ABSTRACT

Title of Thesis: ATTACKING THE TIMINGCAMOUFLAGE+ ALGORITHM

Priya Devi Mittu
Master of Science, 2021

Thesis Directed by: Professor Ankur Srivastava
Department of Electrical and Computer Engineering

In today's world, sending a design to a third party foundry for fabrication poses a serious threat to one's intellectual property. To keep designs safe from adversaries, design obfuscation techniques have been developed to protect the IP details of the design. This thesis explains how the previously considered secure algorithm, TimingCamouflage+, can be thwarted and the original circuit can be recovered [1]. By removing wave-pipelining false paths, the TimingCamouflage+ algorithm is reduced to the unsecure TimingCamouflage algorithm [2]. Since the TimingCamouflage algorithm is vulnerable to the TimingSAT attack, this reduction proves that TimingCamouflage+ is also vulnerable to TimingSAT and not a secure camouflaging technique [3]. This thesis describes how wave-pipelining paths can be removed, and this method of handling false paths is tested on various ISCAS89 benchmarks and shown to be both functionally correct and feasible in complexity.

ATTACKING THE TIMINGCAMOUFLAGE+ ALGORITHM

by

Priya Devi Mittu

Thesis submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Master of Science
2021

Advisory Committee:
Professor Ankur Srivastava, Chair/Advisor
Professor Manoj Franklin
Professor Dana Dachman-Soled

Acknowledgments

I would like to thank all the people who have made this thesis possible.

First I would like to thank my advisor, Professor Ankur Srivastava, mentor, Dr. Yuntao Liu, and Dr. Abhishek Chakraborty for their advice and guidance through this process. They have always made time to help me learn and grow as a student and person. It has been an honor to work with such intelligent individuals.

I would like to thank both Professor Manoj Franklin and Professor Dana Dachman-Soled for participating on my thesis committee. I appreciate the time taken to review my work and provide valuable feedback.

Finally, I would like to thank my family and friends for their unwavering support. I would like to thank my mom and dad, Anjali, Nick, Rohan, and Erik for always being there to get me through stressful periods and push me to be the best I could be. Thank you for the love and encouragement and always believing in me. I would not be where I am today without you all, and I am immensely grateful and privileged to have you in my life.

Table of Contents

Acknowledgements ii

Table of Contents iii

List of Figures iv

Chapter 1: Introduction 1
- 1.1 Security Issue . . . 1
- 1.2 Obfuscation . . . 1
- 1.3 Overview of Thesis . . . 3

Chapter 2: Background 4
- 2.1 Logic Obfuscation Techniques . . . 4
- 2.2 TimingCamouflage . . . 5
- 2.3 TiminigSAT Attack . . . 6

Chapter 3: TimingCamouflage+: An Improvement on TimingCamouflage 8
- 3.1 Types of Paths . . . 8
- 3.2 Algorithm . . . 11
- 3.3 Argument for security . . . 14

Chapter 4: Handling Wave-Pipelining False Paths 15
- 4.1 Overview . . . 15
- 4.2 Controlling Signals . . . 15
- 4.3 Identifying Wave-Pipelining False Paths . . . 19
- 4.4 Reducing Wave-Pipelining False Paths to Wave-Pipelining True Paths . . . 20

Chapter 5: Experimental Results 26
- 5.1 Overview . . . 26
- 5.2 Functionality . . . 27
- 5.3 Runtime . . . 27

Chapter 6: Conclusion and Future Work 31
- 6.1 Conclusion . . . 31
- 6.2 Future Work . . . 31

Bibliography 33

List of Figures

2.1 Transformation Unit 7

3.1 Wave-Pipelining Example 9
3.2 False Path Example 10
3.3 Wave-Pipelining Duplication 12

4.1 Control Signal Tracing Example 18
4.2 Conventional vs Wave-Pipelining False Paths 19
4.3 Flip Flop Reinsertion Example 1 21
4.4 Flip Flop Reinsertion Example 2 22
4.5 Wave-Pipelining False Path Removal 25

5.1 s27 Benchmark 28
5.2 s27 Benchmark Waveforms 29
5.3 Experimental Results 30

Chapter 1: Introduction

1.1 Security Issue

Security has become a major concern in today's production of integrated circuits (IC). The vast majority of IC designers do not own or have access to their own foundries. This means they have to send their designs to untrusted third party foundries for fabrication. Introducing untrusted third parties to the production process also introduces security risks. A third party foundry may have adversaries that attempt to steal design information, over produce chips to sell for a profit on the side, or introduce malicious hardware that would allow one to read secret data from a legitimately obtained chip. In order to prevent an adversary's malicious intentions and protect one's intellectual property (IP), a designer will either lock or obfuscate their design before sending it to an untrusted third party for fabrication.

1.2 Obfuscation

There are many different algorithms to lock a circuit, but they all provide the same function: modify a circuit so that it only behaves as the original circuit does when some secret information is used. To lock a circuit, a designer will introduce new inputs

(collectively called the secret key) and logic structures that will change the circuit's functionality. When one only changes the logic of a circuit during locking it is called logic locking. However, a circuit can also be locked by changing the timing which is called delay locking. In delay locking, the secret key would determine whether timing constraints are met or whether a particular path is a conventional single cycled path or a multicycle path. Only when the correct secret key is input will a locked circuit behave as intended. The correct key must be kept secret from any potential adversaries in order to guarantee security. A locked circuit prevents an adversary from stealing design secrets as the functionality of the design is hidden and over producing chips as these overproduced chips would be useless without the secret key. Malicious hardware insertion may or may not be prevented depending on whether or not an adversary can tell where the desired data is.

A camouflaged circuit is one in which the intended functionality, while there, is not apparent to an adversary examining it. Camouflaging a circuit has a different goal than locking as there is no notion of a secret key. A camouflaged circuit is not protected against overproduction because the black-box functionality of the circuit is unchanged. However, camouflaging does protect against an adversary stealing design secrets as the function of the circuit is hidden. Just like in locking, malicious hardware insertion may or may not be prevented depending on whether or not an adversary can tell where the desired data is. Many camouflaging algorithms perform structural changes in a given circuit in order to hide functionality. Similar to locking, structural changes can be made to either logic or timing components.

1.3 Overview of Thesis

In chapter 2, previously proposed obfuscation algorithms along with attacks are discussed in more detail. In chapter 3, the specifics of one particular camouflaging algorithm, called TimingCamouflage+, are described [1]. This algorithm relies on making structural changes to timing critical paths in order to add ambiguity to a circuit's function. Chapter 4 then presents an attack showing it is not a secure camouflaging algorithm. Chapter 5 provides experimental results showing that the original circuit can be recovered from the obfuscated circuit. Finally, in chapter 6, the paper will conclude by considering possible future work.

Chapter 2: Background

2.1 Logic Obfuscation Techniques

There are many different obfuscation algorithms. One proposed camouflaging technique relies on using dummy contacts between layers to camouflage certain cells [4]. For an XOR-type camouflaging, a contact is inserted in between the polysilicon and metal layers of a buffer (BUF) cell. If the contact is real, the cell functions as a NOT cell, and if the contact is fake, the cell functions as a BUF cell. This method can be extended to work with other types of gates as well and results in a circuit with camouflaged functionality. There are also many other types of camouflaging techniques that use various circuitry characteristics, such as threshold voltage, to obscure gate functionality [5, 6, 7].

A proposed locking scheme inserts logic barriers into a circuit such that every path from an input to an output passes through a logic barrier [8]. These logic barriers each take in a key bit that determines whether the data should pass through unchanged or not. There are also many metering schemes that have been proposed [8, 9, 10]. Metering schemes are locking techniques that insert key-gates (gates that take a key bit as input) in such a way that different chips will require different keys.

All of the logic obfuscation techniques discussed above are vulnerable to SAT (boolean satisfiability) attacks [11]. The SAT attack identifies special input patterns that

allow groups of key values to be ruled out. Input patterns are selected iteratively until a correct key value has been found. A key value found by a SAT attack is guaranteed to be correct. There are also SAT attack formulations specifically for camouflaging as opposed to locking [12].

Researchers have proposed SAT attack resilient camouflaging techniques, including AND-Tree based camouflaging and CamoPerturb [4, 13]. These techniques force the SAT attack to take exponential time to finish, similar to SAT resilient logic locking approaches such as Anti-SAT and Stripped Functionality Logic Locking (SFLL) [14, 15]. However, as long as SAT attacks can be formulated, it has been proved that camouflaging or logic locking approaches must have very low error impact if they want to achieve SAT resiliency [16].

Timing based camouflaging is superior to the above mentioned logic based camouflaging in that it obfuscates the timing, rather than the logic, of the circuit. Therefore, the logic based SAT attack cannot be formulated against timing based camouflaging. This thesis focuses on the evolution of timing based camouflaging techniques and formulates an attack methodology against TimingCamouflage+, the state of the art timing based camouflaging technique [1].

2.2 TimingCamouflage

The TimingCamouflage algorithm creates wave-pipelining true paths in a circuit by removing specific flip flops [2]. These paths are created in such a way that they are indistinguishable from conventional true paths. Details on how these paths are created are

the same as those described in chapter 3. By not knowing whether a path is conventional or wave-pipelining, the correct functionality of the circuit is unknown by inspecting the netlist. A SAT based attack could be used to identify which paths are wave-pipelining (image wave-pipelining paths were represented by a key bit of 1 and conventional paths were represented by a key bit of 0), but would not be able to identify where the flip flop was removed from. This is important as different locations of the flip flop can produce different functionalities. In this way, TimingCamouflage is secure from standard SAT based attacks.

2.3 TiminigSAT Attack

To identify the flip flop removal locations from wave-pipelining true paths, a new SAT-based attack that focuses on timing, TimingSAT, was proposed [3]. TimingSAT relies on inserting transformation units (TUs), shown in figure 2.1, into the camouflaged circuit to convert wave-pipelining paths to conventional paths. The TUs are inserted into each potential flip flop location and introduce key bits into the circuit. Depending on the value of the TU's key, the flip flop in the TU will either be functional or not. At this point, a regular SAT based attack can be applied to determine a correct key. The paths of interest can also first be unrolled in order to convert uncontrollable flip flops into internal nodes, and then a regular SAT based attack can be applied. The TUs that have a key value of 1 mark the locations of where flip flops should be reinserted to regain original functionality. In this way, TimingSAT renders the TimingCamouflage algorithm unsecure. In an attempt to make the TimingCamouflage algorithm secure, wave-pipelining false

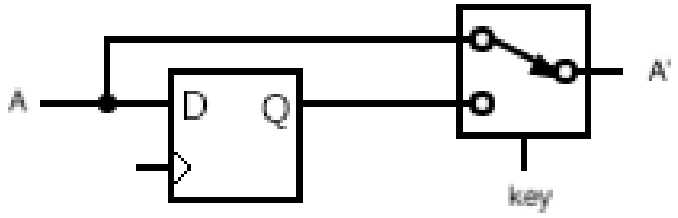


Figure 2.1: Transformation Unit (TU).

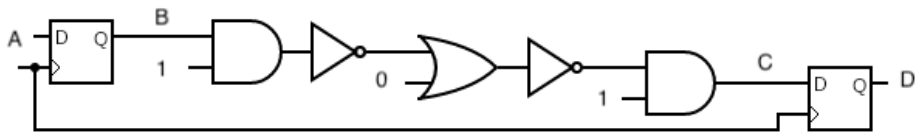
paths were introduced, and the TimingCamouflage+ algorithm was proposed and claimed to be secure against any known attack [1]. This thesis focuses on TimingCamouflage+ and proposes an attack formulation that renders it insecure.

Chapter 3: TimingCamouflage+: An Improvement on TimingCamouflage

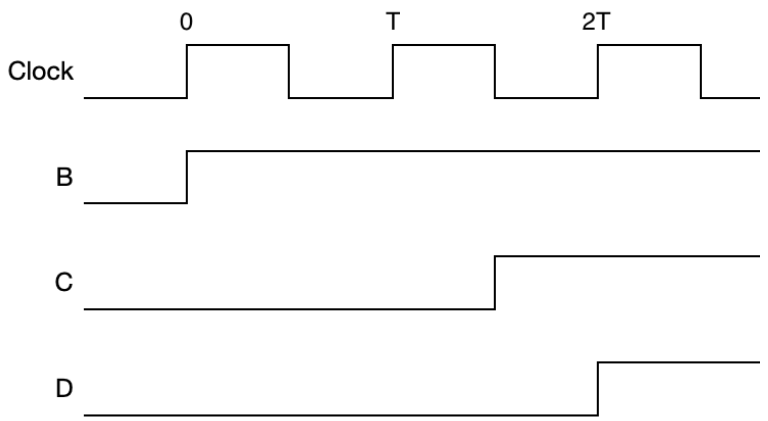
3.1 Types of Paths

The TimingCamouflage+ algorithm is built on creating different kinds of paths, specifically various combinations of true/false paths and conventional/wave-pipelining paths. Traditionally, a circuit is made up of only conventional paths, meaning all paths have a delay less than one clock cycle. A wave-pipelining path is one with a delay greater than one clock cycle where multiple data packets propagate down the path at the same time. As in the TimingCamouflage+ algorithm, wave-pipelining paths will have a delay time greater than one clock cycle and less than 2 clock cycles. Figure 3.1a shows an example of a wave-pipelining path. If at time 0 the signal B changes from 0 to 1, the signal will not propagate to location C in time to be reflected in the output D at time T, where T represents the time of one clock cycle (this will be the case throughout this paper). The propagation of the signal can be seen in Figure 3.1b. The transition of signal B from 0 to 1 at time 0 is reflected in the output D at time 2T. If signal B changes at time T, then there will be two signal changes being propagated in the path between time T and 2T in a pipelined fashion.

In the scope of this paper along with the TimingCamouflage+ paper, a false path is defined as a combinational path that cannot be sensitized because of controlling signals

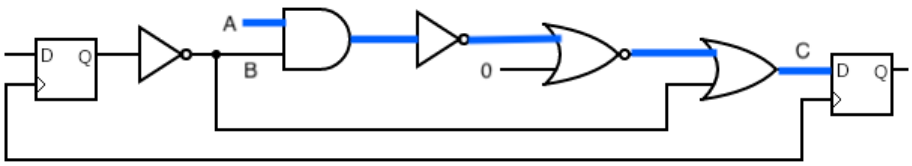


(a) Wave-pipelining path

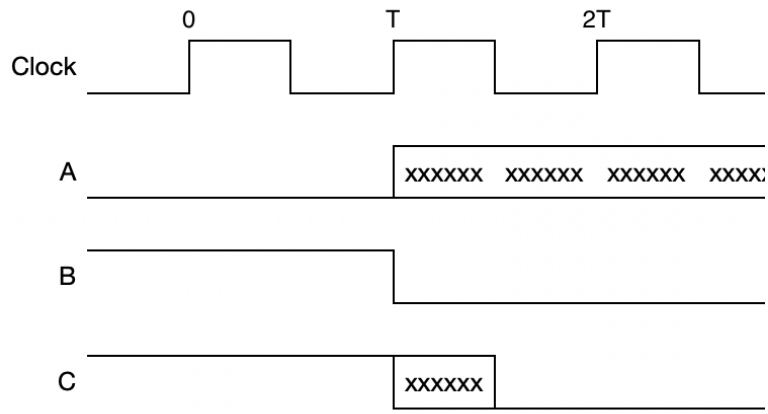


(b) Wave-pipelining waveform

Figure 3.1: Wave-pipelining example.



(a) False path circuit



(b) False path waveform

Figure 3.2: False path example.

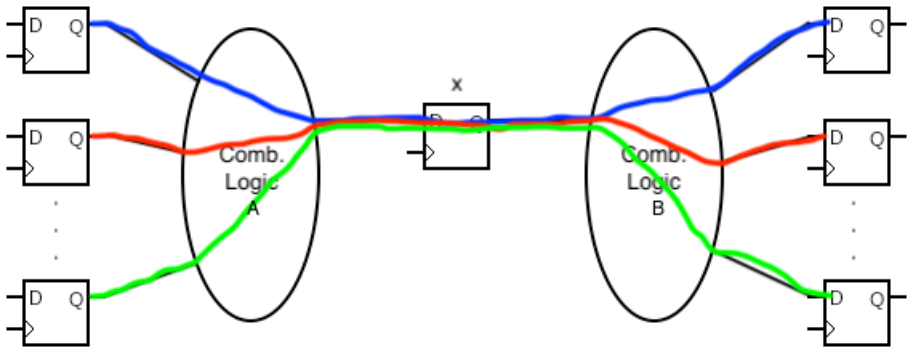
from other paths. Figure 3.2a shows a false path highlighted in blue. Notice that signal A does not affect the result at signal C. When signal B is 1, it controls the OR gate causing C to get a value of 1. When signal B is 0, it controls the AND gate which causes C to get a value of 0. Regardless of what value it is, signal B controls the result at C. In this case, B is the controlling signal. A path that is not false is called true.

Conventional true and false paths follow directly from the definitions above. However, wave-pipelining false paths are defined as wave-pipelining paths that are false when viewed as conventional paths. For instance, if the false path highlighted in blue in figure 3.2a was also a wave-pipelining path it would be considered a wave-pipelining false path.

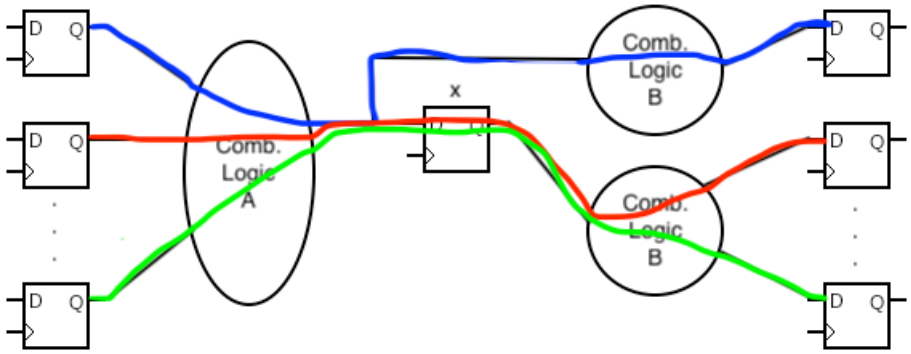
Note that if it were a wave-pipelining false path, signal A could affect the result at C. Consider the signal propagation in figure 3.2b. When signal B is 1 in the first clock cycle, C gets a value of 1. At time T, B becomes 0 and the value of C, once stabilized, depends on the value of A from the previous clock cycle. If the value of A was 0, C will be 0, and if the value of A was 1, C will be 1. In this way, signal A is able to affect the resulting signal at C. The definition of a wave-pipelining true path is synonymous with the definition of a wave-pipelining false path.

3.2 Algorithm

To obfuscate a circuit using TimingCamouflage+, wave-pipelining paths must be created by removing specific flip flops. Note that it is possible to remove a flip flop and have the resulting path still be conventional. However, these flip flops would not be considered eligible to remove. Additionally, removing a single flip flop can affect many paths as it can have many fanin and fanout paths. Consider the flip flop marked x in figure 3.3a. Let the blue path be a wave-pipelining path, the red path be a conventional path, and the green path be either type of path after removing flip flop x. While the blue path may be a desired wave-pipelining path for camouflaging, removing flip flop x would also result in the conventional red path and unknown green path. The red path (and potentially the green path) would change the functionality of the circuit. Changing the functionality of the circuit is not allowed so flip flop x cannot be removed. Because many circuits may not have candidate flip flops only belonging to one path, instead of simply removing the candidate flip flop, the circuit surrounding the flip flop is duplicated so that some paths



(a) Original circuit where a wave-pipelining path will be inserted



(b) Circuit with wave-pipelining path introduced

Figure 3.3: This figure shows the circuit duplication performed when a wave-pipelining path is created.

may use the flip flop and others may have it removed. Duplicating the surrounding circuit also prevents creating too many wave-pipelining paths all in one place. This strategy is illustrated in figure 3.3b. In this case, the green path is treated the same as the red path but could be treated like the blue path if it was also a desired wave-pipelining path.

Creating wave-pipelining paths is not enough on its own to obfuscate a circuit. An attacker could simply perform a static timing analysis to find the wave-pipelining paths and regain the original circuit. However, the TimingCamouflage+ algorithm uses the

notion of a "grey region" to help prevent this attack. Taking into account an error value δ, the grey region is defined as the values between T-δ and T+δ, where T is the period of one clock cycle. After performing static timing analysis, the paths that fall within the grey region cannot be confidently categorized as either conventional or wave-pipelining. When removing flip flops, the goal is to have the newly created wave-pipelining paths fall in the grey region. However, in addition to having some wave-pipelining paths in the grey region, some conventional paths should also be modified to belong in the grey region. This is to prevent an attacker from assuming that all grey region paths are wave-pipelining. By having both conventional and wave-pipelining paths in the grey region, the attacker must use input vectors to test whether a certain path truly is wave-pipelining and what the correct functionality of the path is.

Finally, in order for the camouflaging to be secure, both true and false wave-pipelining paths need to be created in the original circuit. As noted above, true wave-pipelining paths are not fully secure as they can be sensitized by certain input vectors. However, false wave-pipelining paths cannot be sensitized by an input vector. This is similar to how signal A does not affect the result at signal C in figure 3.2a when viewed as a conventional path. Notice that when using input vectors to try to sensitize a path, the paths are viewed as conventional because scan chain access is used for sensitization. When scan chain access is being used, only one input vector can be used (i.e. an input vector is given, and after a specified amount of time, the content of the scan chain is read). While a false wave-pipelining path may be sensitizable in reality (which can be seen in figure 3.2b), it is not sensitizable by an input vector using scan chain access. This is because to be sensitized, a false wave-pipelining path requires 2 consecutive input

vectors which is not supported by scan chain access. By including the 4 types of paths (true conventional, false conventional, true wave-pipelining, and false wave-pipelining), the TimingCamouflage+ paper claims its camouflaging algorithm is secure.

3.3 Argument for security

The TimingCamouflage+ paper argues its algorithm is secure because the attacker supposedly cannot distinguish the 4 types of paths discussed above (specifically the false wave-pipelining paths). In turn, the attacker would then not be able to tell which paths are wave-pipelining or what the correct functionality of the circuit is. The authors of the TimingCamouflage+ paper acknowledge that wave-pipelining true paths can be filtered out by using a boolean satisfiability (SAT) based attack [1]. However, because false wave-pipelining paths are not sensitizable within one clock period, a SAT based attack would not work for identifying these paths. The authors claim that in order for these paths to be found, major changes would need to be made to the scan chain allowing for multiple clock periods to be observed. In this way, the TimingCamouflage+ algorithm is asserted to be a secure camouflaging technique.

Chapter 4: Handling Wave-Pipelining False Paths

4.1 Overview

To uncamouflage a TimingCamouflage+ circuit, no work needs to be done to the conventional paths as they remain the same as in the original circuit. As acknowledged above, there is already a known method to discover true wave-pipelining paths, so this paper will only discuss the false wave-pipelining path case. Once this case is handled, TimingCamouflage+ will no longer be considered a secure camouflaging scheme.

4.2 Controlling Signals

False paths are dependent on the types of gates they are made up of and the orientation of the gates. There are 6 basic logic gates: NOT, AND, OR, XOR, NAND, and NOR. However, NAND and NOR gates are constructed from AND, OR, and NOT gates. Since these gates can easily be reduced to the other 4 types of gates, they will not be discussed. While XOR gates can also be reduced to AND, OR, and NOT gates, the conversion is not as simple as that of NAND and NOR gates, so they will be discussed.

A controlling signal of a gate is an input that determines the output of that gate regardless of what the other input values are. For example, the controlling signal of an

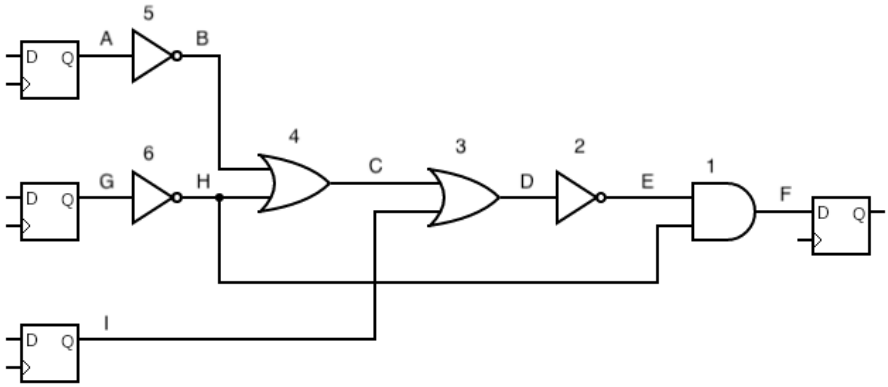
AND gate is 0, because whether the other inputs are 0s or 1s, the output will always be 0. Likewise, the controlling signal of an OR gate is 1 and produces a controlled signal of 1. There isn't the notion of a controlling signal in a NOT gate, because there is only 1 input signal. An XOR gate does not have a controlling signal either, because the output of an XOR gate always relies on all the input values. Any change to any of the input values will always change the output of an XOR gate.

Similar to a gate's controlling signal, a controlling signal of a path is a signal that always determines the output of that path regardless of what the other input values are. In figure 3.2a, B is the controlling signal as it either causes the output to be 1 (through the path directly to the last OR gate) or 0 (through the AND gate and the majority of the path highlighted in blue). The other inputs never affect the output value when all paths are viewed as conventional. Notice that the controlling signal in this case is connected to two gates with opposite controlling values. This allows signal B to be the path's controlling signal as it will always control one of the two gates. This setup of two gates with opposite controlling values on the same path with the same input signal is needed for the path to have a controlling signal (namely that signal shared between the two gates mentioned). A signal that is not a controlling signal is referred to as a non-controlling signal.

A false path is created when there is a controlling signal along it. Controlling signals cause adjacent signals to become expendable and unsensitizable, as the adjacent signals have no impact on the output. This leads them to become false paths. In figure 3.2a, signal A is adjacent to the controlling signal B and produces the false path highlighted in blue as it can never be sensitized.

The first step in handling false wave-pipelining paths is to identify controlling

signals. This is done by doing a reverse topological search on all the paths in the given camouflaged circuit. Starting at either a flip flop input (excluding clock and reset) or an output of the circuit, paths are traced backwards until a flip flop output or circuit input is reached. While paths are being traced, the types of gates are recorded along with their input signals. Two gates with opposing controlling values are being searched for. Once two appropriate gates that share an input are found on the same path, that shared signal is returned as a controlling signal. Figure 4.1 shows an example circuit along with its tracing. Tracing begins at the last flip flop's input and works towards each of the previous flip flops' output. At each gate encountered, the gate along with both its input signals are added to the set of things seen. Note that the set of things seen is path specific, so it is duplicated at a fork. For example, at the AND gate, the set of things seen is copied for both input signals, leading to locations two and six, and then modified accordingly. At any point in the circuit, the set of things seen only includes things topologically in front of the current point. Also note that the NOT gate at location six is visited twice. This is because its output, signal H, drives two separate gate inputs. In other words, it is on two paths and must be handled twice. Once location four is reached, two gates with opposing controlling values (an AND and an OR) that share an input (signal H) are found. This can be seen highlighted in green in figure 4.1b, and at this point, signal H would be returned as a controlling signal.



(a) Circuit for control signal tracing

	Location						
	1	6	2	3	4	5	6
Seen	AND, E	AND, E	AND, E	AND, E	AND, E	AND, E	AND, E
	AND, H	AND, H	AND, H	AND, H	AND, H	AND, H	AND, H
		NOT, G	NOT, D	NOT, D	NOT, D	NOT, D	NOT, D
				OR, C	OR, C	OR, C	OR, C
				OR, I	OR, I	OR, I	OR, I
					OR, B	OR, B	OR, B
					OR, H	OR, H	OR, H
						NOT, A	NOT, G

(b) Progression of signals seen during control signal tracing

Figure 4.1: Control signal tracing example.

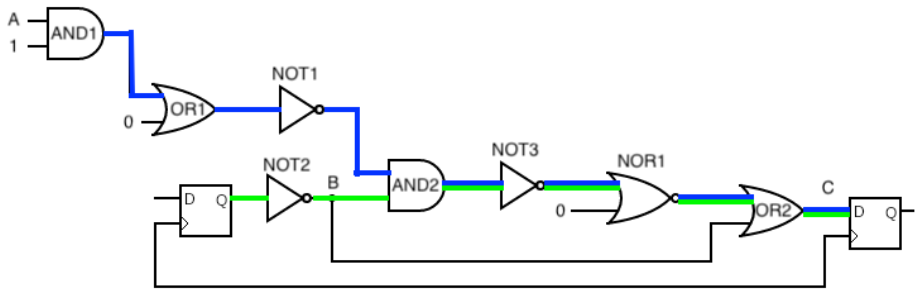


Figure 4.2: This figure is used to help explain when a false path is conventional or wave-pipelinig.

4.3 Identifying Wave-Pipelining False Paths

Once controlling signals have been identified, the corresponding false paths are also found. The next step is to classify these false paths as either conventional or wave-pipelining. This can be challenging, because conventional false paths can appear longer than a single clock cycle. Consider the false path in figure 4.2. If the path highlighted in blue is wave-pipelining, but the path highlighted in green is conventional, then the blue path is still considered a conventional false path. This is because the output at C stabilizes within one clock cycle, and any change in signal A will not ever be reflected in the output at C. In this case, the timing (and functionality) of the path highlighted in blue is irrelevant.

However, if the path highlighted in green is wave-pipelining, then the path in blue becomes a wave-pipelining false path. In order for a false path to be wave-pipelining, the path of the controlling signal must be wave-pipelining. This provides the opportunity for the controlling signal to become non-controlling in certain cases. For example, similar to

the example in figure 3.2, if the controlling signal B is 1 in the first clock cycle, it gives a non-controlling input to AND2 which allows the blue path to have an impact. However, because the green path is wave-pipelining, the impact will not take effect right away. Instead, during the second clock cycle the controlling signal B is set to 0 which gives a non-controlling input to OR2. This allows the effect of signal A from the previous cycle to control the output C. In this way, the controlling signal can become non-controlling, and this is how wave-pipelining false paths are created.

To identify these wave-pipelining false paths, all paths containing controlling signals need to be analyzed for timing. This can be done by setting all side inputs to non-controlling values and performing a static timing analysis to see how long it takes the controlling signal to propagate to the output. If one of the controlling paths is multi-cycled, then the corresponding wave-pipelining false path is found.

4.4 Reducing Wave-Pipelining False Paths to Wave-Pipelining True Paths

Once wave-pipelining false paths have been found, flip flops should be reinserted to uncamouflage the circuit. However, flip flop reinsertion creates another challenge as the exact location of flip flop insertion can be difficult to identify, but necessary for correct functionality. Consider the camouflaged circuit in figure 4.3. If considered wave-pipelining, inserting a flip flop at location ii yields a circuit equivalent to the result of inserting a flip flop at location iii (as long as both sides of the added flip flop have conventional paths). This is because whether the signal is stored at location ii or iii all the same data is used to produce the output. However, inserting a flip flop at location

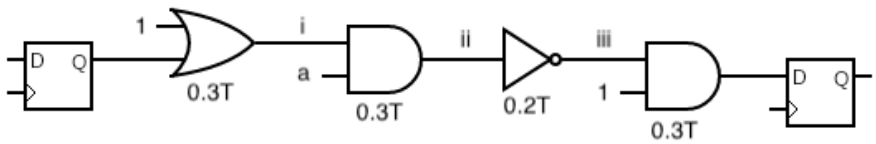


Figure 4.3: This circuit highlights functional problems with performing a flip flop reinsertion.

ii produces a different circuit than when a flip flop is inserted at location i. For example, consider the scenario where signal a is 0 in the first clock cycle and 1 in the second clock cycle.

- If a flip flop is inserted at location i, then at the end of the first clock cycle, the added flip flop will store 1, and at the end of the second clock cycle, the last flip flop will store a 0.
- If a flip flop is inserted at location ii, then at the end of the first clock cycle, the added flip flop will store 0, and at the end of the second clock cycle, the last flip flop will store a 1.

In this way, inserting a flip flop at location i produces a different output at the end of the second clock cycle than a flip flop at location ii would. This example showcases the importance of finding the (or a) correct location for flip flop insertion.

Ideally, the controlling signal path should be traced until the gate that marks the boundary between one and two clock cycles is found. Once this boundary gate is found, a flip flop should be inserted next to the gate. However, this can become ambiguous for a variety of reasons. Firstly, it is not clear whether tracing should be done forwards or

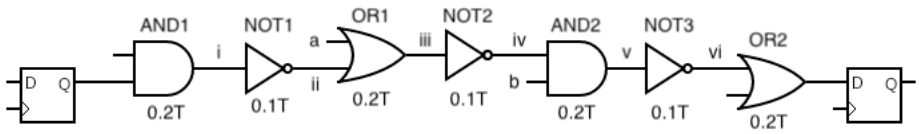


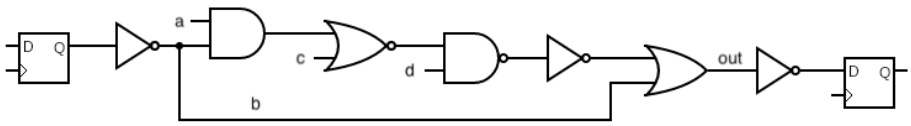
Figure 4.4: This circuit highlights timing problems with performing a flip flop reinsertion.

backwards. For example, in figure 4.3, the OR gate is the boundary gate when tracing backwards and a flip flop would be inserted to the right at location i, but the last AND gate is the boundary gate when tracing forwards and a flip flop would be inserted to the left at location iii. As can be seen in the explanation above, when a flip flop is inserted into these two locations, two functionally different circuits are created. Secondly, because there is some uncertainty in delay values, it may be unclear where the boundary gate is. For instance, if all the 0.3T timings in figure 4.3 became 0.4T, location i would mark exactly T time from a backwards tracing. While i may be the correct location of insertion, if uncertainty in delay is taken into account, the correct location could also be directly before or after location i. Thirdly, it may not be the case that either side of the boundary gate will have a delay of approximately T. In figure 4.4, tracing a full clock cycle forwards or backwards would return either location vi or i as the insertion candidate, respectively. Still, it could be that location iii (or any of the other locations) is the correct place for reinsertion. This would depend on whether the full paths of signals a and b are conventional or wave-pipelining. These ambiguities make flip flop reinsertion a challenging task.

However, this challenge of finding flip flop reinsertion locations is not unique to

false wave-pipelining paths. In fact, the same difficulty is also present in true wave-pipelining paths, and these paths have already been acknowledged as not secure. Therefore, to handle false wave-pipelining paths, they need only be converted to true wave-pipelining paths. To do this, the controlling path of the wave-pipelining false path will be duplicated and replaced to explicitly show the hidden wave-pipelining true paths. Consider the false wave-pipelining path in figure 4.5a. This circuit's functionality can be broken down into four cases depending on two recent values of the controlling signal. These cases are categorized in a truth table in figure 4.5b. The first two cases, when b[t=0]=1, have the same result, so they can be generalized to one case — leaving three cases in total. The controlling path of the original wave-pipelining false path can be replaced with a 3x1 multiplexer with the three out cases as inputs and two recent values of the controlling signal as the select lines. The two values of the controlling signal used are taken at time t=0 and t=-x, where x is the time it takes the controlling signal to go through the controlling path and arrive at the last controlling gate in the original camouflaged circuit (excluding the time of the last controlling gate). This new circuit can be seen in figure 4.5c. Here the original circuit has been converted into an equivalent circuit with two new true paths (one of which is highlighted in blue). The blue path is guaranteed to be a true wave-pipelining path, because it was the original false wave-pipelining path. The green path may or may not be wave-pipelining depending on the location of the removed flip flop in the original circuit, but this information is not needed. Since they yield the same truth table, this conversion is guaranteed to have the same functionality as the original circuit. This process can be generalized to work for all wave-pipelining false paths. Finally, once all wave-pipelining false paths have been converted into wave-pipelining

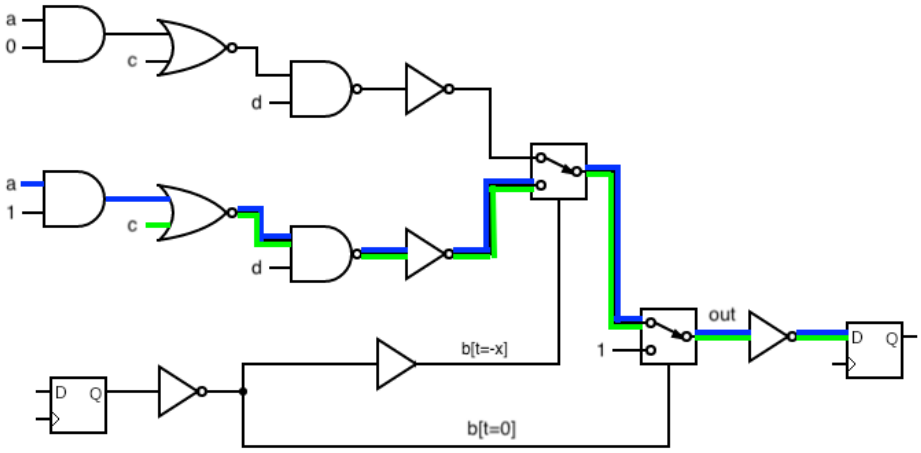
true paths, the circuit is no longer secure. At this point, the wave-pipelining true paths can be converted back to conventional paths using the TimingSat algorithm as described in chapter 2 [3].



(a) Original circuit with wave-pipelining false path

b[t=0]	b[t=-1]	out
1	1	1
1	0	1
0	1	!(!(!(a || c) && d))
0	0	!(!(!c && d))

(b) Truth table for the functionality of the circuit in (a)



(c) Circuit equivalent to (a) that no longer has a wave-pipelining false path

Figure 4.5: Wave-pipelining false path removal example.

Chapter 5: Experimental Results

5.1 Overview

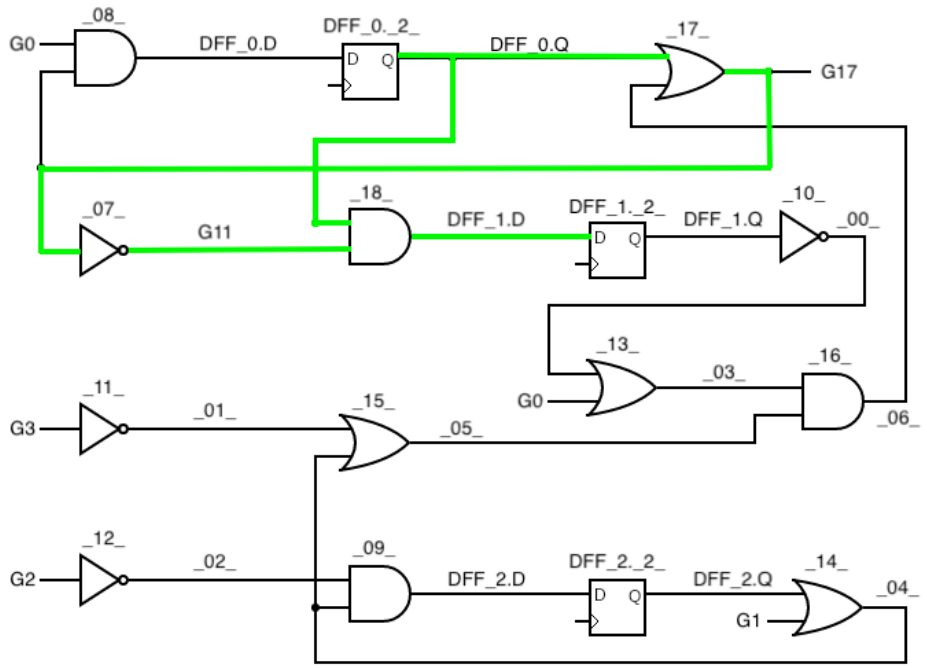
In order to evaluate the algorithm presented above, it was first implemented, then functionality was confirmed, and finally runtime was measured. For camouflaged circuits, the TimingCamouflage+ algorithm was simulated and ISCAS89 benchmarks were made to be consistent with that of the output of TimingCamouflage+. Next, controlling signals, along with their gates, were identified by tracing. Using the controlling signals and gates, controlling paths were also identified by tracing. A timing analysis was done on these controlling paths in order to classify them as conventional or wave-pipelining. Grey paths are treated as wave-pipelining, because if they are conventional, TimingSAT will find no flip flop locations resulting in no issues. On the other hand, if a grey path was treated as a conventional path but was wave-pipelining, the path would not be uncamouflaged. Once controlling paths have been identified, they can be replaced with their equivalent true paths rendering the circuit unsecure.

5.2 Functionality

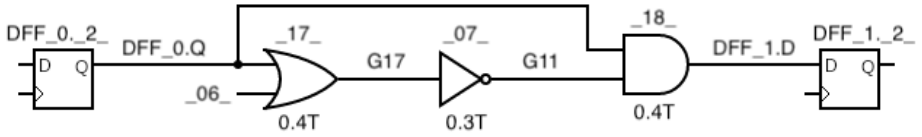
Functionality checking is provided here for the smallest benchmark, s27. Figure 5.1a illustrates the s27 benchmark, and the green path highlighted is shown in a close up in figure 5.1b. Signal DFF_0.Q was identified as a controlling signal of a wave-pipelining path, and the controlling path associated with this controlling signal was found: _17_, _07_, _18_. At this point, the last gate of the controlling path, _18_, was replaced with a 3x1 multiplexer and the rest of the gates were duplicated. Since the controlling signal took 0.7T to propagate to the last controlling gate in the original circuit, the select lines to the 3x1 mux are the current controlling signal and the controlling signal delayed by 0.7T. The resulting circuit is shown in figure 5.1c. The waveforms for the camouflaged and uncamoflagued circuits, figures 5.1b and 5.1c, are depicted in figure 5.2. As shown, the two inputs to DFF_1._2_, DFF_1.D and DFF_1.D2, are consistent with each other. As we have examined all four possibilities of control signal values within two cycles, the circuits before and after the conversion of wave-pipelining false paths to wave-pipelining true paths are equivalent, and the conversion is valid.

5.3 Runtime

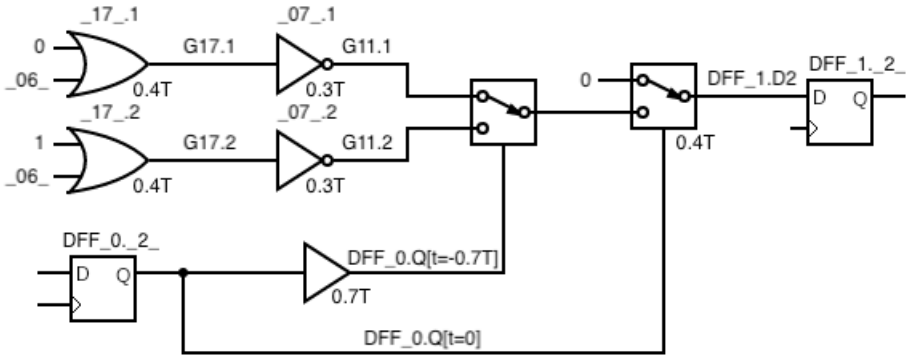
This process was performed on five benchmarks of varying size. Details of these benchmarks can be found in figure 5.3. This table shows that the algorithm's execution time increases with the number of false paths and not with the total size of the circuit as can be seen by benchmarks s1196 and s1423. Additionally, the time needed to handle



(a) s27 benchmark



(b) Wave-pipelining false path highlighted in green in (a)



(c) Circuit equivalent to (b) without wave-pipelining false paths

Figure 5.1: Removal of wave-pipelinig false paths from benchmark s27.

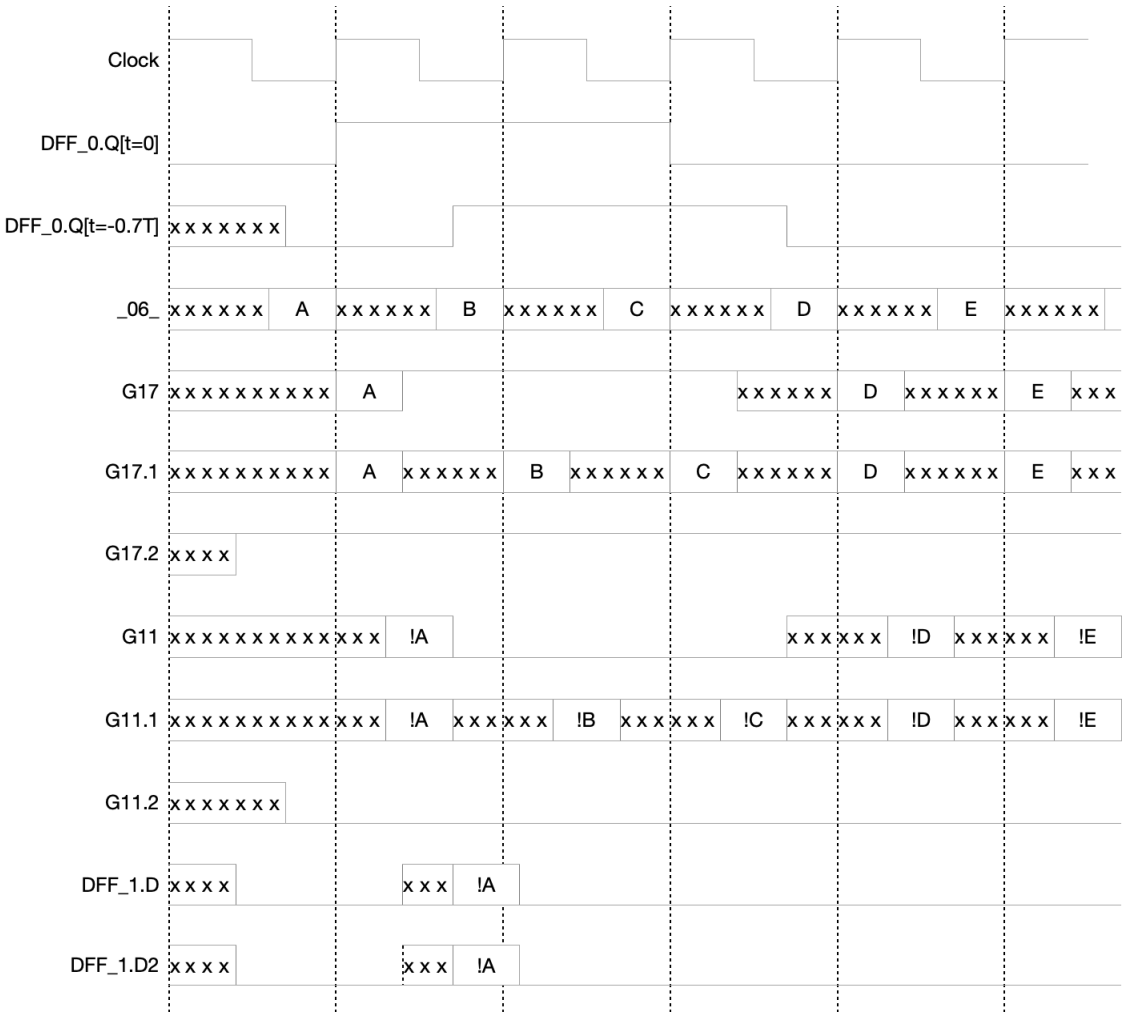


Figure 5.2: Functional verification of benchmark s27; waveforms of 5.1(b) and (c).

Benchmark	# Flip Flops	# Gates	# Conventional False Paths	# Wave-Pipelining False Paths	Execution Time (ms)
s27	3	10	1	1	0.05536
s382	21	158	1	2	0.22442
s526	21	193	3	2	0.26674
s1196	179	2779	7	5	0.71388
s1423	74	657	27	17	12.43602

Figure 5.3: The results of handling wave-pipelining false paths in five ISCAS89 benchmarks.

false wave-pipelining paths is relatively small which means this is a feasible attack algorithm.

Chapter 6: Conclusion and Future Work

6.1 Conclusion

In this thesis, the security of the TimingCamouflage+ algorithm is examined. While it is argued to be secure because of its unidentifiable wave-pipelining false paths, a method is proposed that allows these paths to be converted back to their original conventional path form. In order to do this, controlling signals are first found, controlling paths are then identified, and finally controlling paths are duplicated in a way that converts wave-pipelining false paths to wave-pipelining true paths. At this point, the TimingSAT algorithm can be used to convert wave-pipelining true paths back to conventional paths. By performing these path conversions, this paper has demonstrated that the TimingCamouflage+ algorithm is not a secure camouflaging technique as it is vulnerable to the TimingSAT algorithm after manipulation.

6.2 Future Work

Future work includes more extensive testing. For example, control signals belonging to multiple false paths could be tested. Additionally, more combinations of controlling gates could be considered as well. This thesis only considered the AND/OR pair for

controlling gates. However, it is possible to have an AND/AND or OR/OR pair if the controlling signal is first inverted before it is given to one of the controlling gates. While these future works may require extensions to the given algorithm, they all use the same base algorithm presented in this paper. In terms of creating a secure camouflaging technique, future working could be done to find an algorithm that securely obscures a circuit's timing without relying on wave-pipelining false paths.

Bibliography

[1] G. L. Zhang, B. Li, M. Li, B. Yu, D. Z. Pan, M. Brunner, G. Sigl, and U. Schlichtmann, "Timingcamouflage+: Netlist security enhancement with unconventional timing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 12, pp. 4482–4495, 2020.

[2] G. L. Zhang, B. Li, B. Yu, D. Z. Pan, and U. Schlichtmann, "Timingcamouflage: Improving circuit security against counterfeiting by unconventional timing," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 91–96, IEEE, 2018.

[3] M. Li, K. Shamsi, Y. Jin, and D. Z. Pan, "Timingsat: Decamouflaging timing-based logic obfuscation," in *2018 IEEE International Test Conference (ITC)*, pp. 1–10, IEEE, 2018.

[4] M. Li, K. Shamsi, T. Meade, Z. Zhao, B. Yu, Y. Jin, and D. Z. Pan, "Provably secure camouflaging strategy for ic protection," *IEEE transactions on computer-aided design of integrated circuits and systems*, vol. 38, no. 8, pp. 1399–1412, 2017.

[5] G. Di Crescenzo, J. Rajendran, R. Karri, and N. Memon, "Boolean circuit camouflage: Cryptographic models, limitations, provable results and a random oracle realization," in *Proceedings of the 2017 Workshop on Attacks and Solutions in Hardware Security*, pp. 7–16, 2017.

[6] R. P. Cocchi, J. P. Baukus, L. W. Chow, and B. J. Wang, "Circuit camouflage integration for hardware ip protection," in *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–5, IEEE, 2014.

[7] M. I. M. Collantes, M. El Massad, and S. Garg, "Threshold-dependent camouflaged cells to secure circuits against reverse engineering attacks," in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 443–448, IEEE, 2016.

[8] A. Baumgarten, A. Tyagi, and J. Zambreno, "Preventing ic piracy using reconfigurable logic barriers," *IEEE design & Test of computers*, vol. 27, no. 1, pp. 66–75, 2010.

[9] F. Koushanfar, "Provably secure active ic metering techniques for piracy avoidance and digital rights management," *IEEE Transactions on Information Forensics and Security*, vol. 7, no. 1, pp. 51–63, 2011.

[10] J. A. Roy, F. Koushanfar, and I. L. Markov, "Ending piracy of integrated circuits," *Computer*, vol. 43, no. 10, pp. 30–38, 2010.

[11] P. Subramanyan, S. Ray, and S. Malik, "Evaluating the security of logic encryption algorithms," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pp. 137–143, IEEE, 2015.

[12] M. El Massad, S. Garg, and M. V. Tripunitara, "The sat attack on ic camouflaging: Impact and potential countermeasures," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 8, pp. 1577–1590, 2019.

[13] M. Yasin, B. Mazumdar, O. Sinanoglu, and J. Rajendran, "Camoperturb: Secure ic camouflaging for minterm protection," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, IEEE, 2016.

[14] Y. Xie and A. Srivastava, "Anti-sat: Mitigating sat attack on logic locking," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 2, pp. 199–207, 2018.

[15] M. Yasin, A. Sengupta, M. T. Nabeel, M. Ashraf, J. Rajendran, and O. Sinanoglu, "Provably-secure logic locking: From theory to practice," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1601–1618, 2017.

[16] Y. Liu, M. Zuzak, Y. Xie, A. Chakraborty, and A. Srivastava, "Strong anti-sat: Secure and effective logic locking," in *2020 21st International Symposium on Quality Electronic Design (ISQED)*, pp. 199–205, IEEE, 2020.