ObNoCs: Protecting Network-on-Chip Fabrics Against Reverse-Engineering Attacks

DIPAL HALDER, MANEESH MERUGU, and SANDIP RAY, University of Florida, USA

Modern System-on-Chip designs typically use Network-on-Chip (NoC) fabrics to implement coordination among integrated hardware blocks. An important class of security vulnerabilities involves a rogue foundry reverse-engineering the NoC topology and routing logic. In this paper, we develop an infrastructure, ObNoCs, for protecting NoC fabrics against such attacks. ObNoCs systematically replaces router connections with switches that can be programmed after fabrication to induce the desired topology. Our approach provides provable redaction of NoC functionality: switch configurations induce a large number of legal topologies, only one of which corresponds to the intended topology. We implement the ObNoCs methodology on Intel Quartus™ Platform, and experimental results on realistic SoC designs show that the architecture incurs minimal overhead in power, resource utilization, and system latency.

CCS Concepts: • **Security and privacy → Hardware reverse engineering;**

Additional Key Words and Phrases: Hardware security, obfuscation, supply-chain attacks, communication fabrics

ACM Reference format:
Dipal Halder, Maneesh Merugu, and Sandip Ray. 2023. ObNoCs: Protecting Network-on-Chip Fabrics Against Reverse-Engineering Attacks. *ACM Trans. Embedd. Comput. Syst.* 22, 5s, Article 112 (September 2023), 21 pages.
https://doi.org/10.1145/3609107

1 INTRODUCTION

System-on-Chip (SoC) designs are architected by integration of predesigned hardware blocks, — referred to as "Intellectual Properties" or "IPs", — which communicate through a variety of Network-on-Chip (NoC) fabrics to realize the system functionality. The NoC fabrics realize on-chip communication using a collection of routers connected to form a topology optimized for the workload of the targeted SoC. The NoC fabrics have the advantage of being more scalable and power-efficient than traditional bus-based communications and have proliferated significantly in recent years as central coordination vehicles in SoC designs.

A sensitive security asset in NoC-based designs is the topology of the NoC itself. Reverse-engineering the NoC topology enables a rogue entity to infer a variety of system-level parameters

This research has been partially supported by the Defense Advanced Research Projects Agency under the SAHARA Program, Contract No. HR0011-21-3-0001.
Authors' address: D. Halder, M. Merugu, and S. Ray, University of Florida, Gainesville, FL, USA, 32611; emails: {dipal.halder, maneesh.merugu}@ufl.edu, sandip@ece.ufl.edu.

1539-9087/2023/09-ART112 $15.00
https://doi.org/10.1145/3609107

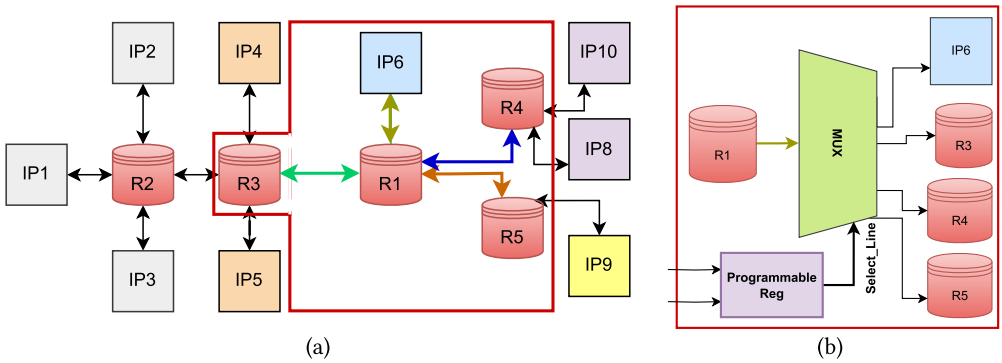


Fig. 1. NoC interconnect transformation topology obfuscation. (a) An NoC-based SoC Design Example with a simple Tree Topology. (b) Transformation of the outgoing edge of router R1.

of the SoC, including the targeted latency for a variety of inter-IP communications on different workloads, sensitivity of communication at different life cycles, etc. On the other hand, SoCs today are designed and fabricated through a complex, globally distributed supply chain, which includes foundries, packaging, assembly, testing facilities, etc. A rogue player in this supply chain can perform *reverse-engineering attacks*, i.e., reconstruct the NoC topology and parameters (see Section 2.3). The reverse-engineering can target a layout-level design of the SoC (in case of a foundry) or a fabricated wafer or chip (in case of assembly, packaging, and testing facility). It is crucial to protect such develop techniques to protect NoC topologies from reverse-engineering in such untrusted facilities.

In this paper, we address this security problem through a novel obfuscation technology to conceal the topology of NoC interconnects in SoC designs. Given an NoC interconnect, our framework ObNoCs enables systematic transformation of router ports in the interconnect with programmable switches. The switches can be configured to realize a variety of topologies after fabrication by programming a specific set of state elements, only one of which corresponds to the original topology of the fabric. We refer to the switch configuration corresponding to the original topology as the *activation package*. We show how to implement this approach with low overhead using a collection of multiplexers controlled by programmable registers. The ObNoCs methodology has been implemented on top of Intel™ Quartus™ platform [2], and our experiments on realistic SoC designs show that the overhead of ObNoCs in power or resource utilization is minimal.

In this section, we first provide a toy motivational example to illustrate the key idea of ObNoCs. We then discuss the contributions of the paper in greater detail.

1.1 Motivational Example

Consider the SoC design shown in Figure 1(a). It includes an NoC with five routers organized in a simple tree network. Suppose we want to redact the connection $R_1 \rightarrow \text{IP}_6$, Figure 1(b) shows the method to transform this connection. In particular, R_1 is connected to the input of a 1×4 demultiplexer and the output of the demultiplexer is connected to the four different hardware blocks. We call this circuitry the *DEMUX switch*. The control of the DEMUX switch is connected to a register that can be programmed after fabrication, and the control bits determine which IP is actually connected to R_1 in this port. If the register is programmed with the bits `00` then R_1 would be connected in this port to IP_6 which corresponds to the original topology. Note that each pattern

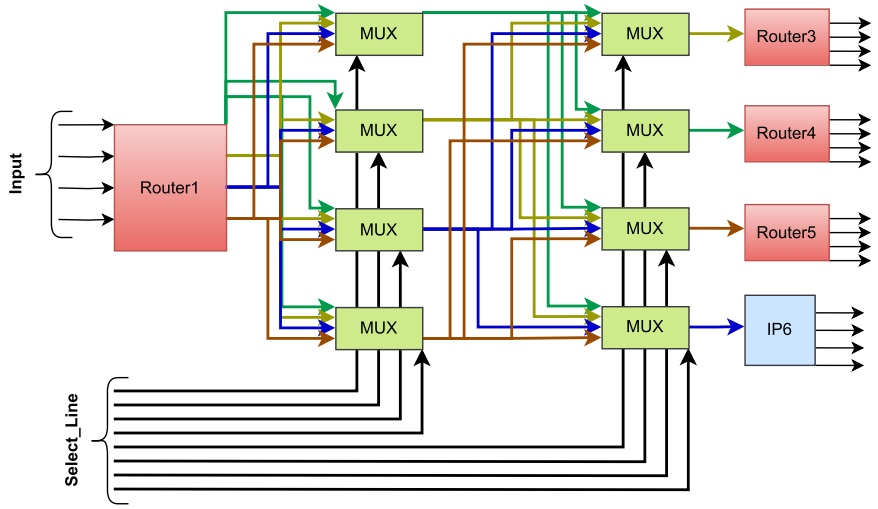


Fig. 2. Example of custom MUX based transformation for outgoing signals of Router1 to IP6, Router 3, Router 4, and Router 5.

of bits from 00 through 11 corresponds to *some connection*, e.g., 01 would correspond to $R_1 \rightarrow R_3$ and 10 to $R_1 \rightarrow R_4$.

Figure 2 shows how the transformation above would be implemented for router R_1. In particular, we exercise 4×1 MUX to achieve a *MUX-DEMUX* based switch for each port of the router that we need to redact. This makes the architecture of ObNoCs 2-stage where the first stage acts as the DEMUX switch and the second stage as the MUX switch. The control bits for this *MUX-DEMUX* switch are connected to a register which can be programmed at runtime.

From the example, it is clear that the NoC obfuscated by ObNoCs would have the same topological behavior as the unobfuscated NoC if the "right" control bits are provided. We refer to this bit pattern as *activation package*. On the other hand, the assignment of the activation package bits to the register only happens after the fabricated SoC is returned to the OEM from the foundry; when the SoC is in an untrusted facility (e.g., foundry, assembly, or test), the activation package is not available. Furthermore, other bit patterns in addition to the activation package also correspond to other perfectly viable NoC topologies; indeed, without access to the activation package, there is no way to decide which one of the viable topologies was the one actually intended by the designer. We make this statement formal in Section 4. Correspondingly, a facility attempting to reverse-engineer the SoC does not have a way to determine the correct topology (among the viable ones) without access to the activation package.

1.2 Contributions of the Paper

There has been significant recent research on hardware obfuscation. We recount some of this work in Section 6. A popular approach for hardware obfuscation is *logic locking*, where a hardware logic design is extended with a dedicated locking circuitry such that a correct output is produced only after a special bit sequence (or "activation package") has been provided. ObNoCs can be viewed as a focused locking technique specifically targeted for NoC interconnects. However, there are important differences between traditional logic locking and ObNoCs. First, logic locking has been generally targeted for IP-level designs. Since the traditional locking circuitries incur significant overhead, the cost of applying it to a complete SoC would be prohibitive. Furthermore, logic-locking techniques have been shown to be vulnerable to a variety of attacks [14]. In contrast, ObNoCs provides

provable protection against reverse-engineering attacks from an adversary without access to the activation package.

The paper makes the following important contributions.

- To our knowledge, ObNoCs represents the first obfuscation methodology for system-level interactions of SoC designs.
- ObNoCs realizes transformation of interconnect topology through a novel switching architecture that systematically redacts connection structures using programmable MUXs.
- Our methodology provides provable assurance of protection of obfuscated topologies against reverse-engineering attacks.
- We demonstrate through extensive experimental results that the methodology incurs minimal overhead in power and resource utilization and minimal impact on timing and path delay on realistic SoC designs.

The remainder of the paper is organized as follows. Section 2 discusses the relevant background. Section 3 describes the ObNoCs obfuscation architecture and Section 4 provides a security analysis. We discuss experimental results in Section 5. We discuss related work in Section 6 and conclude in Section 7.

2 BACKGROUND

2.1 NoC Fabrics

The primary coordination mechanism for IPs in an SoC is message-based communication. Consequently, the communication fabrics constitute a crucial component of an SoC design. Traditional communication fabrics used point-to-point, crossbar, or bus architectures. More recently, NoC fabrics have gained popularity in industrial SoCs. An NoC involves a collection of routers connected to realize a target topology. Many industrial SoC designs make use of a tree topology, although other networks such as cycle, mesh, or torus are also in use [1]. Routers in an NoC typically include configurable routing tables which can be reconfigured if necessary by the operating system at boot-time. A key advantage of NoC architecture is the flexibility provided in implementing power management with low overhead as it is possible to simply shut off (or reducing high-speed functionality) routers in the sub-network when message communication through the sub-network is reduced. Note of course that these benefits do come with a number of challenges, including complex optimization requirements for achieving power utilization, fault tolerance, quality of service (QoS), and CAD support for NoC design [17]. For instance, to address the challenge of high throughput performance, Pathania et al.[29] introduced a unique topology-based performance heterogeneity which exploits many-core heterogeneity to extract more performance. Indeed, our work also accounts for resource utilization as an overhead metric to trade off for achieving security goals.

2.2 SoC Supply Chain Security

Semiconductor design has evolved over the past decade into a global enterprise incorporating 3PIP (third-party IP) vendors, IC design houses, fabrication labs, and testing facilities dispersed over multiple countries and continents. The globalization in the supply chain has been driven by a number of factors, including aggressive time-to-market requirements, miniaturization of VLSI technology, increased fabrication and validation costs, etc. In particular, the exponential shrinkage of transistor nodes over the past decades has enabled the IC designers to pack complex, multi-core and many-core designs with advanced performance in area and power-constrained chip designs, with a resultant increase in the price of fabrication. As a result, a majority of the semiconductor companies are going fabless and outsourcing chip fabrication to globally distributed remote foundries. An unfortunate upshot is that it is possible for a rogue foundry to mount a variety of at-

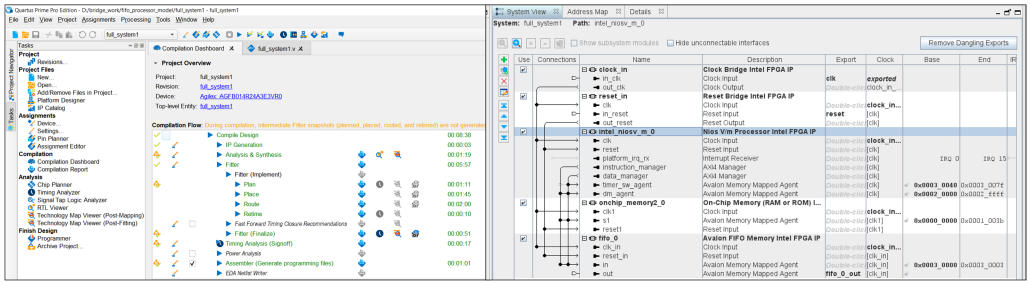


Fig. 3. ObNoCs implementation environment in Quartus Prime and Platform Designer.

tacks subverting the design and production of the system. One class of attacks involves alteration of the design by inserting malicious Trojan circuitry. Another class of attacks includes reverse engineering, possibly targeting the entire SoC or some of the key IPs. Other supply chain attacks include cloning, piracy, counterfeiting, recycling, and overproduction.

2.3 Reverse-Engineering Attacks

Reverse engineering is the technique of extracting data from a chip by dissecting its functional elements and constituent parts. In the context of NoCs, the goal of the reverse-engineering attacks is to extract information of the NoC topology from an SoC implementation (either as netlist or as a fabricated silicon). Although there have not been dedicated techniques specifically targeted to reverse-engineer NoC topologies, this can be accomplished by common methods such as imaging, side-channel analysis, and micro probing all of which have been extensively researched. Torrance et al. have discussed several types of reverse engineering, of which the products are disassembling, system-level analysis, and circuit extraction in [42]. Gomez et al. described a method that uses image processing to recreate a chip's gate netlist after it has been decapped and delayered. Using this method, the whole netlist is rebuilt by teaching an image processing tool to identify common cell layout patterns and extract routing information [16]. Additionally, there are purely algorithmic ways to reverse engineering digital circuits for the purpose of inferring a high-level netlist with components like register files, adders, and counters from unstructured netlists. For instance, Subramanyan et al. [41] segmented the netlist into potential/candidate modules and by determining functionality using methods akin to design synthesis. The goal is to make it easier for a human analyst to comprehend the functionality of an unstructured netlist by identifying as many components as possible. Holler et al. [18] describe a method that combines image processing and machine learning to improve the effectiveness and precision of reverse engineering procedures. Botero et al. have used machine learning algorithms to recognize patterns and categorize the parts, while image processing techniques can be used to extract features from photos of the hardware [9]. Recently, there has also been work on discovering the algorithms to find the specification of the extracted design [8].

2.4 Quartus Environment Basics

Although the ObNoCs infrastructure is independent of the underlying framework used for designing the NoCs being obfuscated, the platform is integrated with the Intel Quartus environment and all SoCs discussed in this paper are designed in Quartus. Quartus is a programmable logic device design software targeted for the Intel FPGA platform. It is an effective tool for creating digital circuits and systems on chips in the realm of embedded systems. Design entry and verification, synthesis, place-and-route, time analysis, and programming are among the features that it offers. The SoC's size, performance, and power consumption can be improved by placing and routing

digital circuits more efficiently thanks to the tool's sophisticated algorithms. However, it can be complex to use and requires significant computational resources. Figure 3 shows the general Quartus environment including Platform Designer. The Intel Quartus Prime software includes Platform Designer, which offers a user-friendly environment for creating and modifying SoCs. It has an Intel IP library that can be used to create any SoC design with the ability to add custom logic and set up interfaces in Platform Designer. Along with options for system configuration and optimization, Platform Designer offers the flexibility to add any Intel outside IP to the design. Platform designer automatically creates the NoC interconnect during SoC generation. The number of routers inside the NoC interconnect varies with the size of the SoC.

3 ObNoCs ARCHITECTURE

3.1 MUX Insertion Methodology

Our motivating example in Section 1.1 identifies the key ingredients of the ObNoCs methodology. ObNoCs takes an SoC design (typically in RTL) together with user directives for routers targeted for obfuscation. The output of the ObNoCs transformation is a transformed design with the routers obfuscated through redaction using the *MUX-DEMUX* switches as discussed above.

Algorithm 1 defines the procedure for inserting the MUX-DEMUX switch at each router $\mathcal{R}$. Roughly, for each $\mathcal{S}_i$ such that there is a link $\mathcal{R} \rightarrow \mathcal{S}_i$ and each $\mathcal{D}_i$ such that there is a link $\mathcal{D}_i \rightarrow \mathcal{R}$, we instantiate a programmable MUX for each end of the router links. Here the function *RandomizeConnections* introduces non-determinism in connecting the outputs of a MUX, e.g., in our motivating example, it will enable us to non-deterministically map the four candidate blocks to the four outputs of the DEMUX switch for R_1. Finally, for each MUX insertion, Algorithm 1 additionally records the bit pattern that must be loaded to the control register to recover the original topology. For the configuration of the MUX on the Source side as mentioned in Algorithm 1, each input of the generated MUX ($MUX_{R\_S_i}$) would be connected to all the signals of the Router outputs $\mathcal{R}[\mathcal{S}_i]$. The MUX output would be appended to a custom array MUX_out_list, which would serve as inputs to the Destination MUX's ($MUX_{R\_D_i}$). For the correct transformation involving the communication links, the MUX select lines have to be configured accordingly, for which the correct configuration for each Source MUX is stored in a variable P_{Ri}, with the input Source signal, the intended destination signal along with the connections of the source MUX ($in_i, S_i, D_i,\ MUX_{R\_S_i}[out]$), respectively. Using a *RandomizeConnections* function, the input signals of the source MUX are randomized, to increase the uncertainty of retracing in case of reverse engineering in a brute force attack scenario. The correct MUX configuration, using the select lines would be retrieved by the array index i.e.the position of the actual input signal by referring to the signals stored in P_{Ri}. The right select line configuration, Sel_R, is then appended to the comprehensive activation package ($Act\_Pkg[R]$) used to configure our entire transformation. The bit pattern required for the DEMUX switches (resp., MUX switches) at the output (resp. *input*) ports of router $\mathcal{R}$ will be referred to as the *output* (resp. *input*) *activation package* for $\mathcal{R}$. The bit pattern obtained by concatenating the activation packages of all routers in the NoC will be referred to as the *activation package of the NoC*, or simply *activation package* (Act_Pkg).

Remark 1. Algorithm 1 inserts MUXes on a port of router $\mathcal{R}$ such that the control configurations induce topologies involving blocks that are already connected to some port of $\mathcal{R}$. However, note that the only requirement for functional correctness is that the intended topology is realized under the MUX control configurations that correspond to the correct activation package; no assumption is required for other configurations. ObNoCs can exploit this observation to connect

ALGORITHM 1: Algorithm for *Custom MUX* Insertion

Input: Router $\mathcal{R}$ with source and destination connections $\mathcal{R}[\mathcal{S}_i]$, $\mathcal{R}[\mathcal{D}_i]$
Output: Activation package for $\mathcal{R}$: $\mathcal{R}[\mathcal{P}_i]$, *Act_Pkg*
Source (DEMUX) Configuration
for single *Router connection* $\mathcal{R}$: $\mathcal{R}[\mathcal{S}_i]$, $\mathcal{R}[\mathcal{D}_i]$. $\mathcal{D}_i \leftarrow \mathcal{S}_i$
  for all $\mathcal{S}_i$ in $\mathcal{R}[\mathcal{S}]$, **do**
    Generate $MUX_{R\_S_i}$.
    MUX input : $MUX_{R\_S_i}[in_i] \leftarrow \mathcal{R}[\mathcal{S}_i]$,
    MUX output : $MUX_{R\_S_i}[out]$
    MUX_out_list ***append*** $MUX_{R\_S_i}[out]$
    Store : $P_{Ri} = [in_i, S_i, D_i, MUX_{R\_S_i}[out]]$
    RandomizeConnections$(MUX_{R\_S_i}[in_i])$
    $Sel_R \leftarrow$ ***getIndex***$(MUX_{R\_S_i}, in_i, S_i)$
    $Act\_Pkg[R]$ ***append*** Sel_R
  end for
  for all $\mathcal{D}_i$ in $\mathcal{R}[\mathcal{D}_i]$, **do**
    Generate $MUX_{R\_D_i}$:
    $MUX_{R\_D_i}[in_i]$ ***append*** $P_{Ri}[MUX_{R\_S_i}[out]]$
    $MUX_{R\_D_i}[out] \leftarrow \mathcal{R}[\mathcal{D}_i]$
  end for
end for
return *Act_Pkg*, MUX_out_list
Destination (MUX) Configuration
Input: Router $\mathcal{R}$ with source and destination connections $\mathcal{R}[\mathcal{S}_i]$, $\mathcal{R}[\mathcal{D}_i]$, $MUX_{R\_S_i}[out]$, MUX_out_list.
Output: Activation package for $\mathcal{R}$: $\mathcal{R}[\mathcal{P}_i]$, *Act_Pkg*
for all *Router connections* $\mathcal{R}_n$: $\mathcal{R}_n[\mathcal{D}_n]$.
    MUX input : $MUX_{R_n\_D_i}[in_i] \leftarrow MUX_{R_n\_S_i}[out]$
    $MUX_{R_n\_D_i}[in_i - 1] \leftarrow$ randomSelect(MUX_out_list, *size*-1)
    MUX output : $MUX_{R_n\_D_i}[out]$
    RandomizeConnections$(MUX_{R_n\_D_i}[in_i])$
    $Sel_{R_n} \leftarrow$ ***getIndex***$(MUX_{R_n}, in_i, MUX_{R_n\_D_i}[out], D_i)$
    $Act\_Pkg[R_n\_S_i]$ ***append*** Sel_{Rn}
  end for
return *Act_Pkg*

$\mathcal{R}$ with other IPs creating additional topologies under different control configuration. This can provide additional protection against SAT attacks and brute force adversaries.

3.2 Activation Package Loader

An NoC design transformed by ObNoCs would realize the original topology when the registers connected to the controls of MUX and DEMUX switches are configured with the activation package. On the other hand, it is not possible to load the activation package directly on the fabricated SoC via parallel load from external inputs given the limited number of input pins available. ObNoCs correspondingly also introduces circuitry to load the activation package through a bit-shifting paradigm.

Figure 4 shows the architecture for loading the activation package. The design is inspired by scan chain designs in VLSI testing. To streamline insertion, we create a single register bank, referred to as *activation package load register* (AP_LOAD_ REG) for holding the activation package of the NoC. *AP_LOAD_REG* works on the Serial Input Parallel Output (SIPO) mechanism, where the

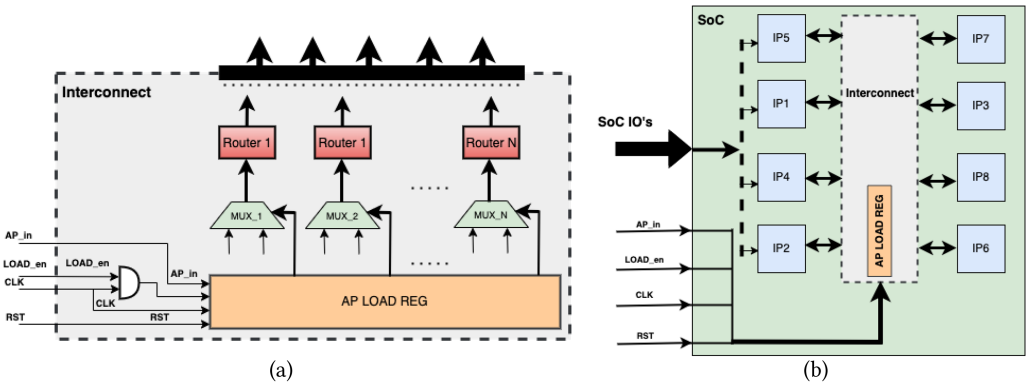


Fig. 4. (a) Architecture of the Activation Package Loader Circuit. (b) Transformed SoC with Integrated Activation Package Loader.

activation package is provided one bit at a time per each clock cycle through the 1-bit primary input *AP_in*. The signal *LOAD_en* is gated with the clock and the bits are loaded into the shift register only when *LOAD_en* is asserted. Once the entire activation package is loaded into the register, the *LOAD_en* is then de-asserted and the parallel outputs of the register are connected to the respective programmable MUX select lines, with the correct configuration realizing the intended topology.

4 SECURITY ANALYSIS

4.1 Theoretical Guarantee

The key security guarantee of ObNoCs is that the original topology of the NoC interconnect cannot be derived from the obfuscated design without knowledge of the activation package. To make this statement formal, let us fix a design $\mathcal{D}$ and let $\mathcal{D}_o$ be an obfuscated design generated from ObNoCs with an n-bit activation package $A \in \{0,1\}^n$. We call the topology graph $\mathcal{T}$ of $\mathcal{D}$ the ***intended topology***. Obviously, $\mathcal{N}$ consists of the IPs and routers in $\mathcal{D}$. We call any topology graph $\mathcal{G}$ over $\mathcal{N}$ a ***legal topology*** if for any node $v \in \mathcal{N}$, the degree of v in $\mathcal{T}$ is the same as the degree of v in $\mathcal{G}$. Given any binary string $b \in \{0,1\}^n$, we refer to graph $\mathcal{G}_b$ derived from $\mathcal{D}_o$ by setting the MUX controls to be the bitstring b as the *topology of $\mathcal{D}_o$ induced by b*. Obviously, the topology of $\mathcal{D}_o$ induced by the activation package A is the intended topology. Then the following property defines the security guarantee of ObNoCs.

Legality Enumeration. Given a binary string $b \in \{0,1\}^n$, let m be the average degree in the topology of $\mathcal{D}_o$ induced by b. Then the total number of unique legal topologies is $m!$.

The property is a straightforward from the ObNoCs construction and the observation that the connections induced by the MUX and DEMUX switches at the input and output of each router respect the original in-degree and out-degree of each node. A consequence of the property is that the intended topology $\mathcal{T}$ is different from all the other legal topologies induced by the obfuscated design $\mathcal{D}_o$ only by the specifics of the bitstream that defines the activation package A. It follows that an adversary without access to A, cannot distinguish the intended topology from other legal topologies.

Remark 2. Informally, the *intended topology* here is really the topology that the designer desires, while all topologies that can be realized through some values of the configuration bitstream are legal topologies. It follows that the intended topology is simply one of the legal topologies, in

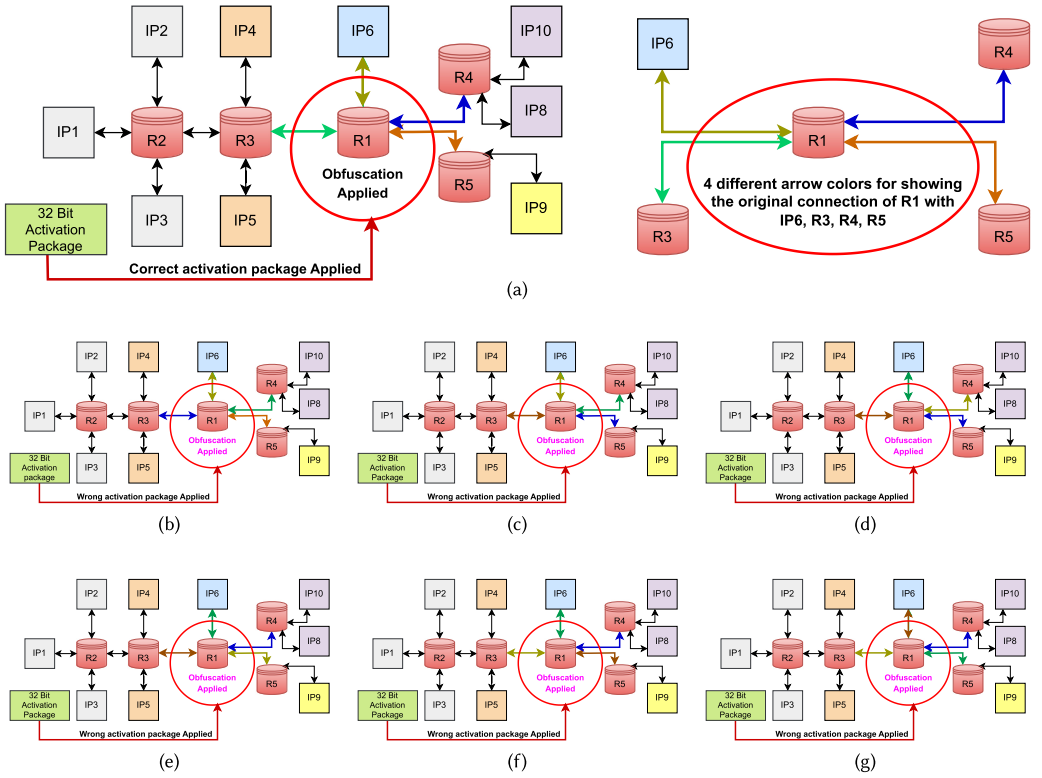


Fig. 5. Different Functional Topology generation using ObNoCs. 5(a) indicates the original topology, while the other six topologies are legally functional, but none of them have the Original Functionality.

particular the specific legal topology that corresponds to the activation package that the designer has in mind. However, other than the specifics of the activation package there is nothing else to distinguish the intended topology from other (unintended) ones. In particular, each legal topology would constitute an SoC in which IPs could communicate with one another. Note that some values of the configuration bitstream would result in a *non-functional* topology in the sense that the same port of two IPs is mapped to the same output. Figure 5 shows some possible *unintended* and *non-functional* topologies, together with the (unique) *intended topology*. It is obvious that an adversary can detect non-functional bit configurations; however, a large number of legal topologies and the uniqueness of the intended topology among the legal ones implies that identifying the intended topology is equivalent to identifying the activation package.

To explain the consequences of the above analysis a bit further, consider the SoC design shown in Figure 1 For each router obfuscation, we used 16 4x1 MUXs. Since each 4x1 MUX requires two select lines, the total number of bits for the activation package is 32. However, not every bit pattern corresponds to a unique legal topology, due to the recombination of the combinational logic involved. Even if only one router is obfuscated for the design in Figure 1, the number of legal topologies is $4! = 24$. Figure 5 illustrates 7 of the 24 possible topologies (including the intended one) whereas Figure 5(a) indicates the intended topology upon insertion of the correct activation package. The four different colored arrows in the red encircled portion with router R_1 indicates the original connection with the R_3, R_4, R_5 and IP_6. These connections from R_1 can be controlled by our ObNoCs which has been depicted in the rest of the figures in Figure 5. On the other hand,

there is no *a priori* reason to determine which one of the legal topologies is the intended one (from a functional standpoint) other than the "idiosyncrasies" of the designer's choice. Furthermore, as our empirical evaluation shows (see below) the different legal topologies induce very different computations; consequently, the adversary cannot get away with identifying a topology close enough to the intended. Finally, the number of possible legal topologies can be compounded by simply composing the obfuscation in stages: for a 2-stage MUX-based obfuscation from Figure 2, there are $4! \cdot 4! = 576$ legal topologies.

We conclude this section with a brief comment on the storage of the activation package. Recall that our threat model considers an untrusted foundry or testing facility that has access to the layout information for the unfabricated SoC as well as silicon implementation of the fabricated SoC. Since the topology information is redacted by ObNoCs through the MUX insertion and the control bits of the MUX are only programmed subsequently with the activation package by the OEM, neither the layout nor the fabricated SoC has the activation package stored in the chip as long as the SoC is in control of the untrusted entity. However, when the SoC goes to field, it includes the activation package (which is obviously required to make the SoC functional). In this paper, we do not specifically discuss the storage of the activation package in field since it is outside the scope of our threat model. However, ObNoCs does not need any specialized mechanism for storage and application of activation package; any mechanism for storing keys used for unlocking an obfuscated circuit (e.g., through logic locking) can be applied for this purpose, e.g., a typical mechanism involves utilizing a small tamper-proof ROM in the design to hold the key bits.[1]

4.2 Resiliency Against Known Attacks on MUX-based Obfuscation

MUX-based transformation is a common strategy for obfuscating hardware functionality. Correspondingly, there has been significant research on attacks against such obfuscation. To demonstrate the effectiveness of ObNoCs we consider some common attack strategies and explain how ObNoCs provides resiliency against these attacks.

Remark 3. Note that there has been no previous work to our knowledge on obfuscation of NoC topologies. Although traditional logic locking techniques can be applied to SoC designs with NoC fabrics, they are generally applicable only to gate-level netlists. Performing them on a complete SoC, while possible in principle, incurs significant computational overhead and is infeasible in practice. Indeed, ObNoCs avoids netlist designs works on RTL-level designs to avoid such computational overhead in obfuscation. Furthermore, the presence of encrypted IPs in Quartus makes it practically infeasible to extract a complete netlist from Quartus to enable a direct comparison with other obfuscation techniques. Furthermore, given these factors, a direct implementation and evaluation of traditional attacks (which also apply to gate-level netlists) could not be performed as part of direct resiliency evaluation. However, ObNoCs architecture in spite of being a MUX-based framework, is robust against typical attacks on MUX-based obfuscation techniques as explained below.

ML-based Attacks. Machine Learning attacks on obfuscation [4, 6] use a variety of ML techniques to predict MUX configurations, thereby inferring unbofuscated links, e.g., Alrahis et al. [6] use GNN to perform such predictions. However, the *success* of the attacks relies on the fact that the obfuscation is performed on a netlist *post synthesis*. Correspondingly, the ML strategies are employed on unobfuscated links indicating dependence on the technology library. However,

[1]Current state-of-the-art locking techniques generally use an on-chip ROM although techniques for storing the key in the cloud together with other provisioned assets is being explored. Generally, the requirement is that the storage should be tamper-proof and communication of the key for its application should satisfy non-observability requirements.

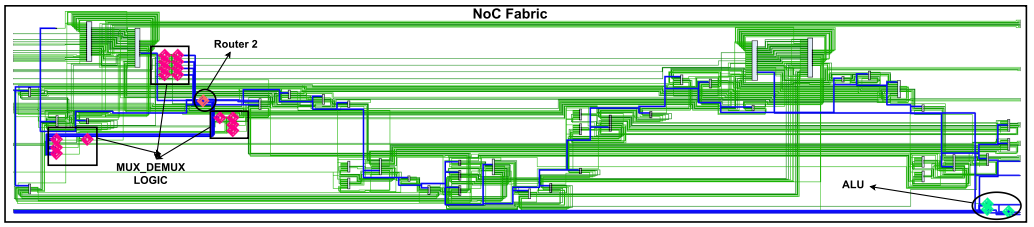


Fig. 6. Schematic of an NoC interconnect in Xilinx Vivado along with an ALU.

in ObNoCs, this assumption is broken since the obfuscation is performed *before synthesis* and MUXs used are architectural entities with no dependence on the underlying technology library for insertion.

SAT Attacks. SAT attacks [40] form another popular class of attacks to break hardware obfuscation. They depend on an oracle (e.g., a fabricated design on which one can execute test patterns) to answer the sequence of SAT queries. The SAT queries are computed to enable reconstruction of specific bits of the activation package. Resiliency against SAT attacks depends on the *query complexity*, i.e., the number of SAT queries involved in the attack. For ObNoCs, the query complexity is defined by the number of legal topologies derived in Section 4.1. Furthermore, recall that ObNoCs enables extensibility by extending the number of router connections with additional IPs, which can be applied to achieve additional resiliency to SAT attacks. Finally, since SAT attack (and other attacks) cannot distinguish the intended topology from other legal topologies, the correct activation package even if produced as a candidate bit pattern through SAT attack, cannot be vetted as indeed inducing the intended topology (see below).

Specific Attack Instances. We have analyzed the resilience of ObNoCs on two specific attack instances: (1) redundancy-based attack strategy [25], and (2) the SNAPSHOT deep learning attack [37]. The key idea of redundancy-based strategy is that an incorrect activation package would lead to incorrect functionality, which is exploited through different redundancy levels to infer the correct key. SNAPSHOT relies on a deep learning strategy to predict the correct activation package that corresponds to "intended" functionality. However, for ObNoCs, legal topologies that differ from the intended one would still produce a functional SoC, simply not the one intended by the SoC designer. Consequently, since the technique relies on functional validation to detect incorrect behavior, it cannot distinguish counterfeit SoCs with legal topologies from the intended one.

Remark 4. Obviously, any of the strategies above can find legal topologies. This leaves open the possibility that the adversary can simply enumerate (and fabricate) all counterfeit SoC variants corresponding to legal topologies. This is indeed a threat if the number of legal topologies is small. However, as discussed above, ObNoCs can apply staging strategy to expand the space of legal topologies by multiplicative factors. Note that adding a single stage would multiply the number of legal topologies by a factor of 24. Indeed, simply using the 2-stage strategy of Figure 2 with 576 possible legal topologies makes it infeasible for the adversary to fabricate all the SoC variants.

4.3 Empirical Analysis

The security analysis above shows that it is infeasible for the adversary to reverse-engineer the intended topology from ObNoCs obfuscation. One objection to that thesis could be that many

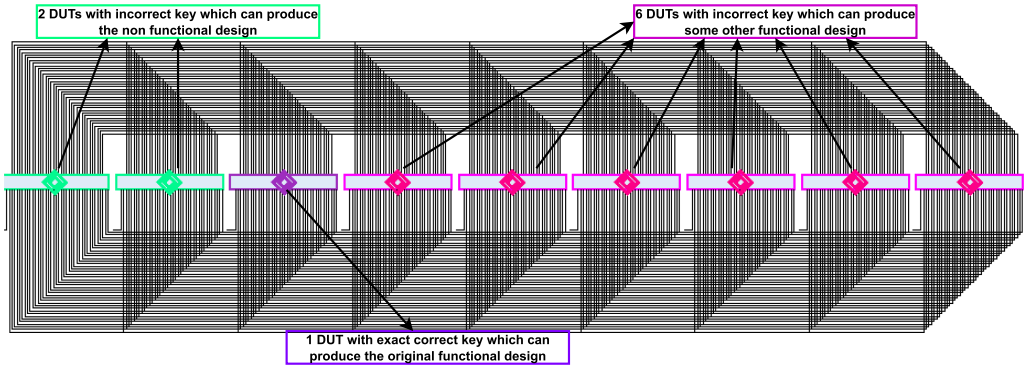


Fig. 7. Multiple DUTs for the same Obfuscated NoC to compare the output for different activation package combination.

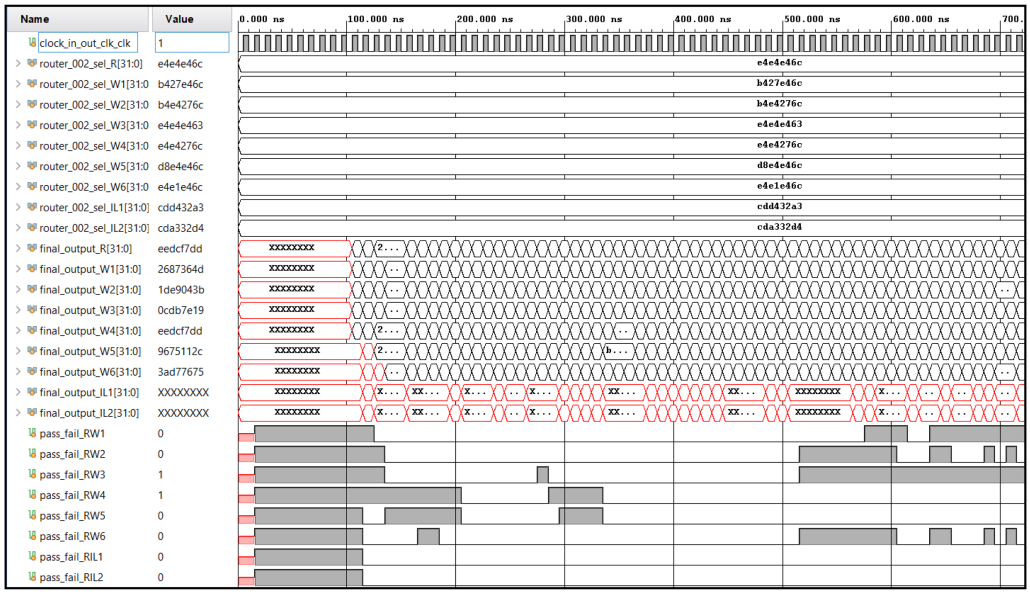


Fig. 8. Comparison of the Output of all 9 DUTs on nine different activation package insertion.

topologies could be functionally similar, so a counterfeit SoC that does not create the intended topology could still serve as a valid counterfeit. To evaluate this possibility, we considered the a NoC interconnect and created a simulation environment with an ALU module integrated for effective analysis of the functionality. Figure 6 shows a Xilinx Vivado setup for this empirical analysis.[2] In this figure, the router including all the required MUXes for the obfuscation and the ALU module have been pointed out for a better understanding of the design. The key idea is to drive the interconnect with different workloads corresponding to inter-IP communications and observe the functional impact on the computation of the ALU. The simulation used a total of 9

[2]Our primary design and experimental testbed is Quartus, which we used for overhead analysis as discussed in Section 5. In this figure, the router including all the MUXes used for this obfuscation, and the ALU module have been highlighted for a better understanding of the design. However, for the empirical security analysis, we used Vivado for ease of simulation of different instances. The interconnect model was exported from Platform Designer to Vivado to perform this simulation.

DUTs (Figure 7) of which only one is driven by the correct activation package while the rest are incorrect. Figure 8 shows the result of the simulation. The correct functionality is derived only with the correct activation package. In the simulation we have provided 9 different activation package for 9 DUTs. On the other hand, the ALU is still functional (but incorrect and different from the functionality in the intended topology) in 6 DUTs, which correspond to legal topologies different from the intended one. Note that functional validation of the ALU would consider all 7 topologies to be correct, thwarting attacks that depend on functional validation.

Recall that the theoretical analysis above showed that the obfuscated design can correspond to a number of different legal topologies of which only one is the intended one. To substantiate this, we have created a simulation environment in Xilinx Vivado. For this simulation, we have taken only the interconnect generated in Platform Designer as our Design Under Test (DUT).

The interconnect has been shown in Figure 6 and we have integrated one ALU module in it for better analysis. The NoC interconnect fabric is responsible for all the data communication for the SoC. To check data transfer from the NoC, we have taken the ALU result as our reference. In our simulation, there are total 9 DUTs in Figure 7 where only one of them is driven by the correct activation package, 8 are driven by the incorrect activation package.

5 OVERHEAD ANALYSIS

5.1 Benchmark SoCs

A key challenge in research on SoC security validation is the lack of an appropriate testbed for evaluation. Evaluation of ObNoCs requires SoC designs with a realistic NoC-based communication fabrics connecting disparate IP cores that reflect realistic design complexity. Unfortunately, there is no open-source SoC design satisfying these requirements. To address this problem, we have developed our own SoC benchmarks, shown in Figure 9. We use three SoCs with different interconnect sizes to ensure that our results are generalizable. The three SoCs are derivatives of each other, comprised of similar IPs but with different interconnect topologies. We refer to the SoCs as *MultiSoC*, *LUTSoC*, and *FlashSoC* respectively. Each SoC is comprised of six subsystems, with each subsystem containing specific IPs; The CPU subsystem is made up of two NIOS processors; the memory subsystem consists of an on-chip memory and a DMA controller; the Communication subsystem includes an SPI module, an Ethernet, and a Serial Flash module; and the Debugging subsystem consists of a JTAG and a Performance Counter IP.

While undoubtedly simple compared to an industrial product, the SoCs are non-trivial; the RTL for the unobfuscated (original) SoC for each variant runs to about 100,000 lines of RTL, with the interconnect fabric accounting for 10,500 lines. The Block Memory bits for the SoCs are 3.6×10^5, 9.8×10^5, and 1.2×10^6 respectively, and there are respectively 12, 15 and 16 routers in the NoC fabrics.

5.2 Platform and Implementation Details

We implemented ObNoCs using the Intel Quartus Pro 21.3. For our SoC design and integration, we have used the Platform Designer frontend of Quartus and implemented the ObNoCs methodology for systematically redacting various router connections as reported below. For evaluating ObNoCs, we have designed the SoCs using the Platform Designer (PD) frontend of Quartus Pro 21.3 and implemented on the Agilex F Series FPGA, AGFA012R24A2E2V. Figure 10 shows the schematic diagram of the obfuscated SoC where the enclosed red rectangle indicates the NoC interconnect of the SoC after transformation by ObNoCs.

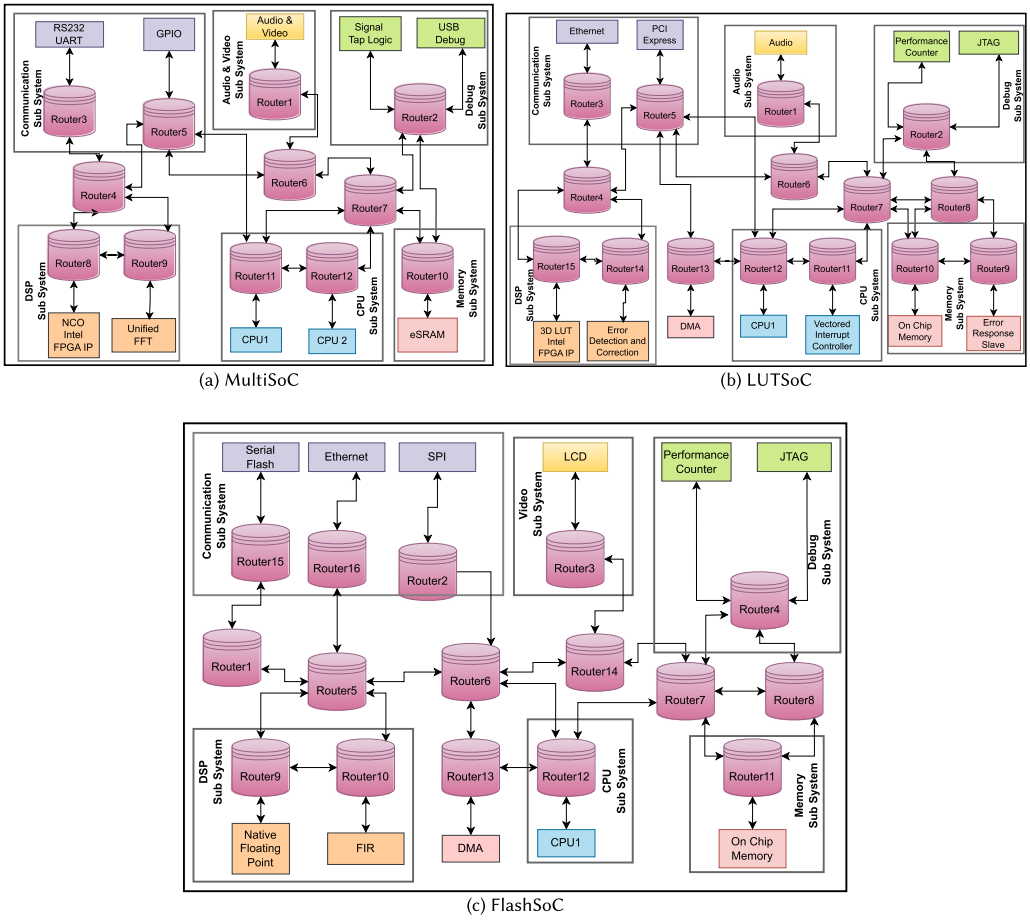


Fig. 9. Three SoC designs Used for ObNoCs Evaluation. All three designs have been implemented on Quartus Prime with Platform Designer.

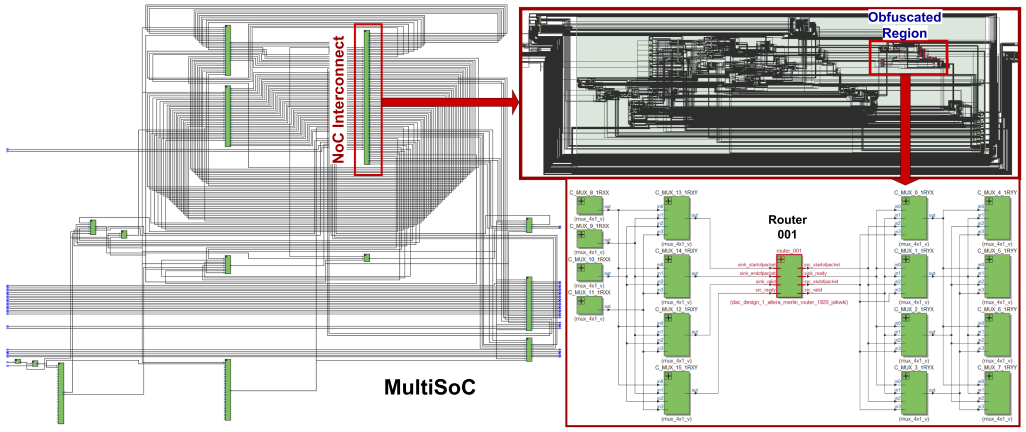
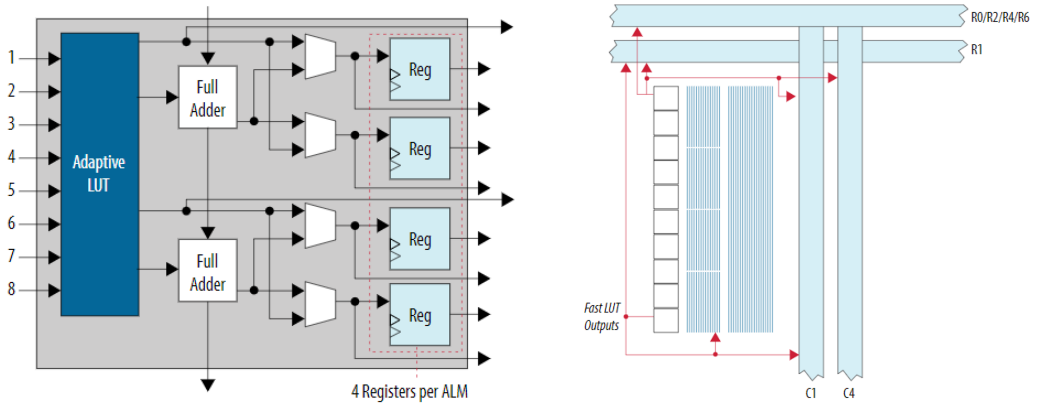


Fig. 10. Schematic of Benchmark SoC in Figure 9(a) from Quartus along with the obfuscated Interconnect using ObNoCs technique.

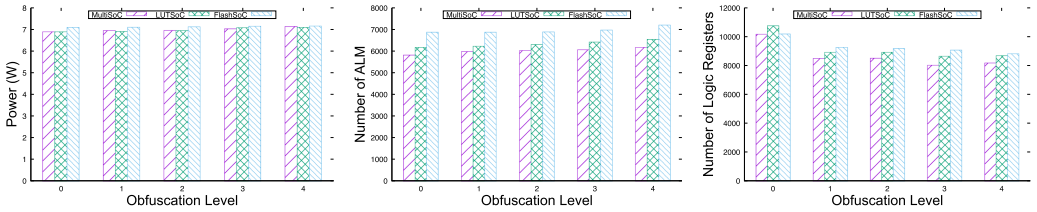
Table 1. Obfuscation Level based on No. of Routers

Obfuscation Level	0	I	II	III	IV
No. of Routers	0	2	4	8	16



(a) ALM (b) MLAB

Fig. 11. ALM and MLAB Architecture in Intel® Agilex™ 7 FPGAs.



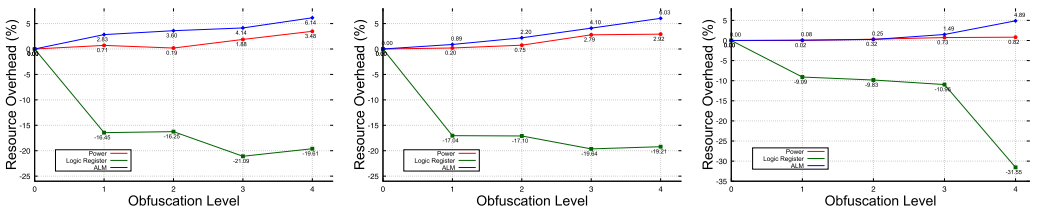
(a) Power (W) (b) Number of ALM (c) Number of Logic Registers

Fig. 12. Resource Analysis for *MultiSoC*, *LUTSoC* and *FlashSoC* at different obfuscation level.

5.3 Result Analysis

To provide a fine-grained analysis of the overhead induced by our interconnect transformation, we define five levels of obfuscation (0-IV) as shown in Table 1 where obfuscation level 0 indicates no obfuscation, i.e., the original design. Estimates of power and hardware resource overheads have been derived from the Quartus platform. Quartus provides three different metrics on the resource overhead: (1) power consumption; (2) number of required Adaptive Logic Modules (ALMs) and (3) number of Logic registers (which estimates the state elements). The Intel Agilex device consists of Logic Array Blocks (LABs) shown in Figure 11(b) and is implemented by enhanced ALM modules mentioned in Figure 11(a) which allows efficient implementation of logic functions. Each ALM consists of two Adaptive Look-up Tables (ALUTs), two dedicated embedded Adders and four dedicated Registers [3]. We have followed the user guidelines from [2] to carefully analyze the resources from Quartus generated reports.

Figures 12(a), 12(b), and 12(c) show the resource utilization for different levels of obfuscation. Since for each obfuscation level there are variations in utilization depending on which specific routers are selected as obfuscation targets (see below), we consider the average utilization for each combination, e.g., the resource overhead for Obfuscation Level I is calculated by



(a) MultiSoC (b) LUTSoC (c) FlashSoC

Fig. 13. Resource Overhead Rate for all 3 SoCs at different obfuscation levels.

considering the $\binom{n}{2}$ results for designs derived by obfuscating each possible pair of routers and considering the arithmetic mean where n is the number of routers available inside the NoC fabric. Power and number of ALM both are increasing with the obfuscation level for all three SoCs *MultiSoC*, *LUTSoC* and *FlashSoC*. Note that the power and number of ALM units are increasing with the obfuscation level increment whereas the number of logic registers are decreasing.[3]

Figures 13(a), 13(b), and 13(c) show the resource overhead rate results corresponding to various obfuscation levels for the three designs. Note that this results in an average of only 5.68% overhead increase in the number of ALM modules, 2.41% overhead increase in power consumption, 23.46% reduction in usage of logic registers, where all of the three resource parameters have been compared with their respective unobfuscated designs. Nevertheless, we can conclude that the addition of logic by ObNoCs appreciably impacts very little in resource overhead.

It is interesting to understand the decrease in the logic register overhead which can be attributed to the increase in the number of ALMs, with each ALM consisting of 4 additional registers. Since the obfuscation technique is implemented using pure combinational logic, the primary effect of the overhead is an increase in the number of ALMs as well as total power. This combinational logic is implemented with ALMs, which leaves fewer resources available to build logic registers. In general, all of the design's logic functions, including both combinational and sequential logic, must share the resources in an FPGA, such as ALMs and flip-flops. Additionally, the number of flip-flops that can be used to build the registers are constrained if the combinational logic calls for a large number of inputs or outputs. Furthermore, additional combinational logic introduced during obfuscation (e.g., MUX and DEMUX structures) permits Quartus to perform aggressive optimizations in logic synthesis.

Finally, the additional hardware logic introduced for obfuscation may affect the critical path of the system. From Figure 14, we observe that the critical path increases due to the obfuscation level. This suggests that the increased security due to the increasing degree of obfuscation induces a trade-off by adding combinational gate delay. However, the delay induced is significant only for obfuscation levels III and IV. Furthermore, in SoCs with a larger number of IPs, the combinational delay in the interconnect would be proportionately less. This timing resource analysis is solely based on Quartus Timing Analyzer report [2] with considering slack and critical path as timing parameters.[4]

[3]We have taken the ALM and Logic Registers values obtained from Quartus synthesis report to represent area overhead. Note that it is not possible to export the netlist synthesized from Quartus to an external environment (e.g., Xilinx Vivado or Synopsys Design Compiler) since the Quartus environment integrates a number of encrypted IPs. Consequently, area estimation using those traditional tools is not possible. For an ASIC implementation, the overhead result can be different.
[4]Our observations are based on pre-synthesis estimates, which can be different from post-synthesis implementation. Implementing ObNoCs in FPGA and considering the correlation between pre-synthesis and post-synthesis timing correlation

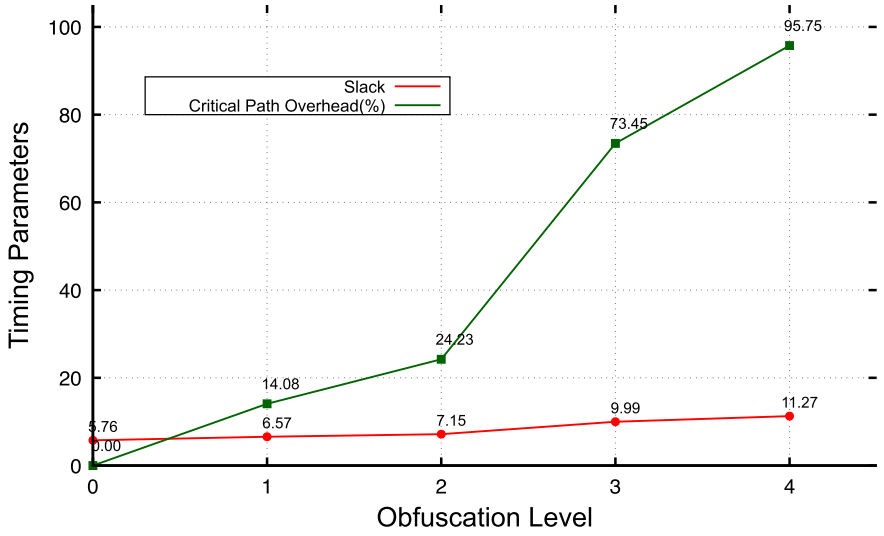


Fig. 14. Timing Analysis for *LUTSoC* at different obfuscation levels.

6 RELATED WORK

There has been significant research on hardware obfuscation techniques primarily for protecting hardware IPs against theft, reverse engineering, structural, data integrity and eavesdropping attacks. Logic locking methods provide enhanced IP security by introducing additional logic to the design [4, 11, 46]. Logic locking technique has been used for protecting integrated IPs in modern SoC architectures from various hardware security threats. While extensive research has been conducted on the applicability and effectiveness of logic locking over the past decade, the security and strength of existing techniques remain questionable due to evolving attacks [22]. State Space Obfuscation techniques additionally include extra state elements to obfuscate the finite state machines [10]. Correspondingly there has been work on attacking various logic-locking schemes based on SAT-based attacks [40] and defenses proposed against such attacks [27, 44, 45]. Paar et al. [14] developed two foundry-level invasive attack schemes based on probing and mask modification which could be used to break the encryption. Other advanced attempts to break obfuscation schemes have been proposed [5]. As machine learning spreads, new strategies for breaking logic-locking systems have emerged. this paper Sisejkovic et al. [38] presents an overview of current breakthroughs in logic-locking attacks and defenses. Many logic locking systems now in use have a flaw that can be exploited by an attacker to recover key bits from the locked chip's design without the need for a working replica of the device. Massad et al. [26] suggests a new logic locking method, Meerkat, to solve this weakness and ensure security by utilizing Reduced Ordered Binary Decision Diagrams.

Shamsi et al. [35] introduced an interconnect locking scheme based on cross bars and layout inclusive. But this technique is targeted only for IPs rather than a complete SoC. Kolbe et al. [24] proposed a dynamic obfuscation technique by using reconfigurable logic and interconnect blocks (RIL-Blocks) from the emerging spin-based devices. This technique is robust against the SAT attacks but is only feasible for the devices with Magnetic Random Access Memory (MRAM). By using fully programable Logic and Routing block (PLR), Kamali et al. proposed a Full-Lock obfuscation

will be considered in future work. However, such correlations are not germane to ObNoCs and follow standard practice of timing correlations between various timing models used pre-synthesis vis-a-vis timing on actual artifacts.

technique [23]. Shamsi et al. [34] introduced dummy paths to the circuit to get the cyclic obfuscation which is also robust against SAT attack. Zhang et al. [47] offers C-SAR, a security architecture to protect against SAT attacks. By expanding the key search space and extending the clock cycles of attack inputs, C-SAR protects against SAT attacks. They demonstrate that the cost of C-SAR is controllable because it only rises linearly as a function of key bits. Juretus et al. provide analysis on the performance of a different obfuscation technique against SAT attack [20]. The obfuscation technique has also been applied at the post-fabrication level using the post-fabrication Transistor Level Programming (TRAP) which can stand against both brute-force and oracle-guided SAT attacks [36].

Authenticated encryption schemes for NoC security [12, 15, 33], focus on the encryption-decryption of the data packets emanating out of an IP in the SoC. Deb Nath et al. [13] proposed an architecture-level solution for run-time detection of security-critical events in SoC designs with NoC fabrics by implementing security policies implemented through a specialized security-policy engine. Securing inter-IP communications by detection of security policy violations by Meng et al. [28] developed a property-based model comparison strategies to validate the NoC communications against information flow violations. There has been work on mitigating the effects of data integrity and hardware Trojans in NoC designs [7, 21]. Wassel et al. [43] developed techniques to increase security by temporal partitioning of data into different domains. Route randomization techniques have also been employed to detect and mitigate various side channel attacks [19, 30–32, 39].

7 CONCLUSION AND FUTURE WORK

We have developed a new methodology, ObNoCs, for redacting NoC topologies in SoC designs to protect against reverse engineering by untrusted fabrication facilities. To our knowledge, ObNoCs represents the first technique for redacting system-level functionality in SoC designs. ObNoCs systematically replaces router connections with switches that can be programmed after fabrication. We show how to implement this redaction functionality efficiently using MUX and DEMUX logic. Our experimental results on representative SoC implementations show that the resource and overhead of transformations through ObNoCs is minimal. Furthermore, ObNoCs provides provable obfuscation of NoC connection against reverse-engineering adversaries.

In future work, we plan to apply ObNoCs on industrial-scale SoC designs. We plan to increase the level of redaction by replacing the routing tables with custom configurable registers leading to enhanced security. We also plan to extend our implementation of ObNoCs into a fully automated CAD infrastructure.

ACKNOWLEDGEMENTS

We thank Kostas Amberiadis for his advice and help during the project, and the Intel Quartus team for help with numerous Quartus issues.

REFERENCES

[1] [n. d.]. Intel Baytrail Products. https://ark.intel.com/content/www/us/en/ark/products/codename/55844/bay-trail.html

[2] [n. d.]. Intel Quartus Prime Software. https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime.html/. [Online].

[3] [n. d.]. Intel® Agilex™ I-Series FPGA and SoC FPGA. https://www.intel.com/content/www/us/en/products/details/fpga/agilex/i-series/docs.html

[4] A. Abdulrahman, A. S. Abuadbba, A. Aldabbagh, and O. Hasan. 2019. Sweep to the secret: A constant propagation attack on logic locking. In *Proceedings of the 2019 IEEE Asian Hardware-Oriented Security and Trust Symposium (AsianHOST'19)*. IEEE, 1–6. https://doi.org/10.1109/AsianHOST45689.2019.8966485

[5] Niels Albartus, Ali Bayrak, and Michael Zohner. 2020. Dana universal dataflow analysis for gate-level netlist reverse engineering. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2020, 2 (2020), 129–155.

[6] Lilas Alrahis, Satwik Patnaik, Muhammad Shafique, and Ozgur Sinanoglu. 2021. MuxLink: Circumventing learning-resilient MUX-locking using graph neural network-based link prediction. *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 694–699.

[7] Dean Michael Ancajas, Harold John Perez, James Angelo Garcia, Paolo Jonathan Isidro, Jeffrey Jade Garcia, Marco Angelo Marcelino, Rafael Salvador, Rianne Villanueva, and Prospero Jr. Flores. 2014. Fort-NoCs: Mitigating the threat of a compromised NoC. In *Proceedings of the 51st Annual Design Automation Conference (DAC'14)*. 1–6.

[8] Leonid Azriel, Julian Speith, Nils Albartus, Ran Ginosara, Avi Mendelson, and Christof Paar. 2021. A Survey of Algorithmic Methods in IC Reverse Engineering. Cryptology ePrint Archive, Paper 2021/1278. https://doi.org/10.1007/s13389-021-00268-5 https://eprint.iacr.org/2021/1278

[9] Ulbert J. Botero, Ronald Wilson, Hangwei Lu, Mir Tanjidur Rahman, Mukhil A. Mallaiyan, Fatemeh Ganji, Navid Asadizanjani, Mark M. Tehranipoor, Damon L. Woodard, and Domenic Forte. 2021. Hardware trust and assurance through reverse engineering: A tutorial and outlook from image analysis and machine learning perspectives. *J. Emerg. Technol. Comput. Syst.* 17, 4, Article 62 (jun 2021), 53 pages. https://doi.org/10.1145/3464959

[10] Rajat Subhra Chakraborty and Swarup Bhunia. 2009. HARPOON: An obfuscation-based SoC design methodology for hardware protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28, 1 (2009), 97–109. https://doi.org/10.1109/TCAD.2008.2006838

[11] Rajat Subhra Chakraborty and Swarup Bhunia. 2010. RTL hardware IP protection using key-based control and data flow obfuscation. In *2010 23rd International Conference on VLSI Design*. 405–410. https://doi.org/10.1109/VLSI.Design.2010.54

[12] Steev Charles and Prabhat Mishra. 2020. Securing network-on-chip using incremental cryptography. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'20)*. IEEE, 53–58.

[13] A. P. Deb Nath, S. Kumar, and S. Mukherjee. 2020. Security assurance of system-on-chip designs with NoC fabrics. *IEEE Signal Processing Society Newsletter* 37, 5 (2020), 31–36. https://doi.org/10.1109/MSP.2020.2992936

[14] Sebastian Engels, Fadi A. El-Moussa, Joseph Rosenblatt, Andreea Homescu, Thomas Schneider, and Stefan Katzenbeisser. 2019. The end of logic locking? A critical view on the security of logic locking. *IACR Cryptology ePrint Archive* 2019 (2019), 747.

[15] C. H. Gebotys and R. J. Gebotys. 2003. A framework for security on NoC technologies. In *IEEE Computer Society Annual Symposium on VLSI, 2003. Proceedings.* 113–117. https://doi.org/10.1109/ISVLSI.2003.1183361

[16] Hector Gomez, Ckristian Duran, and Elkim Roa. 2019. Defeating silicon reverse engineering using a layout-level standard cell camouflage. *IEEE Transactions on Consumer Electronics* 65, 1 (2019), 109–118. https://doi.org/10.1109/TCE.2018.2890616

[17] Klaus Hofmann. 2012. Network-on-chip: Challenges for the interconnect and I/O-architecture. In *2012 International Conference on High Performance Computing & Simulation (HPCS'12)*. 252–253. https://doi.org/10.1109/HPCSim.2012.6266920

[18] Mirko Holler, Michal Odstrcil, Manuel Guizar-Sicairos, Maxime Lebugle, Elisabeth Müller, Simone Finizio, Gemma Tinti, Christian David, Joshua Zusman, Walter G. Unglaub, Oliver Bunk, Jörg Raabe, A. F. J. Levi, and Gabriel Aeppli. 2019. Three-dimensional imaging of integrated circuits with macro- to nanoscale zoom. *Nature Electronics* 2 (2019), 464–470.

[19] Leandro Soares Indrusiak, James Harbin, and Martha Johanna Sepulveda. 2017. Side-channel attack resilience through route randomisation in secure real-time Networks-on-Chip. In *2017 12th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC'17)*. 1–8. https://doi.org/10.1109/ReCoSoC.2017.8016142

[20] Kyle Juretus and Ioannis Savidis. 2020. Characterization of in-cone logic locking resiliency against the SAT attack. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39, 8 (2020), 1607–1620. https://doi.org/10.1109/TCAD.2019.2925387

[21] Manoj Kumar Jyv, Ayass Kant Swain, K SudeendraKumar, Sauvagya Ranjan Sahoo, and Kamala Kanta Mahapatra. 2018. Run time mitigation of performance degradation hardware Trojan attacks in network on chip. *2018 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 738–743.

[22] Hadi Mardani Kamali, Kimia Zamiri Azar, Farimah Farahmandi, and Mark Tehranipoor. 2022. Advances in Logic Locking: Past, Present, and Prospects. Cryptology ePrint Archive, Paper 2022/260. https://eprint.iacr.org/2022/260 https://eprint.iacr.org/2022/260

[23] Hadi Mardani Kamali, Kimia Zamiri Azar, Houman Homayoun, and Avesta Sasan. 2019. Full-lock: Hard distributions of SAT instances for obfuscating circuits using fully configurable logic and routing blocks. In *Proceedings of the 56th Annual Design Automation Conference 2019* (Las Vegas, NV, USA) *(DAC'19)*. Association for Computing Machinery, New York, NY, USA, Article 89, 6 pages. https://doi.org/10.1145/3316781.3317831

[24] Gaurav Kolhe, Soheil Salehi, Tyler David Sheaves, Houman Homayoun, Setareh Rafatirad, Manoj P D Sai, and Avesta Sasan. 2021. Securing hardware via dynamic obfuscation utilizing reconfigurable interconnect and logic blocks. In *2021 58th ACM/IEEE Design Automation Conference (DAC'21)*. 229–234. https://doi.org/10.1109/DAC18074.2021.9586242

[25] Leon Li and Alex Orailoglu. 2023. Redundancy attack: Breaking logic locking through oracleless rationality analysis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 4 (2023), 1044–1057. https://doi.org/10.1109/TCAD.2022.3192793

[26] Mohamed El Massad, Jun Zhang, Siddharth Garg, and Mahesh V. Tripunitara. 2017. Logic locking for secure outsourced chip fabrication: A new attack and provably secure defense mechanism. *ArXiv* abs/1703.10187 (2017).

[27] Travis Meade, Yier Jin, Mark Tehranipoor, and Shaojie Zhang. 2016. Gate-level netlist reverse engineering for hardware security: Control logic register identification. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS'16)*. 1334–1337. https://doi.org/10.1109/ISCAS.2016.7527495

[28] Xingyu Meng, Kshitij Raj, Sandip Ray, and Kanad Basu. 2023. SeVNoC: Security validation of system-on-chip designs with NoC fabrics. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 42, 2 (2023), 672–682. https://doi.org/10.1109/TCAD.2022.3179307

[29] Anuj Pathania and Jörg Henkel. 2018. Task scheduling for many-cores with S-NUCA caches. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE'18)*. 557–562. https://doi.org/10.23919/DATE.2018.8342069

[30] Christian Reinbrecht, Peter Puschner, and Christian Steger. 2020. Guard-NoC: A protection against side-channel attacks for MPSoCs. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'20)*. IEEE, 47–52.

[31] Christian Reinbrecht, Peter Puschner, Christian Steger, and Thomas Krieg. 2016. GOSSIP NoC–avoiding timing side-channel attacks through traffic management. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI (ISVLSI'16)*. IEEE, 197–202.

[32] Cezar Reinbrecht, Altamiro Susin, Lilian Bossuet, Georg Sigl, and Johanna Sepúlveda. 2016. Side channel attack on NoC-based MPSoCs are practical: NoC Prime+Probe attack. In *2016 29th Symposium on Integrated Circuits and Systems Design (SBCCI'16)*. 1–6. https://doi.org/10.1109/SBCCI.2016.7724051

[33] Johanna Sepúlveda, Andreas Zankl, Daniel Flórez, and Georg Sigl. 2017. Towards protected MPSoC communication for information protection against a malicious NoC. *Procedia Computer Science* 108 (2017), 1103–1112. https://doi.org/10.1016/j.procs.2017.05.139

[34] Kaveh Shamsi, Meng Li, Travis Meade, Zheng Zhao, David Z. Pan, and Yier Jin. 2017. Cyclic obfuscation for creating SAT-Unresolvable circuits. In *Proceedings of the on Great Lakes Symposium on VLSI 2017* (Banff, Alberta, Canada) *(GLSVLSI'17)*. Association for Computing Machinery, New York, NY, USA, 173–178. https://doi.org/10.1145/3060403.3060458

[35] Kaveh Shamsi, Meng Li, David Z. Pan, and Yier Jin. 2018. Cross-lock: Dense layout-level interconnect locking using cross-bar architectures. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI* (Chicago, IL, USA) *(GLSVLSI'18)*. Association for Computing Machinery, New York, NY, USA, 147–152. https://doi.org/10.1145/3194554.3194580

[36] Mustafa M. Shihab, Jingxiang Tian, Gaurav Rajavendra Reddy, Bo Hu, William Swartz, Benjamin Carrion Schaefer, Carl Sechen, and Yiorgos Makris. 2019. Design obfuscation through selective post-fabrication transistor-level programming. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE'19)*. 528–533. https://doi.org/10.23919/DATE.2019.8714856

[37] Dominik Sisejkovic, Farhad Merchant, Lennart M. Reimann, Harshit Srivastava, Ahmed Hallawa, and Rainer Leupers. 2021. Challenging the security of logic locking schemes in the era of deep learning: A neuroevolutionary approach. *J. Emerg. Technol. Comput. Syst.* 17, 3, Article 30 (may 2021), 26 pages. https://doi.org/10.1145/3431389

[38] Dominik Sisejkovic, Lennart M. Reimann, Elmira Moussavi, Farhad Merchant, and Rainer Leupers. 2021. Logic locking at the frontiers of machine learning: A survey on developments and opportunities. In *2021 IFIP/IEEE 29th International Conference on Very Large Scale Integration (VLSI-SoC'21)*. 1–6. https://doi.org/10.1109/VLSI-SoC53125.2021.9606979

[39] Ismael L. Soares, César A. M. Pereira, and Luigi Carro. 2019. Side-channel protected MPSoC through secure real-time networks-on-chip. *Microprocessors and Microsystems* 68 (2019), 102888.

[40] Praveen Subramanyan, Swarup Bhunia, and Debdeep Mukhopadhyay. 2015. Evaluating the security of logic encryption algorithms. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST'15)*. IEEE, 112–117.

[41] Pramod Subramanyan, Nestan Tsiskaridze, Wenchao Li, Adrià Gascón, Wei Yang Tan, Ashish Tiwari, Natarajan Shankar, Sanjit A. Seshia, and Sharad Malik. 2014. Reverse engineering digital circuits using structural and functional analyses. *IEEE Transactions on Emerging Topics in Computing* 2, 1 (2014), 63–80. https://doi.org/10.1109/TETC.2013.2294918

[42] Randy Torrance and Dick James. 2011. The state-of-the-art in semiconductor reverse engineering. In *2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC'11)*. 333–338.

[43] Hassan M. G. Wassel, Ying Gao, Jason K. Oberg, Ted Huffmire, Ryan Kastner, Frederic T. Chong, and Timothy Sherwood. 2013. SurfNoC: A low latency and provably non-interfering approach to secure networks-on-chip. *SIGARCH*

Comput. Archit. News 41, 3 (jun 2013), 583–594. https://doi.org/10.1145/2508148.2485972

[44] Muhammad Yasin, Bodhisatwa Mazumdar, Jeyavijayan J. V. Rajendran, and Ozgur Sinanoglu. 2016. SARLock: SAT attack resistant logic locking. In *Proceedings of the 2016 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2016*. Institute of Electrical and Electronics Engineers Inc., 236–241. https://doi.org/10.1109/HST.2016.7495588

[45] Muhammad Yasin, Jeyavijayan Jv Rajendran, Ozgur Sinanoglu, and Ramesh Karri. 2016. On improving the security of logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35, 9 (Sept. 2016), 1411–1424. https://doi.org/10.1109/TCAD.2015.2511144 Publisher Copyright: © 1982-2012 IEEE..

[46] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan (JV) Rajendran, and Ozgur Sinanoglu. 2017. Provably-secure logic locking: From theory to practice. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS'17)*. Association for Computing Machinery, New York, NY, USA, 1601–1618. https://doi.org/10.1145/3133956.3133985

[47] Junyao Zhang, Paul Bogdan, and Shahin Nazarian. 2023. C-SAR: SAT attack resistant logic locking for RSFQ circuits. *ArXiv* abs/2301.10216 (2023).

Received 23 March 2023; revised 2 June 2023; accepted 30 June 2023