Design Obfuscation through Selective Post-Fabrication Transistor-Level Programming

Mustafa M. Shihab, Jingxiang Tian, Gaurav Rajavendra Reddy, Bo Hu, William Swartz Jr.,
Benjamin Carrion Schaefer, Carl Sechen and Yiorgos Makris
Dept. of Electrical and Computer Engineering, The University of Texas at Dallas, Richardson, Texas 75080, USA
Email: {mustafa.shihab, jingxiang.tian, gaurav.reddy, bo.hu4, bill-swartz, schaferb, carl.sechen, yiorgos.makris}@utdallas.edu

***Abstract*—Widespread adoption of the fabless business model and utilization of third-party foundries have increased the exposure of sensitive designs to security threats such as intellectual property (IP) theft and integrated circuit (IC) counterfeiting. As a result, concerted interest in various design obfuscation schemes for deterring reverse engineering and/or unauthorized reproduction and usage of ICs has surfaced. To this end, in this paper we present a novel mechanism for structurally obfuscating sensitive parts of a design through post-fabrication TRAnsistor-level Programming (TRAP). We introduce a transistor-level programmable fabric and we discuss its unique advantages towards design obfuscation, as well as a customized CAD framework for seamlessly integrating this fabric in an ASIC design flow. We theoretically analyze the complexity of attacking TRAP-obfuscated designs through both brute-force and intelligent SAT-based attacks and we present a silicon implementation of a platform for experimenting with TRAP. Effectiveness of the proposed method is evaluated through selective obfuscation of various modules of a modern microprocessor design. Results corroborate that, as compared to an FPGA implementation, TRAP-based obfuscation offers superior resistance against both brute-force and oracle-guided SAT attacks, while incurring an order of magnitude less area, power and delay overhead.**

I. Introduction

Prohibitive ownership cost of contemporary semiconductor manufacturing facilities has driven the majority of IC design companies to go fabless and rely, instead, on a more cost-effective, globalized, third-party foundry business model. While such third-party foundry manufacturing reduces production cost for these companies, it also exposes their products to a range of security threats, such as IP theft, unauthorized over-production and counterfeiting [1]. Consequently, significant effort has been invested in developing solutions to mitigate such risks in various stages of the supply-chain flow, including among others Physical Unclonable Functions (PUFs), IC metering, watermarking, layout camouflaging and design obfuscation [2]. The latter, in particular, has garnered significant interest because of its promise to thwart not only IP theft by an untrusted foundry but also IC reverse engineering and unauthorized usage by untrusted parties.

Among the various design obfuscation methods that have been proposed in the literature, the vast majority seeks to embed the functionality that needs to be obfuscated within a broader functionality with additional inputs, which are typically referred to as "key" inputs. Application of the correct values at the key inputs essentially de-embeds the original functionality and de-obfuscates the design, while application of any incorrect value results in incorrect functionality. Numerous methods have been proposed to achieve such embedding for both combinational and sequential logic, using various primitives such as XOR/XNOR gates, multiplexers, look-up tables, finite state machines, etc. [3]–[6], leveraging the power of an exponentially growing search space in order to protect against brute-force attacks. On the other hand, intelligent attacks, such as the clever use of a Boolean Satisfiability (SAT) solver [7], were also developed and compromised most early obfuscation schemes. This, in turn, has resulted in an on-going race between new defenses and new attacks, reinforcing the need for further research in this area [8]–[10]

In this work, we investigate an alternative design obfuscation approach which does not rely on embedding a function within a key-protected super-function, but rather on omitting parts of it from the fabricated silicon and reinstating them through post-manufacturing programming. While the idea in itself is rather straightforward, realizing it in a cost-effective manner is quite challenging. Indeed, using FPGA-style look-up tables (LUTs) to implement the omitted portion of the function provides high security levels but results in prohibitive area, power and performance overhead [5], [11]. Furthermore, it disrupts the design process as there currently exists no CAD tool-flow that supports combined logic synthesis and timing analysis of a mixed ASIC/FPGA IC design.

To address these limitations, we drew inspiration from earlier work in circuit-level reconfiguration [12] and developed TRAP, a transistor-level programmable fabric which can be seamlessly integrated in a design flow to replace portions of an ASIC. By using a dense "sea-of-transistors" architecture, supporting implementation of custom cell libraries and facilitating fabric time-sharing, TRAP enables a "design obfuscation by omission" solution which not only incurs an order of magnitude less area, power and performance overhead than FPGA-style LUTs, but also raises the level of difficulty for compromising it through sophisticated SAT-based attacks. Specifically, the key contributions of this work are:

- **TRAP-Based Design Obfuscation:** We propose TRAP as a design obfuscation method. To our knowledge, this is the first exploration of the multi-faceted advantages that a transistor-level programmable fabric offers for the purpose of design obfuscation.
- **Security Analysis:** We perform an analytic resilience study of TRAP-obfuscated designs against brute-force and SAT-based attacks. In particular, we detail how the proposed obfuscation mechanism can overwhelm the formation of a potential SAT attack and render it infeasible.
- **Experimentation Platform:** We present a silicon implementation of the TRAP fabric in a 65nm technology and the corresponding test-bed for experimenting with TRAP-based obfuscation. We also discuss the development of a custom CAD flow for seamless integration between TRAP and the custom ASIC parts of a design.
- **Comparative Evaluation:** We evaluate effectiveness of TRAP-based obfuscation by applying it on various modules of the FabScalar [13] microprocessor core. As a point of reference, we also compare its area, power and performance overhead against an equivalent implementation using an FPGA in the same technology node.

978-3-9819263-2-3/DATE19/©2019 EDAA

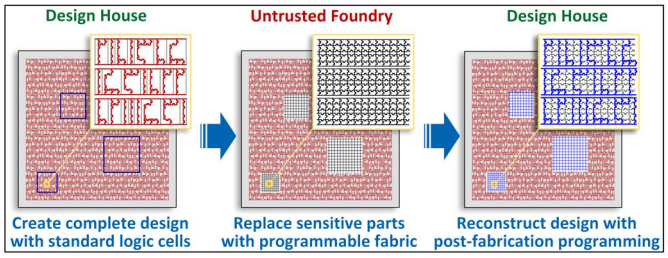


Fig. 1: TRAP-based design obfuscation.

II. Threat Model

Design obfuscation methods typically aim at protecting against three threat models of increasing severity:

Scenario 1: The adversary has access to a fabricated IC implementing an obfuscated design. The adversary knows which pins correspond to key inputs but is not privy to the correct key value, which he/she seeks to somehow guess in order to enable correct IC functionality.

Scenario 2: The adversary has access to the netlist of an obfuscated design, either through an accomplice who leaks the layout files (i.e., GDSII) from an untrusted foundry where the design is fabricated or through reverse engineering of a fabricated chip. Using only the netlist, the adversary seeks to guess the correct key value in order to de-obfuscate the design.

Scenario 3: In addition to the netlist, the adversary also has *black-box* access to an unlocked chip (but not to the correct key value), which he/she can interrogate by providing values at its key and regular inputs and observing values at its outputs, in order to deduce the correct key value.

While scenarios 1 and 2 limit the adversary's options to *brute-force* attacks, scenario 3 sets the stage for *intelligent* solutions, such as the SAT-based attack, which has proved to be particularly effective against most obfuscation schemes.

III. TRAP-based Obfuscation

A. Overview

The proposed method for obfuscating a design through transistor-level programming is conceptually depicted in Figure 1. First, the design-house creates a complete design using standard logic cells. Then, the sensitive parts of the design are removed and replaced by TRAP fabric, before the layout is sent to the foundry. When the fabricated ICs are received, the missing parts are programmed onto the TRAP fabric to reinstate the complete design functionality. As we explain in detail in Section IV, the transistor-level granularity of the TRAP fabric enables cost-effective post-manufacturing programming of both logic gates and storage elements, as well as time-sharing of the fabric through virtualization. Thereby, it sets up insurmountable obstacles to an adversary who seeks to de-obfuscate the design in any of the aforementioned scenarios.

B. Sources of Uncertainty

As shown in Figure 2, TRAP-based obfuscation introduces the following sources of uncertainty that an attacker with access to the obfuscated layout must overcome:

Cell Boundary Obfuscation: At the layout-level, the TRAP fabric appears as a regular sea-of-transistors and contains no markers that can be used to identify the left and right boundaries of logic cells. In other words, it is impossible to know which rows/columns of this sea-of-transistors will be used for instantiating each cell of the omitted logic.

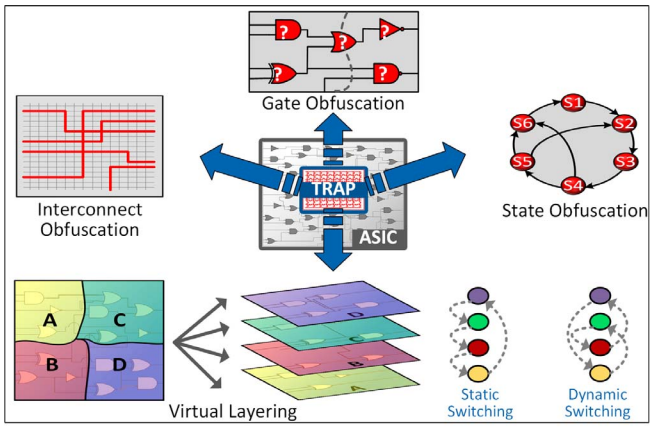


Fig. 2: Sources of uncertainty in TRAP-based obfuscation.

Cell Type Obfuscation: Besides the obfuscated boundary of each cell, its type (i.e., the logic function that it implements) is also unknown and cannot be easily inferred from the size of the array that it occupies. In fact, designers can implement their own standard cell library with proprietary cells and map them on the TRAP fabric. Therefore, deducing functionality and parametrics of each cell is particularly challenging.

Interconnect Obfuscation: The interconnect network for wiring the cells on the fabric is also programmable. In fact, these interconnects can be used both for stitching transistors together into cells and for connecting the cells, thereby further increasing uncertainty. Evidently, the options increase exponentially with the size of the fabric, making it impractical to infer. Furthermore, the routing at the boundary between the ASIC portion of the TRAP-obfuscated design and the fabric itself is also unknown, further exacerbating the problem.

State Obfuscation: In addition to combinational logic, the TRAP fabric can be used to instantiate state-holding elements, such as latches and flip-flops, to support sequential logic. Uncertainty of the state holding variables makes reverse engineering the underlying finite state machine (FSM), which is already a difficult problem, significantly more challenging.

Virtual Layering: As a final obstacle, the TRAP architecture supports decomposition of the obfuscated design into multiple virtual layers which can time-share the physical fabric, either in a fixed or in a dynamically defined order. Each layer can comprise both combinational and sequential logic. TRAP has dedicated flip-flops for retaining layer outputs and transferring them to the next layer. Without knowing the number, functional boundaries, and execution sequence of the virtual layers, it is practically impossible to de-obfuscate a design.

The numerous sources of uncertainty described above make it evident that brute-force attacks, such as in Scenarios 1 and 2, have no hope of success even for small circuits. Interestingly, TRAP-based obfuscation also introduces additional complexity in Scenario 3, wherein intelligent attacks are employed. For example, the SAT-based attack relies on knowing the obfuscated netlist; in the case of TRAP-based obfuscation, however, the netlist changes dynamically during the search, therefore exponentially increasing the size of the SAT formulation, as we explain in detail in the next section.

IV. TRAP Fabric Design and Silicon Prototype

A. TRAP Fabric

The TRAP fabric design follows the general principles of the Field Programmable Transistor Array (FPTA) described

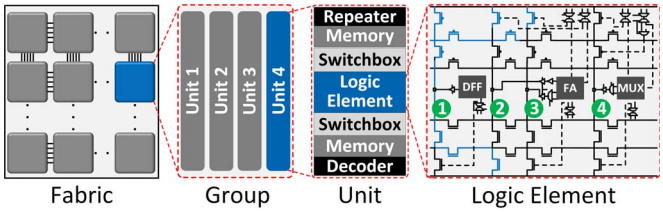


Fig. 3: Hierarchical architecture of TRAP.

by the authors of [12]. TRAP, however, has been designed and implemented in silicon in a newer technology (65nm vs. 130nm) and is denser, as it is specifically tailored to replace portions of an ASIC design for obfuscation purposes.

1) Architecture: The TRAP fabric consists of hierarchically arranged CMOS transistors, as shown in Figure 3.

Logic element: At the lowest level, the logic element (LE) is the core component that is used to implement combinational or sequential logic. It consists of four *columns* of eight transistors (4 vertical + 4 horizontal) which can be stitched together into gates or state-holding components. For better performance and area efficiency of the fabric, an element also contains a built-in flip-flop (DFF), a full adder (FA) and a multiplexer (MUX).

Unit: Each LE is combined with an address decoder, memory blocks, switchboxes and a repeater into a unit. The memory blocks store the programming bits, while the switchboxes and the repeater are used to deliver them to the element.

Group: At the top level, the fabric is organized as an array of functional blocks called groups, each containing four units, which communicate between them and with the surrounding circuitry through programmable interconnect.

2) Fabric Programming: Logic gates and state-holding components are programmed on the TRAP fabric by combining transistors. For smooth integration in an ASIC design-flow, TRAP uses a cell-library with fixed height but variable width. Consequently, the logic gates require varying number of columns, can start from any column and span across logic elements treating the fabric as a sea-of-transistors. Figure 3 depicts the implementation of a NAND3 gate in a logic element. An appropriate bitstream is used to switch on the required transistors (highlighted in blue), and switch off the rest. In addition, the bitstream is required to activate the necessary multiplexers and switches in the corresponding switchboxes.

3) Single vs. Multi-layer TRAP: The TRAP fabric supports time-sharing between multiple programs (layers) which can interact and can be switched instantaneously to collectively form the desired functionality. Such virtualization enables the use of a smaller TRAP fabric at the cost of additional resources such as multiplexers, retention flip-flops, latches, and global signals for controlling the layer order. For example, a four-layer TRAP fabric reduces the number of columns by a factor of 4, as compared to a single-layer TRAP, yet the area per column goes up by 3.5X due to the required resources, resulting in an effective area reduction of 20%. Also, programming the four-layer TRAP requires 31 bits per column, while the single-layer TRAP requires only 17, essentially doubling the bitstream size. However, as mentioned in Section III-B, multi-layer TRAP introduces an additional source of uncertainty (i.e., layer switching order) which is desirable for design obfuscation.

B. CAD Support for TRAP-based Design Obfuscation

To support the proposed method, we developed a synthesis, placement, routing, and verification CAD tool-flow which

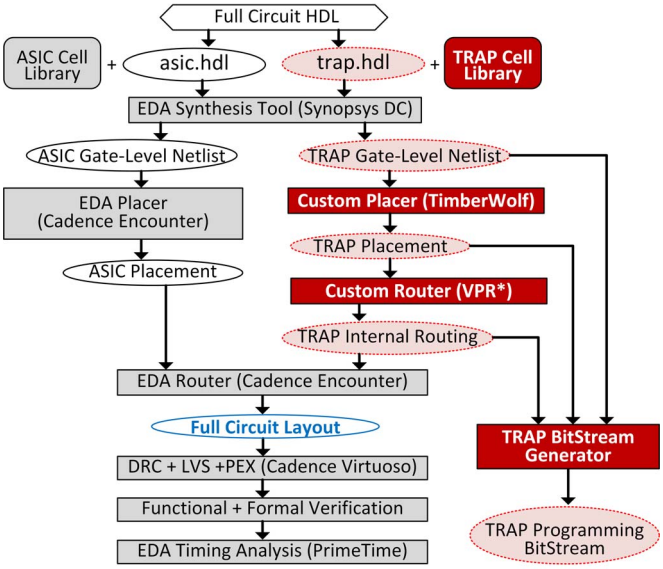


Fig. 4: CAD Tool-flow for TRAP-based Design Obfuscation.

can seamlessly integrate the ASIC and TRAP portions of an obfuscated design. This CAD tool-flow is shown in Figure 4, where the *light red* ovals are steps unique to our approach and the *dark red* squares are tools developed or customized by us.

To begin with, we generated a 65nm TRAP cell-library which contains a total of 137 cells, including custom cells for the built-in DFFs, full-adders, and multiplexers. The library was characterized and compiled with Synopsys SiliconSmart and Library Compiler, respectively. We also created a functionally equivalent cell-library for the ASIC portion of a design.

Our design flow begins with the RTL description of the design, which comprises both the TRAP (e.g., trap.v) and the ASIC (e.g., asic.v) parts. Next, we synthesize the RTL codes using an RTL compiler (e.g., Synopsys Design Compiler). After synthesis, we do placement for the ASIC portion using an EDA placement tool (e.g., Cadence Encounter), while for the TRAP portion we leverage the TimberWolf placement and partitioning tool [14]. We then perform an initial routing for the post-placement TRAP portion to ensure architectural compatibility. For this, we utilize a customized version of the open source VPR tool [15]. Next, the combined design is routed using a commercial EDA router (e.g., Encounter). Finally, the design is run through design rule checker (DRC), layout versus schematic (LVS), and EDA tools for functional/formal verification. In parallel, a custom Python framework processes the netlist, placement, and routing files to generate the programming bitstream for the TRAP portion. Lastly, we note that our tool-flow enables timing analysis of the combined ASIC/TRAP design using standard EDA tools.

C. TRAP-Fabric Silicon Prototype

To demonstrate feasibility and verify functionality of the TRAP fabric, on which the proposed design obfuscation method is crucially dependent, we designed and fabricated in a GlobalFoundries 65nm process (via MOSIS) a 3mmX3mm chip implementing a TRAP array that supports up to 4 layers. We also developed a test-bed to experiment with the chip, which consists of an Opal Kelly FPGA board and a custom-designed interposer printed circuit board (PCB). The interposer PCB houses the TRAP chip and facilitates communication

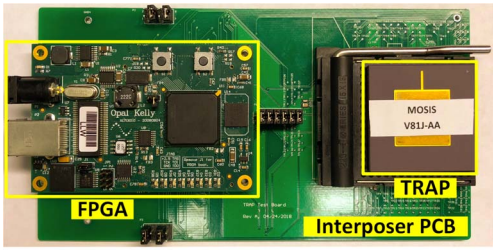


Fig. 5: Hardware testbed for our prototype TRAP chip.

with the FPGA board. The FPGA serves a dual-role: first, it acts as a controller for programming the TRAP chip and capturing its output for further analysis; additionally, in the context of design obfuscation, it is used for prototyping the ASIC portion of the obfuscated design so that the combined ASIC+TRAP functionality can be instantiated in hardware. To date, we have successfully programmed, verified, and characterized the TRAP chip with a wide range of designs.

V. Security Analysis

A. Resilience Against Brute-force Attacks

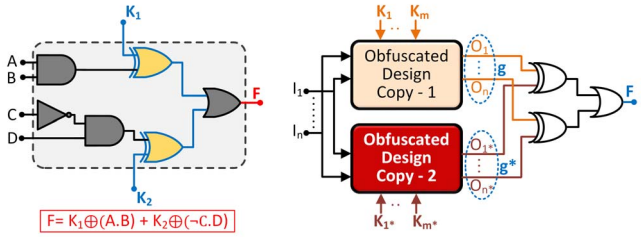
A brute-force attack attempts to de-obfuscate a design by exhaustively trying all possible key combinations. The effectiveness of such attacks is, typically, determined by the number of combinations that need to be evaluated in the worst-case scenario. For example, if a design is obfuscated with a *p-bit* key and evaluation of a key combination takes $\Omega(1)$, then an attacker has to evaluate 2^p combinations, requiring $\Omega(2^p)$ time to de-obfuscate the design. For TRAP-based obfuscation, the programming bits act as the key that reveals the structure of the target design. The size of the bitstream, which reflects the combinations that an attacker has to try, is determined by the number of columns and layers in the fabric:

$$\textit{number of trials in a brute-force attack} = 2^{m \cdot b}, \qquad (1)$$

where m is the total number of columns in the TRAP fabric and b is the number of programming bits per column (i.e., b = 17 and 31, for single and four-layer TRAP, respectively). Considering that a 128-bit key is sufficient for thwarting brute-force attacks [3], it becomes evident that even for a small TRAP fabric such attacks are impractical.

B. Resilience against SAT attack

1) Conventional SAT Attack Methodology: The modus operandi for a SAT attack is to find the correct key by intelligently discarding incorrect ones in an iterative manner. SAT attack relies on availability of the obfuscated netlist from which the corresponding Boolean function can be derived. For example, Figure 6a depicts a design locked with two XOR gates and its corresponding Boolean function. Next, a miter circuit with two copies of the obfuscated design is built, as shown in Figure 6b. The outputs of these two copies (g and g*) are XORed and ORed to form a differential function (F). The conjunctive normal form (CNF) of F is then fed to a SAT-solver for finding a distinguishing input pattern (DIP) for which two different keys generate different outputs. These outputs are compared to the correct one, which is obtained through an unlocked design with black-box access. Key(s) corresponding to incorrect output(s) are then discarded and F is updated with constraint(s) that prune the search space to exclude the offending DIP/output combination(s). This process repeats until no further DIPs can be found, at which point the solver returns the correct key [16].



(a) Logic obfuscation example. (b) Miter circuit for a SAT solver.

Fig. 6: Setup for conventional SAT attacks.

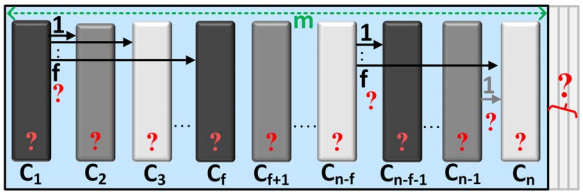


Fig. 7: Gate, interconnect and layer obfuscation in TRAP can deter potential SAT attacks with overwhelming setup cost.

2) Staging a SAT attack on TRAP: The conventional SAT attack method cannot be immediately applied to TRAP. Indeed, since the obfuscated design is omitted from the layout, extracting a netlist from the sea-of-transistors view of the unprogrammed fabric and deriving a Boolean function is not readily possible. Improvising a SAT attack on a TRAP-obfuscated design requires expansion of the CNF formula to consider all Boolean functions that the target fabric can implement. In the following analysis, we conservatively limit our options to only combinational designs.

Impact of cell obfuscation: Figure 7 shows how TRAP hides both the type and the location of the logic gates, forcing the attacker to consider all possible combinations. To err on the side of caution, we assume that an adversary (i) can ascertain the number of columns in the target fabric, and (ii) has access to the TRAP cell-library and the size of each logic gate. The attacker can then prune the search-space and only consider gate combinations that fit on the fabric. In our analysis, we leverage the concept of *dynamic programming (memoization)* to calculate the number of gate combinations (k) for a given number of columns (m). Specifically, our program starts from one edge of the fabric and considers placing each gate option in the first position; then, for each of those placements, it considers all gate options in the second position, and continues this process until it reaches the other edge of the fabric. The lower (k_{min}) and higher (k_{max}) bounds of such gate combinations is defined by:

$$k_{min} = y_L^{m/x_L} < k < k_{max} = y_S^{m/x_S}, \qquad (2)$$

where x_S and x_L are the widths of the smallest and largest gates in the TRAP library, respectively, and y_S and y_L are the counts for the smallest and largest gates, respectively.

Impact of interconnect obfuscation: The number of interconnect combinations depends on the total number of gates in the TRAP fabric (n) and the maximum *fanout* (f) for each gate. From Figure 7, we notice that, going from left to right, gates C_1 through C_{n-f} can connect up to f neighboring gates. However, for gates between C_{n-f-1} and C_n, this limit gradually reduces from $f-1$ to 0. This same pattern of connections exists in the opposite direction (i.e., from C_n to C_1) as well. Based on these observations, the possible

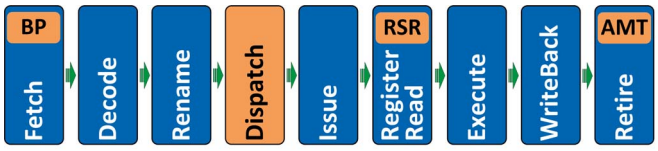


Fig. 8: FabScalar's pipeline (obfuscated modules in orange).

interconnect options are:

$$\textit{possible combinations of interconnects} = 2^{f(2n-f-1)} \quad (3)$$

To conservatively quantify the interconnect combinations, we use the lowest possible gate count (n). This is attained by dividing the number of columns (m) by the width of the largest gate (x_L). Adhering to modern design trends, we set f = 5.

Impact of layer obfuscation: The virtual layers in TRAP cannot be identified from its physical structure and can be executed in any sequence. Thus, for a fabric with l layers, the number of possible combinations is:

$$\textit{possible combinations of layers} = l^l \quad (4)$$

From Equations 2, 3 and 4, the overall impact of TRAP's gate, interconnect and layer obfuscation can be quantified as:

$$\textit{possible total combinations} = k \cdot 2^{f(2n-f-1)} \cdot l^l \quad (5)$$

revealing that the SAT formula grows at a *multiplicative-exponential* rate with design size.

VI. Evaluation

We now proceed to quantify the security level achieved and the overhead incurred when applying the TRAP-based obfuscation method to various modules of the FabScalar [13] microprocessor design. FabScalar is a well-known, synthesizable, open-source microprocessor, whose functionality has been verified through multiple fabricated ICs. The design implements a canonical template of an out-of-order, superscalar core and utilizes the Portable-ISA (PISA) as its instruction set architecture. Figure 8 depicts the various stages in FabScalar's architectural pipeline, along with the design modules that have been obfuscated in this study.

A. Achieved Security

In order to reflect the correlation between the size of the obfuscated portion of a design and the achieved security level, we study three different cases where we obfuscate incrementally larger portions of the FabScalar processor. The target modules are obfuscated with both single (`TRAP-1L`) and four-layer (`TRAP-4L`) TRAP fabric, and we consider both the brute-force and the SAT-based attack model, assuming that the attacker can process 1 billion combinations per second. Table I summarizes our findings.

Case 1 – AMT: We first obfuscate the Architecture Map Table (AMT), which represents a small (3.75%) yet crucial module of FabScalar's operation, as it resides inside the *Retire* stage and participates in register mapping and handling of misspeculated executions. Based on the analysis described in Section V, we observe that obfuscating the AMT with `TRAP-1L` provides excellent security, as the evaluation time for brute-force and SAT attacks is computed at 10^{1716} and 10^{512} hours, respectively. However, `TRAP-4L` provides even better security by elevating the brute-force and SAT attack evaluation time to 10^{3140} and 10^{514} hours, respectively.

Case 2 – AMT+RSR+BP: Next, we obfuscate three modules – the AMT, the result shift register (RSR), and the branch predictor (BP) – located across different pipeline stages, with a total design footprint of 8.58%. In this case, the de-obfuscation time for `TRAP-1L` grows to 10^{4305} and 10^{1317} hours for brute-force and SAT attacks, respectively. For `TRAP-4L`, brute-force and SAT attack would further increase to 10^{7862} and 10^{1319} hours, respectively, to unlock the design.

Case 3 – Dispatch: Lastly, we consider obfuscating the entire Dispatch stage of FabScalar's pipeline. Dispatch plays a critical role as it sits between the front- and back-end of the pipeline, and has a relatively large footprint of 14.34%. In this case, de-obfuscation time for brute-force and SAT attacks on `TRAP-1L` skyrockets to 10^{9315} and 10^{2871} hours, respectively. `TRAP-4L` further pushes these times to 10^{16998} and 10^{2873} hours, for a brute-force and SAT attack, respectively.

B. Incurred Overhead

The security levels achieved by TRAP-based obfuscation come at the cost of area, power and performance overhead. Below, we report this overhead over an ASIC implementation of Fabscalar, and we contrast to the overhead that would be incurred if the obfuscated modules were implemented on an FPGA fabric. For this purpose, we compare our 65nm TRAP fabric vis-a-vis one of the most prominent commercial FPGAs in this technology node, namely the `Xilinx Virtex 5`. Area and latency data were collected from Xilinx's post-synthesis design report, while power results were acquired from the post place-&-route simulation model (XPower Analyzer). We note that the `Virtex 5` FPGA contains large block RAMs (BRAM) to facilitate specific applications (e.g., DSP), which occupy more than 50% of the FPGA area and consume static power even when unused [17]. To make the overhead comparison fair, we disabled BRAM usage during synthesis, subtracted the total BRAM area from the die size, and adjusted the power consumption accordingly. Additionally, to be conservative and keep the field level, we only contrast the FPGA versus the `TRAP-1L` version of our fabric, which does not leverage time-sharing.

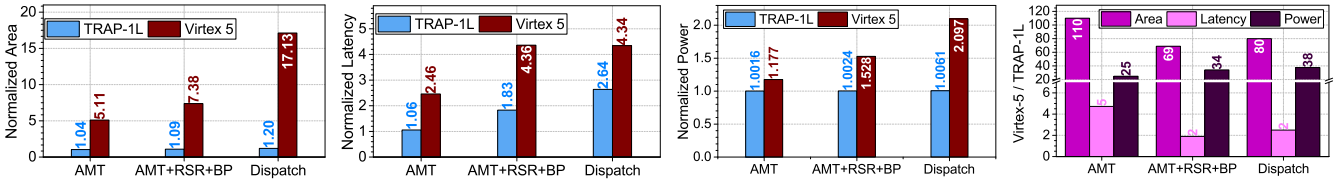
Area Overhead: Figure 9a reports the area overhead for `TRAP-1L` and Virtex 5, normalized to the total area of a 65nm full-ASIC implementation of the entire FabScalar processor. While TRAP-based implementation of the obfuscated modules cannot quite reach the *full-custom* density, its area overhead is vastly better than the FPGA. For `TRAP-1L`, the area overhead for implementing AMT, AMT+RSR+BP, and Dispatch are 1.04X, 1.09X, and 1.20X, respectively, whereas for `Virtex 5`, the area overhead increases to a prohibitive 5.11X, 7.98X and 17.13X, respectively.

Latency Overhead: Figure 9b compares `TRAP-1L` and `Virtex 5` in terms of latency overhead, normalized to that of a full-ASIC Fabscalar implementation. Again, `TRAP-1L` delivers a significantly lower latency overhead as compared to the FPGA. For TRAP, the latency overhead is 1.06X, 1.83X, and 2.64X for the AMT, AMT+RSR+BP, and Dispatch modules, respectively. In comparison, `Virtex 5` latency overhead is significantly higher, at 2.46X, 4.36X, 4.34X, respectively.

Power Overhead: Efficient resource utilization allows TRAP to contain power consumption overhead to very low levels. As shown in Figure 9c, despite the increased area, `TRAP-1L`'s power consumption overhead remains below 1.01X of that of a full-ASIC Fabscalar implementation in all three cases. On the other hand, when implementing the AMT, AMT+RSR+BP, and Dispatch module on `Virtex 5`, the power overhead is much larger, at 1.18X, 1.53X, and 2.10X, respectively.

TABLE I: Security level achieved by TRAP-based obfuscation of various modules of the FabScalar microprocessor design.

Obfuscated Module (footprint) →		**AMT** (3.75%)		**AMT+RSR+BP** (8.58%)		**Dispatch** (14.34%)	
Attack Category	**TRAP Implementation**	**Possible Combinations**	**Eval Time [log10(hrs)]**	**Possible Combinations**	**Eval Time [log10(hrs)]**	**Possible Combinations**	**Eval Time [log10(hrs)]**
Brute-force Attack	TRAP-1L	2^{5746}	1716	2^{14348}	4305	2^{30991}	9315
	TRAP-4L	2^{10478}	3140	2^{26164}	7862	2^{56513}	16998
SAT-based Attack	TRAP-1L	2^{1747}	512	2^{4420}	1317	2^{9583}	2871
	TRAP-4L	2^{1755}	514	2^{4428}	1319	2^{9591}	2873



(a) Area overhead. (b) Latency overhead. (c) Power overhead. (d) TRAP vs. FPGA.

Fig. 9: Overhead incurred by TRAP-based obfuscation of FabScalar modules and comparison to conventional FPGAs.

The comparative advantage of TRAP over FPGA is further elucidated in Figure 9d, where we normalize the overhead of an FPGA-based obfuscation to that of a TRAP-based obfuscation for each of the three modules. As compared to `TRAP-1L`, `Virtex 5` leaves a 110X, 69X, and 80X larger footprint, results in a 2X, 5X, and 2X slower performance, and consumes 25X, 34X and 38X more power when implementing the AMT, AMT+RSR+BP, and Dispatch module, respectively. When also taking into account the tool-flow support for seamlessly integrating and simulating a mixed ASIC-TRAP design and the increased uncertainty that it poses to an adversary, it becomes evident that the TRAP fabric is superior to FPGA-style fabrics for the purpose of design obfuscation.

C. Discussion

We reiterate that TRAP-based obfuscation is fundamentally different than solutions which rely on a key for de-embedding the obfuscated functionality from a larger function. Independent of how embedding is achieved [3], [8], [18], the obfuscated logic is a subset of the visible function, hence intelligent attacks that enable functional pruning of the search space can always be devised. In contrast, the proposed solution structurally obfuscates the sensitive functionality by omitting it entirely. This, in turn, makes the search space overwhelmingly large. While a similar result can be achieved through FPGA-style fabrics, the corresponding overhead is prohibitive. TRAP, however, leverages its transistor-level granularity not only for containing the overhead within realistic levels comparable to embedding-based solutions, but also prevents the formulation of algebraic logic-level intelligent attacks.

We also note that concerns regarding secure handling of the TRAP-fabric bitstream are no different than those for FPGA-style fabrics or key-based solutions. The key or bitstream must remain secret, hence it should be stored either externally and applied/loaded at power-on, or internally through secure, "tamper-proof" memory. Encryption of the bitstream through a symmetric cipher (e.g., AES) and elimination of bitstream read-back capabilities from a programmed device, as currently practiced to protect design IP in commercial FPGAs, can also be employed to further protect the secret from being compromised, or even customize it to each individual chip. Once again, we emphasize that the proposed TRAP-based obfuscation is no different in this regard than previous solutions and can readily adopt the above schemes to ensure maximum protection of the programming bitstream.

VII. Conclusions

Selective post-fabrication transistor-level programming offers a novel solution for obfuscating sensitive designs and preventing their unauthorized use or reproduction. Through a dense transistor-level programmable fabric, virtualization, and appropriate CAD support for seamless integration of the ASIC and TRAP-obfuscated portions of a design, this solution is practically unbreakable through either brute-force or intelligent SAT-based attacks. At the same time, it incurs realistic area, power and performance overhead, on par with key-based solutions and an order of magnitude less than FPGA-style fabrics. Feasibility of TRAP-based obfuscation was corroborated through a silicon implementation of a prototype chip and its effectiveness was comparatively evaluated through obfuscation of modules of a modern microprocessor.

References

[1] U. Guin, K. Huang, D. DiMase *et al.*, "Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1207–1228, 2014.

[2] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[3] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending Piracy of Integrated Circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[4] J. Rajendran, H. Zhang, C. Zhang *et al.*, "Fault Analysis-based Logic Encryption," *IEEE ToC*, vol. 64, no. 2, pp. 410–424, 2015.

[5] B. Liu and B. Wang, "Embedded Reconfigurable Logic for ASIC Design Obfuscation against Supply Chain Attacks," in *DATE*, 2014.

[6] R. S. Chakraborty and S. Bhunia, "Hardware protection and authentication through netlist level obfuscation," in *ICCAD*, 2008.

[7] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the Security of Logic Encryption Algorithms," in *HOST*, 2015.

[8] Y. Xie and A. Srivastava, "Mitigating SAT Attack on Logic Locking," in *Conference on Cryptographic Hardware and Embedded Systems*, 2016.

[9] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "Sarlock: Sat attack resistant logic locking," in *HOST*, 2016.

[10] K. Shamsi, M. Li, T. Meade *et al.*, "AppSAT: Approximately Deobfuscating Integrated Circuits," in *HOST*, 2017.

[11] T. Winograd, H. Salmani, H. Mahmoodi *et al.*, "Hybrid STT-CMOS Designs for Reverse-Engineering Prevention," in *DAC*, 2016.

[12] J. Tian *et al.*, "A field programmable transistor array featuring single-cycle partial/full dynamic reconfiguration," in *DATE*, 2017.

[13] N. Choudhary, S. Wadhavkar, T. Shah *et al.*, "Fabscalar: Automating superscalar core design," *IEEE Micro*, vol. 32, no. 3, pp. 48–59, 2012.

[14] "TimberWolf Systems Inc." http://www.twolf.com/, 2017.

[15] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research," in *FPL*, 1997.

[16] U. Schöning and J. Torán, *The Satisfiability Problem: Algorithms and Analyses*. Lehmanns Media, 2013.

[17] M. Lin, A. El Gamal, Y.-C. Lu, and S. Wong, "Performance benefits of monolithically stacked 3-d fpga," *IEEE TCAD*, vol. 26 -2, 2007.

[18] S. Patnaik, M. Ashra, J. Knechtel *et al.*, "Obfuscating the interconnects: Low-cost and resilient full-chip layout camouflaging," in *ICCAD*, 2017.