Data Flow Obfuscation: A New Paradigm for Obfuscating Circuits

Kimia Zamiri Azar, Hadi Mardani Kamali, Shervin Roshanisefat, Houman Homayoun, Christos P. Sotiriou, and Avesta Sasan, *Member, IEEE*

***Abstract*— In this article, unlike almost all state-of-the-art obfuscation solutions that focus on functional/logic obfuscation, we introduce a new paradigm, called data flow obfuscation, which exploits the essence of asynchronicity. In data flow obfuscation, by benefiting from the handshaking mechanism of asynchronous circuits, the system's FFs/latches will operate out of sync. Hence, the adversary has no sufficient knowledge to apply unrolling/BMC. Also, due to the inherited asynchronicity, the exact time of writing/capturing data into/from the scan chain becomes hidden. Hence, the SAT attack cannot be applied even while scan chain access is open. Moreover, our new proposed paradigm creates stateful/oscillating combinational cycles into the design which extensively boosts the difficulty of modeling this technique. We also demonstrate how data flow obfuscation could easily be integrated with any circuit at low overhead while there is no limitation such as compromising test flow.**

***Index Terms*— Desynchronization, logic obfuscation.**

I. Introduction

THE ever-increasing cost of integrated circuits (ICs) manufacturing has forced many design houses to become fabless [1]. Outsourcing the stages of the manufacturing supply chain to the third-party facilities with no reliable monitoring on them results in emerging multiple forms of security threats such as IC overproduction, reverse engineering (RE), and intellectual property (IP) theft [2], [3]. To overcome these threats, logic obfuscation *a.k.a.* logic locking, as a proactive scheme adds post-manufacturing programming capability into the circuits [4], [5]. Logic obfuscation is the process of hiding the correct functionality of a circuit, during the stages at untrusted parties, when the programming value, referred to as the key, is unknown/incorrect. Only once the correct key is provided, the circuit behaves correctly, and the correct key would be initiated in its tamper-proof nonvolatile memory (tpNVM) after fabrication via a trusted party.

Manuscript received October 19, 2020; revised December 31, 2020; accepted January 31, 2021. Date of publication March 4, 2021; date of current version April 1, 2021. This work was supported in part by the National Science Foundation (NSF) under Award 1718434 and in part by the Semiconductor Research Corporation (SRC) TaskID 2772.001. *(Corresponding author: Hadi Mardani Kamali.)*

Kimia Zamiri Azar, Hadi Mardani Kamali, Shervin Roshanisefat, and Avesta Sasan are with the Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA 22030 USA (e-mail: kzamiria@gmu.edu; hmardani@gmu.edu; sroshani@gmu.edu; asasan@gmu.edu).

Houman Homayoun is with the Department of Electrical and Computer Engineering, University of California at Davis, Davis, CA 95616 USA (e-mail: hhomayoun@ucdavis.edu).

Christos P. Sotiriou is with the Department of Electrical and Computer Engineering, University of Thessaly, 38221 Volos, Greece (e-mail: csotiriou@gmail.com).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2021.3060345.

Digital Object Identifier 10.1109/TVLSI.2021.3060345

Due to the importance of logic obfuscation, many studies have evaluated the effectiveness of this countermeasure [6], [7]. Amongst all state-of-the-art threats on logic obfuscation, the Boolean satisfiability (SAT) attack has seriously challenged the effectiveness of the vast majority of existing logic obfuscation solutions [8]. In the SAT attack, as an oracle-guided attack, the adversary has access to (1) one successfully reverse-engineered yet locked netlist,[1] and (2) the activated/functional IC with open access to the scan chain. By getting inspiration from the miter circuit used in formal verification, in the SAT attack, a SAT solver is employed to iteratively find a specific set of inputs (one per each iteration), called discriminating input patterns (DIPs) that eliminate all incorrect keys leading to recovering the correct functionality of the circuit.

The main strength of the SAT attack comes from two important factors: 1) the pruning power of each DIP that can rule out a high portion of incorrect keys and 2) exploiting the scan chain access to apply the SAT attack on each combinational logic, separately. Hence, most studies on logic obfuscation could be categorized into four different groups all demonstrated in Table 1.

A. Motivation and Contributions

Table I provides a comprehensive comparison of state-of-the-art logic obfuscation techniques. As shown in Table I, a reliable logic obfuscation technique must be: 1) resilient against both combinational and sequential SAT attack and 2) added without compromising the test flow. To fulfill these requirements, in this article, we introduce a new logic obfuscation paradigm, called data flow obfuscation, whose main contributions are as follows.

1) In data flow obfuscation, getting inspired by asynchronous circuits, the data flow is asynchronously key-controlled. Since the sequential SAT attack unrolls the FF-to-FF (flip-flop) paths per each iteration (each iteration resembles each clock cycle), in data flow obfuscation, a small portion of the FF would be replaced with latches controlled by asynchronous obfuscated controllers. Hence, the timing (flow) of the latches is unknown for the adversary (locked). So, the adversary has no sufficient knowledge to do the unrolling cycle-accurately, and the sequential SAT attack is no longer applicable to a data flow obfuscated circuit.
2) With an obfuscated asynchronous controller, having access to the scan chain without any information about

[1]Two scenarios: (1) Adversary as end-user purchases and delayers the activated chip; however, the content of tpNVM will be wiped-out during RE. (2) Adversary as staff at foundry before fabrication does have access to the layout, but the key is not provided to the foundry [9].

1063-8210 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISON OF STATE-OF-THE-ART LOGIC OBFUSCATION TECHNIQUES

	Defense	Corruptibility	Resilient against ⇓ SAT [8]	Removal [10]	Approx. [11]	FALL [12]	cycSAT [13]	icySAT [14]	SMT [15]	CP&SAT [16]	NNgSAT [17]	seq-SAT [18, 19]	Shift&Leak [20, 21]	Scan Unlock [22, 23]	Overhead	Test/Implementation Issues: Limitation	Test Complexity	Test Time
CAT 1	SARLock [24]	low	✓	✗	✗	✓	✓	✓	✓	✓	✓	N/A	N/A	N/A	low	NO change	NO change	low
	Anti-SAT [25]			✗		✓												
	SFLL [26]			✓		✗												
CAT 2	Cyclic [27]	high	✓	✓	✓	✓	✗	✗	✓	✓	✓	N/A	N/A	N/A	high	NO change	NO change	low
	SRCLock [28]						✓	✗	✓							NO change	NO change	low
	Mem-Cyclic [29]						✓	✗	✓							NO change	NO change	low
	DLL [30]						✓	✓	✗							NO change	NO change	low
CAT 3	Cross-lock [31]	very high	✓	✓	✓	✓	✓	✓	✓	✗	✗	N/A	N/A	N/A	very high	NO change	NO change	low
	Full-Lock [32]	very high								✗	✗				very high			
	InterLock [16]	very high								✓	✓				high			
CAT 4	EFF+RLL [33]	high	✓	✓	✓	✓	✓	✓	✓	✓	✓	✗	✓	✗	low	NO change	NO change	low
	FORTIS+SLL [34]											✗	✗	✓	moderate	test coverage	NO change	low
	R-DFS+SLL [35]											✗	✗	✓	moderate	test coverage	NO change	low
	MSSD+RLL [20]											✓	✗	✓	moderate	test coverage	key init	high
	DynScan+SLL [36]											✓	✓	✗	high	trusted tester	NO change	low
	DisORC+TRLL [37]											✓	✓	✓	high	trusted tester	NO change	moderate
Data Flow Obfuscation		high	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	low	NO change	NO change	low

test coverage: In FORTIS, R-DFS, and MSSD, the scan chain would be blocked after key loading. Functional Test would be limited to observing POs.
key int: In MSSD, since any form of shift operation has been blocked after key loading, a sys_rst is required for each test pattern, which results in extremely high test time.

the exact timing (locked timing) of arrival/departure of data does not provide any advantages for the adversary. Hence, as shown in Table I, with open scan chain access, and without any limitation on the test phase, the data flow obfuscation is resilient against all state-of-the-art attacks at low overhead.

3) The latches controller will be implemented based on asynchronicity. Since the asynchronous latches controller is full of stateful/oscillating cycles, without any restriction on manufacturing stages or any incompatibility with conventional EDA tools, we show why the adversary is no longer able to engage any form of the SAT attack on this technique.
4) We thoroughly evaluate and compare the proposed data flow obfuscation with state-of-the-art countermeasures in terms of overhead and security, showing why this new paradigm will be strongly resilient against the existing attacks.

II. BACKGROUND

A. Combinational De-Obfuscation

For categories 1, 2, and 3 in Table I, an important threat model assumption is that the attack model is oracle-guided. In an oracle-guided attack model for combinational de-obfuscation, the adversary has access to an unlocked/activated chip (oracle) with open scan chain access, as well as the reverse-engineered yet locked netlist of the oracle. In the SAT attack, for any arbitrary obfuscated combinational logic ($c_{\text{comb\_lock}}$), by getting inspiration from the miter circuit used in formal verification, a (distinguishing) miter circuit has been built as miter $\equiv c_{\text{comb\_lock}}(\text{dip}, k_1) \neq c_{\text{comb\_lock}}(\text{dip}, k_2)$, which returns a specific discriminating input pattern (dip) that produces different output for two different keys k_1 and k_2. Then, this dip is queried on the oracle, c_{comb}, eval $\leftarrow c_{\text{comb}}(\text{dip})$ and the I/O-constraint $c_{\text{comb\_lock}}(\text{dip}, k_1) = c_{\text{comb\_lock}}(\text{dip}, k_2) = \text{eval}$ is stored back in the SAT solver and the miter circuit would be solved again. When the miter + constraints problem has no longer satisfying assignment, it could identify the correct key.

B. Sequential De-Obfuscation

Since the SAT attack is only applicable when the access to the scan chain is open, the studies in category 4 evaluate the security of traditional logic obfuscation techniques [4], [43] while the access to the scan chain is blocked/obfuscated. In this case, the adversary has only access to the PI/PO, and PO would be a function of PI and the state of the circuit, which makes it impossible for the SAT attack to formulate it at once.

To still exploit the combinational SAT attack while the scan access is restricted, few recent studies have engaged unrolling technique as a pre-processing step to formulate the sequential obfuscation using the combinational SAT attack [18], [19], [22], [23]. As shown in Fig. 1, the adversary unrolls the sequential circuits τ times. A τ-time unrolled circuit is an equivalent combinational model of a sequential circuit for τ clock cycles. It takes in τ input patterns (as a sequence), and produces τ outputs, while the intermediate states are cascaded.[2] After unrolling, similar to the combinational de-obfuscation, the SAT attack would find the sequences of inputs $(i_0, i_1, i_2, \ldots, i_{\tau-1})$, called distinguishing input sequence (dis) with two different keys k_1 and k_2 such that the outputs $(o_0, o_1, o_2, \ldots, o_{\tau-1})$ will differ. Every time the unrolled miter becomes unsatisfiable at some depth d (no more dis), the adversary extends the unrolling until a termination condition. Termination conditions are unique completion (UC): when there is only one key that satisfies the I/O-constraints for an unrolled circuit (correct key); combinational equivalence (CE): where the transition function is combinationally equivalent between two duplicated circuits with two keys (k_1 and k_2) (shadow key), and unbounded model check (UMC): if a call to an unbounded model checker ($\tau = \infty$) concludes that the result is invariant in the reachable state-space [18], [19].

[2]Similarly, model checking could be used to check the satisfying assignment in a sequential (transition) system. A model checker with bounded depth corresponds to bounded model checking (BMC) and an unbounded one to unbounded model checking (UMC).

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

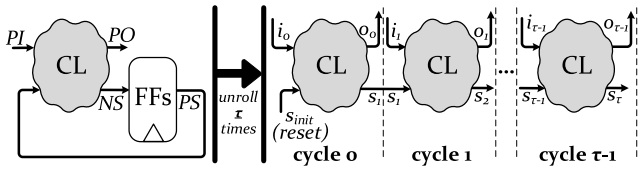

Fig. 1. Sequential circuit versus its combinational counterpart (τ cycles).

The point is that the unrolling step relies on the synchronicity of FFs in the obfuscated sequential circuit. When the sequential circuit is synchronous, moving forward from any arbitrary clock cycle to the next one (cycle $t \rightarrow t+1$) updates the FFs only once at positive (negative) edges of the clock signal. Hence, in the unrolling step, the combinational parts would be replicated only once per each clock cycle. But, for circuits and systems that *asynchronously* control the data flow in the circuit, the unrolling-based SAT or BMC faces a big obstacle during the unrolling step to build the equivalent combinational model for a specific number of clock cycles.

C. Asynchronicity

To asynchronously control the data flow in a circuit (partially or fully), one could adopt the asynchronous circuit paradigm. The asynchronous circuits have multiple advantages over synchronous circuits, particularly for newer technology nodes, such as no clock skew problems, robustness toward process variations, as well as advantages in terms of power consumption and electromagnetic emissions [44].

For two main reasons, most designers consider asynchronous circuits as a perilous approach: 1) the lack of electronic design automation (EDA) tools and 2) opposition to change designers' mentality toward asynchronicity. However, the ever-increasing attention on these circuits results in introducing powerful synthesis and verification tools for asynchronous circuits [45]–[47]. It allows any designer to nondisruptively incorporate asynchronicity in an EDA flow, and there is no need for the designer to change the synchronous mentality/structure. As an instance, An ARM is an ultra-energy-efficient asynchronous ARM processor that is successfully implemented and fabricated using the STMicroelectronics 28 nm technology, using standard cells and conventional CAD tools while achieving a 59% improvement in energy when compared with the ARM Cortex-A7 [48]. As of today, widespread application of asynchronous circuits could be seen in IoTs, NoCs, mixed-signal circuits, etc. [49], [50].

D. From Synchronicity to Asynchronicity

Signal transition graph (STG) is the formal specification of the asynchronous circuits, which is used in most asynchronous synthesis and verification tools [51]. The STG could be drawn from scratch by the designer based on the specification of the design. However, one could use the desynchronization paradigm that generates the equivalent asynchronous model of any synchronous circuit. By providing formal proofs of correctness based on the theory of Petri nets [52], the desynchronization [53] provides a fully automated flow for building the flow-equivalent asynchronous counterpart.

To build the flow-equivalent asynchronous model of any synchronous circuit using desynchronization, as shown in Fig. 2, after removing the clock signal, FFs would be

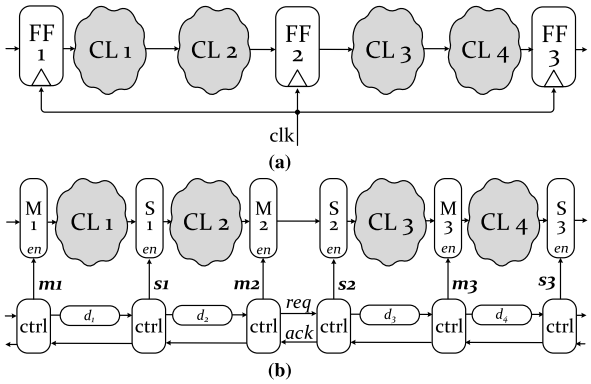

Fig. 2. Synchronous to asynchronous conversion [53]. (a) Synchronous circuit. (b) Desynchronized circuit.

replaced with master (M) and slave (S) latch pairs. All latch enable signals (en) must be controlled using new macros, called asynchronous latches controllers, which use a handshaking structure (req, ack) to emulate FFs' behavior. For example, in four-phase handshaking, as the most prevalent handshaking protocol, $\phi 1$ is enabling req by a sender for valid data. $\phi 2$ is enabling ack by the receiver, acknowledging the arrival of the new data. $\phi 3$ is lowering (disabling) previous req, and finally $\phi 4$ is lowering the corresponded ack. Handshake signals are not related to a global clock and are based on the local, relative timing relationships between the opening and neighboring latch enable signals. Also, during desynchronization, delay elements must be added per each combinational logic (CL) to mimic the delay of all timing paths and asynchronous latch controller. Also, the first/last latches of the asynchronous part [m_1 and s_3 in Fig. 2(b)] that are dealing with other (synchronized) parts of the circuit will be handled by some controlling signals, e.g., a specific state of the circuit, or controlling signals like FFs enables.

It is also worth mentioning that all latches operate on the basis of their controllers. By using desynchronization, one latch will be enabled when tokens are ready and will be disabled after receiving the ack corresponding to the new data (lowering req). It will prevent extra propagation when inputs of the latch change, which avoids increasing power consumption. Furthermore, latch controllers are the only added parts when the desynchronization is accomplished; however, we show that since the proposed solution is required to be accomplished on a small part of the circuit, it does not incur large area/resource overhead.

E. Desynchronization

Three main steps of the Desynchronization, which provides a fully automated methodology to build the flow-equivalent asynchronous model of any synchronous circuit are: 1) converting FFs to M and S latches, with decoupled enable signals (e.g., in Fig. 2(b) FF_i is replaced with M_i and S_i whose controllers are m_i and s_i), 2) matched delays generation for combinational logics (CLs), based on their timing path delays (e.g., d_{1-4} are matched delay for FF_{1-4} to mimic the delay of their timing paths as well as the delay of each asynchronous latch controller (m_{1-3} and s_{1-3})), and 3) implementation of the asynchronous controller of each latch, e.g., ctrls in Fig. 2(b), based on the data flow dependences in the original netlist.

In step 1, after replacing FFs with latches, re-timing is often used as a performance improvement technique [54]. By using

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

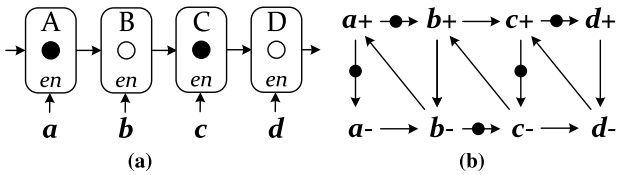

Fig. 3. Desynchronization model for a pipeline [53]. "•" are tokens (ready data). "○" are bubbles (not-ready data). Latches are transparent when "*en*"s are high. "$a+$" means rising transition, and "$a-$" means falling transition. (a) Four cascaded latches. (b) Corresponding STG.

re-timing, latches are moved across CLs (e.g., M_3 is placed before CL_4. S_1 is placed after CL_1 in Fig. 2).

In step 2, matched delay elements are generated for emulating the timing path delay of their corresponding CLs. These will be connected to corresponding controllers in the next step. In this step, the netlist is synthesized for the target cycle time T_T, using a conventional synthesis tool. The T_T is captured using $\{T_T \geq T_{CQ} + T_C + T_L\}$, in which the T_T is a delay between two rising edges of control signal of the latch, T_{CQ} is the delay of local clock propagation through a latch, T_C is the delay of the CL, and T_L is the latch controller delay. By using this inequality, and based on the delay of critical paths in each CL, these matched delays are generated. When T_Cs are equal in all CLs (balanced timing paths), then the separation time between adjacent rising edges of every local clock equals T_T. Also, in any desynchronized circuit, the ith rising transition of a local clock cannot appear later than $(i-1) \times T_T$, showing that the temporal behaviors of the desynchronized circuits are also similar to synchronous counterpart [53].

Step 3 implements the asynchronous controller for each latch. These controllers are connected to the controllers of neighboring latches with the delay elements built during step 2. A variety of desynchronization models exist to implement these asynchronous controllers. The behavior of these models can be typically specified using STG, which is a decision-free subclass of Petri nets [52]. An STG, as shown in Fig. 3(b), may be defined as a 3-tuple (Φ, $\rightarrow$, I_0), where Φ is the set of events, and events are the latch enable values (high/low) in asynchronous controllers. $\rightarrow$ corresponds to an arc, which illustrates event transitions, and for a latch controller, it determines changes in latch enable values. I_0 is the initial marking, called a token, and denotes the initial event signal states. In desynchronization, it is crucial to properly define I_0, as the initial tokens, and it is fully dependent on the handshaking protocol used for desynchronization [53]. Tokens determine which data are ready. In STGs, as in Petri nets, these tokens could be updated (moved) based on the interaction between different latches. For example, a signal is enabled when all its predecessor arcs are marked with a token. An enabled signal can fire, removing tokens from all its predecessors' arcs, and populate tokens to its successors' arcs. For instance, Fig. 3(a) shows a part of a pipeline with cascaded latches. Fig. 3(b) depicts an STG representing the behavior of these latches. Over time, based on the location of data, tokens move around determining which latch will catch new data. Without loss of generality, we use semi-decoupled four-phase control for handshaking [55], which represents a good trade-off between simplicity and performance, however, any valid desynchronization latch controller may be used instead [53].

Based on the generalization of semi-decoupled four-phase control, there are four rules imposed on the latch control

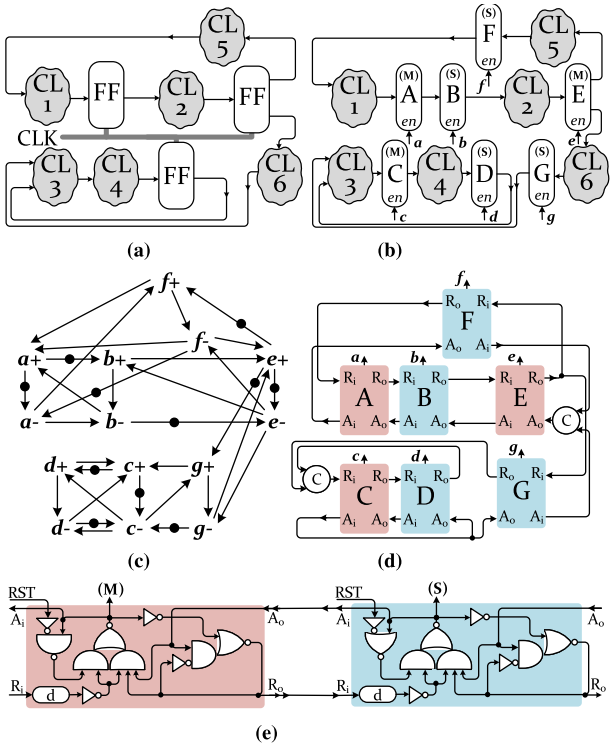

Fig. 4. Desynchronization flow [53]. (a) Synchronous circuit. (b) FFs→{Ms and Ss} + re-timing. (c) STG generation using *semi-decoupled four-phase control*. (d) Building the asynchronous controller based on its generated STG. (e) Latch controller (Left: Ms, Right: Ss) with matched delays (d).

signals of the STG. Using these four rules, the designer can specify the corresponding STG for any circuit: 1) $a+ \rightarrow a-$: rising of each signal (each latch enable) should be followed by falling of that signal, 2) $b- \rightarrow a+$: For latch A (master) to read a new data, latch B (slave) must have completed the reading of a previous token from A, 3) $a- \rightarrow b-$: For latch B (slave) to complete the reading of a data token coming from A (master), it must first wait for latch A to complete the reading of that data token, and 4) $a+ \rightarrow b+$: For latch B (slave) to read a new data, it must wait for latch A to read that new data token.

Considering these four rules, we now illustrate an example of the desynchronization methodology based on semi-decoupled four-phase control for a small circuit. Fig. 4(a) is an arbitrary synchronous netlist with three FFs. In Fig. 4b, all FFs are replaced with latches subjecting to re-timing (e.g., CL_4 is placed between latches C and D). Since the fan-out of rightmost FF in Fig. 4(a) is two, it is converted to one M, (E), and two Ss (F and G). After converting FFs to latches, the corresponding STG is generated based on all four aforementioned rules [see Fig. 4(c)]. Then, based on the drawn STG, the corresponding asynchronous controller for latches enables is implemented. Considering that we use the semi-decoupled four-phase control for this article, the circuit depicted in Fig. 4(e) must be engaged for each latch controller. It is latch controller (left (red) one for Ms and right (blue) one for Ss) based on semi-decoupled four-phase control. The handshaking signals between Ms and Ss are connected directly. However, for multiple dependences in STG (i.e., one to many latches, or vice versa), the handshaking must be handled by merging req or ack signals. This merge is performed using C-elements, which is an event-driven AND gate. A possible

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

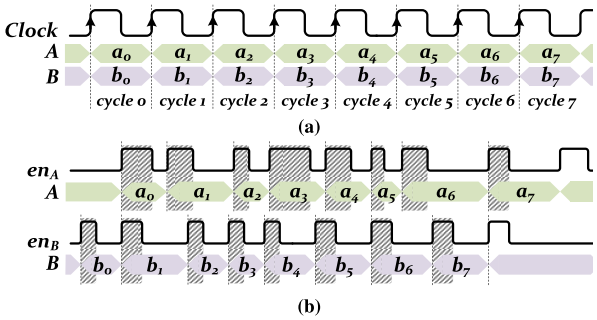

Fig. 5. Timing diagram of synchronous versus asynchronous circuits [53]. (a) Synchronous behavior. (b) Asynchronous (desynchronized) behavior.

implementation is using the function $Z = AB + ZA + ZB$, where A and B are the C-element inputs, and Z is its output. For example, as shown in Fig. 4(c), latch C is dependent on both D and G. So, in Fig. 4(d), R_i of latch C is driven using a conjunction of R_os of D and G using a C-element.

After implementing the asynchronous latch controller, it must be connected to the circuit depicted in Fig. 4(b) to build the complete desynchronized counterpart of the synchronous circuit. It is proven in [53] that: 1) desynchronized circuit never halts (liveness property) and 2) sequence of data values of a desynchronized circuit are identical to its synchronous counterpart (flow-equivalence).

Also, by using desynchronization, physical design, verification, and design for testability can be accomplished "as is," using conventional synchronous EDA tools [53]. For instance, for the design for testability, a low-frequency clock may be distributed to latches in test mode [56]. For testing the asynchronous controllers, as rising and falling of reqs and acks follow each other, in the presence of a stuck-at fault on either req or ack, either the environment or the circuit will wait forever, and cause a deadlock, which may be easily detected during design for testability. Circuits that have the property that they halt for all faults are called self-testing.

F. Concept of Data Flow Obfuscation

Since the sequential SAT attack relies on the synchronous unrolling mechanism, the preserved flow-equivalency after desynchronization, motivates us to propose a new obfuscation paradigm. In general, two circuits could be called flow-equivalent if there is no difference between the sequence of values stored at each latch. The observation is done independently for each latch. As an example of flow-equivalent circuits Fig. 5 demonstrates two flow-equivalent circuits. Fig. 5(a), shows the synchronous behavior, while Fig. 5(b) shows the desynchronized behavior. Using this characteristic of asynchronous circuits, in Section IV, we show how our proposed obfuscation technique could get benefit from this flow-equivalency concept to introduce a new obfuscation paradigm, called data flow obfuscation.

III. Threat Model

Similar to categories 1, 2, and 3 in Table I, we make the following assumption about the adversary capabilities: 1) the adversary can successfully do the reverse-engineering on the chip, and retrieve the gate-level netlist (yet locked), 2) the adversary can purchase an activated/unlocked chip (oracle) from the market, and 3) the access to the scan chain of the oracle is not restricted. So, the adversary could apply any query to FFs using SI and read the updated values through SO after one clock cycle (capture mode).

IV. Proposed Scheme: Data Flow Obfuscation

As discussed previously, in all existing logic obfuscation techniques, synchronicity is kept intact during the manufacturing stages. However, due to the synchronicity, these techniques were vulnerable to unrolling-based SAT or BMC even while the scan chain is restricted. It should be noted that, for the most potent attacks on logic obfuscation, there exists a big inspiration from a formal verification method such that the attack relied on and adopted from the verification method to successfully de-obfuscate circuits, e.g., miter circuit in traditional SAT [8], or BMC/unrolling in sequential SAT [18]. Hence, the main aim of this new obfuscation paradigm is to add ambiguity in a way such that it turns the obfuscated circuit into a completely new form that cannot be modeled using any of the existing formal verification methods. We target part of the data flow in a circuit to be obfuscated using asynchronicity. When asynchronicity is used in a circuit, due to the high nondeterministic behavior, it is extremely challenging to come up with an automated approach to establish invariance properties, which are vital in proving the correctness of a circuit with asynchronous parts. There exist a few methods that ease the formal verification in asynchronous parts [57], [58], helping the designers to do formal verification for datapath of asynchronous circuits. However, to prevent any form of easing, in our proposed data flow obfuscation, we target to obfuscate the asynchronous controllers, which is the source of desynchronization with the self-testable feature. Also, since we assume that the scan access must be still fully open, the proposed obfuscation must be in a way that conceals the writing/capturing into/from the storage elements. Hence, in the proposed solution, the controller of latches is obfuscated such that without the correct key, the temporal characteristics of the datapath will be hidden.

A. How Data Flow Obfuscation Works?

The main steps of our proposed data flow obfuscation are as follows.

1) Converting the targeted synchronous part(s) of the circuit to its (their) flow-equivalent asynchronous counterpart using desynchronization described in Sections II-D and II-E[3,4]
2) Inserting false paths into the desynchronized circuit. Each false path could be extra wiring from the output of one latch to any arbitrary combinational logic.
3) Updating the corresponding STG based on the added false paths. For each false path, few extra transitions with an initial token must be added to the STG to reflect the changes.
4) Obfuscating the asynchronous latches controller circuit (based on STG with respect to the new false paths)

[3]The conversion could be done fully (whole circuit) or partially (part(s) of the circuit). However, targeting and obfuscating only critical part(s) of the circuit (partially) guarantees security at lower overhead.

[4]Re-timing will be done as a part of de-synchronization in data flow obfuscation. However, it might not change the location of latches (relocation). So, we need to apply a minor relocation for some latches to avoid any form of re-synchronization-based attack described in Sections VI-A and VI-B.

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

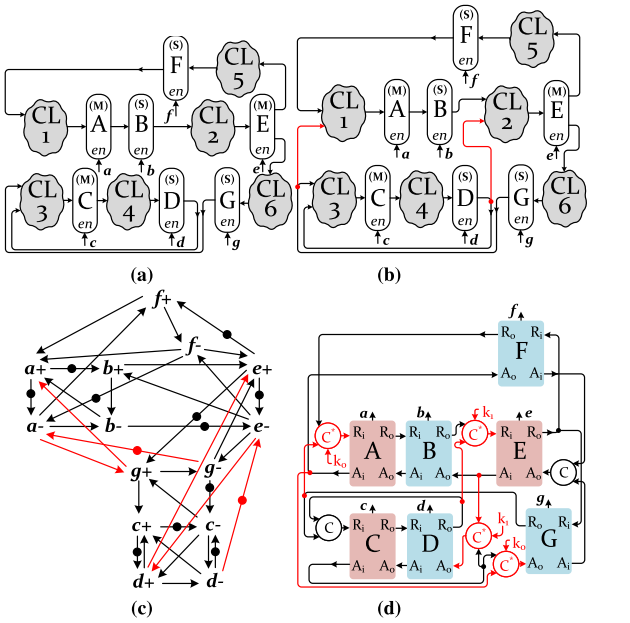

Fig. 6. An example of data flow obfuscation by inserting only false paths into the circuit controlled by obfuscated asynchronous controller. (a) Original asynchronous circuit. (b) False paths insertion. (c) Updating the STG. (d) Locked asynchronous control.

by using proposed C*-element that is a key-controlled event-driven AND gate.

Fig. 6 demonstrates step-by-step implementation of the data flow obfuscation on the circuit from Fig. 4(a). First, the targeted parts of synchronous circuit [see Fig. 4(a)] are converted to their asynchronous counterpart [see Fig. 6(a)]. Then, in the desynchronized circuit a specific number of false paths are inserted (e.g., $D \rightarrow CL_2(E)$ and $G \rightarrow CL_1(A)$ in Fig. 6(b)). Since the connectivity between latches is altered, the STG should be updated [see Fig. 6(c)]. Also, the changes must be reflected into the asynchronous controllers. For instance, before adding the false paths, F was the only predecessor of A. So, R_o of F was directly connected to R_i of A. However, after adding the false path $G \rightarrow A$, both F and G are the predecessors of A. Thus, a C-element must be added to merge their req signals. The C-element here implies that latch A may only be opened whenever data from both G and F are ready. However, for any false path like $G \rightarrow A$, it should have no impact on timing when the key is correct. To achieve this, we introduce a C*-element, in which a key-controlled MUX is used to control the C-element's inputs. As shown in Fig. 7(a), based on the key value, the C-element input is either $\{A, B\}$ or $\{A, A\}$, and based on the C-element's definition, if both inputs are the same, the output will be equal to the identical input pair: $Z = AA + ZA + ZA = A$, meaning that with the correct key, the added false paths will have no timing effect.

As shown in Fig. 6, the only obfuscated part in data flow obfuscation is the usage of C*-elements that alter the behavior of controllers based on the key value. However, these C*-elements only control the timing (behavior) of latches. Hence, regardless of the key value, each false path is connected directly to one arbitrary chosen CL and could affect its functionality. For instance, for false path $G \rightarrow A$, regardless of the key value (k_0), latch G would affect the functionality of CL_1, and the key value only controls the time of the act. To avoid this problem, the false paths can be connected to the

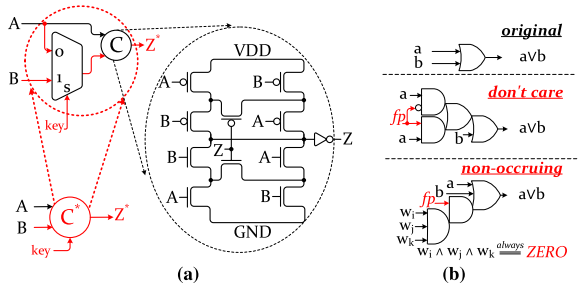

Fig. 7. Essential modules for false path insertion. (a) C*-element. (b) Function bypassing.

chosen CL as don't cares, or nonoccurring inputs. Fig. 7(b) shows these two models for a simple circuit. As shown, the output of all cases is the same, i.e., $a \vee b$, and the added false path fp does not affect the output in both cases. For instance, in nonoccurring,[5] fp is ANDed with $w_i \wedge w_j \wedge w_k$, which is always $ZERO$, and has no impact on the logic.

B. Shortcomings of False Path Insertion

With inserting *only* false paths, the data flow obfuscation is, however, vulnerable to re-synthesis and removal attack. Since the false paths are connected to corresponding CLs as don't care or nonoccurring, they explicitly have no impact on the circuit's functionality. Hence, the attacker could re-synthesis the reverse-engineered netlist, and by using logic optimization effort during the synthesis, the false paths will be removed during optimization. Then, the adversary can find some extra elements/connections in the controller that have no corresponding part in the datapath (already removed). So, he/she can distinguish between the original parts and the extra logic added for the false paths in the controller and retrieve the original circuit. So, to combat this issue, we add one more step which adding extra false latches on false paths.

C. Adding False Latches on the False Paths

We updated and added one more step in our proposed data flow obfuscation to support adding false latches.

(1) (Same Step) desynchronization + re-timing (relocate).
(2) (Same Step) Inserting false paths into the circuit.
(3) (New Step) Inserting false latches (M/S pairs) on the false paths + an asynchronous controller for each added false latch to control its behavior.
(4) (Same Step) Updating the STG based on new insertions.
(5) (Same Step) Obfuscating the controller via C* elements.

Inserting pairs of false latches in false paths allows us to control the logic value of these paths. So, there is no longer a need to add false paths as don't care or nonoccurring, and the re-synthesis and removal attack is no longer a valid attack. The concept of insertion the false {paths + latches} is visualized in Fig. 8. Similar to the previous example, first, the circuit must be converted to its asynchronous counterpart (desynchronized). Fig. 8(a) shows the asynchronous model of our simple circuit from Fig. 4(a). After that, one false path is added from $CL_2(B)$ to $CL_1(A)$, then a pair of master and slave latches (I and H), are added in this path [see Fig. 8(b)].

[5]*Non-occurring* cases can be found using SAT solver. First, few wires must be selected, then its condition clause must be added, and solved by SAT. If SAT solver returns *UNSAT*, that condition is *non-occurring*.

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

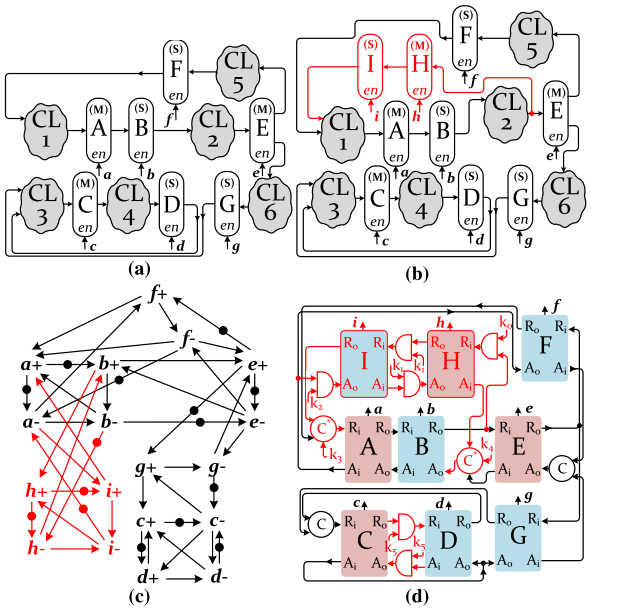

Fig. 8. An example of data flow obfuscation by inserting false {Paths + Latches} into the circuit controlled by obfuscated asynchronous controller. (a) Original asynchronous circuit. (b) False {Paths + Latches} insertion. (c) Updating the STG. (d) Locked asynchronous control.

Based on these changes, the STG is updated [see Fig. 8(c)]. Compared to the STG of the previous example, not only new transitions are added, the STG has new nodes describing the events of new false latches. Finally, these updates must be reflected into the obfuscated asynchronous controller [see Fig. 8(d)]. Note that extra controller modules are added for false latches. Also, the key-controlled gates must be added to properly control the behavior of latches H and I. When the key is correct, the added false path must have a value that does not affect the functionality of CL_1. For this purpose, the behavior of the false latches is controlled using k_{0-2}. So, while the k_{0-2} is correct (000 in this case), the AND gates mask the handshaking of H and I with their neighboring latches. Hence, these latches are disabled (and no new data will be captured in them). So, the initial value of these latches will be kept intact and will be used as the new input of CL_1. In CL_1, based on the initial value of these latches, this false path will be connected to an arbitrary gate (e.g., with initial value 0, it could be connected to an OR or XOR gate). However, while the key is not correct, the behavior of these latches would be changed repeatedly, resulting in corrupting the functionality of CL_1.

Additionally, false {paths + latches} must not affect their neighboring latches when the key value is correct (e.g., latches H and I must have no effect on A and B in Fig. 8(d)). This is achieved by adding two C*-element before A and B, controlled by k_3 and k_4, respectively, to effectively eliminate this temporal relation (e.g., in path $I \rightarrow A$, C*-element skips the effect of the behavior of I on the behavior of A).

D. Key Classification

The keys added to asynchronous latches controllers will be categorized into two main groups: 1) handshake-in keys: keys that control the impact of incoming signals to false latches (k_{0-2} in Fig. 8(d)) and 2) handshake-out keys: keys that control the impact of outgoing signals from false latches (k_{3-4} in Fig. 8(d)). Based on the value of these two groups, different scenarios could happen.

1) *Correct Functionality*: while both groups are correct. In this case, similar to the timing diagram depicted in Fig. 9(a), the firing of latches alternates appropriately.
2) *Halt in Data Flow*: While handshake-in keys are correct, but handshake-out keys are incorrect, halts will happen (e.g., if $k_{0-2} = 000$ and $k_{3-4} \neq 00$, H and I would halt). As shown in Fig. 9(b), after latch H controller (h) is halted, more halts are happened in other paths and results in a complete deadlock in the whole circuit.
3) *Incorrect Functionality*: While the handshake-in keys are incorrect, regardless of the handshake-out keys, the function will be incorrect.

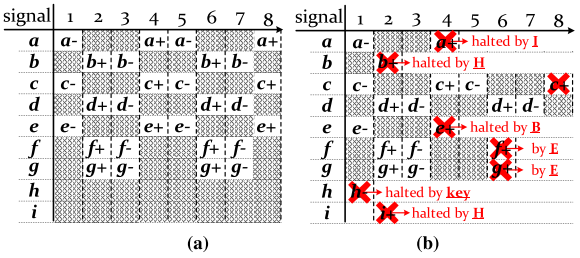

Fig. 9. Comparison of timing diagram of latch enables. (a) Original circuit. (b) Locked desynchronized circuit.

V. Security/Testability Analysis

A. Security versus the SAT Attack

Since the adversary has access to the scan chain in data flow obfuscation, he/she is able to apply combinational de-obfuscation for any accessible part of the circuit using the SAT attack. However, for two important reasons, the traditional SAT attack cannot be applied on data flow obfuscated circuit.

1) In the SAT attack, it is crucial to know the exact time of writing/capturing into/from the scan chain; but, in data flow obfuscation, this timing is controlled (locked) by an asynchronous controller. The adversary cannot determine when he/she must write into the scan, and when the updated data is ready to be observed.
2) Due to the nature of asynchronous controllers, the latch enable controller consists of many stateful cycles. The SAT solver works perfectly fine if the circuit is a directed acyclic graph (DAG), and only structural cycles could be analyzed using a pre-processing engine before running the SAT solver. Depending on whether the cycle is oscillating or stateful, the SAT solver will either be trapped in an infinite loop or will return UNSAT. Moreover, in attacks such as BeSAT [40] that can track and detect nonstructural cycles, the very first assumption is that the circuit has no stateful combinational cycle by itself.

B. Security versus Sequential SAT Attack

The adversary may attempt to engage either an unrolling-SAT attack or SAT integrated with BMC (SAT-BMC) by creating the unfolded combinational equivalent circuit to find *dises*. The length of *dises* determines the number of unrolling required before running the SAT solver. When the circuit is synchronous, per each clock cycle, the FFs will be updated only once. So, the adversary can replicate

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

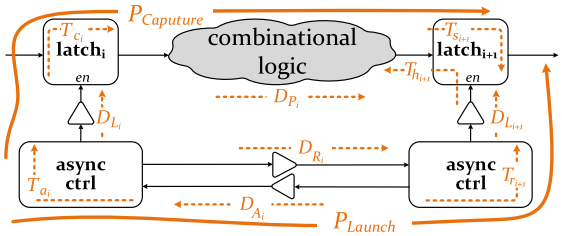

Fig. 10. Hold and setup paths in an asynchronous circuit [59].

whole CLs iteratively (per each clock cycle) to build the unrolled circuit. However, when we use asynchronicity in data flow obfuscation, the adversary needs to know the list of enabled latches continuously and cycle-accurately to unroll those parts that are triggered with new latched data. But, in the data flow obfuscation paradigm, the asynchronous controller that determines which latches must be enabled/disabled is obfuscated. So, the adversary cannot build the unrolled circuit to still get the benefit of the SAT attack, and thus, the sequential SAT attack cannot be applied to this technique.

C. Security versus Re-Synchronization + Sequential SAT Attack

Since the sequential SAT attack cannot be applied directly to data flow obfuscation, the adversary might add a pre-processing step, such as re-synchronization, to make this attack valid. In re-synchronization, which is the reverse of desynchronization, the asynchronous controller is removed, M and S latches are merged as FFs, and all FFs are connected to a global clock. However, for a few reasons, re-synchronization of the obfuscated desynchronized netlist is not possible:

First, since each controller requires a local clock tree in asynchronous circuits, and these local trees do not have the same delay [59], the adversary needs to confirm two constraints below, which contradict each other.

1) Theoretically, the adversary must use a clock period larger than any delay element, to avoid metastability happening in the asynchronous controllers from the delay element. The delay constraints of an asynchronous circuit could be modeled using the following formulas obtained from Fig. 10:

$$D_{R_i} + T_{r_{i+1}} + D_{L_{i+1}} > D_{L_i} + T_{c_i} + D_{P_i} + T_{s_{i+1}} \quad (1)$$

$$D_{A_i} + T_{a_i} + D_{L_i} + T_{c_i} + D_{P_i} > D_{L_{i+1}} + T_{h_{i+1}}. \quad (2)$$

2) But, the adversary must use a very small clock period to utilize the delay element as a time offset. Hence, based on these two constraints, the timing correctness conditions cannot be satisfied. Also, the timing constraints between the datapath and the asynchronous controller must be preserved making it more challenging [59].

Second, as a step during re-synchronization, the attacker must analyze every connectivity in the netlist, and effectively create a mapping problem using bipartite graphs between pairs of $\{M, S\}$ to FFs. To accomplish this, the attacker must have prior knowledge of the design methodology used for creating the asynchronous circuit (2-phase latches or 3-phase latches, handshaking protocol, initial marking (tokens), etc.). Even while the adversary has access to this prior knowledge, since the connectivity is obfuscated using false paths, the false mapping will be added to this bipartite graph, leading to failure of correct matching between Ms and Ss.

Additionally, as shown in Figs. 2 and 4, during desynchroniztion flow, re-timing has been engaged in data flow obfuscation by moving parts of CLs before/after the latches. By doing so, re-synchronization cannot be accomplished directly. Re-timed asynchronous circuit can be converted to a 2-phase nonoverlapping synchronous design, with two clock signals, $\phi 1$ and $\phi 2$. However, false paths with extra latches makes this 2-phase nonoverlapping synchronous netlist malfunction. For example, latches H and I in Fig. 8(b) would be connected to clock signal $\phi 1$ and $\phi 2$ (non-overlapping clock signals). By connecting these two false latches to clock signals, their values would be updated which alters the functionality of CL_1.

D. Security versus Structural-Based Attacks

The attacker might try to guess the value of the keys based on the overall structure of the locked netlist. For instance, all handshaking signals to/from latches H and I in Fig. 8(d) are controlled using C*-element and key-gates (ANDs). Hence, the attacker might guess that this pair of latches are false latches located on a false path. So, the value of the keys can be retrieved easily. However, to avoid such circumstances, these key-gates and C*-element will be added for a set of arbitrary (actual) latches in the netlist. For example, in Fig. 8(d), the same key-gates are added between C and D. However, they always must be active. Also, the original C-element before C and E could be replaced with C*-element. So, unlike C*-*element* before A and B, in which only one of the inputs is valid, in these C*-elements, both inputs are valid. By using this simple mechanism, the attacker cannot start guessing/detecting the false {paths + latches} based on the location/type of key gates.

E. Security versus Other State-of-the-Art Attacks

As is shown in Table I, there exist many attacks on different logic obfuscation techniques, each is modeled to break one or more specific techniques. However, Table II explains why none of these attacks is applicable to the proposed solution. The biggest advantage of the proposed data flow obfuscation is the usage of extensive nondeterminism of asynchronicity for obfuscation purposes. In data flow obfuscation, the main source of this nondeterminism, which is the asynchronous controller, is the main target of obfuscation. Obfuscating an asynchronous controller makes every step of simplification dependent on the key value, and it extremely boosts up the state space in the asynchronous part. This implies the difficulty of attacking the proposed data flow obfuscation, where it requires an extensive (likely impossible) investigation on how the existing formal verification methods might be fit and useful to be adopted in this case.

F. Testability of Data Flow Obfuscation

As discussed previously, the handshaking asynchronous controller is a self-testing circuit. However, since we use the halt in false paths for logic locking purposes, data flow obfuscation prevents the self-testing and contradicts this property. To protect against an untrusted test, this contradiction enforces the designer to use an incorrect key to keep the self-testing property of these circuits. For instance, k_{0-2} in Fig. 8(d) must be 111 to avoid any halt (the correct key is 000). As another example, similarly, k_5 in Fig. 8(d)s must be 1 to avoid halt on

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

TABLE II

OUR PROPOSED DATA FLOW OBFUSCATION AGAINST STATE-OF-THE-ART ATTACKS ON LOGIC LOCKING

Attack	To break	The reason why this attack fails breaking the proposed data flow obfuscation
Traditional SAT [8]	Primitive Logic [4, 43]	It requires to write/capture into/from scan flip flops, but the exact time of writing/capturing into/from the latches (replaced with FFs) and that of neighboring FFs is hidden (locked) by the obfuscated asynchronous controller.
Removal [10]	SARLock, AntiSAT [24, 25]	When the added logic for locking is completely separated from the original circuit, it finds and removes the locking part to retrieve the original circuit. In the data flow obfuscated circuit, after removing all key-related parts from an asynchronous controller, extra latches+paths will remain active in the datapath affecting the original functionality.
Approximate SAT [11]	AntiSAT/SARLock + Primitive [10]	It guesses a key with low error rate when low and high-corruption techniques are compounded. But, since the data flow obfuscation is not added on a specific point, the output corruptibility is high, and in this case, the error rate of the approximate key will be high (useless).
FALL [12]	SFLL [26]	This attack is an upgraded version of removal attack combined with functional analysis, and specialized for SFLL (SFLL is an expanded version of SARLock). However, since the false {latches+paths} are not detectable in data flow obfuscation, after applying removal on data flow obfuscated circuit, the functional analysis cannot deal with the malfunction caused by false {latches+paths}.
cyclic-SAT [13, 14, 40]	cyclic locking [27–29]	As a basic assumption, in a cyclic-based SAT attack, the stateful cycles will be skipped before pre-processing the combinational cycles. However, in data flow obfuscated circuit, the keys are located in stateful cycles, and this attack cannot formulate this form of cycles.
SMT [15]	Delay Locking [30]	Using a graph theory solver, the SMT formulates setup/hold time to guarantee the timing constraint. In data flow obfuscated circuit, there is no timing violation when the key is not correct making this attack useless.
CP&SAT [16]	Cross-lock [31], Full-lock [32]	It finds the CNFs corresponded to routing modules, and simplifies them using cardinality constraints. Then It uses the SAT attack on the simplified CNF. It only works on routing-based obfuscation and has no efficiency on data flow obfuscation.
Sequential SAT [18, 19]	Scan/Sequential Lock [33–35]	It requires unrolling, and the time of unrolling in synchronous circuits was at rising/falling clock edge, and synchronous for all FFs. But, in data flow obfuscated circuit, the time that latches are updated is desynchronized and hidden using an obfuscated asynchronous controller.
Shift/Leak [20, 21]	Scan Block [20, 35]	It is very specialized for only one case that has leaking possibilities. Not Applicable to data flow obfuscated circuit.
Scan Unlock [22, 23]	Scan Lock [33, 36]	It detects the structure of LFSR used for dynamicity. No special deterministic structure is used in data flow obfuscation to be detected using the same approach.

the other path (but in this case, the correct key is 1). Also, since C*-element does not create a halt in any path, keys connected to C*-element could be an arbitrary value to choose a path in latches controllers. It shows that there is no relation, e.g., bit flipping, between the correct key and key used for the test. Using incorrect key allows false latches located in the false paths to be updated. Hence, false paths can affect the CLs' functionality. Although the designer can generate test patterns that avoid making them driving, since the incorrect key only adds false paths to the original netlist, few more test patterns are required to test these paths, which has no impact on test patterns generated for original parts of the netlist. Hence, there is no restriction for test pattern generation.

VI. EXPERIMENTAL RESULTS

We evaluate the data flow obfuscation over three sets of benchmark circuits, all listed in Table III. The experiments are all executed on a 24-core Intel Xeon processors running at 2.4 GHz with 256 GB of RAM. Area, power, and delay overhead of the data flow obfuscation are obtained using conventional Synopsys Design Compiler along with Synopsys generic 32 nm library. We evaluate the security/overhead of the data flow obfuscation based on the following methods.

1) *Obfuscation Overhead:* Selection, desynchronization, and insertion of false {paths + latches} depend on the circuit size.
2) *Key Size:* Regardless of the circuit size, for a key size, a nearly fixed part of a circuit will be selected, desynchronized, and false {paths + latches} will be inserted.

A. Modeling of Attacks on Data Flow Obfuscation

Since the proposed data flow is dependent on the locked asynchronous controller, the timing of writing/capturing into latches is hidden, and the traditional SAT attack does not work even while the scan chain is available. So, to evaluate the security of the data flow obfuscation, we deploy two new versions

TABLE III

SPECIFICATIONS OF THE BENCHMARK CIRCUITS (ISCAS'89, ITC'99, AND WELL-KNOWN ASICS/MICROPROCESSORS)

Small Circuit:	s298	s526	s1423	s5378	s9234	s13207	s15850	s35932	s38584
# of Inputs	3	3	17	35	36	62	77	35	38
# of Outputs	6	6	5	49	39	152	150	320	304
# of Gates	119	193	657	2779	5597	7951	9772	16165	18253
# of FFs	14	21	74	179	211	638	534	1728	1426

Large Circuit:	b17	b18	b19	MC8051	AES-GCM	SPARC
# of Inputs	37	37	24	52	116	95
# of Outputs	97	23	30	112	15	108
# of Gates	∼28K	∼95K	∼190K	∼6.6K	∼49.5K	233K
# of FFs	∼1.5K	∼3.3K	6.6K	∼1K	∼5.1K	12K

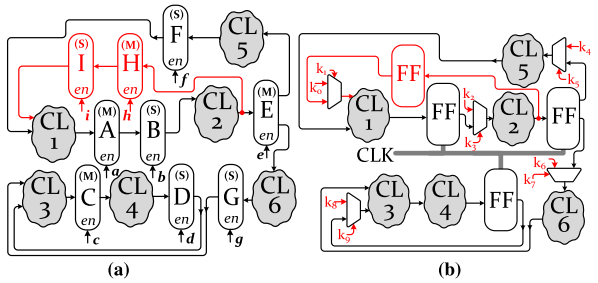

Fig. 11. Re-synchronization using MUX-based path selection. (a) Obfuscated asynchronous circuit. (b) Re-synchronized circuit.

of sequential SAT: 1) S_SAT: Resync + BMC + SAT and 2) S_BeSAT/S_icySAT: BMC + BeSAT/icySAT [14]/[40].

Regarding the former version of the deployed attack, due to the failure of unrolling on desynchronized circuits, we need a pre-processing step to re-produce the re-synchronized version of the obfuscated circuit. We discussed in Section V-C that the exact re-synchronization is almost impossible. In this section, to validate our claim, regardless of the timing criteria, we developed an intuitive re-synchronization technique. The steps of the re-synchronization are as follows: 1) removing the obfuscated asynchronous controller, 2) merging each pair of M and S latches based on the connectivity of them (moving them across CLs), 3) replacing each pair of M and S latches

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

TABLE IV

RUNTIME OF RE-SYNCHRONIZATION + SEQUENTIAL SAT INTEGRATED WITH PURE SAT (S_SAT), INTEGRATED WITH BESAT (S_BESAT), AND INTEGRATED WITH ICYSAT (S_ICYSAT), ON THE DATA FLOW OBFUSCATION WITH 1%, 5%, AND 10% OBFUSCATION OVERHEAD, AND ON THE DATA FLOW OBFUSCATION WITH KEY SIZE = 100, 200

Circuit	Obfuscation Overhead = 1%			Obfuscation Overhead = 5%			Obfuscation Overhead = 10%			Key Size = 100			Key Size = 200		
	S_SAT	S_BeSAT	S_icySAT	S_SAT	S_BeSAT	S_icySAT	S_SAT	S_BeSAT	S_icySAT	S_SAT	S_BeSAT	S_icySAT	S_SAT	S_BeSAT	S_icySAT
s298	~~139.8~~ UNSAT	*UNSAT*	inf/l	~~463.5~~ UNSAT	*UNSAT*	inf/l	w/k	*UNSAT*	*UNSAT*	~~207.1~~ w/k	inf/l	inf/l	*UNSAT*	inf/l	inf/l
s526	~~1580.4~~ w/k	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l
s1423	w/k	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*	inf/l	w/k	inf/l	inf/l	~~568.7~~ UNSAT	inf/l	inf/l	w/k	*UNSAT*	inf/l
s5378	~~552.7~~ w/k	inf/l	inf/l	w/k	inf/l	*UNSAT*	*UNSAT*	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	inf/l	*UNSAT*	inf/l	inf/l
s9234	w/k	*UNSAT*	inf/l	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	*UNSAT*	inf/l	w/k	inf/l	inf/l	*UNSAT*	*UNSAT*	inf/l
s13207	timeout	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	*UNSAT*	w/k	inf/l	*UNSAT*
s15850	*UNSAT*	inf/l	inf/l	w/k	inf/l	inf/l	w/k	inf/l	*UNSAT*	*UNSAT*	inf/l	inf/l	w/k	*UNSAT*	inf/l
s35932	timeout	inf/l	inf/l	w/k	inf/l	inf/l	w/k	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l	w/k	inf/l	*UNSAT*
s38584	w/k	*UNSAT*	*UNSAT*	w/k	*UNSAT*	*UNSAT*	*UNSAT*	inf/l	*UNSAT*	w/k	inf/l	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*
b17	timeout	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	inf/l	w/k	inf/l	*UNSAT*	w/k	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l
b18	w/k	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*	*UNSAT*	inf/l	inf/l	w/k	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l
b19	timeout	*UNSAT*	*UNSAT*	*UNSAT*	inf/l	inf/l	*UNSAT*	*UNSAT*	inf/l	w/k	*UNSAT*	*UNSAT*	w/k	inf/l	*UNSAT*
MC8051	w/k	inf/l	inf/l	w/k	*UNSAT*	inf/l	w/k	inf/l	inf/l	w/k	inf/l	inf/l	w/k	inf/l	inf/l
AES-GCM	*UNSAT*	inf/l	inf/l	w/k	inf/l	inf/l	*UNSAT*	inf/l	*UNSAT*	w/k	inf/l	inf/l	w/k	*UNSAT*	inf/l
SPARC	*UNSAT*	*UNSAT*	inf/l	*UNSAT*	*UNSAT*	*UNSAT*	timeout	*UNSAT*	*UNSAT*	w/k	*UNSAT*	inf/l	w/k	*UNSAT*	inf/l

- All numbers are in *Seconds*. - timeout: 10^5 Seconds $\approx$ one day (Stop the attack process when time reaches timeout)
- UNSAT: Could not find the satisfiable assignment (The attack was not able to formulate the data flow obfuscated circuit correctly).
- inf/l: Infinite Loop (for stateful cyclic, the attack (cyclic-based) cannot find a decision for a stateful cycle).
- w/k: wrong key (The attack was not able to formulate, or the cycles lead to an incorrect formulation, both leading to a wrong key.

TABLE V

KEY SIZE AND OVERHEAD (OO) RELATION IN DIFFERENT SCENARIOS

Circuit	Key Size			Obfuscation (Area) Overhead	
	OO = 1%	OO = 5%	OO = 10%	Key Size = 100	Key Size = 200
s298	18	22	27	132.47%	306.71%
s526	20	24	31	68.20%	154.24%
s1423	17	20	32	27.71%	43.54%
s5378	24	34	54	12.67%	21.02%
s9234	29	37	67	6.29%	14.21%
s13207	31	95	201	5.52%	9.35%
s15850	27	88	176	4.26%	7.31%
s35932	52	221	472	1.34%	2.80%
s38584	47	237	460	1.24%	2.01%
b17	43	307	561	0.57%	0.90%
b18	137	408	835	0.12%	0.31%
b19	221	557	926	0.09%	0.17%
MC8051	34	107	271	5.51%	8.28%
AES-GCM	189	643	1017	0.21%	0.40%
SPARC	324	1004	1722	0.07%	0.14%

TABLE VI

DATA FLOW OBFUSCATION OVERHEAD (LOCKING OVERHEAD = 10%)

Circuit	Area $_{um^2}$			Max Delay $_{ns}$			Power $_{uW}$		
	original	locked	↑↓%	original	locked	↑↓%	original	locked	↑↓%
s298	152.1	165.2	8.62%	0.31	0.33	6.45%	14.17	15.34	8.23%
s526	284.3	307.4	8.16%	0.26	0.25	-3.84%	13.34	14.49	8.64%
1423	1018.7	1101.8	8.16%	1.22	1.25	2.45%	44.84	48.40	7.94%
s5378	2181.5	2316.5	6.19%	0.62	0.58	-6.4%	103.2	109.45	6.1%
s9234	2895.1	3019.0	4.28%	0.38	0.37	-2.65%	131.3	136.92	4.28%
s13207	5023.5	5361.0	6.72%	1.28	1.23	-3.9%	214.9	228.13	6.1%
s15850	6077.1	6404.0	5.38%	1.25	1.18	-5.5%	272.6	286.1	4.9%
s35932	15413.4	16657.3	8.07%	1.15	1.24	7.7%	1609.4	1730.18	7.5%
s38584	23118.6	24639.5	6.58%	1.14	1.13	-0.8%	1786.9	1902.15	6.4%
b17	46872.9	49319	5.22%	1.34	1.31	-2.24%	1927.6	2025.7	5.1%
b18	134829	139737	3.64%	1.82	1.83	0.55%	2269.9	2360.4	3.9%
b19	252945	262001	3.58%	1.97	1.94	-1.52%	2982.7	3070.1	2.9%
MC8051	4982.9	5474.7	9.87%	1.29	1.31	1.55%	188.6	206.3	9.3%
AES-GCM	105319	113681	7.94%	1.76	1.77	0.57%	1876.4	2018.8	7.6%
SPARC	298231	313650	5.17%	1.38	1.41	2.17%	3380.7	3533.4	4.5%

with a FF, 4) connecting FFs to a synchronous clock signal, and 5) adding an extra key-controlled MUX for each path that comes from the output of FFs. (For each MUX, input comes from the output of the FF, and one input is an extra key. The selector of the MUX is another key input.) Using these five steps, Fig. 11 shows the re-synchronized version of the obfuscated circuit from Fig. 8(b). Now, this re-synchronized version could be the input of the Sequential SAT attack. By using this model, if a path is a false path, then the logic value of this path is always fixed. So, the MUX must select the extra key input with corresponded value; however, if a path is an actual one, the other input of the MUX must be selected.

Regarding the latter versions of the deployed attack, on the other hand, no re-synchronization has been used, and since the asynchronous controller is in place with lots of combinational cycles, we replaced the traditional SAT attack with existing cyclic-SAT attacks, i.e., BeSAT and icySAT [14], [40].

B. Attack Results

Table IV shows the results of two attacks. For all cases, both attacks failed to break the obfuscated desynchronized circuit. The result of the attacks, in both versions, might return a wrong key, might return UNSAT, or might trap in an infinite loop. These three scenarios happen for a few main reasons: (1) regarding the wrong key (w/k) and UNSAT in S_SAT, the unrolling could not build the correct unrolled version due to asynchronicity; (2) regarding facing an infinite loop in S_BeSAT and S_icySAT, all are because of facing lots of stateful cycles in the asynchronous controller; and (3) regarding the UNSAT in S_BeSAT and S_icySAT, before facing an infinite loop, the solver is trapped in a wrong decision leading to UNSAT (because of incorrect formulation).

As shown in Table IV, in some rare cases, we see some numbers are struck out and replaced with *w/k* and *UNSAT* when we apply the S_SAT on re-synchronized circuits. In these cases, the S_SAT was able to find the correct key values. However, we found that re-timing did not relocate the latches after desynchronization for such cases. So, we force the re-timing step to do a minor relocation for a set of latches to eliminate the possibility of applying any form of re-synchronization. In this case, after enforcing those minor relocations, the S_SAT fails to break them. Also, for some cases, we face time-out (10^5 s), implicitly showing the complexity of SAT circuit. For all other cases, since we remove all stateful combinational cycles during re-synchronization, we faced only with w/k or UNSAT. Table IV implies that the existing attacks cannot formulate the proposed solution properly to break it, regardless of the

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

TABLE VII

AREA BREAKDOWN (LOCKING OVERHEAD = 10%)

Circuit	Combinational Logic (%)	FF (%)	Latches (%)	Delay Units (%)	Asynchronous Controller (%)
s13207	52.01%	33.80%	7.97%	3.59%	2.63%
s35932	42.76%	40.31%	9.51%	4.28%	3.14%
b17	59.77%	28.33%	6.68%	3.01%	2.21%
AES-GCM	44.06%	39.39%	9.30%	4.18%	3.07%
b18	69.67%	21.36%	5.04%	2.27%	1.66%
SPARC	59.33%	28.64%	6.76%	3.04%	2.23%

TABLE VIII

DATA FLOW OBFUSCATION OVERHEAD (KEY SIZE = 200)

Circuit	Area $_{um^2}$			Max Delay $_{ns}$			Power $_{uW}$		
	original	locked	↑↓%	original	locked	↑↓%	original	locked	↑↓%
s298	152.1	618.6	306.71%	0.31	0.32	3.7%	14.17	26.41	86.4%
s526	284.3	722.8	154.24%	0.26	0.28	8.2%	13.34	22.76	70.6%
s1423	1018.7	1462.2	43.54%	1.22	1.12	-8.3%	44.84	51.05	13.9%
s5378	2181.5	2640	21.02%	0.62	0.64	2.9%	103.2	114.48	10.9%
s9234	2895.1	3306.6	14.21%	0.38	0.36	-6.3%	131.3	141.35	7.7%
s13207	5023.5	5493	9.35%	1.28	1.25	-2.0%	214.9	227.50	5.9%
s15850	6077.1	6521.6	7.31%	1.25	1.16	-6.8%	272.6	281.37	3.2%
s35932	15413.4	15844.9	2.80%	1.15	1.25	8.6%	1609.4	1648.8	2.4%
s38584	23118.6	23583.1	2.01%	1.14	1.18	3.7%	1786.9	1814.7	1.6%
b17	46872.9	47295.4	0.90%	1.34	1.37	2.3%	1927.6	1944.7	0.89%
b18	134829	135242	0.31%	1.82	1.74	-4.1%	2269.9	2282.6	0.56%
b19	252945	253382	0.17%	1.97	1.98	0.5%	2982.7	2994.0	0.38%
MC8051	4982.9	5395.4	8.28%	1.29	1.23	-4.7%	188.6	201.4	6.83%
AES-GCM	105319	105741	0.40%	1.76	1.79	1.7%	1876.4	1894.6	0.97%
SPARC	298231	298644	0.14%	1.38	1.33	3.6%	3380.7	3390.5	0.29%

size/portion of the circuit, and regardless of the number of false paths inserted into the circuit.

Based on the two obfuscation metrics, we evaluated these deployed attacks in five different scenarios, in which a specific value for one of the metrics has been fixed. Table V shows that for each scenario with a fixed metric, what the value of the other metric is, which helps us to have an estimated relationship between these two metrics.

C. Area/Power/Delay Overhead Comparison

In this section, we evaluate the post-synthesis overhead of our data flow obfuscation. Table VI compares the power, performance (delay), and the area (PPA) of the original versus obfuscated circuits while the obfuscation overhead is set to 10%. 10% obfuscation overhead means that 10% of all FFs in a circuit must be converted to latches using desynchronization. Also, for any obfuscation overhead percentage, the number of extra {paths + latches} is set to be less than 10% of the total latches. Furthermore, the actual latches that are obfuscated using the same key gates (to prevent any key-guessing or structural attacks) are set to be less than 10% of the total latches. For example, for b17 with ∼1.5K FFs, for 10% obfuscation overhead, we replace 150 FFs with latches; we insert up to 15 false {paths + latches}, and up to 15 actual latches are obfuscated using the same key gates. PPA overhead in the data flow obfuscation is the consequence of two operations: 1) desynchronization and 2) adding false {paths + latches}. The overhead of desynchronization is dominant while the ratio of FFs to the total number of gates is higher in the original netlist. For instance, for s13207, whose FFs' ratio to all gates is $638/7951 = 8.03\%$, the area overhead is 6.72%. However, in s9234 with a ratio of 3.76%, the area overhead is only 4.28%. Table VII demonstrates the area breakdown of some of the circuits when the obfuscation overhead is 10%.

Regarding the delay overhead, since re-timing is used during desynchronization, in some cases we even achieved slight delay improvement. However, in some cases, it imposes only a very slight difference in cycle time by up to 8%. Regarding the power overhead, due to moving from edge-triggered design to level-triggered, the power overhead is less compared to area overhead. As seen in Table VI, our data flow obfuscation paradigm increased the power consumption by up to 10%.

TABLE IX

OVERHEAD COMPARISON BETWEEN DATA FLOW OBFUSCATION VERSUS STATE-OF-THE-ART OBFUSCATION TECHNIQUES

Circuit	Overhead	CAT 1	CAT 2		CAT 3		CAT 4		**Proposed**
		SFLL [26]	DLL [30]	Cyclic Obf. [29]	Full Lock [32]	Inter Lock [16]	R-DFS +SLL [35]	DisORC +TRLL [37]	**Data Flow Obf.**
s35932	Area ↑↓%	22.6%	5.3%	24.4%	28.1%	25.3%	9.7%	8.5%	**2.8%**
	Power ↑↓%	59.6%	3.7%	37.7%	34.9%	21.1%	14.7%	17.8%	**2.4%**
	Delay ↑↓%	0.5%	34.2%	17.5%	56.2%	5.8%	6.5%	8.7%	**8.6%**
s38584	Area ↑↓%	28.9%	4.7%	18.6%	26.9%	21.4%	10.4%	7.8%	**2.0%**
	Power ↑↓%	47.6%	2.4%	37.7%	30.1%	17.5%	14.8%	13.5%	**1.6%**
	Delay ↑↓%	0.8%	36.7%	13.7%	52.7%	14.2%	6.2%	9.5%	**3.7%**
b18	Area ↑↓%	5.6%	4.7%	13.9%	18.7%	3.4%	7.1%	2.7%	**0.3%**
	Power ↑↓%	11.6%	3.1%	28.8%	23.2%	1.8%	9.8%	6.9%	**0.6%**
	Delay ↑↓%	0.6%	39.9%	9.6%	53.9%	5.7%	5.9%	8.4%	**-4.1%**
b19	Area ↑↓%	2.7%	4.8%	9.7%	14.2%	2.8%	6.9%	1.6%	**0.2%**
	Power ↑↓%	6.2%	3.5%	22.3%	21.7%	2.2%	9.2%	1.2%	**0.4%**
	Delay ↑↓%	0.6%	44.7%	9.0%	52.6%	4.8%	5.4%	7.5%	**0.5%**
Overhead		Low	High Delay	High Power	Very High	High	Moderate	Moderate	Low
Attacked by		FALL [12]	SMT [15]	BeSAT [40]	CP&SAT [16]	–	Shift & Leak [35]	–	–

Table VIII compares the PPA of the original circuits versus obfuscated circuits when the key size is 200. As implied in Table V, in data flow obfuscation, for each extra {paths + latches}, as well as for any actual latch that are obfuscated to disable key-guessing, 3–5 keys could be added. So, when the key size is 200, regardless of the size of the circuit, 50–60 latches are needed. Accordingly, when the size of the key is 200, the area overhead is much higher in small circuits. However, for larger circuits, the ratio of false latches compared to the size of the circuit is significantly low, and since the real applications (ICs) are far larger than small circuits listed in Table III, the area overhead is low in this approach. For instance, in AES-GCM, the area overhead is even less than 1% (0.4%). To reflect a better evaluation of overhead, in Table IX, we compare the overhead of data flow obfuscation when the key size is set to 200 with state-of-the-art logic obfuscation techniques.[6] As shown, on average, the overhead incurred by data flow obfuscation is much lower compared to almost all techniques. In some techniques, the overhead of one metric might be better than that of data flow obfuscation, but on average, it could be concluded that the overhead of the proposed technique is completely acceptable.

D. Comparison With State-of-the-Art

Most recently, a new study has evaluated the possibility of latch-based architecture as a new means of logic obfuscation [60]. In this study, as shown in Fig. 12, key-programmable latches could be used as: 1) regular storage elements (FFs that are replaced with green latches subjected to re-timing), 2) programmable logic decoys (red latches/CLs) with constant output (always zero with no driving effect), and 3) programmable path delay decoy for delay manipulation (yellow

[6]Since the strength of the data flow obfuscation does not depend on the number of {paths + latches}, we could add as much as the key size (e.g., ≥64) to prevent breaking them using attacks like brute-force.

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

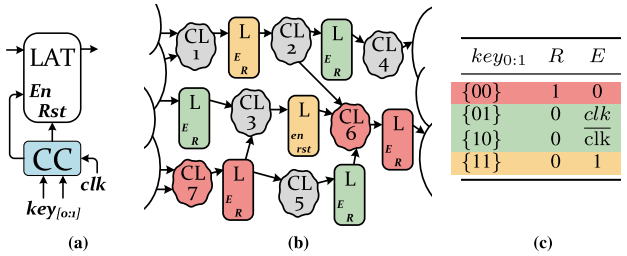

Fig. 12. Latch-based logic locking scheme [60]. (a) Key-controlled latches. (b) Example of latch-based logic locking. (c) Functions of key-controlled latches.

latches). However, unlike our proposed data flow obfuscation, it is still fully dependent on the clock signal (reset and enable of latches are a function of the clock signal) allowing us to convert this solution to a synchronous obfuscated problem. Clock-dependent latches operate as storage elements with synchronized gated clock, and due to this synchronicity, by integration with two pre-processing steps, it still could be modeled and broken using sequential SAT integrated with cyclic model: (step 1) Generating a single-clock synchronized locked circuit using a generalized/automated model, and (step 2) Detection of (some) programmable logic decoys with constant output through (pseudo-exhaustive) test patterns on testable points of the oracle.

Unlike latch-based logic locking that uses BMC with two copies of circuit per each clock cycle (due to level-triggering of latches) [60], we first generate a single-clock model to avoid this duplication per cycle. Fig. 13 shows we could build a single-clock fully synchronized circuit from the latch-based logic locked circuit. Based on the modes of latches demonstrated in Fig.12(c), Latches are replaced with a FF with two MUXes (one 2-to-1 preceding and one 4-to-1 following). The 4-to-1 MUX builds all three modes illustrated in Fig.12(c), and the 2-to-1 MUX is for keeping FF values when clk is not triggering (latching). Also, a 2-to-1 MUX will be added before each neighboring FF (immediate neighbors of latches). Now, all FFs are connected to a low-frequency generalized clock signal (clk_g). The selector of 2-to-1 MUXes of FFs corresponded to latches will be connected to the original clock signal, and the selector of that of neighboring FFs will be connected to a gated clock signal for falling (clk_f) or rising (clk_r) levels. This generalized model could be used for different scenarios with more complexities, such as multi-clock systems, and circuits with gated clock, all could be synchronized using generalized clock signal clk_g [61]. It allows us to use BMC with *ONLY* one copy of the circuit per each clock cycle, which improves the scalability significantly.

Also, since programmable logic decoys (red latches and red CLs) always generate constant output (zero output), and since testable pin are dedicated for latches (using extra MUXes and duplicate FFs) in the existing latch-based approach [60], the adversary would be able to apply test patterns (stuck-at-fault or pseudo-exhaustive) on testable points at cone-of-influence (COI) of oracle to detect latches with constant (zero) output. For some fundamental reasons, the programmable logic decoys could not be large enough to make this test infeasible: 1) programmable logic decoys are hardware overhead, which must be limited, 2) these logic decoys add difficulties to P&R which compromises the performance, and 3) it should not have an impact on maximum frequency of the circuit before adding obfuscation. Detecting these latches helps to

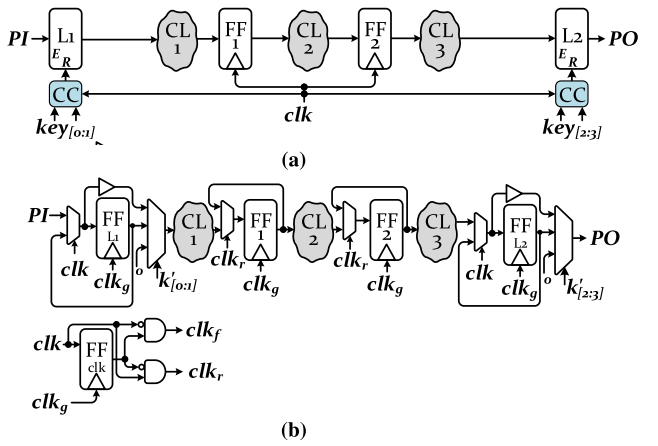

Fig. 13. Re-synchronization in latch-based logic locking. (a) Latch-based logic locked circuit. (b) Single-clock synchronized latch-based logic locked circuit.

TABLE X

RUNTIME OF RE-SYNCHRONIZATION + DECOY COI REDUCTION + SEQUENTIAL SAT INTEGRATED WITH BESAT (S_BESAT) ON THE EXISTING LATCH-BASED LOGIC LOCKING WITH KEY SIZE = 20, 50, 100, 200

Circuit	key size = 20	key size = 50	key size = 100	key size = 200
	exe time	exe time	exe time	exe time
s1423	468.5	521.6	321.1	1450.8
s5378	1422.1	1475.9	1855.6	2991.3
s15850	225.8	3006.5	2284.5	4712.6
s35932	3824.5	2071.9	10225.4	11452.8

timeout: 10^5 Seconds $\approx$ one day (Stop the attack when time reaches timeout)

TABLE XI

PROPOSED DATA FLOW OBFUSCATION VERSUS LATCH-BASED LOGIC LOCKING

Scheme	**Test Unit**	**False Latches**	**Clock Dependency**	**Synchronizable**
Proposed	Self Testable	Paired	Truly Asynchronous	✗
Latch-based [60]	Duplicate FFs	Single	Fully Clock-dependent	✓
Scheme	**BMC Model ?**		**Cyclic ?**	**Cyclic Model ?**
Proposed	✗		Stateful/Oscillating in Ctrls	hard
Latch-based [60]	✓ (with resynchronizing)		structural in datapath	easy
Scheme	**Overhead**		**Area*** ↑↓%	**Power*** ↑↓%	**Delay*** ↑↓%
Proposed	Latches + Ctrls		1.32%	1.25%	2.17%
Latch-based [60]	Latches + Ctrls + Decoy Logic		13.7%	5.4%	2.11%

∗: Average on s35932, s38584, b18, and b19 when key size = 200.

reduce the cone-of-influence, and consequently the SAT circuit before running the sequential SAT attack. Table X shows the execution time of the sequential SAT attack when it is integrated with these two pre-processing steps. However, since our proposed data flow obfuscation is truly desynchronized (token-based with no dependence on a clock signal), this form of re-synchronization is not applicable to it.

Table XI also shows some major advantages of the proposed solution against existing latch-based logic locking [60]. For example, we use a self-testable asynchronous controller; however, latch-based adds extra MUXes/FFs to make the latches testable, resulting in extra overhead. We add false latches *pair by pair*, which makes any structural/functional analysis exponentially harder (any pair of neighboring latches is a point of analysis); however, latch-based adds decoy latches *one by one*, which allows analyzing them linearly (each latch is a

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

point of analysis). In the proposed scheme, the latch enables controller consists of many stateful cycles that boost the difficulties for the adversary (no cyclic modeling). However, only easy-to-track structural cycles might be added into the existing latch-based logic locking when the key is not correct. Also, the overhead is much higher in the existing approach due to adding programmable logic decoys.

VII. Conclusion

To combat state-of-the-art attacks on logic obfuscation, in particular, the SAT attack and the sequential SAT, in this article, we introduced a new obfuscation paradigm called data flow obfuscation. By exploiting the concept of asynchronicity data flow obfuscation, we show how the flow of the data could be obfuscated in any arbitrary circuit. In data flow obfuscation, we engage false {paths + latches} using the asynchronous structure to control the flow of data in specific timing paths. Using this mechanism, we show that the SAT attack has no longer an advantage for the adversary even while the scan access is not restricted. Also, we showed that how asynchronicity combat the sequential SAT attack by invalidating the unrolling step in these attacks. We comprehensively investigated the effectiveness of this new obfuscation paradigm over wide-range benchmark families. Our experiments showed the resiliency of this new paradigm against all existing attacks at significantly low overhead.

References

[1] A. Yeh, "Trends in the global IC design service market," in *Proc. DIGITIMES*, 2012. [Online]. Available: http://www.digitimes.com/news/a20120313RS400.html?chid=2

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

[3] M. M. Tehranipoor, U. Guin, and S. Bhunia, "Invasion of the hardware snatchers," *IEEE Spectr.*, vol. 54, no. 5, pp. 36–41, May 2017.

[4] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: Ending piracy of integrated circuits," in *Proc. Design, Autom. Test Eur.*, Mar. 2008, pp. 1069–1074.

[5] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. 49th Annu. Design Autom. Conf. (DAC)*, 2012, pp. 83–89.

[6] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "Threats on logic locking: A decade later," in *Proc. Great Lakes Symp. (VLSI)*, May 2019, pp. 471–476.

[7] K. Shamsi, M. Li, K. Plaks, S. Fazzari, D. Z. Pan, and Y. Jin, "IP protection and supply chain security through logic obfuscation," *ACM Trans. Design Autom. Electron. Syst.*, vol. 24, no. 6, pp. 1–36, Nov. 2019.

[8] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2015, pp. 137–143.

[9] S. E. Quadir *et al.*, "A survey on chip to system reverse engineering," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 13, no. 1, pp. 1–34, Dec. 2016.

[10] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Removal attacks on logic locking and camouflaging techniques," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 2, pp. 517–532, Apr. 2020.

[11] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "AppSAT: Approximately deobfuscating integrated circuits," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2017, pp. 95–100.

[12] D. Sirone and P. Subramanyan, "Functional analysis attacks on logic locking," *IEEE Trans. Inf. Forensics Security*, vol. 15, pp. 2514–2527, 2020.

[13] H. Zhou, R. Jiang, and S. Kong, "CycSAT: SAT-based attack on cyclic logic encryptions," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 49–56.

[14] K. Shamsi, D. Z. Pan, and Y. Jin, "IcySAT: Improved SAT-based attacks on cyclic locked circuits," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2019, pp. 1–7.

[15] K. Z. Azar *et al.*, "SMT attack: Next generation attack on obfuscated circuits with capabilities and performance beyond the SAT attacks," in *Proc. IACR Trans. Cryptograph. Hardw. Embedded Syst.*, vol. 1, Spring 2019, pp. 97–122.

[16] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "InterLock: An intercorrelated logic and routing locking," in *Proc. 39th Int. Conf. Computer-Aided Design*, Nov. 2020, pp. 1–9.

[17] K. Z. Azar, H. M. Kamali, H. Homayoun, and A. Sasan, "NNgSAT: Neural network guided SAT attack on logic locked complex structures," in *Proc. 39th Int. Conf. Comput.-Aided Design*, Nov. 2020, pp. 1–9.

[18] M. E. Massad, S. Garg, and M. Tripunitara, "Reverse engineering camouflaged sequential circuits without scan access," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, Nov. 2017, pp. 33–40.

[19] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "KC2: Key-condition crunching for fast sequential circuit deobfuscation," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2019, pp. 534–539.

[20] N. Limaye, A. Sengupta, M. Nabeel, and O. Sinanoglu, "Is robust design-for-security robust enough? Attack on locked circuits with restricted scan chain access" in *Proc. Int. Conf. Comput.-Aided Design (ICCAD)*, 2019, pp. 1–8.

[21] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "On designing secure and robust scan chain for protecting obfuscated logic," in *Proc. Great Lakes Symp. (VLSI)*, Sep. 2020, pp. 1–6.

[22] L. Alrahis, M. Yasin, H. Saleh, B. Mohammad, M. Al-Qutayri, and O. Sinanoglu, "ScanSAT: Unlocking obfuscated scan chains," in *Proc. 24th Asia South Pacific Design Autom. Conf.*, Jan. 2019, pp. 352–357.

[23] N. Limaye and O. Sinanoglu, "DynUnlock: Unlocking scan chains obfuscated using dynamic keys," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2020, pp. 270–273.

[24] M. Yasin, B. Mazumdar, J. J. V. Rajendran, and O. Sinanoglu, "SARLock: SAT attack resistant logic locking," in *Proc. IEEE Int. Symp. Hardw. Oriented Secur. Trust (HOST)*, May 2016, pp. 236–241.

[25] Y. Xie *et al.*, "Mitigating sat attack on logic locking," in *Proc. Int. Conf. Cryptograph. Hardw. Embedded Syst. (CHES)*, 2016, pp. 127–146.

[26] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1601–1618.

[27] K. Shamsi, M. Li, T. Meade, Z. Zhao, D. Z. Pan, and Y. Jin, "Cyclic obfuscation for creating SAT-unresolvable circuits," in *Proc. Great Lakes Symp. (VLSI)*, May 2017, pp. 173–178.

[28] S. Roshanisefat, H. M. Kamali, and A. Sasan, "SRCLock: SAT-resistant cyclic logic locking for protecting the hardware," in *Proc. Great Lakes Symp. (VLSI)*, May 2018, pp. 153–158.

[29] A. Rezaei, Y. Shen, S. Kong, J. Gu, and H. Zhou, "Cyclic locking and memristor-based obfuscation against CycSAT and inside foundry attacks," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2018, pp. 85–90.

[30] Y. Xie and A. Srivastava, "Delay locking: Security enhancement of logic locking against IC counterfeiting and overproduction," in *Proc. 54th Annu. Design Autom. Conf. (DAC)*, Jun. 2017, pp. 1–6.

[31] K. Shamsi, M. Li, D. Z. Pan, and Y. Jin, "Cross-lock: Dense layout-level interconnect locking using cross-bar architectures," in *Proc. Great Lakes Symp. (VLSI GLSVLSI)*, May 2018, pp. 147–152.

[32] H. M. Kamali, K. Z. Azar, H. Homayoun, and A. Sasan, "Full-lock: Hard distributions of SAT instances for obfuscating circuits using fully configurable logic and routing blocks," in *Proc. 56th Annu. Design Autom. Conf. (DAC)*, Jun. 2019, pp. 89–94.

[33] R. Karmakar, S. Chatopadhyay, and R. Kapur, "Encrypt flip-flop: A novel logic encryption technique for sequential circuits," 2018, *arXiv:1801.04961*. [Online]. Available: http://arxiv.org/abs/1801.04961

[34] U. Guin *et al.*, "FORTIS: A comprehensive solution for establishing forward trust for protecting IPs and ICs," *ACM TODAES*, vol. 21, no. 4, p. 63, 2016.

[35] U. Guin, Z. Zhou, and A. Singh, "Robust design-for-security architecture for enabling trust in IC manufacturing and test," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 26, no. 5, pp. 818–830, May 2018.

[36] R. Karmakar *et al.*, "Efficient key-gate placement and dynamic scan obfuscation towards robust logic encryption," *IEEE Trans. Emerg. Topics Comput.*, early access, Dec. 31, 2019, doi: 10.1109/TETC.2019.2963094.

[37] N. Limaye *et al.*, "Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, early access, Oct. 6, 2020, doi: 10.1109/TCAD.2020.3029133.

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.

[38] X. Xu *et al.*, "Novel bypass attack and BDD-based tradeoff analysis against all known logic locking attacks," in *Proc. CHES*, 2017, pp. 189–210.
[39] S. Roshanisefat, H. Mardani Kamali, H. Homayoun, and A. Sasan, "SAT-hard cyclic logic obfuscation for protecting the IP in the manufacturing supply chain," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 4, pp. 954–967, Apr. 2020.
[40] Y. Shen, Y. Li, A. Rezaei, S. Kong, D. Dlott, and H. Zhou, "BeSAT: Behavioral SAT-based attack on cyclic logic encryption," in *Proc. 24th Asia South Pacific Design Autom. Conf.*, Jan. 2019, pp. 657–662.
[41] X. Wang, D. Zhang, M. He, D. Su, and M. Tehranipoor, "Secure scan and test using obfuscation throughout supply chain," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 37, no. 9, pp. 1867–1880, Sep. 2018.
[42] Y. Kasarabada, S. R. T. Raman, and R. Vemuri, "Deep state encryption for sequential logic circuits," in *Proc. IEEE Comput. Soc. Annu. Symp. VLSI (ISVLSI)*, Jul. 2019, pp. 338–343.
[43] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2013, pp. 709–720.
[44] S. M. Nowick *et al.*, "Asynchronous design—Part 1: Overview and recent advances," *IEEE Design Test*, vol. 32, no. 3, pp. 5–18, Mar. 2015.
[45] V. Khomenko *et al.*, "Logic synthesis for asynchronous circuits based on STG unfoldings and SAT," *Fundamenta Informaticae*, vol. 70, nos. 1–2, pp. 49–73, 2006.
[46] A. Moreno, D. Sokolov, and J. Cortadella, "Synthesis from waveform transition graphs," in *Proc. 25th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2019, pp. 60–67.
[47] S. Ataei and R. Manohar, "AMC: An asynchronous memory compiler," in *Proc. 25th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2019, pp. 1–8.
[48] M. Fiorentino *et al.*, "AnARM: A 28 nm energy efficient ARM processor based on octasic asynchronous technology," in *Proc. 25th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2019, pp. 58–59.
[49] E. Beigne *et al.*, "Asynchronous circuit designs for the Internet of everything: A methodology for ultralow-power circuits with GALS architecture," *IEEE Solid State Circuits Mag.*, vol. 8, no. 4, pp. 39–47, Fall 2016.
[50] D. Sokolov, V. Dubikhin, V. Khomenko, D. Lloyd, A. Mokhov, and A. Yakovlev, "Benefits of asynchronous control for analog electronics: Multiphase buck case study," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 1751–1756.
[51] J. Cortadella *et al.*, "A tool for manipulating concurrent specifications and synthesis of asynchronous controllers," *IEICE*, vol. 80, no. 3, pp. 315–325, 1997.
[52] T. Murata, "Petri nets: Properties, analysis and applications," *Proc. IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.
[53] J. Cortadella, A. Kondratyev, L. Lavagno, and C. P. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications," *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. TCAD*, vol. 25, no. 10, pp. 1904–1921, Oct. 2006.
[54] B. Van Antwerpen *et al.*, "Register retiming technique," U.S. Patent 7 120 883, Oct. 10, 2006.
[55] S. B. Furber and P. Day, "Four-phase micropipeline latch control circuits," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 4, no. 2, pp. 247–253, Jun. 1996.
[56] H. Hulgaard, S. M. Burns, and G. Borriello, "Testing asynchronous circuits: A survey," *Integration*, vol. 19, no. 3, pp. 111–131, Nov. 1995.
[57] A. Bouzafour, M. Renaudin, H. Garavel, R. Mateescu, and W. Serwe, "Model-checking synthesizable systemverilog descriptions of asynchronous circuits," in *Proc. 24th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2018, pp. 34–42.
[58] G. Tarawneh and A. Mokhov, "Formal verification of mixed synchronous asynchronous systems using industrial tools," in *Proc. 24th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2018, pp. 43–50.
[59] G. Gimenez, A. Cherkaoui, G. Cogniard, and L. Fesquet, "Static timing analysis of asynchronous bundled-data circuits," in *Proc. 24th IEEE Int. Symp. Asynchronous Circuits Syst. (ASYNC)*, May 2018, pp. 110–118.
[60] J. Sweeney, M. Zackriya V, S. Pagliarini, and L. Pileggi, "Latch-based logic locking," 2020, *arXiv:2005.10649*. [Online]. Available: http://arxiv.org/abs/2005.10649
[61] M. K. Ganai and A. Gupta, "Efficient BMC for multi-clock systems with clocked specifications," in *Proc. Asia South Pacific Design Autom. Conf.*, Jan. 2007, pp. 310–315.

Kimia Zamiri Azar received the B.Sc. degree in computer engineering from Khajeh Nasir Toosi University, Tehran, Iran, in 2013, and the M.Sc. degree in computer engineering from Shahid Beheshti University, Tehran, in 2015. She is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA.

Her research interests include the areas of hardware security and trust, security for supply chain, and VLSI design and test.

Hadi Mardani Kamali received the B.Sc. degree in computer engineering from Khajeh Nasir Toosi University, Tehran, Iran, in 2011, and the M.Sc. degree in computer engineering from the Sharif University of Technology, Tehran, Iran in 2013. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA.

His research focuses on hardware security, hardware/software acceleration applications, and power management in on-chip communication.

Shervin Roshanisefat received the M.Sc. degree in computer engineering from the University of Tehran, Iran, in 2015. He is currently pursuing the Ph.D. degree with the Department of Electrical and Computer Engineering, George Mason University, Fairfax, VA, USA.

He has worked earlier on telecommunication devices for SRD in Iran. His research interests include the areas of approximate computing, low power design, hardware-level functional safety, and security.

Houman Homayoun received the Ph.D. degree from the Department of Computer Science, University of California at Irvine, Irvine, CA, USA, in 2010.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of California at Davis, Davis, CA, USA.

He was the Technical Program Co-Chair of GLSVLSI 2018 and the General Chair of the 2019 GLSVLSI Conference. Since 2017, he has been serving as an Associate Editor for IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS.

Christos P. Sotiriou received the Ph.D. degree in computer science from The University of Edinburgh, Edinburgh, U.K., in 2001.

He is currently an Associate Professor with the Department of Electrical and Computer Engineering, University of Thessaly, Volos, Greece. His research interests include CAD algorithms and tools, and design methods and approaches for asynchronous timing. He served as the General Chair for the International Symposium on Advanced Research in Asynchronous Circuits and Systems for many years.

Avesta Sasan (Member, IEEE) received the B.Sc. degree in computer engineering and the M.Sc. and Ph.D. degrees in electrical and computer engineering from the University of California at Irvine, Irvine, CA, USA, in 2005, 2006, and 2010, respectively.

He worked at Broadcom Inc., and Qualcomm Company till 2016. He joined George Mason University in 2016, where he is currently serving as an Associate Professor for the Department of Electrical and Computer Engineering.

Authorized licensed use limited to: George Mason University. Downloaded on October 09,2021 at 06:24:31 UTC from IEEE Xplore. Restrictions apply.