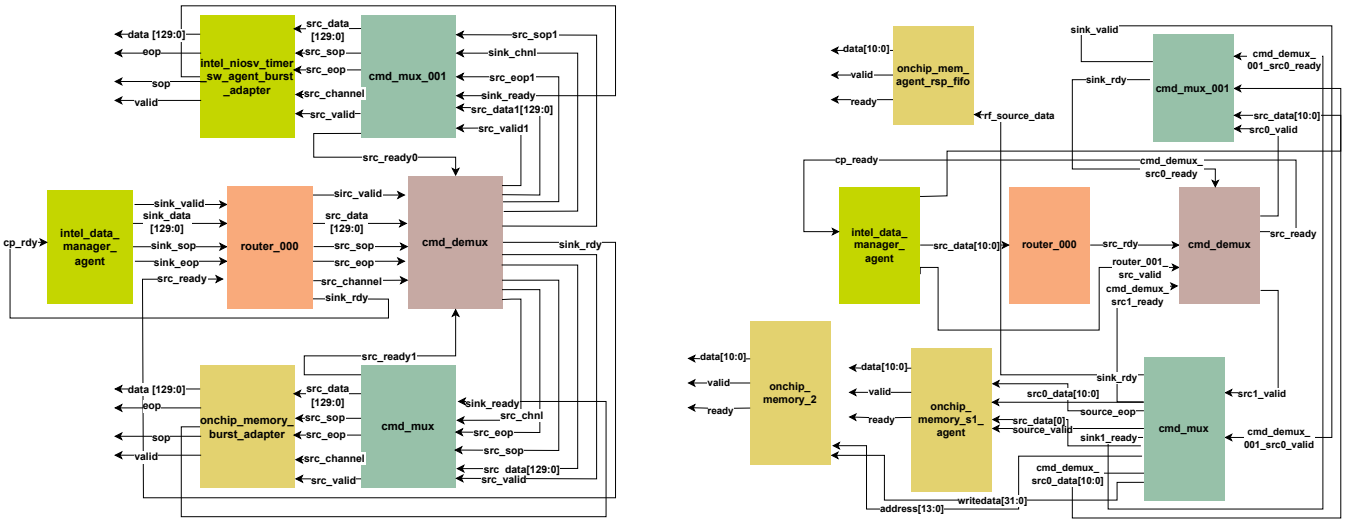

Fig. 3: Analysis of a Single Router Connection inside a NoC fabric (a) Pre-Synthesis (b) Post-Synthesis

other hand, the SAT attack does not succeed on the obfuscation performed by PoTeNt (on the same netlist) as illustrated in Section V.

B. Challenges in Post-Synthesis Obfuscation

Post-synthesis obfuscation, which operates directly on the gate-level netlist, can avoid the challenges associated with pre-synthesis but introduce a new set of challenges. One primary challenge is the increased complexity of the gate-level netlist compared to the RTL description, making it difficult to identify and isolate specific parts of the NoC for obfuscation without disrupting the design's functionality or performance [18]. Obfuscation techniques should preserve the original functionality and performance while minimizing overhead in terms of area, power, and latency. This requires careful analysis and optimization of the modifications introduced during the obfuscation process [19]. Another challenge is resilience against attack scenarios such as functional and structural analysis [20]. The obfuscation methodology must introduce sufficient ambiguity and complexity into the netlist to hinder attackers from inferring the original NoC topology, communication paths, and connected IP cores.

IV. PoTeNt Framework

The PoTeNt framework consists of two stages. In that first stage, a connectivity matrix is created, which serves as the main building block for PoTeNt. With the help of this matrix, we integrate the obfuscation switches into the router locations of the NoC. The second stage involves developing the obfuscation switch, which is integrated into the gate-level netlist according to the connectivity matrix.

A. Connectivity Matrix Generation

In a SoC, an NoC fabric contains multiple routers which are connected to different IPs. In our benchmark design, there are a total of 6 routers inside the NoC fabric. For each router, a connectivity matrix is generated based on the pre-synthesis and post-synthesis netlist information. Fig. 3 shows a subset of the NoC fabric with *router_000* and the corresponding connections of this router with the IPs for both the pre-synthesis and post-synthesis netlist.

Algorithm 1 Connectivity Matrix Generation

Require: Router R with input R_i and output R_o
Ensure: Connectivity Matrix M_{final}
Initialize M_{pre} and M_{post} as zero matrices for pre- and post-synthesis connections.
for $i = 1$ to $|R_i|$ **do**
  for $j = 1$ to $|R_o|$ **do**
    Set $M_{\text{pre},ij} = 1$ if input i is connected to output j in pre-synthesis, else $M_{\text{pre},ij} = 0$
    Set $M_{\text{post},ij} = 1$ if input i is connected to output j in post-synthesis, else $M_{\text{post},ij} = 0$
  end for
end for
Connectivity Matrix Generation for Router R:
Initialize matrix M_{final} with dimensions $|R_i| \times |R_o|$
for $i = 1$ to $|R_i|$ **do**
  for $j = 1$ to $|R_o|$ **do**
    if $M_{\text{pre},ij} = 1$ AND $M_{\text{post},ij} = 1$ **then**
      Set $M_{\text{final},ij} = 1$ indicating the signal is preserved in both
    else
      Set $M_{\text{final},ij} = 0$
    end if
  end for
end for
return Connectivity Matrix M_{final} for Router R

TABLE I: Connectivity Matrix for ***router_000***

Connected IP1 (*intel_data_manager_agent*)		**Targeted ROUTER (*router_000*)**		**Connected IP2 (*cmd_demux*)**	
Output	**Connected Wire**	**Input**	**Output**	**Connected Wire**	**Input**
align_address_to_size128	N43845[14]	align_address_to_size	reduce_nor_010	N43857	router
align_address_to_size129	N43845[13]	align_address_to_size0		N43851[2]	write_addr_data_both_valid
align_address_to_size130	N43845[12]	align_address_to_size1		N43851[1]	sink0_ready
align_address_to_size131	N43845[11]	align_address_to_size2		N43851[0]	cmd_mux_001
align_address_to_size132	N43845[10]	align_address_to_size3			
align_address_to_size133	N43845[9]	align_address_to_size4			
align_address_to_size134	N43845[8]	align_address_to_size5			
align_address_to_size135	N43845[7]	align_address_to_size6			
align_address_to_size136	N43845[6]	align_address_to_size7			
align_address_to_size137	N43845[5]	align_address_to_size8			
align_address_to_size138	N43845[4]	align_address_to_size9			

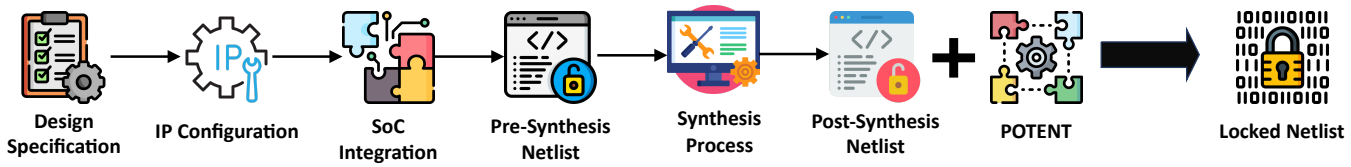

Fig. 4: PoTeNt Integration Flow

The connectivity matrix generation process, as outlined in Algorithm 1, takes the router R with its input R_i and output R_o as input and generates the final connectivity matrix M_{final}. The algorithm first generates connectivity maps for the router R, initializing M_{pre} and M_{post} as zero matrices for pre- and post-synthesis connections, respectively. It then sets the values in these matrices based on the connections between the inputs and outputs in the pre-synthesis and post-synthesis netlist. Finally, the algorithm generates the connectivity matrix M_{final} by comparing the pre-synthesis and post-synthesis connectivity maps, which are shown in Table I for *router_000*.

B. Obfuscation Switch Generation and Integration

The core of PoTeNt lies in the obfuscation switch, Ω, which is integrated into each NoC router. For a router with n ports, Ω employs a permutation-based obfuscation approach to reconfigure signal paths dynamically.

The set of possible signal connections, S, undergoes permutations, generating $n!$ mappings. Each mapping is associated with a unique key from the key space K, a binary vector space of dimension b, resulting in $|K| = 2^b$ potential keys. The relationship between keys and signal mapping is $P(k_i)$.

$$P(k_i) = \sigma_i, \quad \forall i \in \{1, 2, \ldots, n!\}$$

Here σ_i represents the i-th permutation of signals within S. As the number of routers N_R increases with the number of IPs, the dimension of K grows exponentially, providing a large space to defend against attackers. The total key space required for N_R routers is given by $|K_{\text{total}}| = 2^{(N_R)\cdot b}$. The integration of Ω across the routers helps to create multiple unique network topologies under each unique key, resulting in an updated connectivity matrix, $M_{\text{final}}^{(R_{\text{obf}})}$.

$$M_{\text{final},ij}^{(R_{\text{obf}})} = \begin{cases} 1 & \text{if } P(k)(i,j) \text{ yields a valid} \\ & \text{connection under } \Omega, \\ 0 & \text{otherwise} \end{cases}$$

Algorithm 2 describes the process of generating and integrating Ω with the router R to form the obfuscated router R_{obf}. The algorithm takes the connectivity matrix M_{final} from Algorithm 1 as input and outputs an obfuscated router R_{obf}.

Algorithm 2 Obfuscation Switch Generation and Integration

Require: Connectivity Matrix M_{final} from Algorithm 1
Ensure: Obfuscated Router R_{obf}
Generate Obfuscation Switch Ω***:***
Let S be the set of all signals in M_{final}
Define P as the set of all permutations of signals, $P : S \to S$, where $|P| = 4! = 24$
Assign a unique 5-bit key K to each permutation P associated with Ω, with $K \in \{0,1\}^5$ and $|K| = 32$
Keys beyond the first 24 are defined as "ZERO," rendering the output null for any key from 24 to 31
for $k = 0$ to 23 **do**
    Map key K_k to a specific permutation P_k in Ω
end for
Integrate Switch Ω **with Router** R **to form** R_{obf}**:**
for each (i,j) in S where $M_{\text{final},ij} = 1$ **do**
    Select key K corresponding to the desired permutation P in Ω for (i,j)
    Apply $P(K)$ from Ω to (i,j), updating the connection in $M_{\text{final},ij}$
end for
return Obfuscated Router R_{obf}

The resulting obfuscated router R_{obf} represents the router after integrating with Ω, effectively transforming network

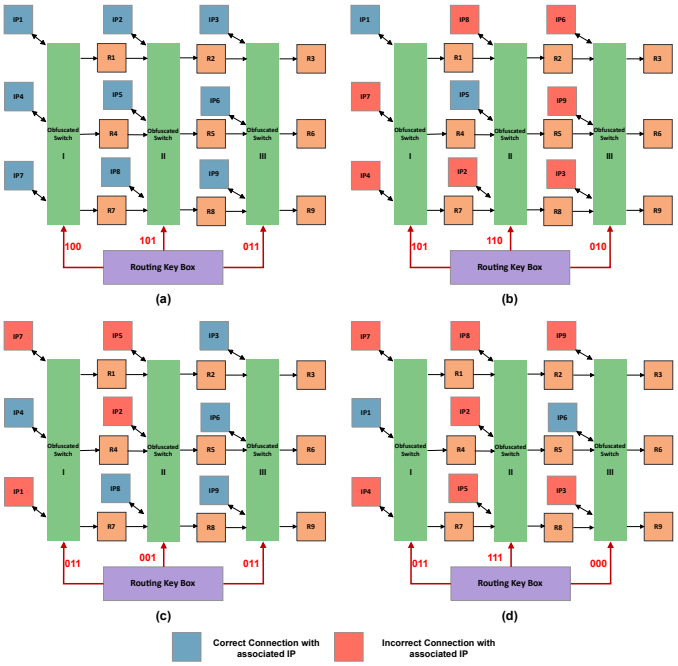

Fig. 5: Different Functional Topologies under various Keys

topology. By inserting Ω dynamically into each router to reconfigure the signal paths based on unique keys, PoTeNt introduces a robust layer of protection, while the updated connectivity matrix $M_{\text{final}}^{(R_{\text{obf}})}$ captures the obfuscated topology.

The PoTeNt integration Flow, illustrated in Fig. 4, seamlessly incorporates the obfuscation process into the standard SoC design flow. The router R is transformed into its obfuscated counterpart R_{obf}, effectively concealing the original network architecture while maintaining operational integrity. Fig. 5 demonstrates the effectiveness of the obfuscation switch Ω in an NoC-based multi-core design.

Fig. 5(a) presents the original network configuration S_{orig}, which is preserved under the correct key via the identity permutation function P_{id}.

$$P_{\text{id}}(S_{\text{orig}}) = S_{\text{orig}}$$

In the correct topology, the 9 IPs (IP1 to IP9) are connected with the corresponding router (R1 to R9), respectively, under the key $100, 101, 011$ that is applied to the obfuscation switch I, II, III. In contrast, alternative keys generate distinct topologies S', altering the NoC topology landscape as depicted in Fig. 5(b), 5(c), and 5(d). In these three topologies, we insert a different key, which leads to a unique topology, though they cannot provide the correct functionality. For example, in Fig. 5(c), we see that the correct key 011 is applied to obfuscation switch III, making the following IPs to router connection the correct topology. However, the obfuscation switches I and II have the incorrect key $(011, 001)$ applied, respectively, which results in the following IP to router connection being different compared to the correct topology. These changes are quantified by a permutation function P, where each key

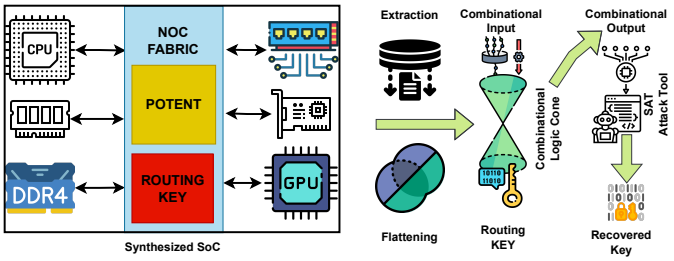

Fig. 6: SAT Attack Flow

$k \in K$ maps S_{orig} to a new permutation S', representing a reconfigured topology.

$$P(k)(S_{\text{orig}}) = S', \quad k \neq \text{key}_{\text{correct}}$$

This transformation for router-to-IP connections introduces a permutation space that inherently increases the combinatorial diversity and, consequently, the system's security. Each non-identity permutation adds a layer of obfuscation, ensuring that the true network topology remains concealed without the correct key.

V. Evaluation and Security Analysis

In the assessment of the PoTeNt framework, we employed the Quartus Pro 23.2 to construct a sophisticated SoC. This SoC contains a processing unit, memory unit, and communication unit, and all of these IPs were integrated with an NoC fabric. The PoTeNt framework was applied after generating the initial pre-synthesis netlist from Quartus Platform Designer and subsequent post-synthesis in DesignNavigator, with integrity checks performed by Synopsys Spyglass to ensure the absence of combinational loops. The synthesized unobfuscated SoC comprises a total of approximately $10K$ gates.

A. Security Evaluation against SAT Attacks

In our security evaluation against SAT attacks, we have rigorously assessed the resistance of PoTeNt within an obfuscated SoC context. Traditional SAT-based attacks that target isolated hardware modules are ineffective against PoTeNt, which secures the NoC on a holistic SoC scale. Advancing beyond the methodologies applied in [20], we have fortified a comprehensive toolchain augmentation, encapsulated by a suite of TCL scripts and open-source utilities, catalyzing the transition from gate-level netlist to an SAT-compatible format. This paradigm shift is symbolized by $\mathcal{F} : Net_{post} \rightarrow Net_{SAT}$, where Net_{post} is the gate-level netlist and Net_{SAT} is the netlist formatted for SAT analysis.

Our security evaluation flow on PoTeNt is shown in Fig. 6. From the formatted netlist Net_{SAT}, we get the combinational outputs, which then feed into the SAT attack tool Σ_{SAT} to find the recovered key $K_{recovered}$. This $K_{recovered}$ is then compared with the correct key $K_{correct}$. We have found that the $K_{recovered}$ obtained from the SAT attack was incorrect, *i.e.*, different from $K_{correct}$. In fact, $K_{recovered}$ corresponds to a different topology T_{alt}. As PoTeNt is designed in this way

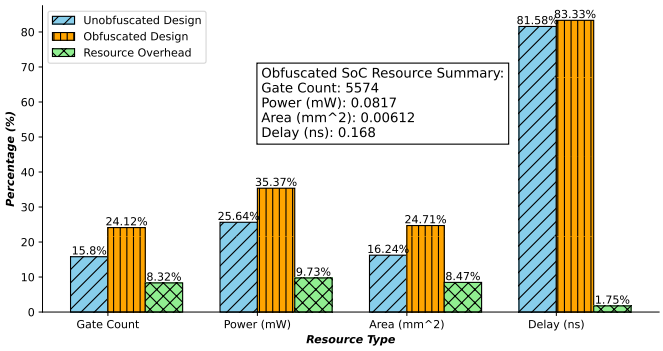

Fig. 7: Resource Utilization Analysis

to generate each different topology under each unique key, this key-topology binding is expressed by $\Phi(K_{recovered}) \neq T_{original}$. The implications are twofold.

1) The $K_{recovered}$ that did not reinstate the original functionality underscores PoTeNt's unique construction, where $\Phi : K \mapsto T$ is a bijection for only the correct key, $K_{correct}$, ensuring that any $K \neq K_{correct}$ leads to an altered topology T_{alt}.
2) From the attacker's perspective, the SAT attack does not recover the original topology $T_{original}$ and, hence, the original functionality of the design.

In summary, the SAT attack's inability to compromise the original design via $K_{recovered}$ affirms the efficacy of PoTeNt.

B. Resource Overhead Analysis

The comparative resource overhead analysis has been shown in Fig. 7. We used gate count, power consumption, area, and delay as the primary evaluation metrics to quantify the resource overhead between the unobfuscated and obfuscated design. Compared to unobfuscated design, design with PoTeNt has an overhead of 8.32 % in gate count. This trend persists in the area overhead, which stands at 8.47 %. The integration of PoTeNt also exerts on the total power consumption, resulting in a 9.83% increase. The NoC fabric in the original unobfuscated SoC is responsible for approximately 25 % of the total gate count, and the SoC itself is comparatively small. If the same obfuscation is applied to an industrial SoC where NoC fabric typically accounts for a much smaller gate count (the footprint is dominated by large IPs like the CPU), the overhead would obviously be much less. Considering this fact, the resource overhead on area, power, and gate count, which is between 8 % to 10 % in this SoC implementation, is not that significant. On the other hand, as the primary communication backbone, this NoC fabric inherently exhibits the highest delay, which is 81.58 % within the SoC. However, the obfuscation process introduces only 2 % overhead in the final obfuscated design. Overall, PoTeNt does not incur significant resource overhead, considering the security robustness it provides.

VI. Conclusion and Future Work

This paper developed PoTeNt, an obfuscation technique designed to protect NoC fabrics within SoCs against reverse engineering attacks. By targeting obfuscation at the gate-level netlist, PoTeNt overcomes the pre-synthesis obfuscation challenges induced by logic synthesis optimizations. On the other hand, it provides the flow to enable pre-synthesis collateral, such as RTL, to guide the post-synthesis obfuscation process. Our experiments show that PoTeNt is resilient to SAT attacks while not incurring high resource overhead.

In future work, we plan to apply PoTeNt in larger SoC designs with different NoC architectures. We will also extend our implementation of PoTeNt into a fully automated CAD infrastructure.

DISCLAIMER

The opinions, recommendations, findings, and conclusions in this article do not necessarily reflect the views or policies of NIST or the United States Government.

References

[1] M. Rostami, F. Koushanfar, J. Rajendran, and R. Karri, "Hardware security: threat models and metrics," ser. ICCAD, 2013, p. 819–823.

[2] D. Halder, M. Merugu, and S. Ray, "Obnocs: Protecting network-on-chip fabrics against reverse-engineering attacks," *ACM Trans. Embed. Comput. Syst.*, vol. 22, no. 5s, sep 2023.

[3] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu, "Sarlock: Sat attack resistant logic locking," in *IEEE HOST*, 2016, pp. 236–241.

[4] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *ACM CCS*, 2017, p. 1601–1618.

[5] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Interlock: an intercorrelated logic and routing locking," in *ICCAD*, ser. ICCAD, 2020.

[6] H. M. G. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, "Surfnoc: a low latency and provably non-interfering approach to secure networks-on-chip," *SIGARCH Comput. Archit. News*, vol. 41, no. 3, p. 583–594, jun 2013.

[7] J. Sepúlveda, A. Zankl, D. Flórez, and G. Sigl, "Towards protected mpsoc communication for information protection against a malicious noc," *Procedia Computer Science*, vol. 108, pp. 1103–1112, 2017, iCCS.

[8] S. Patnaik, M. Ashraf, J. Knechtel, and O. Sinanoglu, "Obfuscating the interconnects: low-cost and resilient full-chip layout camouflaging," in *Proceedings of the 36th International Conference on Computer-Aided Design*, ser. ICCAD. IEEE Press, 2017, p. 41–48.

[9] H. Chakraborty and R. Vemuri, "Rtl interconnect obfuscation by polymorphic switch boxes for secure hardware generation," *arXiv preprint arXiv:2404.07426*, 2024.

[10] T. Yu, Y. Xu, S. Deng, Z. Zhao, N. Jao, Y. Kim, S. Dünkel, S. Beyer, K. Ni, S. George, and V. Narayanan, "Hardware functional obfuscation with ferroelectric active interconnects," *Nature Communications*, vol. 13, 04 2022.

[11] A. Shalaby, Y. Tavva, T. E. Carlson, and L.-S. Peh, "Sentry-noc: a statically-scheduled noc for secure socs," ser. NOCS, 2021, p. 67–74.

[12] M. M. Shihab, J. Tian, G. R. Reddy, B. Hu, W. Swartz, B. Carrion Schaefer, C. Sechen, and Y. Makris, "Design obfuscation through selective post-fabrication transistor-level programming," in *DATE*, 2019, pp. 528–533.

[13] S. A. Islam, L. K. Sah, and S. Katkoori, "High-level synthesis of key-obfuscated rtl ip with design lockout and camouflaging," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 26, no. 1, oct 2020.

[14] C. Pilato, F. Regazzoni, R. Karri, and S. Garg, "Tao: Techniques for algorithm-level obfuscation during high-level synthesis," in *DAC*, 2018, pp. 1–6.

[15] J. Frey and Q. Yu, "Exploiting state obfuscation to detect hardware trojans in noc network interfaces," in *MWSCAS*, 2015, pp. 1–4.

[16] A. Patooghy, M. Hasanzadeh, A. Sarihi, M. Abdelrehim, and A.-H. A. Badawy, "Securing network-on-chips against fault-injection and crypto-analysis attacks via stochastic anonymous routing," *J. Emerg. Technol. Comput. Syst.*, vol. 19, no. 3, jun 2023.

[17] A. Sarihi, A. Patooghy, M. Hasanzadeh, M. Abdelrehim, and A.-H. A. Badawy, "Securing network-on-chips via novel anonymous routing," in *NOCS*, 2021, pp. 29–34.

[18] R. Chakraborty and S. Bhunia, "Harpoon: An obfuscation-based soc design methodology for hardware protection," *IEEE Tran. on Computer Aided Design*, vol. 28, pp. 1493 – 1502, 11 2009.

[19] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," ser. DAC, 2012, p. 83–89.

[20] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *IEEE HOST*, 2015, pp. 137–143.