TimingCamouflage: Improving Circuit Security against Counterfeiting by Unconventional Timing

Grace Li Zhang[1], Bing Li[1], Bei Yu[2], David Z. Pan[3] and Ulf Schlichtmann[1]
[1]Institute for Electronic Design Automation, Technical University of Munich (TUM), Munich, Germany
[2]CSE Department, The Chinese University of Hong Kong, Hong Kong
[3]ECE Department, University of Texas at Austin, Austin, TX, USA
Email: {grace-li.zhang, b.li, ulf.schlichtmann}@tum.de, byu@cse.cuhk.edu.hk, dpan@ece.utexas.edu

***Abstract*—With recent advances in reverse engineering, attackers can reconstruct a netlist to counterfeit chips by opening the die and scanning all layers of original chips. This relatively easy counterfeiting is made possible by the use of the standard simple clocking scheme where all combinational blocks function within one clock period. In this paper, we propose a method to invalidate the assumption that a netlist completely represents the function of a circuit. With the help of wave-pipelining paths, this method forces attackers to capture delay information from manufactured chips, which is a very challenging task because we also introduce false paths. Experimental results confirm that wave-pipelining paths and false paths can be constructed in benchmark circuits successfully with only a negligible cost, while the potential attack techniques can be thwarted.**

I. Introduction

Today's semiconductor business model involves many global vendors from various countries and regions. This distributed supply chain makes integrated circuits vulnerable to attacks and counterfeiting in nearly all phases from design to post-fabrication. Consequently, the research community has invested a great effort to deal with security challenges [1].

A major IC counterfeiting threat is the production of illegal chips by a third party with a netlist reverse engineered from authentic chips. In reverse engineering, authentic chips are delayered and imaged to identify logic gates, flip-flops, and their connections. Afterwards, the recognized netlist can be processed by a standard IC design flow and manufactured in a foundry, even with a different technology. This reverse engineering flow gives counterfeiters much freedom in reproducing authentic chips, because the recognized netlist carries all necessary design information and counterfeiters can revise and optimize it freely.

Several techniques have been proposed to thwart reverse engineering attacks on authentic chips. Firstly, IC camouflage tries to prevent the netlist from being recognized easily. In [2] transistors are manipulated with a stealthy doping technique during manufacturing so that they function differently than they appear. The work in [3]–[5] mixes real and dummy contacts to camouflage standard cells. The method in [6] explores netlist obfuscation by iterative logic fanin cone analysis at circuit level. Moreover, the method in [7] introduces a quantitative security criterion and proposes camouflaging techniques with a low-overhead cell library and an AND-tree structure. In addition, logic locking inserts additional logic gates, e.g., XOR/XNOR in [8], [9], AND/OR in [10] and MUX in [11], into the netlist to disable its function if the correct key is not applied. This method is expanded in [12] to incorporate delay information into the locking mechanism.

The methods discussed above all focus on either making the netlist more difficult to be recognized, or making the correct behavior of the circuit dependent on additional input information even after the netlist is recognized. In this paper, we propose a new perspective to counter counterfeiting based on reverse engineering. By integrating unconventional timing information, a netlist, even if recognized exactly through reverse engineering, does not function correctly anymore when a conventional timing scheme is assumed.

The advantages of the introduced method include:

- The camouflaged netlist only works with a given set of timing information, which, however, is difficult to be recognized exactly by reverse engineering even with much additional effort and cost.
- The camouflaged netlist only contains normal logic gates, so that it is challenging for attackers to isolate and then identify the timing encryption locations.
- The introduced wave-pipelining false paths obstruct test-based counterfeiting methods further by camouflaging originally testable paths as false paths.
- The proposed method is fully compatible with other security techniques introduced previously, so that they can be combined seamlessly.

The rest of this paper is organized as follows. In Section II, we explain the motivation and the basic idea of the proposed method. In Section III, we give a detailed description of the wave-pipelining technique. In Section IV, we analyze potential attack techniques and propose counter measures to thwart them. We describe the implementation details of constructing wave-pipelining paths and false paths in Section V. Experimental results are reported in Section VI. Conclusion is stated in Section VII.

II. Motivation and Basic Concept

Digital circuits rely on their structures to define their functions. A netlist is usually sufficient to reproduce a correctly working circuit. To prevent a netlist from being recognized by reverse engineering, techniques from physical level to netlist level can be applied to camouflage the logic. These methods, however, are still restricted to the conventional single-period clocking timing model so that attackers only need to recognize the netlist correctly.

In the conventional single-period clocking timing model, all the paths in a combinational block operate within one clock period. Figure 1(a) shows a part of a sequential circuit with three flip-flops F1, F2 and F3. At each sampling clock edge, assumed as the rising clock edge henceforth, the data at the inputs of the flip-flops are latched. To guarantee the correct operation of the flip-flops, the data at the input of a flip-flop must become stable t_{su} time before the rising clock edge, and the data must stay stable t_h time after the rising clock edge.

With the single-period clocking model, designers only need to guarantee that the logic functions of combinational blocks are correct without having to worry about the interaction between

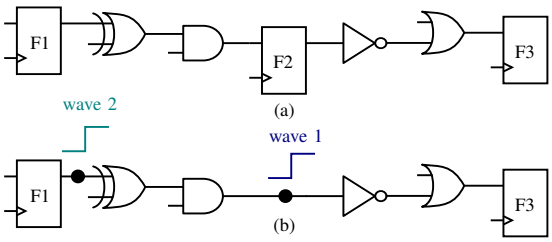

Figure 1: Conventional timing and wave-pipelining: (a) Single-period clocking; (b) Pipelining with two data waves.

different clock stages. Consequently, the netlist carries all logic information and this simplification allows attackers to counterfeit chips relatively easily because they only need to recognize the logic types of gates, flip-flops, and interconnect connections during reverse engineering.

To thwart the attack attempt on a design, we propose to invalidate the conventional timing model in some parts of the circuit. For example, we can remove the flip-flop in the middle of Fig. 1(a) to construct the circuit in Fig. 1(b). On the combinational path from F1 to F3, there are now two data waves without a flip-flop separating them. If the second wave does not catch the first one before it is latched by F3, the correct function of the circuit is still maintained. This technique is called *wave-pipelining (WP)* and has been investigated for circuit optimization [13]–[15]. When attackers recognize a netlist as in Fig. 1(b), they face the challenge to determine whether there should be one or two logic waves. If they assume the former and process the netlist using a standard EDA flow, the circuit loses synchronization because the data at the input of F3 is latched one clock period earlier. If they want to determine whether it is the latter case, additional effort is required to extract the timing information for the combinational path.

In the circuit in Fig. 1(b), at each rising clock edge, a new data is injected into the combinational path by flip-flop F1, so that the two waves are always separated by one clock period initially. To guarantee that the second data wave does not flush the first data wave when it is waiting for the next rising clock edge to be latched by F3, the path delay between F1 and F3 should be larger than one clock period. This path delay is, however, not contained in the extracted netlist from conventional reverse engineering. Consequently, the function of the circuit depends on both its structure and the timing of combinational paths.

Though attackers may have access to the standard cell library, e.g., through a third-party IP vendor, it is still very hard to obtain accurate interconnect/RC parasitics by delayering authentic chips, due to unknown process parameters, challenges in 3D RC extraction, and switching-window-dependent crosstalk-induced delay variations, etc. In any case, the more accurate the original timing information should be recognized from delayered chips, the harder and more expensive it becomes. In combination with other obfuscation methods, such as, dopant-level camouflage gate delay [2], [16] and dummy contact insertion [3]–[5], the unconventional timing concept has a potential to open up a new dimension of netlist security.

Although wave-pipelining paths look similar to multiple-cycle paths in digital design, the essential difference is that there is only one wave on a multiple-cycle path at a moment and the circuit still works if a multiple-cycle path is optimized to finish its calculation in one clock period, or if the clock frequency is lowered to make it work in one clock period. Therefore, multiple-cycle paths cannot be used to replace wave-pipelining paths to increase netlist security.

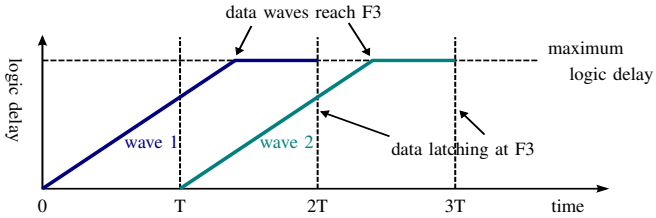

Figure 2: Temporal/spatial diagram for wave propagation on a combinational path.

III. Wave-Pipelining

A wave-pipelining path such as the one in Fig. 1(b) allows two data waves propagating on the path at the same time. Since the second data wave should not catch the first one, special timing constraints should be specified for this path. The scenario of data wave propagation is illustrated in Fig. 2. At first, wave 1 is injected into the path by F1. This data wave propagates along the path continuously and should reach F3 after the first rising clock edge at T and before the second rising clock edge at $2T$. At time $2T$, the first data is latched by F3. The second wave is injected by F1 at the rising clock edge at time T and it starts to propagate along the same path. Since this wave arrives at F3 with a delay larger than T, it does not catch the first wave at any time during the propagation, shown as the vertical gap between the two data waves in Fig. 2. Consequently, the two data waves on the path never interfere and F3 always latches the same value as in the original circuit shown in Fig. 1(a).

In forming wave-pipelining paths, a flip-flop is removed from the circuit as in the example from Fig. 1(a) to 1(b). In practice, this operation may lead to many paths with wave pipelining, because any combinational path reaching F2 together with any path starting from F2 forms a new wave-pipelining path. All these wave-pipelining paths should meet two constraints. First, the delay of a path should be larger than the clock period T; otherwise, the data wave is latched at the first rising clock edge instead of the second by F3. Second, the delay of the path should be no larger than $2T$ to guarantee that the data is latched by F3 in time. Assume the set of all these paths is P and the delay of a path $p \in P$ is d_p. The timing constraints for all these paths can be written as

$$d_p \geq T + t_h, \forall p \in P \iff \min_{p \in P}\{d_p - t_h\} \geq T \quad (1)$$

$$d_p \leq 2T - t_{su}, \forall p \in P \iff \max_{p \in P}\{d_p + t_{su}\} \leq 2T. \quad (2)$$

After removing a flip-flop from the circuit, if all the wave-pipelining paths meet the two constraints (1) and (2), the wave-pipelining version of the circuit is functionally equivalent to the original circuit.

IV. Attack Techniques and Counter Measures

In attacking a design with wave-pipelining, if attackers have no knowledge that this technique has been applied, the recognized netlist by reverse engineering does not function correctly. Once attackers become aware of this technique, various methods may be deployed to identify where the wave-pipelining paths are or to circumvent them simply. In *the assumed attack model, the available information includes a netlist recognized by reverse engineering and estimated delays of logic gates as well as interconnects with an inaccuracy factor τ. The objective of the attack is to identify on which combinational paths in the netlist wave-pipelining is applied.* The potential attack techniques are summarized in Fig. 3.

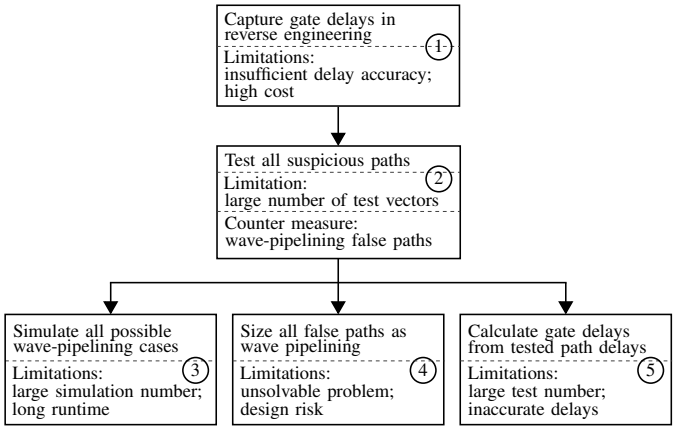

Figure 3: Attack techniques to identify or circumvent wave pipelining, where the last three techniques may be combined to reduce the problem space of attack.

The first attack technique is to measure all gate and interconnect delays while the netlist is recognized by reverse engineering. With all gate and interconnect delays known, path delays can be calculated from the netlist easily. Since the delays of wave-pipelining paths are between T and $2T$ as defined in (1) and (2), these paths can therefore be identified. The challenge of this attack technique is that it is difficult to extract accurate gate and interconnect delays just from reverse engineering. Assume that the real delay of a path is d and the delay recognition technique suffers an inaccuracy factor τ ($0 < \tau < 1$). Consequently, this path delay can be any value in the range $[(1-\tau)d, (1+\tau)d]$ when recognized. If the upper bound of a path delay is smaller than T, this path is definitely a single-period clocking path. If the lower bound of a path delay is larger than T, the path is definitely a wave-pipelining path. However, if a path delay covers the clock period T, namely,

$$(1-\tau)d \leq T \leq (1+\tau)d. \tag{3}$$

This path can only be considered as suspicious of wave-pipelining but without a clear differentiation. In the following, we call the range $[(1-\tau)d, (1+\tau)d]$ the *gray region* for a path with delay d. In reality, a well-optimized design contains many critical paths with delays close to the clock period T so that their gray regions cover T easily. When constructing wave-pipelining paths in the proposed method, we also guarantee that their delays are in the gray region.

With the estimated delays, attackers can actually narrow down the number of potential wave-pipelining paths, because paths with delays definitely smaller or larger than T considering the inaccuracy in delay estimation can be screened out. *The second attack technique* is to test the delays of the remaining paths using authentic chips from the market. With the netlist recognized, it is not difficult to determine test vectors to trigger the suspicious paths. Since the only information of interest is whether a path delay is larger than T, only one delay test for each path is sufficient. Without considering the cost to test many paths, this test strategy is in fact able to differentiate wave-pipelining paths from other paths eventually.

To prevent all suspicious paths from being tested, we introduce a counter measure to create unsensitizable paths with wave-pipelining. When we construct wave-pipelining paths by removing flip-flops, we prefer the paths that, viewed directly with the conventional single-period clocking, are false paths, which cannot be sensitized by any test vectors.

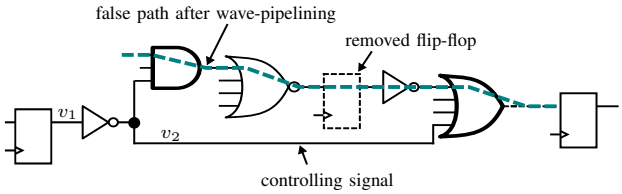

Figure 4: Two true paths form a wave-pipelining false path.

Definition 1. *False Path: A combinational path which cannot be activated in functional mode or test due to controlling signals from other paths [17], [18]. On the contrary, true paths can be activated in functional mode or test.*

Definition 2. *Wave-Pipelining False Path (WP False Paths): A combinational path with wave pipelining that is a false path when viewed with the conventional single-period clocking.*

Wave-pipelining false paths are true paths with data waves propagating along them when the circuit is running, but they are false paths when the netlist is examined only. An example of wave-pipelining false paths is shown in Fig. 4, which is a snippet of the s298 circuit from the ISCAS89 benchmark set. When the flip-flop in the middle is removed, the dashed path becomes a wave-pipelining path and also a false path, if it is considered as working within a single clock period. In this case, a signal switching at the beginning of the dashed path never reaches the final flip-flop. If the signal v_2 has a value '1', which is the controlling signal to an OR gate, it blocks the dashed path at the last OR gate; if the signal v_2 has a value '0', it blocks the dashed path at the AND gate right away. Consequently, the dashed path cannot be triggered for delay test and attackers have no way to differentiate it from all the other false paths in the original circuit, which may contribute up to 75% of all the combinational paths in real circuits [19].

Since the delays of false paths cannot be tested, *the third attack technique*, brute-force logic simulation, could be considered to differentiate the camouflaged false paths from real false paths. In this method, each false path that cannot be excluded by delay screening in the first step is assumed to be a real false path once and a wave-pipelining false path once. Assuming the number of such paths is n, then 2^n simulations of the complete circuit should be performed to check which combination is correct. In theory, this method can eventually find the correct combination of real false paths and wave-pipelining false paths. However, it is still impractical because of the unaffordable simulation time due to the large number of false paths in the original design [17], [19] and the very long runtime for a full simulation of the complete circuit.

The fourth technique to attack wave-pipelining false paths is to consider all false paths in the circuit as wave-pipelining paths and size logic gates so that delays of all these paths meet the constraints (1) and (2). The concept behind this technique is that false paths are not triggered anyway so that they do not affect the logic of the circuit if their delays are larger than the clock period T. This assumption, however, is too optimistic because false paths sized to have delays larger than T may still affect the normal circuit operation [18]. Another challenge of this attack technique is that it is very difficult to find a solution to size so many false paths without affecting the normal true paths whose delays should be smaller than T.

The fifth technique to identify wave-pipelining paths is to calculate all gate delays in a circuit from path delays measured by at-speed test, such as applied in [20]. Since path delays are linear

combinations of gate delays, the measured path delays can be used to calculate gate delays by linear algebra. The challenges of this method are: 1) a large number of combinational paths should be tested in a commercial design; 2) all logic gates should appear on testable paths in a way that the coefficient matrix of linear equations has a rank equal to the number of gate/interconnect delays, even in view of a large percentage of false paths [17], [19]; 3) inaccuracy in at-speed test of path delays due to environmental factors such as noise and temperature as well as the nature of binary-search of at-speed delay test.

V. Wave-pipelining Construction

When constructing wave-pipelining paths into a circuit while maintaining its original function, we need to guarantee that the constructed paths meet the timing constraints (1) and (2). To counter the attack techniques discussed in Section IV, the constructed paths should not be screened out easily by delay test and estimation. Furthermore, the constructed wave-pipelining paths should contain false paths when considered as single-period clocking paths. The wave-pipelining construction problem can thus be formulated as follows.

Inputs: An optimized design; delay information; the given clock period T; the delay recognition inaccuracy factor τ ($0 < \tau < 1$); the required numbers of wave-pipelining true and false paths n_{wpt} and n_{wpf}.

Outputs: A revised design containing at least the given numbers of wave-pipelining true and false paths. The delays of these wave-pipelining paths should meet the gray region requirement (3).

Objectives: The original design should be kept unchanged as much as possible; the increased resource usage should be as little as possible.

V-A. Work flow of wave-pipelining construction

The major steps to construct wave-pipelining paths are shown in Fig. 5. To construct wave-pipelining false paths, we visit flip-flops in the netlist iteratively. At each flip-flop ff_i, we check whether there are wave-pipelining false paths formed from single-period true paths on the left and on the right of ff_i. The number of such paths is stored in n_f as shown in L5. Thereafter, we construct wave-pipelining false paths at this flip-flop with the function construct_WP_paths(ff_i, T, τ) which will be explained later.

As shown in Fig. 1(b), a wave-pipelining path requires that the flip-flop at the beginning of the path and the flip-flop at the end of the path are kept in the circuit. These fanin and fanout flip-flops are inserted into the set F_w and all the flip-flops tracked by F_w cannot be considered as candidates to construct wave-pipelining paths.

In the last step of our method, we construct additional wave-pipelining paths that are still true when viewed with the single-period clocking model. These paths are used to guarantee that attackers must test all single-period clocking or wave-pipelining true paths whose delays are in the gray region. Without these paths, attackers can assume all testable paths are clocked by a single clock period and skip the expensive test procedure. The path construction in this step is nearly the same as L3–L11 in Fig. 5. The only differences are that at L5 we should check wave-pipelining true paths and in L9 and L10 we should use n_{wpt} as the number of such paths to be constructed.

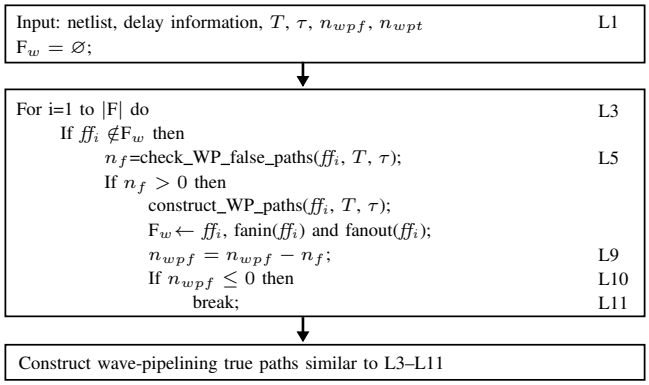

Figure 5: Major steps of wave-pipelining construction.

V-B. False path checking

In the work flow above, we need to check whether a path is a false path. In our method, we consider the statically unsensitizable paths as false paths [21], [22], such as the false path shown in Fig. 4. In this example, the path cannot be sensitized because the controlling signal blocks either the AND gate or the last OR gate no matter what its value is.

To verify whether a path is statically unsensitizable, we assign a Boolean variable to the output of each gate and formulate false path checking as a SAT problem [22]. The logic relations between these variables are established according to functions of logic gates. If a path can be sensitized, all the side inputs of the path must be set to the non-controlling values. For example, the path in Fig. 4 requires that the condition $(v_2 \wedge \neg v_2)$ is true, which is, however, always false. In implementing the function check_WP_false_paths(ff_i, T, τ) in Fig. 5, we randomly select 500 paths that drive the current flip-flop ff_i and exclude the false paths from them, because the wave-pipelining paths to be constructed should be formed by two single-period clocking true paths. Similarly we select 500 paths that are driven by ff_i and exclude the false paths. The selected number is in fact abundant in the circuits as demonstrated by experimental results in Section VI. The concept of this path selection is illustrated in Fig. 6(a).

V-C. Wave-pipelining path construction

At flip-flop ff_i, we need to construct wave-pipelining paths in the circuit with the function construct_WP_paths(ff_i, T, τ) in Fig. 5. Unfortunately, the intuitive idea to remove flip-flop ff_i in the middle is not a viable solution, because there usually are many short paths on the left and on the right of ff_i and connecting them directly generates many paths whose delays are too small to meet the lower bound of the path delay constraint (1).

To solve this problem, we duplicate the logic in the circuit and size the gates so that the delays of all wave-pipelining paths meet (1) and (2) as illustrated in Fig. 6(b). In the duplicated circuit on the right of ff_i, we only keep the flip-flops at which wave-pipelining paths terminate. The other flip-flops stay in the original circuit. Afterwards, we delete the logic gates backwards to remove those gates that do not drive any flip-flop to reduce resource usage. When duplicating the logic on the left of ff_i, however, we need to keep all the logic gates to maintain the correct function of the circuit.

In the duplicated logic in Fig. 6(b), we do not duplicate flip-flop ff_i. Therefore, all combinational paths in the duplicated logic are wave-pipelining paths and their delays should meet the gray region requirement (3) as well as (1)–(2). To meet these constraints, we size the gates in the duplicated logic with an ILP

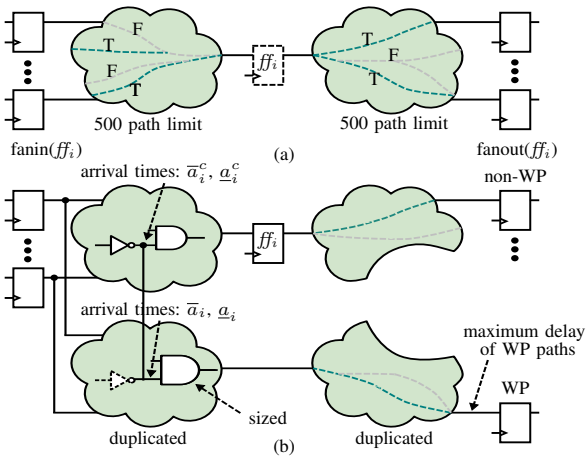

Figure 6: WP path construction. (a) The number of paths on each side of ff_i is limited to 500. A WP false path is constructed by two single-period clocking true paths. (b) Logic duplication and gate sizing.

formulation. In this formulation, we assign two variables to a pin of a logic gate to represent the latest and earliest arrival times, respectively. Assume that an input pin of a gate is indexed by i and the variables are written as $\overline{a}_i$ and $\underline{a}_i$. Similarly, assume that the output pin of the gate is indexed by j and the two variables are $\overline{a}_j$ and $\underline{a}_j$. Furthermore, the gate delay from an input pin to the output pin is written as d_{ij}, which is a variable since the corresponding logic gate is sized. With these definitions, the arrival time constraints from an input pin to the output pin can be written as

$$\overline{a}_j \geq \overline{a}_i + d_{ij} \tag{4}$$
$$\underline{a}_j \leq \underline{a}_i + d_{ij}. \tag{5}$$

To reduce the number of duplicated gates, we try to connect the input pins of logic gates in the duplicated logic to the original gates as much as possible, as illustrated in Fig. 6(b). In the original logic, the latest and the earliest arrival times are constants. Assume that the two arrival times to the original counterpart of an input pin are $\overline{a}_i^c$ and $\underline{a}_i^c$, and a 0-1 variable p_i indicates whether the input pin in the duplicated logic should be driven by the original logic. We can then extend the constraints (4)–(5) as

$$\overline{a}_j \geq \overline{a}_i + d_{ij} - p_i M \tag{6}$$
$$\overline{a}_j \geq \overline{a}_i^c + d_{ij} - (1 - p_i) M \tag{7}$$
$$\underline{a}_j \leq \underline{a}_i + d_{ij} + p_i M \tag{8}$$
$$\underline{a}_j \leq \underline{a}_i^c + d_{ij} + (1 - p_i) M, \tag{9}$$

where M is a very large positive constant used to transform the conditional constraints to linear constraints [23]. In either case when the input pin is connected or disconnected in the duplicated logic, only two constraints in (6)–(9) are valid.

In the description above, we do not bound gate delays strictly. Instead, we allow them to exceed the maximum gate delays defined in the library, respectively, so that the path delay constraints (1)–(2) and the gray region constraint (3) can be guaranteed. However, we try to keep the increased gate delays as small as possible, so that they can be absorbed by interconnect delays during physical design. To reduce resource usage and avoid excessive delay padding, we formulate the optimization problem as

$$\text{minimize} \quad \alpha \sum_I d_{ij} - \beta \sum_I p_i \tag{10}$$
$$\text{subject to} \quad (1)\text{–}(2),\ (3),\ (6)\text{–}(9), \tag{11}$$

Table I: Results of Constructing WP Paths

Circuit			WP Cons.					Runtime
	n_s	n_g	n_t	n_{wpt}	n_{wpf}	n_d	n_p	$t_r(s)$
s35932	1728	16065	180039	20	1022	178	80	625.29
s38584	1452	19253	502561	48	431	130	117	3685.88
s38417	1636	22179	298922	82	63	321	65	1711.01
s15850	522	9772	361544	20	838	186	141	3018.06
s13207	669	3716	927424	20	115	152	74	446.17
s9234	228	5597	10922	20	983	148	83	291.45
s5378	179	2779	10143	401	78	139	55	266.022
s4863	104	2342	4140	680	0	184	77	3766.98
s1423	74	657	8506	450	12	75	213	1170.71
s1238	12	508	15	3	4	94	90	2.07

where $\alpha \gg \beta$ and I is the index set of all input pins. After the ILP problem above is solved, the gates that do not drive any other gates in the duplicated logic are removed from the circuit.

VI. Experimental Results

The proposed method was implemented in C++ and tested using a 3.20 GHz CPU. We demonstrate the results using circuits from the ISCAS89 benchmark set. The number of flip-flops and the number of logic gates are shown in the columns $\boldsymbol{n_s}$ and $\boldsymbol{n_g}$ in Table I, respectively. The benchmark circuits were sized using a 45 nm library. We kept 15% of timing margin to tolerate PVT (Process, Voltage and Temperature) variations and we set the inaccuracy factor τ of delay estimation in (3) to 20%. We used Gurobi [24] to solve the optimization problems in the proposed method.

The results of wave-pipelining path construction are shown in Table I. The column $\boldsymbol{n_t}$ shows the number of single-period clocking combinational paths that are true paths in the original circuits and whose delays meet the gray region requirement (3). When attackers try to detect the locations of wave-pipelining paths, these true paths need to be tested to determine whether their delays are actually larger or smaller than T. These results show that attackers need to perform many expensive test iterations to attack a chip even if they can estimate gate delays to some degree.

The column $\boldsymbol{n_{wpt}}$ shows the numbers of wave-pipelining true paths whose delays are in the gray region. These paths are used to guarantee that attackers must test all single-period clocking or wave-pipelining true paths whose delays are in the gray region. The column $\mathbf{n_{wpf}}$ shows the numbers of wave-pipelining false paths whose delays are in the gray region. These paths are used to obstruct the attempt that attackers test all paths to determine the wave-pipelining paths. In our experiments, we set the target numbers of wave-pipelining true and false paths both to 10. We executed the construction of wave-pipelining true and false paths shown as in Fig. 5. When we constructed wave-pipelining false paths using the technique illustrated in Fig. 6, we also found wave-pipelining true paths in the duplicated circuit snippet. In addition, we found wave-pipelining false paths in the circuit snippet duplicated to construct wave-pipelining true paths. Consequently, the numbers of these paths shown in the columns n_{wpt} and n_{wpf} are larger than 10 for many test cases except s4863 and s1238. In s4863 there is no wave-pipelining false path and in s1238 the numbers of wave-pipelining paths are very small due to the limited circuit size. In all the large test cases, however, wave-pipelining paths have been constructed successfully,

The column $\boldsymbol{n_d}$ in Table I shows the number of logic gates duplicated in the final circuits. Since we only inserted wave-pipelining paths at limited locations, generally the number of duplicated gates does not increase with respect to circuit size. The column $\boldsymbol{n_p}$ shows the number of delay units equivalent to

Table II: Wave-pipelining False Paths in Test Cases

Circuit	n_f	$\tau = 0.2$	$\tau = 0.1$
s5378	122757	80386	4845
s4863	0	0	0
s1423	2331927	58992	37312
s1238	392	0	0

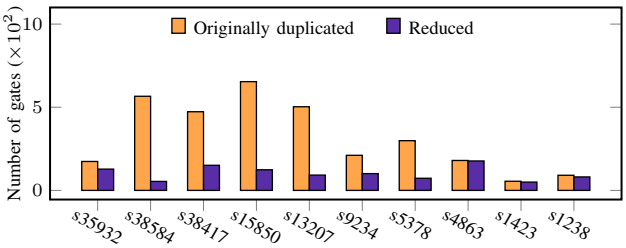

Figure 7: Comparison of gate numbers before/after reduction.

buffer delays that were inserted to extend wave-pipelining path delays. Since the number of duplicated gates does not increase with respect to circuit size, the area cost for constructing wave-pipelining paths is negligible in relatively large circuits. The last column t_r in Table I shows the runtime of the proposed method, which is acceptable because wave-pipelining construction is a one-time effort.

In the proposed method, the number of wave-pipelining false paths depends on the original circuit structure. If there is no such a path in a circuit, we cannot use this technique to thwart test-based attack. To verify whether this feature is common for most circuits, we checked the numbers of wave-pipelining false paths in the test cases and the results are shown in Table II, where the column n_f shows the numbers of wave-pipelining false paths without considering path delays. The columns $\tau = 0.2$ and $\tau = 0.1$ show the numbers of such paths with delays meeting the gray region requirement (3). Since $\tau = 0.1$ means that the gray region is smaller, the numbers of wave-pipelining paths under this condition decrease compared with the $\tau = 0.2$ cases. For all the other test cases not appearing in Table II, the numbers of such paths corresponding to the three columns are all larger than 100k, meaning that there are plenty of wave-pipelining false paths which can be used to camouflage the timing of these circuits.

In our wave-pipelining construction formulation (10)–(11), we maximize the number of signals that can be driven by the original circuit as illustrated in Fig. 6. Consequently, the number of logic gates in the duplicated circuit can be reduced. Fig. 7 compares the numbers of gates in the originally duplicated circuit before the removed flip-flop in Fig. 6 and the number of gates after reduction. In all the test cases, the numbers of duplicated gates were reduced significantly.

In our experiments, we also simulated the gate sizing attack on the netlist as discussed in Section IV. The basic idea was that all false paths whose delays were in the gray region were treated as wave-pipelining paths and their delays were sized to meet (1)–(2). The results of this simulated attack are shown in Fig. 8, where the first bar shows the number of false paths we used to simulate the attack. The last bar shows the number of false paths that were not sized successfully. In all these simulation cases, no sizing attack succeeded. As discussed in Section IV, false paths may be sensitized if their delays exceed one clock period. The second bar in Fig. 8 shows the number of the false paths that can be sensitized when considered as wave-pipelining paths in the attack. Obviously many of them can be sensitized so that the circuit does not work even if the sizing attack could succeed.

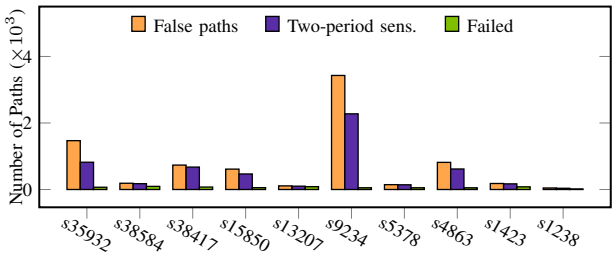

Figure 8: Results of false path sizing attack.

VII. Conclusion

In this paper, we have proposed a new timing camouflage technique to secure circuit netlists against counterfeiting. Since a netlist itself does not carry all design information anymore, the difficulty of attack has been increased significantly due to additional test cost and the introduced wave-pipelining false paths. This technique potentially opens up a new dimension of circuit security and it is fully compatible with all previous anti-counterfeiting methods. Future work includes incorporating gate delay camouflage by doping modification [2], [16] to decouple gate delays from layout further. In addition, clock skew scheduling in [25]–[28] would also be explored in the same timing dimension to enhance the security of netlists.

Acknowledgement

This work was partially supported by Fraunhofer High Performance Center Connected Secure Systems Munich.

References

[1] U. Guin, K. Huang, D. DiMase, J. M. Carulli, M. Tehranipoor, and Y. Makris, "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proc. IEEE*, vol. 102, no. 8, pp. 1207–1228, 2014.
[2] G. T. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans: extended version," *J. Cryptographic Engineering*, vol. 4, no. 1, pp. 19–31, 2014.
[3] S. Malik, G. T. Becker, C. Paar, and W. P. Burleson, "Development of a layout-level hardware obfuscation tool," in *Comput. Society Ann. Symp. on VLSI*, 2015, pp. 204–209.
[4] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security analysis of integrated circuit camouflaging," in *Proc. Conf. on Comput. & Commun. Security*, 2013, pp. 709–720.
[5] J. Rajendran, O. Sinanoglu, and R. Karri, "VLSI testing based security metric for IC camouflaging," in *Proc. Int. Test Conf.*, 2013, pp. 1–4.
[6] Y. Lee and N. A. Touba, "Improving logic obfuscation via logic cone analysis," in *Latin-American Test Symp.*, 2015, pp. 1–6.
[7] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan, "Provably secure camouflaging strategy for IC protection," in *Proc. Int. Conf. Comput.-Aided Des.*, 2016, pp. 28–35.
[8] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proc. Design Autom. Conf.*, 2012, pp. 83–89.
[9] J. A. Roy, F. Koushanfar, and I. L. Markov, "EPIC: ending piracy of integrated circuits," in *Proc. Design, Autom., and Test Europe Conf.*, 2008, pp. 1069–1074.
[10] S. Dupuis, P. Ba, G. D. Natale, M. Flottes, and B. Rouzeyre, "A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans," in *Int. On-Line Testing Symp.*, 2014, pp. 49–54.
[11] S. M. Plaza and I. L. Markov, "Solving the third-shift problem in IC piracy with test-aware logic locking," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 6, pp. 961–971, 2015.
[12] A. S. Yang Xie, "Delay locking: Security enhancement of logic locking against ic counterfeiting and overproduction," in *Proc. Design Autom. Conf.*, 2017.
[13] H.-Y. Hsieh, W. Liu, R. K. Cavin, III, and C. T. Gray, "Concurrent timing optimization of latch-based digital systems," in *Proc. Int. Conf. Comput. Des.*, 1995, pp. 680–685.
[14] W. P. Burleson, M. Ciesielski, F. Klass, and W. Liu, "Wave-pipelining: A tutorial and research survey," *IEEE Trans. VLSI Syst.*, vol. 6, no. 3, pp. 464–474, Sep. 1998.
[15] G. Seetharaman and B. Venkataramani, "Automation schemes for FPGA implementation of wave-pipelined circuits," *ACM Trans. Reconf. Tech. Sys.*, vol. 2, no. 2, 2009.
[16] A. V. Vinay C. Patil and S. Kundu, "Manufacturer turned attacker: Dangers of stealthy trojans via threshold voltage manipulation," in *IEEE North Atlantic Test Workshop(NATW)*, 2017, pp. 1–6.
[17] F. Yuan and Q. Xu, "On timing-independent false path identification," in *Proc. Int. Conf. Comput.-Aided Des.*, 2010, pp. 532–535.
[18] D. H. Du, S. H. Yen, and S. Ghanta, "On the general false path problem in timing analysis," in *Proc. Design Autom. Conf.*, 1989, pp. 555–560.
[19] K. Heragu, J. H. Patel, and V. D. Agrawal, "Fast identification of untestable delay faults using implications," in *Proc. Int. Conf. Comput.-Aided Des.*, 1997, pp. 642–647.
[20] K. Vaidyanathan, B. P. Das, and L. T. Pileggi, "Detecting reliability attacks during split fabrication using test-only BEOL stack," in *Proc. Design Autom. Conf.*, 2014, pp. 156:1–156:6.
[21] S. Devadas, K. Keutzer, and S. Malik, "Computation of floating mode delay in combinational circuits: Theory and algorithms," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 12, pp. 1913–1923, Nov. 2006.
[22] O. Coudert, "An efficient algorithm to verify generalized false paths," in *Proc. Design Autom. Conf.*, 2010, pp. 188–193.
[23] D. Chen, R. Batson, and Y. Dang, *Applied Integer Programming: Modeling and Solution*. Wiley, 2011.
[24] Gurobi Optimization, Inc., "Gurobi optimizer reference manual," 2013. [Online]. Available: http://www.gurobi.com
[25] B. Li and U. Schlichtmann, "Statistical timing analysis and criticality computation for circuits with post-silicon clock tuning elements," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 34, no. 11, pp. 1784–1797, 2015.
[26] G. L. Zhang, B. Li, and U. Schlichtmann, "Sampling-based buffer insertion for post-silicon yield improvement under process variability," in *Proc. Design, Autom., and Test Europe Conf.*, 2016, pp. 1457–1460.
[27] G. L. Zhang, B. Li, J. Liu, Y. Shi, and U. Schlichtmann, "Design-phase buffer allocation for post-silicon clock binning by iterative learning," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, 2017, doi: 10.1109/TCAD.2017.2702632.
[28] G. L. Zhang, B. Li, and U. Schlichtmann, "EffiTest: Efficient delay test and statistical prediction for configuring post-silicon tunable buffers," in *Proc. Design Autom. Conf.*, 2016, pp. 60:1–60:6.