IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2025, No. 2, pp. 579–603. DOI:10.46586/tches.v2025.i2.579-603

A TRAP for SAT: On the Imperviousness of a Transistor-Level Programmable Fabric to Satisfiability-Based Attacks

Aric Fowler[1], Shayan Mohammed[1], Mustafa Shihab[1], Thomas Broadfoot[1], Peter Beerel[2], Carl Sechen[1] and Yiorgos Makris[1]

[1] University of Texas at Dallas, Richardson, TX, USA
[2] University of Southern California, Los Angeles, CA, USA

Abstract. Locking-based intellectual property (IP) protection for integrated circuits (ICs) being manufactured at untrusted facilities has been largely defeated by the satisfiability (SAT) attack, which can retrieve the secret key needed for instantiating proprietary functionality on locked circuits. As a result, redaction-based methods have gained popularity as a more secure way of protecting hardware IP. Among these methods, transistor-level programming (TRAP) prohibits the outright use of SAT attacks due to the mismatch between the logic-level at which SAT attack operates and the switch-level at which the TRAP fabric is programmed. Herein, we discuss the challenges involved in launching SAT attacks on TRAP and we propose solutions which enable expression of TRAP in propositional logic modeling in a way that accurately reflects switch-level circuit capabilities. Results obtained using a transistor-level SAT attack tool-set that we developed and are releasing corroborate that SAT attacks can be launched against TRAP. However, the increased complexity of switch-level circuit modeling prevents the attack from realistically compromising all but the most trivial IP-protected designs.

Keywords: Logic Redaction · SAT Attack · TRAP

1 Introduction

Contemporary semiconductor manufacturing predominantly follows a fabless business model, wherein the silicon fabrication of integrated circuits (ICs) is outsourced to external foundries around the globe [VVGY21, IC 21]. While this approach is justified by financial and geopolitical reasons, it introduces security challenges since the blueprint of a design must be released to a potentially untrusted foundry for fabrication. This exposes the intellectual property (IP) within the design to risks of reverse engineering, theft, and unauthorized overproduction. In response to these threats, various methods such as logic locking [BTZ10a, CB09, RKM08, XS19], gate camouflaging [CBCW14, BCCW12], and split manufacturing [VDS+14, IEGT13, JM07] were developed to prevent IP from falling into adversarial hands [CB14, YSN+17].

As is typical in any arms race, alongside these IP protection efforts, numerous attacks seeking to compromise confidentiality of a design have also been developed [EMGT19, WCHR16, LPS+19, SS19, RPSK12]. These attacks aim to retrieve a secret key value which instantiates the correct functionality within a protected design. Among them, the invention of the SAT attack [SRM15] was a pivotal point in this race. Together with its many extensions [HZY+21, ZJK17, SPJ19, KCV19, JHCK20, LPS21, EMGT19, AYS+19], the SAT attack has successfully broken most of these defenses and has raised the bar for IP protection methods. Its success stems from (i) clever formulations of **satisfiability**

Licensed under Creative Commons License CC-BY 4.0.
Received: 2024-10-15 Accepted: 2024-12-15 Published: 2025-03-04

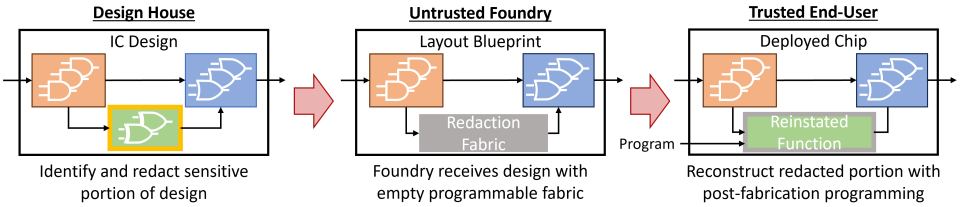


Figure 1: Hardware IP protection through IC redaction

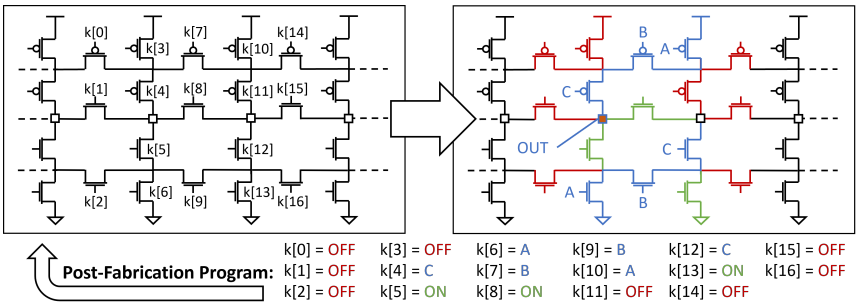


Figure 2: Programming TRAP with the key for NOR3 functionality ($OUT = \overline{(A \vee B \vee C)}$)

(SAT) problems whose solutions enable rapid pruning of the viable key space, (ii) recent advancements in SAT solvers (*i.e.*, the SAT attack engine) [GV21, AAASM19], and (iii) the concordance of the attack's logic-level operation with the logic gate composition of locking IP protection methods.

In response to the success of SAT attacks, **logic redaction** methods which replace sensitive functionalities with a programmable circuit fabric that reinstates the missing functionality after IC fabrication have emerged [BTZ10b, MAS+21, BMT+23, BTMT+21, KBK24, WMMS21, GRK+23, APMP23], as shown in Figure 1. Among these methods, the **transistor-level programmable (TRAP)** fabric [TRW+17] stands out because it lowers the granularity of programmability and function implementation below the level of logic gates. In this approach, the secret key is a bitstream that turns select elements in a sea of transistors ON or OFF, stitching them together into logic functions. An example of this paradigm implementing a NOR3 gate is shown in Figure 2, with further details provided in Section 5. Consequently, unlike gate-level protection methods where an erroneous key value results in an incorrect function, an incorrect TRAP key value may result in a transistor topology that does not even implement a logic function.

TRAP owes its resilience against SAT attacks to this discrepancy between logical program and electrical operation, which SAT solvers and, by extension, SAT attacks, do not natively understand. Specifically, notions such as signal direction, electrical drive, and high impedance, which are indispensable circuit-level attributes, are foreign to SAT attacks. The same holds true for cyclical topologies and tri-state signals, which are common in transistor-level designs but do not exist in gate-level circuits. As a result, the utility and efficacy of SAT attack in breaking TRAP has yet to be explored.

A theoretical security assessment of the TRAP architecture outlined in [TRW+17] was conducted in [STR+19]. However, no quantitative experimental results from actual SAT attacks were provided and no conclusions based on actual metrics were drawn. Toward filling this void, the novel contributions of this paper include:

- An analysis of the challenges involved in modeling a transistor-level redaction fabric and a derivation of solutions to enable application of SAT attacks to such circuits.

- A quantitative assessment of the resilience of TRAP to SAT attacks, which corroborates the major breakthrough offered by logic redaction methods to the problem of hardware IP protection.

The remainder of this paper is structured as follows: Section 2 states the threat model considered in this work. Section 3 provides an overview of the conventional SAT attack. Section 4 elucidates the additional provisions required for extending applicability of SAT attacks to encompass switch-level transistor behavior. Section 5 describes the architecture of TRAP, for which the full propositional logic model is derived in Section 6. Section 7 details our custom tool-set which enables SAT attacks to be launched on transistor-level circuits (including TRAP) and Section 8 highlights the resistance of TRAP to attacks launched by this tool-set, which is further discussed in Section 9.

2 Threat Model

In the attack scenario considered in this work, an adversary is attempting to steal IP from an IC design, a portion of which has been redacted using a TRAP fabric, by way of a SAT attack. To launch the attack, we assume that the adversary has access to two elements: (i) a simulatable netlist of the protected IC design, and (ii) an unlocked operational part, commonly referred to as an oracle, which allows black-box interrogation while preserving the secrecy of the key. Furthermore, we assume that the adversary is able to access directly the I/Os of the TRAP fabric through the use of a boundary scan chain, which is typically available for the purpose of manufacturing testing, thereby eliminating the need to include the much larger non-redacted circuit in the SAT attack. Armed with these capabilities, a SAT attack can be launched to seek the secret key that instantiates the IP functionality onto an unprogrammed circuit.

3 Gate-Level SAT Attack Fundamentals

To fully appreciate the challenges that arise while modeling and attacking TRAP with a SAT attack, we start with a short review of its fundamental mechanics. The SAT attack is an elegant algorithmic approach for retrieving a secret key that is necessary for realizing some intended functionality embedded within a protected digital circuit.

The overarching strategy of the SAT attack is to rapidly prune incorrect values from the possible key space in an iterative manner, until only correct key(s) remain. To this end, it leverages the information gained by contrasting erroneous simulated circuit outputs from incorrect keys against the correct oracle output for judiciously selected inputs values. The original SAT attack [SRM15] and its numerous extensions [KCV19, JHCK20, LPS21, HZY+21, ZJK17, SPJ19] have been shown to be highly effective at deriving secret keys and breaking protected gate-level circuits that are too complex to solve using brute-force methodologies.

3.1 SAT Attack Flow

The SAT attack procedure involves four distinct tasks:

1. **Miter Formation:** In this task, two copies of the protected circuit are instantiated with common primary inputs but separate key inputs. A comparator then assesses equivalence of the outputs of these two copies. This construct is referred to as a **miter** and, expressed in propositional logic, serves as the basis for the SAT attack.

2. **DIP Identification:** In this task, a SAT solver (*i.e.*, the algorithmic engine that SAT attacks run on) searches for a primary input pattern for which the two copies

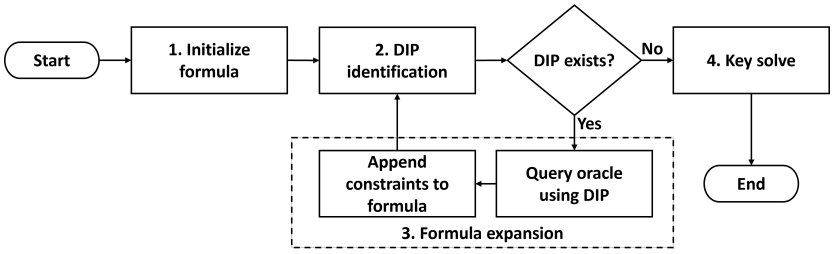


Figure 3: SAT attack flow diagram

of the protected circuit in the miter produce different outputs. This is achieved by solving the formula of the miter such that it produces a logic '1' at its output. The resulting primary input pattern causing the output discrepancy is referred to as a **distinguishing input pattern (DIP)**. Existence of a DIP implies that at least one of the two keys separately feeding the circuit copies is wrong.

3. **Miter Expansion:** In this task, the oracle is queried with the DIP value from the previous task and the corresponding correct function output is recorded. The DIP and output are then used to further constrain the search space. Specifically, two new copies of the protected circuit are added to the miter, with their primary inputs matched to the logic values of the DIP and their key inputs tied to the key inputs of the original two copies of the protected circuit. The outputs of the two new copies are forced to match the oracle output for the DIP, as a prerequisite for this expanded miter to produce a '1' at its output. Essentially, the expanded miter implicitly eliminates from the viable key options all values for which the previously identified DIP produces an output that differs from the oracle output for that DIP.

4. **Key Retrieval:** In this final task, the information gathered from interrogating the oracle with DIPs and collecting the corresponding responses is used to retrieve the correct key through a new SAT formulation. A new construct is devised, wherein a copy of the protected circuit is instantiated for each DIP found. Each copy receives a unique DIP as a constant input and is forced to produce the corresponding oracle output. All copies share the same key inputs. The SAT solver is then charged with solving for one key such that all constraints of the formula representing this construct are satisfied. In turn, this guarantees that the resulting key will instantiate a circuit that will agree with the output of the oracle for every DIP included in the construct. This correlation extends to every possible input pattern, not just DIPs.

As shown in the flow diagram of Figure 3, the SAT attack begins with miter formation. In the first round, solving this miter identifies a DIP. If a DIP is found, the oracle is consulted and the miter is expanded with additional constraints that prohibit revisiting the DIP. This process repeats in successive rounds until no more DIPs exist. At this point, all available information has been collected and the key retrieval task is performed to obtain the correct key. In cases where different keys instate identical circuit functionality, the key returned by the key retrieval task is guaranteed to realize the same outputs as the oracle for all input combinations, but is not guaranteed to be the exact key used in the oracle.

3.2 Miter Comparator

Among the components involved in the SAT attack, we call attention to the comparator that checks for output equivalence during DIP identification. Shown in Figure 4 is a two-level OR-of-XORs logic gate representation of the comparator typically used to detect any difference in the binary output vectors A and B produced by the two copies of the

protected circuit. When this comparator is included in the miter expression and its output is explicitly forced to '1', a SAT solver attempts to manipulate the two circuit copies such that their outputs differ in at least one bit. While this works well for gate-level digital circuits, modifications are required to accommodate a more elaborate definition of output equivalence needed for transistor-level circuits, as we discuss in Section 4.4.

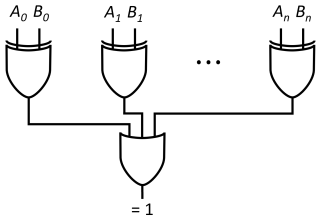


Figure 4: Comparator for n-output circuit

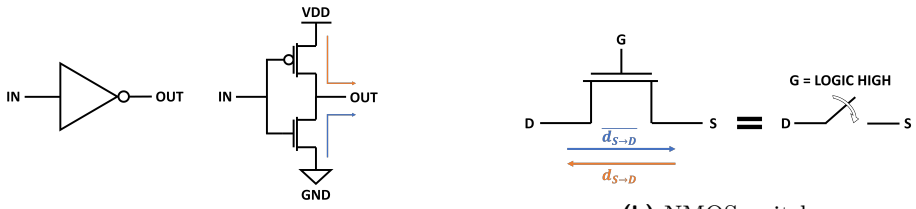
3.3 SAT Solvers

Possibly the most important consideration to make when writing propositional logic models that faithfully reflect circuit operation is to understand how a SAT solver works. SAT solvers employ variations of the Davis–Putnam–Logemann–Loveland (DPLL) algorithm [DP60, BM07] to search for possible solutions to a conjunctive normal form (CNF) logic formula by assigning values to variables such that no clauses within the formula contradict each other. This work utilizes a satisfiability modulo theories (SMT) solver, which includes SAT solving in its capabilities, to convert handwritten models into CNF using internal optimization tactics before solving [dMB08]. Poising a SAT or SMT solver to find a solution to a miter[1] causes the solver to return the first immediate solution it discovers; if no solution exists, the solver definitively proves so before declaring the formula unsatisfiable. Failing to properly constrain models gives agency to the solver to manipulate them in ways that the modeled circuitry is incapable of, which may elicit unfaithful solutions.

4 Transistor-Level SAT Attack Adaptation

Applying the SAT attack described in Section 3 to TRAP accentuates multiple complications that arise when applying SAT attacks to any design with transistor-level switch operation. Notions such as high-impedance, current direction, and electrical drive are indispensable when describing transistor-level circuit functionality, yet they are not innately understood by SAT solvers. Therefore, additional modeling effort is needed to express these electrical notions as Boolean constructs and make them compatible with the operating level of SAT solvers. Diligence in this modeling effort is particularly important, as incompletely constrained formulations may allow the SAT solver to arrive at solutions which satisfy the logic expression but do not reflect the actual capabilities of the circuit. In this section, we elaborate on these challenges and introduce solutions to overcome them.

[1] Miter formation and solution is an inherent equivalence checking capability of logic synthesis and verification packages, such as ABC [Ber12]. Therein, the traditional SAT formulation is complemented by heuristics (*e.g.*, *fraiging* in ABC). While such heuristics have proven helpful in the context of synthesis, they are not effective in expediting SAT attacks, hence the hardware security community has not incorporated them in state-of-the-art SAT attack software implementations. A short study comparing SAT attack effectiveness when using ABC's SAT solver [ES04] with and without fraiging, as well as when using a contemporary SMT solver (*i.e.*, Z3), can be found in https://github.com/aric-fowler/TRANSAT.git



(a) Inverter symbol and CMOS schematic

(b) NMOS switch

Figure 5: Transistor-Level Circuits

4.1 Transistor-Level Modeling of CMOS Gates

We first explain why modeling conventional CMOS logic gates at the transistor-level does not suffer from the aforementioned challenges. From a logic perspective, transistors are three-terminal switches that connect their source and drain terminals when the proper voltage is placed at their gate terminal. In digital logic, signal voltages are discrete values, so the operation of a transistor as a switch can be modeled using Boolean implication.

Consider the example CMOS topology of an inverter shown in Figure 5a. As in all CMOS architectures [WH05], this design consists of transistor switches that compose a pull-up and a pull-down path. The pull-up path (in orange) connects power supply VDD to OUT when $IN = 0$ while the pull-down path (in blue) connects ground GND to OUT when $IN = 1$. Connecting two nodes in this manner through a transistor switch is referred to as **logical causality** and is expressed in propositional logic as a Boolean implication. Here, Equation (1) describes the pull-up path and Equation (2) describes the pull-down path.

$$\overline{IN} \Rightarrow (VDD \Leftrightarrow OUT) \tag{1}$$
$$IN \Rightarrow (GND \Leftrightarrow OUT) \tag{2}$$

Transistor-level implementation of CMOS logic gates relies on a couple of assumptions which simplify their propositional logic models. The first assumption is that gate inputs are arranged such that the pull-up and pull-down paths are mutually exclusive. In the case of the CMOS inverter of Figure 5a, the logic value of input IN can never be simultaneously high and low, so the implications described in Equations (1) and (2) can be combined into the single equivalence statement of Equation (3). The second assumption is that the signal direction of the pull-up and pull-down paths is implicit because VDD and GND are power rails enforcing a constant logical value. Since they always have greater electrical charge than the CMOS gate output, OUT will always take on a logical value from VDD or GND; never the other way around. This second assumption also allows us to substitute VDD and GND with constant '1' and '0' values, respectively, thereby reducing Equation (3) to the familiar gate-level Equation (4). This final simplification shows that, while it is possible to model a conventional CMOS gate at the transistor level, it always reduces to the gate-level formula, circumnavigating any complications that arise from transistor-level modeling.

$$OUT \Leftrightarrow (IN \wedge GND) \vee (\overline{IN} \wedge VDD) \tag{3}$$
$$OUT \Leftrightarrow \overline{IN} \tag{4}$$

4.2 Enabling Single-Transistor Modeling

The CMOS inverter covered in Section 4.1 is naturally compatible with a SAT attack because the complete behavior of the digital circuit, Equation (4), is described by the

same logic-level implications that a SAT solver natively understands. In non-CMOS transistor-level implementations of digital circuits, however, accurate modeling of even a single transistor becomes more involved, as it needs to incorporate the notions of **signal direction** and **electrical drive**. Consider the NMOS transistor switch shown in Figure 5b, for which Equation (5) describes logical causality between the logic values at the gate G, source S, and drain D.

$$G \Rightarrow (S \Leftrightarrow D) \tag{5}$$

When the transistor is ON (closed switch), S and D share a logical value. This is the same relationship that was previously used in transistor-level modeling of the CMOS logic inverter in Equations (1) and (2). However, unlike in CMOS logic gates, where the direction of signal flow is implicitly known (*i.e.*, the source drives the drain of each transistor in the gate), the signal flow through a single NMOS transistor could be either from the source to the drain or vice versa. Furthermore, when the transistor is OFF (open switch), the source and drain are unable to influence each other, and one or both may become unconstrained. A SAT solver is free to assign these unconstrained variables any value it chooses, ignorant of electrical operation. This renders Equation (5) insufficient to enforce complete causality between G, D, and S of the NMOS switch. Additionally, if S and D are both floating nodes, they would be incapable of electrically driving one another, even when the transistor is ON. Note that this issue does not exist in gate-level logic, where all outputs are guaranteed to be electrically driven.

Electrical drive and signal direction, which are both necessary for deriving an accurate propositional logic model for the NMOS transistor, can be captured by introducing appropriate Boolean variables. Specifically, a drive variable can be assigned to each electrical node to supplement their existing logic values (*e.g.*, V_D for D, V_S for S). When a drive value at a particular node is '1', it implies that the node is electrically driven and, consequently, its logic value is **valid**. Conversely, if the drive value of a node is '0', then the logic value of the node should be considered arbitrary and inconsequential. A direction variable can be assigned to each transistor (*e.g.*, $d_{S\rightarrow D}$ for the NMOS switch). A value of '1' for a direction variable indicates that the source is driving the drain, while a value of '0' indicates that the drain is driving the source, as shown by the arrows in Figure 5b.

Using these variables, we can determine whether the value of a transistor terminal is valid by examining whether it is electrically connected through a path to a power rail. For the source terminal, this can be established (i) through a direct connection to a primary input, *VDD* or *GND*, which we express through a variable $S_{isInOrPWR}$ set to '1', or (ii) through the transistor itself, if the value of the drain terminal is driven, the transistor is ON, and the direction variable of the transistor indicates that the signal flow is from the drain to the source. For the drain terminal, this can be established (i) through a direct connection to a primary input, which we express through a variable D_{isIn} set to '1', or (ii) through the transistor itself, if the value of the source terminal is valid, the transistor is ON, and the direction variable of the transistor indicates that the signal flow is from the source to the drain. In propositional logic, these conditions are captured by the inclusive disjunctions of Equations (6) and (7). We note that these are equivalences rather than implications, because assessing electrical drive is absolute; either a node sees a path to an electrical source or it does not. Together with Equation (5), these clauses form a complete model for a single NMOS transistor.

$$V_S \Leftrightarrow (S_{isInOrPWR} \vee (V_D \wedge G \wedge \overline{d_{S\rightarrow D}})) \tag{6}$$

$$V_D \Leftrightarrow (D_{isIn} \vee (V_S \wedge G \wedge d_{S\rightarrow D})) \tag{7}$$

A propositional logic model for a PMOS transistor can be derived similarly using the same logical, drive, and direction variables as for the NMOS transistor. Note that the logical variable G must be negated to reflect the reversed ON state of PMOS operation.

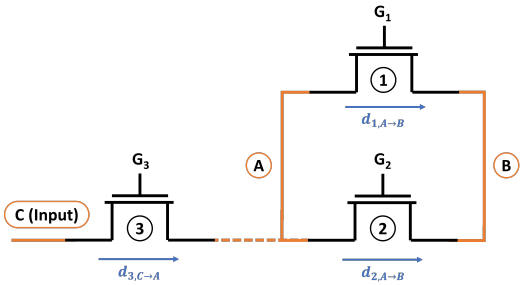


Figure 6: Cyclical structure example in transistor-level circuit

4.3 Handling Cyclical Structures

Another challenge while modeling transistor-level circuits in propositional logic arises due to cyclical connectivity. Consider the example two-transistor loop in Figure 6, along with the third transistor which selectively connects an externally-driven input C to node A of the loop. Using the methods devised for a single transistor in Section 4.2, a model for the circuit can be derived, as shown in Equations (8) to (11).

$$(G_i \Rightarrow (S \Leftrightarrow D), \quad (i, S, D) \in \{(1, A, B), (2, A, B), (3, C, A)\} \tag{8}$$

$$V_A \Leftrightarrow ((V_B \wedge G_1 \wedge \overline{d_{1,A\to B}}) \vee (V_B \wedge G_2 \wedge \overline{d_{2,A\to B}}) \vee (V_C \wedge G_3 \wedge d_{3,C\to A})) \tag{9}$$

$$V_B \Leftrightarrow ((V_A \wedge G_1 \wedge d_{1,A\to B}) \vee ((V_A \wedge G_2 \wedge d_{2,A\to B})) \tag{10}$$

$$V_C \Leftrightarrow (C_{isIn} \vee (V_A \wedge G_3 \wedge \overline{d_{3,C\to A}})) \tag{11}$$

At first glance, these equations appear to form a fully causal model for the transistor loop in Figure 6. However, this model is not completely faithful to the operation of electrical circuitry: under the circumstances described in Example 1, this model can reduce to a **circular reasoning fallacy**, creating a **cyclical logic loop** that allows an electrically undriven signal path to be reasoned as driven by a solver.

Example 1. Let us assume that, in the circuit of Figure 6, which is described by Equations (8) to (11), node C is externally driven by an input signal ($C_{isIn} = 1, \therefore V_C = 1$). If transistor 3 is OFF ($G_3 = 0$), then nodes A and B in the circuit are floating, so their logical values should be disregarded as they cannot be electrically justified. Nonetheless, the propositional logic model for the circuit can still be satisfied if transistors 1 and 2 are ON ($G_1 = 1, G_2 = 1$). When both transistors are ON, direction variables $d_{1,A\to B}, d_{2,A\to B}$ must take on opposite values (*e.g.*, $d_{1,A\to B} = 1, d_{2,A\to B} = 0$) in order to simultaneously satisfy Equations (9) and (10). Upon the direction variables assuming values, Equations (9) and (10) reduce to duplicate expressions $V_A \Leftrightarrow V_B$. This means that the only constraint now is that the drive variables of nodes A and B (V_A and V_B) must be identical. Hence, they can both be arbitrarily set to '1', implying that their logical values (which are also shared by Equation (8)) are electrically driven! In short, the logic model does not reflect accurately the electrical operation of the circuit.

This inconsistency occurs because direction variables in the transistor loop are allowed to point toward each other in cyclical succession to drive a signal while still satisfying the constraints of the model. In hardware, this corresponds to a signal feeding into and driving itself, which is not physically possible. To prevent a SAT solver from arriving at this interpretation of the transistor loop model, a **path count** variable can be introduced at each node. These new integer count variables[2] cnt_A, cnt_B, and cnt_C are accompanied

[2]Integer variables can be translated to bit-vectors for SAT compatibility.

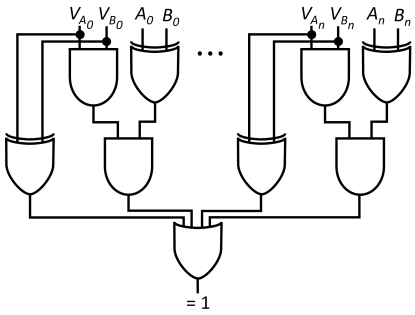


Figure 7: Comparator for tri-state encoded n-output circuit

by additional logic clauses that prevent direction variables from forming a loop in order satisfy electrical drive evaluation clauses such as Equations (9) to (11) when all transistors in the loop are turned ON. Thereby, they forbid these clauses from reducing to a circularity. Equations (12) to (15) dictate that when a transistor is ON and the direction variable indicates that a signal flows from source to drain, then the count variable value at the source must be less than the count variable value at the drain. The opposite applies when the direction variable delineates that the signal flows from the drain to the source.

Collectively, Equations (8) to (15) compose an enhanced model that resolves the inconsistency between logic and circuit operation incurred by cyclic structures. Revisiting Example 1, an attempt to set the direction variables in a clockwise path (*i.e.,* $d_{1,A\to B} = 1, d_{2,A\to B} = 0$) causes Equations (12) and (15) to contradict each other. Similarly, an attempt to set the direction variables in a counterclockwise path (*i.e.,* $d_{1,A\to B} = 0, d_{2,A\to B} = 1$) that creates a loop would cause a conflict between Equations (13) and (14). Therefore, the enhanced model prevents cyclic structures from inducing circular reasoning, ensuring that the propositional logic is congruent with the actual operation of the circuit.

$$(G_1 \wedge d_{1,A\to B}) \Rightarrow (cnt_A < cnt_B) \tag{12}$$
$$(G_1 \wedge \overline{d_{1,A\to B}}) \Rightarrow (cnt_B < cnt_A) \tag{13}$$
$$(G_2 \wedge d_{2,A\to B}) \Rightarrow (cnt_A < cnt_B) \tag{14}$$
$$(G_2 \wedge \overline{d_{2,A\to B}}) \Rightarrow (cnt_B < cnt_A) \tag{15}$$

4.4 Supporting Tri-State Nodes in SAT Attack

In addition to the challenges associated with deriving a propositional level model that accurately reflects the capabilities of a transistor-level design, which were addressed in Sections 4.2 and 4.3, launching a SAT attack on such a design faces obstacles related to the high-impedance state that may occur on internal nodes or even outputs.

First, since SAT solvers only understand binary abstractions but nodes may be in one of three states (*i.e.,* '0', '1' or 'high-impedance'), a second Boolean variable is required to express the state of a node. Conveniently, the electrical drive Boolean variable introduced in Section 4.2 fulfills this role perfectly, since a node is in a high-impedance state whenever it is not electrically driven. Together, the variable expressing the logical value of a node A and the variable expressing electrical drive validity V_A sufficiently encode tri-state logic.

Second, since the tri-state outputs of a transistor-level circuit have to be encoded using both a logic variable and an electrical drive variable, the miter comparator must also be adjusted to account for high-impedance when assessing circuit output differences. The original comparator described in Section 3.2 assesses whether two signals (A_i, B_i) are different through a simple XOR gate. However, in the case of tri-stated signals,

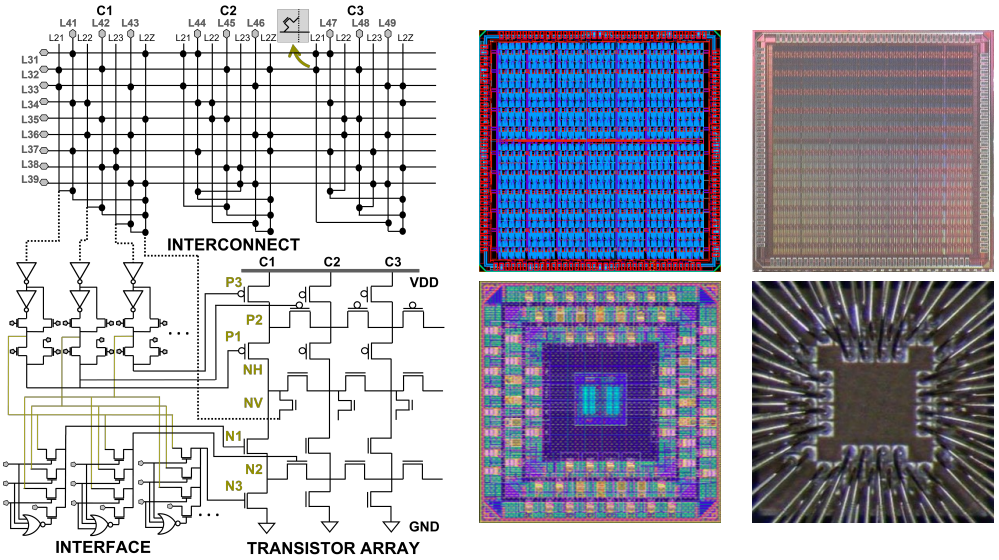


Figure 8: Architecture of a single TRAP unit, layouts, and die photos of TRAP fabrics

such a difference is only meaningful when the two signals are electrically driven (*i.e.,* $V_{A_i} = V_{B_i} = 1$). With transistor-level granularity, additional opportunities to discover DIPs exist when one of the two signals is electrically driven while the other is in a high-impedance state (*i.e.,* $V_{A_i} \neq V_{B_i}$), as these two signals can never be equivalent. However, this is not the case when both signals are in high-impedance state, since a generalized transistor-level digital circuit may express legitimate tri-state outputs. A gate-level representation of an augmented comparator that can handle the high-impedance state of encoded tri-state signals based on this discussion is shown in Figure 7.

5 TRAP Architecture

Applying the generalized modeling principles from Sections 4.1 to 4.3 to TRAP requires sufficient knowledge of the target circuitry. TRAP [TRW+17] is a patented topology [tra19, tra20, tra22], developed for protecting hardware IP through redaction [STR+19]. TRAP has been demonstrated in silicon in both planar (*i.e.,* GlobalFoundries 65nm) and FinFET (*i.e.,* GlobalFoundries 12nm) fabrication technologies. A high-level architectural description of TRAP is provided in this section, to assist with understanding the detailed modeling of TRAP in Section 6.

Conceptually, TRAP is a continuous sea of both PMOS and NMOS transistors, on which digital logic (including latches and flip-flops) can be formed by connecting these transistors in appropriate topologies. These connections are delegated by programming bits that can control the ON/OFF state of select transistors, and signals are routed from the surrounding interface and interconnect network. In practice, a TRAP fabric is implemented by tiling identical **TRAP units** in a rectangular array. Neighboring units can seamlessly pass signals between them and accommodate implementation of functions that transcend unit boundaries, thereby supporting the view of the TRAP fabric as a sea of transistors.

The prototypical TRAP unit, whose architecture is shown in Figure 8, is composed of three distinct sections. Collectively, the unit houses configurable logic functions within its **transistor array (TA)**, supplies inputs and control signals to the TA through its **interface**, and routes signals internally and to other TRAP units through the **interconnect**.

5.1 Transistor Array (TA)

The transistor array is an arrangement of twenty-four transistors that instate logic functions through the formation of pull-up paths to VDD and pull-down paths to GND. Its transistor-level programmability allows for numerous configurations to realize a particular function, some of which may follow the paradigm of standard CMOS logic gates while others may explore alternative design styles. As shown in Figure 8, the TA of a single unit contains three columns (C1, C2, and C3). Each TA column consists of four vertically-oriented transistors (P3, P1, N1, and N3) and three horizontally-oriented transistors (P2, NH, and N2) for logic implementation. An eighth transistor (NV) connects TA outputs to the interconnect. TAs of horizontally-adjacent TRAP units can be directly connected by turning on the horizontal transistors along the TA periphery (C3P2, C3NH, C3N2).

5.2 Interface

Each column of the TA is driven by its own programmable interface, which is the only part of the TRAP unit with predetermined signal flow. Figure 8 shows the interface for the first column C1 of the TA. The interface consists of two halves which are separately tasked with feeding inputs to the PMOS and NMOS transistors of the TA, respectively. Transistors P1, P2, and P3 within the TA may receive a signal from the interconnect, the complement of that signal, or a constant programmable value. Transistors N1, N2, and N3 within the TA each may receive one of the three signals intended for a PMOS transistor within their column, the complement of any of those signals, or a constant value. The interface also provides constant signals to the NH and NV transistors.

5.3 Interconnect

The TRAP interconnect is a programmable structure tasked with routing input, output, and intermediate logic signals around the fabric. Signals coming into or leaving TRAP are introduced at the interconnect via physical wire connections. The interconnect is composed of three sets of wire tracks (denoted L2, L3, and L4) and the programmable transistor switches that connect them. A switch connection does not exist for every combination of wire tracks; existing connections are indicated as junctions in Figure 8. The twelve L2 tracks are local connections meant to carry signals to the interface and from the TA. The nine L3 tracks and nine L4 tracks are horizontal and vertical tracks, respectively, which can route signals around the TRAP unit. Additionally, two sets of nine transistor switches connect L3 and L4 tracks from a unit's interconnect to the L3 and L4 tracks of the neighboring units to the right and above. These devices are not shown in Figure 8 but are present and must be accounted for when modeling multi-unit TRAP fabrics.

6 Modeling TRAP in Propositional Logic

Using the modeling principles discussed in Sections 4.1 to 4.3, we now derive a propositional logic model for TRAP that is faithful to the circuitry described in Section 5. To intuitively explain the TRAP propositional model, we begin with the transistor array and extend upwards through the interface and interconnect for a complete TRAP unit model.

6.1 Transistor Array (TA) Model

Constructing a propositional logic model for the TA shown in Figure 8 builds upon the transistor-level modeling principles from Sections 4.2 and 4.3. Since all TA columns are identical, the model for a single column C_x, shown in Figure 9, can be tiled to describe the entire TA. Each transistor gate terminal is assigned a logic variable, while each net between

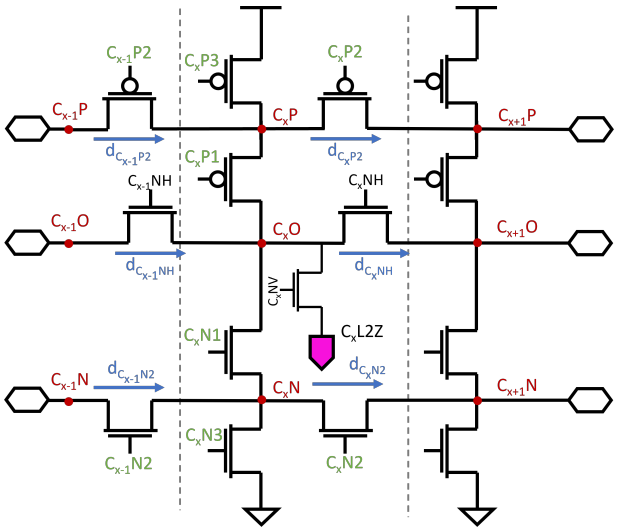


Figure 9: Modeling a single TA column C_x (dashed lines delineate column boundaries)

transistors is assigned a logic variable, a drive variable, and a count variable. Logical causation clauses, Equations (16) to (23), resemble Equation (5), with simplifications for transistors with a source terminal connected to power ($VDD = 1$) or ground ($GND = 0$). Constraints for electrical drive and direction, Equations (24) to (28), address the same considerations as Equations (6) and (7), but with an additional simplification: vertically-aligned transistors do not require direction modeling because it is assumed that the signal will always flow from VDD or GND (the same reasoning is used in Section 4.1). Thus, P2, N2 and NH transistors are the only TA devices that receive direction variables, shown as blue arrows in Figure 9. Count variables for Equations (29) to (37) are assigned just as in Section 4.3. The complete model for a single TA column C_x in Equations (16) to (37) is applicable to all TA columns within a unit, $x \in [1, 3]$.

$$\overline{C_xP3} \Rightarrow C_xP \tag{16}$$

$$\overline{C_xP2} \Rightarrow (C_xP \Leftrightarrow C_{x+1}P) \tag{17}$$

$$\overline{C_xP1} \Rightarrow (C_xP \Leftrightarrow C_xO) \tag{18}$$

$$C_xNH \Rightarrow (C_xO \Leftrightarrow C_{x+1}O) \tag{19}$$

$$C_xNV \Rightarrow (C_xO \Leftrightarrow C_xL2Z) \tag{20}$$

$$C_xN1 \Rightarrow (C_xN \Leftrightarrow C_xO) \tag{21}$$

$$C_xN2 \Rightarrow (C_xN \Leftrightarrow C_{x+1}N) \tag{22}$$

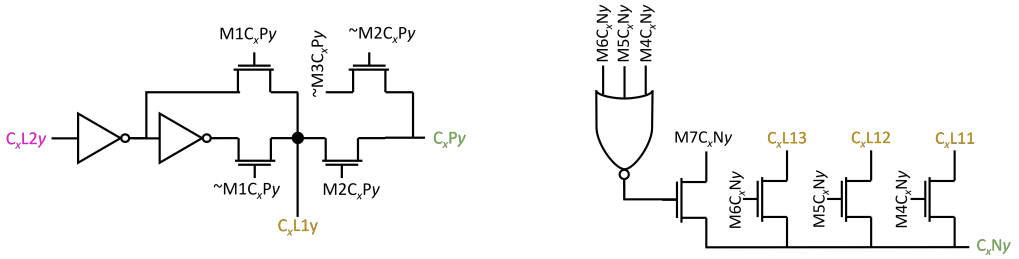
$$C_xN3 \Rightarrow \overline{C_xN} \tag{23}$$

$$C_xP1 \Rightarrow (d_{C_{x-1}P2} \wedge \overline{d_{C_xP2}}) \tag{24}$$

$$C_xN1 \Rightarrow (d_{C_{x-1}N2} \wedge \overline{d_{CxN2}}) \tag{25}$$

$$\begin{aligned} V_{C_xP} \Leftrightarrow\ & (\overline{C_xP3} \vee (\overline{C_{x-1}P2} \wedge V_{C_{x-1}P} \wedge d_{C_{x-1}P2}) \\ & \vee (\overline{C_{x+1}P2} \wedge V_{C_{x+1}P} \wedge \overline{d_{C_xP2}})) \end{aligned} \tag{26}$$

$$\begin{aligned} V_{CxO} \Leftrightarrow\ & ((\overline{C_xP1} \wedge V_{C_xP}) \vee (C_xN1 \wedge V_{C_xN}) \\ & \vee (C_{x-1}NH \wedge V_{C_{x-1}O} \wedge d_{C_{x-1}NH}) \\ & \vee (C_xNH \wedge V_{C_{x+1}O} \wedge \overline{d_{C_xNH}})) \end{aligned} \tag{27}$$



(a) PMOS-driving subsection

(b) NMOS-driving subsection

Figure 10: Detailed TRAP interface for a PMOS/NMOS pair

$$V_{CxN} \Leftrightarrow (C_xN3 \vee (C_{x-1}N2 \wedge V_{C_{x-1}N} \wedge d_{C_{x-1}N2}) \vee (C_{x+1}N2 \wedge V_{C_{x+1}N} \wedge \overline{d_{C_xN2}})) \tag{28}$$

$$(\overline{C_xP2} \wedge d_{C_xP2}) \Rightarrow (cnt_{C_xP} < cnt_{C_{x+1}P}) \tag{29}$$

$$(\overline{C_xP2} \wedge \overline{d_{C_xP2}}) \Rightarrow (cnt_{C_{x+1}P} < cnt_{C_xP}) \tag{30}$$

$$\overline{C_xP1} \Rightarrow (cnt_{C_xP} < cnt_{C_xO}) \tag{31}$$

$$(C_xNH \wedge d_{C_xNH}) \Rightarrow (cnt_{C_xO} < cnt_{C_{x+1}O}) \tag{32}$$

$$(C_xNH \wedge \overline{d_{C_xNH}}) \Rightarrow (cnt_{C_{x+1}O} < cnt_{C_xO}) \tag{33}$$

$$C_xNV \Rightarrow (cnt_{C_xO} < cnt_{C_xL2Z}) \tag{34}$$

$$C_xN1 \Rightarrow (cnt_{C_xN} < cnt_{C_xO}) \tag{35}$$

$$(C_xN2 \wedge d_{C_xN2}) \Rightarrow (cnt_{C_xN} < cnt_{C_{x+1}N}) \tag{36}$$

$$(C_xN2 \wedge d_{C_xN2}) \Rightarrow (cnt_{C_{x+1}N} < cnt_{C_xN}) \tag{37}$$

For TA columns along the edge of a TRAP fabric, additional clauses for boundary conditions must be asserted to prevent unconstrained nets from compromising faithful functionality of the model. In instances where the TA column model clauses reference C_0 (when $x = 1$) or C_4 (when $x = 3$), C_0 and C_4 represent the rightmost column of the adjacent left unit and the leftmost column of the adjacent right unit in the TRAP fabric, respectively. If no unit is present to the left, then the missing transistors $C_{x-1}P2$, $C_{x-1}NH$, and $C_{x-1}N2$ are turned OFF using Equation (38). If no unit is present to the right, then the transistors C_3P2, C_3NH, and C_3N2 are turned OFF by Equation (39).

$$C_{x-1}P2 \wedge \overline{C_{x-1}NH} \wedge \overline{C_{x-1}N2} \tag{38}$$

$$C_3P2 \wedge \overline{C_3NH} \wedge \overline{C_3N2} \tag{39}$$

6.2 Interface Model

The predetermined signal flow of the TRAP interface, which is shown in Figure 10, eliminates the need for direction variables and simplifies evaluation of electrical drive of internal interface nodes. Modeling the interface uses principles for both logic gates (Section 4.1) and transistors (Section 4.2). Since the interface for each complementary transistor pair Py, Ny in a TA column C_x is identical, the equations describing the interface can be generalized for any transistor $y \in [1, 3]$ in any column $x \in [1, 3]$ of the TA. The complete propositional model is defined in Equations (40) to (52), where programmable values $M1$, $M2$, $M3$, $M4$, $M5$, $M6$, and $M7$ determine the interface configuration.

$$M1C_xPy \Rightarrow (C_xL1y \Leftrightarrow \overline{C_xL2y}) \tag{40}$$

$$\overline{M1C_xPy} \Rightarrow (C_xL1y \Leftrightarrow C_xL2y) \tag{41}$$

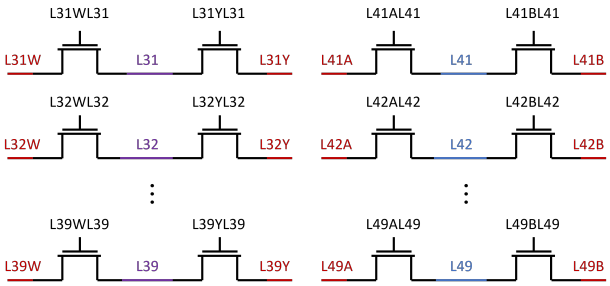


Figure 11: Outgoing interconnect to neighboring TRAP units above, below, left, and right

$$M2C_xPy \Rightarrow (C_xPy \Leftrightarrow C_xL1y) \tag{42}$$
$$\overline{M2C_xPy} \Rightarrow (C_xPy \Leftrightarrow \overline{M3C_xPy}) \tag{43}$$
$$\overline{V_{C_xL2y}} \Rightarrow \overline{M2C_xPy} \tag{44}$$
$$M4C_xNy \Rightarrow (C_xNy \Leftrightarrow C_xL11) \tag{45}$$
$$M5C_xNy \Rightarrow (C_xNy \Leftrightarrow C_xL12) \tag{46}$$
$$M6C_xNy \Rightarrow (C_xNy \Leftrightarrow C_xL13) \tag{47}$$
$$\overline{(M4C_xNy} \wedge \overline{M5C_xNy} \wedge \overline{M6C_xNy)} \Rightarrow (C_xNy \Leftrightarrow M7C_xNY) \tag{48}$$
$$M4C_xNy \Rightarrow \overline{M5C_xNy \vee M6C_xNy} \tag{49}$$
$$M5C_xNy \Rightarrow \overline{M4C_xNy \vee M6C_xNy} \tag{50}$$
$$M6C_xNy \Rightarrow \overline{M4C_xNy \vee M5C_xNy} \tag{51}$$
$$\overline{V_{C_xL2y}} \Rightarrow \overline{(M4C_xNy \vee M5C_xNy \vee M6C_xNy)} \tag{52}$$

Electrical drive variables are not required to ensure that the isolated interface model properly emulates circuitry, but they must be considered when the interface model is combined with the TA model. The TA model assumes that all incoming signals are driven, so the interconnect model must guarantee that signals it feeds to the TA are driven. Equations (43), (44), (48) and (52) are written such that if an interface input is not driven, then the interface delivers a driven constant $M3C_xNy$ or $M7C_xNy$ instead.

6.3 Interconnect Model

The TRAP interconnect is a signal routing network composed of metal tracks selectively connected by transistor switches. Hence, similar to modeling the TA, modeling the interconnect in propositional logic requires the principles previously discussed in Sections 4.2 and 4.3. Unlike the TA model though, simplifications to the interconnect model based on implicit signal direction cannot be established by structural inspection due to the absence of power rails. All transistor gate terminals are driven by programmable values. For the connections present in the interconnect shown in Figure 8, logic signal causality for transistors connecting L4 and L3 wires, L3 and L2 wires, or L4 and L2 wires is generally expressed by Equation (53), Equation (54), and Equations (55) and (56), respectively, for $j \in [1, 9]$, $k \in [1, 9]$. For L4 and L3 connections to neighboring TRAP units above ($L4A$), below ($L4B$), left ($L3W$), and right ($L3Y$) of a given TRAP unit, as shown in Figure 11, logic causality is described by Equations (57) to (60).

$$L3jL4k \Rightarrow (L3j \Leftrightarrow L4k) \tag{53}$$
$$L3jC_xL2y \Rightarrow (L3j \Leftrightarrow C_xL2y) \tag{54}$$

$$L4kC_xL2y \Rightarrow (L4k \Leftrightarrow C_xL2y) \tag{55}$$
$$L4kC_xL2y \Rightarrow (L4k \Leftrightarrow C_xL2Z) \tag{56}$$
$$L3jWL3k \Rightarrow (L3jW \Leftrightarrow L3j) \tag{57}$$
$$L3jYL3k \Rightarrow (L3j \Leftrightarrow L3jY) \tag{58}$$
$$L4kAL4k \Rightarrow (L4kA \Leftrightarrow L4k) \tag{59}$$
$$L4kBL4k \Rightarrow (L4k \Leftrightarrow L4kB) \tag{60}$$

Determining electrical drive within the entire interconnect structure follows the same rules as within the TA, with one altered consideration: rather than connecting directly to power rails, interconnect L3 and L4 nets may receive externally-driven inputs, so all clauses resemble Equation (7). L4 tracks connect to three L3 tracks, two L2 tracks, and the L4 tracks above and below, resulting in the disjunction Equation (61). L3 tracks connect to three L4 tracks, three L2 tracks, and the L3 tracks to the left and right, resulting in the disjunction Equation (62). L2 (not including L2Z) wires are not candidates for receiving external inputs, and connect to two L3 tracks and one L4 track, so their electrical validity is determined by Equation (63). L2Z tracks also cannot receive external inputs, and lead to L4 and L3 wires, so their electrical validity is determined by Equation (64).

$$\begin{aligned} V_{L4k} \Leftrightarrow (&isIn_{L4k} \vee (L4kAL4k \wedge \overline{d_{L4kAL4k}} \wedge V_{L4kA}) \\ &\vee (L4kBL4k \wedge d_{L4kBL4k} \wedge V_{L4kB}) \\ &\vee (L3j_1L4k \wedge d_{L3j_1L4k} \wedge V_{L3j_1}) \\ &\vee (L3j_2L4k \wedge d_{L3j_2L4k} \wedge V_{L3j_2}) \\ &\vee (L3j_3L4k \wedge d_{L3j_3L4k} \wedge V_{L3j_3}) \\ &\vee (L4kC_xL2y \wedge \overline{d_{L4kC_xL2y}} \wedge V_{C_xL2y}) \\ &\vee (L4kC_xL2Z \wedge \overline{d_{L4kC_xL2Z}} \wedge V_{C_xL2Z})) \end{aligned} \tag{61}$$

$$\begin{aligned} V_{L3j} \Leftrightarrow (&isIn_{L3j} \vee (L3kYL3k \wedge \overline{d_{L3kYL3k}} \wedge V_{L3kY}) \\ &\vee (L3kWL3k \wedge d_{L3kWL3k} \wedge V_{L3kW}) \\ &\vee (L3jL4k_1 \wedge \overline{d_{L3j_1L4k}} \wedge V_{L3j_1}) \\ &\vee (L3jL4k_2 \wedge \overline{d_{L3j_2L4k}} \wedge V_{L3j_2}) \\ &\vee (L3jL4k_3 \wedge \overline{d_{L3j_3L4k}} \wedge V_{L3j_3}) \\ &\vee (L3jC_{x1}L2y \wedge \overline{d_{L3jC_{x1}L2y}} \wedge V_{C_{x1}L2y}) \\ &\vee (L3jC_{x2}L2y \wedge \overline{d_{L3jC_{x2}L2y}} \wedge V_{C_{x2}L2y}) \\ &\vee (L3jC_{x3}L2Z \wedge \overline{d_{L3jC_{x3}L2Z}} \wedge V_{C_{x3}L2Z})) \end{aligned} \tag{62}$$

$$\begin{aligned} V_{C_xL2y} \Leftrightarrow (&L3j_1C_xL2y \wedge d_{L3j_1C_xL2y} \wedge V_{L3j_1}) \\ &\vee (L3j_2C_xL2y \wedge d_{L3j_2C_xL2y} \wedge V_{L3j_2}) \\ &\vee (L4kC_xL2y \wedge d_{L4kC_xL2y} \wedge V_{L4k}) \end{aligned} \tag{63}$$

$$\begin{aligned} V_{C_xL2Z} \Leftrightarrow (&L3j_1C_xL2Z \wedge d_{L3j_1C_xL2Z} \wedge V_{L3j_1}) \\ &\vee (L3j_2C_xL2Z \wedge d_{L3j_2C_xL2Z} \wedge V_{L3j_2}) \\ &\vee (L3j_3C_xL2Z \wedge d_{L3j_3C_xL2Z} \wedge V_{L3j_3}) \\ &\vee (L4k_1C_xL2Z \wedge d_{L4k_1C_xL2Z} \wedge V_{L4k_1}) \\ &\vee (L4k_2C_xL2Z \wedge d_{L4k_2C_xL2Z} \wedge V_{L4k_2}) \\ &\vee (L4k_3C_xL2Z \wedge d_{L4k_3C_xL2Z} \wedge V_{L4k_3}) \end{aligned} \tag{64}$$

To prevent illegitimate loops from forming within the interconnect, such as in Example 1, path count values are assessed at every connection between an L4 and an L3 wire, an L3

and an L2 wire, or an L4 and an L2 wire, as shown in Equations (65) to (74). Two clauses per transistor are used in the same fashion as Equations (12) to (15).

$$(L3jL4k \wedge d_{L3jL4k}) \Rightarrow (cnt_{L3j} < cnt_{L4k}) \tag{65}$$
$$(L3jL4k \wedge \overline{d_{L3jL4k}}) \Rightarrow (cnt_{L4k} < cnt_{L3j}) \tag{66}$$
$$(L3jC_xL2y \wedge d_{L3jC_xL2y}) \Rightarrow (cnt_{L3j} < cnt_{C_xL2y}) \tag{67}$$
$$(L3jC_xL2y \wedge \overline{d_{L3jC_xL2y}}) \Rightarrow (cnt_{C_xL2y} < cnt_{L3j}) \tag{68}$$
$$(L3jC_xL2Z \wedge d_{L3jC_xL2Z}) \Rightarrow (cnt_{L3j} < cnt_{C_xL2Z}) \tag{69}$$
$$(L3jC_xL2Z \wedge \overline{d_{L3jC_xL2Z}}) \Rightarrow (cnt_{C_xL2Z} < cnt_{L3j}) \tag{70}$$
$$(L4kC_xL2y \wedge d_{L4kC_xL2y}) \Rightarrow (cnt_{L4k} < cnt_{C_xL2y}) \tag{71}$$
$$(L4kC_xL2y \wedge \overline{d_{L4kC_xL2y}}) \Rightarrow (cnt_{C_xL2y} < cnt_{L4k}) \tag{72}$$
$$(L4kC_xL2Z \wedge d_{L4kC_xL2Z}) \Rightarrow (cnt_{L4k} < cnt_{C_xL2Z}) \tag{73}$$
$$(L4kC_xL2Z \wedge \overline{d_{L4kC_xL2Z}}) \Rightarrow (cnt_{C_xL2Z} < cnt_{L4k}) \tag{74}$$

Boundary conditions constitute the final part of modeling the interconnect. Floating nodes occur along the TRAP model perimeter when the interconnect model clauses point to hypothetical L3 and L4 nets that are not actually present. For TRAP units without at least one neighboring unit in any direction, an additional clause may be written to forcefully turn all unconnected boundary transistors OFF and prevent floating nodes from impacting model behavior. The most extensive example of this clause is shown in Equation (75), which is the case where a TRAP unit has no neighbors in any direction.

$$\overline{L31WL31} \wedge \cdots \wedge \overline{L31YL31} \wedge \cdots \wedge \overline{L41AL41} \wedge \cdots \wedge \overline{L41BL41} \wedge \cdots \wedge \overline{L49BL49} \tag{75}$$

6.4 Complete TRAP Unit Model

Combining the TA, interface, and interconnect models described in Sections 6.1 to 6.3 produces a propositional logic model for a singular TRAP unit. However, one last facet remains before the model fully embodies TRAP functionality. The path count variables present within the interconnect and the TA effectively inhibit self-validating loops within their own respective sections, but when combined with the interface, a cyclical topology can be formed through the entire TRAP unit. This loop is unintentionally allowed because the path count variables of the interconnect are not conveyed through the interface to the transistor array path count clauses. This break in the path count variable chain allows a signal's count to reset when traveling through the interface. This creates a satisfiable condition where a signal with no legitimate electrical origin can drive itself from the interconnect to the interface, through the TA, and back up to the interconnect. Preventing this loop requires that the path count variables of the interconnect are communicated to the TA path count clauses, as facilitated by Equations (76) to (91). These clauses complete the TRAP unit model, which is fully characterized by Equations (16) to (91). Multiple copies of this unit model, each with unique identifiers, can be combined to represent a grid of interlinked units that compose a TRAP fabric.

$$M2C_xP3 \Rightarrow (cnt_{C_xL23} < cnt_{C_xP}) \tag{76}$$
$$(M2C_xP2 \wedge d_{C_xP2}) \Rightarrow (cnt_{C_xL22} < cnt_{C_{x+1}P}) \tag{77}$$
$$(M2C_xP2 \wedge \overline{d_{C_xP2}}) \Rightarrow (cnt_{C_xL22} < cnt_{C_xP}) \tag{78}$$
$$M2C_xP1 \Rightarrow (cnt_{C_xL21} < cnt_{C_xO}) \tag{79}$$
$$M4C_xN1 \Rightarrow (cnt_{C_xL21} < cnt_{C_xO}) \tag{80}$$
$$M5C_xN1 \Rightarrow (cnt_{C_xL22} < cnt_{C_xO}) \tag{81}$$

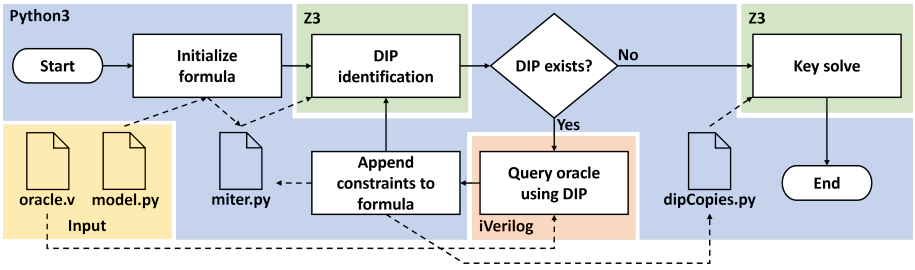


Figure 12: Flow diagram for our SAT attack tool-set

$$M6C_xN1 \Rightarrow (cnt_{C_xL23} < cnt_{C_xO}) \tag{82}$$
$$(M4C_xN2 \wedge d_{C_xN2}) \Rightarrow (cnt_{C_xL21} < cnt_{C_{x+1}N}) \tag{83}$$
$$(M5C_xN2 \wedge d_{C_xN2}) \Rightarrow (cnt_{C_xL22} < cnt_{C_{x+1}N}) \tag{84}$$
$$(M6C_xN2 \wedge d_{C_xN2}) \Rightarrow (cnt_{C_xL23} < cnt_{C_{x+1}N}) \tag{85}$$
$$(M4C_xN2 \wedge \overline{d_{C_xN2}}) \Rightarrow (cnt_{C_xL21} < cnt_{C_xN}) \tag{86}$$
$$(M5C_xN2 \wedge \overline{d_{C_xN2}}) \Rightarrow (cnt_{C_xL22} < cnt_{C_xN}) \tag{87}$$
$$(M6C_xN2 \wedge \overline{d_{C_xN2}}) \Rightarrow (cnt_{C_xL23} < cnt_{C_xN}) \tag{88}$$
$$M4C_xN3 \Rightarrow (cnt_{C_xL21} < cnt_{C_xN}) \tag{89}$$
$$M5C_xN3 \Rightarrow (cnt_{C_xL22} < cnt_{C_xN}) \tag{90}$$
$$M6C_xN3 \Rightarrow (cnt_{C_xL23} < cnt_{C_xN}) \tag{91}$$

7 SAT Attack Implementation

In order to launch a SAT attack against the TRAP model developed in Section 6, we developed and are releasing a tool-set[3] that constitutes the first known embodiment of a SAT attack for transistor-level digital circuits. The flow of our tool-set, shown in Figure 12, follows the generic SAT attack flow outlined in Figure 3. However, unlike the latter, it can handle switch-level models that express tri-state signals, as discussed in Section 4.4.

Our attack tool is facilitated by (i) Python 3 [VRD09], in which miter formation, miter expansion, key extraction circuit formation and overarching management of the SAT attack are scripted, (ii) Microsoft's Z3 satisfiability modulo theories (SMT) solver [dMB08, BdMNW19], which is called for SAT solving and formula pre-processing tactics, and (iii) the Icarus Verilog (iVerilog) simulator [Wil24], which queries oracle netlists to obtain correct circuit responses.

We note that the extensive configurability of transistor-level protected circuits may force the SAT attack to identify as many DIPs as there are unique input patterns. At this point, if all input patterns have been found to be DIPs, the next DIP identification step is guaranteed to return an unsatisfiable decision. To accelerate run-time, our tool skips the DIP identification step in this scenario (*i.e.,* it will not call the SAT solver) and assumes an unsatisfied decision so that the attack can proceed directly to the key retrieval task.

8 Experimental Results

To evaluate the resilience of TRAP to SAT attacks, we staged a series of SAT attacks against the TRAP fabric model outlined in Section 6 using the tool-set described in Section 7. These attacks target increasingly more complex versions of TRAP, starting with

[3]See link: https://github.com/aric-fowler/TRANSAT.git

Table 1: Results for SAT attacks carried out on TRAP models

Circuit		# Inputs	# Outputs	# Key Inputs	CNF Model		Attack	
					# Variables	# Clauses	Time to Break (h:mm:ss)	Break Round of Rounds Possible
TA	NAND2	2	1	20	125	324	0:00:01	5 / 5
	NOR3	3	1	18	126	337	0:00:02	5 / 9
	AOI22	4	1	16	127	341	0:00:01	4 / 17
TA+Ifc	NAND2	2	1	69	296	834	0:00:04	4 / 5
	NOR3	3	1	69	297	836	0:00:09	8 / 9
	AOI22	4	1	69	298	841	0:00:22	13 / 17
TRAP Unit	NAND2	2	1	177	549	2954	0:00:24	5 / 5
	NOR3	3	1	177	550	2956	0:04:02	9 / 9
	AOI22	4	1	177	551	2953	42:38:42	17 / 17
2 × 2 Fabric	NAND2	2	1	708	1935	11607	0:07:22	5 / 5
	NOR3	3	1	708	1936	11601	10:28:31	9 / 9
	AOI22	4	1	708	1937	11603	DNF†	DNF† / 17
	C17	5	2	708	1939	11607	DNF‡	DNF‡ / 33

† Terminated after 189 days, 5 hours, with 13 DIPs discovered
‡ Terminated after 99 days, 21 hours, with 11 DIPs discovered

a standalone TA and gradually adding the interface and the interconnect to complete a unit, before proceeding to a 2×2 (*i.e.*, two-row, two-column) fabric. In each case, three basic logic gates with increasing number of inputs (NAND2, NOR3, and AOI22) are used as the target function. In the case of the 2×2 fabric, we also attack C17, a small combinational benchmark circuit [BF85] with 5 inputs and 2 outputs. Table 1 summarizes our results and reports model size[4], attack runtime, and the number of attack rounds completed before extracting a valid key, along with the number of possible rounds ($2^m + 1$, where m is the number of oracle inputs). All attacks were single-threaded processes executed on Linux-based Dell R630 servers, equipped with Xeon E5-2660 processors operating at 2.6 GHz. Each attack was repeated ten times and we report the shortest run-time result, noting that no statistically significant variance was observed across these iterations.

Standalone TA: The smallest part of TRAP that can implement a logic function is the TA, for which a model was derived in Section 6.1. To attack the TA, we pre-positioned the inputs and outputs on the TA, such that at least one configuration implementing the target functionality was possible. The absence of an interface necessitates this. All unused transistor gate terminals were designated as key inputs. Because of this, the attack for NAND2 functionality has more keys than the attacks for NOR3 and AOI22 functionality, since NAND2 has fewer inputs. Our transistor-level SAT attack tool was able to easily retrieve the key for each of these three functions in a similar number of rounds.

Combined TA and Interface: Next, we increased the target circuit complexity by adding an interface, for which a model was derived in Section 6.2, to the TA. The absence of an interconnect necessitates that inputs are pre-positioned on the interconnect to allow for at least one correct implementation. In this case, however, key bits were not sacrificed due to an increase in the number of inputs because the interface allocates separate nets for inputs and keys. Compared to the attacks on the standalone TA, results of the attack on the combined TA and interface show a small increase in attack time. This is justified by the added functionality introduced by the interface, which increases the complexity of the propositional logic model. Based on the results, the adverse impact of including the interface is more pronounced as the number of inputs and complexity of the target functionality increases. Indeed, both runtime and number of rounds required to break the combined TA and interface rises as we step up from the NAND2 to the NOR3 and then from the NOR3 to the AOI22.

Complete TRAP Unit: Adding an interconnect network, for which a model was derived in Section 6.3, to the combined TA and interface model, and taking into consideration the

[4]While model size is weakly correlated to SAT hardness [Bj9], we note it to serve as a proxy to a true complexity metric. Converting a model containing integer variables to CNF decomposes its integer elements into binary equivalents, making models with different ranges of integers comparable.

caveats described in Section 6.4, completes an entire TRAP unit. At this point, attacks for the three functions start to exhibit great disparity in runtime. A key for NAND2 functionality on a TRAP unit can be retrieved in twenty-four seconds, while for NOR3 it takes four minutes and for AOI22 it takes over forty hours. We also observe that while solving for each of these functionalities, every single input pattern must be discovered as a DIP. This is explained by the fact that the TRAP unit can implement a very large set of different functions and is consistent with the difficulty SAT attack faces when launched against highly configurable fabrics at any level [BMT+23].
2 × 2 TRAP Fabric: Extending our model to even a small 2 × 2 array of TRAP fabric, which can implement more complex functionality, results in a significant increase in SAT attack runtime for the three target functions. In fact, after running continuously for six and three months, respectively, the attacks on the AOI22 function and the C17 benchmark (which is composed of six NAND2 gates), did not return a solution and were forcefully terminated. Similar to the TRAP unit, for the two functions that completed on the 2 × 2 fabric, all input pattern combinations were visited as DIPs and the addition of an extra input (from NAND2 to NOR3) resulted in a significant increase in runtime.

Collectively, our results on TRAP corroborate the following:

1. Increased model complexity arising from transistor-level circuit implementation produces lengthier formulas that require longer solver runtime per round, resulting in increased overall attack runtime.

2. Given a fixed-complexity model, capable of instating a variety of functions with variable input cardinality, SAT attacks seeking to break functions with a larger number of inputs require more DIP identification rounds, resulting in increased overall attack runtime.

3. Owing to the model expansion resulting from its transistor-level granularity, as well as its extensive space of implementable functions, TRAP is resistant to SAT attacks for all but the simplest functions.

9 Discussion

While the propositional logic model that we developed accounts correctly and intuitively for the circuit intricacies of the TRAP fabric (*i.e.*, signal directionality, tri-state logic and cyclical structures), it is relatively large and complex. Nevertheless, we posit that the difficulty to compromise a circuit redacted through TRAP does not stem from the complexity of the modeling effort but rather from the inherent complexity of the fabric. In other words, while it may be possible to develop alternative, functionally equivalent models for TRAP, it is unlikely that such models will materially reduce the runtime of attacks. To support the position that TRAP is resistant to SAT attacks independent of fabric expression, we provide the following three observations:

1. As documented in the recent literature [MAS+21, BMT+23], SAT attacks have been unable to retrieve the programming key for reinstating the missing logic of redacted circuits, **even when the redaction is performed with logic-level programmable fabrics**. This is in sharp contrast to the results of SAT attacks on logic-locked versions of these circuits, which can be broken in a relatively practical amount of time. The reason behind the SAT attack's difficulty to solve redacted circuits has to do with the very large number of unique functions that these circuits can be programmed to implement, as opposed to the limited number of functions that logic-locked circuits can implement for the possible key values. The large number of unique functions that a programmable fabric can instantiate, along with the

concomitant complexity of the CNF model required to accurately express a fabric's capability to realize these functions, presents an obstacle that SAT attacks are unable to surmount for practical circuits.

2. As a transistor-level programmable fabric, TRAP introduces additional challenges to SAT attack, over and above what logic-level programmable fabrics present:

   (a) In a logic-level programmable fabric, a signal can be in two different states (low or high), which can be fully described through a single Boolean variable. In contrast, the sea-of-transistors architecture of TRAP implies that a signal can be in one of five different states (low driving in one direction, low driving in the opposite direction, high driving in one direction, high driving in the opposite direction, or electrically disconnected). Therefore, at least three Boolean variables are required to express the state of a signal. Our formulation has exactly three variables, the logic value, the direction, and the electrical drive, hence **none of these variables are redundant.**

   (b) In a logic-level programmable fabric, a hierarchical approach can be used to decompose the complete Boolean functionality into logic functions which are implemented by well-defined sub-modules of the fabric (*e.g.,* LUTs in an FPGA). Hierarchical modeling assists SAT attacks as the number of local choices is contained at the boundary of each sub-module. In contrast, owing to its sea-of-transistors architecture, **TRAP cannot be modeled hierarchically** because there is no pre-defined logic-level module boundary. Any transistor in TRAP can be combined with any other transistor anywhere else on TRAP to form logic functions. Therefore, it is not possible to group transistors together and reduce the number of variables required to express their functionality. The only option is to model the logic-level capabilities of the entire TRAP fabric as a giant LUT. This is not only specific to a given TRAP fabric size, but also impractical to express explicitly and use in an attack.

3. Despite the elaborate capabilities that it must encompass and the inherent lack of logic-level modularity, **the proposed model scales linearly with fabric size, attesting to its compactness**. Also, when used in a SAT attack, it elicits a faithful response (*i.e.,* produced bitstreams are directly loadable onto the fabric). While alternative models may contain different representations of TRAP, the extensive capabilities of the fabric and the vast space of options for composing functions out of transistors make it unlikely that an alternative model could be materially more compact. Also, while alternative CNF formulations may lead to different SAT-solving runtimes [Ala10], contemporary solvers, such as Z3, pre-process a given formulation and generate their own internal representation on which they perform optimization heuristics before solving, thereby ameliorating the impact of modeling inefficiency.

Based on these observations and considering the NP-complete nature of the problem at hand, we conjecture that any improvement in the formulation of TRAP in CNF cannot make a material difference with respect to TRAP's resistance to SAT-attacks.

10 Conclusion

Despite their extensive success in compromising IP protection solutions for gate-level circuits, conventional SAT attack methods are not equipped to handle transistor-level programmable fabrics, such as TRAP. Concepts such as signal directionality, tri-state logic, and cyclical structures, which are crucial for correctly expressing switch-level circuits, are not natively reflected in the formulation used in logic-level SAT attacks. Nevertheless,

additional modeling provisions that account for signal direction and electrical drive (Sections 4.2 and 4.3), along with an expanded signal equivalence definition that accounts for high-impedance outputs (Section 4.4), can be introduced to make transistor-level granularity compatible with SAT attacks. These solutions have been incorporated into a generalized SAT attack tool-set, which in turn has been used to evaluate TRAP fabrics for susceptibility to SAT attacks. Results from SAT attacks conducted on this fabric provide insight regarding the complexity of transistor-level SAT attacks and elicit the conclusion that TRAP is resilient to SAT attacks in all but the most trivial cases. This latter finding supports the conjecture that a TRAP fabric will not reveal any reasonably-sized IP programmed onto it, making it a prime protection topology for any practical IC redaction application. Future modeling efforts will seek to extend applicability of recently-developed sequential SAT attack methods [KCV19, SLPJ19, HZY+21] to include circuits with transistor-level implementations of sequential digital circuits.

References

[AAASM19] Sahel Alouneh, Sa'ed Abed, Mohammad H. Al Shayeji, and Raed Mesleh. A Comprehensive Study and Analysis on SAT-Solvers: Advances, Usages and Achievements. *Artificial Intellifence Review*, 38(2):199–207, 2019.

[Ala10] Alan M. Frisch and Paul A. Giannaros. SAT Encodings of the At-Most-k Constraint Some Old , Some New , Some Fast , Some Slow. In *10th International Workshop of Constraint Modelling and Reformulation*, 2010.

[APMP23] Zain Ul Abideen, Tiago Diadami Perez, Mayler Martins, and Samuel Pagliarini. A Security-Aware and LUT-Based CAD Flow for the Physical Synthesis of hASICs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(10):3157–3170, 2023.

[AYS+19] Lilas Alrahis, Muhammad Yasin, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. ScanSAT: Unlocking Obfuscated Scan Chains. *IEEE/ACM Asia and South Pacific Design Automation Conference (DAC)*, pages 352–357, 2019.

[BCCW12] J.P. Baukus, L.W. Chow, R.P. Cocchi, and B.J. Wang. Method and Apparatus for Camouflaging a Standard Cell based Integrated Circuit with Micro Circuits and Post Processing. *US Patent no. 20120139582*, 2012.

[BdMNW19] Nikolaj Bjørner, Leonardo de Moura, Lev Nachmanson, and Christoph Wintersteiger. Programming Z3. https://theory.stanford.edu/~nikolaj/programmingz3.html, 2019.

[Ber12] Berkeley Logic Synthesis and Verification Group. ABC: A System for Sequential Synthesis and Verification, 2012.

[BF85] Franc Brglez and Hideo Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Targeted Translator in FORTRAN. In *IEEE International Symposium on Circuits and Systems (ISCAS), Special Session on Recent Algorithms for Gate-Level ATPG with Fault Simulation and Their Performance Assessment*, 1985.

[Bj9] Magnus Björk. Successful SAT Encoding Techniques. *Journal on Satisfiability, Boolean Modeling and Computation*, 7:189–201, Jul 2009.

[BM07] Aaron R. Bradley and Zohar Manna. *The Calculus of Computation*, chapter 1, pages 3–32. Springer Berlin, Heidelberg, Publication Location, 2007.

[BMT+23] Jitendra Bhandari, Abdul Khader Thalakkattu Moosa, Benjamin Tan, Christian Pilato, Ganesh Gore, Xifan Tang, Scott Temple, Pierre-Emmanuel Gaillardon, and Ramesh Karri. Not All Fabrics Are Created Equal: Exploring eFPGA Parameters for IP Redaction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 31(10):1459–1471, 2023.

[BTMT+21] Jitendra Bhandari, Abdul Khader Thalakkattu Moosa, Benjamin Tan, Christian Pilato, Ganesh Gore, Xifan Tang, Scott Temple, PierreEmmanuel Gaillardon, and Ramesh Karri. Exploring eFPGA-based Redaction for IP Protection. *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021.

[BTZ10a] Alex Baumgarten, Akhilesh Tyagi, and Joseph Zambreno. Preventing IC Piracy Using Reconfigurable Logic Barriers. *IEEE Design & Test of Computers*, 27(1):66–75, 2010.

[BTZ10b] Alex Baumgarten, Akhilesh Tyagi, and Joseph Zambreno. Preventing IC Piracy Using Reconfigurable Logic Barriers. *IEEE Design & Test of Computers*, 27(1):66–75, 2010.

[CB09] Rajat Subhra Chakraborty and Swarup Bhunia. HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(10):1493–1502, 2009.

[CB14] Brice Colombier and Lilian Bossuet. Survey of Hardware Protection of Design Data for Integrated Circuits and Intellectual Properties. *IET Computers & Digital Techniques*, 8(6):274–287, 2014.

[CBCW14] Ronald P. Cocchi, James P. Baukus, Lap Wai Chow, and Bryan J. Wang. Circuit Camouflage Integration for Hardware IP Protection. In *ACM/IEEE Design Automation Conference (DAC)*, pages 1–5, 2014.

[dMB08] Leonardo de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, pages 337–340, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[DP60] Martin Davis and Hilary Putnam. A Computing Procedure for Quantification Theory. *J. ACM*, 7(3):201–215, Jul 1960.

[EMGT19] Mohamed El Massad, Siddharth Garg, and Mahesh V Tripunitara. The SAT Attack on IC Camouflaging: Impact and Potential Countermeasures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1577–1590, 2019.

[ES04] Niklas Eén and Niklas Sörensson. An Extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *Theory and Applications of Satisfiability Testing*, pages 502–518, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[GRK+23] Rui Guo, M Sazadur Rahman, Hadi M Kamali, Fahim Rahman, Farimah Farahmandi, and Mark Tehranipoor. EvoLUTe: Evaluation of Look-Up-Table-Based Fine-Grained IP Redaction. In *Design, Automation & Test in Europe Conference (DATE)*, pages 1–6, 2023.

[GV21] Vijay Ganesh and Moshe Y. Vardi. *On the Unreasonable Effectiveness of SAT Solvers*, page 547–566. Cambridge University Press, 2021.

[HZY+21] Yinghua Hu, Yuke Zhang, Kaixin Yang, Dake Chen, Peter A. Beerel, and Pierluigi Nuzzo. Fun-SAT: Functional Corruptibility-Guided SAT-Based Attack on Sequential Logic Encryption. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 281–291, 2021.

[IC 21] IC Insights. McClean Report. https://www.icinsights.com/services/mcclean-report/, 2021. Last accessed on 10/04/2022.

[IEGT13] Frank Imeson, Ariq Emtenan, Siddharth Garg, and Mahesh Tripunitara. Securing Computer Hardware Using 3D Integrated Circuit (IC) Technology and Split Manufacturing for Obfuscation. *USENIX Security Symposium*, pages 495–510, 2013.

[JHCK20] Melbin John, Aadil Hoda, Ramanuj Chouksey, and Chandan Karfa. SAT Based Partial Attack on Compound Logic Locking. In *Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6, 2020.

[JM07] R.W. Jarvis and M.G. McIntyre. Split Manufacturing Method for Advanced Semiconductor Circuits. *US Patent no. 7,195,931*, 2007.

[KBK24] Praveen Karmakar, Marpina Bharani, and Chandan Karfa. Evaluating the Robustness of Large Scale eFPGA-based Hardware Redaction. In *37th International Conference on VLSI Design and 23rd International Conference on Embedded Systems (VLSID)*, pages 517–522, 2024.

[KCV19] Yasaswy Kasarabada, Suyuan Chen, and Ranga Vemuri. On SAT-Based Attacks On Encrypted Sequential Logic Circuits. In *International Symposium on Quality Electronic Design (ISQED)*, pages 204–211, 2019.

[LPS+19] Haocheng Li, Satwik Patnaik, Abhrajit Sengupta, Haoyu Yang, Johann Knechtel, Bei Yu, Evangeline FY Young, and Ozgur Sinanoglu. Attacking Split Manufacturing from a Deep Learning Perspective. *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2019.

[LPS21] Nimisha Limaye, Satwik Patnaik, and Ozgur Sinanoglu. Fa-SAT: Fault-Aided SAT-based Attack on Compound Logic Locking Techniques. In *Design, Automation & Test in Europe Conference (DATE)*, pages 1166–1171, 2021.

[MAS+21] Prashanth Mohan, Oguz Atli, Joseph Sweeney, Onur Kibar, Larry Pileggi, and Ken Mai. Hardware Redaction via Designer-Directed Fine-Grained eFPGA Insertion. *IEEE Design, Automation & Test in Europe Conference (DATE)*, pages 1186–1191, 2021.

[RKM08] Jarrod A. Roy, Farinaz Koushanfar, and Igor L. Markov. EPIC: Ending Piracy of Integrated Circuits. *IEEE/ACM Design, Automation & Test in Europe Conference (DATE)*, pages 1069–1074, 2008.

[RPSK12] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri. Security Analysis of Logic Obfuscation. *IEEE/ACM Design Automation Conference (DAC)*, pages 83–89, 2012.

[SLPJ19] Kaveh Shamsi, Meng Li, David Z. Pan, and Yier Jin. KC2: Key-Condition Crunching for Fast Sequential Circuit Deobfuscation. In *Design, Automation & Test in Europe Conference (DATE)*, pages 534–539, 2019.

[SPJ19] Kaveh Shamsi, David Z. Pan, and Yier Jin. IcySAT: Improved SAT-based Attacks on Cyclic Locked Circuits. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–7, 2019.

[SRM15] Pramod Subramanyan, Sayak Ray, and Sharad Malik. Evaluating the Security of Logic Encryption Algorithms. In *IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 137–143, 2015.

[SS19] Deepak Sirone and Pramod Subramanyan. Functional Analysis Attacks on Logic Locking. *IEEE Design, Automation & Test in Europe Conference (DATE)*, pages 936–939, 2019.

[STR+19] Mustafa M. Shihab, Jingxiang Tian, Gaurav Rajavendra Reddy, Bo Hu, William Swartz, Benjamin Carrion Schaefer, Carl Sechen, and Yiorgos Makris. Design Obfuscation Through Selective Post-Fabrication Transistor-Level Programming. In *Design, Automation & Test in Europe Conference (DATE)*, pages 528–533, 2019.

[tra19] Field Programmable Transistor Arrays, 2019. US Patent 10,511,308.

[tra20] Continuation of Application Field Programmable Transistor Arrays, 2020. US Patent 10,855,285.

[tra22] Continuation of Application Field Programmable Transistor Arrays, 2022. US Patent 11,362,362.

[TRW+17] Jingxiang Tian, Gaurav Rajavendra Reddy, Jiajia Wang, William Swartz, Yiorgos Makris, and Carl Sechen. A Field Programmable Transistor Array Featuring Single-Cycle Partial/Full Dynamic Reconfiguration. In *Design, Automation & Test in Europe Conference (DATE)*, pages 1336–1341, 2017.

[VDS+14] Kaushik Vaidyanathan, Bishnu P Das, Ekin Sumbul, Renzhi Liu, and Larry Pileggi. Building Trusted ICs Using Split Fabrication. In *IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 1–6, 2014.

[VRD09] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[VVGY21] Antonio Varas, Raj Varadarajan, Jimmy Goodrich, and Falan Yinug. Strenghtening The Global Semiconductor supply Chain In An Uncertain Era. Technical report, Boston Consulting Group and Semiconductor Industry Association, 2021.

[WCHR16] Yujie Wang, Pu Chen, Jiang Hu, and Jeyavijayan Rajendran. The Cat and Mouse in Split Manufacturing. *ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2016.

[WH05] Neil H.E. Weste and David Harris. *CMOS VLSI Design*, chapter 6. Addison-Wesley, 2005.

[Wil24] Stephen Williams. Icarus Verilog. https://steveicarus.github.io/iverilog/index.html, 2024. Last accessed on Feb 5 2024.

[WMMS21] Zi Wang, Shayan Omais Mohammed, Yiorgos Makris, and Benjamin Carrion Schafer. Functional Locking through Omission: From HLS to Obfuscated Design. In *IEEE International Conference on Computer Design (ICCD)*, pages 591–598, 2021.

[XS19] Yang Xie and Ankur Srivastava. Anti-SAT: Mitigating SAT Attack on Logic Locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(2):199–207, 2019.

[YSN+17] Muhammad Yasin, Abhrajit Sengupta, Mohammed Thari Nabeel, Mohammed Ashraf, Jeyavijayan Rajendran, and Ozgur Sinanoglu. Provably-Secure Logic Locking: From Theory To Practice. *ACM SIGSAC Conference on Computer & Communications Security*, pages 1601–1618, 2017.

[ZJK17] Hai Zhou, Ruifeng Jiang, and Shuyu Kong. CycSAT: SAT-Based attack On Cyclic Logic Encryptions. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 49–56, 2017.