A Security-Aware and LUT-Based CAD Flow for the Physical Synthesis of hASICs

Zain Ul Abideen, *Graduate Student Member, IEEE*, Tiago Diadami Perez, *Graduate Student Member, IEEE*, Mayler Martins, and Samuel Pagliarini, *Member, IEEE*

***Abstract*—Numerous threats are associated with the globalized integrated circuit (IC) supply chain, such as piracy, reverse engineering, overproduction, and malicious logic insertion. Many obfuscation approaches have been proposed to mitigate these threats by preventing an adversary from fully understanding the IC (or parts of it). The use of reconfigurable elements inside an IC is a known obfuscation technique, either as a coarse grain reconfigurable block (i.e., eFPGA) or as a fine grain element (i.e., FPGA-like lookup tables). This article presents a security-aware CAD flow that is LUT-based yet still compatible with the standard cell-based physical synthesis flow. More precisely, our CAD flow explores the FPGA-ASIC design space and produces heavily obfuscated designs where only small portions of the logic resemble an ASIC. Therefore, we term this specialized solution a hybrid ASIC (hASIC). Nevertheless, even for heavily LUT-dominated designs, our proposed decomposition and pin swapping algorithms allow for performance gains that enable performance levels that only ASICs would otherwise achieve. On the security side, we have developed novel template-based attacks and also applied existing attacks, both oracle-free and oracle-based. Our security analysis revealed that the obfuscation rate for an SHA-256 study case should be at least 45% for withstanding traditional attacks and at least 80% for withstanding template-based attacks. When the 80% obfuscated SHA-256 design is physically implemented, it achieves a remarkable frequency of 368 MHz in a 65-nm commercial technology, whereas its FPGA implementation (in a superior technology) achieves only 77 MHz.**

***Index Terms*—Hardware obfuscation, hybrid ASIC (hASIC), LUT-based obfuscation, reverse engineering, secure ASIC design.**

I. Introduction

NOWADAYS, high-performance and energy-efficient integrated circuits (ICs) are enablers in a variety of application domains. However, this demands the fabrication of ICs in advanced technology nodes. Current predictions are that the sales of semiconductor devices will rise to $680B in 2022, the first time this mark has been surpassed in a calendar year since 2020 [1]. In tandem, the majority of IC design houses are adhering to a globalized supply chain to outsource fabrication from pure-play foundries. Even very large semiconductor companies rely on the so-called fab-for-hire model [2], [3], a framework that originates from the technological and financial challenges of developing and maintaining a foundry. The estimated cost to build a 3-nm production line is $15B–$20B [4]. The trend is clear: more than ever, fabless design companies rely on outsourcing the manufacturing of their ICs.

While this business model enables design houses to have access to high-end manufacturing, the integrity and trustworthiness of the ICs are potentially affected. For manufacturing an IC, the design house must share a blueprint of the IC with the foundry. This blueprint inevitably exposes all aspects of the IC and its many parts. A rogue element within the foundry can entirely or partially copy the design, i.e., the foundry and its employees are considered *potential adversaries*. Many potential threats are associated with the untrusted fabrication aspect of a globalized IC supply chain [5]. Such threats include tampering, counterfeiting, reverse engineering, and overproduction.

Numerous techniques have been devised to protect against the aforementioned security threats. Countermeasures to secure an IC also apply to a malicious end user that can be interested in reverse engineering a design. Noteworthy examples of countermeasures are Logic Locking [6], [7], [8], [9], [10], IC Camouflaging [11], [12], [13], Split Manufacturing [14], [15], and FPGA-like obfuscation approaches [16], [17], [18], [19], [20], [21], [22], [23], [24]. The latter style of obfuscation attempts to exploit an FPGA (or FPGA-like) fabric, where the functionality of the circuit is hidden by the configuration and the *bitstream serves as a key to unlock the design*.

Generally, the fabric in an FPGA device contains many reconfigurable blocks that can be leveraged for obfuscation purposes. The ability to reconfigure a device does incur performance penalties (i.e., FPGA versus ASIC). Being so, custom solutions where only a small portion of the design is reconfigurable have been sought, a solution typically termed eFPGA. This work also takes advantage of this possibility. A visualization of the obfuscation landscape is given in Fig. 1. As illustrated, performance increases if we move from right to left. Contrarily, obfuscation and flexibility increase if we move from left to right. However, we argue that *neither extremes of the landscape are a good design point* for circuits with stringent security and performance constraints. A midpoint solution is a better tradeoff, which is precisely the motivation for our work. We term our midpoint solution a hybrid ASIC (hASIC).

In [25], we have described an initial attempt at exploring and automating the design spaces captured in Fig. 1. In this

Manuscript received 13 July 2022; revised 25 October 2022 and 3 February 2023; accepted 10 February 2023. Date of publication 14 February 2023; date of current version 20 September 2023. This work was supported by the Project "ICT Programme" which was supported by the European Union through the ESF. This article was recommended by Associate Editor J. Rajendran. *(Corresponding author: Zain Ul Abideen.)*

Zain Ul Abideen, Tiago Diadami Perez, and Samuel Pagliarini are with the Department of Computer Systems, Centre for Hardware Security, Tallinn University of Technology, 12616 Tallinn, Estonia (e-mail: zain.abideen@taltech.ee; tiago.perez@taltech.ee; samuel.pagliarini@taltech.ee).

Mayler Martins is with SRG, Synopsys Inc., Mountain View, CA 94085 USA (e-mail: mayler.martins@synopsys.com).

Digital Object Identifier 10.1109/TCAD.2023.3244879

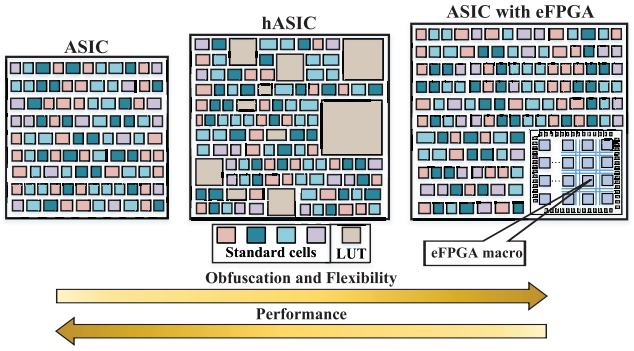

Fig. 1. Design obfuscation landscape.

work, we extend and improve our results considerably while keeping the same general theme: we seek to obfuscate a circuit by generating a hybrid design that consists of a reconfigurable portion and static logic. The *reconfigurable* part provides the obfuscation while the *static* logic provides performance benefits. We perform our design space exploration at the block level. Finally, the architecture of the generated block is a mix of *reconfigurable* and *static* cells. The reconfigurable part is implemented with programmable LUTs; the circuit is largely nonfunctional until it is programmed.

Earlier obfuscation techniques utilizing reconfigurable elements have focused on keeping the reconfigurable part as small as possible. Understandably, the goal would be to avoid large performance and area overheads. However, we emphasize (and later provide results) that proper hiding of the circuit's intent requires a *high degree of obfuscation* that is generally not investigated in the state-of-the-art. For this reason, in [25], we have proposed a CAD tool for automatically obfuscating a design, thus, generating a specialized solution called hASIC that is compatible with standard-cell-based flows and current design and fabrication practices. In this work, we markedly extend the CAD tool from [25]. The main contributions of this work are as follows.

1) Specialized algorithms for performance improvement of hASIC designs, including LUT decomposition and pin swapping approaches.
2) An analysis of performance versus obfuscation and area versus obfuscation tradeoffs for numerous designs, including known benchmarks.
3) A detailed analysis (physical synthesis) of performance, power, and area versus obfuscation for SHA-256, including tapeout-ready layouts in a 65-nm commercial technology.
4) Thorough analysis of hASIC's security against custom attacks and known oracle-based and oracle-less attacks.

II. CAD Flow for hASIC

Our CAD flow utilizes a custom tool named tuneable design obfuscation technique using hASIC (TOTe). Our custom tool produces an hASIC design with reconfigurable and static logic. For the reconfigurable portion, we implement the logic utilizing the notion of programmable lookup tables (LUTs)—same as in FPGAs. The complete process for obfuscating a design is fully automated and infers a marginal increase in design time (when compared to a traditional ASIC flow).

At its core, TOTe looks for critical paths and replaces "slow" reconfigurable elements with "fast" static ones. From this point of view, TOTe's design decisions are decoupled from security decisions. Later, in Section VI, we introduce a well-defined threat model and provide insights into the security of an hASIC design. A designer using TOTe only has to define an obfuscation target obf_c which is the percentage of LUTs that should remain reconfigurable (obfuscated).

A. Overview of TOTe

Initially, a commercial FPGA synthesis tool is utilized to synthesize the design under obfuscation (DUO), described in the register-transfer level (RTL) form. The DUO does not require any particular change in its representation. Then, the commercial FPGA synthesis tool generates a synthesized netlist and a timing report. This netlist includes all the typical FPGA primitives, i.e., LUTs, MUXs, and FFs.

Next, TOTe takes the ASIC standard cell library of choice as well as the outputs generated by the FPGA synthesis. The parser of TOTe reads the elements from the netlist and the paths from the timing report, which are then processed by a timing engine. The primary goal of TOTe is to *replace FPGA cells for ASIC cells*. More precisely, TOTe selects LUTs in the critical path (i.e., the path with the highest delay) and replaces them with standard cells that implement the same logic (except the programmability aspect is taken out). This process is repeated until enough LUTs have been converted to standard cells according to a user-provided obfuscation target (obf_c). The obfuscation target determines the ratio of the LUTs that must remain programmable. Replacing LUTs for static logic reduces the area, power, and delay (thus, improving the performance of the design). Finally, an obfuscated *hybrid* Verilog file containing reconfigurable and static LUTs is generated as the output.

In order to finalize the hASIC design, a commercial physical synthesis tool is used to implement it. Then, the foundry receives the layout and fabricates the design.

B. Detailed Flow and Internal Architecture of TOTe

The complete design flow for obfuscating a design, generating an hASIC along with logical and physical synthesis, is illustrated in Fig. 2 and comprises a total of seven steps, which we represent as circled numbers in the text that follows.

In Step ①, the DUO's RTL is synthesized using a commercial FPGA synthesis tool. The DUO requires no special annotations, no synthesis pragmas, nor any other change in its representation. Outputs from Step ① are in the form of a synthesized netlist and a timing report. The netlist comprises all the typical FPGA primitives, i.e., MUXs, LUTs, and FFs. We note that, at this point, the logic of the design is 100% obfuscated since it is entirely captured by LUTs. In very short words, the next steps of TOTe will find LUTs that are good candidates for being replaced by static logic. This is the *core functionality* of TOTe and is illustrated in the bottom left corner of Fig. 2.

Next, in Step ②, preprocessing takes place. This step aims to filter and interpret the timing report and Verilog netlist. The parsing of the timing report is a relatively trivial task. The

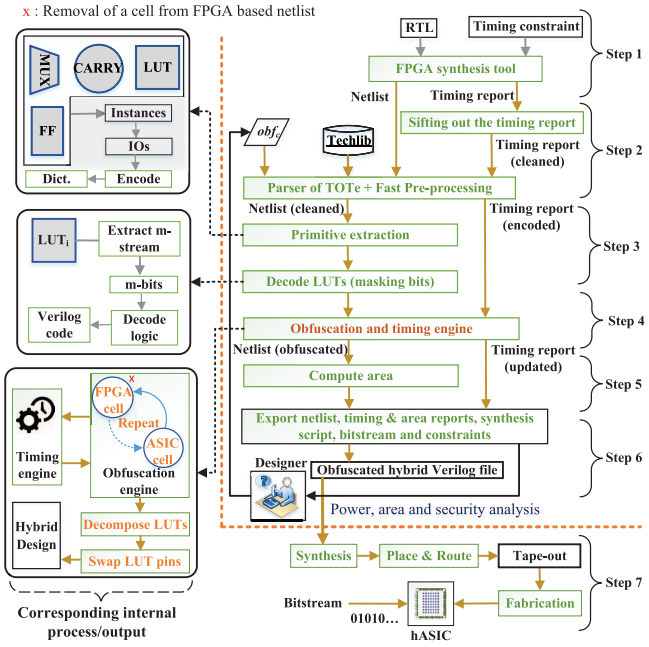

Fig. 2. Overview of TOTe's obfuscation flow and its inner steps.

Algorithm 1: TOTe's Obfuscation Procedure

Input: L (*list of LUTs*), P (*list of paths*), obf_c (*obfuscation criterion*)
Output: $hASIC \leftarrow f(input)$

1 $L_{ST} \leftarrow \phi$, $L_{RE} \leftarrow L$
2 **while** $SIZE\_OF\ (L_{ST}) \leq obf_c$ **do**
3 $path \leftarrow$ FIND_CRITICAL(P)
4 $lut \leftarrow$ FIND_SLOWEST($path$)
5 **if** $lut \in L_{RE}$ **then**
6 INSERT(lut, L_{ST})
7 REMOVE(lut, L_{RE})
8 UPDATE_TIMING(lut, P)
9 **else**
10 REMOVE($path$, P)
11 **for** *each* $lut \in L_{ST}$ **do**
12 DECODE(lut)
13 **for** *each* $lut \in L_{RE}$ **do**
14 GEN_CASE_0_1(lut)
15 DECOMPOSE_OPT(lut)
16 SWAP_PINS(lut)
17 $hASIC \leftarrow L_{ST} \cup L_{RE}$

timing report contains information that should be discarded (empty lines, headers, etc.) for which a bash script has been written. After filtering the timing report, every analyzed path may now contain four FPGA primitives: FF, CARRY, LUT$_i$, and MUX. TOTe encodes (hashes) the instance names to avoid lengthy string representations. The preprocessing step ends when TOTe produces a list of timed paths, where each path contains a list of hashed instances and associated delay values. Note that an instance can appear in many paths and also can appear in many paths under different timing arcs. Finally, the list of timed paths is sorted in ascending order. As a result, the path that has the highest delay (critical path) is referred to as CP and the sum of all CPs is referred to as sumCP.[1]

Step ③ is the process of primitive extraction and LUT decoding. TOTe builds a graph representation of the netlist to keep track of port connections such that the circuit structure can be preserved once optimizations are applied. Under the graph representation, primitive types are annotated for every instance; For LUTs, in particular, the tool also annotates their masking patterns (i.e., configuration bits of an individual LUT). In practice, TOTe is able to interpret the LUT encoding scheme utilized in the netlist coming from FPGA synthesis. For the case of a LUT$_6$, the 64-bit masking pattern extracted from the netlist is converted to a truth table with six inputs and one output. The masking pattern determines which combinations of inputs generate outputs as 1s and 0s. The process is identical for smaller LUTs, which then have smaller truth tables. By using truth tables populated by the masking patterns, TOTe builds combinational logic that is equivalent to the LUT's logic. The truth tables are exported as synthesizable Verilog code. Other primitives, such as FF and MUX, require no decoding and are directly translated to their ASIC equivalents.

[1]CP and SumCP are analogous to WNS and TNS in traditional static timing analysis (STA), except all paths, here, are assumed to pass timing checks. For simplicity, no negative values are, therefore, considered in this analysis.

TOTe comes with obfuscation and timing engines that drive the security versus performance objectives of the tool. These engines are utilized in Step ④ and are responsible for different important tasks, including timing analysis, critical path identification, and replacement of reconfigurable cells for static cells. Algorithm 1 describes the different operations inside the obfuscation main loop of the tool, where L is a list of LUTs, P is a list of timed paths, and obf_c is the obfuscation criterion. The internal variables L_{ST} and L_{RE} are lists of LUTs in static and reconfigurable form, respectively. Initially, all LUTs are considered (line 1). Then, the obfuscation engine executes until the desired number of LUTs is made static (line 2), where the SIZE_OF function returns the size of a list. Inside the obfuscation inner loop, the critical path is identified (line 3) using the FIND_CRITICAL function, then the slowest LUT on that path is identified using the FIND_SLOWEST function (line 4). If the identified LUT is a reconfigurable LUT (line 5), the lists of LUTs are updated (lines 6 and 7) and the timing engine recalculates the affected paths (line 8). If the identified LUT is not reconfigurable (line 9), the path is removed (line 10) and the loop continues (line 2). The INSERT and REMOVE functions update the lists as hinted by their names.

A few additional steps take place after the obfuscation criterion has been met (lines 11–17). These steps are already related to the implementation of hASIC, but we list then, here, for completeness. The DECODE function operates on every LUT that was assigned to be static. From Step ③, TOTe already possesses their description in Verilog as truth tables. TOTe then executes the ASIC synthesis of the truth tables to obtain netlists composed of standard cells. The function GEN_CASE_0_1 generates the "force logic" to be used for timing and power analysis during physical synthesis; otherwise, each LUT would be timed for its worst timing arc instead of the actual implemented timing arc when the LUT is programmed. DECOMPOSE_OPT decomposes the larger LUTs into smaller LUTs. Due to the complexity of this operation, we

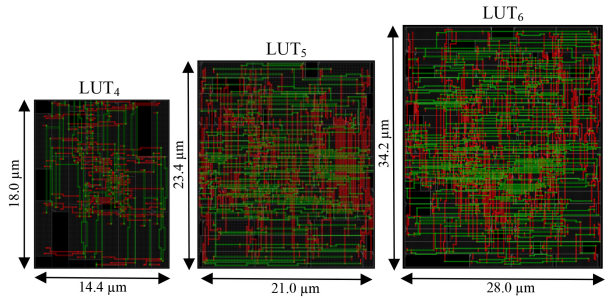

Fig. 3. Layout of macros for LUT_4, LUT_5, and LUT_6. Implementation was executed in Cadence Innovus.

dedicate an entire section to it (see Section III). SWAP_PINS performs a final timing optimization that is an attempt to swap the LUT pins in order to improve the delay, also discussed later in Section III-C. Finally, the algorithm merges L_{ST} and L_{RE} to generate hASIC and returns.

In Step ⑤, area estimation is performed. The estimated area of the hASIC design is calculated as $A = A_{re} + A_{st}$, where A_{re} is the area of the reconfigurable part and A_{st} is the area of the static part. For calculating A_{re}, we sum the area of the LUTs that remain reconfigurable. For calculating A_{st}, we sum the area of the standard cells of the static LUTs. Later, a very precise area estimation is done in an industry-strength physical synthesis tool, where congestion is properly accounted for.

Step ⑥ mostly relates to the generation of files that describe the hASIC intent. This step exports an obfuscated hybrid Verilog file, a timing report, and an area report. A designer can repeat this procedure until he/she achieves his obfuscation (security) and performance targets. Finally, in Step ⑦, the obfuscated netlist is implemented in a commercial physical synthesis tool where traditional P&R, CTS, DRC, etc., steps are executed and the resulting tapeout database is sent to the foundry for fabrication. Once the fabricated parts are delivered, they have to be programmed for the hASIC design to be functional. The programming step requires a bitstream, the same as in an FPGA design.

III. LUT-Specific Approaches to Improve QoR

In this section, we discuss LUT-related optimizations and decisions taken in order to make an hASIC design display the high-performance characteristics of an ASIC and the obfuscation capability of an FPGA fabric.

A. Custom Standard Cell-Based LUTs

We have designed our own custom LUTs (LUT_1, LUT_2, ..., LUT_6) out of *regular standard cells* and by following VPR's template [26]. The layouts for LUT_4, LUT_5, and LUT_6 macros are shown in Fig. 3. Table I shows the average delays and other characteristics of the implemented LUTs.

Commercial FPGAs typically implement only one LUT size, but hASIC provides the flexibility to implement the design with different LUT sizes. This is because the generated hASIC solution is design-specific, meaning that the reconfigurability notion of an FPGA is no longer sought. Moreover, our LUT macros are highly compact, which helps placement to achieve high-density designs. Every single LUT contains a number of flip-flops for storing the configuration bits that serve as a lock for the obfuscated design. Each LUT also makes use of three extra pins (serial_in, serial_out, and enable) to configure these registers. The LUTs are serially connected to one another, forming a daisy chain that is analogous to a scan chain. The choice of a flip-flop-based implementation makes our framework technology-agnostic while making the floorplan and placement almost effortless. Moreover, the LUTs themselves are also treated as regular standard cells during physical synthesis. This allows us to take full advantage of placement algorithms from commercial EDA tools, thus, eliminating the need for any extra custom scripts for placing the LUT macros.

TABLE I
Block Implementation Results for LUT_i

Macro	Area (μm^2)	Density (%)	FFs	Comb. cells	Avg. delay (ns)
LUT_1	36.00	76.00	2	1	0.049
LUT_2	64.80	76.26	4	1	0.052
LUT_3	117.00	89.23	8	8	0.119
LUT_4	259.20	85.23	16	15	0.192
LUT_5	491.40	91.50	32	33	0.257
LUT_6	957.60	91.09	64	36	0.295

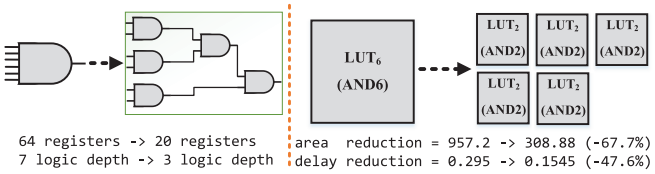

Fig. 4. Logic conversion and decomposition of LUT_6.

B. LUT Decomposition

The area and delay of a LUT are directly related to its number of inputs: the area is mainly bounded by the number of sequential elements used to store the LUT's truth table, whereas the delay is proportional to the LUT's internal MUX tree. However, not all 6-input functions require a LUT_6 to be implemented. For instance, an AND6 can be decomposed in 5 AND2s, as presented in Fig. 4. Referring to Table I, it is clear that the area almost doubles for each input added. Also, the delay vastly increases, where a LUT_6 has almost 6× more average delay than LUT_2. The previous example shows that a LUT_6 decomposition can reduce the area to less than one-third. Moreover, the delay is reduced to approximately half. This example presents a promising approach to improving both area and timing of the circuit.

To decompose our LUTs, we will use functional composition (FC) [27], an approach that can perform bottom-up association of Boolean functions and control the costs in the composition process. Such capability contrasts with traditional top-down functional decomposition, which does not provide a final cost until the decomposition is complete.

1) Functional Composition for LUTs: A summary of the FC paradigm and its application for LUTs will be presented. Readers can obtain more details from [27], [28]. FC is a bottom-up paradigm that has five principles: 1) it uses bonded pairs (BPs) that have a functional part (canonical implementations of a Boolean function, i.e., BDDs or truth tables) and an implementation part (the structure that is being optimized, i.e.,

Algorithm 2: FC-OPT-LUT Algorithm

```
   Input: N (number of LUT inputs), C (cost function)
   Output: ALL_IMP
1  ALL_IMP ← φ, B ← φ, i ← 1
2  MAX_COST ←LUT_COST(C, N)
3  B.add (CREATE_INITIAL_FUNCTIONS (N))
4  AT ←NEXT_BUCKET(B, i, C)
5  while COST(AT, C) < MAX_COST do
6  |  B.add (ASSOCIATE (AT, C, ALL_IMP))
7  |  i ← i + 1
8  |  AT ←NEXT_BUCKET(B, i, C)
9  if SIZE_OF(IMP_LUT) < 2^(2^N) then
10 |  CREATE_NAIVE_IMPS(ALL_IMP, N)
11 return ALL_IMP
```

Algorithm 3: FC-HEUR-LUT Algorithm

```
   Input: F (target function), C (cost function)
   Output: IMP
1  ALL_IMP ← φ, B ← φ
2  B.add (CREATE_INITIAL_FUNCTIONS (F, ALL_IMP))
3  IMP ← ALL_IMP(F)
4  if IMP ≠ φ then
5  |  return IMP
6  ALL_COF ← EXTRACT_ALL_COFACTORS (F)
7  foreach cofactor COF ∈ ALL_COF do
8  |  6 COF_IMP ← FC_HEUR_LUT(COF, C)
9  |  ALL_IMP ← [COF, COF_IMP]
10 ALL_IMP ← [F, GET_NAIVE_SOLUTION(F)]
11 COMBINE_COFACTORS(ALL_COF, ALL_IMP)
12 ASSOCIATE_FUNCTIONS (ALL_COF, ALL_IMP, C)
13 IMP ← ALL_IMP(F)
14 return IMP
```

a fanout-free LUT circuit in this article); 2) every BP association performs independently the functional/implementation operations, allowing for more complex implementations with simple functional operations; 3) using partial ordering and dynamic programming, all BPs with the same cost are stored together in a set (bucket), allowing the use of intermediate solutions as subproblems and to perform associations in a cost-increasing fashion; 4) to start any FC algorithm, initial BPs are required, i.e., constants and single input variables; and 5) it allows the heuristic selection of a subset of allowed functions to reduce the composition search space.

2) Exhaustive LUT FC Method: FC can be applied exhaustively, providing fanout-free implementations that have optimal cost. Algorithm 2 generates all minimal LUT fanout-free implementations for functions up to four variables. The algorithm to generate functions with N inputs consists of generating all implementations using LUT($N-1$) with a smaller cost than the LUT(N). Functions with up to two inputs are by definition not decomposable. For functions containing N inputs, a set of functions that will serve as Boolean operators are required. The Boolean operators are the NPN class functions from 2 up to $N-1$ inputs and their negated and permuted variants.

We take area values from the LUT macros' bounding box for the cost functions, and delay values are the average delay of all timing arcs. If a more sophisticated timing analysis is done, some permutations can generate different delays, yielding different results. This difference can be mitigated at the end of the flow by the SWAP_PINS capability later presented in Section III-C.

The FC-OPT-LUT algorithm takes the number of LUT inputs N and the cost function C, which accounts for area and delay. The result is ALL_IMP, a map of functions and LUT implementations. Lines 1–3 initialize the variable B, which is the bucket list containing all functions already implemented, and MAX_COST, which will provide the single LUT-N cost. Also, B is initialized with constants and single variables through the method CREATE_INITIAL_FUNCTIONS. In line 4, the association of tuples and arrival time (AT) is computed, which consists of tuples containing the indices of the buckets used to combine the functions, from index 0 to $i-1$, where the cost needs to be higher than $B[i-1]$. Still, at the same time, it is the smaller one of all possibilities. As an example, if the candidate ATs have cost 14, 10, 10, and 12 and $B[i-1]$ cost is 9, the candidate ATs with cost 10 will be selected.

The while loop in lines 5–8 checks, at each iteration, if the cost of AT is not higher than MAX_COST. In such cases, it is better to use the naive solution. If the cost is smaller, the method ASSOCIATE will process the list of AT, combining them 2 by 2 or 3 by 3 (depending on the tuple size), using the cost function for breaking ties. The result is added to the bucket list. A new list of ATs is computed, which will continue or break the loop. Finally, in lines 9 and 10, if there are remaining functions, they are considered not decomposable (i.e., the cost to decompose them is higher than the naive solution). The method CREATE_NAIVE_IMPS will look for Boolean functions that do not have an implementation on ALL_IMP and add a naive one, guaranteeing that all Boolean functions of up to N inputs are present on the ALL_IMP map, returned in line 11.

3) Heuristic LUT FC Method: The method described in the previous section can only generate optimal LUTs of up to four inputs. A heuristic is required to deal with more complex LUTs. A LUT decomposition can be thought of as a factorization problem, where there is a direct conversion from a factored form to a LUT tree (i.e., a fanout-free) structure. With some modifications, we can apply the Boolean factoring method presented in [28] to perform decompositions. This approach is called FC-HEUR-LUT. These modifications are necessary to derive LUT decompositions that can have a better cost than the naive solution.

FC-HEUR-LUT is presented in Algorithm 3. The algorithm takes the target function F and the cost function C. The result is the LUT implementation IMP, which is the LUT circuit containing the decomposed naive solution. Lines 1–3 initialize the variables ALL_IMP, which represents a map storing all known implementations for the functions already decomposed, and B, which contains the buckets. The method CREATE_INITIAL_FUNCTIONS remains the same as in FC_OPT_LUT. Lines 4 and 5 check if it is a trivial case and returns if so. In line 6, the method EXTRACT_ALL_COFACTORS is executed. This method computes all the cofactors and cubecofactors (excluding constants) from F and stores them in the ALL_COF set.

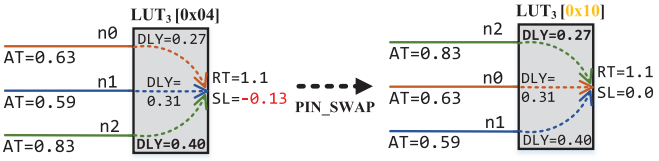

Fig. 5. Example of a beneficial pin swap.

Lines 7–9 are a recursive call to the algorithm, providing a LUT implementation to all cofactors and cubecofactors. With the cofactors and cubecofactors derived, the combination of cofactors takes place in line 11, using the same strategies presented to expand the "allowed functions" set, as explained in [28]. This expansion guarantees at least 2 factored subfunctions that, when associated in the next step, will provide at least one functionally equivalent solution. In line 12, the ASSOCIATE_FUNCTION will perform AND/OR/XOR operations using the rules mentioned and NAND/NOR/XNOR associations using the "not comparable" functions. These associations are discarded if they are not the target function F. If the association is functionally equivalent to F, the cost function C will compare the current solution (which initially is the naive one) with the current one, replacing it in the case of a better cost. Finally, lines 13 and 14 will collect the resulting implementation IMP and return.

Some techniques applied to greatly speed-up FC-HEUR-LUT include using FC-OPT-LUT results to aid FC-HEUR-LUT. The ALL_IMP map is used at the beginning of the algorithm to quickly return the optimal implementation if the function F has a support of 4 or fewer inputs while also improving the decomposition quality of results (QoR). Another technique applied is the limit on the number of associations of not comparable Boolean functions because those can be a considerable number (i.e., more than 100 thousand). This limit avoids substantial runtime trying to decompose more complex functions, which generally have worse costs when decomposed.

C. Pin Swap Approach

The method SWAP_PINS takes advantage of the fact that a LUT function can have an arbitrary input pin swap if the truth table is permuted accordingly. So, our method takes a LUT function and timing information as input and provides the permuted truth table and the new order of input pins/nets. The example presented in Fig. 5 shows an effective pin swap that improved the slack of the design. The pin swap algorithm takes the LUT function, the AT of each input net (termed [$n0$, $n1$, $n2$]), the cell arc delay (DLY) associated with each input, and the required time (RT) at the output. In the example, RT = 1.1, and the critical arc is $n2$, with a total delay of 1.23. The algorithm initially tries all the input permutations, trying to minimize WNS. If two or more arcs have negative slack, it also tries to reduce TNS. Once all permutations are tried, and a new order improves WNS and/or TNS, the truth table is permuted accordingly to keep the same functionality. The algorithm returns the truth table $0x10$ and the new net order [$n2$, $n0$, $n1$].

IV. Experimental Results Using TOTe

Without loss of generality, for all experimental results, we have executed FPGA synthesis in Vivado and the target is a Kintex-7 XC7K325T-2FFG900C device which contains 6-input LUTs [29]. Following, Cadence Genus is used for the logic synthesis with three flavors of a commercial 65-nm standard cell library (LVT/SVT/HVT). However, we emphasize that TOTe is *agnostic with respect to PDKs, libraries, and tools*.

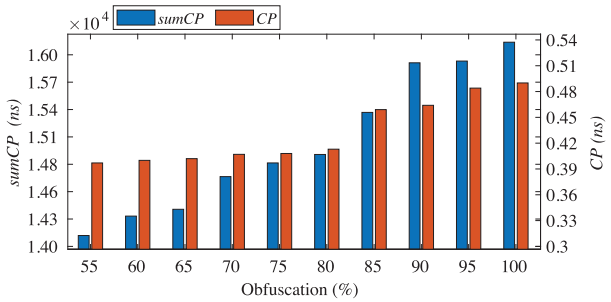

Fig. 6. TOTe's obfuscation versus performance for SBM.

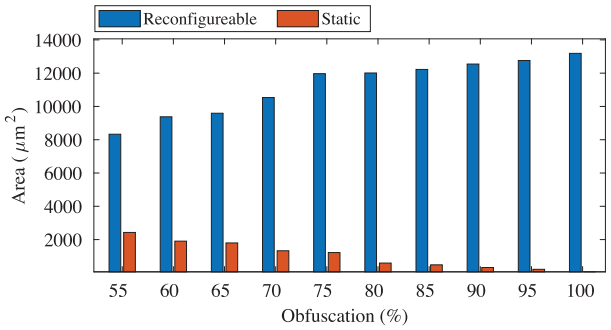

Fig. 7. TOTe's obfuscation versus area for SBM.

For our first experiment, we considered a small but pragmatic design that covers all possible FPGA primitives. We selected a schoolbook multiplier (SBM) design as DUO [30]. We obfuscated an 8-bit SBM by varying obf_c from 55% to 100% and evaluated the obfuscation versus performance and obfuscation versus area trends. We have synthesized the SBM design targeting a challenging frequency of 540 MHz. As calculated by TOTe's timing engine, the CP and sumCP values become 0.490 and 16088.69 ns, respectively. These values correspond to a design obfuscated at 100%, i.e., all LUTs are reconfigurable. At this stage, these values represent a simplistic timing analysis, realistic timing values will be obtained when the final timing analysis is performed using a commercial physical synthesis tool. However, as we move along with the obfuscation process, CP and sumCP remain consistent in relative terms, which is sufficient to generally determine critical paths to target.

After performing the obfuscation for different levels, the timing characteristics for the 8-bit SBM are illustrated in Fig. 6. The analysis of CP and sumCP shows that performance is decreasing as we increase the level of obfuscation. Conversely, decreasing the level of obfuscation increases the performance of the design. The trend depicted in Fig. 6 is that CP improves inversely with the obfuscation, but it is saturated when the obfuscation is below 80%. This fact is not true for sumCP, the decrease in obfuscation causes continuous improvement as expected. Similarly, the performance versus area profile of the 8-bit SBM is illustrated in Fig. 7.

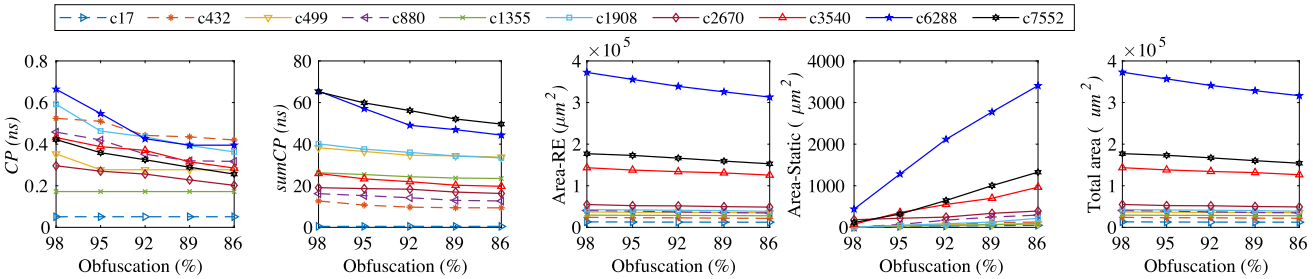

Fig. 8. Obfuscation results for ISCAS'85 benchmarks using TOTe.

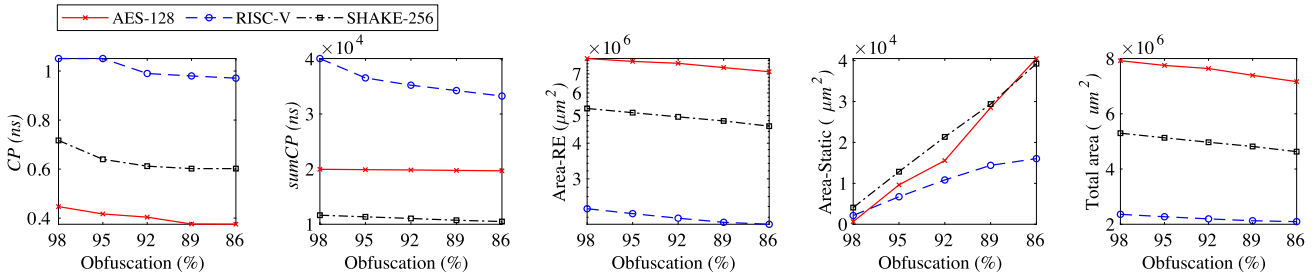

Fig. 9. Obfuscation results for AES-128, RISC-V, and SHAKE-256 using TOTe.

Next, we investigate whether the same saturation would appear for other designs. We first selected the ISCAS'85 benchmarks and the results are depicted in Fig. 8. These relatively outdated combinational benchmarks were selected for the reason that they have a single stage of logic, so the CP and sumCP correlation is easy to follow (i.e., the critical path does not change from different reg2reg paths). Even in these simplistic designs, saturation occurs remarkably fast.

Naturally, we have also obfuscated more representative designs. In [25], detailed results are provided for SBM, SHA-256, and FPU [31] designs. For the sake of brevity, we do not repeat those results here. Additional results are also provided in graphical form in Fig. 9 for AES, RISC-V, and SHAKE-256 designs so the trends are easy to visualize.

In summary, the results presented in this section confirm that TOTe is a generic tool for obfuscation and it can obfuscate a design regardless of its complexity. It also becomes clear that the reliance on a LUT-based representation of the circuit, akin to an FPGA, has different implications for delay and area. For area, the trend is clear: the lesser is the obfuscation target, the more compact the circuit becomes. However, for delay, it appears that hASIC brings performance penalties that cannot be overcome by simply reducing the targeted obfuscation level. Therefore, other strategies are needed for achieving better performance. In the next section, we will present a more detailed analysis of physical synthesis. We will also apply the optimization methods described in Section III for improving performance.

V. Physical Synthesis for hASIC

This section contains the physical implementation results for an obfuscated SHA-256 [33] design. Cadence Innovus is utilized for physical synthesis, together with a commercial 65-nm PDK. We have selected SHA-256 as it is popular and widely used in cryptography. The variants of the design with different obfuscation levels are implemented with the aid of the LUTs defined in Section III-A. The results obtained after implementation are focused on performance versus area tradeoffs for the 80%–100% obfuscation range, thus, avoiding the saturation trend highlighted in Section IV.

Initially, we synthesized and implemented SHA-256 on FPGA, the target device being a Kintex-7. The FPGA implementation achieves a frequency of only 77 MHz (for reference, the Kintex-7 family is produced on a 28-nm CMOS technology). To start the analysis, we select 100% obfuscation as a baseline design because it is fully reconfigurable and somewhat analogous to an FPGA design. The implementation results for 80%, 85%, 90%, and 100% obfuscation are given in Table II. The timing results are obtained after physical synthesis and are for the worst process corner (SS), VDD = 0.9*$\text{VDD}_{\text{nominal}}$, and a temperature of 125 °C.

From the results, it is clear that the level of obfuscation does not affect the utilization density of the design (i.e., the ratio of placement sites that are occupied versus empty). For all designs, we achieved around 80% utilization density, which is very high considering a large number of macros. In other words, our macros do not compromise global routing resources. It is noteworthy that the performance of TOTe-generated designs is increasing as we decrease the level of obfuscation; our baseline hASIC design is running at 223 MHz (as shown in Freq. column of Table II) and it increases as obf_c decreases. This behavior matches the goal we set from the start: to establish a tradeoff between performance (ASIC) and security (FPGA).

The area of the design is proportional to the obfuscation level, which means that increasing the security of the design comes with an area penalty. As we only exploit LUT primitives for promoting obfuscation, the number of LUTs increases with the obfuscation level. In the same manner, leakage and dynamic power figures are proportional to security as reconfigurable logic is less efficient than static. This is mainly because

TABLE II
RESULTS FOR THE IMPLEMENTATION OF SHA-256 FOR DIFFERENT OBFUSCATION LEVELS

CAD flow	Obf.	Density	Area (μm^2)	Freq. (MHz)	Leakage (mW)	Dynamic Power (mW)	# LUT	# Buffer	# Comb.	# Inv.	# Sequential	Total Wirelength (μm)
FPGA	100%	–	–	77	2.4	191	2238	–	–	–	1830*	–
TOTe	100%	81%	1751500	223	14.85	505.05	2238	5846	93470	6175	105128	9247654
TOTe	90%	77%	1638500	234	12.23	438.47	2015	4626	84107	5017	94876	7505590
TOTe	85%	80%	1507000	241	12.10	430.98	1904	4846	80304	5585	90420	7207023
TOTe	80%	80%	1409700	248	11.05	386.89	1792	4406	75083	4564	83790	6724434
ASIC	NONE	92%	34208	248	0.18	9.37	–	167	3244	190	1806	158003
ASIC	NONE	91%	40804	550	0.299	23.86	–	675	7981	1456	1806	181441

* Vivado performs flip-flop cloning for solving high fanout buffering. Thus, there is an increase in the number of registers w.r.t. ASIC.

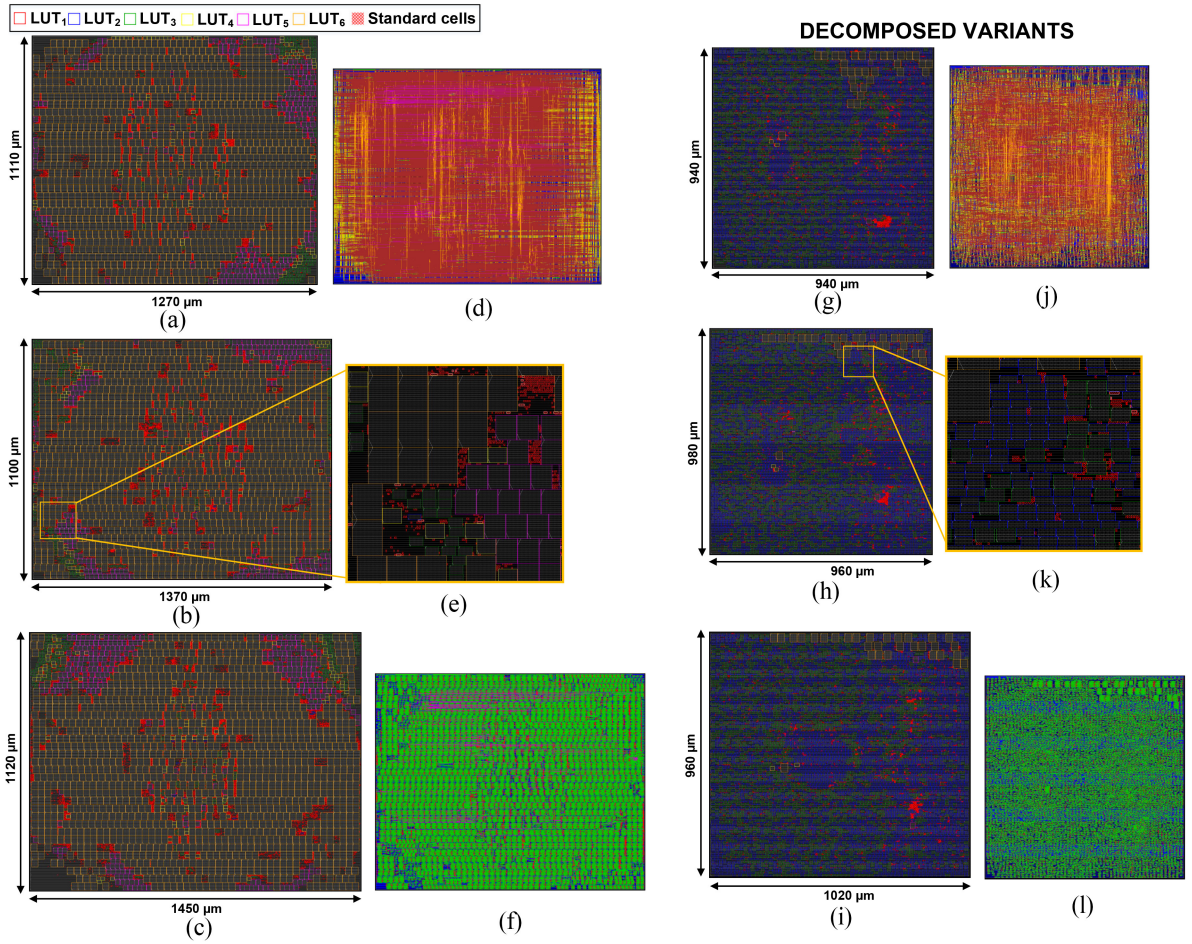

Fig. 10. Implementation results for SHA-256 with different obfuscation levels. (a) SHA-256 with 80% obfuscation. (b) SHA-256 with 85% obfuscation. (c) SHA-256 with 90% obfuscation. (d) SHA-256 with 85% obfuscation (routed layout). (e) SHA-256 with 85% obfuscation (magnified view). (f) SHA-256 with 85% obfuscation (routed and assembled layout). (g) SHA-256 with 80% obfuscation. (h) SHA-256 with 85% obfuscation. (i) SHA-256 with 90% obfuscation. (j) SHA-256 with 85% obfuscation (routed layout). (k) SHA-256 with 85% obfuscation (magnified view). (l) SHA-256 with 85% obfuscation (routed and assembled layout).

of the use of flip-flops to store the LUT truth tables. The last five columns of Table II show the resource requirements for hASIC (number of buffers, combinational cells, inverters, sequential cells, and the total wirelength).

In Fig. 10, we show many different views of the SHA-256 layouts under different obfuscation targets. The considered metal stack has seven metals assigned to signal routing. Panels (a)–(c) of Fig. 10 illustrate the layouts for 80%, 85%, and 90% obfuscation levels. The dimensions of the layouts are indicated on the bottom and left sides of each panel. All six variants of LUTs are highlighted with different colors and the static part of hASIC is highlighted in red—notice that, as expected, the design remains primarily a sea of LUTs. The majority of those LUTs are LUT_6, thus, the layouts appear to be dominated by yellow boxes.

Panel (d) of the same figure demonstrates the final post-route layout of hASIC. Notice how the post route design contains mostly vertical orange lines which correspond to M6. Panel

TABLE III
RESULTS FOR THE IMPLEMENTATION OF SHA-256 FOR DIFFERENT OBFUSCATION LEVELS WITH DECOMPOSED LUTS

CAD flow	Obf.	Density	Area (μm^2)	Freq. (MHz)	Leakage (mW)	Dynamic Power (mW)	#LUT	#Buf.	#Comb.	# Inv.	#Seq.	Total Wirelength (μm)
TOTe	100%	61%	1155000	307	8.00	301.49	10182	3583	29352	15261	53868	3391742
TOTe	90%	65%	979200	312	7.55	273.54	9127	1797	27115	13538	49016	3242970
TOTe	85%	67%	940800	322	7.03	256.36	8676	1882	26011	13136	46796	2982627
TOTe	80%	64%	883600	357	6.44	278.37	8124	1726	24614	12340	43830	2889253
TOTe (Swap)	80%	64%	883600	368	5.93	283.35	8124	1726	24614	12340	43830	2889760

(e) of Fig. 10 shows the magnified view of the placement in an hASIC design. The mixed structure of LUT macros and standard cells clearly depicts the placement pattern and the spacing between the macros is usually filled with standard cells. Notice how the LUT macros align with the standard cell rows, allowing for the entire design to have a uniform power rail and power stripe configuration. In panel (f) of the same figure, we illustrate the same design but filter out some routing layers (only M2, M3, and M4 are shown). As depicted in Fig. 3, the implemented LUTs utilize the aforementioned metal layers, therefore, the assembled view of a panel (f) represents how visually regular the hASIC structure is.

In the results shown in panels (g)–(l) of Fig. 10, we have utilized the same design and conditions but applied LUT decomposition for improving performance. Notice that the layouts are drawn to scale to highlight the area reduction brought by decomposition. In particular, when comparing panels (f) and (l), we note that regularity is still present even after decomposition, as expected.

The detailed results for these designs are listed in Table III. With decomposition, the baseline frequency increased significantly, from 223 to 307 MHz. As in the nondecomposed version, the performance increases inversely with the obfuscation level. On top of that, the area was reduced by more than half, along with the power consumption. However, due to a large number of small LUTs (mostly LUT_2s), placing and routing become slightly more challenging. For this reason, the maximum utilization density across the optimized designs is approximately 65%. Nevertheless, decomposition is very beneficial: the gain in PPA when compared with the nonoptimized versions is significant. We argue that since decomposition has a negligible impact on the runtime of the physical synthesis flow, it should always be applied. For instance, the runtime to apply the decompositions in the SHA-256 circuit with 100% obfuscation containing 2238 LUTs was 11 min in an Intel Core i7-6700K. The decomposition achieved improvements of 50% in the LUT area and 32% in the total LUT delay.

While the LUT decomposition brings significant performance improvement, we seek to achieve performance levels that are as close to the ASIC implementation as possible. For that reason, after the decomposition, we also applied the pin swapping technique. As we noted earlier, this is possible because the same logic function can be generated with different input orders and different masking bits (truth table). Therefore, we can search for LUTs that appear on the critical path(s) and swap their pins to reduce the total delay.

For illustrating the capability of pin swapping, we first create an artificial scenario where we increase the target frequency of the design until several paths violate setup timing. The frequency increase determines the number of violating paths that indirectly determines the number of LUTs that will be considered for pin swap purposes.[2] All LUTs from violating paths are chosen as candidates and saved in a list. Then, iteratively, starting with the worst violating path, the pins of the LUTs are swapped until the critical path is improved (i.e., WNS). The number of swaps performed versus the TNS and WNS is illustrated in Fig. 11. From this figure, the first few swaps improved the WNS while the TNS remain the same. The continuing swapping starts to improve the TNS without any change in the WNS. If the TNS is improving, that means there is a chance to reach a better WNS. Thus, we performed the swapping until the next jump in the WNS. After attempting 200 swaps, WNS improved by approximately 80 ps and TNS by 2 ns, thus, making the design 11 MHz faster.

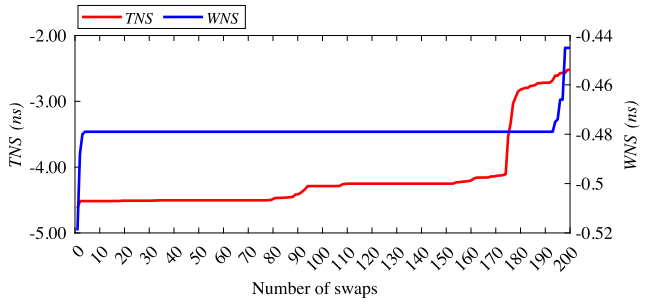

Fig. 11. Change in the TNS/WNS with respect to the swap of LUT pins.

VI. SECURITY ANALYSIS

A. Threat Model

In our considered threat model, the primary adversary is the *untrusted foundry*. We make no distinction whether the adversary is institutional or a rogue employee. Assuming the security of an hASIC design is a function of its static logic (fully exposed) and reconfigurable logic (protected by a bitstream that serves as a key), we make the following assumptions.

1) The main adversary goal is to reverse engineer the design in order to pirate its IPs, overproduce the IC, or even insert sophisticated hardware trojans. For this goal, the adversary *must* recreate the bitstream.
2) The adversary goal might also be to identify the circuit intent, even in the presence of obfuscation. For this goal, the adversary *does not need* to recreate the bitstream.

[2]Here, we establish a runtime versus QoR tradeoff. The more aggressive the frequency target is, the more LUTs are considered for pin swap.

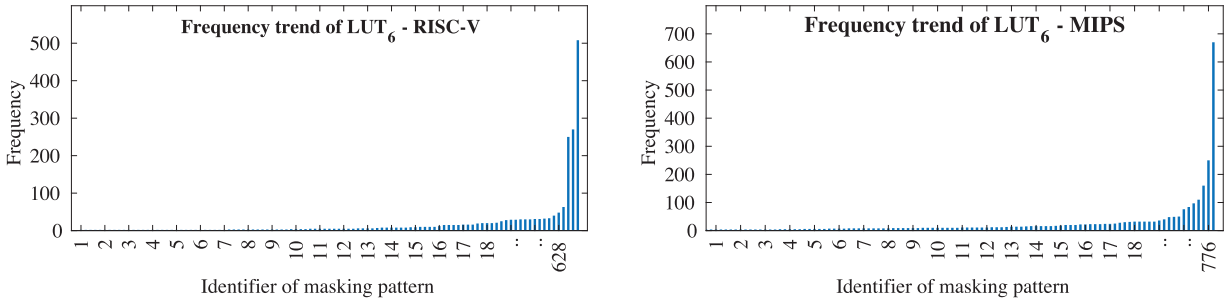

Fig. 12. Frequency of masking patterns for RISC-V and MIPS.

3) The adversary has access to the GDSII file of the hASIC design sent for fabrication. The adversary is skilled in IC design and has the knowledge and tools required for understanding this layout representation.
4) The adversary can recognize the standard cells, therefore, the gate-level netlist of the obfuscated circuit can be easily recovered [35].
5) The adversary can identify reconfiguration pins [36], [37], thus, being able to effortlessly enumerate all LUTs and their programming order.
6) The adversary can group the standard cells present in the static logic and convert them back into a LUT representation.[3]

We have proposed two different attacks to evaluate the security hardness of hASIC: one based on the *structure* of design and another based on the *composition* of known different circuits. We assert that an adversary can learn and extract information by exploiting the static portion of the design, including the frequency of specific masking patterns. This capability would allow an adversary to shrink the search space for the key that unlocks the design.

Similarly, the notion of masking pattern frequency can be utilized as a template to compare different designs. In other words, the composition of the LUTs in a design would allow for a template-based attack. Moreover, we have also evaluated the security hardness of hASIC for conventional oracle-guided and oracle-less attacks borrowed from logic-locking attacks. All the experiments reported in this section were run on a server equipped with 32 processors (Intel Xeon Platinum 8356H CPU @ 3.90 GHz) with 1.48 TB of RAM.

B. Structural Analysis Attack

Goal: By statistical analysis means, decrease the key search space before attempting to recover the bitstream.

We recall again that TOTe's obfuscation engine utilizes six variants of LUTs. However, the majority of the LUTs are LUT_6 due to the packing algorithm executed during FPGA implementation. The decomposition only applies to the reconfigurable part of the design. The initial knowledge that the adversary acquires is from the static part, which remains unchanged whether decomposition is used or not used. The static part is composed mostly of LUT_6, so the adversary ought to keep his/her analysis geared at LUT_6 too. Therefore, we consider this scenario and present our analysis of it.

For a LUT_6, the possible number of keys is 2^{64}. But this number is only realistic if the FPGA synthesis tool is genuinely able to exercise the entire key search space. This does not appear to be true: We have synthesized a considerable number of representative designs (>30) and extracted all unique LUT_6 masking patterns from the corresponding netlists. We term these values m_i. We considered designs of varied size, complexity, and functionality until the combined number of unique masking patterns forms a set of $M = \sum m_i = 3376$ elements that appear to settle. This result alone, albeit being empirical, reduces the global search space from 2^{64} to $3376 = 2^{11.72}$.

With this information at hand, we hypothesize that an attacker can exploit the frequency at which LUTs appear in a netlist in order to mount attacks, thus, the name structural analysis attack. In other words, the adversary is interested in finding the values of m_i for a given circuit C_i. However, the adversary only has partial knowledge of the design. The question then becomes whether the adversary can estimate m_i by performing statistical analysis on a portion of C_i. To this end, we targeted two processor designs in our statistical analysis: MIPS and RISC-V. For each circuit, we utilize tuples of ⟨pattern, frequency⟩ for tracking how often masking patterns repeat. The pattern is a 64-bit hexadecimal number. The tuples are referenced by integer identifiers and ordered by frequency as shown on the bar charts in Fig. 12. Notice that the MIPS netlist has 776 unique LUTs and that there are very few outliers that occur more than 50 times. Similarly, for RISC-V, there are 628 unique LUTs and only 3 occur more than 100 times.

We investigate this by analyzing the behavior of the frequency of masking patterns as depicted in Fig. 13. For this, we utilized netlists generated by TOTe at 98%, 95%, 92%, 89%, and 86% obfuscation levels. Therefore, for this experiment, we assume the attacker only has visibility over 2%, 5%, 8%, 11%, and 14% of the LUTs, respectively. The adversary then attempts to predict the distribution of actual masking patterns in the design from his/her observation of the small percentage of LUTs that are exposed in the static portion of hASIC. In Fig. 13, the adversary's guessing attempt is performed with the aid of polynomial trendlines. For MIPS and RISC-V, it appears that the adversary can estimate to some degree what masking patterns are the outliers. The actual number of unique patterns, m_i, is not trivial to determine from extrapolation since many patterns appear a single time or very few times (see Fig. 12). We clarify that several circuits studied in this article (e.g., PID, IIR, GPU, SHA-256, etc.) have a

[3]This is a very generous concession since the static logic is repeatedly optimized during synthesis. Nevertheless, we assume the adversary can achieve a perfect reconstruction of LUTs.

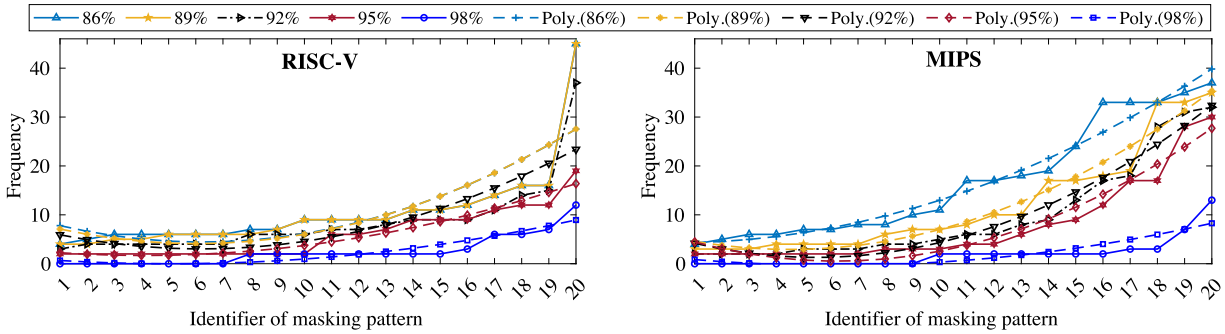

Fig. 13. Structural analysis of RISC-V and MIPS.

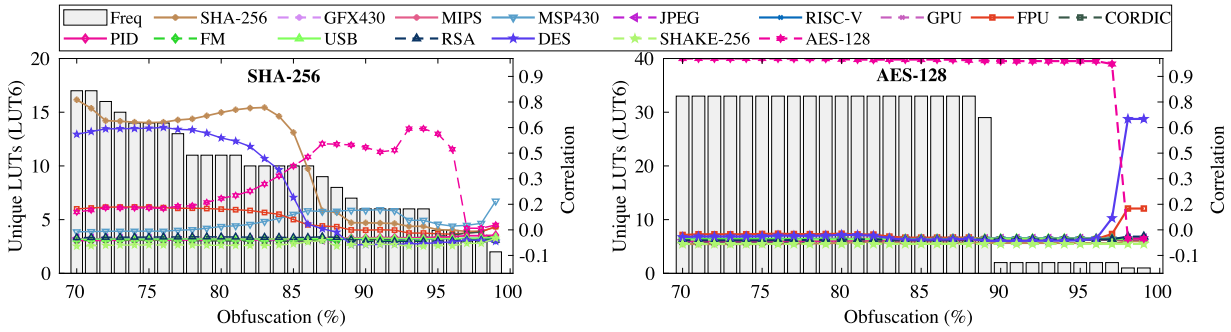

Fig. 14. Correlation of SHA-256 and AES-128 versus numerous other designs.

similar profile, where only a handful of high-frequency LUTs appear. As stated earlier, the attack exploits only the static part. But, when the decomposition is applied the adversary needs an in-depth knowledge of the decomposition algorithm to estimate the frequency of outliers for the reconfigurable part. Therefore, it remains to be studied if the knowledge gathered from this attack can be useful for some form of hill climbing attack (or even a biased version of SAT).

C. Composition Analysis Attack

Goal: identify the circuit by correlation to known circuits.

This attack also exploits the frequency of the LUTs, but, here, we correlate several designs against each other based on their composition, thus, the name. We suppose that the attack is already successful if the adversary is able to identify the circuit (i.e., breaking the key is not necessary).

In the experiment depicted in Fig. 14, we carried out correlation analysis for two different crypto cores: 1) SHA-256 and 2) AES-128. The goal of this analysis is to examine the leaked information from the static part against a database[4] of circuits that are known to the attacker. We perform obfuscation of SHA-256 and AES-128 in the 70%–100% range and then correlate their static portions with the designs in the database. The x-axis of Fig. 14 is the obfuscation level, and the y-axis is the number of unique LUTs (left) and Pearson correlation (right).

The correlation results reveal very interesting trends. For SHA-256, three regions of interest can be defined based on the degree of obfuscation: 97%–100% (no correlation), 86%–96% (strong correlation to another circuit), and 70%–85% (correlation to itself). A similar analysis has been performed for AES-128 as shown in the right side of Fig. 14. The correlation between obfuscated AES-128 versus AES-128 is almost one for obfuscation $< 97\%$. Conversely, the correlation for obfuscated AES-128 versus other designs is almost zero obfuscation in the same range ($< 97\%$).

Assuming that the adversary's objective is exclusively to recognize the circuit's intent ("what is this circuit?"), hASIC could prove as vulnerable as an ASIC design. This is the case for the AES-128 circuit, while the SHA-256 case reveals a contrasting trend: there are obfuscation ranges that can be targeted on purpose to confuse an adversary. For SHA-256, this range appears to be 86%–96%.

Finally, for an adversary that is interested in obtaining the bitstream, we hypothesize that the correlation analysis herein depicted might be useful to shrink the key search space further. In practice, if the attacker could know for a fact that the obfuscated circuits are indeed AES, or SHA-256, or C_i, his key guessing would be based on the m_i of the circuit with the highest correlation. It is noteworthy to mention that this attack leverages the design attributes and the frequency of LUTs derived from the static part of the design. Hence, the attack's success or failure depends upon three key points: First, the adversary's ability to reconstruct the LUTs from the static part; Second, the availability of enough datapoints in the database of known circuits; Third, the design has static parts, i.e., the obfuscation level is below 100%. Referring again to the example from the previous section, the search space would shrink further; from 3376 to 776 for MIPS and from 3376 to 628 for RISC-V. As stated in Section VI-B, this attack also exploits only the static part. Therefore, identifying the design has no relationship with decomposition, i.e., a decomposed design is not easier (or harder) to correlate. From this point

[4]We assume that the adversary can obtain samples of open-source cores from repositories and execute FPGA synthesis on them to create a database.

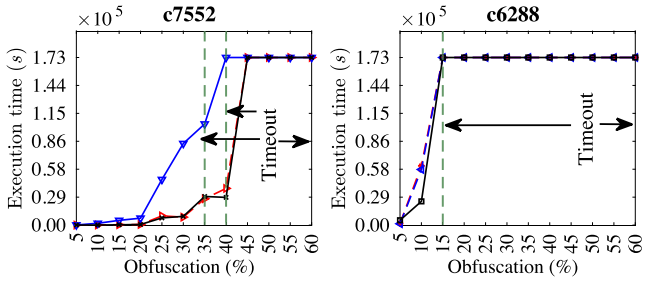

Fig. 15. Execution time of SAT attacks for two different designs.

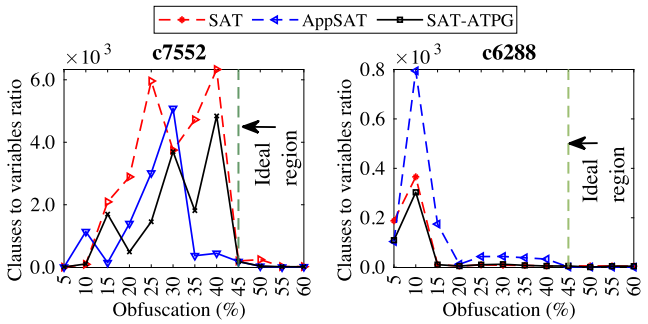

Fig. 16. Variables to clauses ratio of SAT attacks for two different designs.

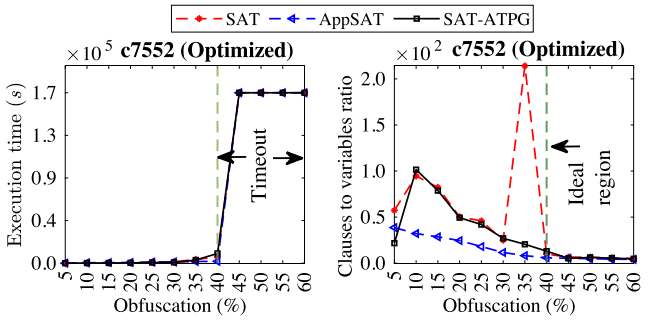

Fig. 17. Optimized results for c7552 with regards to the execution time and the ratio of variables to clauses in SAT attacks.

onward, in order to obtain the actual key, an adversary would still have to resort to other attacks (not specific to hASIC). We discuss such attacks in the text that follows.

D. Oracle-Guided Attacks

Goal: To retrieve a key or a key guess.

As compared to conventional logic locking [38], the LUTs introduced in hASIC are the elements that serve as key gates. A LUT_6, in theory, introduces 64 bits of key, akin to 64 XOR/XNOR gates in conventional logic locking. The very first circuit we introduced in Section IV, the SBM, has 25 LUTs (out of which 11 are LUT_6) when its obfuscation rate is 86%. In turn, the key search space would be $2^{11\times64}$ for LUT_6 alone. Such a large search space would discourage an adversary from performing SAT attacks on hASIC.

However, enumerating the key search space is a very simplistic/naive approach. One has to perform actual attacks in order to evaluate the security of the designs, especially, well-known satisfiability-based attacks. We have, therefore, considered three different SAT attacks to evaluate the security hardness of hASIC: 1) Conventional SAT [36]; 2) AppSAT [39]; and 3) ATPG-based SAT [40].

We have selected large combinational circuits (c7552 and c6288) from the ISCAS'85 suite to evaluate the security hardness of hASIC against SAT-based attacks. Importantly, we present the results for two different variants of hASIC, optimized and nonoptimized, for the selected designs. Concerning the nonoptimized variant, Fig. 15 illustrates the execution time for different SAT attacks and different obfuscation rates, where the x-axis is the obfuscation level and the y-axis is the execution time. As expected, the execution time increases as we increase the obfuscation level. The region to the left of the green line shows successful SAT attacks. However, the region on the right corresponds to unsuccessful attacks, where timeout (48 h) was achieved before the solver returned an answer. In principle, this is an encouraging result since even a very small and combinational-only circuit like c6288 leads to timeouts at relatively low obfuscation rates $\sim$15%.

More intriguingly, another combinational-only circuit c7552 led to timeout after reaching 40% obfuscation rate. Thus, the obfuscation rate where the SAT solver returns a timeout varies in different designs. Additional statistics about the behavior of the SAT solver for the obfuscated circuits are given in Fig. 16. The SAT solver determines when a Boolean formula is satisfiable or not. One approach to measuring the chance of convergence of the attack is to measure the ratio of clauses to variables of the SAT solver. With the help of this ratio, an obfuscated design can be labeled SAT-hard if the ratio is around 4.2 [41]. Fig. 16 shows the evolution of the number of clauses to variables with respect to the obfuscation level. The x-axis is the obfuscation level (%) and the y-axis shows the ratio of clauses to variables. We label the right region of Fig. 16 as "Ideal region" because the clauses to variables ratio are near the ideal value of 4.2.

TOTe automatically optimizes designs by decomposing LUTs into smaller LUTs. While power, area, and performance are improved, the decomposition reduces the size of the key for unlocking the design. We must, therefore, verify that this reduction does not make the hASIC designs vulnerable to existing attacks. For the optimized version of c7552 in Fig. 17, we can see that there is a reduction in the time that it takes for the successful attacks to complete. However, none of the attacks is successful beyond 40% obfuscation. Further details for c7552 are given in Table IV where we also list the key sizes for different designs and two obfuscation rates, namely, 55% and 60%. Note that, counter-intuitively, the decomposed designs have better variables-to-clause ratios. Our interpretation is that decomposition keeps keys that are less correlated to one another, thus, each individual key bit is more effective. Readers are directed to [41] for details on the SAT attack and to [40] for a discussion on key interference.

E. Oracle-Less Attacks

Goal: To retrieve a key or a key guess.

Oracle-less attacks do not require an oracle (i.e., a functional IC). Instead, they operate directly on the netlist of the obfuscated circuit. One of such attack is the synthesis-based constant propagation attack on logic locking (SCOPE) [42].

TABLE IV
ANALYSIS OF VARIABLES-TO-CLAUSES RATIO FOR $obf_c = 55\%$ AND 60%

Attack	Obf. (%)	*Key length* (bits)	*Variables*	Clauses	Iterations	Ratio
SAT [36]	55	7494	32318070	1597559	820	20.8
	60	8582	33315150	1898618	540	17.5
	55∗	4014	3434578	598599	139	5.7
	60∗	4406	2829008	564533	106	5.0
AppSAT [39]	55	7994	15843074	1127281	8	14.0
	60	8582	15412584	1181330	7	13.0
	55∗	4014	1320652	302801	2	4.36
	60∗	4406	1419176	346847	2	4.09
ATPG-SAT [40]	55	7494	27243806	1800999	630	15.1
	60	8582	31166810	2247522	636	13.8
	55∗	4014	3642116	664817	148	5.4
	60∗	4406	2274084	480548	85	4.7

∗ Results for the optimized designs

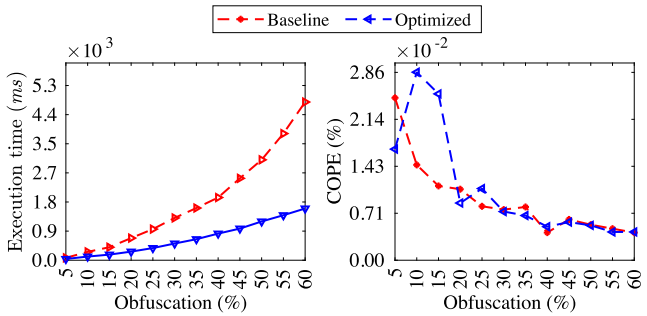

Fig. 18. Comparison between the baseline and optimized design for the oracle-less SCOPE attack.

This attack does not require any knowledge about the locking technique or the obfuscated design. SCOPE performs a synthesis-based analysis on a single key-input port and extracts important design features that may assist to derive the correct key bits. Fig. 18 illustrates the comparison of execution time, COPE metric for the baseline, and optimized design of the c7552 design. It is clear from the left panel that the execution time is exponentially increasing with the obfuscation level. Similar trends are seen for the baseline and optimized design, both are exponential but present different rates. The right panel of Fig. 18 shows the COPE metric, which decreases with the obfuscation level. The details for the calculation of COPE metric are available in [42]. For simplicity, we clarify that the COPE metric is a rough estimate of the level of vulnerability (%) to the SCOPE attack.

When SCOPE concludes, a key guess is produced. For each bit of the key, SCOPE assigns either a "1," a "0," or an "X" (undetermined). When matching the guess from SCOPE with our known key, the result is that about 50% of the key bits are correctly guessed. This percentage is not related to the obfuscation level, the result is always the same for baseline and optimized designs. In other words, SCOPE cannot perform better than a random guess for hASIC.

VII. DISCUSSION

A recent trend in obfuscation research is the use of embedded FPGA (eFPGA) [17], [18]. While there are advantages to this practice, it has been used selectively to only protect key portions of a design and, therefore, keep the performance penalty as low as possible. The challenge is in determining which portions/modules of the circuit merit protection and which ones do not. Our hASIC approach bypasses this question almost completely by only revealing (portions of) critical paths when they are selected to become static logic, which we consider an advantage. Mohan et al. [44] presented a top-down methodology to implement ASICs with eFPGAs. Their designs share many of the advantages of our hASIC solution while presenting more regularity than our designs (they make use of logic tiles as in commercial FPGAs). Our tile-free design trades this regularity for performance as evidenced by the layout in Fig. 10 and the corresponding results in Table II.

VIII. CONCLUSION

The main finding of our work is that an hASIC solution contrasts with the current practice of eFPGA obfuscation; our experimental results illustrate that obfuscation rates have to be high to secure the design's intent. To this end, we have presented a custom tool that obfuscates a design and generates an hASIC block. Our LUT decomposition, along with the pin swapping, improves the performance and reduces the area of hASIC designs. We have also validated our results in a commercial physical synthesis tool with industry-strength timing and power analysis. Our security analysis, anchored by the results from diverse attacks, confirms that obfuscation rates should be high.

REFERENCES

[1] "2022 semiconductor sales to grow 11% after surging 25% in 2021." IC Insights. 2022. [Online]. Available: https://www.icinsights.com/data/articles/documents/1424.pdf
[2] "Apple's M3 chips on track for 2023 as next-gen 3nm process begins." Mac World. 2021. [Online]. Available: https://www.macworld.com/article/557232/m3-chips-apple-devices-tsmc-3nm-process-2023.html
[3] "TSMC to kick off mass production of Intel CPUs in 2H21 as Intel shifts its CPU manufacturing strategies, says TrendForce." TrendForce. 2021. [Online]. Available: https://www.trendforce.com/presscenter/news/20210113-10651.html
[4] "Big trouble at 3nm." Semiconductor Engineering. 2018. [Online]. Available: https://semiengineering.com/big-trouble-at-3nm/
[5] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proc. IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.
[6] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "Threats on logic locking: A decade later," in *Proc. Great Lakes Symp. VLSI*, 2019, pp. 471–476.
[7] Y. Xie and A. Srivastava, "Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction," in *Proc. 54th ACM/EDAC/IEEE Des. Autom. Conf. (DAC)*, 2017, pp. 1–6.
[8] M. Yasin, J. Rajendran, and O. Sinanoglu, *Trustworthy Hardware Design: Combinational Logic Locking Techniques*. Cham, Switzerland: Springer, 2019.
[9] M. Yasin and O. Sinanoglu, "Evolution of logic locking," in *Proc. IFIP/IEEE Int. Conf. Very Large Scale Integr. (VLSI-SoC)*, 2017, pp. 1–6.
[10] J. Sweeney, V. M. Zackriya, S. Pagliarini, and L. Pileggi, "Latch-based logic locking," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST)*, 2020, pp. 132–141.
[11] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 517–532, Apr.–Jun. 2020.
[12] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware IP protection," in *Proc. 51st ACM/EDAC/IEEE Des. Autom. Conf. (DAC)*, 2014, pp. 1–5.
[13] M. Li et al., "Provably secure camouflaging strategy for IC protection," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 8, pp. 1399–1412, Aug. 2019.

[14] T. D. Perez and S. Pagliarini, "A survey on split manufacturing: Attacks, defenses, and challenges," *IEEE Access*, vol. 8, pp. 184013–184035, 2020.
[15] J. Rajendran, O. Sinanoglu, and R. Karri, "Is split manufacturing secure?" in *Proc. Des. Autom. Test Eur. Conf. Exhibit. (DATE)*, 2013, pp. 1259–1264.
[16] B. Liu and B. Wang, "Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks," in *Proc. Des. Autom. Test Eur. Conf. Exhibit. (DATE)*, 2014, pp. 1–6.
[17] B. Hu et al., "Functional obfuscation of hardware accelerators through selective partial design extraction onto an embedded FPGA," in *Proc. Great Lakes Symp. VLSI*, 2019, pp. 171–176.
[18] J. Chen, M. Zaman, Y. Makris, R. D. S. Blanton, S. Mitra, and B. C. Schafer, "DECOY: DEflection-driven HLS-based computation partitioning for obfuscating intellectual propertY," in *Proc. 57th ACM/EDAC/IEEE Des. Autom. Conf.*, 2020, pp. 1–6.
[19] J. Bhandari et al., "Exploring eFPGA-based redaction for IP protection," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, 2021, pp. 1–9.
[20] H. M. Kamali, K. Z. Azar, K. Gaj, H. Homayoun, and A. Sasan, "LUT-lock: A novel LUT-based logic obfuscation for FPGA-bitstream and ASIC-hardware protection," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, 2018, pp. 405–410.
[21] G. Kolhe, P. D. S. Manoj, S. Rafatirad, H. Mahmoodi, A. Sasan, and H. Homayoun, "On custom LUT-based obfuscation," in *Proc. Great Lakes Symp. VLSI*, 2019, pp. 477–482.
[22] G. Kolhe et al., "Security and complexity analysis of LUT-based obfuscation: From blueprint to reality," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Des. (ICCAD)*, 2019, pp. 1–8.
[23] S. D. Chowdhury, G. Zhang, Y. Hu, and P. Nuzzo, "Enhancing SAT-attack resiliency and cost-effectiveness of reconfigurable-logic-based circuit obfuscation," in *Proc. IEEE Int. Symp. Circuits Syst. (ISCAS)*, 2021, pp. 1–5.
[24] G. Kolhe et al., "Breaking the design and security trade-off of look-up-table–based obfuscation," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 27, no. 6, pp. 1–29, 2022.
[25] Z. U. Abideen, T. D. Perez, and S. Pagliarini, "From FPGAs to obfuscated eASICs: Design and security trade-offs," in *Proc. Asian Hardw. Orient. Security Trust Symp. (AsianHOST)*, 2021, pp. 1–4.
[26] K. E. Murray et al., "VTR 8: High-performance CAD and customizable FPGA architecture modelling," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 13, no. 2, pp. 1–55, 2020.
[27] M. G. A. Martins, R. P. Ribas, and A. I. Reis, "Functional composition: A new paradigm for performing logic synthesis," in *Proc. 13th Int. Symp. Qual. Electron. Des. (ISQED)*, 2012, pp. 236–242.
[28] M. G. A. Martins, L. Rosa, A. B. Rasmussen, R. P. Ribas, and A. I. Reis, "Boolean factoring with multi-objective goals," in *Proc. Int. Conf. Comput. Des. (ICCD)*, 2010, pp. 229–234.
[29] "Xilinx Kintex-7 FPGA KC705 evaluation kit." Xilinx, Inc. 2021. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/ek-k7-kc705-g.html
[30] M. Imran, Z. U. Abideen, and S. Pagliarini, "An open-source library of large integer polynomial multipliers," in *Proc. 24th Int. Symp. Des. Diagnos. Electron. Circuits Syst. (DDECS)*, 2021, pp. 145–150.
[31] J. Al-Eryani. "Floating-point unit (FPU) controller." 2017. [Online]. Available: https://opencores.org/projects/fpu100
[32] J. Carlos. "FPGA-based median filter." 2014. [Online]. Available: https://opencores.org/projects/fpu100
[33] S. Joachim. "SHA-256." 2020. [Online]. Available: https://github.com/secworks/sha256
[34] O. Kindgren and M. John. "FPGA-based median filter." 2015. [Online]. Available: https://github.com/openrisc/or1200
[35] R. S. Rajarathnam, Y. Lin, Y. Jin, and D. Z. Pan, "ReGDS: A reverse engineering framework from GDSII to gate-level netlist," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST)*, 2020, pp. 154–163.
[36] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST)*, 2015, pp. 137–143.
[37] M. Yasin, A. Sengupta, M. T. Nabeel, A. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security*, 2017, pp. 1601–1618.
[38] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. Des. Autom. Test Eur.*, 2008, pp. 1069–1074.
[39] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. Hardw. Orient. Security Trust (HOST)*, 2017, pp. 95–100.
[40] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. Des. Autom. Conf.*, 2012, pp. 83–89.
[41] E. Nudelman, K. Leyton-Brown, H. H. Hoos, A. Devkar, and Y. Shoham, "Understanding random SAT: Beyond the clauses-to-variables ratio," in *Principles and Practice of Constraint Programming (CP)*, M. Wallace, Ed. Berlin, Germany: Springer, 2004, pp. 438–452.
[42] A. Alaql, M. M. Rahman, and S. Bhunia, "SCOPE: Synthesis-based constant propagation attack on logic locking," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 29, no. 8, pp. 1529–1542, Aug. 2021.
[43] "Using encryption and authentication to secure an ultrascale/ultrascale+ FPGA bitstream." Xilinx. 2022. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/application_notes/xapp1267-encryp-efuse-program.pdf
[44] P. Mohan, O. Atli, O. Kibar, M. Zackriya, L. Pileggi, and K. Mai, "Top–down physical design of soft embedded FPGA fabrics," in *Proc. ACM/SIGDA Int. Symp. Field-Program. Gate Arrays*, 2021, pp. 1–10.

Zain Ul Abideen (Graduate Student Member, IEEE) received the M.S. degree in computer engineering (Master in Integration, Security and Trust in Embedded Systems) from the Grenoble Institute of Technology, Grenoble, France, in 2019. He is currently pursuing the Doctoral degree with the Tallinn University of Technology, Tallinn, Estonia.

During his master's studies, he was associated with the Cybersecurity Institute, Univ. Grenoble Alpes, Grenoble. He worked on hardware security and side-channel attacks. His research work is mainly focused on hardware security and obfuscation-based ASIC design.

Tiago Diadami Perez (Graduate Student Member, IEEE) received the M.S. degree in electrical engineering from the University of Campinas, São Paulo, Brazil, in 2019. He is currently pursuing the Ph.D. degree with the Tallinn University of Technology, Tallinn, Estonia.

From 2014 to 2019, he was a Digital Designer Engineer with Eldorado Research Institute, São Paulo. His current research interests include the study of hardware security from the point of view of digital circuit design and IC implementation.

Mayler Martins received the M.S. (*summa cum laude*) and Ph.D. degrees in microelectronics from the Universidade Federal do Rio Grande do Sul, Porto Alegre, Brazil, in 2012 and 2015, respectively.

From 2016 to 2018, he was a Research Scientist with the ECE Department, Carnegie Mellon University, Pittsburgh, PA, USA. He worked as a Lead Engineer with Siemens EDA, Fremont, CA, USA, from 2018 to 2022. He is currently a Research and Development Engineer Staff with Synopsys, Sunnyvale, CA, USA. His current research interests include logic synthesis methods focusing on QoR optimization.

Samuel Pagliarini (Member, IEEE) received the Ph.D. degree from Telecom ParisTech, Paris, France, in 2013.

He has held research positions with the University of Bristol, Bristol, U.K., and Carnegie Mellon University, Pittsburgh, PA, USA. He is currently a Professor with the Tallinn University of Technology, Tallinn, Estonia, where he leads the Centre for Hardware Security. His current research interests include many facets of digital circuit design and hardware security.