ABSTRACT

Title of Thesis: SECURITY THROUGH OBSCURITY: LAYOUT OBFUSCATION OF DIGITAL INTEGRATED CIRCUITS USING DON'T CARE CONDITIONS

Sana Mehmood Awan, Master of Science, 2015

Directed By: Professor Gang Qu, Department of Electrical and Computer Engineering and Institute for Systems Research, University of Maryland

Contemporary integrated circuits are designed and manufactured in a globalized environment leading to concerns of piracy, overproduction and counterfeiting. Contemporary integrated circuits are designed and manufactured in a globalized environment leading to concerns of piracy, overproduction and counterfeiting. One class of techniques to combat these threats is circuit obfuscation which seeks to modify the gate-level (or structural) description of a circuit without affecting its functionality in order to increase the complexity and cost of reverse engineering. Most of the existing circuit obfuscation methods are based on the insertion of additional logic (called "key gates") or camouflaging existing gates in order to make it difficult for a malicious user to get the complete layout information without extensive computations to determine key-gate values. However, when the netlist or the circuit layout, although camouflaged, is available to the attacker, he/she can use advanced logic analysis and circuit simulation tools and Boolean SAT solvers to reveal the unknown gate-level information without exhaustively trying all the input vectors, thus bringing down the complexity of reverse engineering. To counter this problem, some 'provably

secure' logic encryption algorithms that emphasize methodical selection of camouflaged gates have been proposed previously in literature [5, 6, 15]. The contribution of this paper is the creation and simulation of a new layout obfuscation method that uses don't care conditions. We also present proof-of-concept of a new functional or logic obfuscation technique that not only conceals, but modifies the circuit functionality in addition to the gate-level description, and can be implemented automatically during the design process. Our layout obfuscation technique utilizes don't care conditions (namely, Observability and Satisfiability Don't Cares) inherent in the circuit to camouflage selected gates and modify sub-circuit functionality while meeting the overall circuit specification. Here, camouflaging or obfuscating a gate means replacing the candidate gate by a 4X1 Multiplexer which can be configured to perform all possible 2-input/ 1-output functions as proposed by Bao et al. [16]. It is important to emphasize that our approach not only obfuscates but alters sub-circuit level functionality in an attempt to make IP piracy difficult. The choice of gates to obfuscate determines the effort required to reverse engineer or brute force the design. As such, we propose a method of camouflaged gate selection based on the intersection of output logic cones. By choosing these candidate gates methodically, the complexity of reverse engineering can be made exponential, thus making it computationally very expensive to determine the true circuit functionality. We propose several heuristic algorithms to maximize the RE complexity based on don't care based obfuscation and methodical gate selection. Thus, the goal of protecting the design IP from malicious end-users is achieved. It also makes it significantly harder for rogue elements in the supply chain to use, copy or replicate the same design with a different logic. We analyze the reverse engineering complexity by applying our obfuscation algorithm on ISCAS-85 benchmarks. Our experimental results indicate that significant reverse engineering complexity can be achieved at minimal design overhead (average area overhead for the proposed layout obfuscation methods is 5.51% and average

delay overhead is about 7.732%). We discuss the strengths and limitations of our approach and suggest directions that may lead to improved logic encryption algorithms in the future.

SECURITY THROUGH OBSCURITY: LAYOUT OBFUSCATION OF DIGITAL INTEGRATED CIRCUITS USING DON'T CARE CONDITIONS

By

Sana Mehmood Awan

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
2015

Advisory Committee:
Professor Gang Qu, Chair
Associate Professor Mark Austin
Assistant Professor Dana Dachman-Soled

Dedication

This work is dedicated to my *Baba Jan* who gave me my life's greatest lesson in optimism, courage and faith.

Acknowledgements

I would like to thank my advisor, Dr Gang Qu for his insightful guidance, support and help throughout the completion of this work, and for being not just a great teacher but also a wonderful person. I am also grateful to Mingze Gao for his help and for all the fruitful discussions we had on the topic.

Additionally, I would like to thank the Fulbright Program for their financial support over the course of my graduate studies.

Finally, I would take this opportunity to express my gratitude to my mother whose constant love and encouragement mean the world to me.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

"There is no such thing as true security, only varying levels of insecurity."

CHAPTER 1: THE NEED FOR HARDWARE IP SECURITY

1.1 Introduction

The dynamic and rapidly evolving field of electronics has changed our lives immeasurably and irrevocably since the invention of the transistor in 1948. The power, speed and availability of information will continue to define our social, political and economic landscape. The all-pervasive nature of Electronics can be felt in every aspect of human life in this "silicon age". Starting from 1960 when commercial semiconductor manufacturing became a viable business, the industry has grown with a prodigious average annual growth rate of 16.1% between 1975 and 2000 [1, 2]. According to the Semiconductor Industry Association (SIA), representing U.S. leadership in semiconductor manufacturing and design, worldwide semiconductor sales for 2013 reached $305.6 billion, the industry's highest-ever annual total and an increase of 4.8 percent from the 2012 total of $291.6 billion. The industry saw strong demand in several product segments during 2013, especially logic and memory. Logic was the largest semiconductor category by sales, reaching $85.9 billion in 2013, a 5.2 percent increase compared to 2012 while memory was the fastest growing field recorded, increasing 17.6 percent in 2013 [3]. The key to this rapid advancement in electronics is the aggressive reduction in device dimensions in integrated circuits (ICs) over generations, along with scaling of supply voltage (VDD) and transistor "threshold voltage" (Vth), a phenomenon known as "technology scaling". Device integration density (the number of devices per unit area in an IC) has roughly doubled every two years over the last five decades, a trend first predicted in [4], and now famously termed as "Moore's Law". This has enabled a corresponding exponential increase in computing capabilities and functionality of ICs.

With the rapid development of embedded systems, application specific integrated circuits (ASICs) of small and medium size are playing a more and more important role in the electronics market. Under the umbrella of Systems Engineering, new and pre-existing devices are being connected in novel ways over different forms of communication channels to facilitate increasingly connected system-of-systems. However, with rapidly increasing computing power and integration density, several issues in design, performance and manufacturing of these ICs have come to surface. Increasing power consumption (both due to higher integration density and increasing “leakage" power consumption in the off-state) increased cost of testing and verification of complex integrated circuits and photolithographic complexities in manufacturing nanometer devices (with feature sizes much smaller than the smallest wave-length of light that are used for patterning) are the some of the major issues with IC design and manufacturing in the nanometer regime.

1.2 The Emergence of Re-use based System-on-Chip Devices and their Supply Chain Security Implications

To overcome some of the challenges mentioned above and meet design and testing costs and aggressive time-to-market targets effectively, re-use based System-on-Chip (SOC) design using hardware Intellectual Property (IP) cores has become a pervasive practice in the industry to realize bug-free complex System-on-Chip (SoC) devices. System designers can pick integrated circuits (ICs), considered as intellectual property (IP), that are produced for specific functionality and fit them together to achieve a specific goal. This leads to a culture of re-use based design. As a result, IP theft has become profitable as well as a threat to IP developers, vendors, and the SoC industry in general, which motivates the IP protection problem [18].

Considerations of capital costs and economies of scale dictate that semiconductor manufacturing is now reliant on offshore or contract foundries that are organizationally separate and geographically distant from the design houses that design and validate integrated circuits. Qualcomm became the first fabless semiconductor company to rank among top 10 IC producers worldwide and even AMD has been

outsourcing some of its production to foundries throughout the world. However, with the growth of manufacturing potential in Asia, piracy has become rampant, thanks to loose IP protection policies and weak enforcement of IP anti-piracy laws. To make such design and manufacturing feasible, an IC design house is commonly aided by the following external agencies [5]:

(i) Electronic design automation (EDA) companies that supply the sophisticated software tools that facilitate design, verification and testing of modern ICs.

(ii) Intellectual property (IP) and standard cell library vendors, who supply pre-verified, high performance, functional hardware IPs or library cells to the IC design facilities. These help to reduce the design time drastically, improve reliability and yield, and enable meeting hard time-to-market targets.

(iii) Semiconductor manufacturing companies, which provide the fabrication facilities (“fabs") where the design is actually manufactured and sometimes tested, before being sent back to the design house. The cost of maintaining a cutting edge fab runs into billions of dollars every year, encouraging most IC vendors (over 250 strong in 2009 [8]) to follow a “fabless" business model where the design database is outsourced to the fabs for manufacturing [7].

From the above description of prevalent industry practices, it is evident that the control exercised by the IC vendors over the design and manufacturing of their own products is decreasing. The increasing complexity and cost of modern nanometer scale ICs are the main drivers behind this trend. Reduced control on the IC life-cycle accentuates various security issues associated with ICs. Hence, security of hardware IPs and ICs has emerged as a major challenge in nanometer IC design and test.

These IP cores usually come in the form of synthesizable Register Transfer Level (RTL) descriptions (Soft IP), or gate-level designs directly implementable in hardware (Firm IP), or GDS-II design database (Hard IP).

As electronic design automation (EDA) tools and semiconductor technology continues to evolve rapidly, a company will not typically have all the expertise and capability to do in-house design and to fabricate the system it wants to build. If the system is designed and fabricated by others, ''***how can the company be convinced that the system is trusted, that is, the delivered system does exactly what the company wants, no more and no less?***'' This is known as the trusted IC design challenge, particularly for military and civilian systems that require security and access control [Defense Science Board 2005; Cohen 2007; Irvine and Levitt 2007; Trimberger 2007; Suh and Devadas 2007; Roy et al. 2008; Rajendran et al. 2012; Gu et al. 2009].

When the company gives the system's specifications to a design house for layout verification or simulation and synthesis, and then gives the layout information to a foundry for manufacture, the company will lose full control of the system's functionality and specifications. An adversary can simply add additional circuitry, such as hardware or software Trojan horses, to maliciously modify the system. For example, a Trojan can disable or destroy system components, perform incorrect or unwanted computation, or leak sensitive information [11].

Another issue is the widespread use of **Design for Testability (DfT)** practices that add certain testability features to a hardware product design. The premise of the added features is that they make it easier to develop and apply manufacturing tests for the designed hardware and reduce time-to-market. These added features, such as scan-chains for sequential circuits, facilitate validation that the product hardware contains no manufacturing defects that could otherwise adversely affect the product's correct functioning.

In addition, IP vendors also allow evaluation versions of their IPs to be downloaded and evaluated by the IC designers. This is an important part of their business because it enables wide publicity of their product leading to increased market share. It also helps them to get feedback about their product from potential customers. From the designers' perspective, IP evaluation is an important step as well since it helps them

explore alternative IP cores with regard to correctness, quality (power, performance, die-area), configurability, testability, and compatibility with other modules in a SoC design.

In the following section, we formally describe different security threats associated with IC design and test cycle introduced due to current industry practices some of which are mentioned above, which will justify our need for effective hardware security solutions.

1.3 Security Threats to IC Supply Chain

The active participation of various external agents in the design and manufacturing process of ICs, as indicated in section 1.2, has made the entire process highly vulnerable to various security threats.

Fig. 1 shows the level of “trust" that can be assigned to the various stages of a typical modern IC design flow [5]. The design life-cycle of IPs shows that these IPs are highly vulnerable to piracy issues at different stages. As a result, the security of integrated circuits supply chain and the reliability of IC functionality in the face of supply chain threats have emerged as a major concern at different stages, spanning design, test, fabrication and deployment.

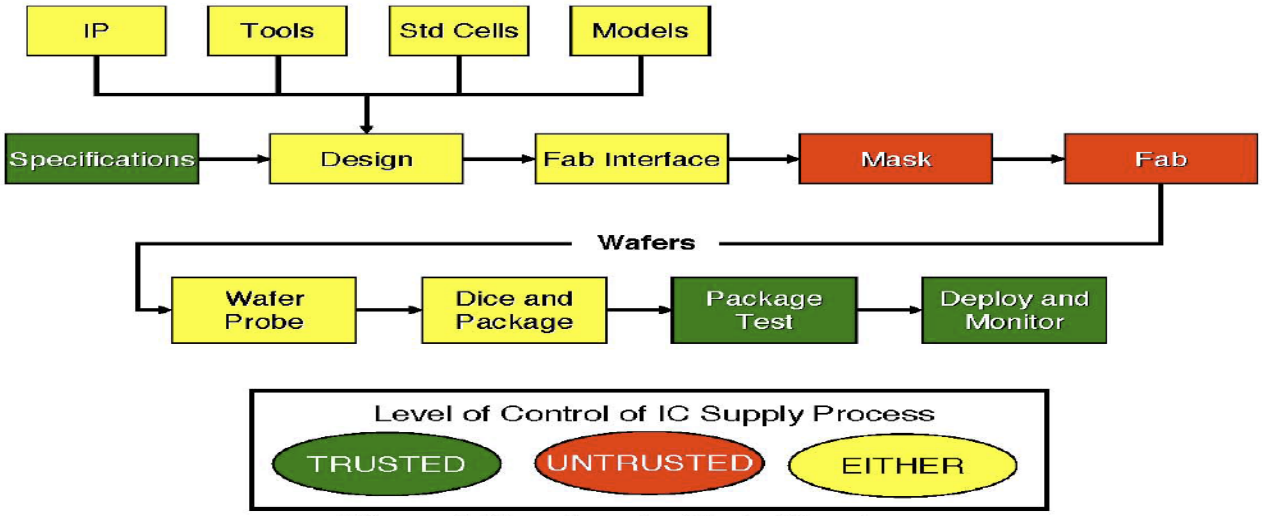


Figure 1.1: Vulnerabilities in the IC Design Cycle for IP Vendors [5], [26]

These security threats include reverse-engineering efforts to facilitate cloning, counterfeiting, unauthorized overproduction by the contract foundry or remarking of ICs as well as malicious alterations by untrusted third-party vendors. It is worth noting that the “design" stage itself is designated as one of the partially insecure stages of the entire flow. This takes into consideration the possible presence of untrusted personnel in the design house with access to the design, who might sabotage the design to serve other interests. Additionally, if the specifications for the proposed design are not complete so that there are unused states in the resulting design (for example a specification for two-bit multiplication table implemented using a 4-bit multiplier) then the inherent redundancy introduced in the circuit by such an implementation can be exploited by a malicious fabrication house to make the circuit have additional unwanted functionality or potentially expose “safe” states. This has been demonstrated in [11] with solutions to counter the problem.

Foundries with access to layout and mask information can easily extract the gate-level netlist and compare circuit simulation results with the responses obtained from an activated IC that has been purchased off the market or provided by the vendor for testing purposes. Thus, a malicious foundry with a means for applying arbitrary input patterns and observing the resultant outputs on an activated IC can easily extract and/or modify circuit functionality. Similarly, package testing may or may not be trusted, depending on who it is contracted out to.

In addition to the above, more critical threats come from the supply chain and compromise hardware integrity. In today’s global IC industry, a supply chain adversary, such as an IP provider, an IC design house, a CAD company, or a foundry may have access to the source code of the design, and may easily tamper a hardware system by planting time bombs which compromise hardware computation integrity, or create back doors which enable information leak, or bypass access control mechanisms at higher (e.g., OS and application) levels. The recently released Comprehensive National Cyber Security Initiative has identified this supply chain risk management problem as a top national priority. The issue of malicious modifications

to the design, particularly by the use of *hardware Trojans and hardware time bombs* has emerged recently due to the widely prevalent industrial practice of fabrication of ICs in potentially untrusted foundries. *Hardware Trojans* are malicious modifications of a circuit that can cause it to fail during deployment, with potentially disastrous consequences. These Trojans would typically evade detection during conventional post-manufacturing test because they trigger malfunction only under extremely rare conditions unlikely to be exercised during test. Moreover, the Trojans can be tiny relative to the original circuit which makes them difficult to detect by *side-channel measurements*. Some Trojan circuits, referred to as *sequential Trojans*, represent FSM structures, which go through a sequence of state transitions before getting activated. On activation, they trigger malfunction, typically by altering logic values at some payload nodes or by leaking secret information.

Figure 1.2 (a) details some of the vulnerabilities of the IC design flow which involves design, fabrication, assembly, distribution, usage in the system, and finally end-of-life. As seen, there are vulnerabilities associated with each step in this supply chain. In design stage, an IP may be stolen or a hardware Trojan may be inserted into the design. An untrusted foundry or assembly can insert a hardware Trojan or produce different types of counterfeits. The design house can use illegally obtained IPs in their designs. Overproduced and out-of-spec/defective parts can be entered into the supply chain in the fabrication stage. Untrusted foundries can potentially sell these parts in the open market. They can also tamper with the design to create a backdoor for getting secret information from the field. These parts also get into the supply chain in the assembly phase. An untrusted assembler can possibly sell these parts or tamper with the designs. Illegal activities during distribution, during the IC lifetime, and even at end of useful life may bring different types of counterfeits back into the supply chain (recycled, remarked, etc.) [14].

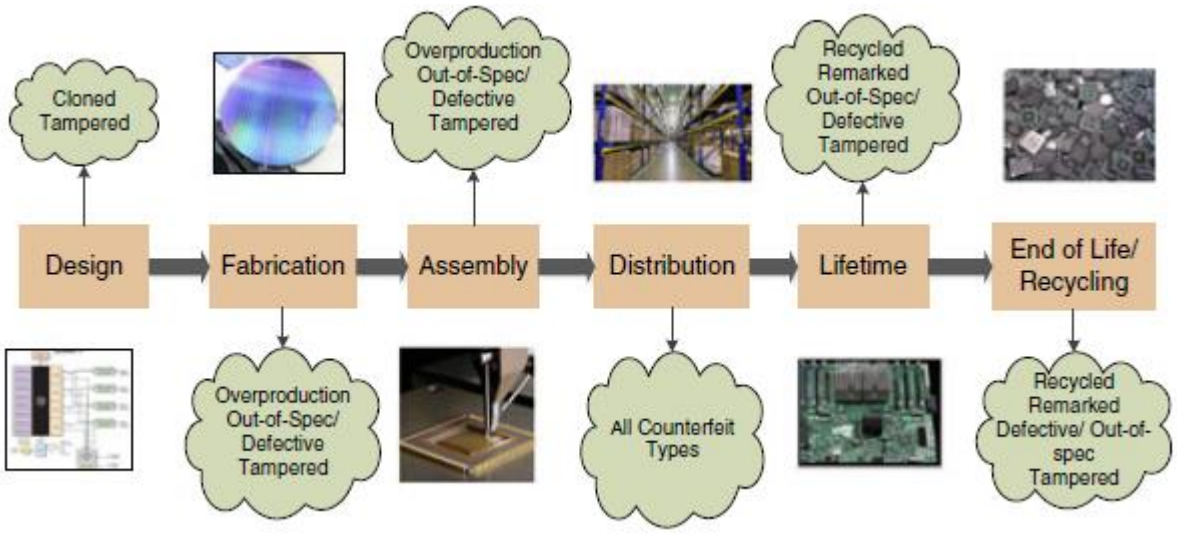


Figure 1.2(a): Electronic Component Supply-chain Vulnerabilities (Tehranipoor et al. 2014 [14]). "All counterfeit types" indicated in the Figure are detailed in Figure 2.2

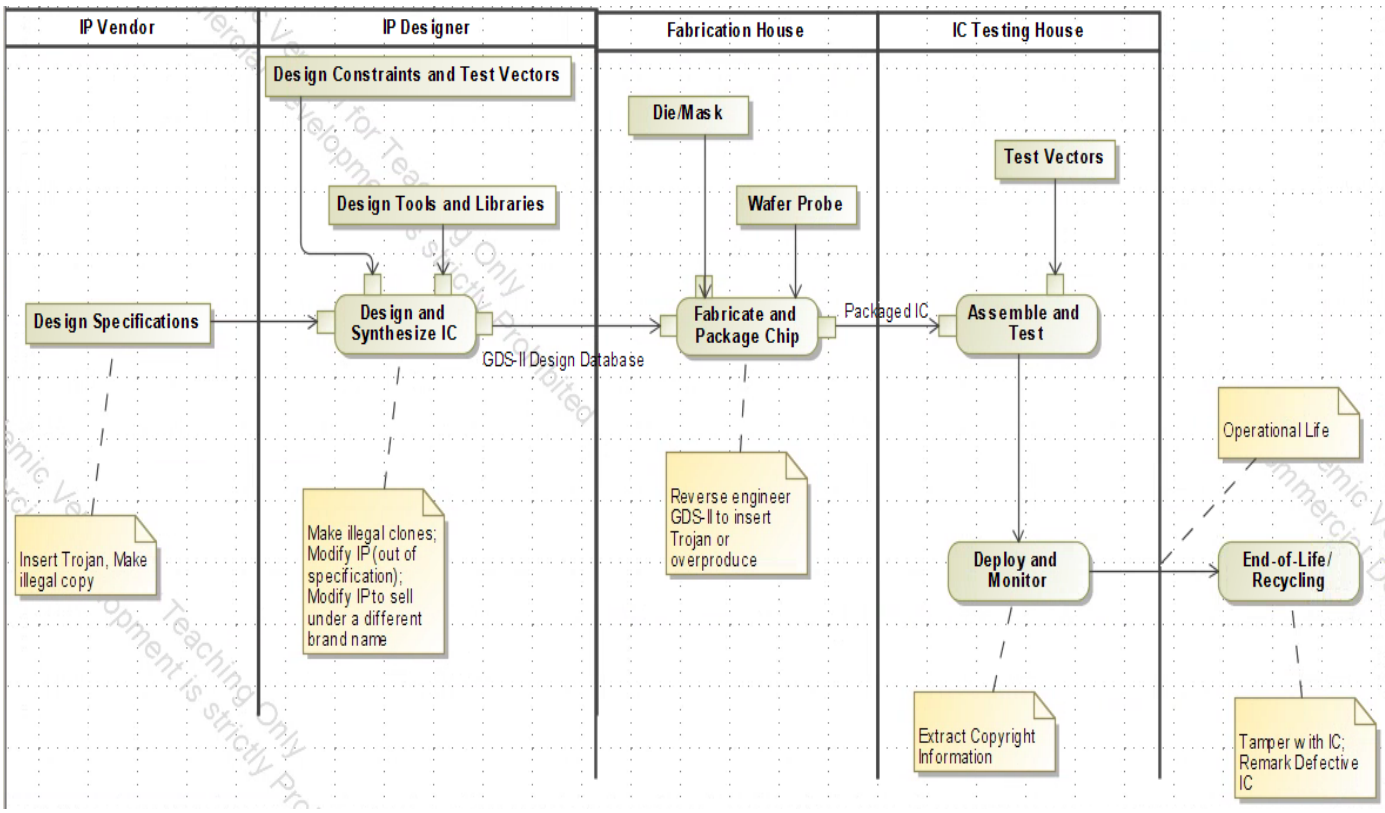


Figure 1.2 (b): Activity Diagram showing IP Design Flow. Examples of Security Threats from the perspective of the IP vendor in various stages of IP Design are shown in text boxes corresponding to the different stages.

The main mechanisms and motivations behind the threats in Figure 1.2 can be broadly classified as follows [5]:

(i) IP piracy: This is a scenario where the IP is illegally used and/or copied without paying the lawful royalty to the IP vendor.

(ii) IC piracy: In this scenario, the manufacturing fab illegally copies and reverse engineers the design database of an IC sent for fabrication to manufacture illegal copies (“clones") of the IC.

(iii) Hardware Trojan insertion: The design can be modified in the design house or in the fab by malicious insertion, deletion or modification of circuits, referred to as Hardware Trojans, which cause the IC to deviate from its intended functional behavior during deployment. Typically, these Trojan circuits are stealthy by design, which makes it extremely challenging to detect them by traditional post-manufacturing testing.

(iv) Secret information leakage: Although the “deploy and monitor" step of the design flow has been shown to be completely trustable in Figure 1.1, in reality, it has been shown that secret information can be extracted by an adversary from secure ICs with cryptographic functionality in this stage (for example, by side-channel analysis). Such threats increase with increasing controllability and observability of the internal nodes of the circuit as a result of widespread adoption of “Design for Testability" (DfT) techniques in modern ICs.

As a result of more recent advances in computation, malicious foundries and fabrication houses may also use satisfiability checking tools such as *binary SAT solvers* along with state-of-the-art circuit simulation tools to find hidden copyright information such as keys belonging to the holder of IP rights. This powerful logic decryption technique has been recently demonstrated in [9].

In the face of these vulnerabilities, an IP Vendor may have the following validation questions about the security of the IC design cycle:

(i) Is the design true to specifications?

(ii) Does the IC perform its intended functions and nothing more? If not, how to check for unknown behaviors or additional unwanted/ unintended functionality?

(iii) For reconfigurable devices, does the configuration data accurately represent the specification, design, and synthesis?

(iv) How to check for inserted circuits or hardware time bombs during fabrication?

(v) How to check for authentication post-fabrication?

These are important questions. To complicate matters, modern ICs consist of over 20 patterned layers of metals, insulators and semiconductors, with smallest feature sizes at 45nm and decreasing. The patterns are "burned in" by shining a 193-nm ArF laser through chromium-quartz masks in a tightly controlled process at fabrication facilities. A mask set contains a complete physical representation of an IC. Contract fabs produce masks from large computer files supplied by their clients. The IC descriptions given to fabs are often customized to satisfy a fab's specific requirements, but if stolen, they may conceivably be adjusted to another fab, and leading-edge fabs are concerned about this. Another form of piracy is for the contracted fab to produce more chips than authorized, at a very small additional cost, and sell them on the black market. A simple anti-piracy measure is ***wafer banking,*** also called ***split fabrication,*** i.e., contracting out different layers of a chip to different manufacturers. Not only is this expensive, but it prevents fabs from testing ICs which hampers yield analysis and improvement. Fabricating features smaller than half of 193nm (the ArF laser's wavelength) is increasingly difficult, and no viable replacements to ArF lasers are expected in the near future.

To compensate for optical diffraction, mask patterns are much more complex than the manufactured patterns and may be harder to reverse-engineer by delamination or otherwise. Physically modifying fine-grain features of ICs after manufacturing, to defeat anti-piracy measures, is very difficult.

The Focused Ion Beam (FIB) technique is sometimes used to reconnect wires during post-silicon debugging, but remains too slow and expensive for mass production, and will likely be infeasible for ICs with 32nm features [6].

The mechanisms employed in pirating IP designs demonstrate that the threat to the IP vendor is very real, with many untrustworthy components in the global supply chain. This has huge implications for the semiconductor industry, where Reverse Engineering has become a powerful tool for intellectual property (IP) piracy, in which the attacker analyzes the design and reproduces it with no or much less investment in research and development. These low cost illegitimate products can be sold at a much lower price, giving them an unfair competitive edge against the authentic products. Moreover, when the high level functionality of the target circuit is extracted, the attacker can redesign the circuit to obtain a new design that will appear different from the target circuit in order to avoid the infringement of copyright. The financial risk alone due to counterfeit and unauthorized ICs was estimated to be over $169 billion a year by IHS Technology in 2012 [10]. Besides financial losses, this issue potentially has national security implications. The Semiconductor Industry Association (SIA) estimates that 15% of all spare and replacement semiconductors purchased by the Pentagon are counterfeit with a large contribution coming from hardware IPs [9].

Having established the importance of, and need for secure hardware IP design, devising an effective circuit obfuscation methodology that mitigates the effects of the security vulnerabilities in the IC design cycle with minimal overhead on design, fabrication and testing of ICs is the main motivation for this research.

"At the end of the day, the goals are simple: Safety and Security."

--Jodi Rell

CHAPTER 2: SURVEY OF EXISTING HARDWARE IP SECURITY SOLUTIONS

From the discussion in Chapter 1, it is clear that hardware IP has become the foundation and the root of trust of any security system. In recent years, a growing number of software-based security solutions have been migrated to hardware-based security solutions for much enhanced resistance to software-based security threats. Such systems range from smartcards to specialized secure co-processing boxes, wherein hardware provides the source of security and trust for a number of security primitives. Examples include Trusted Platform Module (TPM) and Physical Unclonable Function (PUF) applications.

As previously mentioned, a supply chain adversary's capability is rooted in his knowledge of the hardware design. Successful hardware design obfuscation would severely limit a supply chain adversary's capability if not prevent all supply chain attacks. However, not all designs are obfuscatable in traditional technologies.

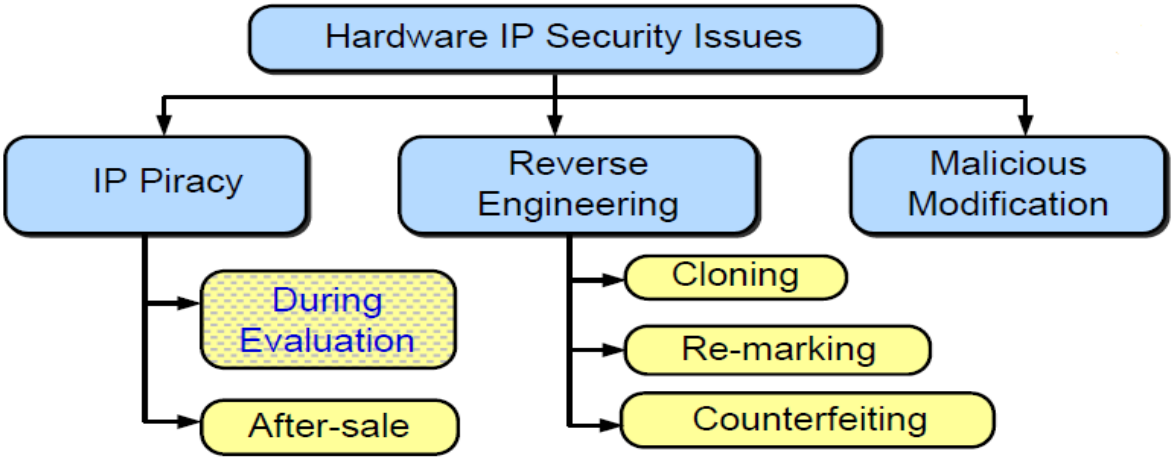


Figure 2.1: Taxonomy of Hardware IP Security Issues (from Narasimhan et al.2012)

Evaluation of hardware Intellectual Property (IP) cores is an important step in an IP-based system-on-chip **(**SoC) design flow. From the perspective of both IP vendors and Integrated Circuit (IC) designers, it is desirable that hardware IPs can be freely evaluated before purchase, similar to their software counterparts. However, protection of these IPs against piracy during evaluation is a major concern for the IP vendors. Existing solutions typically use encryption and vendor-specific toolsets, which may be unacceptable due to lack of flexibility to use in-house or third-party design tools. To prevent IP piracy, the IP vendors traditionally enforce a binding licensing agreement with the design house. Alternatively, they provide an IP in encrypted form. The decryption process is accomplished with a vendor-specific design platform for simulation and synthesis. The latter approach is prevalent in field programmable gate array (FPGA)-based design framework. On the other hand, some IP vendors allow simulation of the downloaded IP, but do not allow it to be synthesized to gate-level designs or bit streams (for FPGA platforms). Such practices, however, force a SoC designer to evaluate only the IP's functional behavior, but not the important quality parameters. To overcome the shortcomings of existing IP evaluation practices, a recent industry initiative has resulted in a design platform, which allows designers to download and use encrypted IPs from vendor websites. The synthesis and simulation tools in this design platform are capable of working in a user transparent manner based on a technology undergoing IEEE standardization. However, it mandates the use of a particular design platform throughout the design flow, which may not be acceptable for modern SoC designers, who typically use different software tools from diverse vendors as well as in-house design tools [12].

Post evaluation, and after-sale, the threat to IP security grows with increased exposure to market elements. A counterfeit component *(i)* is an unauthorized copy; *(ii)* does not conform to the Original Component Manufacturer (OCM) design, model, and/or performance standards; *(iii)* is not produced by the OCM or is produced by unauthorized contractors; *(iv)* is an off-specification, defective, or used OCM product sold as "new" or working; or *(v)* has incorrect or false markings and/or documentation [14].

Based on the definitions above and analyzing supply chain vulnerabilities, we classify the counterfeit types into seven distinct categories shown in Fig. 2.2.

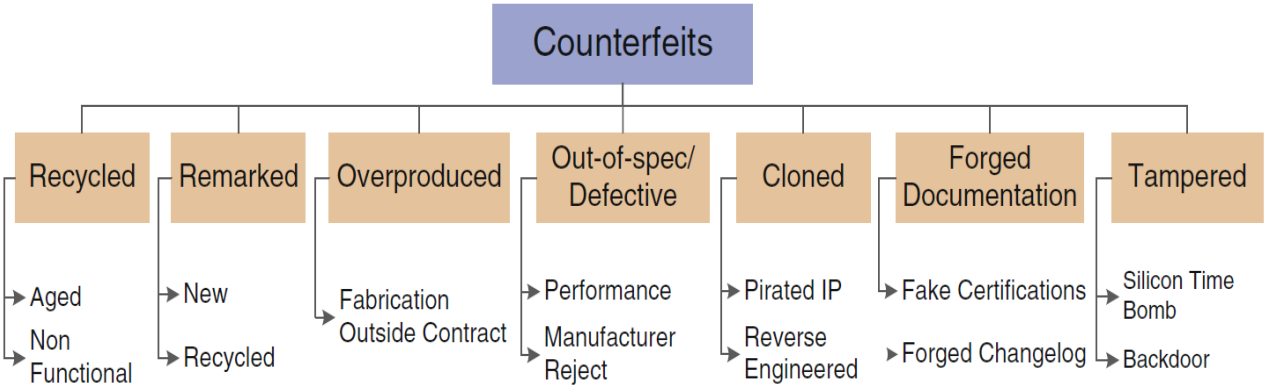


Figure 2.2: Taxonomy of Counterfeit types (from Tehranipoor et al. 2014 [14])

As recently as 2011, it was estimated that counterfeit circuits make up approximately 1% of the market with a financial loss of approximately $100 billion worldwide [5]. In addition, the number of reported counterfeiting incidents quadrupled between 2009 and 2011 [6]. Many of these counterfeit devices find their way into mission critical devices for the military and aerospace. These ICs can be of poorer quality and fail quicker than brand new devices from a trusted seller, as seen in Figure 2.3. A quicker fail time may lead to a number of ill-effects as specified in **Error! Reference source not found.**.

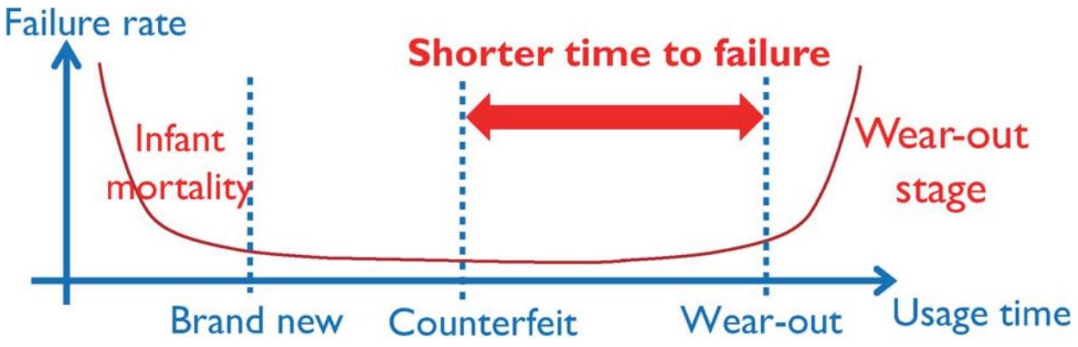


Figure 2.3: Classical bathtub curve, illustrating typical device failure characteristics[27]

Table I: Possible Effects of Counterfeit ICs [15]

Government	Industry	Consumer
National security or civilian safety issues	Costs to mitigate this risk	Costs when products fail due to lower quality and reliability of counterfeit parts
Costs of enforcement	Costs to replace failed parts	
Lost tax revenue due to illegal sales of counterfeit parts	Lost sales	Potential safety concerns
	Lost brand value or damage to business image	

With this increasing vulnerability of hardware IPs to piracy, investigation of IP protection techniques has become an emerging area of research. Recent investigations have targeted hardware IP protection benefiting IP vendors, IC designers, or both.

Existing solutions to protect IPs from piracy and reverse-engineering both in the evaluation phase and after sale, include **passive defenses like *digital watermarking*** which incorporates a hard-to-remove *digital signature* in an IP; wherein the designer embeds additional features (e.g. in the form of design constraints) into the Finite State Machine (FSM) which is similar to hardware Trojan insertion. It helps to establish the author's ownership in case of litigation. However, the added information is for proof of authorship, and does not carry any malicious functionality (as a hardware Trojan does). There is a rich body of research work on FSM watermarking for the protection of FSM design intellectual property. These techniques usually rely on the modification of the state transition graph at the behavioral synthesis level to embed a watermark related to user-specific information for identification purposes. Oliveira, in his famous paper, proposes creating watermarks based on a set of redundant states that can only be traversed when a user-specific

input sequence is loaded [Oliveira 2001]. Others have proposed watermarking schemes based on state encoding, or by introducing extra state transitions in the FSM to produce output that carries the watermark. An improvement on this method was suggested by utilizing the existing transitions for watermarking to successfully reduce the high-overhead caused by extra state transitions. These approaches are referenced in [9]. Additionally, some *soft IP* protection methods perform string modifications of the RTL plain-text to obfuscate it by affecting its human comprehensibility. However, they do not affect the *black box functionality* of the IP.

Other prevalent techniques to counter hardware IP Piracy incorporate **active defenses like encryption** (coupled with requirement to use vendor-specific tools), hardware metering, and layout level circuit obfuscation. In [6], an IC protection technique ensures that every instance of an IC manufactured in the foundry requires an instance-specific enabling pattern from the IC designer to be operational. It prevents the manufacturing of illegal clones of an IC in a fabrication house. In [5], a gate-level IP obfuscation technique has been proposed that allows the IP to be used only after a pre-defined *initialization key* (sequence of input vectors) is applied.

Note that these IP protection techniques are not directly applicable for protecting evaluation versions of an IP. Passive techniques for IP protection do not prevent a SoC designer from stealing the IP, cloning it, or performing illegal fabrication. For example; the digital watermark does not affect the functionality and usability of a stolen IP. The techniques in [5] and [6] which involve key insertion are only useful for protection of an IP post-evaluation, because the *initialization key i*s provided to a trusted design house after legal purchase.

At a closer analysis, all the existing cryptographic primitives have proofs of security based on two broad assumptions:

(1) **Read-proof hardware;** that is, hardware that prevents an enemy from reading anything about the information stored within it, e.g. due to the computational complexity of the task; and

(2) **Tamper-proof hardware;** that is, hardware that prevents an enemy from changing anything in the information stored within it. In particular, existing cryptographic schemes consist of an algorithm which the adversary knows, but cannot change, (i.e., the key is stored in tamper-proof hardware), and a secret key, which the adversary does not know and cannot change (i.e., stored in hardware which is both read-proof and tamper-proof).

2.1 Hardware IP Protection Approaches

Hardware IP protection has been investigated earlier in diverse contexts, addressing licensed as well as the pre-license evaluation version of an IP. Previous work on IP protection can be broadly classified into two main categories:

(i) Authentication based protection, and

(ii) Obfuscation based protection

Figure 2.4 shows the taxonomy of hardware IP protection techniques which are broadly classified into authentication-based IP protection and obfuscation-based IP protection.

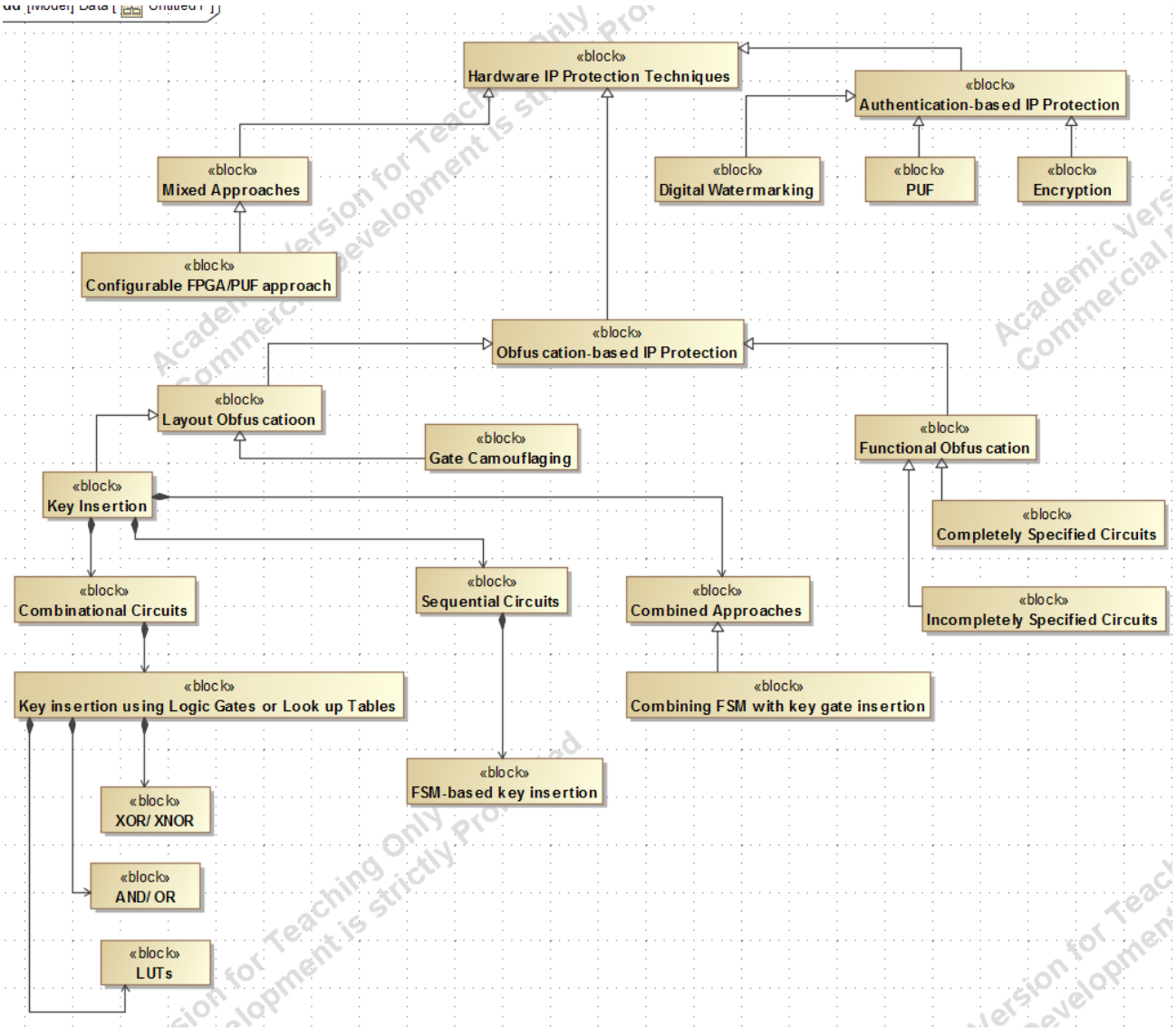


Figure 2.4: Taxonomy of Hardware IP Protection Techniques (see Appendix-1)

These two approaches are briefly explained next.

2.1.1 Authentication based IP Protection

To protect the rights of the IP vendor through authentication, the approaches proposed are directed towards embedding a Digital Watermark or identifier in the design which helps to authenticate the design at a later stage or in case of litigation. Since a digital copy of data is the same as the original, digital watermarking is a passive protection tool. It just marks data, but does not degrade it or control access to

the data. One application of digital watermarking is *source tracking* in a commercial enterprise (such as movie distribution). A watermark is embedded into a digital signal at each point of distribution. If a copy of the work is found later, then the watermark may be retrieved from the copy and the source of the distribution is known. This technique reportedly has been used to detect the source of illegally copied movies, and is now being applied to hardware design. Typically this is inserted by design modifications which result in one or multiple input-output response pair(s) which do not arise during the normal functioning of the IP. Such digital signatures are known only to the IP vendor. Since this digital watermark (or signature) cannot be removed from the IP, it helps to prove an illegal use of such a component in litigation. However, the effectiveness of authentication-based IP protection schemes is limited by the fact that these techniques are passive and hence they cannot prevent the stolen IP from being used as a black box.

The approaches directed towards protecting the rights of the IC designer, on the other hand, ensure that the design house has knowledge of every IC instance manufactured and sold in the market using a technique called ***'IC fingerprinting***' which ensures that a unique identifier is associated with every instance of the IC sold in the market. A recent IC fingerprinting technique has made use of observability and satisfiability don't care conditions inherent in the circuit to hide fingerprinting information which is essentially a unique ID for the integrated IC or batch of ICs. (Carson Dunbar and Gang Qu, 2014).

It is interesting to note that the need of obfuscation to protect ICs against possible reverse-engineering and copy was investigated long back in the mid-1970s, coinciding with the commercial release of the first generation microprocessors. However, the scalability of the technique to larger designs by adopting a systematic approach of hardware obfuscation was not explored. Usually, this is implemented by including a locking mechanism in the IC. The fabrication facility would require a unique bit sequence provided by the design house to "unlock" the IC. However, such approaches cannot prevent the possibility of reverse-engineering a design to expose its functionality as well as the security scheme. Moreover, they do not

address protecting the right of an IP vendor because illegal copies of the device can be used as a black box after the key has been revealed to the design house. In this case, a PUF-based approach is more authentic compared to a key-based encryption technique as it uniquely identifies each instance of the IC.

Authentication based on Physical Unclonable Function (PUF)

Physical Unclonable Function (PUF) is a multi-input hardware device that produces difficult to predict outputs that are unique to each instantiation of identical devices by extracting a silicon chip's inherent fabrication variability. When a physical stimulus is applied to the structure, it reacts in an unpredictable (but repeatable) way due to the complex interaction of the stimulus with the physical microstructure of the device (see Figure 2.5). This exact microstructure depends on physical factors introduced during manufacturing which are unpredictable (like a fair coin). The applied stimulus is called the challenge, and the reaction of the PUF is called the response. A specific challenge and its corresponding response together form a challenge–response pair or CRP. For example, in Figure 2.5, challenge bits C1,C2,C3,…CN are applied to the PUF at the rising edge of the clock. Each block in the PUF is a multiplexer that dictates the path of the bits through the structure. Analog timing difference on the two paths determines the response bit of the PUF, as determined by the Arbiter, which may be a D flip-flop. The device's identity is therefore established by the properties of the microstructure itself. As this structure is not directly revealed by the challenge-response mechanism such a device is resistant to spoofing and piracy attacks. These variability-induced delays or switching thresholds can be conveniently used for authentication as a PUF response will be unique and reliable even for chips manufactured from the same wafer.

Due to these features, PUF is one of the most popular and widely-used hardware security primitives. It is used in a variety of hardware-based security protocols ranging from generation of random numbers to secure storage of privileged information and public key cryptography.

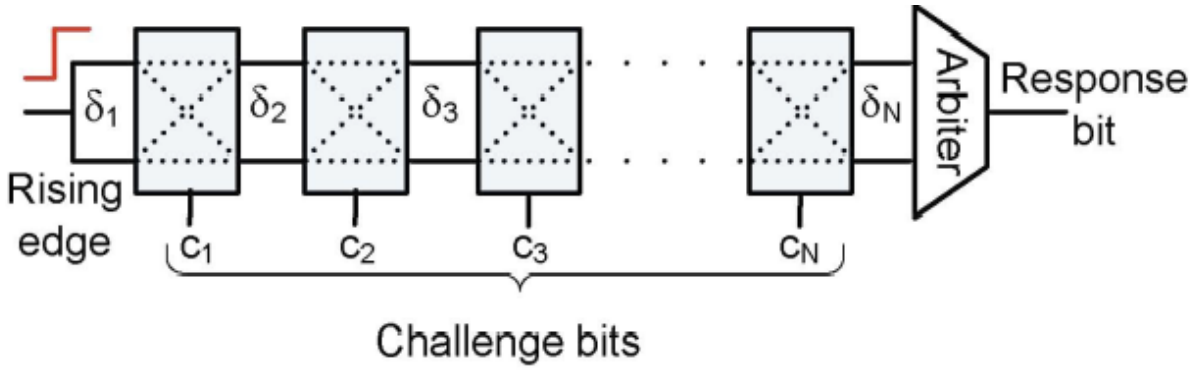


Figure 2.5: A Delay-based Arbiter PUF

However, until recently all PUFs have been analog devices and therefore greatly influenced by operational and environmental conditions and subject to change of their mapping function due to device aging. Very recently, digital PUFs have been developed that eliminate all these drawbacks of analog PUFs. Even more importantly, they are faster, require less energy, and can be easily integrated within conventional digital logic.

However, digital PUFs still face certain limitations. Of most importance are potential susceptibility to side-channel attacks and a relatively small number of challenge-response pairs. When using a PUF, particularly a delay based arbiter PUF as shown in Figure 2.5, the delay overhead will increase dramatically. More importantly, PUF has an inherent defect: reliability. If the reliability is not guaranteed, that is when the bit flips happen, the circuit will output wrong result.

The second kind of IP Protection approach is based on obfuscation, i.e. modifying the layout (structure) or functionality (behavior) of a circuit so as to make it difficult to reverse engineer i.e. more time-consuming or computationally expensive.

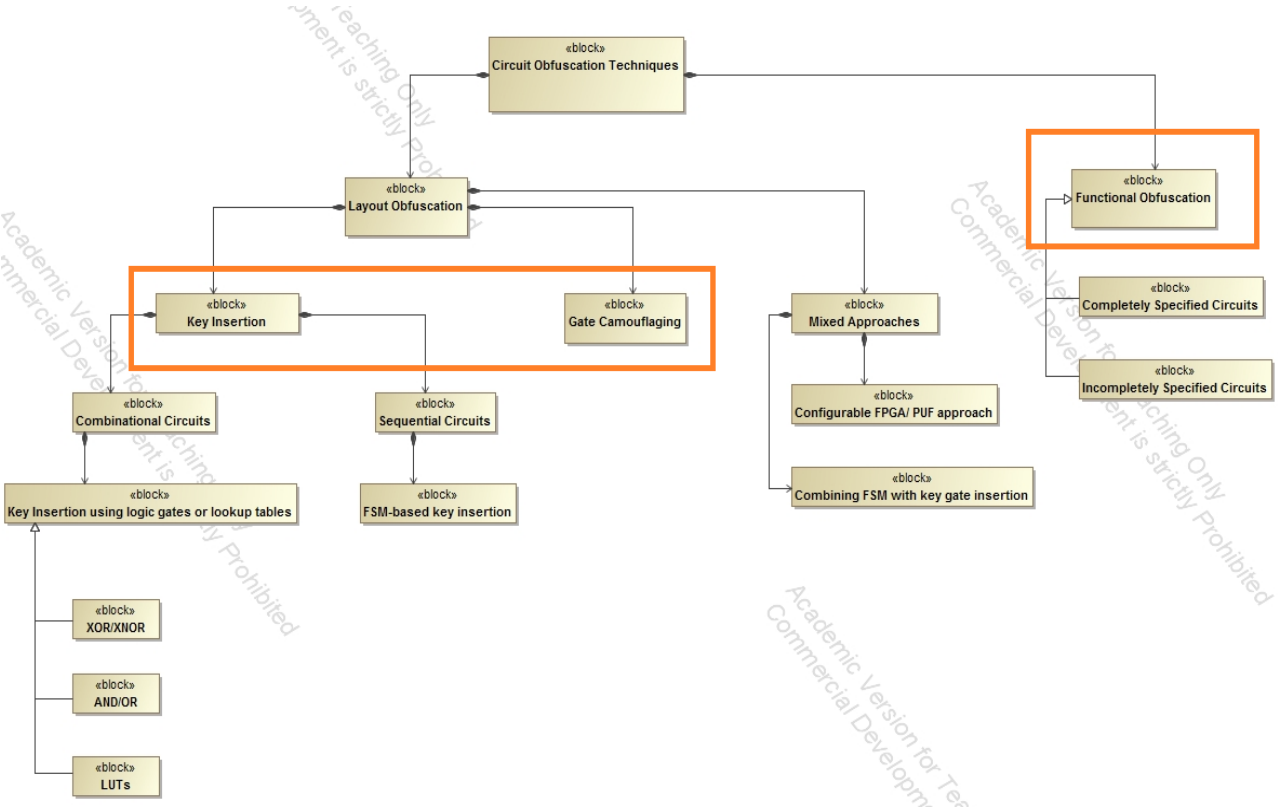


Figure 2.6: Circuit Obfuscation techniques (a) Layout Obfuscation, and (b) Functional or Logic Obfuscation

2.1.2 Obfuscation based IP Protection

Hardware obfuscation is a technique by which the structure or the functional description (behavior) of electronic hardware is modified to intentionally conceal its functionality, which makes it significantly more difficult to reverse-engineer. In essence, it is different from digital watermarking (where proof of ownership is concealed in the digital content itself), or from hardware intellectual property (IP) watermarking where the ownership information is embedded and concealed in the description of a circuit. Hardware obfuscation aims at minimizing the threats at IP or chip level discussed earlier, such as IP infringement, reverse engineering of the manufactured ICs (in fabrication facilities) to produce counterfeit or clone ICs; and malicious modifications of an IP through the insertion of hardware Trojan to cause in-field functional failure by making it difficult for an adversary to comprehend the actual functionality of a design through structural or behavioral modifications. Hardware obfuscation techniques can be classified into two main categories:

(a) the techniques like structural or layout-level modifications, which do not directly affect the functionality of the electronic system, and (b) those techniques which directly alter the functionality of the system.

2.1.2.1 Layout or Structural Obfuscation

This approach modifies the circuit layout without affecting its functionality. Layout-level or structural obfuscation is performed in two ways: (a) key insertion and (b) gate camouflaging.

2.1.2.1.1 Key insertion based approach

Key insertion based approaches insert the so-called key-gates (usually XOR and XNOR) to lock the circuit such that normal functionality of the obfuscated design can only be enabled by the successful application of a single pre-determined key (or a key from its equivalent class of keys) or a sequence of secret keys at the input; otherwise the circuit operates in a mode, which exhibits incorrect functionality. This can be done by embedding a well-hidden finite state machine (FSM) in the sequential circuit or by introducing key gates in a combinational circuit to control the functional modes based on application of key. One input of the key-gate will come from the primary input of the circuit so that the user can enter the key, the other input is a circuit node. The circuit will be unlocked when the inputs to the key gates are fed with a valid key. Otherwise the circuit will not perform the desired functionality. Multiplexers and LUTs have also been used to provide a logic barrier between normal and obfuscated modes of circuit.

The technique of key-based, active hardware obfuscation is similar in principle to private-key cryptographic approaches for information protection, since the "key sequence" for the obfuscated design plays a similar role as the cryptographic key. The technique can be applied at different levels of hardware description, namely gate-level or register transfer level (RTL) design and hence can be used to protect soft, firm or hard IP cores. In a practice widely adopted by the industry, the HDL source code is encrypted and the IP vendor provides the key to decrypt the source code only to its valid customers. However, this technique may enforce the use of a particular design platform, a situation that might be unacceptable to many SoC

designers who seek the flexibility of multiple tools from diverse vendors for design and evaluation purposes. Moreover, none of these techniques prevent possible reverse-engineering effort at later stages of the design and manufacturing flow.

Key Insertion Mechanism and Attack Strategy

The first and one of the most well-studied key insertion approach is the "EPIC" mechanism ("Ending Piracy in Integrated Circuits") proposed by Roy et al. in 2008 [6]. Figure 2.7 is a motivation example of their approach. It shows a circuit with two key-gates K1 and K2 with key values K1=0 and K2=1. We can see that with incorrect key values, the obfuscated circuit will not function correctly. Therefore, the success of this approach relies on the assumption that RE attackers can only study the input-output behavior revealing the functionality of the circuit. The weakness of EPIC, as pointed out in [15], is that: when an attacker opens up the chip and learns particulars of the design, he can use deliberately selected input signals to control the circuit such that the secret keys will be exposed at the output pins (for example, by setting I1=1, I2=0, and I3=0 in Figure 2.7 the value of K1 is sensitized to output O1). Therefore, the complexity of obfuscation can be reduced to linear time rather than the exponential time as expected. Even if brute forcing the circuit is not computationally feasible due to a large number of key gates, other logic decryption techniques have been proposed that use Boolean SAT solvers to resolve the key gates in the circuit, no matter how numerous [9].

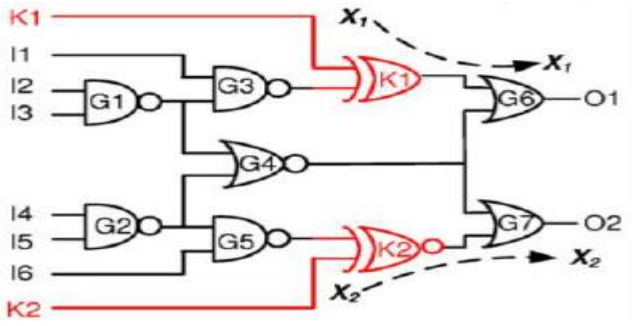


Figure 2.7: Key-gate Insertion: Motivation Example in [6]

Rajendran et al. [21] propose an enhanced obfuscation method to prevent the key values from being propagated to the output (as shown in Figure 2.8).

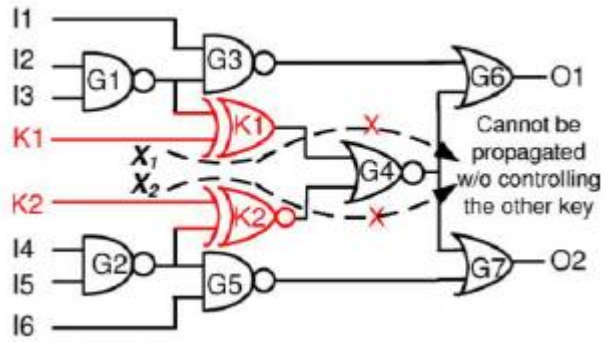


Figure 2.8: A smarter key insertion strategy to thwart key propagation to the output. K1 cannot be determined without knowing K2 and vice versa

They partition the circuit into the following categories:

1) **Runs of key-gates:** A set of key-gates connected in back-to-back fashion. They can be replaced by a single key gate, reducing the complexity of brute force.

2) **Isolated key-gates:** A gate that has no path from it to all the other key-gates. In this case, resolve the key gate individually, without considering the other key gate.

3) **Dominating key-gates:** Gate K2 is the dominating key-gate of K1 if K2 lies on every path between K1 and the outputs. In this case, the strategy is to "divide and conquer", i.e. by muting the gate K1 and simultaneously sensitizing the value of the dominating key gate to an output, to resolve key gate K1.

4) **Convergent key-gates:** Even if there are no paths between two key-gates, the sensitization paths might interfere. Such scenarios happen if these two or more key-gates converge. Depending upon the type of convergence, key-gates can be classified into 1) concurrently mutable, 2) sequentially mutable, and 3) non-mutable key-gates.

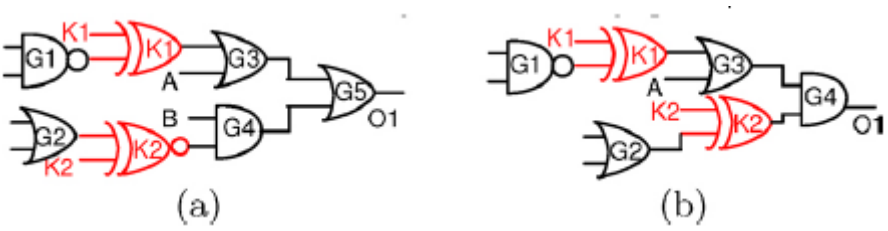


Figure 2.9: (a) Concurrently mutable key-gates: K1 and K2 converge at G5 and can be muted. (b) Sequentially mutable key gates: K1 and K2 converge at G4, but only K1 can be muted [15].

If two key-gates K1 and K2 converge at some other gate, such that neither of the key bits can be muted, then K1 and K2 are called non-mutable convergent key-gates. The only way to resolve such gates is through brute force [21]. Therefore, a designer would want to maximize the number of non-mutable key gates in the circuit.

The authors in [21] therefore recommend an insertion strategy to maximally reduce the probability of finding golden patterns (patterns that simultaneously sensitize the target key to the output while muting the other keys) and to increase the brute force search efforts. Their idea is to let the inserted key-gates interfere with each other. One has to try all the combinations of the keys rather than isolating, muting or sensitizing them. The weakness of this recommended approach is that a SAT solver can be used to reduce the effort required to brute force the circuit and extract the keys by controlling the inputs to the key gates. Another potential weakness is that don't care conditions inherent in the circuit can be used to effectively isolate the key gates from each other, reducing RE complexity. Once this occurs, it may be possible to find more golden patterns in the circuit and resolve the functionality of key gates (see section 3.3.3).

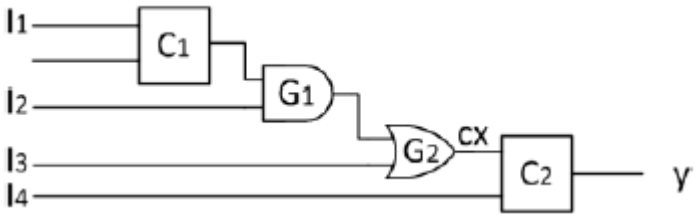


Figure 2.10: Don't care conditions in the circuit lower RE Complexity; C2 can be resolved easily by setting I3=1 and controlling I4, essentially making it an isolated key-gate

Finally, another defect is that the fan-in cone of different outputs will break the interference of key-gates, making RE attacks much easier than anticipated. A key gate that falls outside of the intersection of two or more outputs can be easily isolated and sensitized to its output. Locating such gates will bypass the influences from other key-gates in other cones (see 3.4.2). We use this observation to come up with an arguably stronger encryption strategy in Chapter 3. Next, we mention an emerging technique for logic decryption using Boolean SAT solvers. This technique is very effective against encryption algorithms that do not judiciously select key gates and that do not camouflage the said key gates. Judicious selection here means selecting gates that are non-mutable and cannot be isolated from each other.

Logic Decryption Attacks using SAT Solvers

The attacker's goal is to determine the key values to unlock the circuit. Once the attacker unlocks the circuit, he/she can extract the functionality for some malicious usage. The key insertion based obfuscation approach illustrated in Figure 2.8 is not very effective against such an attacker, even if brute forcing the circuit is computationally infeasible. An explanation of the weakness of the approach follows.

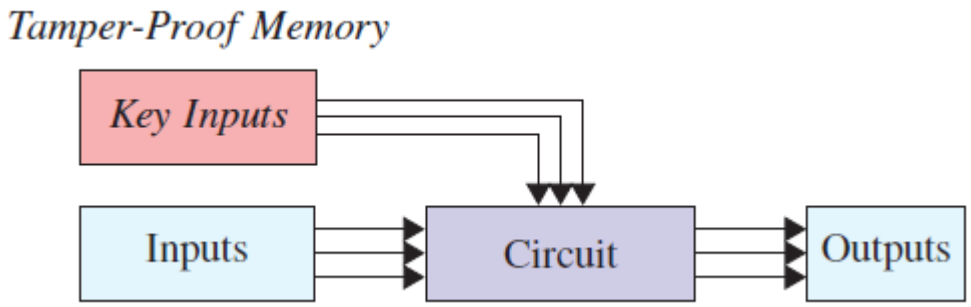


Figure 2.11: Overview of Key Insertion based Approach

Key insertion approach rests upon the assumption that *the foundry does not know and cannot compute* the correct values of the key inputs. Otherwise, the foundry could just program these values and overproduction could not be prevented.

Attack Model: Considering a malicious foundry, we assume the attacker has access to layout and mask information. The gate-level netlist can be reverse-engineered from this.

We also assume that the attacker has access to an activated IC on which to apply input patterns and observe outputs. This could be obtained by purchasing an activated IC from the open market. *The components of our attack model are therefore: (i) a gate-level netlist of the encrypted IC and (ii) a means for applying arbitrary input patterns and observing the resultant outputs on an activated IC.*

Potential Attacks: Given the above attack model, an attack is possible when an attacker can determine the correct values of the key inputs. Let us consider potential attacks.

The naive idea of brute-force search does not work. If the circuit has *M* inputs and *L* key inputs, this requires 2*M* observations from an activated IC and $O(2^{M+L})$ computations on the encrypted design. Clearly, this is not practical.

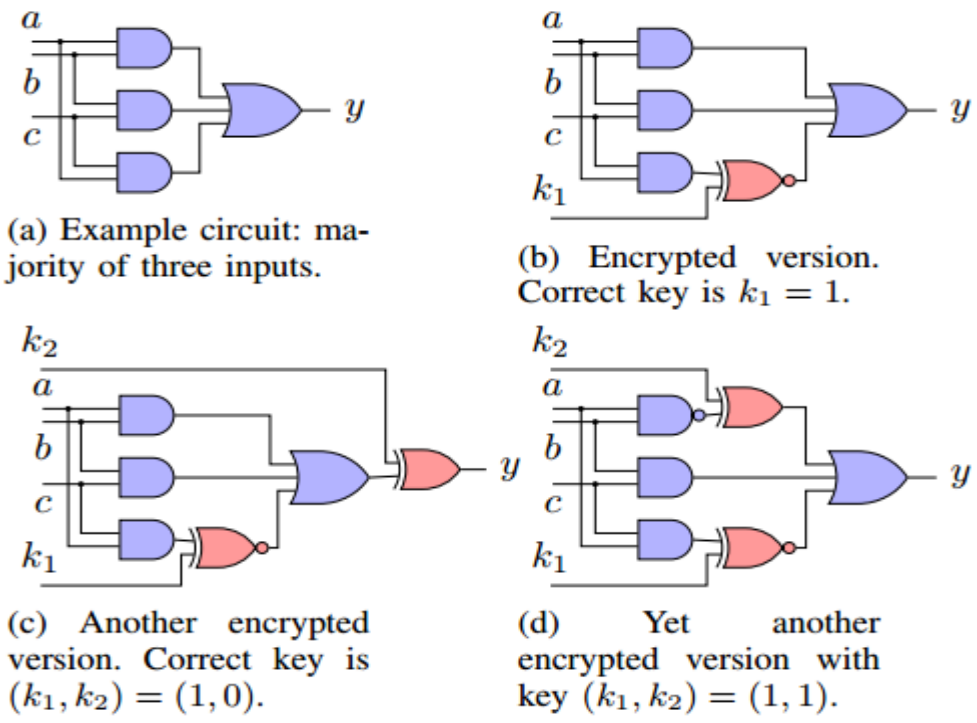


Figure 2.12: Some simple key-insertion examples [9]

Rajendran et al. [15] propose using automatic test pattern generation (ATPG) tools to generate input patterns that expose the value of a key input. In Figure 2.12(b), when the input is $a = b = c = 0$, the output is $y = \neg k1$. Therefore, an ATPG tool can find such patterns to reveal keys. But in Figure 2.12(c), no input pattern can expose $k1$ if $k2$ is also unknown. The solution is to first attack $k2$ with the pattern $a = b = c = 1$. From this we deduce $k2 = 0$. Now attacking $k1$ is possible. But even this strategy does not guarantee a successful attack. For the circuit in Figure 2.12(d), there is no single input pattern that can reveal the value of either $k1$ or $k2$. ***Therefore, if logic encryption is done carefully, the fault-analysis attack is ineffective [9].***

Formal analyses can potentially be used to attack logic encryption. It is possible to formulate the attack as a solution to a Quantified Boolean Formula (QBF). Suppose the original circuit is represented by the Boolean function $C_o(X)$, and its encrypted version is represented by $C(X, K)$. Here X and K are the circuit's primary inputs and key inputs respectively. We can retrieve the key by solving the QBF:

$$\exists K \; \forall X \; C(X, K) = C_o(X)$$

In plain English, the QBF attempts finds an assignment to the key inputs K such that for all values of the primary inputs X , the functions C and C_o are equal. *While we do have relatively efficient algorithms to solve this particular type of QBF, the formulation itself is moot. The attacker does not have access to the unencrypted IC's gate-level netlist and cannot construct the formula* $C_o(X)$.

The attacker can only observe the outputs for a small set of input patterns on the activated IC and must somehow determine the key values from these observations. Given a set of input/output observations, a SAT solver can determine a key value that is consistent with *these* observations. But this key value may not be correct. For example, suppose we make the observation $y = 1$ when $a = b = c = 1$ in Figure 2.12(b). Given this information, a SAT solver may assign $k1 = 0$ since this value is consistent with the above observation. Even if we provide additional observations to the solver, *e.g.*, $(a, b, c, y) = (1, 1, 0, 1)$ and $(a, b, c, y) = (0, 1, 1, 1)$, $k1 = 0$ is still a valid solution. *While we are guaranteed that the solver returns a key value that yields the correct output for the input patterns observed thus far, there may be other input patterns for which this key value produces incorrect output.* Furthermore, even if the SAT solver returns the correct key, verifying its correctness seemingly requires evaluation of outputs for all possible input patterns (2^M in a circuit with M inputs). Clearly, it is impractical to apply these many input patterns and so a simple SAT formulation is not enough.

Malik et.al. [9] provide interesting insights to resolve this SAT problem. First, instead of considering key values individually, they consider *equivalence classes of keys*. They define two keys $K1$ and $K2$ to be equivalent, denoted as $K1 \equiv K2$, if and only if for each input value Xi, the encrypted circuit produces the same output value Yi for both keys $K1$ and $K2$. Precisely stated:

$$K1 \equiv K2 \text{ iff } \forall Xi : C(Xi, K1, Yi) \wedge C(Xi, K2, Yi).$$

The intuition is that instead of finding *the* correct key, they are looking for *a member of the equivalence class of keys* which produces the correct output for all input patterns. To "zero-in" on the correct equivalence class, they iteratively rule out equivalence classes which produce the wrong output value for at least one input pattern. Given two key values K1 and K2, define the input pattern X*d* as a *distinguishing input pattern* if the encrypted circuit outputs different values Y*d*1 and Y*d*2 when the key inputs are set to K1 and K2 respectively. More precisely, X*d* is a distinguishing input pattern for K1 and K2 iff *C*(X*d,*K1,Y*d*1) ∧ *C*(X*d,*K2,Y*d*2) ∧ (Y*d*1≠ Y*d*2).

The second insight is that if a distinguishing input pattern X*d* is found, then they could examine the output of the activated IC for input X*d* and use this to rule out one (or both) of K1 and K2 as not being in the equivalence class of correct keys. This suggests Algorithm 1 which repeatedly finds distinguishing inputs (*line 4*) for some two keys *K*1 and *K*2, while asserting that the encrypted circuit must have outputs consistent with the input/output patterns observed thus far on the activated IC (*line 6*). The loop ends when no distinguishing inputs can be found. The correct key value is any assignment to *K*1 or *K*2 that satisfies the formula *Fi* (*line 9*).

Algorithm 1 Logic Decryption Algorithm

Function: *decrypt.*
Inputs: C and *eval.*
Output: $\vec{\mathrm{K}_\mathrm{C}}$.

1: $i := 1$
2: $F_1 = C(\vec{X}, \vec{K}_1, \vec{Y}_1) \wedge C(\vec{X}, \vec{K}_2, \vec{Y}_2)$
3: **while** $sat[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$ **do**
4: $\quad \vec{\mathrm{X}}_i^d := sat\_assignment_{\vec{X}}[F_i \wedge (\vec{Y}_1 \neq \vec{Y}_2)]$
5: $\quad \vec{\mathrm{Y}}_i^d := eval(\vec{\mathrm{X}}_i^d)$
6: $\quad F_{i+1} := F_i \wedge C(\vec{\mathrm{X}}_i^d, \vec{K}_1, \vec{\mathrm{Y}}_i^d) \wedge C(\vec{\mathrm{X}}_i^d, \vec{K}_2, \vec{\mathrm{Y}}_i^d)$
7: $\quad i := i + 1$
8: **end while**
9: $\vec{\mathrm{K}_\mathrm{C}} := sat\_assignment_{\vec{K}_1}(F_i)$

Logic Decryption Algorithm suggested in [9]: the algorithm uses SAT solvers to resolve key gates and reduce complexity of brute force. Here C is the Boolean formulation of the gate-level circuit and eval is the output from the legitimate activated IC.

We note that other attacks are possible. For example, an attacker may evade/subvert the tamper-resistant packaging and probe the signals corresponding to the key inputs. However, we do not consider side-channel analysis attacks in this discussion.

2.1.2.1.2 Gate Camouflaging or Replacement

Gate camouflaging takes the key insertion approach one step further by not only judiciously selecting key gates so that they interfere with each other, but also replacing these key gates with configurable cells (toform so called camouflaged gates) that can perform one of many equally likely and equally valid logic functions. This thwarts attempts by an attacker with boolean SAT solving capability. This approach takes advantage of the configurable feature of introduced cell. Camouflaging is a layout-level technique that hampers an attacker from reverse engineering by introducing, in one embodiment, dummy contacts into the layout. By using a mix of real and dummy contacts, one can camouflage a standard cell whose functionality can be one of many. The true contacts dictate the functionality of a camouflaged gate. The use of optical and electrical microscopy will fail to identify the hidden functionality as they cannot differentiate between true and dummy contacts, complicating reverse engineering. Hence, if an attacker cannot resolve the functionality of a camouflaged gate, he/she will extract an incorrect netlist.

The camouflaged cell can be configured as different logic gates without being observed. Several novel techniques have been proposed to implement camouflaged form of basic logical gates. In one embodiment of IC camouflaging presented in [15], the layouts of logic gates are designed to look identical, resulting in an incorrect extraction. For example, the layout of regular NAND gate and NOR gate look different and are hence easy to differentiate on inspection. However, the layout of camouflaged NAND gate and camouflaged

NOR gate look identical and are difficult to differentiate. When deceived into incorrectly interpreting the functionality of the camouflaged gate, the attacker may obtain a reverse engineered netlist that is different from the original, where the functionality of the camouflaged gates is arbitrarily assigned.

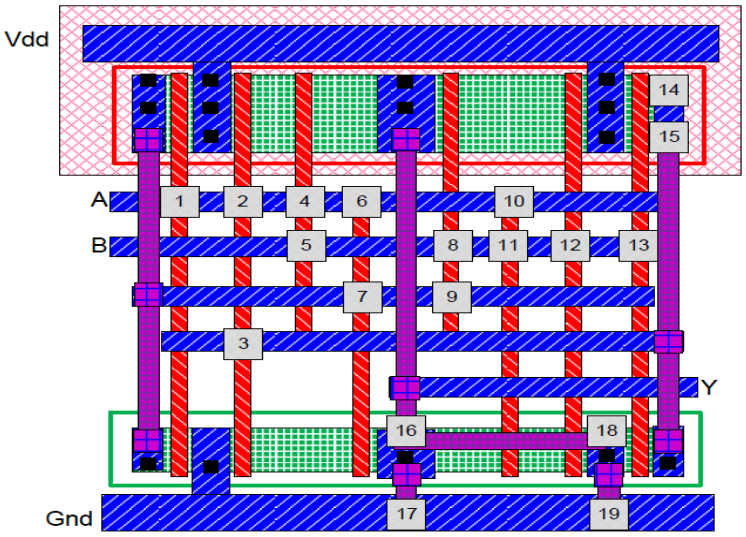


Figure 2.13: A generic camouflaged layout that can perform either as an XOR, NAND or NOR gate based on which contacts are true and dummy. The numbered boxes are the contacts [15].

Table II : List of true and dummy contacts to realize different functions using the camouflaged layout shown in Figure 2.13 (from [15])

Function	Contacts	
	True	Dummy
NAND	2, 4, 6, 8, 11, 12, 16, 17	1, 3, 5, 7, 9, 10, 13, 14, 15, 18, 19
NOR	2, 5, 6, 11, 12, 18, 19	1, 3, 4, 7, 8, 9, 10, 13, 14, 15, 16, 17
XOR	1, 3, 4, 7, 9, 10, 12, 13, 14, 15, 18, 19	2, 5, 6, 8, 11, 16, 17

From the top view of the chip, both the true and dummy contacts appear identical under a microscope. Therefore, by carefully selecting gates to camouflage, the outputs can be controllably corrupted.

The more functionalities a camouflaged standard cell could implement, the more difficult the reverse engineering becomes. It is essential for a camouflaged standard cell to have a large number of transistors that can be connected in different ways to realize different logic functions. Of all the gates in the standard cell library, XOR and XNOR gates have the highest number of transistors [12]. XOR/XNOR gates are also more difficult to resolve using boolean SAT solvers. Hence, most gate camouflaging based obfuscation approaches use XOR/ XNOR gates as one of the arbitrary functional assignments. In section 3.4, we show how to achieve even greater versatility in the number of logic functions that can be performed by a camouflaged gate using a 4x1 multiplexer as recommended by Baukus et.al in [17].

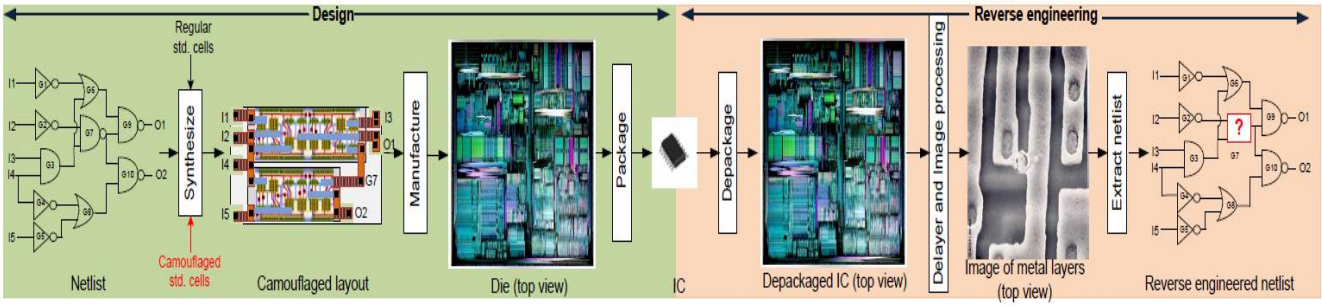


Figure 2.14: Camouflaged Circuit Design Flow [15]: A design is synthesized into layout by using both regular standard cells and camouflaged standard cells. This layout is manufactured to obtain the IC. An attacker buys the camouflaged IC and depackages it. He/she delayers and images the metal layers and transistors. He/she then obtains the gate-level netlist by processing those images. However, the functionality of camouflaged gates cannot be resolved. For example, the functionality of G7 cannot be resolved.

Figure 2.14 shows how camouflaging protects an IC design against reverse engineering. A designer camouflages certain gates in the design. For example, the NAND gate, G7, in Figure 2.14 is camouflaged. This design with camouflaged gates is then manufactured at a foundry. The manufactured IC is sold in the market. An attacker reverse engineers an IC by depackaging, delayering, imaging the layers, and extracting the netlist. However, in the extracted netlist, the functionality of the camouflaged gates are unknown. For example, in Figure 2.14, the functionality of G7 is unknown. An attacker assigns an arbitrary two-input function to it or uses a Boolean SAT solver to try to resolve the functionality. If the position of the camouflaged gate in the circuit is chosen methodically so that it interferes with other camouflaged gates,

the attacker has to resort to brute force to resolve the functionality of a camouflaged gate. Consequently, he/ she may obtain an incorrect netlist. Hence, the camouflaged gates are chosen in such a manner that their output cannot be observed by controlling other input port values in the circuit. This is explained further in Chapter 3 when we present our obfuscation methodology.

Hardware watermarking can be used in conjunction with hardware obfuscation. In an obfuscated design, watermarking can be effective in providing a second line of defense against piracy efforts through design authentication.

However, there are some *passive* techniques that modify the circuit description in a *soft* form (e.g. syntactic changes), such that it becomes difficult for a human reader to understand the functionality of the circuit. These approaches typically employ either string-substitution (including variable name change, comment removal, etc.), or structural change in the hardware description language (HDL) description of a circuit (including loop unrolling, register renaming, etc.). A major shortcoming of the *passive* approaches is that they do not modify the black box functionality of a circuit, and hence cannot prevent potential usage of an IP as *black box* in a design. Moreover, the actual strength of such passive obfuscation is debatable since in general, *black box obfuscation* does not exist, at least for software programs computing certain mathematical functions.

The existing decryption techniques mostly focus on information leak from a hardware system: An adversary may extract cryptographic keys and confidential information from a system by testing, reverse engineering, or side-channel analysis. Similarly, circuit layout obfuscation can be thwarted if the adversary has access to modern computation tools like SAT Solvers and complete or partial mask and/or layout information. More importantly, both IP Protection approaches do not address the potential use of the IP by an adversary as a black box.

2.1.3 Mixed or Combined Approaches

In one approach, simultaneous obfuscation and authentication of the design is achieved for an arbitrary circuit by replacing it with PUF-based logic. PUF-based logic directly implements the original functionality of the replaced circuit while also hiding its functionality. Since the functionality of the PUF is a byproduct of manufacturing processes, its characterization is not known until after fabrication. Once fabricated, the designer characterizes the PUF through special supporting hardware, then burns the pertinent access wires to disallow any subsequent unauthorized characterization. Now that the PUF's unique functionality is known only to the designer, an FPGA fabric is programmed using standard synthesis tools in order to implement the original replaced circuitry in conjunction with the corresponding PUF. In this way, part of the original circuit is replaced by an FPGA. Therefore, simultaneous obfuscation of the design is achieved along with an authentication signature in the form of challenge-response pair(s) from the PUF-based implementation.

In another mixed approach, key-gate insertion is combined with FSM-based key approach to obfuscate the sequential circuit.

2.2 Motivation for this research

As described above, authentication-based and obfuscation-based IP protection are two of the main approaches used for hardware IP obfuscation today. Both have drawbacks that limit their effectiveness against IP piracy. Mainly, authentication-based approaches are passive, in that they do not prevent the IP from being used as a black box. Similarly, the layout level obfuscation technique described above does not prevent an adversary from knowing the high-level functionality (input-output pairs) of the circuit.

In this research, we explore a novel hardware IP obfuscation approach that incorporates layout obfuscation using don't care conditions and also propose proof of concept of a logic (or functional) obfuscation technique that effectively protect a design against IP theft by malicious users by adding a second layer of

complexity in the form of modification of sub-circuit functionality, while incurring low hardware and computational overheads and meeting overall specifications. This not only makes derivation of gate-level characteristics of the circuit difficult but also limits usage of the IP as a black box because parts of the circuit do not perform 'as expected' and it is computationally expensive to find out the true circuit functionality, right down to the sub-circuit level. To the best of our knowledge, this is the first recommended approach to develop a systematic and provably robust hardware obfuscation approach that enables hardware protection at different stages of the IC design cycle, including circuit level or gate-level, using functional obfuscation of the design.

Effectiveness of protection against IP reverse-engineering and piracy at the layout level using don't care conditions is evaluated with ISCAS-85 benchmark circuits. First, this method offers a new approach to finding valid (non-mutable) obfuscatable locations and a corresponding exponential increase in the reverse engineering effort. Second, experiments conducted on ISCAS-85 benchmark circuits show an acceptable overhead and ease of detection of obfuscation locations by the developer and difficulty to detect or remove by the adversary. These locations can then be used with some modifications as proof of authorship by the developer. Finally, the method will reduce the cost of manufacturing by being able to be finalized in the post-silicon stage of the IC design cycle.

2.3 Dissertation Outline

This dissertation shows the work done to construct two procedures for creating practical obfuscated circuits at minimal area and delay overheads that can be finalized in the post-fabrication stage of the design cycle, thus reducing their cost of implementation. Chapter 2 discussed the background of hardware obfuscation techniques, covered necessary details on the current state of the art as well as related IC security primitives. Chapters 3-5 of the dissertation will describe our new technique for layout (structural) and functional (behavioral) obfuscation of digital circuits using don't cares.

Specifically, Chapter 3 presents the motivations, assumptions and techniques behind layout obfuscation and implements some important recommendations that first appeared in [13]. Chapter 4 gives two methods of layout obfuscation using Satisfiability Don't Cares and Observability Don't Cares inherent in the circuit along with simulation results for ISCAS-85 Benchmarks. The first method attempts to obfuscate the valid function of the circuit by using satisfiability don't cares, which occur in almost every combinational circuit of non-trivial size. The second method utilizes observability don't cares which also occur frequently in ICs, for the same purpose. Chapter 5 also develops the idea further to functionally obfuscate the circuit, so that not only the layout but the sub-circuit functionality is altered. Two examples of functional obfuscation are presented as proof-of-concept. Chapter 6 gives an overview of GateVision PRO, a netlist visualization tool that serves as a useful complement to the obfuscation algorithm detailed in Chapters 4 and 5. Finally, the dissertation conclusions are presented in the same chapter.

2.4 Main Contributions

1. Attack Model for Hardware Obfuscation and enumeration of Attacker's Goals and Capabilities. The attacker may be a rogue element in the IC supply chain or a malicious end-user.

2. Our obfuscation methodology uses don't care conditions inherent in the circuit to camouflage selected gates in the circuit layout. We also propose proof of concept of a novel hardware IP protection technique based on logic or functional obfuscation, as opposed to just layout level obfuscation studied in literature that also utilizes don't care conditions. This technique modifies sub-circuit functionality while meeting the overall circuit specification.

3. A smarter Gate Replacement strategy that overcomes the deficiencies in gate selection approaches studied in key insertion-based and gate camouflaging approaches [6, 21].

4. A secure IC design methodology based on our gate selection and replacement algorithm that can be seamlessly integrated into the existing IC Design flow using EDA tool support. Some steps of the camouflaged IC Design flow can be moved to post-fabrication stages of the design cycle (i.e. routing and placement) to lower overall costs by manufacturing "different" secure ICs with "different" don't care conditions to strengthen security. Such steps, e.g. configuration of 4x1 Multiplexers, the modifications using ODCs and the selection of deliberately introduced invalid functions may be deferred to the post-fabrication stage (routing and placement). However, it should be noted that the fan-out of camouflaged gates may affect the critical path delay.
5. Simulation results that demonstrate relatively low design overheads for ODC and SDC-based Obfuscation and directions for implementing delay-driven or performance driven obfuscation.
6. Some recommendations for future work including derivation of output corruptibility metric, gate selection and obfuscation strategies based on output cone extraction and input cone expansion; combining don't care based fingerprinting for simultaneous authentication and obfuscation; and analysis of the effect of design obfuscation on circuit and system testability and reliability.

"Though this be madness, yet there is method in't"

--Shakespeare, Hamlet

CHAPTER 3: LAYOUT OBFUSCATION: MOTIVATIONS, ASSUMPTIONS AND TECHNIQUES

In this chapter, we will briefly present the essential motivation, assumptions and techniques behind key insertion and gate camouflaging-based layout obfuscation and build on this method to define the Attack Model for hardware obfuscation and thereby develop a logic-based security primitive for obfuscation using don't care conditions.

The basic idea behind layout or gate-level obfuscation is selecting logic gates from the circuit and camouflaging them (i.e. replacing them with another kind of gate that performs the same function but can also perform other logic functions in addition to this function, which complicates malicious reverse engineering) in such a manner that it is computationally expensive to derive the complete gate-level description of the circuit. In this way, the best an adversary can hope for is determining the input-output relationships of the IC and possibly using the IP as a black box. The heart of the approach is judicious gate selection that overcomes sensitization of key-gate values to the primary output and gate camouflaging which hampers an attacker from reverse engineering by introducing, in one embodiment, dummy contacts into the layout. By using a mix of real and dummy contacts, one can camouflage a standard cell whose functionality can be one of many. If an attacker cannot resolve the functionality of a camouflaged gate, he/she will extract an incorrect netlist (see section 2.1.2 for details). Judicious selection of gates to camouflage ensures that their correct functionality can only be resolved by brute force and that the outputs of the extracted netlist are controllably corrupted. The techniques leverage IC testing principles such as justification and sensitization. The important question then is how to select the gates to camouflage

out of the tens of thousands of gates in the IC so as to maximize the effort required to reverse engineer? The following example illustrates the concept, and points out the weakness in the current gate-level obfuscation approach (Rajendran et.al [15]).

3.1 Example of Gate Camouflaging: The Attack Model

The attacker may be a rogue element in the supply chain or a malicious end-user with the aim of illegally copying the IP design. The attacker can get the high level functional description (primary input-output relationships) from the secure chip whose layout is obfuscated, but he/she would like a complete gate-level description so that the same functionality can be implemented by a different logic.

By comparing the outputs of a secure activated chip with an unpackaged chip simulated in software (using SAT solvers and logic synthesis tools, if needed) the attacker will not be able to extract any "useful" layout information due to the presence of camouflaged gates and the fact that sensitization and justification will not work because the camouflaged gates belong to the same intersection cone (see Section 4.2 and 5.2 for details of the obfuscation methodology). So the best the attacker can hope to achieve is a high-level description (which can be obtained by purchasing a secure chip off the market). The attacker would not be able to replicate the design, and sub-circuit functionality will be ambiguous to him.

Suppose the attacker:

(i) has tools to reverse engineer an IC, i.e., the setup to delayer an IC, optical microscope or SEM (Scanning Electron Microscope) to image the layers, and image processing software tools so that he can image the top layer.

(ii) can differentiate between a camouflaged standard cell and a regular standard cell from the images of different layers. This is possible because the images of regular and camouflaged standard cells are publicly available.

(iii) knows the list of functions that a camouflaged cell can implement.

3.2 The Attacker's Goals

1) Goal-1 is to extract high-level circuit description (input-output relationships) so that the attacker can rebuild the IC with the same functionality but different implementation logic, making it difficult to identify the violation of copyright.

2) The attacker wants to use RE to fully unveil the design IP and find exact layout, (Goal-2). Besides understanding the functionality, the attacker may also want to directly copy the original design because it has been well optimized in terms of power, delay, and area.

Attacker's RE method

1) Buys two copies of the target chip from the market.

2) Extracts the netlist with camouflaged gates from the first chip by depackaging, delayering and imaging the top layer using scanning electron microscope and rebuilds the circuit with unknown or camouflaged gates in computer software. (A2)

3) Solves the SAT problem to control the inputs of each camouflaged cell, even though the configurable (camouflaged) cells could be hidden deeply in IC and far away from primary inputs (A3).

4) Uses the other activated (packaged) chip to traverse and compare the possible inputs for every camouflaged cell and collect its corresponding outputs, using the principles of justification and sensitization mentioned at the end of section 3.2. Then use the input output-pairs to build the truth table (A1).

5) Uses the truth table to guess out the hidden logic of each configurable cell through the truth table. If the cells interfere with each other and cannot be solved directly,

6a) For Goal-1 (extracting high-level input/output relationships): Rebuild the obfuscated part using truth table (i.e. determine input-output relationships for each obfuscated gate)

6b) For Goal-2: Try all the possible logic combinations for inputs to each configurable cell (e.g. in the brute force way) until finding one combination whose outputs perfectly match the truth table values under all the inputs.

8) Resolve the functionality of camouflaged gates.

Suppose that a camouflaged cell can implement one of three logic functions {XOR, NAND, NOR}.

The objective of a reverse engineer is to determine the function implemented by every camouflaged gate and thereby extract the complete gate-level representation of the circuit. The second objective is to obtain all possible input-output relationships.

Therefore, the critical step above is to generate the input patterns that help resolve the functionality of the camouflaged gates. The following two principles form the foundation of VLSI testing for manufacturing defects in ICs [15]:

Justification: The output of a known gate can be justified to a known value by controlling one or more of its inputs. For example, an AND gate's output can be justified to '0' by setting one of its inputs to '0.'

Sensitization: A net can be sensitized to an output by setting all the side inputs of each gate in between that gate and the output to the non-controlling value of the gate. This way the value on the net is bijectively mapped to the value on the output.

The camouflaged cells are chosen in such a way that justification and sensitization do not reveal the characteristics of a particular gate. This gate selection and replacement method is explained in section 3.4. Therefore, the attacker may be able to use the IP as a black box and extract the high-level circuit description (input-output relationships) but he/she will not be able to able to extract any "useful" layout information that might allow him/her to understand the function and implement the same with a different logic and claim as one's own design. This is because sub-circuit functionality is concealed by introducing an "intentionally incorrect" or invalid function while ensuring that the circuit meets the overall specification by using don't cares to nullify the invalid function or always select the valid function only. So Goal-1 is achievable for an attacker with access to advanced logic synthesis tools and Boolean SAT solvers, but Goal-2 is thwarted because the circuit functionality is ambiguous. Our functional obfuscation approach introduced in Chapter 4 makes RE difficult for an attacker with Goal-1 by using redundant primary inputs to modify sub-circuit functionality.

In future work, we recommend combining our layout obfuscation with don't care based fingerprinting approach introduced in [20] to overcome an adversary with Goal-1 and further minimize attempts at illegal copying and re-use.

3.3 Some Observations from the Analysis of RE Attack Strategy

From the attacking scenario above, the security engineer should notice that:

3.3.1 Lemma 1: Simply hiding the configurable cells deeply inside the IC will not increase the RE attack complexity, if these gates are on the path from primary inputs to known gates, SAT Solvers can be used to control the inputs to these gates.

Definition*: Obfuscated inputs are the inputs of configurable cells. RE complexity is number of attempts to solve the functionality of IC by exhaustive search.*

If the obfuscated inputs of configurable cells are very close to the primary inputs, attacker can easily access these cells and traverse the obfuscated inputs. For instance, the inputs of cells can be directly accessed if they are part of IC's primary inputs. Therefore, a security engineer may have illusion that the deeper the cell is buried in the IC, it will be inaccessible and therefore, safe. However, wherever the cell is, its inputs can be easily accessed by solving the boolean SAT problem [9]. Even though SAT is the first NP-complete problem, there are many fast and efficient tools available like miniSAT, Lingeling, picoSAT etc to solve SAT circuit problem in conjunction with EDA tools.

3.3.2 Lemma 2: Isolated camouflaged cells can be solved individually in linear time. Interference of these cells will increase the complexity to exponential time.

Two gates being isolated means that there exists no path that connects these two gates. For an individual 2-input camouflaged cell, the attacker can traverse all possible inputs {00, 01, 10, 11} and read the corresponding output to resolve the gate functionality. So the RE complexity of a single cell is 2^2=4. Suppose we replace N two-input logic gates, and all of them are isolated from each other. The complexity in this case is 4N, and the obfuscated IC can be resolved in linear time. However, if we replace the gates which are interfering with each other so that one cannot be fully resolved without knowing the other, the complexity of reverse engineering increases. To figure out or resolve the functionality of these two cells, the attacker cannot test them individually. This is due to the complications arising from justification and sensitization in the presence of multiple interfering camouflaged gates. Therefore, all the inputs of these two gates have to be traversed together, which dramatically increases the RE complexity.

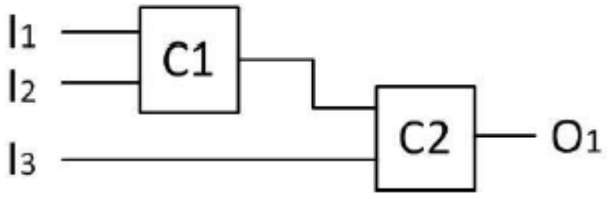


Figure 3.1: Two interfering camouflaged gates

To solve C1 in Figure 3.1, the attacker must solve C2 first. So the RE complexity, under Goal-1 (determining high-level input-output relationships) is 2^3=8.

Rajendran et al. [21] have done outstanding work in gate replacement based obfuscation. They divide the camouflaged cells into the categories based on the cells' location:

1) **Isolated cells:** As we discussed before, isolated camouflaged cells can be solved individually in linear time. For example, assume that the camouflaged gates C1 and C2 in Figure 3.2 can each perform one of {XOR, NOR, NAND} functions, The functionality of C1 can be resolved to be XOR by applying '010XXXX' at the inputs. This input pattern will justify the inputs of C1 to '00' and sensitizes the output of C1 to O1. If O1 is'0', then the functionality of C1 is resolved as XOR. Otherwise, the functionality of C1 can be resolved to be NOR by applying '110XXXX' at the inputs. This input pattern will justify the inputs of C1 to '10' and sensitize the output of C1 to O1. If O1 is '0', then the functionality of C1 is resolved as NOR. Otherwise, the functionality of C1 is resolved as NAND. In this way, the output of C1 is sensitized to the primary output O1. Therefore, C1 can be resolved without knowing C2 as there is no interference between them. The same argument holds for finding C2. Therefore the complexity of resolving the circuit is only 2^2 and not 2^4 as it may seem due to the presence of two camouflaged gates.

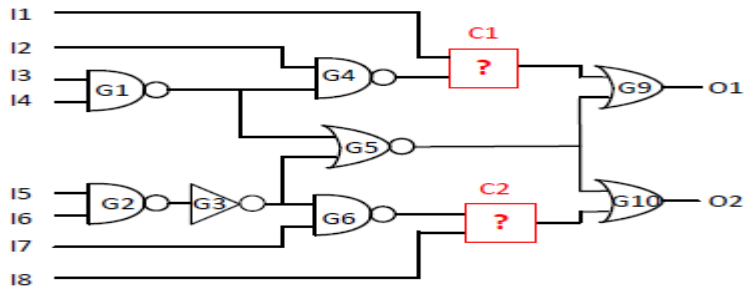


Figure 3.2: C1 and C2 are isolated camouflaged gates. An attacker can resolve the functionality C1 and C2 independent of each other [15]

2) **Partially resolvable cells:** Even though these camouflaged cells have dependency on others, there exist some input patterns that can bypass the dependency and isolate the cells to partially, but not fully, resolve them. For example, suppose each camouflaged cell can perform {XOR, NOR,NAND} functions, and for the only applicable input pattern {0,1}, the output is 1, the camouflaged cell may be a NAND or an XOR.

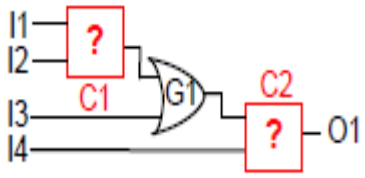


Figure 3.3: Interference between camouflaged gates, C1 and C2. An attacker targets C2 before targeting C1

Consider the circuit in Figure 3.3 as an example of interfering cells. To identify the functionality of C2, an attacker has to find an input pattern that can justify the output of C1 to a known value and simultaneously resolve the functionality of C2 with the help of this known value. Suppose the functionality of C1 and C2 can be one of {XOR, NAND, NOR}. The functionality of C2 can be resolved to be an XOR by applying '1100' at the inputs. This pattern will justify the output of C1 to '0' irrespective of whether it functions as an XOR, NAND, or NOR, justify the inputs of C2 to '00,' and sensitize the output of C2 to O1. If O1 is '0', then the functionality of C2 is resolved as XOR.

Otherwise, the functionality of C2 can be resolved to be either NOR or NAND by applying '1101' at the inputs. This pattern will justify the output of C1 to '0' irrespective of whether it functions as an XOR, NAND, or NOR, justify the inputs of C2 to '01,' and sensitize the output of C2 to O1. If O1 is 0, then the functionality of C2 is resolved as NOR. Otherwise, the functionality of C2 is resolved as NAND.

After resolving the functionality of C2, C1 becomes an isolated camouflaged gate and its functionality can be resolved using the procedure specified for isolated camouflaged gates.

3) **Non-resolvable cells:** There is no input pattern that can void the dependency between these camouflaged cells. To solve the functionality, one has to use exhaustive search.

3.3.3 Lemma 3: Don't Care Conditions in the Circuit Lower the Complexity of RE Attacks

Don't Care conditions in the circuit lower the complexity of RE attacks because two apparently interfering key gates can actually be isolated from each other by utilizing don't care conditions on the path between these two key gates. This is shown in Figure 2.10 and reproduced below:

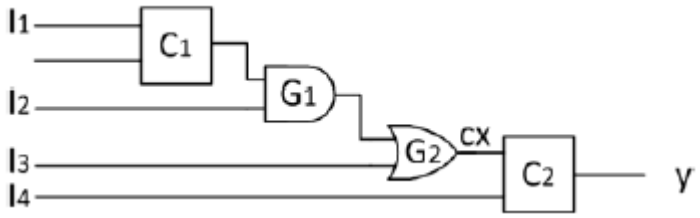


Figure 3.4: Presence of Don't Care conditions in the circuit lowers RE complexity. The attacker can resolve C2 by setting I3=1 and Controlling I4. The attacker then targets C1.

In the above example, the presence of don't cares lowers the complexity of reverse engineering effort from 2^4 to 2^3.

When setting {I2=0, I3=0} above, cx will be forced to 0; when setting {I3=1}, cx will be forced to 1. Thus, C2's inputs can be traversed without considering C1. C2 is a dominating gate with respect to C1 since it lies on the path from C1 to output. However, by taking advantage of Don't Cares, the attacker can easily invade and control the internal signal, justifying it to a known value. This intrusion can breach the connections between the configurable cells, and lower down the complexity of RE dramatically.

3.3.4 Lemma 4: Cone splitting, if possible in the circuit, lowers down the complexity of RE attacks

This is a very interesting feature in camouflaged circuits which should be kept in mind by the security-aware designer in choosing candidate gates to obfuscate. Cone splitting can dramatically bring down the complexity of reverse engineering attacks, resulting in wasted effort when designing a secure chip. We take this property into consideration when devising the algorithm to select appropriate gates to obfuscate. Cone splitting is explained by the following example:

The following figure shows c17, one of the benchmarks in the ISCAS-85 family that is composed of 6 NAND gates. Let us try to determine suitable gates in this circuit to obfuscate in the light of cone-splitting. It might appear that G10 and G19 are good candidates for gate replacement or camouflaging: they do not have any don't care conditions on the path between them and the apparent complexity of RE in this case is 2^4.

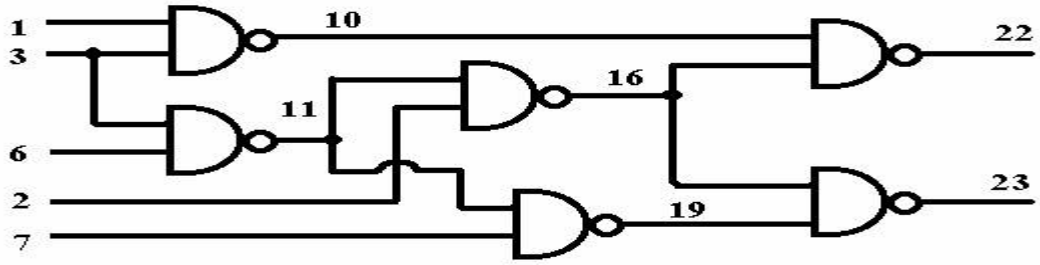


Figure 3.5: C17 Benchmark Circuit from ISCAS-85 family

Here is an important insight: G10 only affects output G22, while G19 on affects output G23. Therefore, an attacker would simply control inputs I1 and I3 to justify and sensitize the value of key gate G10 to the output G22, and simply use primary inputs I2 and I7 to sensitize G19 to the output G23. So the complexity of RE for Goal-1 in this case is not 2^4 but 2^2. A better choice for obfuscating this benchmark circuit is camouflaging G11 and G16 as they affect both outputs G22 and G23. Essentially***, these gates lie in the intersection of the fan-in cones of the primary outputs, and therefore there is no way to resolve one without resolving the other.*** Therefore the complexity of RE in this case would be 2^4(all possible number of

inputs of the two gates that need to traversed to resolve both) much better than our first choice of candidate gates. The effect is even more pronounced for larger and more complex circuits. In addition, if these two gates are camouflaged so that the configurable cell they are replaced with can perform, say one of 3 possible logic functions, for example, {NAND, NOR, XOR} then the effort of RE for Goal-2 is $3x3x2^4$ for the two replaced cells**.** Of course, the complexity is a bit lower in this particular replacement because applying input {1,1} to this configurable cell always results in '0' at the output of this cell, for each possible logic function. We discuss overcoming this issue by introducing, in the next section, configurability into the IC using a 4x1 Multiplexer that can perform one of 16 possible functions.

As a second example of the effectiveness of cone-splitting consider the circuit in Figure 3.6.

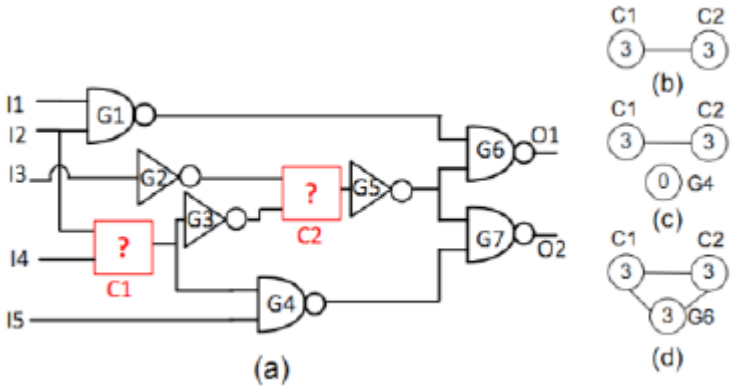


Figure 3.6: Motivation Example demonstrating (a) cone-splitting and (b) the use of interference graphs (taken from [15])

Since the complexity of non-resolvable cells is exponential, [15] proposed an algorithm to find the candidate gates that can form non-resolvable interference graph and replace the candidates with configurable cells that can perform one of three logic functions {NAND, NOR, XOR}.

In this circuit, C1 and C2 are non-resolvable camouflaged gates that interfere with each other. Hence, their nodes have a weight of 3 and are connected by an edge as shown in the interference graph in Figure 8(b). Let us try to determine a third suitable gate to camouflage in this circuit, to

demonstrate the concept of cone-splitting. If G4 is selected as the third gate to be camouflaged, then it creates an isolated node in the interference graph. This is because an attacker can use the patterns 'X1010' and 'X1011' to resolve the functionality of G4 (recall that input pattern {1,1} at the camouflaged gate results in the output '0' in every case).

However, if G6 is selected as the third gate to be camouflaged, then it creates a non-resolvable node in the interference graph since it will be directly or indirectly connected to the nodes C1 and C2. This will apparently increase the maximum clique size from 6 to 9. However, in addition to non-resolvability and interference between nodes, an important consideration here is that the attacker can still isolate the fan-in cones of the two outputs and proceed to attack them separately. Since G6 does not lie in the fan-in cone of O2, the attacker can simply manipulate the primary inputs {I3,I4,I5} that belong to the fan-in cone of O2 to target C1 and C2 first, and then resolve G6 by only considering the fan-in cone of O1. Therefore, cone-splitting can be used to bring down the complexity of RE, and reduce the maximum clique size to 6.

Therefore, instead of solving C1, C2 and G6 together, one can target the fan-in cone of O2 and solve the functionality of C1 and C2. Fan-in cone of O2 consists of all the gates and primary inputs that affect O2, noted as $C_o(O2)$. The fan-in cone can be easily found by Depth First Search algorithm using the output as a starting node. Once C1 and C2 get solved, the attacker can replace them with known gates and then target G6.

So for two camouflaged cells $Ci \in C_o(Om)$ and $Cj \in C_o(On)$, no matter what kind of interference they form, they can be resolved by splitting and targeting different cones. Similarly in key insertion based approach, if the key K1 and K2 are hidden in different fan-in cones of the primary outputs,

they can still be resolved separately even if they interfere with each other. If Ci, Cj or K1, K2 only belong to one cone and they are tangled, the cone cannot be split to resolve the camouflaged gates.

In the examples above, we assume that the attacker is more concerned with Goal-2, which is to resolve the functionality of every camouflaged cell, so that one can directly copy the design. For Goal-1, the attacker doesn't care what the exact logic of the cell is, as long as the input-output relationships are all known (so that the attacker can implement the same functionality with a different logic). Note that the attacker's goals are listed in section 3.1. For Goal-1, instead of traversing all possible logic inputs of each camouflaged cell, the attacker only needs to traverse the obfuscated inputs and read the outputs. Then use these input-output pairs to recover the truth table of the circuit. So for the circuit with n replaced cells, m obfuscated inputs (m <= 2n), the RE complexity is 2^m. Therefore, normally the complexity of achieving Goal-1 is less than that of Goal-2. Assume that the camouflaged cell can implement x different 2-input Boolean logic functions, ideally complexity of achieving Goal-2 is x^n where 'n' is the number of camouflaged gates. In Figure 3.6 (a), x=3 because {XOR, NOR, NAND} functions can be performed by each camouflaged cell. If instead, a 4x1 multiplexer is used as the replacement cell, x=16 because it can perform one of 16 equally likely and equally valid logic functions. We show this in the next section.

3.3.5 Lemma 5: The complexity of circuit for Goal-1 (determining input-output relationships) is limited by the number of inputs of the largest camouflaged cone. The largest cone is the cone that has most primary inputs.

The proof consists of two steps:

a) No matter how many cells have been replaced inside a cone, the upper bound of RE complexity for Goal-1 is O(2^{ci}), where ci is the number of primary inputs of this cone. To rebuild the truth table, attacker needs to traverse all possible input vectors of the primary inputs. Thus the worst case complexity is 2^{ci}. Note that this is the complexity for Goal-1 where the attacker is only interested in the high-level functionality or determining the input-output relationships of the circuit.

b) No matter how many camouflaged gates or cells interfere with each other, one can target the cells within the same cone and bypass the influences from the camouflaged gates in other cones. Therefore, choosing gates belonging to the intersection of fan-ins of two or more outputs can thwart an attacker from splitting the cones to extract gate functionality at a lower RE complexity. Please refer to Figure 3.6 and the relevant discussion for an example illustrating the concept of cone-splicing attack.

From above, we conclude that the attacker can target the cones individually, and the complexity of resolving cones is limited by the number of its primary inputs (for Goal-1). Suppose a circuit consists of the logic cones (with respect to the outputs) having sizes of n1, n2 … nm (suppose n1≥n2≥…≥nm) where the size of the cone is the number of primary inputs to the cone. The complexity of RE for Goal-1 is O($2^{n1}+2^{n2}+\cdots+2^{nm}$). Since n1≥n2≥…≥nm we can conclude that the complexity upper bound is O (2^{n1}). As an example, consider the circuit in Figure 3.7:

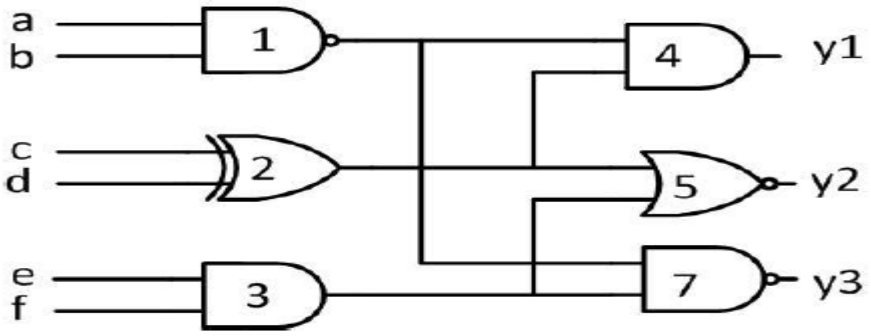


Figure 3.7: Complexity Analysis for Goal-1 (determining high level circuit functionality or truth table)

If we replace all the gates in Figure 3.7 with configurable cells, intuitively we need to traverse all 6 inputs with the complexity of 2^6. However, even though the attacker does not know the exact logic function of every camouflaged cell, he can see the wire routing and connections. In this circuit, there are three cones, C_{o1}{1,2;4}, C_{o2}{2, 3; 5} and C_{o3}{1, 3; 7}. The largest cone size (number of inputs to the cone) is 4.

Attack Strategy using Cone Splicing:

1) Try all combinations of (a,b,c,d) and get y1 --2^4 times

2) Try all combinations of (c,d,e,f) and get y2 --2^4 times

3) Try all combinations of (a,b,e,f) and get y3 --2^4 times

4) Analyze the result and figure out the functionality of each logic gate.

Therefore, using cone-splicing, the attacker can test all possible inputs patterns with a complexity less than 2^6. Moreover, in the real time testing environment, the computations can be performed

in parallel, reducing total time cost to $(3 \times 2^4)/2 = 1.5 \times 2^4$ (assuming two computational engines are running in parallel).

The above discussion analyzes the complexity for Goal-1 i.e. when the attacker wants to derive the truth table or input-output relationships of the circuit. Similarly, for Goal-2, the RE complexity is limited by the cone that has the largest number of configurable cells. We utilize this concept in our algorithm, which determines the intersection of the fan-ins of outputs with the largest number of gates. This intersection determines the complexity of reverse engineering and the candidate gates in this pool are camouflaged. Since these gates are further connected to other gates in the circuit which may or may not be in the intersection, camouflaging these gates increases the attacker's RE efforts.

Lemma 5 brings new insight to the current obfuscation approaches. The existing work of either key insertion or gate replacement aims to determine candidate gates that are non-mutable or non-resolvable and camouflage them without regard to the cone in which they lie whereas here we are concerned with the intersection of fan-in cones of the outputs. If the 'tangled' or interfering clique can be subjected to the cone-splitting attack, then the fact that the camouflaged gates are interfering is moot. The attacker, for example, may be able to target a small cone and solve out a few of the obfuscated primitives. These solved primitives could prove to the weak link in the interference graph and hence help in resolving other gates in the previously unresolvable clique. They could either expose the 'golden pattern', as mentioned in [21] or make the replaced cells partially or even fully resolvable [15] by providing don't care inputs.

The proofs above demonstrate that the presence of don't care conditions in the circuit and cone-splitting feature in the circuit can be used by the attacker to his/ her advantage to bring down the complexity of reverse engineering attack.

3.4 Recommendations for a Smarter Obfuscation Strategy

As we analyzed above, there are drawbacks in all the existing obfuscation approaches. When RE attackers have powerful abilities such as the use of **Boolean SAT solvers** or very smart RE strategy that takes advantage of cone splitting, the current obfuscation approaches fail to protect the design IP from RE attacks. In this section, some recommendations are presented based on the observations made above. These recommendations can improve the effectiveness of circuit obfuscation as a countermeasure against RE attacks, and were incorporated in our algorithm as explained in Chapters 4 and 5 that follow.

3.4.1 Pick the gates that belong to the same fan-in cone for obfuscation.

If the candidate gates only belong to one fan-in cone (with respect to the outputs), the attacker will not be able to split the cone and test the configurable cells individually because there is only one output pin whose value might be affected by all these cells inside the fan-in cone. Each of the candidate gates will have the output gate in its fan-out. Candidate gates which share the same output will thus be grouped. The complexity of this approach will be determined by the group that has the most candidate gates.

The drawback of this approach is as follows: there is no guarantee on the number of such candidate gates that can be found in a circuit. The complexity, or the enhancement of the obfuscation method, might be limited by the small number of possible candidate gates in a group.

Therefore, it is interesting to study how to locate large group of candidate gates or re-engineering the circuit to construct additional candidate gates to increase clique size as needed. In addition, when the quantity of the candidate gates can be guaranteed, the gates may be clustered in a small region on the circuit. The performance overhead, especially circuit delay, may be very high and might not be acceptable especially if these gates lie on the critical path. We propose to overcome some of these issues by utilizing the don't care conditions inherent in the circuit to trigger these camouflaged gates. Since Satisfiability Don't Cares, for example, are patterns or input combinations that never happen, the delay in this case might be negligible. We perform tests on various benchmark circuits from the ISCAS-85 family to test this hypothesis.

3.4.2 Pick the gates that are in a common subset (intersection) of several fan-in cones and do not belong to any other cones.

This is equivalent to finding the largest group 'G' such that each gate 'gi' in G will satisfy: gi $\in$ CCS(CCS is the common subset of CONE, where CONE is a group that contains one or more fan-in cones of primary outputs, the size and elements of CONE is not fixed) and gi $\notin$CONE' (CONE' is the complementary set of CONE containing the remaining gates of the fan-in cones). Note that Recommendation No. 1 mentioned above is a special case of Recommendation No. 2, when CONE contains only one fan-in cone (when i = 1). Even though the candidate group 'G' can be read in several fan-in cones, but it cannot be partitioned by these cones i.e. the group 'G' lies in the intersection of two or more output logic cones. Therefore, any output of this subset or intersection will be affected by all the elements in 'G'. So after replacing the candidate gates in 'G', these configurable cells cannot be divided and conquered. In this manner, the cone-splitting attack will be ineffective.

3.4.3 Increase the size of fan-in cones.

As mentioned in Lemma 5, the complexity upper bound of a circuit for Goal-1 depends on the number of inputs to the largest cone. Thus, increasing the cone size (number of possible candidate gates in the cone) or in the case of Goal-1 the number of primary inputs to the cone, can effectively increase the complexity upper bound. In some circuits, even though the total number of primary inputs is large, the largest cone doesn't occupy majority of them. Thus, the upper bound of complexity is limited. One way to solve such problem is to increase the size of fan-in cones without changing the circuit's functionality. This can be achieved by utilizing a 4x1 multiplexer that can be configured to perform one of many functions, including a buffer or inverter. This feature of the 4x1 multiplexer can be used to effectively increase the number of primary inputs, or change the layout of the circuit without affecting the overall functionality.

3.4.4 Use the 4x1 Universal Multiplexer for Gate Camouflaging

Bao et al. [16] introduce a novel CMOS cell that can be configured as any one of all possible 2-input logic functions using a programmable connector and a 4x1 multiplexer. 4x1 multiplexer is one of the most widely used logic gates. [16] briefly introduces the configurable feature of 4x1 multiplexer. By filling in the different truth values as shown in Table III, the 4x1 multiplexer can be configured to implement all possible 2-input, 1-output digital logic functions, including buffer and inverter functions.

Figure 3.8 shows how to implement a 2-input AND gate operation on bits (a,b) using 4x1 multiplexer. (a,b) act as the select bits of the multiplexer and the logic function can be configured into the memory. Depending on this configuration, sixteen possible functions can be performed as shown in Table III. If the configurable bits are carefully hidden in one-time programmable memory

cells (or tamper-proof memory) so that they cannot be determined under optical or electron microscopy, the type of logic will not be detected directly. The only way to know the logic would be to traverse all the possible inputs of (a, b) and read the corresponding output f. The configuration memory cells can be effectively hardcoded through the programmable camouflage connector proposed in [17].

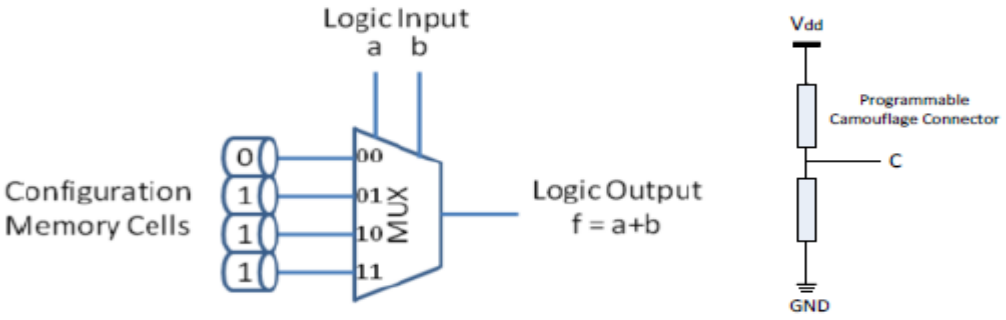


Figure 3.8: MUX-based 2x1 OR gate proposed in [16] and a programmable connector proposed in [17]

The following truth table shows that a 4x1 universal multiplexer cell can implement 16 logic functions depending on the configuration bits stored in memory cells. For example, if the configuration memory cells store {0,1,1,1} corresponding to select bits a and b, then we know that f(a,b) is an OR function on the select bits {a,b}. Similarly, if the configuration bits are {0,0,0,1} then the 2-input logic function f(a,b) performed by the 4x1 multiplexer is the AND function. In this manner, 16 possible 2-input functions can be performed as shown in the following table.

Table III: 16 Boolean Logic Functions f (a,b) that can be performed by a 4x1 multiplexer depending on the configuration bits.

00	01	10	11	Logic expression:	
0	0	0	0	*const* 0	0
0	0	0	1	AB	1
0	0	1	0	$A\bar{B}$	2
0	0	1	1	$A + 0 \cdot B = A$	3
0	1	0	0	$\bar{A}B$	4
0	1	0	1	$0 \cdot A + B = B$	5
0	1	1	0	$A \oplus B$	6
0	1	1	1	$A + B$	7
1	0	0	0	$\bar{A} \cdot \bar{B} = \overline{A + B}$	8
1	0	0	1	$A \odot B$	9
1	0	1	0	$\bar{B}$	10
1	0	1	1	$A + \bar{B}$	11
1	1	0	0	$\bar{A}$	12
1	1	0	1	$\bar{A} + B$	13
1	1	1	0	$\overline{AB}$	14
1	1	1	1	*const* 1	15

As seen from Table III, the 4x1 multiplexer cell can perform some very unusual logic functions, like constant “1” and “0”. Besides, row 10 and 12 show that a 2-input camouflaged cell can act like an inverter, with one don't care input. Similarly, row 3 and 5 shows a cell can perform as a wire or buffer with one don't care input. Inspired by this, we can insert these wire-like cells or inverter-like cells into the circuit and add one dummy input as the primary input to increase the cone size (number of primary inputs for attacker with Goal-1: see lemma 5 above) instead of directly feeding the buffered or inverted input as in the original, uncamouflaged circuit. Even though we know that these dummy inputs will not affect the overall circuit functionality (since they act as don’t care

inputs in the truth table), the attacker cannot observe which input is the dummy input. So he/she would need to consider these inputs during the testing phase, thus increasing the RE complexity.

An important advantage of using a 4X1 MUX is its contribution to RE complexity. For an individual MUX, the attacker can traverse all 4 possible inputs {00, 01, 10, 11} to resolve its functionality. So the RE complexity of single obfuscated MUX is 2^2=4. Suppose we replace N single gates, and any two of them are non-interfering with respect to each other. The complexity in this case is 4N. Apparently this is a bad strategy, because the IC can be resolved in linear time, O(N). However, if we replace the gates which are interfering, such as those belonging to the same cone, the input of one gate will depend on the output of another. To resolve these two gates, the attacker cannot test them individually. Therefore, all the inputs of these two gates have to be traversed together, which dramatically increases the RE complexity.

Having established the Attack Model for gate camouflaging, and studied the motivations, assumptions and weaknesses of the state of the art obfuscation techniques, we use the recommendations presented in this chapter to create, simulate and test a smarter obfuscation strategy that makes use of the don't care conditions inherently present in almost all circuits of non-trivial size.

Please note that the lemmas and recommendations presented in this chapter first appeared in [13]. They have been reproduced here with the explicit permission of the authors. This research work applies these recommendations to devise an algorithm for smarter hardware IP obfuscation.

"Not every puzzle is intended to be solved. Some are in place to test your limits..."

--Vera Nazarian, The Perpetual Calendar of Inspiration

CHAPTER 4: HARDWARE OBFUSCATION USING DON'T CARES

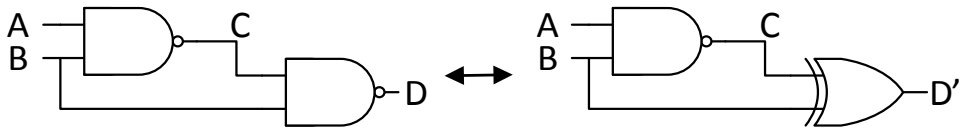
In this chapter, we explain how don't care conditions inherent in a combinational circuit can be utilized for layout obfuscation. These conditions can be used to select appropriate gates to obfuscate, and to create small modifications at the sub-circuit level while meeting the overall circuit specification. At the same time, the area and delay overhead caused by making these changes at the sub-circuit level should be acceptable while ensuring that the RE complexity is exponentially high. There is potential to add to this approach by obfuscating other gates at the functional level (rather than layout level), and add another layer of complexity by obfuscating pre-existing multiplexers (if any) in the circuit. We present functional obfuscation at the end of Chapter 5. This method of functional obfuscation works for incompletely specified circuits.

We start with an introduction to don't care conditions, and explain Satisfiability Don't Cares with examples. Observability Don't Cares are presented in Chapter 5.

4.1 Introduction to Satisfiability Don't Care (SDC) based Obfuscation

We propose a new methodology for hardware IP obfuscation that relies on the concept of a Satisfiability Don't Care (SDC), a condition that occurs regularly in virtually every circuit of non-trivial size. This new method for obfuscation allows us to obfuscate a circuit's function or behavior instead of (or in addition to) modifying a circuit's structure or gate-level description. A by-product of the method is that it can be implemented in the later stages of the VLSI design cycle, thus

bypassing the need for expensive redesign. Figure 4.1 shows a trivial example of a Satisfiability Don't Care (SDC).



(a)

A	*B*	*C*	D	D'
0	0	1	1	1
0	1	1	0	0
1	0	1	1	1
1	1	0	1	1

(b)

Figure 4.1: Example SDC modification (a) Gate Replacement (b) Truth table for both circuits

Both circuits in Figure 4.1 are identical, where in the second circuit, the second NAND gate is replaced with an XOR gate. This modification will not make a functional change at the primary outputs of the circuit, but it will make internal functionality different if tested. As a further example involving only one NAND gate, we notice from the truth table for this logic function C=NAND{A,B} that when A=0, the input B acts as a don't care while the output C is always 1. We conclude that for a NAND gate the condition {A=0, C=0} can never happen. This particular combination is called a Satisfiability Don't Care, a pattern that will never occur in a circuit. Similarly for an AND gate C=AND{A,B}, the combination {A=0, C =1} never occurs, and no matter what the input combination to this AND gate, (**A'.**C) is always zero.

We use this concept to develop a method for obfuscating circuit layout or functionality using SDCs. Specifically, we first determine candidate gates that belong to an un-splittable intersection cone. The

gates are valid candidates if they belong to the fan-in cone of two or more outputs (Section 3.3, recommendation 2). This is done to avoid the cone-splicing attack detailed in section 3.2 (Lemma 4). Let us call these locations 'valid function locations'. Next, we deliberately introduce invalid functions (possibly from other cones) and then insert a 2x1 multiplexer in our design, **controlled by the SDC condition as the select bit**, configuring it to always output the valid function only. Because the SDC gate itself is camouflaged by a 4x1 multiplexer, there is no way for the attacker to determine the correct select bit without traversing all the possible 16 logic functions for the multiplexer. The attacker is also unable to sensitize the 'valid function' output to the primary output because it is chosen from an un-splittable intersection cone, and there are many valid functions multiplexed with invalid functions in the same cone. In this manner the layout of the circuit has been modified. In Chapter 5, we introduce the concept of functional obfuscation at the sub-circuit level with examples.

In attempting to create a robust, reliable obfuscation method, several challenges need to be addressed. First, we need to determine where we can make modifications to our IC such that the overall functionality is unchanged but there are sub-circuit level layout modifications that make it more difficult to understand the true functionality. We will try to bypass the cone-splicing attack in determining appropriate modifiable locations. The second challenge is to find a computationally inexpensive way to determine SDC conditions that trigger the 2x1 multiplexer without utilizing the deterministic exhaustive search usually associated with locating SDCs. Our final challenge is to make it difficult for adversaries to distinguish between true and introduced sub-circuit modifications.

Boolean Formulation to determine Satisfiability Don't Cares

Satisfiability Don't Cares are a Boolean concept used in circuit design optimization. Considering all the primary input (PI) signals, and the internal signals from each logic gate in a circuit, SDC conditions describe the signal combinations that cannot occur. For example, consider again the 2-input NAND gate from Figure 4.1, C=NAND(A,B), we cannot have {A=1,B=1,C=1}, {A=0,C=0}, or {B=0,C=0}. In general, for a signal *y* generated from logic gate $G(x_1,x_2, \ldots, x_k)$, the SDC at this gate can be computed by the equation

$$SDC = G(x_1,x_2,\ldots,x_k) \oplus y \quad \ldots\ldots (1)$$

which means that when the expected output *y* is at odds with the actual output of the gate function G an SDC condition has occurred. In the example of the above 2-input NAND gate, the SDC conditions can be obtained from Equation (1) as follows:

$$\overline{AB} \oplus C = \overline{AB}\bar{C} + ABC \quad \ldots\ldots (2)$$

When some of these signals fan-in to the same gate later in the circuit, the SDC conditions can be used to obfuscate the circuit by applying them as select bit for a 2x1 multiplexer that has a valid and a deliberately introduced invalid function as input.

The main idea presented in this work is how SDCs can be utilized to obfuscate the layout or functional description of the circuit that is both robust against attack (computationally expensive to reverse engineer) and has acceptable delay and area overheads. By locating gates that have SDCs leading into them, which we refer to as *valid function locations*, we can modify the sub-circuit by deliberately introducing these and other deliberately invalid functions as inputs to a 2x1 multiplexer controlled by an SDC from within the larger circuit. The final step is camouflaging this SDC gate to

make it harder for an attacker to determine the SDC condition. This is repeated for all the valid function locations in the un-splittable cone of maximum size so that the output of each obfuscated gate cannot be individually determined.

4.2 The SDC-based Obfuscation Problem

Problem Statement

We now formally present the SDC based obfuscation problem:

Given an IP in the form of a gate level netlist, find a set of valid function locations and a corresponding set of deliberately introduced invalid functions, determine the SDC conditions in the fan-in of each of the valid locations (or elsewhere in the circuit; does not matter which SDC condition is chosen), and define an obfuscation scheme or metric that maximizes the RE complexity at minimum performance overhead.

4.2.1 Layout Obfuscation Method 1:

Before presenting our solution to the problem, we list the necessary assumptions and define the terminologies.

A1. The given netlist should be sufficiently large to accommodate or obfuscate a reasonable number of valid functions.

A2. The given netlist is optimized but may have internal gates producing constant outputs. Circuits normally can be simplified if we replace constant-valued variables with their value (0 or 1).

A3. All primary inputs (PI) to the circuits are independent. If one PI depends on other PIs (e.g. the complementary variables in dual-rail logic), we can consider this PI as an internal signal.

We consider again the C17 benchmark from the ISCAS-85 family introduced in Figure 3.5 and reproduced below:

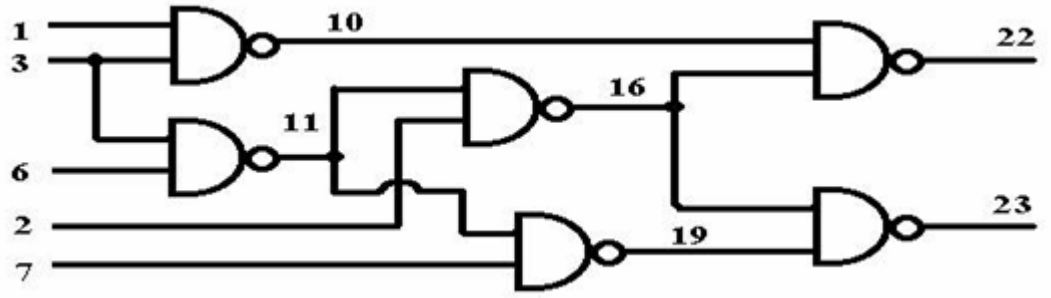


Figure 4.2 (a) C17 Benchmark from the ISCAS-85 family

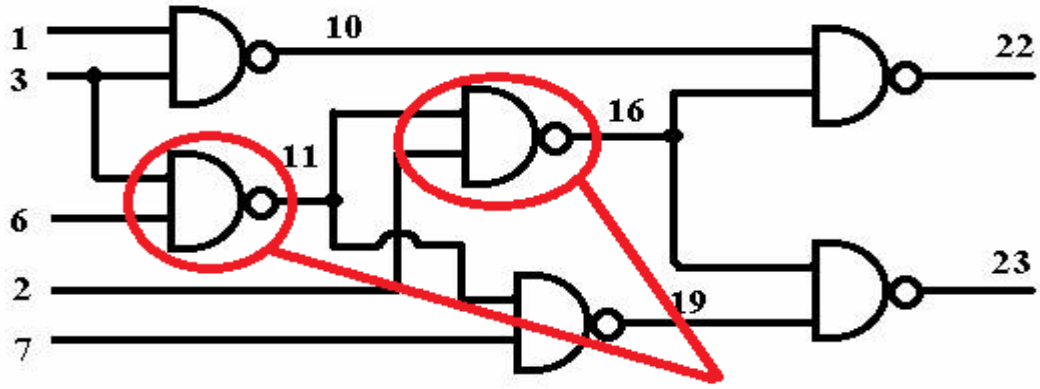


Figure 4.2 (b) G11 and G16 lie in the intersection of the two output cones and therefore constitute possible 'valid function locations'

In this circuit, G16 is a valid function location because it lies in the intersection cone of both outputs O22 and O23. We illustrate how functional obfuscation is achieved with this valid function and an SDC condition identified within the circuit. The four ingredients: a set of valid and corresponding deliberately invalid function, the SDC gate and a 2x1 multiplexer, together with the final gate camouflaging step completely specify our don't care-based obfuscation methodology.

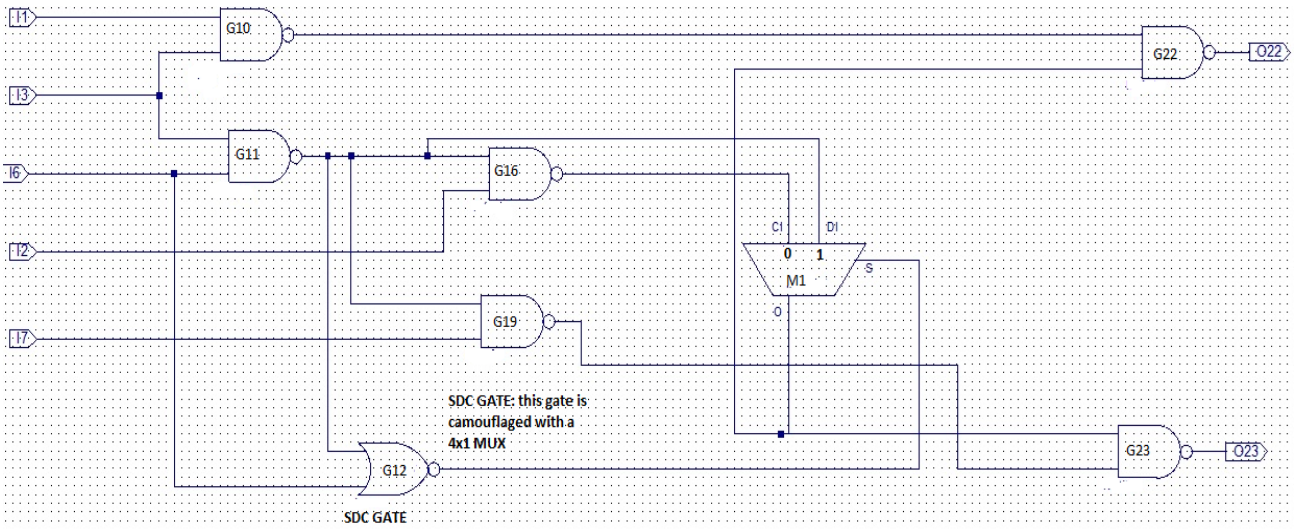


Figure 4.2 (c) Illustration of SDC-based obfuscation for C17 benchmark

Suppose that the invalid function corresponding to the valid function G16 is the output of G11 (preferably, the invalid function should be selected from some cone other than the one to which the valid function belongs to increase the likelihood of increase in RE complexity). In this case we pick G11 simply because the circuit is so small and G16 belongs to both output cones. Once we have selected the valid and invalid functions, the next step is to select an SDC condition from the fan-in cone of the valid function or elsewhere in the larger circuit. In this case, the SDC location is again G11 for illustrative purposes. Since G11 is a NAND gate (C=NAND (A,B)), the condition {A=0, C=0} never occurs for this gate, therefore if A and C are NOR'ed as in gate G12, the output of the NOR gate G12 is always zero. This constant-zero output is used as select bit for a 2x1 multiplexer that has the valid function and the invalid function as its inputs, with the following condition:

$$C = S\ ?\ A:B \quad \text{----------(2)}$$

i.e. for the 2x1 multiplexer shown with C as the output, S as the select bit and A(in this case, deliberately introduced invalid function) and B(valid function) as the respective inputs, the output C equals B when S=0, and C=A when S=1. Of course, in this situation, S is always '0' so the valid

function G16 is always outputted by the multiplexer. The final step is camouflaging the NOR gate (henceforth called the SDC gate) so that the SDC condition that triggers the 2x1 Multiplexer may not be observed. So the NOR gate is replaced by a 4x1 multiplexer that performs the same function (the multiplexer is configured as Row 14 in Table III). In the figure, we do not show the gate camouflaging step because it would unnecessarily complicate the diagram and take attention away from the essential details of the procedure.

To recapitulate, the following figure shows the main ingredients of our layout obfuscation methodology. The SDC Condition is the same as in Figure 4.2 (b) {A=0, C=0}, therefore, the output of the camouflaged NOR gate is always '0', always selecting the valid function.

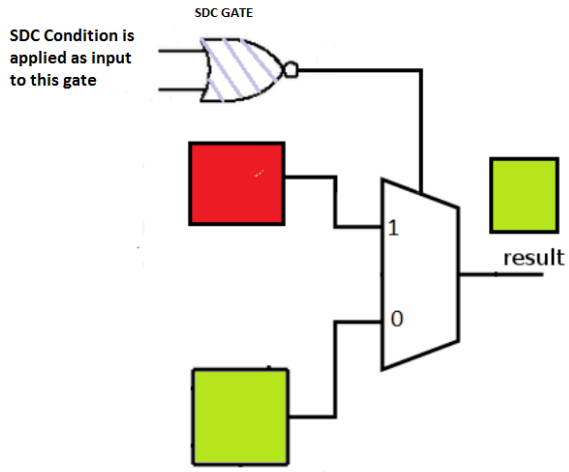


Figure 4.3: Essential Ingredients of SDC-based Layout Obfuscation (a) the green block represents a valid function (b) the red block is the deliberately introduced invalid function (c) the 2x1 MUX is triggered by a camouflaged gate output determined by an SDC condition in the circuit. The purple stripes in the figure represent camouflaging

In figure 4.3 only the obfuscated part and related circuitry is shown while the other details of the circuit are abstracted out. In this case, the camouflaged gate's output is always '0'.

Note that we made the assumption earlier that the circuit is optimized but may include gates with constant outputs.

4.2.2 Layout Obfuscation Method 2

If, on the other hand, we want the camouflaged gate to not have a constant-zero or constant-1 output we can make use of logic identities and the fact that a 4x1 Multiplexer can be configured as a buffer to have the obfuscated circuitry look as shown in Figure 4.4 (the unnecessary parts of the circuit have been abstracted out). Please note that in this circuit, the multiplexers M1, M2 and M3 are used to implement a 4x1 Multiplexer that inputs the valid function and a "distorted version" into the 2x1 Multiplexer M4. The truth table values fed by a programmable connector determine the functionality of the 4x1 Multiplexer. The 4X1 Multiplexer is represented by the green block in the figure.

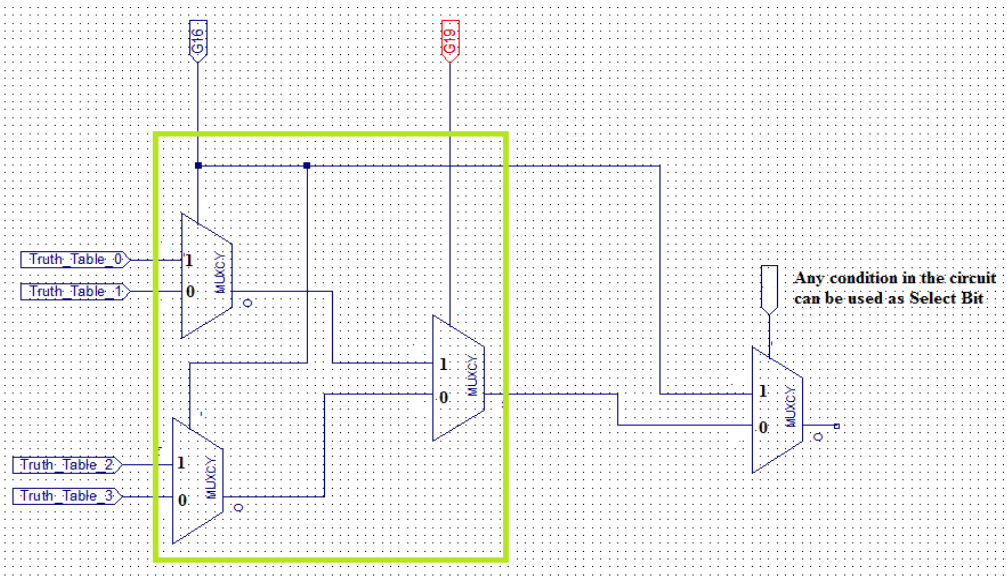


Figure 4.4: A second method of Layout Obfuscation of G16 in C17 benchmark. The output of G16 is fed directly into the 4x1 MUX as one input; the other input is a camouflaged form of the valid function. The M4 select bit is triggered by any signal in the circuit, the M4 output will always be equal to G16, regardless of the select bit.

Suppose the truth table is configured with {0,0,1,1}. This means that the output of M3 will be the G16, regardless of what the input B is (in this case, we choose the select bit B to the 4x1 MUX to be G19, this input is a don't care because the 4x1 multiplexer is configured as a buffer). We feed these signals into M4. This means that the inputs to M4 are both actually the same, but this is obfuscated from the (malicious) user. One interesting thing about this method of obfuscation is that any signal in the circuit can be used to trigger M4 to further confuse the attacker. Whether the select bit is a zero or a 1, the output of M4 will always be our desired valid function G16.

Although this method is presented as a possible layout obfuscation method, we simulated and tested method 1, depicted in Figure 4.2 and 4.3, utilizing Satisfiability Don't Care conditions in the circuit for obfuscation purposes. For Method 2, we utilized Observability Don't Cares as select bit for the 2x1 Multiplexer. The results are presented in Chapter 5.

The discussion that follows next and in the coming chapters involves our implementation of method 1.

A necessary condition for valid function locations: *A valid function location lies in the intersection of two or more output logic cones.*

When a gate, say G19 in Figure 4.2(a), does not lie in the intersection of the two output cones but only in one of these cones, then it will be easy for an attacker to only consider the fan-in cone of G23 when reverse engineering the circuit and completely ignore the contribution from the fan-in cone of G22. Conversely, if we choose G16 and G11 as valid function locations for obfuscation purposes, then the attacker has to necessarily consider the contributions from both the cones. Even for an attacker who is interested in the high-level functional description of the circuit, the complexity of RE in this case would be 2^3 instead of 2^2 when intersection of cones is not taken into

consideration and only G19 is obfuscated. If, on the other hand, there is just one output logic cone, then all camouflaged gates will belong to the same output logic cone (see Lemma 1 in section 3.3.1).

Based on the above observations, we propose the following heuristics to find 'valid function locations' and obfuscate the circuit so that the RE complexity is 2^{4m} or 16^m where 'm' is the number of 'valid function locations' in the largest intersection of two or more output logic cones. The largest intersection is chosen for obfuscation to maximize RE complexity.

4.3 Pseudocode for SDC-based Obfuscation

Input: ***Original Netlist 'G', Number of Gates to be Obfuscated (gateNum) or Maximum Acceptable Delay Overhead or Acceptable RE complexity***

Output: ***Obfuscated Netlist G"***

Layout Obfuscation using SDCs:

1. *for each* $PO\ Z_i$, *calculate corresponding* $FFC_i = \{G_j | \exists path, G_j \rightarrow Z_i\}$;*Here, PO is the Primary Output.*

2. *for each gate* G_i *in the circuit, calculate corresponding output logic coneSet that contains all primary output gates:*

   $coneSet_i = \{Z_i \mid G_i \in FFC_i\};$

3. *Determine the gate(s) 'S_i' which have the same coneSet, say '$coneSet_x$'.*

4. *Pick the largest set of gates Si as 'valid function locations.'*

5. Find SDC for a gate in the fan-in of a valid function or elsewhere in the circuit.

5a. Use this SDC as select bit for a 2x1 MUX.

5b. The MUX Inputs are: Input 1 is a valid function determined in Step 4. Input 2 will be from any other output cone FFC_j such that $FFC_j \notin coneSet$

5c. Use as many SDCs as desired (=GateNum or number of valid functions or until delay constraint is met or Acceptable RE complexity is achieved for the largest set S_i determined in Step 4).

5d.Camouflage the SDC gates

6. Calculate the delay overhead for the obfuscated netlist.

7. If GateNum> No. of valid functions, repeat steps 3, 5, 6 for an intersection cone other than the largest.

8. If Calculated Delay > Max Delay, delete obfuscated gate(s) until delay overhead is met

8. Return the obfuscated Netlist

First, we determine the fan-in cones of each output in the circuit. Next, we determine the primary output gate for each gate in the netlist and thereby determine valid function locations, which are

gates that belong to two or more output cones (intersections). The largest of such gate sets is chosen to maximize RE complexity. Example of such valid gate locations for C17 benchmark will be G11 and G16 which belong to both output cones O22 and O23. This valid function and another deliberately introduced ('invalid') function chosen from some other output cone are fed into a 2x1 MUX as inputs.

The next step is to find an SDC trigger condition, which may be in the fan-in cone of the valid function or elsewhere in the circuit. This SDC gate (the NOR gate in Figure 4.2 (c), for example) is camouflaged with a 4x1 multiplexer. Of course, the SDC condition is triggered so that it always picks the valid function but gate camouflaging hides this from the attacker. The choice of the other input (the 'invalid' function) from some other output cone not belonging to the largest intersection would further complicate matters for the attacker. With many valid function locations in the largest intersection cone as determined in step 4 camouflaged, the complexity of RE goes up exponentially as there is no way to find a valid function from this intersection without finding all the other camouflaged valid functions.

4.3.1 Correctness of the heuristics

The heuristics will find all the possible intersections of the output cones, and determine the largest intersection from this group. These would form our 'valid function locations'.

This claim states that our heuristics will not report any false intersections, but will find all possible intersections and report the largest one out of these. This ensures that when we do the gate replacement for all the valid functions in this largest intersection cone, the overall function of the original circuit will not be altered but RE complexity will be high. We will omit the proof of this claim due to space limitation.

4.3.2 Complexity of the heuristics

This is determined by the size of the largest intersection of the output cones. All the gates in this intersection are valid candidates for obfuscation using SDC conditions. An attacker would have to resolve all of these gates by determining their SDC triggers and justify and sensitize their values to the output for Goal-2. However, if the output gates of the intersection are also obfuscated using SDC triggers in addition to the other valid candidate gates of this intersection, then the complexity increases even further and there is no way for the attacker to observe and distinguish the outputs of the valid candidates from each other. The heuristics' run time is in seconds or minutes for all benchmarks depending on the number of valid locations in the largest intersection cone. Of course, all the valid candidates could be camouflaged in this manner, but we stop when an acceptable RE complexity and area/ delay overhead is reached. Note that, the valid functions themselves are not explicitly camouflaged, but the SDC conditions controlling the 2x1 multiplexers that these functions feed into, are obfuscated, resulting in their indirect obfuscation. Additionally, if pre-existing 2x1 Multiplexers in the original circuit description are also camouflaged in a similar manner, it would further complicate matters for the attacker.

4.4 Gate Camouflaging Scheme

For each SDC condition found for each of the valid functions, we propose two replacement methods for camouflaging:

R1. Replace the gate at the SDC condition by another library gate where the two gates have different outputs only on the SDC condition at that location (as shown in Figure 4.1).

R2. Replace the gate at the SDC location by a multiplexer.

Figure 4.1 shows one example of **R1**, where a 2-input NAND gate and a 2-input XOR gate become interchangeable when the input combination {0,0} is the SDC condition. In this case, the SDC gate is not added to the netlist, but we search for pre-existing SDC conditions within the circuit.

The SDC gate is then camouflaged.

In **R2,** we explicitly insert an SDC gate in the netlist and use it as a trigger for the 2x1 MUX that has the valid and invalid functions as inputs. An *m*-input SDC gate can be replaced with a 2^m x1 multiplexer (MUX). The selection lines of the MUX are tied to the original inputs of the gate and the data inputs are tied to either VDD or GND, to match the patterns to implement the needed gate.

Option **R1** will require that new masks be created for each identified SDC condition, which is an expensive process. This leads us to prefer option **R2** which gives us the flexibility for post-silicon configuration. With this option we can utilize fuses, or other engineering changes such as a programmable connector presented in [17], to implement functional obfuscation at the post-silicon phase. However this comes at the cost of comparatively high design overhead due to the large size and delay of the MUX. Figure 4.5 illustrates how the fuses will be used to implement the fingerprint.

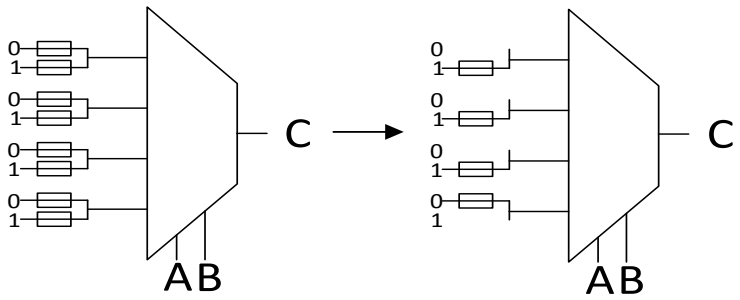


Figure 4.5: Multiplexer replacement technique. Left: An unconfigured MUX; Right: A MUX configured to run as a 2-input NAND gate.

Note that in our implementation, we limited ourselves to only camouflaging 2-input gates using replacement method R2.

4.5 Security Analysis of Obfuscation using Don't Care Conditions

We first briefly discuss detection of valid function locations because this is directly related to most attacks. When an adversary can detect these locations, he may have an easier time to remove or change them than with no knowledge about them.

4.5.1 Valid Function Detection

When we are allowed to open up the chip and view its layout, we can recover the valid functionality by identifying the gate type at each valid function location (for **R1**) or checking the configuration at each MUX (for **R2**).

As we will show in the next section, there are abundant SDC locations in real-life circuits. Therefore we can choose any SDC condition, in the fan-in of the valid function or elsewhere in the circuit. Then when we inject the SDC conditions as triggers to the valid function locations, we can tell the gate type from the output values, unless the primary outputs of the intersection themselves and other valid function locations (gates in the intersection) are obfuscated in a similar manner.

In addition to these gates, if pre-existing 2x1 Multiplexers in the circuit are camouflaged in a similar manner, it may be difficult to tell them apart from the 'valid function locations' in the intersection. Care should be exercised so that not only the performance overhead is acceptable, but also, that it is not possible to resolve these gates using justification and sensitization.

Now we consider the following attacking scenario based on the adversary's capabilities.

Simple Removal Attack

The most obvious attack against obfuscation is to simply remove the obfuscated part and simulate the (partially correct) circuit to compare with the original. This requires that an adversary know every location on an IC that our obfuscation algorithm has modified, and more importantly, a way to remove these without affecting the functionality of the original IC. In both **R1** and **R2**, because the SDC locations are required to provide the correct functionality of the circuit/IP, simply removing them will destroy the design and make the IP useless. Additionally, if pre-existing 2x1 multiplexers are also camouflaged, then it would be impossible for the adversary to have a correctly functioning IC after this removal attack.

Simple Modification Attack

An adversary may also attempt to modify, instead of removing, an obfuscated SDC location to attempt to distribute additional copies of an IC that were not approved by the original developer. These copies will behave the same way as the original IC but will contain camouflaged locations that the original developer did not produce. To achieve this, an adversary can attempt to modify intersections other than the largest that our algorithm is based on.

For **R1**, this is the same as modifying the gate replacement based watermarks, which is known to be hard. For **R2**, the attacker can find the obfuscated locations by looking for the Multiplexers. He/she can try to change the configuration of these MUXs or add additional multiplexers, but without fully reverse engineering the design, it will be hard to maintain the correct functionality.

Finally, we mention that our results show that we can easily identify many possible SDC locations. Also, instead of obfuscating just the largest intersection, it is possible to obfuscate smaller cones so that the 'hidden' gates are not concentrated in just one part of the layout or the circuit. The only limiting factor in our approach is the performance overhead caused by the introduction of 4x1

multiplexers. So we do have room to choose valid functions in the layout that lie in different cones. Since we wanted to determine the complexity of RE and the resistance our approach offered to potential attackers, we considered only the largest intersection cone as this would determine the upper bound on RE complexity.

4.6 Simulation Results and Discussion

To validate our proposed SDC-based obfuscation approach, we first see how many 'valid function locations' are identifiable in ISCAS-5 Benchmarks using our obfuscation algorithm, and second, we want to know how much design overhead we would have from embedding the camouflaged SDC gates. Most importantly, we want to determine the RE complexity introduced by this obfuscation scheme.

To accomplish this, a program was written in C++ to find the largest intersection of output logic cones and determine the valid function locations (gates in the largest intersection) and then SDC conditions were determined for each of these valid functions and used as triggers that control the selection of these valid function locations. The SDC gates were then camouflaged by replacing them with 4x1 multiplexers, following the methodology presented in section 4.2. As mentioned earlier, we limited ourselves to only camouflaging 2-input gates. It is possible to camouflage 3-input or 4-input gates with configurable multiplexers but we did not attempt to do this as part of our implementation. In some cases, we used layout obfuscation to directly obfuscate a logic gate instead of using an SDC condition to obfuscate it. This was run on a number of circuit benchmarks from the ISCAS-85 benchmark libraries, with gate characteristics given under Original Circuit Information section of Table IV. The library used for delay and area measurements is the Oklahoma State University standard cell library based on the TSMC 0.35μm technology [22]. Measurements for area and delay were collected using the program ABC [23]. The results of applying Satisfiability

Don't Care-based Methodology on these benchmark circuits is shown in Table V. Additionally, GateVision PRO, a netlist visualization tool for converting netlists into gate-level circuit descriptions was used to verify cone extraction function of our algorithm. The main features of GateVision PRO are discussed in Chapter 6.

RE Complexity

The maximum RE complexity we were able to achieve for the replacement method R2 are given in Table V along with the corresponding design overhead. The number of valid function locations considered for R2 is also listed. It is worthwhile to note here that since we only considered the worst-case complexity of reverse engineering that SDC-based obfuscation offered, we only considered valid function locations in the largest intersection cone of the outputs. As our results indicate, there is plenty of margin to obfuscate additional gates in other intersections at a higher design overhead. The only word of caution here is that in case obfuscation involves gate elements on the critical path, the delay overhead might increase dramatically. This is seen in row 2 and row 3 of Table V, where we see that obfuscation of just 4 more gates results in a noticeable increase in the area and delay overhead of the obfuscated circuit. Since these 4 additional gates belong to a different intersection cone, the complexity of Reverse Engineering is O ($16^5 + 16^4$) which is approximately equal to 16^5(where we assume that these 'valid' gates are replaced with 4x1 Multiplexers). That is why we only consider the complexity associated with the largest intersection cone for these benchmarks.

Table IV: ISCAS-85 Benchmark Circuits

Original Circuit							
Benchmark	Gate Count	I/O	Nodes	Edges	Area	Delay	Levels
C17	6	5/2	6	12	14.4	0.56	3
C432	160	36/7	183	366	456	3.94	23
C432-2	160	36/7	183	366	456	3.94	23
C499	202	41/32	454	986	1164.8	3.10	19
C880	383	60/26	305	615	797.6	3.19	20
C1355	546	41/32	454	986	1164.8	3.10	19
C1908	880	33/25	398	818	1068	4.73	29
C2670	1193	233/140	648	1300	1672.8	2.74	19
C3540	1669	50/22	855	1967	2357.6	4.91	31
C5315	2307	178/123	1317	2782	3549.6	5.82	35
C6288	2416	32/32	2303	5247	6210.4	15.18	89
C7552	3512	207/108	1626	3374	4653.6	4.76	28

Table V: Results of Satisfiability Don't Care-based Obfuscation methodology applied on ISCAS-85 benchmarks

Obfuscated Circuit								
Benchmark	No. of Valid Function Locations Considered	Maximum RE Complexity	Percentage of Gates Obfuscated	Area Increase	Delay Increase	Levels	Area Overhead	Delay Overhead
C17	4	65536	67%	16	0.65	4	11.11%	16.07%
C432	5	1048576	3.13%	484	4.21	24	6.14%	6.85%
C432-2	9	1048576	5.63%	534.40	4.75	27	17.19%	20.56%
C499	21	1.15292E+18	10.40%	1205.60	3.44	21	3.50%	10.97%
C880	10	65536	3%	831.20	3.19	20	4.21%	0%
C1355	10	1.09951E+12	1.80%	1185.60	3.28	21	1.79%	5.81%
C1908	10	1.09951E+12	1.14%	1125.60	4.84	31	5.39%	2.33%
C2670	13	268435456	1.09%	1804	3.30	23	7.84%	20%
C3540	9	68719476736	0.54%	2528	5.23	32	7.23%	6.52%
C5315	12	2.81475E+14	0.52%	3659.20	5.82	35	3.09%	0%
C6288	9	68719476736	0.37%	6252	16.27	96	1%	7.18%
C7552	18	4.72237E+21	0.50%	4786.40	4.79	28	2.85%	0.63%

Table IV also shows that the size of the original circuit which justifies the resulting design overhead and RE complexity introduced by our method. As expected, for smaller circuits such as c432, c880, etc. the algorithm identifies relatively few valid function locations for the R2 replacement method. So even though a greater percentage of gates is obfuscated, the resulting RE complexity is rather low for these small circuits. We believe this is acceptable because a circuit of this size could be reverse engineered relatively easily, and any security features could be removed. Despite this, the **R2** replacement method, especially for larger circuits, can produce sufficiently high RE complexity even if just the largest intersection cone is camouflaged. Moreover, most practical circuits usually contain a large number of gates.

The design overhead depends entirely on the size of the original circuit and the percentage of gates obfuscated (especially important for area overhead), and whether these gates lie on the critical path (important for delay overhead). The delay overhead also depends on what functions derive their input from the camouflaged valid function. Obviously, if the camouflaged function itself lies on the critical path, then connecting it to other functions as their inputs would affect the overall delay. Since 4x1 multiplexers are typically larger and slower than the two-input logic gates they replace, the fan-out of these camouflaged gates might affect the overall delay. If the obfuscated gates lie on the critical path, such as for C432 with 9 gates obfuscated, or in the case of benchmarks C3540 and C6288, the increase in overhead is large. If on the other hand, the obfuscated gates do not lie on the critical path such as for benchmarks C880, C5315 (zero delay overhead for these two circuits, and no change in the total number of levels in the obfuscated circuit, see column 7 in Table V) and others with very low delay overheads, the RE complexity is relatively high even for low overheads.. This is because the maximum RE complexity is dependent on the number of gates in the largest intersection cone, it does not depend on the percentage of total gates obfuscated. C880 represents an

interesting case where the maximum RE complexity is only 16^4 because the maximum intersecting cone has 4 gates only. However, six more gates from other cones (3 gates from each cone) are also obfuscated, so the total number of gates obfuscated is 10. But the maximum RE complexity is still approximately16^4 (16^4 +16^3 +16^3) for the three cones considered. This idea can be extended to larger, more complicated IC's, and works to the advantage of the security-aware circuit designer. It is possible to obfuscate more gates from other intersection cones (while taking care that it is not possible to justify and sensitize their values to the outputs) to offer greater resistance to reverse engineering effort. For example, **R2** can find locations for more than 21 valid functions in the largest intersection cone for C7552. Thus, even if these candidate gates in the largest intersection are obfuscated, the resulting Maximum RE complexity is 16^{23} which is phenomenal. In Table V, we only report obfuscation of 18 of the candidate gates from the largest intersection for benchmark C7552, which is still a huge number. So, for C7552, not all valid function locations were camouflaged as camouflaging just 18 of those functions gave sufficient RE complexity.

The only concern that can be raised with this approach is that it may to concentrate all the obfuscation together in one part of the circuit, while the rest of the circuit characteristics are known. To overcome this problem, we note that it is indeed possible to target other intersection cones elsewhere in the circuit to make the exact layout and function harder to comprehend (while making sure that they do not lie on the critical path or that the design overhead is acceptable). However, the maximum RE complexity will still be determined by the largest intersection cone and therefore we are concerned with this parameter in this analysis.

Finally, we note that the runtime complexity goes up exponentially with the size of the circuit. This explains the fact that we have runtimes in the range of seconds for smaller circuits and several minutes for larger circuits. It would also be useful to determine quantitatively how different the

output of an obfuscated circuit is compared to the original, especially for functional obfuscation, as discussed in Chapter 5.

4.6.1 Design Overhead

Overhead is a major concern with IP security techniques. There will always be a trade-off between circuit performance and security because security features need to be built on top of the original optimized ICs.

Table V shows that the average area overhead for **R2** is only about 5.945%, which is acceptable. The average delay overhead for the benchmarks is 8.077% which is also feasible. We highlight again that it is indeed possible to obfuscate other possible candidates from other intersections (at higher design overheads) but we focused on obfuscating the largest intersection cone to determine the maximum RE complexity. It is also worth repeating that the delay overhead largely depends on whether the obfuscated gates lie on the critical path and the fan-out of the camouflaged gates while the area overhead depends on the size of the original circuit and the percentage of gates obfuscated.

For small circuits, the area overhead for **R2** is expected to be significant. This is because we are substituting standard logic gates, such as AND, NAND, etc., with multiplexers which are significantly larger and slower. As stated above, the overhead will be mitigated by the size of the circuit. In Figure 4.6, we can see a general downward trend for area overhead for circuits as the size of the circuits gets larger. The delay overhead is largely determined by whether the valid function locations are on the critical path (this critical path might change as we utilize the SDC conditions to obfuscate the circuit). Note that, following the approach in [15] and [21] we could have utilized a performance-driven method that aims to achieve a trade-off between percentage of gates obfuscated and the resulting overhead. We see that our algorithm can help us identify all possible valid

functions and so this method is implementable with our algorithm, but we chose to focus on the maximum RE complexity as that appears to be a more useful metric in secure design.

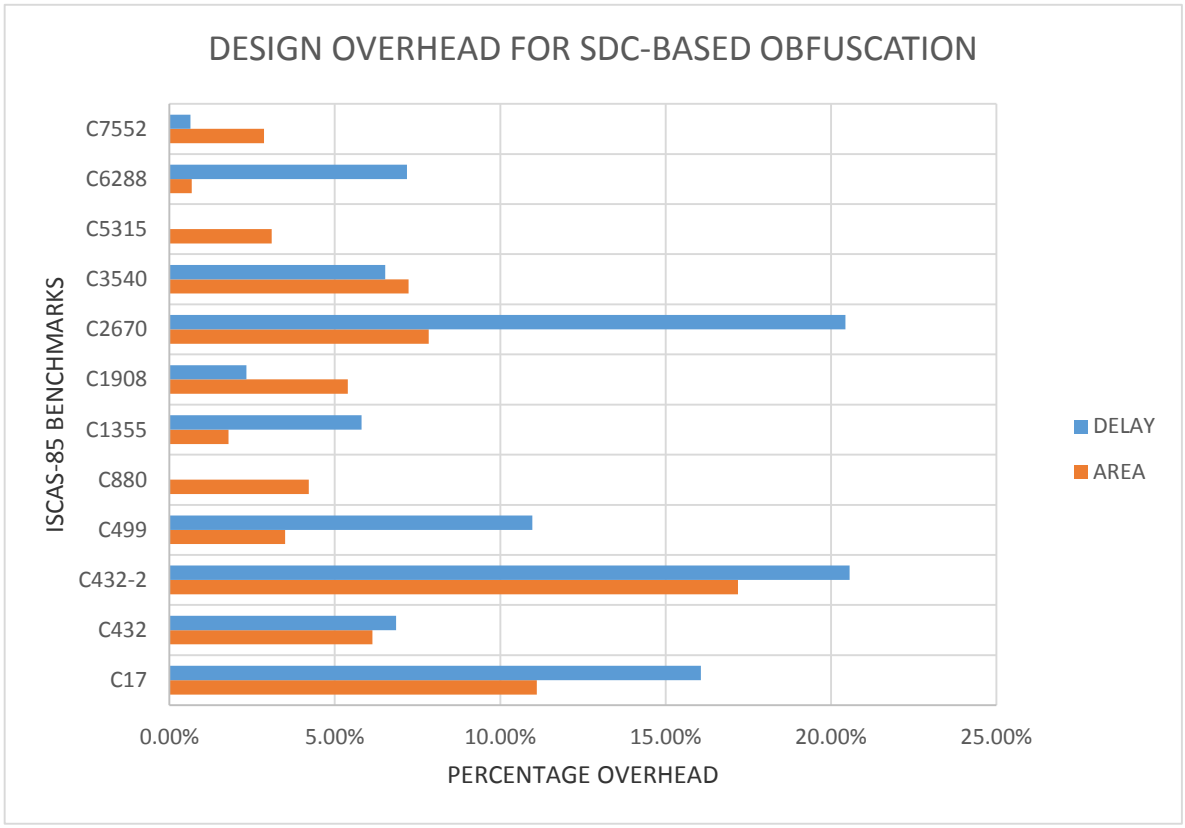


Figure 4.6: Satisfiability Don't Care-based Obfuscation: Overhead results for ISCAS-85 Benchmark Family. C432-2 represents C432 benchmark with two intersection cones camouflaged (with sizes 5 and 4 respectively).

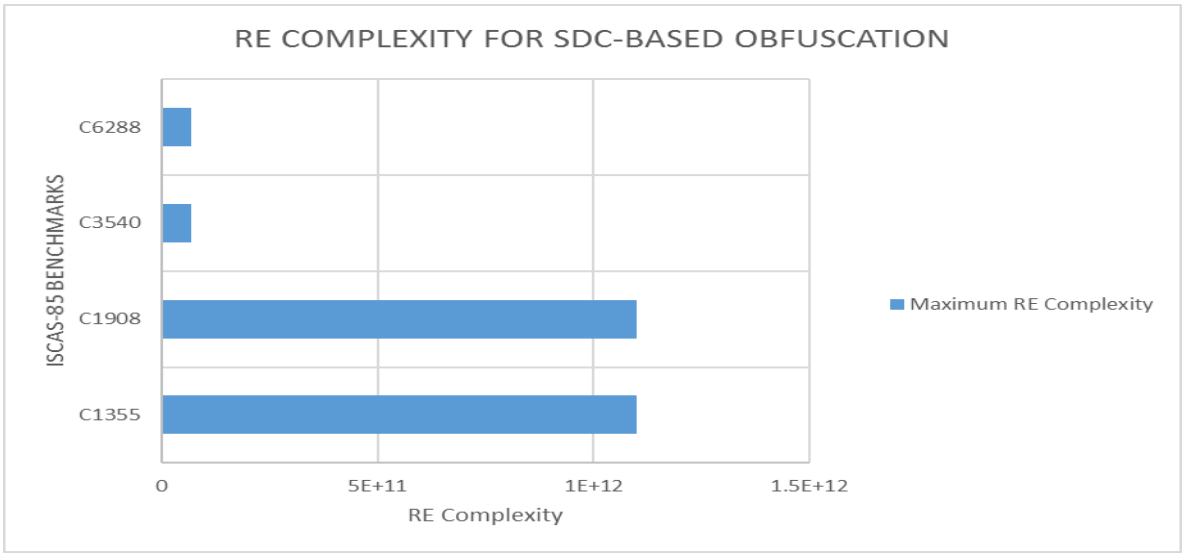


Figure 4.7: SDC-based Obfuscation: RE Complexity for some selected benchmarks

C432-2 represents C432 benchmark with two intersection cones camouflaged (sizes 5 and 4 respectively). The maximum RE complexity in this case is still 16^5 where 5 is the size of the largest intersection cone.

Note that RE Complexity for most practical circuits will be equal to or greater than these values because most practical circuits usually have a large number of gates.

Other overhead considerations

There is another important fact to consider when looking at these experimental results. The tests run were done on circuits that had at most thousands of gates, and many had much less. The RE Complexity exclusively depends on the number of possible candidate gates in the largest intersection cone. Therefore, an n-gate replacement in a 500 gate circuit will result in a similar complexity/ RE effort to that of a 5,000,000 gate circuit, but in the latter it will give a much smaller percentage area overhead. Figure 4.8 (taken from [20]) shows the sizes of the ASICs produced in both Taiwan and China in 2004 and 2005, and as can be seen, approximately 75% of the circuits have a size greater than 100,000 gates. This will only trend upwards with time, as process

technology improves. This means that even with the data in Figure 4.6 and Figure 4.7, most production circuits will have a negligible area overhead compared to our results for a comparable or higher RE Complexity. The delay overhead will depend on whether the obfuscated gates lie on the critical path or influence it.

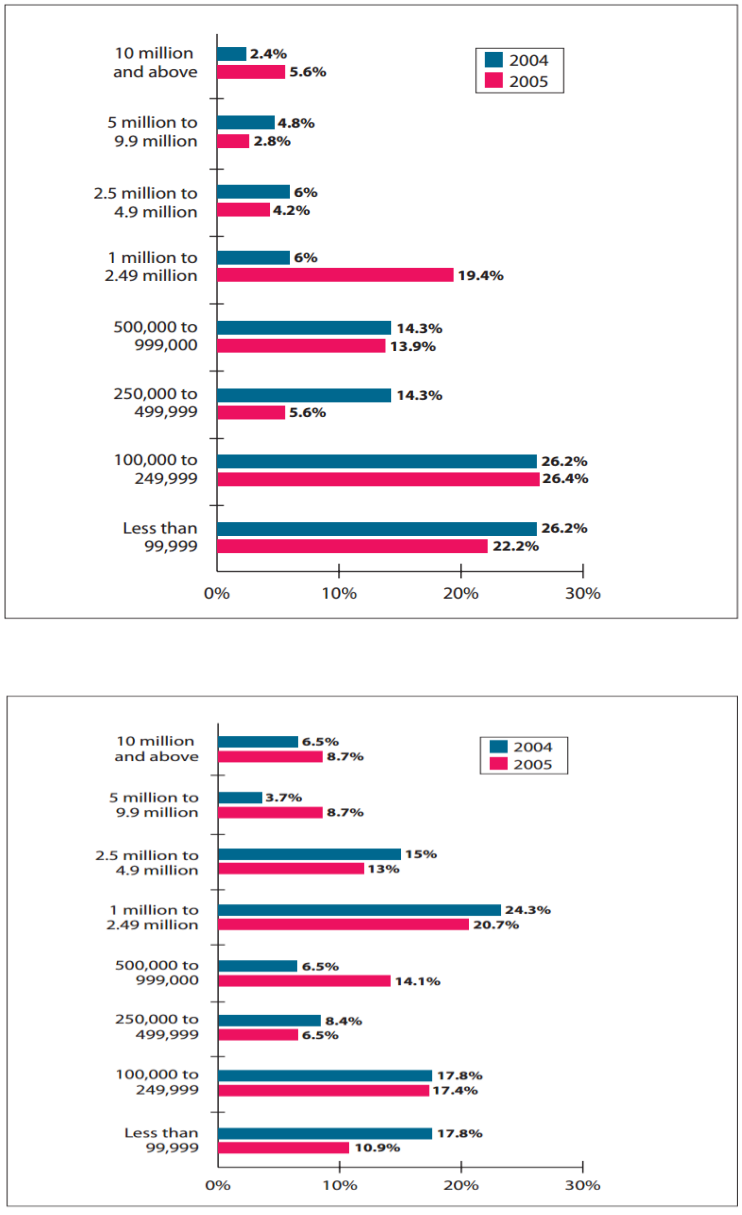


Figure 4.8 : From 2005, (a) ASIC gate counts in Taiwan (b) ASIC gate counts in China [41]

4.7 Conclusion

We propose a novel layout obfuscation approach for IP protection based on the well-known concept of Satisfiability Don't Care (SDC) conditions. Utilizing the SDCs in the circuit, we find alternatives to replace a gate, and use these conditions to select valid functions in the circuit. More importantly, we propose to use configurable cells (such as multiplexers) as the replacement, which allows us to do the configuration at the post-silicon phase of the IC design cycle. Figure 4.9 shows our proposed secure IC design flow for the fabless business model that takes advantage of "split manufacturing" to protect the rights of the IP vendor. We assume that the IP Vendor, in this case, has the complete gate-level description and after camouflaging certain gates according to our obfuscation scheme, the netlist with camouflaged gates i.e. gates replaced with 4x1 multiplexers, is forwarded to the design house for analysis and synthesis of GDS-II database file or hard IP form. The Truth Table values for the 4x1 Multiplexers, and connections for some SDC/ODC signals that do not affect overall circuit functionality, as well as the connection of deliberately introduced invalid functions can be deferred to the very last stage of IC design during assembly and test to bring down design costs. It should be noted, however, that the fan-out of camouflaged gates may affect the critical path delay or even change the critical path of the circuit so these modifications should be made carefully, preferably during simulation and synthesis. However, for some circuits this step can be deferred to the later stages of chip design such as routing and placement, given that they do not impact the critical path. A programmable connector that can be connected to VDD or GND as proposed in [17] can be used to configure the 4x1 multiplexers. At the same time, test vectors are supplied at each stage of the design and test to validate the relevant features of design. For example, test vectors that do not impact the camouflaged gates can be applied before the 4x1 multiplexers are configured during the testing phase. At the fabrication stage, only high-level input/ output relationships are

tested. This deferment of configuring truth table values and some circuit modifications until the routing and placement and final assembly and testing stage is expected to strengthen the security of the design flow. It would be especially useful in keeping the complete layout information obscured from the IC design house. The testing house can be provided with just the high-level description and the location of the Multiplexers while withholding the details of the circuit functionality.

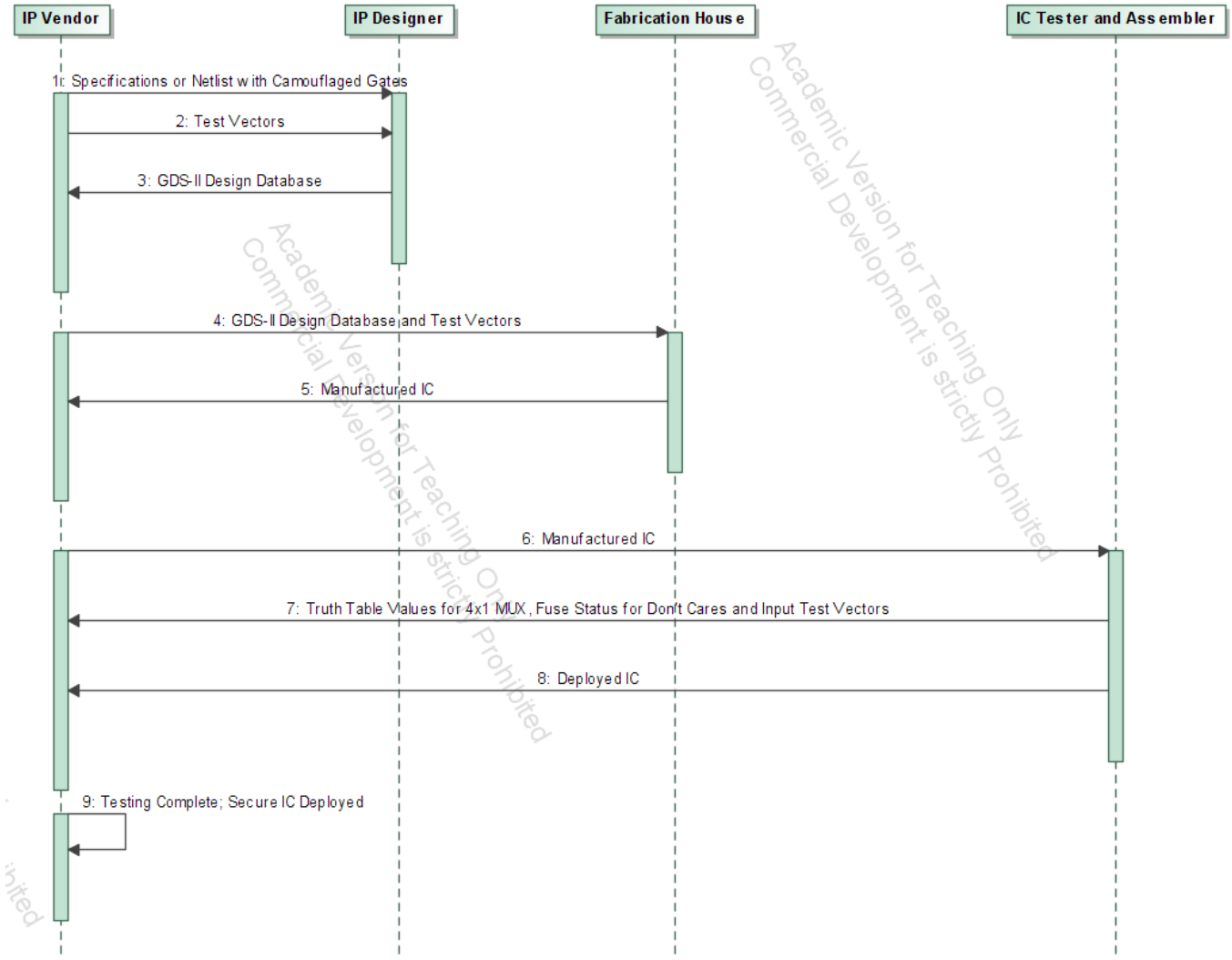


Figure 4.9: Secure "Split Manufacturing" Approach in IC Design that involves the participation and co-operation of several organizationally and geographically separate entities but protects the rights of the IP Vendor.

This approach is promising, as validated by the simulations which demonstrate the RE complexity for these obfuscated circuits. We make some recommendations to further improve the strength of obfuscation (such as selecting gates other than from the largest intersection cone for obfuscation, obfuscating 3-input or 4-input gates for layout obfuscation, etc.) The design overhead for this approach is found to be acceptable as we do not consider a delay-driven design. Instead, we chose valid function locations from the largest intersection of the output cones which enables us to eliminate weak obfuscation locations that can be resolved simply by justification and sensitization. Furthermore, provided that we avoid picking gates from critical path, we can choose other obfuscation locations from cones other than the largest intersection cone. Future work may include detailed evaluation of performance-driven methods.

CHAPTER 5: OBSERVABILITY DON'T CARE-BASED OBFUSCATION

Our second work in this dissertation was to develop a method for layout obfuscation of circuits using Observability Don't Care conditions (ODCs). Observability don't cares (ODCs) can be found in almost any circuit that has more than a trivial number of logic gates, and thus offer an excellent method for hardware obfuscation of IPs. We also propose a method for functional obfuscation using don't cares that modifies sub-circuit functionality while meeting the overall circuit functionality. Our method generates little overhead and a large number of potential ODC-enabled camouflaged functions are identified. The method is very similar to the SDC-based Functional Obfuscation described in Chapter 4, so many of the details will be omitted if they have been described before.

5.1 Introduction to ODC-based Obfuscation

To start we first use a small example to show the basic idea of ODCs.

Illustrative example

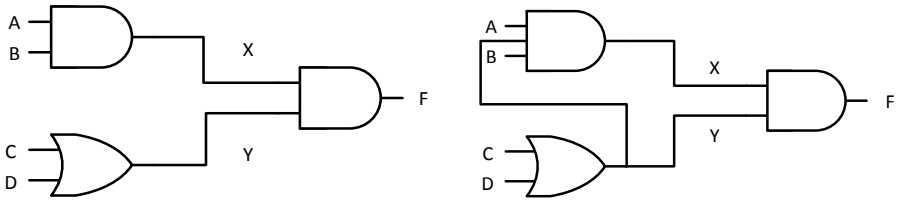


Figure 5.1: Two 4-input circuits that implement the same function

The left circuit in 5.1 realizes the function $(AB)(C + D) = F$. When the Y input to the AND is zero, the output F will be zero regardless of the value of X input; however, when Y=1, F will be determined by the X input. So when we direct signal Y to the AND gate that generates X, as shown in the right of 5.1, we can easily verify that this circuit implements the same function F. However,

these two circuits are clearly distinct. The ODC condition in this case is $ODC_{X} = Y'$ which means that when Y=0, the input X to the AND gate can be ignored, and is a "don't care".

One key feature of this approach is that the changes we make to the circuit are minute. We can make a connection, as shown on the right circuit in Figure 5.1, during routing and placement; then use the don't care signal in the circuit as select bit for a 2x1 MUX with valid and 'deliberately invalid' functions as inputs. For example, we might have AND (Y',F) as the select bit for the 2x1 MUX. In this case, the select bit will necessarily be '0' and always pick the valid function connected to input '0'. In another embodiment, explained further in section 5.3.1, valid function and its camouflaged version are applied as inputs to the 2x1 MUX.

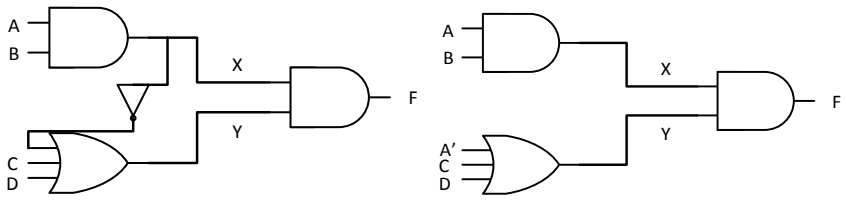


Figure 5.2: Two more implementation of the same function

Challenges and contributions. As simple as the above example suggests, there remain several technically challenging questions to develop a systematic obfuscation technique based on this idea. For example, the two circuits shown in Figure 5.2 also implement the same function F as the circuits in Figure 5.1 where $ODC_{Y=}X'$. The contribution of this paper is the discovery of the proposed practical obfuscation method based on ODCs and its implementation.

First, how to efficiently identify the locations in the circuit where we can make changes to obfuscate the circuit? Second, at such identified locations, what kind of changes to the circuit can we make? Third, what is the reverse engineering complexity offered by this approach to obfuscation? Finally, what is the impact to the design quality such as area and delay after camouflaging the circuit?

We propose to first determine ‘valid function locations’ as defined in Chapter 4. These are gates in the circuit that belong to the intersection of two or more output logic cones. We pick the largest of such intersections i.e. the intersection with the largest number of gates and proceed to camouflage them using ODCs. This ensures that a malicious user would have to resolve all the camouflage gates to be completely assured of their functionality, but because these gates are interfering this is a complicated task. Therefore, the RE complexity would be relatively high and it would be a computationally expensive operation, discouraging the malicious user.

Once the ‘valid function locations’ are identified, we analyze the circuit locally to determine what kinds of changes can be made by identifying ODC conditions in the fan-in of the valid functions or elsewhere in the circuit. Because ODC conditions exist almost everywhere in any combinational circuit, this provides us a large space for gate camouflaging (replacing the ODC gate with a 4x1 MUX as explained in section 4.4). When we design circuits by the consideration of adding obfuscation after fabrication, we realize that reasonable area and power overhead have incurred, but the delay penalty is too large to accept. Therefore, one recommendation is a delay-driven heuristics to create obfuscated copies under delay constraints. In our implementation, we just obfuscate the largest intersection cone which provides us a measure of the maximum RE complexity.

Observability Don’t Cares

In this work we use the concept of observability don’t cares (ODCs) to introduce layout obfuscation in the circuit, so first we will go over the concept of an ODC.

Observability don’t cares are a concept in Boolean computation. The conditions by which an ODC occurs are when local signal changes cannot be observed at the primary output. An example of this can be seen in Figure 5.3 below. When the bottom AND gate has an input equal to zero, it will generate a zero as its output. This output will be propagated to the next AND gate and generate

another zero as the primary output for this circuit. The signals from C, A, and B are all blocked and cannot be observed at the primary output.

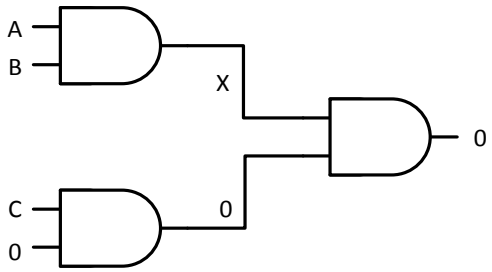


Figure 5.3: ODC on the AND gate

ODCs can be several layers deep and can cause several different signals to be blocked, depending on the input to the circuit.

Formally, the ODC conditions of a function *F* with respect to one of its input signal *x* can be defined as the following Boolean difference:

$$ODC_x = \left(\frac{\partial F}{\partial x}\right)' = (F_x \oplus F_{x'})' = F_x F_{x'} + F_x' F_{x'}' \qquad (1)$$

This equation has a few parts to define. First we have ODC_x which simply represents if we have an ODC that occurs at signal *x*, essentially that is there some condition that causes signal *x* to be ignored. The next part of the equation $\left(\frac{\partial F}{\partial x}\right)'$ states that what we are looking for when determining ODC_x is when the output of function F does not change despite a change in the signal *x*. The next two pieces of this equation, $(F_x \oplus F_{x'})'$ and $F_x F_{x'} + F_x' F_{x'}'$ represent the actual functional situation where an ODC occurs at *x*. F_x represents the normal output of function F when $x = 1$ and $F_{x'}$ represents the output of F when x = 0. The next two symbols F_x' and $F_{x'}'$ simply represent the inverted output of F_x and $F_{x'}$.

Basically (1) states that when we have a function *F*, and a variable *x*, when the condition ODC_x is satisfied, the value of variable *x* will not have any impact to the value of the function *F*. In the above example, the final output is produced by a 2-input AND gate, for one of its input *x*, if the other input is *y*, then from equation (1), we have $ODC_x = y'$. Hence, when the other input has value zero (as shown in Figure 5.3), input *x* becomes a don't care.

Using this equation we are able to derive a list of ODC calculations for the gates in the library we use in the experiments for this work (packaged with ABC [23]), as seen in Table VI. Gates such as inverters, XORs, and XNORs do not have ODCs and are omitted from the table. Other gates such as 3- and 4-input NORs and NANDs, and 4-6 input AOIs and OAIs are simple extensions of the gates listed in Table VI. For NAND, NOR, AOI22, and OAI22, all the inputs are symmetric, so we only compute the ODC for one of the inputS. For gates like AOI21 and OAI21 that have asymmetric inputs, we compute ODC for all the asymmetric inputs.

Table VI: ODC Calculations for the Library Cells (from [20])

Gate	*Gate Equation*	*ODC Calculation*
NAND	$O = \overline{AB}$	$ODC_A = \bar{B}$
NOR	$O = \overline{A + B}$	$ODC_A = B$
AOI22	$O = AB + CD$	$ODC_A = \bar{B} + CD$
OAI22	$O = (A + B)(C + D)$	$ODC_A = B + \bar{C}\bar{D}$
AOI21	$O = A + BC$	$ODC_A = BC$
		$ODC_B = A$
OAI21	$O = A(B + C)$	$ODC_A = \bar{B}\bar{C}$
		$ODC_B = \bar{A}$

5.2 ODC-based Layout Obfuscation Methodology

The goal of this work is to find a way to improve upon current obfuscation techniques. Most current techniques, such as the work in [15] and [21], use gate-level obfuscation to camouflage circuits. These approaches aim to obfuscate the layout of the circuit by inserting key gates (in the form of XOR/ XNOR gates) into the circuit or by replacing candidate gates with other gates that can

perform a variety of functions that the attacker would not be able to decipher without performing extensive computations (see section 2.1.2 in Chapter 2 for details). The key gates are chosen so that the attacker would not be able to leverage the principles of justification and sensitization.

Our method attempts to use ODCs to create small changes in the circuit to obfuscate the layout; we also propose proof-of-concept of functional obfuscation using don't cares that modifies the underlying functionality of the sub-circuit while meeting the overall circuit specification. Additionally, the final step of camouflaging the valid function or the ODC trigger condition, as we see in section 5.3, can be implemented in the later stages of the VLSI design cycle (routing and placement). Although our implementation concentrates on camouflaging just the largest intersection cone of two or more outputs for maximum RE complexity at minimum overhead, the results indicate that it is indeed viable to obfuscate more gates in the circuit for strengthening security.

Although this work is centered on the concept of utilizing ODCs to create small modifications throughout a circuit, several challenges prevent it from being a simple task. First, conditions need to be established to find locations suitable for modification. Once a location is established, a modification may be made. The second challenge is that the modification is dependent on the location and gates at that location. Finally, once all of the modifications have been made, it is important to prevent unacceptable overhead from occurring due to the modifications. In the rest of this section, we elaborate on these challenges and our solutions to solve them.

5.2.1 Finding Locations for Circuit Modification based on ODCs

Every logic circuit that is created uses a library of gates that determines the logical relationships that can occur. Most libraries contain gates that create ODCs as defined using Equation (1), but not every instance of these gates will be able to be modified to accommodate our obfuscation method.

Definition. (Valid Function Location). *A valid function location is defined as a gate that lies in the intersection of two or more output logic cones. The gates in the largest intersection are chosen for obfuscation as they determine the upper-bound on RE complexity.*

A potential ODC location is defined as two or more gates that can be considered for modification for a circuit without changing the overall functionality of the circuit. These gates consist of a single primary gate and one or more gates that generate inputs for the primary gate that meet the following criteria: (this list of criteria is taken from [20] where ODC locations are used to embed a fingerprint in the circuit).

1. *The primary gate must have at least one input that is not a primary input of the circuit.*
2. *The primary gate must have at least one input which is the output signal of a fan-out free cone (FFC), which means that this signal only goes into the primary gate.*
3. *The FFC in criterion 2 must have either a gate with non-zero ODC (such as those in Table 2) or a single input gate (e.g. an inverter).*
4. *The primary gate must have a non-zero ODC with respect to one or more of its input signals other than the one from the FFC.*

Criterion 1 is necessary for making local minor changes to the circuit (for obfuscation purpose). Criterion 2 ensures that the changes made to the FFC will not affect the functionality of the circuit elsewhere. Criteria 4 provide a possible signal that can be added to a gate in the FFC (in criterion 3). Each ODC location, in a circuit is analyzed using this criteria and if it satisfies all the criteria in the definition, it is then considered to be a modifiable location, a location where the circuit can be modified to incorporate obfuscation.

5.2.2 Determining Potential ODC Modifications

For each modifiable location found above, a modification can be applied to the gate's inputs. A generic modification is depicted in 5.4.

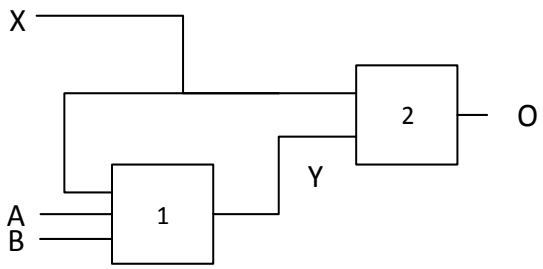


Figure 5.4. Generic ODC modification

Figure 5.4 has two generic gates, represented as boxes 1 and 2, three primary inputs (X, A, and B), and one primary output (O). Gate 2 represents the primary gate, gate 1 represents the gate within the FFC that generates signal Y, and signal X is independent of the FFC that generates signal Y. Suppose that signal X satisfies ODC_Y, thus we can add signal X into the FFC of Y, i.e. gate 1 as shown in Figure 5.4, either in its regular form X or its complement form X'. However, when we make this addition, we need to guarantee that when signal X takes the value that does not satisfy ODC_Y, it will not change the correct output value Y. In the rest of this paper, signal X will be known as an ODC trigger signal, as defined below.

Definition. (ODC Trigger Signal) *An ODC Trigger signal is a signal that feeds into a gate, with a non-zero set of ODC conditions, which causes the ODC condition to activate. In the context of this work it also represents the signal that is used to modify the input gate to the primary gate for obfuscation purposes.*

In order for this to work, the relationship between the signal X, gate 2, and gate 1 must be analyzed so that X only changes gate 1's output, Y, when it also triggers the ODC, criterion 3 in the definition

of ODC location. For every possible pair of gates that can be considered a modifiable location, similar to gate 1 and gate 2, a structural change must be proposed in order to modify that location. Modifications with certain gates may not always be feasible, especially if the overhead costs are too large.

For simple changes like the one in Figure 5.4 or those in the motivation example, each location like this is an ODC location. For each circuit that is manufactured, this ODC location can be used to control a 2x1 multiplexer that has a valid function (gates in the largest intersection of output cones) and camouflaged form of the valid function as inputs (section 5.3.2).

In addition to the simple change in Figure 5.4, an ODC location may also consist of more than one input gate as stated in the Definition. If this is the case, a modification similar to the one in Figure 5.4 may be applied to each of the input gate in the FFC that abides by criterion 3. If this situation occurs, these additional input gate locations can be used to control other 2x1 multiplexers in the circuit.

It is also possible to leverage certain gate combinations to add to our ODC locations and also to reduce the delay that can be caused by rerouting signals. If the ODC gate that is being considered, is immediately preceded by another ODC gate with the same ODC trigger signal, such as the two ANDs in Figure 5.5 or for example a NOR preceded by an AND, the fact that the ODC would be triggered before the X signal is generated in Figure 5.5 can be utilized to directly feed one or two input signals into the gate from the fan out free cone. The OR gate in Figure 5.5 shows an example of this. Input signals, A or B are simply inverted and directed into the OR gate, removing the need to feed signal X into the OR gate which cause a delay if the gates on the left hand side have equal delay.

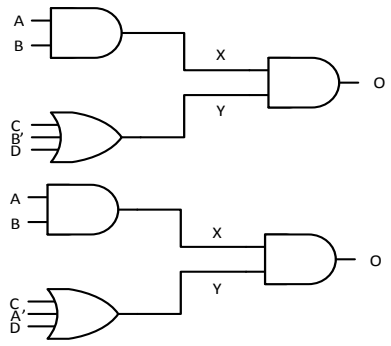


Figure 5.5. ODC change that reroutes earlier signals

This additional method of ODC change allows us to consider more ODC locations. For every gate combination like this we are able to add at least $n(n+1)/2$ potential changes, where n is the number of inputs to the ODC trigger gate, because all combinations of the inputs to the ODC trigger gate can be added to the inputs of the fan out free cone gate. In sum, with only one ODC location, we may be able to modify the circuit in potentially many different ways and thus facilitate obfuscation.

The above approaches are general recommendations taken from [20], and may come in handy when trying to obfuscate large circuits. The approach we use for our test circuits from the ISCAS-85 Family is to simply use the ODC condition (e.g. $ODC_{x=}Y'$) and use this condition in conjunction with the output 'F' in some manner to be used as select bit for 2x1 multiplexer. This multiplexer has the valid and invalid functions as inputs. For example, we can use camouflaged version of AND (Y', F) and use its output as select bit for this 2x1 multiplexer as explained in the next section. In another embodiment, we use the don't care signal resulting from the ODC condition being satisfied, directly as select bit, and feed the valid function and its camouflaged version as inputs to the 2x1 multiplexer. Yet another method of obfuscation using ODC's is presented in section 5.3.2.

Maintaining Overhead Constraints

The ODC-based modifications proposed can cause a large overhead, relative to the circuit's initial performance, especially if the valid function locations defined earlier in 5.2.1 lie on the critical path. Rerouting paths, utilizing all the input gates to the primary gate, and introducing new inverters, and camouflaging the ODC gate with a 4x1 multiplexer are the cause of the overhead. Two heuristic methods are recommended for reducing this overhead, a reactive method and proactive method. These recommendations are inspired from [20].

Of the two methods, the reactive method is easier to implement but appears difficult to scale. This method involves taking a fully obfuscated circuit (that has all the gates in all the intersections obfuscated) and by removing one modification at a time, analyzing the difference in overhead, whether it be area, delay, power, or something else. The modification that results in the largest change to the overhead is removed and the resulting circuit is tested again. This is done until a certain overhead constraint is met or there are no more modifications to remove.

The proactive method is more difficult to implement, but because it is done as modifications are applied it scales well with larger circuits. This heuristic requires that each modification is analyzed before being implemented. For area and power this is simple because any new gates or changes in gates will result in an overhead that can be estimated using information about the cells in the library. Delay is more difficult to analyze because not every modification will slow the circuit down. As modifications are added the critical path may change which changes where new modifications should be considered. The delay can be estimated by determining the slack on each gate and updating the information every time a modification is made, but this can be time consuming for large gates that will have a large number of modifications. For this proactive method, modifications would be added until a certain overhead constraint was met, the opposite of the reactive method.

5.3 ODC-based Modification Methodology

We present two methods for ODC-based layout obfuscation. These methods are combined with direct layout obfuscation (i.e. directly replacing a candidate gate with a 4x1 multiplexer that implements the same function) and SDC-based obfuscation and the results are shown at the end of the chapter.

5.3.1 Method 1

Although these recommendations are made at this point, the approach we adopted combines ODC-based modifications with SDC obfuscation method described in Chapter 4. We considered valid function locations which are gates located in the largest intersection of two or more output logic cones. The valid function is applied as input to a 2x1 multiplexer while the other input can be a misleading function from some other cone. The select bit of the 2x1 multiplexer is the don't care signal resulting from an ODC condition being satisfied. For example, for a two input AND gate with inputs X and Y, we note that ODC_X =Y' and therefore, the signal AND (Y', F) would always be zero. On the other hand, OR (Y', F) is always 1. This latter condition can be considered inverse of the SDC condition (a condition that always holds but is camouflaged from the user). Therefore, we find that OR(Y', F) can be camouflaged by replacing it with a 4x1 Multiplexer, and its output signal can be used as select bit for the 2x1 multiplexer while the valid function and a deliberately introduced invalid function are applied as inputs to the 2x1 multiplexer This is very similar to what we discussed for the SDC-based obfuscation method. In another embodiment illustrated in Figure 5.6, the don't care signal itself (which may be 0 or 1) resulting from an ODC condition being satisfied is used directly as the select bit, while the valid function and its camouflaged version are applied as inputs to the 2x1 multiplexer. Therefore, regardless of whether the don't care signal is zero or 1, the correct value is always outputted by the multiplexer. In general, we combine these

SDC-like modifications (conditions that are always true but camouflaged) with ODC-based obfuscation illustrated in Figures 5.4 and Figure 5.5. So, ODC-based modifications are very similar to SDC-based modifications as seen in the similarities between Figure 4.3 and Figure 5.6.

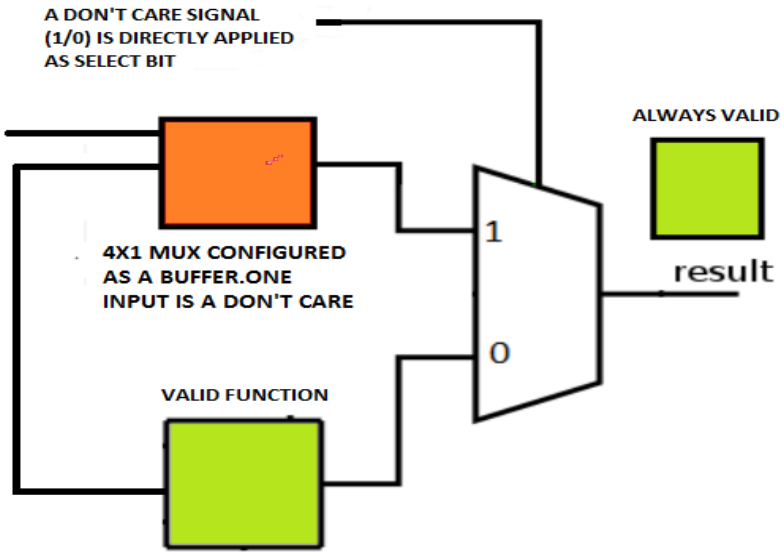


Figure 5.6: Essential Ingredients of ODC-based Layout Obfuscation (a) the green block represents a valid function (b) the orange block is the camouflaged (buffered) version of the same valid function with one misleading input (c) the 2x1 MUX is triggered by a don't care signal in the circuit. The ODC Gate generating the don't care signal does not need to be camouflaged.

We know that for a NAND gate with inputs X and Y, ODC_x=Y'. If this is the case, the trigger gate for the 2x1 multiplexer can be AND (Y', F) as shown in Figure 5.7: the output of this AND Gate may be 0 or 1. The valid function and its camouflaged version are applied as inputs to the 2x1 Multiplexer, therefore the output of this multiplexer is always G16, regardless of whether the select bit is 0 or 1.

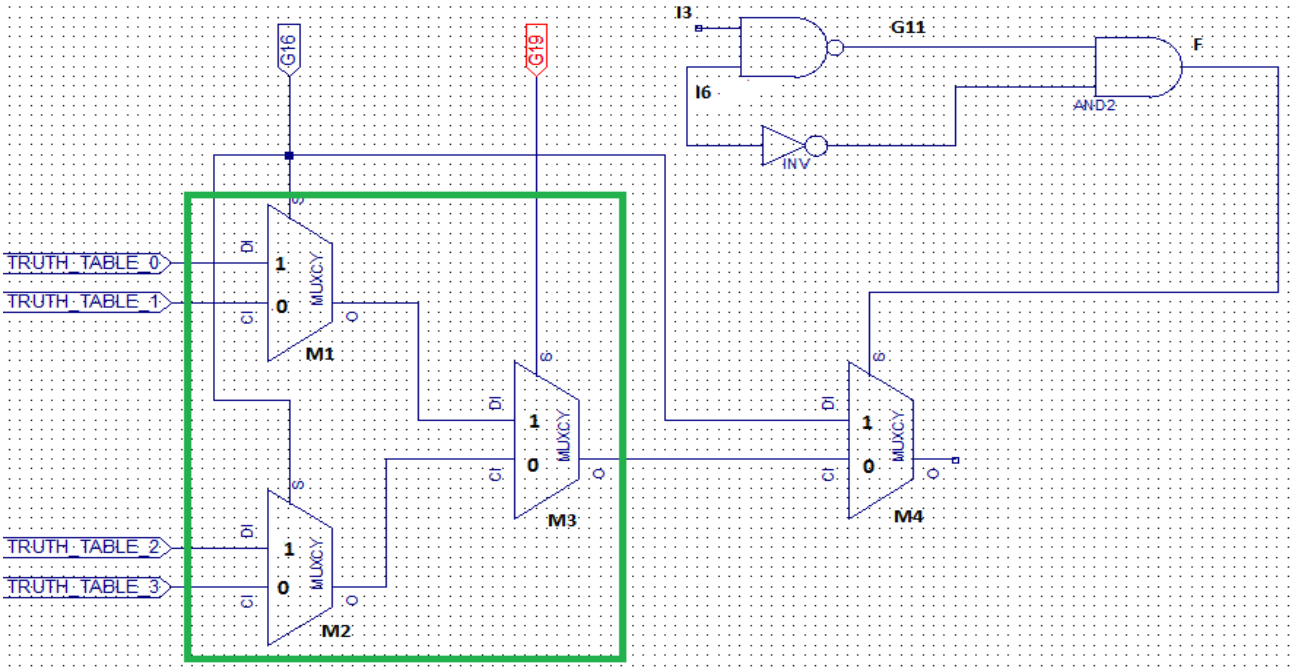


Figure 5.7: Layout Obfuscation using ODC (Method 1): The AND gate does not need to be camouflaged

The green block in the figure above represents a 4x1 multiplexer. Suppose the truth table values for the 4x1 multiplexer are {0, 0, 1, 1}. This configures the 4x1 multiplexer as a buffer with output equal to G16. Since the select bit of the 2x1 multiplexer M4 in this case can be either '1' or '0', the output of M4 is always G16. Suppose that, instead of the AND gate, we have OR of Y and F as the select bit of the 2x1 multiplexer. In this case, the select bit will always be 1 (Y+F =1). We term this condition as an SDC-like condition (a condition that always holds but is camouflaged from the user), and we can connect a deliberately incorrect or misleading function to input '0' of the multiplexer. This is shown in Figure 5.8. The valid function is the output of G16, while the invalid or deliberately misleading function is G19. These SDC-like conditions are especially useful if a fan-out free cone (a signal in the fan-in of the ODC gate that does not lead anywhere else) does not exist. When combined with ODC-based modifications, they can be used to introduce additional confusion for the attacker.

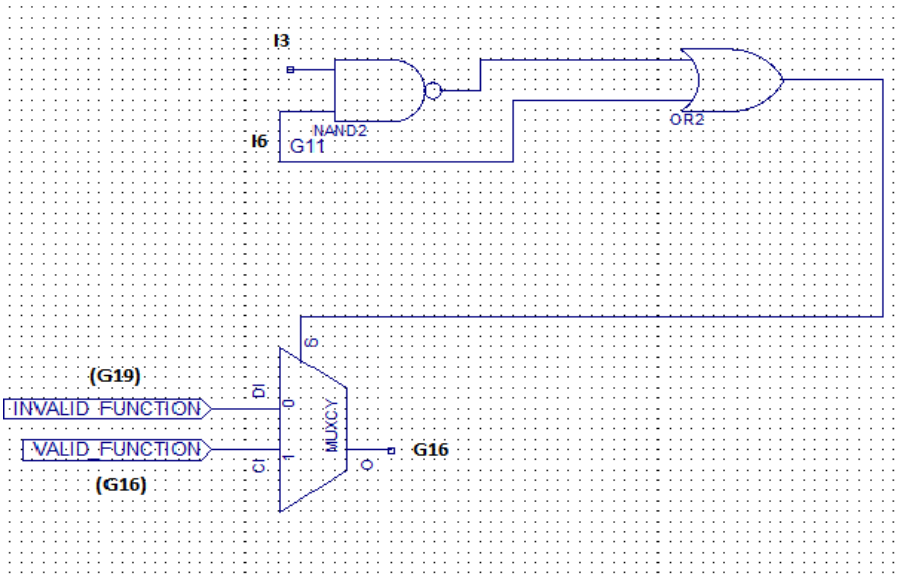


Figure 5.8: Layout obfuscation using SDC-like gate. The OR gate is camouflaged and provides the select bit for the 2x1 multiplexer that always selects the valid function G16.

Figure 5.9 shows yet another approach that directly uses the ODC (similar to the don't care signal X in Figure 5.1) as select bit for the 2x1 multiplexer. This is very similar to the method presented in Figure 5.7 but uses the don't care directly as the select bit of 2x1 multiplexer.

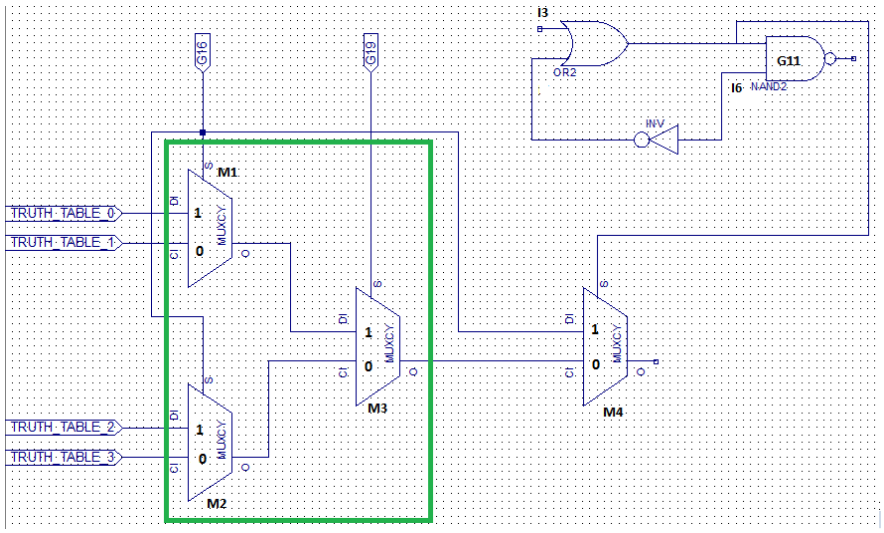


Figure 5.9: A don't care, such as input X from Figure 5.1 (right), is used as select bit for the 2x1 multiplexer. Both the inputs to this multiplexer are the same 'valid function' but they are

camouflaged so an attacker cannot directly observe them. The OR Gate does not need to be camouflaged in this case.

In this Figure, suppose the truth table values for the 4x1 multiplexer are {0, 0, 1,1}. Then, whether the select bit is '0' or '1' (a don't care) the output of this circuit is always the valid function, G16. This is because the camouflaged 4x1 multiplexer implements a buffer with one don't care input G19 (see Row 3 in Table III).Since G11 is a NAND gate, the ODC condition for this gate (C=NAND(X,Y)) is $ODC_X = Y'$; when I6=0, output C=1 but when I6=1, C may be '0' or '1' depending on the input I3. Regardless of whether it is '0' or '1', the output of M4 is G16 because of the way the 4x1 multiplexer is set up. This approach is useful because the select bit is not constant in this case.

5.3.2 Method 2

In Method 1 described above, the strength of obfuscation relies on the attacker not being able to determine that one input of the 2x1 multiplexer is the camouflaged version of the other input (i.e. 4x1 MUX configured as a buffer). Therefore, regardless of the value of select bit (a don't care), the correct function will always be outputted.

However, there is one weakness in the approach as described: the inputs to the 2x1 multiplexer are asymmetric which may make it relatively easy to guess the configuration of the buffered input. Therefore, we propose making the inputs to the 2x1 MUX symmetric by using a don't care input (the select bit) and a valid function. In place of the AND gate, we can also use a camouflaged OR gate. In this case, we would be exploiting the ODC Condition on the camouflaged OR Gate at the input of the 2x1 MUX. The set-up is shown in the following figure (using AND gate; the setup is identical to when an OR gate is used):

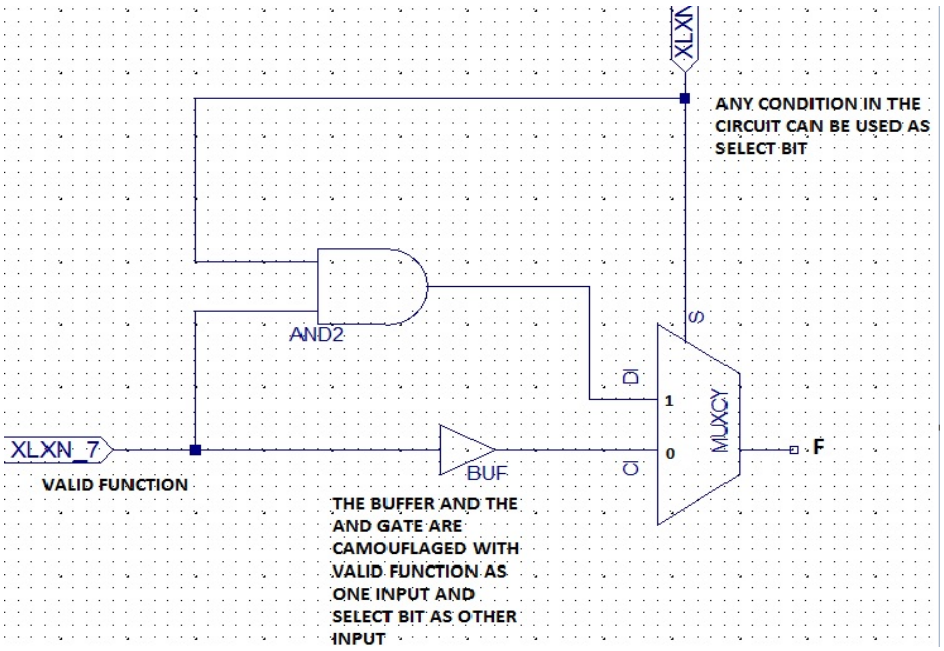


Figure 5.10: Making the inputs to the 2x1 MUX symmetric for don't care-based obfuscation

The buffer and AND gate are camouflaged i.e. replaced with a 4x1 MUX that has select bit as one input and the valid function as other input. We repeated the results for this configuration, using a mixture of SDC and ODC conditions, as well as directly camouflaging some of the candidate gates to obtain the results for the ISCAS-85 Benchmarks given in Table IX**.**

5.4 Security Analysis

Like the SDC-based obfuscation method, the strength of the ODC-based obfuscation depends on the RE complexity introduced by the largest intersection cone. It is trivial for the IP designer to detect the obfuscation embedded by our proposed approach because the designer can compare the obfuscated IP with the design that does not have any obfuscation to check whether and what change has occurred at each location.

However, it is computationally infeasible for an attacker to reveal all the valid function locations or ODC locations. This is especially true for the methods recommended in Figure 5.4 and Figure 5.5.

When a particular ODC location is modified, the FFC of the obfuscated IP will include the signal that is not in the FFC in the original design when the modifiable location is identified. Consider the left circuit in Figure 5.1 of the motivational example, the FFC that generates signal X contains only the 2-input AND gate with A and B as input. But when signal Y is added to this AND gate, the FFC will include the 2-input OR gate with C and D as input. This will make this portion of the circuit not an ODC location (criterion 4 is violated).

More importantly, in our approach SDC-like modifications (using conditions that are always true but camouflaged) is combined with ODC-based obfuscation, and some pre-existing gates are camouflaged (replaced with 4x1 multiplexers) together with obfuscating the gates preceding existing 2x1 multiplexers, the effort required to reverse engineer the circuit would increase significantly.

The attacker can still potentially identify all obfuscated locations by simply observing the layout. As noted above, if the ODC-based obfuscation is combined with SDC-based obfuscation, and if pre-existing 2x1 multiplexers are obfuscated similarly (their preceding gates replaced with configurable 4x1 multiplexers), then this attack can be foiled due to computational complexity of reverse engineering.

We study the performance implications of the proposed approach after a brief description of the experimental setup.

5.5 Experimental Setup

To prove the usefulness of this new method, a circuit modifier was constructed in C++ based on the description in the previous section. We started with the ISCAS-85 benchmark circuits in the bench format, which specifies the circuits' physical layout and connectivity, but not the logical behavior. The netlist was converted to a directed graph which was manipulated by an algorithm written in

C++ to find all the intersections of the output cones. The largest such intersection was chosen for modification using ODCs and SDCs (the gates within this largest intersection are termed 'valid function locations' i.e. valid candidates for obfuscation). From here, the circuit was put through Berkeley's ABC [35] program with a library of standard logic gates (OSU035 library from Oklahoma State University was used for this purpose in the genlib format). The ABC program can map a bench file to a Verilog netlist using standard gates in the library.

For each valid function in this intersection identified by the algorithm, we determined an ODC or SDC-like condition (in the fan-in of the valid function or elsewhere in the circuit) which was used as select bit for a 2x1 multiplexer. The valid function itself and a deliberately chosen invalid function were fed as inputs to the 2x1 multiplexer for SDC-based obfuscation. The final step was replacing the SDC gate with a 4x1 multiplexer that implements the same function. In case of ODC Method 1 (Figure 5.9), the valid function and its camouflaged version are fed into the 2x1 multiplexer. ABC also allowed us to get both the area and delay of the modified benchmark circuit to compare with the design parameters obtained for the original benchmark. For Method 2 (Figure 5.10), we introduce a camouflaged "buffered" valid function at one input of the 2x1 MUX, a camouflaged AND gate at the second input, configured so that these two inputs appear symmetric, and introduce virtually any randomly chosen circuit signal as the select bit. This same randomly chosen signal is also input to the (camouflaged) AND gate and the buffer, while the other input to these two logic gates is the valid function (see Figure 5.10). Both methods of ODC-based obfuscation as described in this paragraph are combined with SDC-based modifications in the final implementation.

Pseudocode for ODC-based Obfuscation

Our first goal was to create a program that could implement the details of section 5.3 (Method 1 illustrated in Figure 5.9 combined with SDC and SDC-like modifications). Then, we slightly modify the approach in Method 1 to implement Method 2 (Figure 5.10) that makes the inputs to the 2x1 Multiplexer symmetric. We started by gathering all of the gate data from the benchmarks, including: gate type, fan-outs, fan-ins, and the number and types of gates that lie in the intersection of two or more output cones.

Combining this information with the ODC calculations in Table VI allowed us to determine what gate combinations were viable obfuscation locations.

Input: ***Original Netlist 'G', Number of Gates to be Obfuscated (gateNum) or Maximum Acceptable Delay Overhead or Acceptable RE complexity***

Output: ***Obfuscated Netlist G"***

Layout Obfuscation using ODC (Method 1; Figure 5.9) and SDC-like Conditions:

1. *for each* $PO\ Z_i$, *calculate corresponding* $FFC_i = \{G_j | \exists path, G_j \rightarrow Z_i\}$;*Here, PO is the Primary Output.i.e. all gates in the fan-in cone of each output.*

2. *for each gate* G_i *in the circuit, calculate corresponding coneSet:*

   $coneSet_i = \{Z_i \mid G_i \in FFC_i\}$ *where* Z_i *is the output gate;*

3. *Determine the gate(s) '*S_i*' which have the same coneSet, say '*$coneSet_x$*' Pick the largest set of gates Si as 'valid function locations.'*

4. *Find ODC for a gate in the fan-out free cone of valid function. Or find SDC-like condition if a fan-out free cone does not exist.*

5a. Use this ODC or SDC as select bit for a 2x1 MUX.

5b. The MUX Inputs are: Input 1 is a valid function determined in Step 4. Input 2 is the obfuscated version of the valid function. In case of SDC-like obfuscation, input 2 will be from any other output cone FFC_j such that $FFC_j \notin coneSet_x$

5c. Use as many ODCs and/ or as desired (=GateNum or number of valid functions or until delay constraint is met or Acceptable RE complexity is achieved for the largest set S_i determined in Step 3).

5d.Camouflage the SDC gates. In case of ODCs, camouflage the valid function input to the 2x1 multiplexer.

6. Calculate the delay overhead for the obfuscated netlist;

7. If Calculated Delay > Max Delay, delete obfuscated gate(s) until delay overhead is met

8. Return the obfuscated Netlist

First, we determine the fan-in cones of each output in the circuit. Next, we determine the primary output gate for each gate in the netlist and thereby determine valid function locations, which are gates that belong to two or more output cones (intersections). The largest of such gate sets is chosen to maximize RE complexity. Example of such valid gate locations for C17 benchmark will be G11 and G16 which belong to both output cones O22 and O23.

The next step is to find an ODC trigger condition, which may be in the fan-in cone of the valid function or elsewhere in the circuit. In Method 1 from section 5.2 (Figure 5.9), the SDC-like condition (which is always true but camouflaged from a user) is triggered so that it always picks the valid function but gate camouflaging hides the trigger function from the attacker. The choice of the other input (the 'invalid' function) from some other output cone not belonging to the intersection would further complicate matters for the attacker. In the other embodiment, since we use the don't care signal (which can be '1' or '0') itself as select bit for the 2x1 multiplexer, the valid function and its camouflaged version are applied as inputs to the 2x1 multiplexer. With many valid function locations in the largest intersection cone as determined in step 4 camouflaged, the complexity of RE goes up exponentially as there is no way to find a valid function from this intersection without finding all the other camouflaged valid functions. One advantage of using ODC conditions in conjunction with SDC-like conditions is that the output of an SDC-gate is constant, while the output of the 2x1 multiplexer with camouflaged inputs, when using ODCs, varies depending on the don't care signal that triggers it. Using both of these conditions together helps strengthen security, at an acceptable performance overhead. An activity diagram illustrating the process of obfuscation based on don't care conditions is given below:

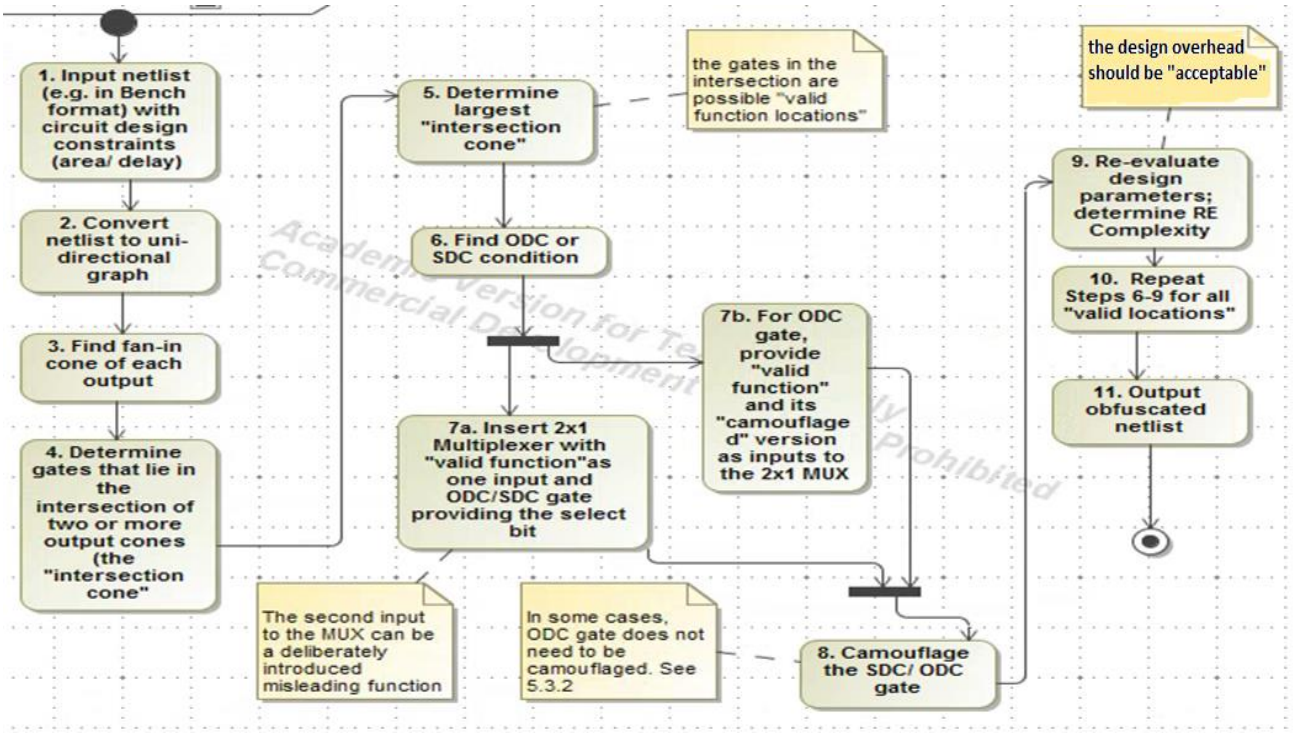


Figure 5.11: Activity Diagram of Don't Care-based Obfuscation Methodology (Method 1)

Please note that we assume the "acceptable" design overhead is less than or equal to 10%-15%. We used a combination of ODC-based obfuscation Method 1 illustrated in Figure 5.9 and SDC-like modifications to obtain the results listed in Table VIII. Then, we slightly modified Method 2 to make the inputs to the multiplexer symmetric, exploiting the ODC Condition on the camouflaged OR gate at one input to the 2x1 multiplexer (Figure 5.10 uses a camouflaged AND gate for simi;ar results). The results are reported in Table IX. The original combinational benchmarks from ISCAS-85 family are reproduced here along with their basic design characteristics:

Table VII: ISCAS-85 Benchmark Circuits

Original Circuit							
Benchmark	Gate Count	I/O	Nodes	Edges	Area	Delay	Levels
C17	6	5/2	6	12	14.4	0.56	3
C432	160	36/7	183	366	456	3.94	23
C432-2	160	36/7	183	366	456	3.94	23
C499	202	41/32	454	986	1164.8	3.10	19
C880	383	60/26	305	615	797.6	3.19	20
C1355	546	41/32	454	986	1164.8	3.10	19
C1908	880	33/25	398	818	1068	4.73	29
C2670	1193	233/140	648	1300	1672.8	2.74	19
C3540	1669	50/22	855	1967	2357.6	4.91	31
C5315	2307	178/123	1317	2782	3549.6	5.82	35
C6288	2416	32/32	2303	5247	6210.4	15.18	89
C7552	3512	207/108	1626	3374	4653.6	4.76	28

The results for the ISCAS-85 Benchmarks when Obfuscation Method 1 (Figure 5.9) is combined with SDC and SDC-like modifications are given below in Table VIII. The table shows that the average area overhead is only about 5.52%, which is acceptable. The average delay overhead for the benchmarks is 8% which is also feasible.

Table VIII: Results of Observability and Satisfiability Don't Care-based Obfuscation methodology applied on ISCAS-85 benchmarks (Method 1 combined with SDC modifications)

Obfuscated Circuit								
Benchmark	No. of Valid Function Locations Considered	Maximum RE Complexity	Percentage of Gates Obfuscated	Area Increase	Delay Increase	Levels	Area Overhead	Delay Overhead
C17	4	65536	67%	16	0.65	4	11.11%	16.07%
C432	5	1048576	3.13%	480.80	4.29	25	5.44%	8.88%
C432-2	9	1048576	5.63%	509.60	4.38	26	11.75%	11.17%
C499	16	1.84467E+19	7.92%	1207.20	3.53	22	3.64%	13.87%
C880	10	65536	3%	858.40	3.35	22	7.62%	5%
C1355	12	2.81475E+14	2.20%	1209.6	3.41	21	3.85%	10%
C1908	10	1.09951E+12	1.14%	1142.4	4.80	29	6.97%	1.48%

C2670	7	268435456	0.50%	1729.6	2.95	19	3.40%	8%
C3540	9	68719476736	0.54%	2435.2	5.33	32	3.30%	8.55%
C5315	12	2.81475E+14	0.52%	3608.80	5.82	35	1.67%	0%
C6288	9	68719476736	0.37%	6524.80	16.91	98	5%	11.4%
C7552	16	1.84467E+19	0.46%	4764.8	4.80	28	2.49%	1.48%

Next, we present results when a combination of method 2 (Figure 5.10) and SDC modifications are applied to ISCAS-85 Benchmarks. The average area overhead in this case (when the inputs to the 2x1 multiplexer are made symmetric for ODC-based modifications) is about 5.04%. The average delay overhead for the benchmarks is 7.12% which are both well within are specified overhead range of 10-15%.

Table IX: Results of Observability and Satisfiability Don't Care-based Obfuscation methodology applied on ISCAS-85 benchmarks (Method 2 combined with SDC modifications)

Obfuscated Circuit								
Benchmark	No. of Valid Function Locations Considered	Maximum RE Complexity	Percentage of Gates Obfuscated	Area Increase	Delay Increase	Levels	Area Overhead	Delay Overhead
C17	4	1048576	67%	17.6	0.71	3	22.22%	26.80%
C432	5	268435456	3.13%	472	4.28	25	3.51%	8.63%
C432-2	9	4.5036E+15	5.63%	475.20	4.18	25	4.21%	6.09%
C499	16	4.95176E+27	7.92%	1209.60	3.45	22	3.85%	11.30%
C880	10	2.81475E+14	3%	856.80	3.35	22	7.42%	5%
C1355	14	3.24519E+32	2.56%	1192.80	3.41	22	2.40%	10%
C1908	11	4.72237E+21	1.25%	1116.80	4.87	29	4.57%	2.96%
C2670	7	4.5036E+15	0.60%	1823.20	2.76	19	9%	1%
C3540	9	2.95148E+20	0.54%	2380.80	5.34	33	0.98%	8.76%
C5315	12	3.09485E+26	0.52%	3590.40	5.82	35	1.15%	0%
C6288	9	1.84467E+19	0.37%	6220	15.93	94	0.155%	4.94%
C7552	15	7.92282E+28	0.43%	4710.40	4.76	28	1.22%	0%

The maximum RE complexity we were able to achieve for the given number of valid functions considered are given in Table VIII and Table IX along with the corresponding design overhead. Again, since we are interested in determining the worst-case complexity of reverse engineering that

obfuscation offered, we only considered valid function locations in the largest intersection cone of the primary outputs. It is indeed possible to obfuscate additional gates in other intersections at a higher design overhead. However, the delay overhead (and also the area overhead) might be large as additional gates from other intersections are camouflaged. This is seen again in case of C432 in row 2 and row 3 of Table VIII (C432 has 5 gates in the largest intersection cone. C432-2 represents C432 when four additional gates from an intersection cone other than the largest intersection cone are camouflaged), we see that obfuscation of just 4 more gates doubles the area overhead and also significantly increases the delay overhead. Since these 4 additional gates belong to a different intersection cone, the complexity of Reverse Engineering is O ($16^5 + 16^4$) which is approximately equal to 16^5, where we assume that all of these 'valid' gates are camouflaged using 4x1 multiplexers. That is why we only consider the complexity associated with the largest intersection cone for these benchmarks, as explained earlier for SDC-based obfuscation approach. Therefore, it is possible to obfuscate additional gates from other cones at higher design overheads. Care should be exercised in this case so that the functionality of these gates is not revealed by justification and sensitization i.e. by controlling signal values in the circuit to known values.

As seen in Table VIII and Table IX, the RE complexity is rather low for some smaller circuits, even at relatively large area overhead. This is true for circuits with few valid function locations. Despite this, our obfuscation method combining layout obfuscation (i.e. directly replacing a candidate gate with a 4x1 multiplexer that implements the same function) with ODC and SDC-based obfuscation methods works remarkably well and appears to be scalable to larger designs. It can produce sufficiently high RE complexity even if just the largest intersection cone is camouflaged. Moreover, most practical circuits usually contain a larger number of gates.

The design overhead depends entirely on the size of the original circuit and the percentage of gates obfuscated (especially important for area overhead), and whether these gates lie on the critical path (important for delay overhead). The delay overhead also depends on what functions derive their input from the camouflaged valid function. Obviously, if the camouflaged function itself lies on the critical path, then connecting it to other functions as their inputs would affect the overall delay. If the obfuscated gates lie on the critical path, such as for C432 with 9 gates obfuscated, or in the case of benchmarks C3540, C1355 and C6288, the increase in overhead is significant. If, on the other hand, the obfuscated gates do not lie on the critical path such as for benchmarks C5315 (zero delay overhead and no change in the total number of levels in the obfuscated circuit, see column 7 in Table VIII and column 8 in Table IX) the RE complexity is relatively high even for low overheads. This is because the maximum RE complexity is dependent on the number of gates in the largest intersection cone, it does not depend on the percentage of total gates obfuscated. C880 represents an interesting case where the maximum RE complexity is only 16^4 because the maximum intersecting cone has 4 gates only. However, six more gates from other cones (3 gates from each cone) are also obfuscated, so the total number of gates obfuscated is 10. But the maximum RE complexity is still approximately16^4 (16^4 +16^3 +16^3) for the three cones considered. This idea can be extended to larger, more complicated IC's, and works to the advantage of the security-aware circuit designer. For C7552, not all valid function locations were camouflaged as camouflaging just 18 of those functions gave sufficiently high RE complexity.

The only concern that can be raised with this approach is that it tends to concentrate all the obfuscation together in (possibly) one part of the circuit, while the rest of the circuit characteristics are known. To overcome this problem, we note that it is indeed possible to target other intersection cones elsewhere in the circuit to make the exact layout harder to comprehend (while making sure

that they do not lie on the critical path so that the design overhead is acceptable). Our algorithm successfully picks out all possible obfuscatable locations, and it is possible to achieve even greater RE complexity than reported by merely picking functions from other than the largest intersection cone (at higher design overhead). However, the maximum RE complexity will still be determined by the largest intersection cone and therefore we are concerned with this parameter in this analysis.

Finally, we note that the runtime complexity goes up exponentially with the size of the circuit. This explains the fact that we have runtimes in the range of seconds for smaller circuits and several minutes for larger circuits.

It is also worth repeating that the delay overhead largely depends on whether the obfuscated gates lie on the critical path and the fan-out of the camouflaged gates while the area overhead depends on the size of the original circuit and the percentage of gates obfuscated. Both area and delay overheads are comparable (though slightly smaller) than the corresponding area and delay overheads for SDC-based obfuscation. This shows that our approach is indeed scalable and can be successfully implemented for larger, practical circuits.

For small circuits, the area overhead is expected to be significant. This is because we are substituting standard logic gates, such as AND, NAND, etc., with multiplexers which are significantly larger and slower. As stated above, the overhead will be mitigated by the size of the circuit. In Figure 4.6, we can see a general downward trend for area overhead for circuits as the size of the circuits gets larger. The delay overhead is largely determined by whether the valid function locations are on the critical path (this critical path might change as we utilize different don't care conditions to obfuscate the circuit). Note that, following the approach in [15] and [21] we could have utilized a performance-driven method that aims to achieve a trade-off between percentage of gates obfuscated and the resulting overhead. We see that our algorithm can help us identify all

possible valid functions and so this method is implementable with our algorithm, but we chose to focus on the maximum RE complexity as that appears to be a more useful metric in secure design.

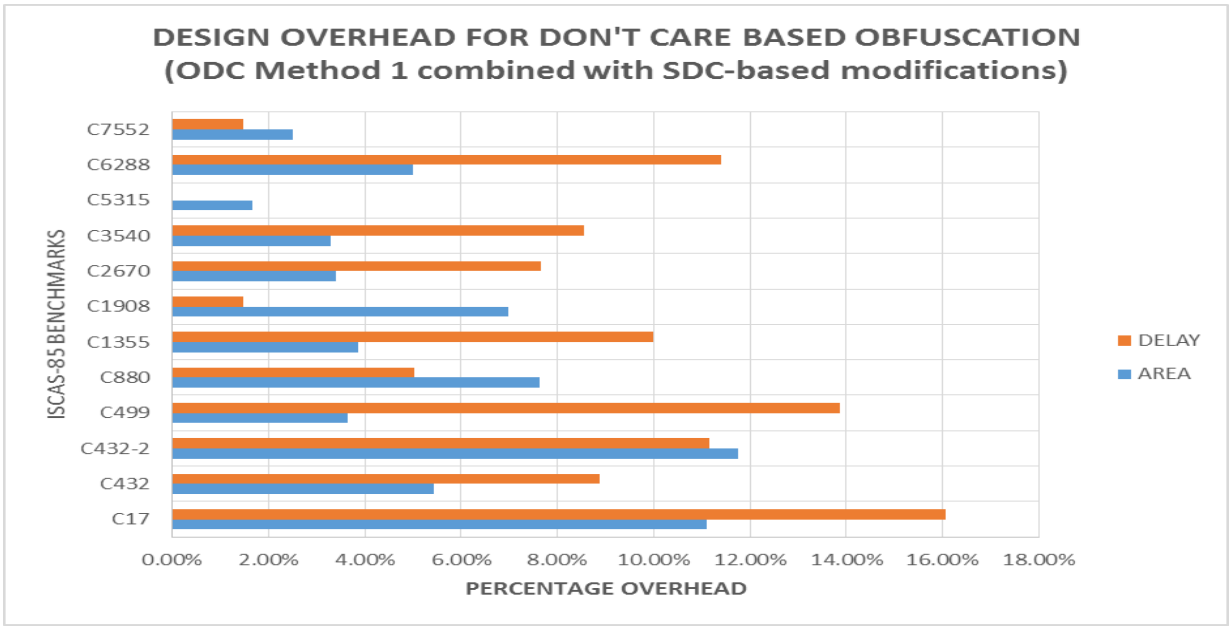


Figure 5.12: Don't Care-based Obfuscation (Method 1 combined with SDC modifications): Overhead results for ISCAS-85 Benchmark Family. C432-2 represents C432 benchmark with two intersection cones camouflaged (total number of obfuscated valid functions: 9).

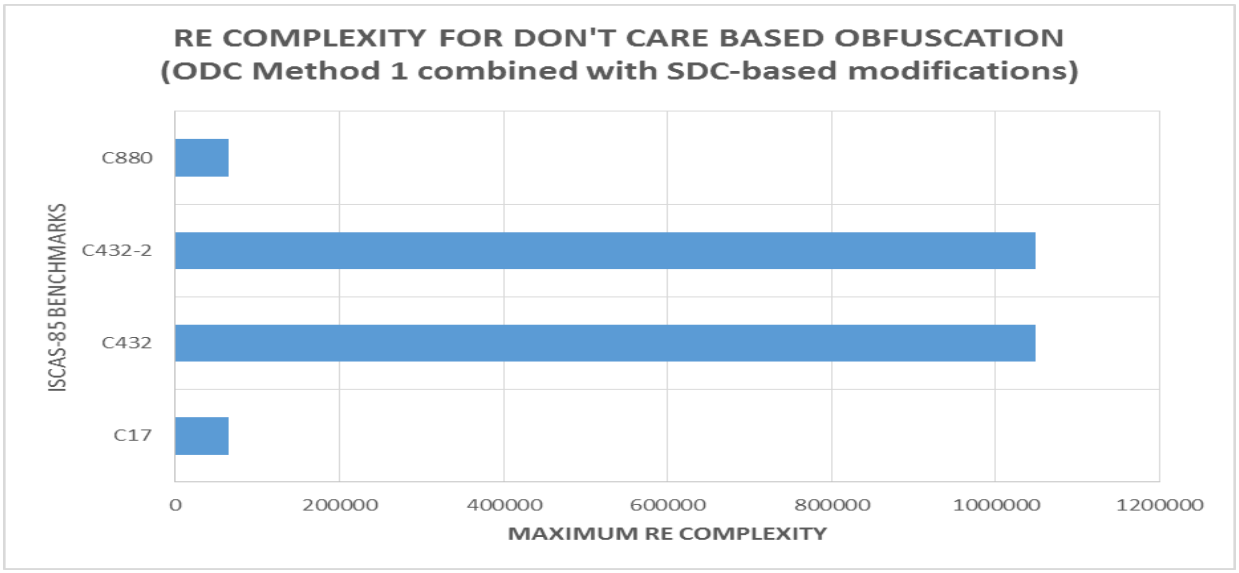


Figure 5.13: RE Complexity for some selected benchmarks (using a combination of Method 1 combined with SDC modifications)

C432-2 represents C432 benchmark with two intersection cones camouflaged (sizes 5 and 4 respectively). The maximum RE complexity in this case is still 16^5 where 5 is the size of the largest intersection cone.

Next, we graphically show the results for when SDC-like modifications which are always true but camouflaged are combined with ODC Method 2 (where the 2x1 MUX is set up to have symmetric inputs by using a camouflaged Buffer and an AND gate). The results indicate that area and delay overheads are well within the acceptable range. The RE Complexity in this case is higher than using a combination of Methods 1 and SDC shown in Figure 5.13. This is because we consider more valid function locations from the largest intersection cone for some circuits such as C1355, essentially raising the RE Complexity. This can be seen in column 3 of Table IX. The conclusions drawn from these results are similar: the area overhead is largely determined by the percentage of gates obfuscated and the size of the original circuit, while the delay overhead depends on whether the obfuscated gates lie on the critical path, as well as the fan-out of the obfuscated gates. Again, we observe that smaller circuits with relatively few 'valid function locations' have relatively low RE complexity, despite high area overhead.

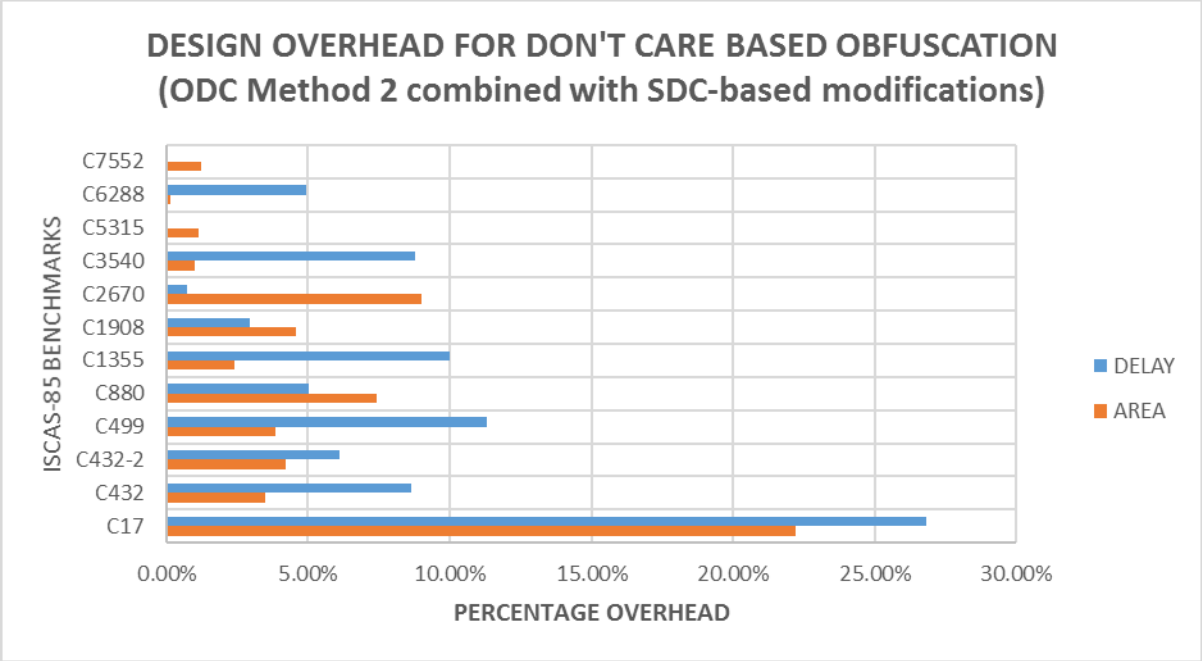


Figure 5.14: Don't Care-based Obfuscation (Method 2 combined with SDC modifications): Overhead results for ISCAS-85 Benchmark Family. C432-2 represents C432 benchmark with two intersection cones camouflaged (total number of obfuscated valid functions: 9).

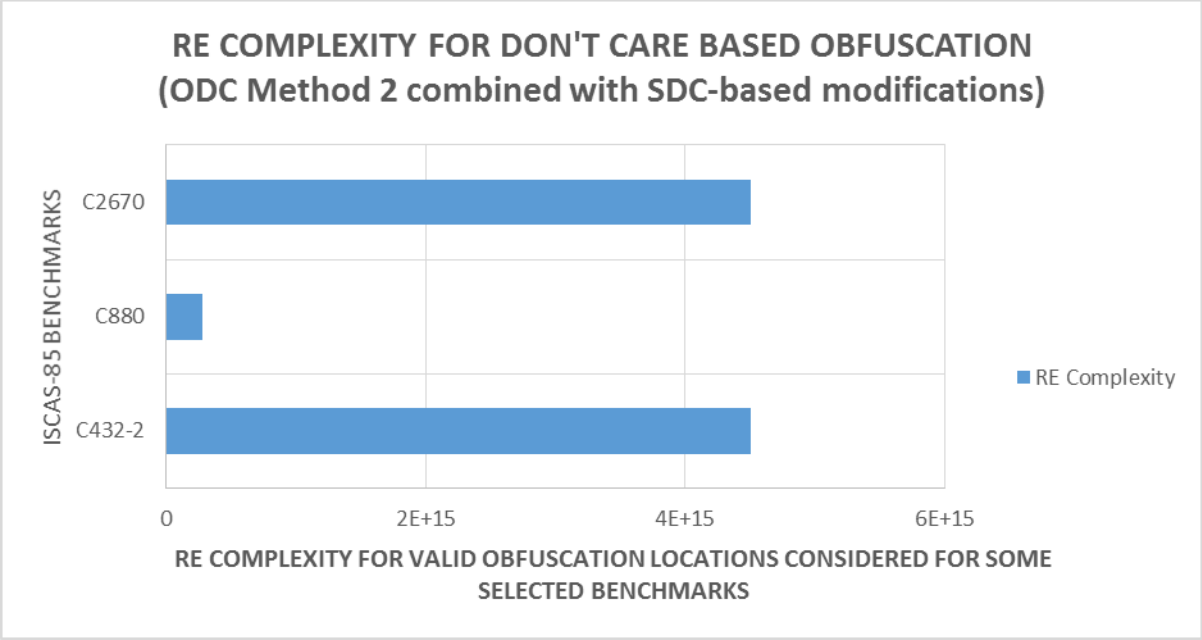


Figure 5.15: RE Complexity for some selected benchmarks (using a combination of Method 2 combined with SDC modifications)

Note that RE Complexity for most practical circuits will be equal to or greater than these values.

5.6 Functional Obfuscation for Incompletely Specified Circuits

Function obfuscation or logic obfuscation is conceptually different from all the circuit obfuscation methods discussed earlier in literature. In all the existing circuit obfuscation approaches, the objective is to introduce "confusion" in the layout to increase the complexity of RE attacks, while preserving the circuit's correct functionality. This gives RE attackers hope to discover the true functionality. The idea behind our functional obfuscation is to modify (through gate camouflaging) the functionality at the sub-circuit level while ensuring that the overall specification is met. Since most practical circuits are incompletely specified, that is, not all primary inputs and states are utilized, it is possible to employ these unused inputs to modify the sub-circuit function in such a manner that on these inputs, the outputs will not follow the definition of the function that the circuit is supposed to implement. Therefore, even with the complete information of high-level circuit description (Goal-1), an adversary can never retrieve the correct functionality because what the circuit implements is a "different" functionality. Meanwhile, Goal-2 of exactly retrieving the circuit layout is thwarted by gate camouflaging.

Formally speaking, suppose that we want to implement a function f: X→ Y that maps each element $x \epsilon X$ to some $y=f(x) \epsilon Y$. When we represent elements in X and Y in binary (or digital), let DX and DY be the sets of all the possible values, in addition to X and Y respectively, that can be represented using the same convention. We refer to DX and DY as the digitized domain of X and the digitized image of Y, respectively. Clearly we have $X \epsilon DX$ and $Y \epsilon DY$. In our function obfuscation approach, we will map some $x \epsilon DX\backslash X$ (that is, $x \epsilon DX$ but $x \notin X$) to $y \epsilon DY$ such that $y \neq f(x)$. Suppose, for example; the implementation of two-input multiplication table for input range of 0 to 9. So, for this multiplication table, we have $X=\{(x1,x2):0 \leq (x1,x2) \leq 9\}$ and $Y=\{0,1,2,\ldots,10,12,14,15,16,18,20, \ldots 81\}$. When the standard binary representation is used, there

will be 8 input lines (4 bits to represent each of the x1 and x2) and 7 bits for output (which can be as large as 81). So we have the digitized domain DX= {(x1,x2): $0 \leq (x1,x2) \leq 15$} and the digitized image DY={0,1,…,127}. Now if we map and implement, for example, (10, 12) to 53, it will be impossible for an adversary to discover that the f(x1,x2)= x1*x2. The above example takes advantage of the Don't Care input vectors for functional obfuscation. It is also possible to utilize the intrinsic don't care conditions to implement the correct functionality as desired but some "misleadingly incorrect" functionality on those don't care conditions simultaneously to confuse the RE adversaries. Observability and Satisfiability don't cares can be used to nullify this intentionally incorrect function (and camouflage it) somewhere else in the circuit.

We present two examples of functional obfuscation using ISCAS-85 benchmarks C2670 and C5315. These examples are presented here as proof of concept of functional obfuscation. As part of future work, it is suggested to take this approach further by considering the implications for large, complex circuits in terms of design overhead and RE complexity. Functional obfuscation can also be used in conjunction with layout obfuscation (with or without using don't cares) and authentication-based approaches.

5.6.1 Example 1: Functional Obfuscation of C2670 Benchmark

Statistics: 233 inputs; 140 outputs; 1193 gates

Function: 12-bit ALU and controller

This benchmark consists of an ALU with a comparator, an equality checker, and several parity trees. The comparator has two 12-bit inputs X and Y, and computes Y>X using a carry look-ahead adder (CLA) that performs the addition function !X+Y. It can be programmed to do a 4, 6, 8 or 12-bit comparison of its inputs. An interesting feature of the comparator is that it uses two identical CLAs that have identical inputs, a redundancy technique commonly used in fault-tolerant systems. The

CLAs have a fairly standard structure with 3, 4 and 5-bit blocks. The carry output signal of each CLA gives the result of (Y>X). The output labeled *OutYgreaterX_Equal* (line number 231) is constant 1 if the outputs of the two CLAs are identical, as would normally be the case. If, however, the CLAs produced different results, the*OutYgreaterX_Equal* output would be **logic 0**, implying an error in the circuit. This would happen, for example, if there were manufacturing defects in one of the CLAs.

Module M7 *(EqualZ_W)* performs an equality check on two 17-bit buses. The *ParityChecker* module (M8) contains five 10-input parity trees, whose outputs are all ANDed. This module performs a sanity check on the input buses of c2670. There are also several other small pieces of logic.

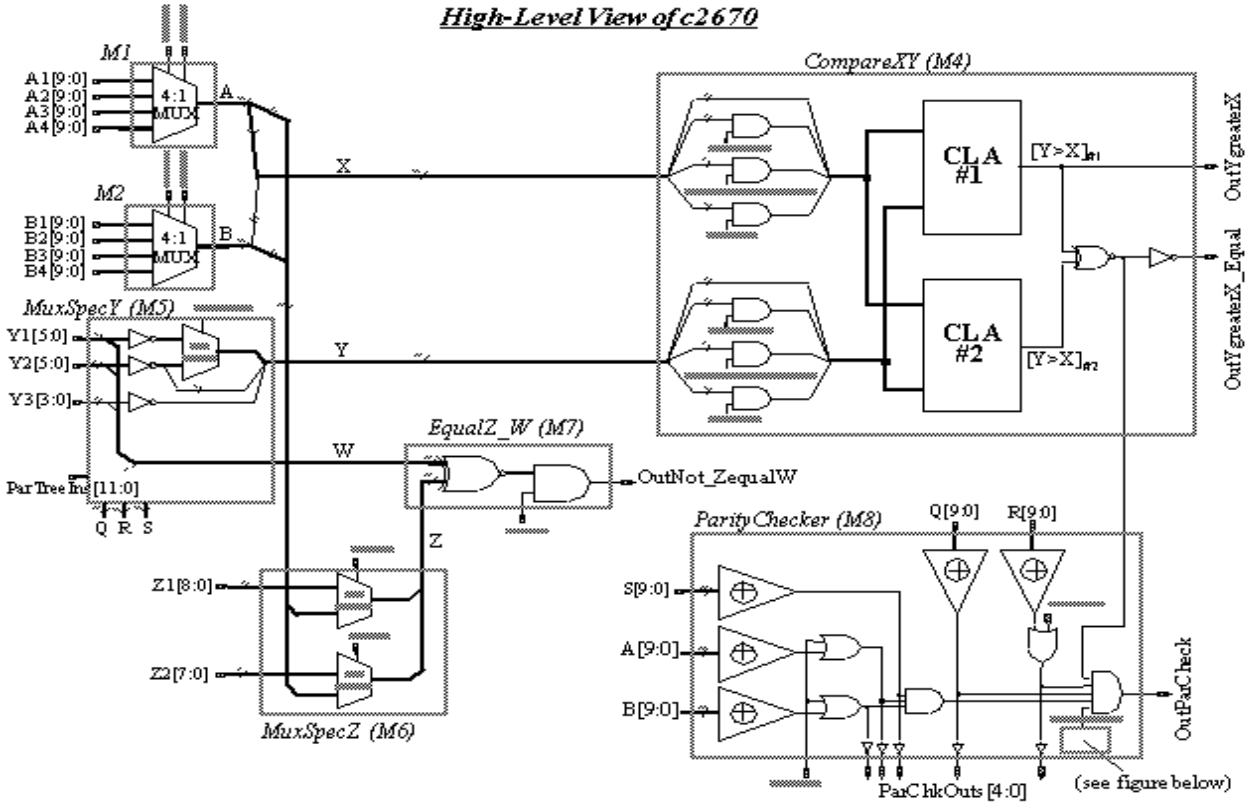


Figure 5.16: High-level Model of C2670

As in most practical circuits, there are several unused inputs. This becomes evident when the input buses are represented in binary. We utilize the unused inputs of buses Y1, Y2, Y3, A and B to modify the sub-circuit functionality while meeting the overall circuit specification. This is done by adding functionality that utilizes both the new and some useful inputs but the effect is nullified by using don't care conditions later in the circuit. We can also make use of valid circuit conditions such as the fact that the CLA comparison wire 231 is always 1 when the two CLA's compute the correct and identical result. OR-ing this with any intentionally incorrect function output will always give output of 1 for the valid set of inputs. Similarly, we can recognize other circuit conditions and use them for functional obfuscation. Additionally, we recognize the largest intersection cone of the circuit and use don't cares to obfuscate the circuit layout as before. In this way, the attacker's attempts to achieve Goal 1 and Goal 2 expend more effort as both the high-level circuit description and the gate-level layout is modified. For example, by using just 17 additional (previously unused) inputs, the complexity of reverse engineering the high-level description is increased by 2^{17} while the RE complexity of determining the exact layout description is 16^8 (because 8 valid function locations are considered for layout obfuscation using don't cares). Additionally, the functional description itself is obfuscated as we are not only concealing but modifying the sub-circuit functionality. This is achieved at acceptable design overheads for these cases.

Table X: Functional Obfuscation of some selected Benchmarks from ISCAS-85 family (a) Original Circuit (b) Obfuscated Circuit

Original Circuit							
Benchmark	**Gate Count**	**I/O**	**Nodes**	**Edges**	**Area**	**Delay**	**Levels**
C2670	**1193**	**233/140**	**649**	**1301**	**1674.40**	**2.74**	**19**
C5315	**2307**	**178/123**	**1317**	**2782**	**3549.6**	**5.82**	**35**

Benchmark	No. of Valid Function Locations Considered for layout obfuscation	Maximum RE Complexity	Number of Unused Inputs Utilized	Increase in Complexity for Goal 1 (High-level Input/Output relationships)	Area Increase	Delay Increase	Levels	Area Overhead	Delay Overhead
C2670	**8**	**4294967296**	**17**	**2^{17}**	**1804.80**	**2.94**	**19**	**7.8%**	**7.3%**
C5315	**11**	**1.7592186e+13**	**35**	**2^{35}**	**3680.80**	**5.82**	**35**	**3.7%**	**0%**

Table X also shows results of functional obfuscation of combinational circuit C5315 from the ISCAS-85 family is discussed below.

5.6.2 Example 2: Functional Obfuscation of C5315

Statistics: 178 inputs; 123 outputs; 2406 gates

Function: 9-bit ALU

This benchmark is an ALU that performs arithmetic and logic operations simultaneously on two 9-bit input data words, and also computes the parity of the results. When the inputs are represented in binary form for these 9-bit data words, we realize that there are several unused inputs. Modules M6 and M7 each compute an arithmetic or logic operation specified by the control input bus CF[7:0]. Module M5 consists of multiplexers that route the results of M6 and M7 and four input buses to its four outputs. Output buses OF1 and OF2 can also be set to logic 0 by MuxSel[8]. Modules M3 and M4 compute the parity of the result of the operation given by CP=CF[7:4]. Module M5 contains four multiplexers which direct the parity results and an additional set of four inputs to its outputs. The adders in M6 and M7 as well as the parity logic for the arithmetic operations in M3 and M4 use a carry-select scheme with 4-bit (low-order) and 5-bit (high-order) blocks. The circuit also includes logic for calculating various zero and parity flags of the input buses.

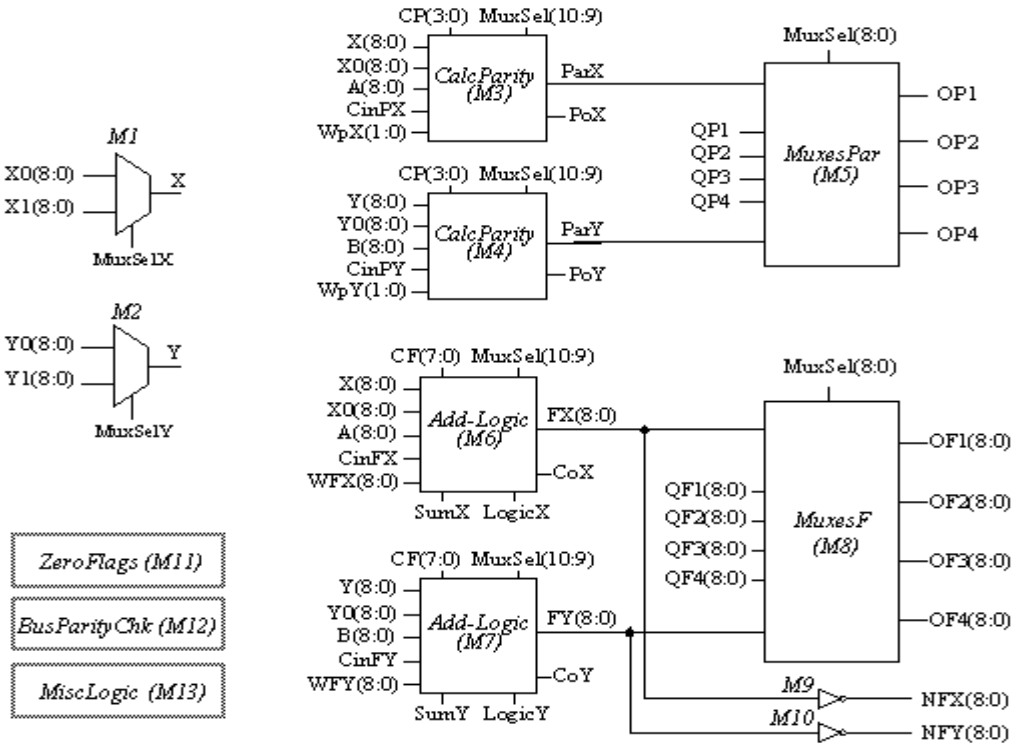


Figure 5.17: High-level Model of C5315 (abstracted out to show one part of the whole circuit description)

Again, we utilize the unused inputs of buses X0, X1, Y0 and Y1 to modify the sub-circuit functionality while meeting the overall circuit specification. This is done by adding functionality that utilizes both the new and some useful inputs but the effect is nullified by using don't care conditions later in the circuit. We can also make use of valid circuit conditions such as constant logic 0 or logic 1 signals within the circuit for functional obfuscation purposes (because the effect of the added functionality has to be nullified). The don't care conditions can also be utilized to simultaneously obfuscate the circuit layout as we described in chapters 4 and 5, using Observability and Satisfiability Don't Cares. For this, we recognize the largest intersection cone of the circuit and use don't cares to obfuscate the circuit layout as before. In this way, the attacker's attempts to achieve Goal 1 and Goal 2 expend more effort as both the high-level circuit description and the gate-level layout is modified. For example, by using just 35 additional inputs for C5315, the complexity

of reverse engineering the high-level description is increased by 2^{35} while the RE complexity of determining the exact layout description is 16^{11} (because 11 valid function locations are considered for layout obfuscation). Additionally, the functional description itself is concealed and modified. This is achieved at acceptable design overheads for the cases considered, as reported in Table X.

We believe that this can be a very promising obfuscation tool, the only caveat is to implement the method in a manner that minimizes the design overhead and maximizes the RE efforts of the attackers. Meanwhile, it will be interesting to study whether there will be any counter-countermeasures for the attackers.

5.7 Conclusion

This work has shown that we can create a robust obfuscation scheme based on don't care conditions in the circuit that guarantees a high RE complexity at acceptable performance overhead.

There is a potential for future work that compares the different methods of ODC obfuscation recommended in section 5.2.1 since we only implemented a combination of methods 1 and 2. Potential methods include using fuses as the connections for the added lines for all possible ODC locations so we can decide which ones are active, and be able to add connections (such as for the deliberately misleading functions) after fabricating the rest of the circuit or reroute lines in post-production in order to come up with a number of different secure designs.

The major result of this work, is the determination of obfuscatable locations based on RE complexity. It should be noted that as is possible for the SDC-based obfuscation method, it is possible to add more obfuscated gates from cones other than the largest intersection cone. We concentrated on the largest intersection cone to be able to determine the maximum RE complexity. Similarly, it is possible to adopt a performance or delay-driven approach in deciding on the number

of obfuscated locations in the final circuit. The circuit size, the percentage of gates obfuscated, and whether the obfuscated gates lie on the critical path, as well as the fan-out of the camouflaged gates contribute greatly to the area and delay overhead in the modified circuit.

The summary of simulation results obtained for the different don't care-based obfuscation methods is given below in Table XI:

Table XI: Summary of Simulation Results of different Don't Care-based Obfuscation Methods applied to the ISCAS-85 Benchmarks

Obfuscation Method	**Average Area Overhead**	**Average Delay overhead**
(i) Purely SDC-based Obfuscation	5.945%	8.077%
(ii) ODC-based Obfuscation Method 1 combined with SDC-based Obfuscation	5.52%	8%
(iii) ODC-based Obfuscation Method 2 combined with SDC-based Obfuscation	5.04%	7.12%
(iv) Functional Obfuscation (for two selected benchmarks	Average Area expected to be almost double the values obtained for ODC Method 1(obtained in (ii)). The delay overhead s expected to be similar to (ii)	

"In order to carry a positive action we must develop a positive vision."

Dalai Lama

CHAPTER 6: NETLIST DEBUGGING AND VIEWING USING GATEVISION PRO

GateVision PRO is the third generation of graphical gate-level netlist analyzers and netlist viewers from Concept Engineering, based in Freiburg, Germany. GateVision PRO provides the designer of even the largest chips and SoCs with intuitive design navigation, netlist viewing, waveform viewing, logic cone extraction, interactive logic cone viewing for netlist debugging and design documentation. As such, GateVision PRO provides a useful complement to our C++ algorithm that translates the netlist into a directed graph for easy circuit manipulation and cone extraction. This chapter discusses the use of the tool to visualize the netlist, extract logic cones and visualize and compare results obtained from our C++ algorithm. The main features of GateVision PRO are briefly explained below:

GateVision PRO has an ultra-fast gate-level netlist debugger and netlist viewer netlist viewer that can accept a variety of formats including EDIF, bench, Spice netlists and standard Verilog files. It can process even the largest Verilog netlists, EDIF netlists and LEF/DEF netlists. By reading Verilog, EDIF and LEF/DEF netlists, GateVision PRO fits seamlessly into any design environment. Schematics are generated on the fly and the intuitive GUI lets the designer incrementally and easily navigate through the largest netlist files.

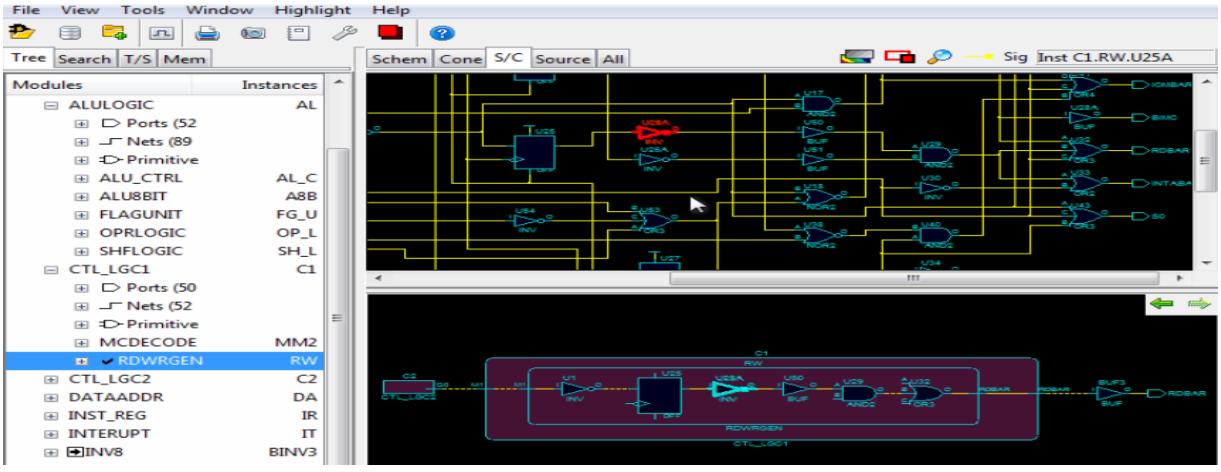


Figure 6.1 (a) : Gate-level description of an example circuit that has been converted from its verilog netlist. The Window at the bottom shows one of the output cones extracted using the cone extraction feature in GateVision PRO

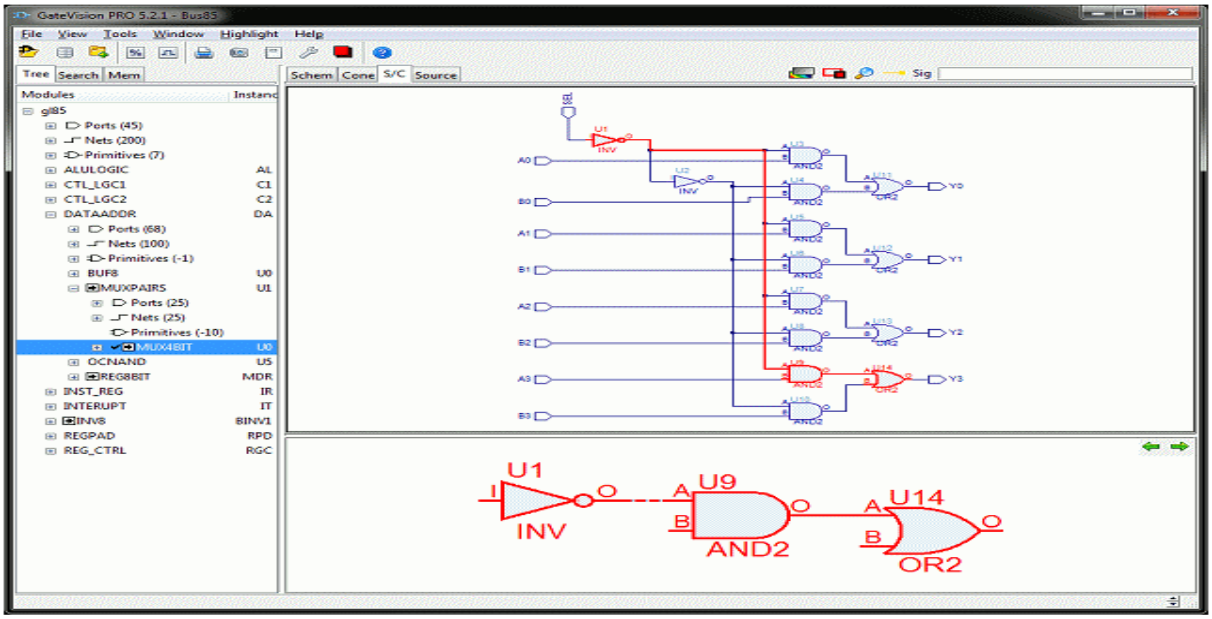


Figure 6.1(b): Another example of Schematic Capture using GateVision PRO; the netlist file is converted directly into gate-level description; the bottom window shows output cone extraction

6.1 Waveform Viewer and Signal Tracing

GateVision PRO comes with a fully integrated waveform viewer and with support for interactive signal tracing in the source code, schematic view and waveform window. GateVision PRO compiles simulation data into its own high-speed format for accelerated waveform browsing and signal tracing. This is useful for netlist debugging. This feature is similar to the waveform viewer that comes with the debugging tools in Xilinx IDE.

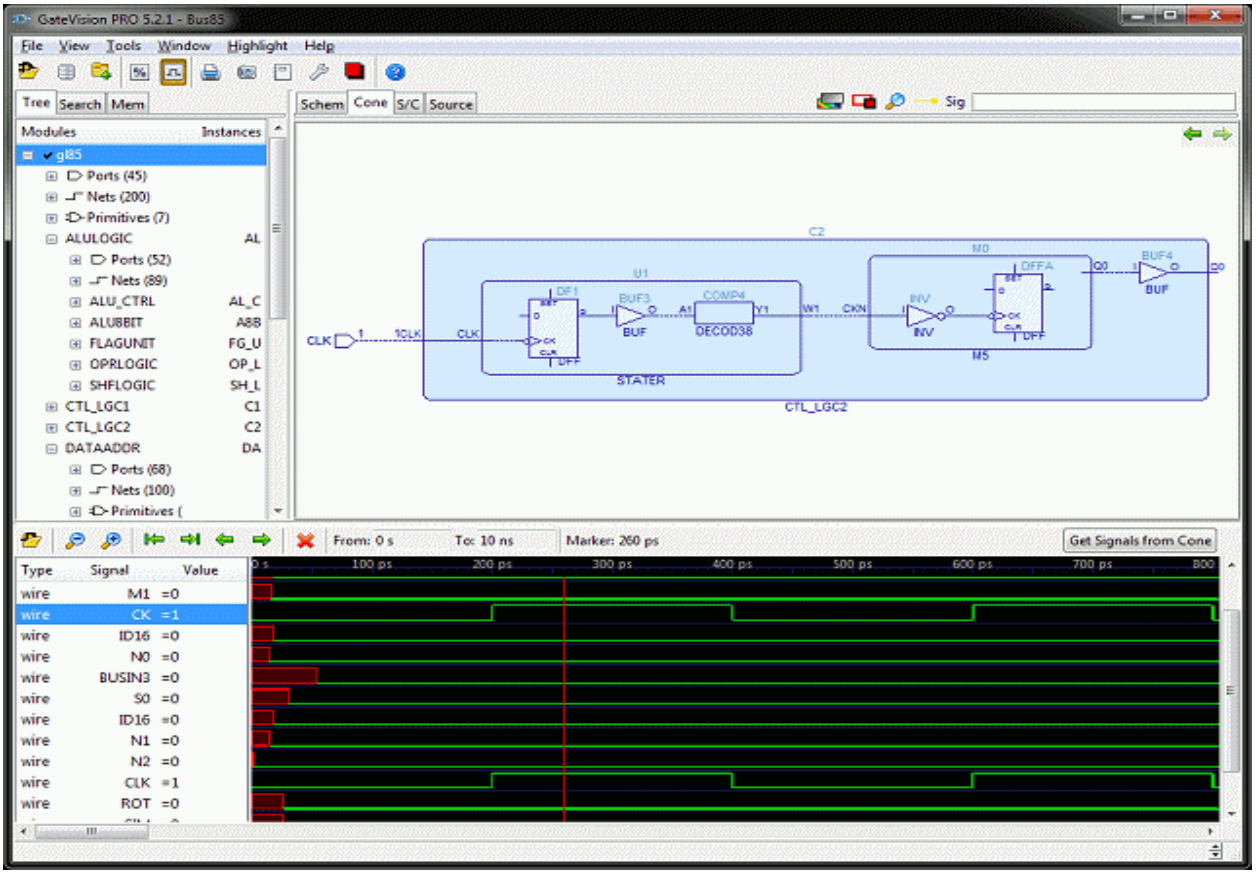


Figure 6.2: Waveform Viewing using GateVision PRO

6.2 Logic Cone Extraction Feature

The two features described in 6.1 together with the logic cone and path extraction feature made this tool very useful in visualizing and analyzing ISCAS-85 benchmark circuits for gathering our results based on don't care condition-based circuit obfuscation. The GateVision PRO logic cone view provides interactive navigation within a schematic fragment i.e. the portion of the circuit that is most relevant. This can be extended or reduced for signal path tracing through the complete design hierarchy.

6.3 Path extraction and Verilog Simulation

In addition to logic cone extraction feature, the customizable path extraction engine can automatically identify and extract critical paths in a design. These can be explored and cross-probed in different views to reduce both the complexity and time of the debug cycle or when picking appropriate gates to obfuscate. Path fragments can be exported as Verilog netlists for critical path Verilog simulation.

6.4 Debugging Views

Finally, built into GateVision Pro are a variety of view options, including schematic view, schematic fraction or cone view, source code view, hierarchy tree view, waveform view, clock domain view, and object search view. Through these, and through cross-probing between views, it is easy to gain a deeper understanding of the device being debugged and to improve the debugging process. It also helps in editing the source/ bench file during gate replacement step (replacing a logic gate with a 4x1 multiplexer that implements the same function as explained in section 3.4.4).

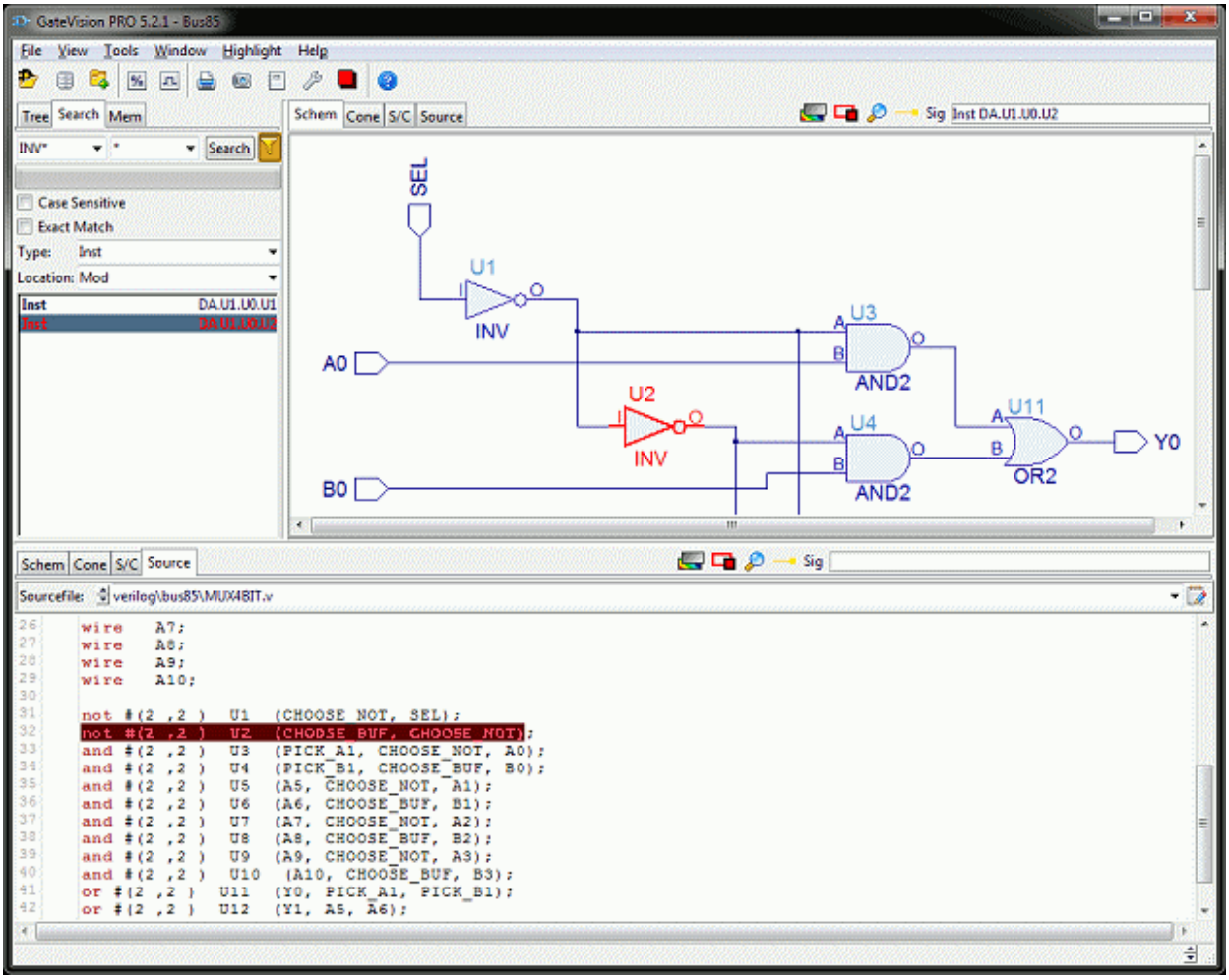


Figure 6.3: Source code and Schematic views (viewable side-by-side)

The following table lists the main features of GateVision PRO at a glance:

Table XII: GateVision PRO Features

Features	Benefits
Ultra fast netlist readers	Netlist to schematics on the fly (within seconds)
Integrated waveform viewer	For easy signal tracing and simulation results analysis (accelerated VCD viewer)
Automatic clock tree and clock domain extraction and visualization	Faster detection and resolution for clock domain problems
Con Window	Incremental schematic navigation for big designs
Verilog Netlist Export	Fragments of a circuit can be saved as Verilog netlist files
Tcl UserWare API	Allows interfacing with tool flow and definition of electrical rule checks
Netlist to schematics	Verilog viewer, EDIF viewer, and LEF/DEF viewer in one tool allows debugging of almost any netlist file format
Powerful GUI	Multiple views, including tree, schematic, waveform, cone and source file for increased circuit understanding plus drag-and-drop between different views.
Automatic path extraction	Automatically extracts logic cones from user-defined reference points, and shows only the relevant portion of the circuit, Reduces complexity in the design for improved and faster netlist debugging
Search-and-show capability	Easy location of specific objects shortens debug time
Design hierarchy browser	Provides easy navigation through the design hierarchy and gives compact hierarchy overview
Object cross-probing	Highlights selected objects in all design views (schematic, logic cone and

	HDL source code view) and shortens debug time
Context-sensitive menus	Easy-to-use GUI

The above description provides a brief overview of the important features of GateVision PRO. The tool served as a useful complement to our algorithm written in C++ that determines gates that lie in the intersection of two or more output cones. The results from the algorithm can be analyzed and compared with those obtained using the cone extraction feature in GateVision PRO. Moreover, GateVision PRO directly converts netlists from a variety of formats into gate-level circuit descriptions that provide greater visibility of the circuit and help with debugging and embedding security primitives.

7 CONCLUSION

Economic reasons as well as the complexity of modern IC design process dictate the participation of several external agents in modern IC design and manufacturing. IC design houses are becoming increasingly dependent on hardware IP modules and CAD software tools licensed from external vendors. Due to the spiraling cost of maintaining a fabrication facility, most semiconductor companies follow a “fabless” business model where the design is finalized and the IC is manufactured in offshore fabrication facilities by companies specializing in semiconductor manufacturing. These practices have reduced the level of control that IC vendors used to exercise over the ICs they design and manufacture. As a result, an IC can be maliciously modified at different stages of the design and manufacturing flow by the insertion of so-called “hardware Trojans", modification of the FSM to reach “safe states” and extraction of information from gate-

level netlist, masks and layout information. In addition, “clones" of an IC might be manufactured illegally in offshore fabrication facilities resulting in loss of market share for the IC design house. Similarly, “Design for Testability" practices provide new opportunities for the adversaries to discover secret information from secure ICs. Therefore, release of IPs to unscrupulous IC design houses makes the IP vendors vulnerable to IP piracy through illegal copying.

Obfuscation is a technique that makes comprehending and reverse-engineering a design difficult. Various obfuscation techniques have been proposed in the past to counter the IP piracy problem that aim at making it difficult to interpret the circuit layout by inserting key gates in the circuit (XOR/ XNOR gates that control correct functioning of the IC) or by replacing some selected gates with other gates that can be configured to perform one of several logic functions (called “gate camouflaging”). In this research, we have developed hardware obfuscation techniques based on the presence of don’t care conditions that exist in virtually every circuit of non-trivial size to implement layout obfuscation; we also propose proof of concept for functional obfuscation that effectively counters the threats to the IP from malicious users by not only concealing but also modifying sub-circuit functionality while meeting the overall circuit specification. Alongside, an enhanced IC design methodology is developed where the security aspect is seamlessly integrated with the traditional design steps through EDA software tool support to develop a piracy-proof SoC design methodology. The major contribution of this work is the creation of a robust obfuscation scheme based on don’t care conditions in the circuit that guarantees a high RE complexity at acceptable performance overhead. The most important aspect, from a security-aware designer’s point-of-view, is the determination of obfuscatable locations based on RE complexity. As already stated, it is possible to add more obfuscated gates from cones other than the largest intersection cone in order to achieve even greater RE complexity at higher design overhead. We concentrated on the largest

intersection cone to be able to determine the maximum RE complexity. Similarly, it is possible to adopt a performance or delay-driven approach in deciding on the number of obfuscated locations in the final circuit. The circuit size, the percentage of gates obfuscated, and whether the obfuscated gates lie on the critical path, as well as the fan-out of the camouflaged gates contribute greatly to the area and delay overhead in the modified circuit.

We believe that the proposed obfuscation procedure can be used to realize a low-cost and robust obfuscation feature by introducing appropriate modification in the circuit layout or sub-circuit functionality (while meeting the overall circuit specification). Similarly, for incompletely specified circuits, certain input combinations that remain unused can be utilized for obfuscation purposes. A quick comparison of our proposed obfuscation technique with the existing approaches found in literature is given below:

7.1 Comparison of Don't Care-based Obfuscation with Existing Hardware IP Security Solutions

1. **HARPOON: An Obfuscation-based SoC Design Methodology for Hardware Protection [5]:**

In this approach, the sub-circuit functionality of the circuit is intentionally modified by introducing FSMs into the design to discourage simulation-based reverse engineering or structural analysis of the netlist. The approach targets to achieve simulation mismatch for the maximum possible input vectors, as well as structural mismatch for maximum possible circuit nodes. To achieve structural mismatch, the state transition function or internal logic structure of ISCAS-89 sequential circuits is modified by introducing an incorrect function 'g' and an enable signal 'en' corresponding to each randomly selected correct function 'f'. Note that this approach randomly chooses the nodes to

obfuscate, but does not guarantee that the nodes are interfering. Additionally, the authors employ a performance-driven approach where they continue to obfuscate randomly-chosen gates until a certain overhead constraint is met. The effectiveness of the algorithm is measured in terms of an "obfuscation metric" or "output corruptibility metric" that computes the number of failing input patterns. Although we do not formally compute output corruptibility for our functional obfuscation method but instead focus on maximizing the RE complexity for ISCAS-85 combinational benchmarks, the RE complexity for Goal-2 is expected to be higher for our approach because of gate replacement with 4x1 multiplexers. The number of failing patterns for Goal-1 is also expected to be comparable or higher because of the presence of configurable 4x1 multiplexers for the same overhead constraint. The delay overhead is expected to be slightly higher for our case, because of non-randomly chosen camouflaged gates.

2. Security Analysis of Logic Obfuscation [21]

This methodology hides the functionality of a design by inserting additional gates into the original design in the form of XOR/ XNOR gates for embedding '0' and '1' into the circuit respectively. In order for the design to exhibit correct functionality, a valid key has to be supplied to the key-gates which are classified as 'isolated', 'dominating' and 'convergent' key gates that can be resolved sequentially or concurrently. By considering all the different kinds of interference between key-gates, the attacker tries to find a golden pattern that simultaneously mutes one key while justifying the other key to the primary output and vice versa (section 2.1.2). Strong logic obfuscation hinges on inserting key gates with complex interferences among them. This is done by constructing interference graphs. However, interference graphs do not rule out observing each key gate output by observing just the relevant output logic cone in which it lies. Therefore, there are some weaknesses to the approach as described. The authors analyzed their obfuscation technique using ISCAS-85

Benchmarks by selecting key size as 5% of the gates in the circuit, under the assumption that a golden pattern always exists. For every brute force attempt, 1000 random patterns were applied to determine the value of the key. The effectiveness of four types of insertions was analyzed: random insertion, random insertion with no runs of gates (runs of gates are key-gates occurring together, in a chain), unweighted insertion and weighted insertion where non-mutable gates were given a higher weight of '2' than mutable gates (weight =1). When an attacker tries to attack, not all golden patterns will exist; therefore, from an attacker's perspective, the effective key-size is the largest key size on which brute force is attempted. If the number of brute force attempts is 2^M, then the effective key size is M. The weighted insertion method gives the largest effective key size, which is smaller than the key size in our functional approach which utilizes unused primary inputs and don't cares for simultaneously obfuscating the layout and high-level functionality. The area overhead is more than double the area overhead obtained for our approach because 5% of gates is always obfuscated even if it results in redundancy because the interference graphs can be broken with respect to the output logic cones. Therefore, the RE complexity reported for this key-insertion approach would be lower than reported.

3. Security Analysis of Integrated Circuit Camouflaging [15]

Camouflaging is a layout level technique that hampers an attacker from reverse engineering by introducing dummy contacts into the layout (section 2.1.2, Figure 2.13). Since our layout level obfuscation uses a 4x1 multiplexer for camouflaging, our approach offers a larger pool of possible valid functions (16 in total, i.e. all possible 2-input, 1-output logic functions) compared to this approach which offers choice of 3 possible functions (XOR/NAND/NOR), therefore the RE complexity offered is higher with our approach for the same percentage of gates obfuscated. The approach, as described in [15] uses interference graphs to form cliques of interfering nodes but they

overlook the fact that some of these key-gates can be resolved by concentrating on just the output logic cone in which they lie. A comparison of the effective number of camouflaged gates reveals that our approach offers far greater resistance to reverse engineering. Moreover, the gates chosen using our don't care based algorithm are always non-resolvable because they lie in the same intersection cone. The area and delay overheads for our approach are also significantly lower. The area overhead for C7552 AND C5315 using [15] is about 60% while the delay overhead is about 25-30%. In comparison, for our methods, the average area overhead is 5.502 % while the average delay overhead for all methods presented is 7.73 %.

4. EPIC: Ending Piracy of Integrated Circuits proposed in [6]

Similarly, the public-key cryptography and combinational locking method proposed in the EPIC approach [6] recommends a key length of 64 bits. Note that this approach does not impose any restriction on the choice of key-gates; therefore using isolated key gates would significantly lower RE complexity. Comparing EPIC to our approach only requires 16 gates from the intersection cone to be camouflaged for the same number of brute force attempts. This is indeed possible for most practical circuits. The RE Complexity for Goal-2 would again be higher using our approach due to gate camouflaging which is not used in EPIC. Moreover, for the EPIC approach, it is possible to have more than 1 valid combinational key, further lowering reverse engineering effort.

As part of future work, it is recommended to determine output corruptibility metric for both don't care-based layout and functional obfuscation methods to allow better comparison of results. This can be done by determining the Hamming distance between the outputs of the original and deceiving netlists, similar to [15].

Obfuscation can also be used in combination with existing “design for security" approaches to enhance the overall security, and is equally applicable to combinational and sequential circuits. Obfuscation and authentication go hand in hand; thus, effectiveness of an authentication technique can be substantially increased in presence of obfuscation. These sub-circuit modifications can also be used to simultaneously authenticate the design. This issue can be further explored as part of future work.

It is important to minimize the impact of obfuscation on designer and test engineer. More importantly, the added obfuscation methodology should be transparent to the end-user and have minimal impact in terms of design overhead. We explored these issues by applying the obfuscation methodology to ISCAS-85 benchmark circuits.

We have shown through simulation results that the obfuscation based techniques proposed in this research can be implemented at low hardware and/or computational overheads, with minimal impact on the end-user experience. Our results also indicate that the approach is indeed scalable to larger and more complex ICs.

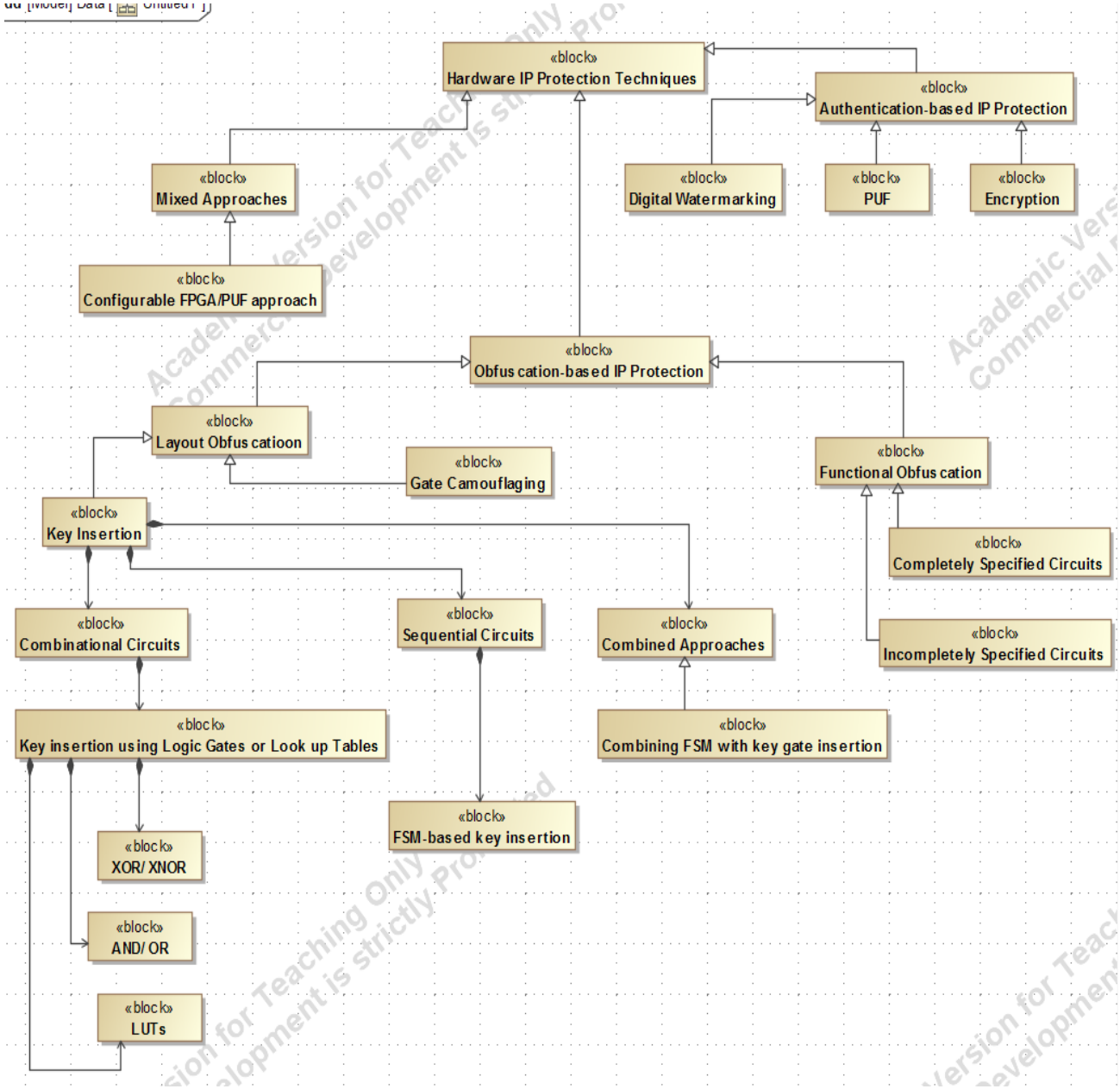
We presented two examples of functional obfuscation using ISCAS-85 benchmarks C2670 and C5315. These examples are given here as proof of concept. As part of future work, it is suggested to take this approach further by considering the implications for large, complex circuits in terms of design overhead and RE complexity. Functional obfuscation can also be used in conjunction with layout obfuscation and authentication-based approaches.

There is a potential for future work that compares the different methods of obfuscation in terms of their impact on the IC design flow and performance overhead. Potential methods include using fuses as the connections for the added lines for all possible ODC locations or “deliberately misleading”

functions so we can decide which ones are active, and be able to add connections (such as for the deliberately misleading functions) after fabricating the rest of the circuit or post-production.

Future work may also explore approaches to make the obfuscation methodology scalable by considering the issues associated with application of the proposed obfuscation process to complex SoCs, computation of an output corruptibility metric (i.e. how different an obfuscated circuit output compared to the original circuit), a more detailed comparison (including power consumption analysis) of the proposed obfuscation approaches and combinations of obfuscation and authentication approaches based on don't cares [20], and ways to increase the security against insider's attack, i.e. attacks where an associate of the design house itself is the adversary. Analysis of the effect of design obfuscation using input cone expansion as discussed in section 3.4.3 on circuit and system testability and reliability can also be investigated.

APPENDIX 1: TAXONOMY OF HARDWARE IP PROTECTION TECHNIQUES



REFERENCES

[1] Semiconductor Industry Association, “Industry factsheet." http://www.sia-online.org/cs/industry_resources/industry_fact_sheet, 2009.

[2] Semiconductor Industry Association, “Economy factsheet."

http://www.sia-online.org/cs/economy, 2009.

[3] Semiconductor Industry Association, “Industry Statistics: December 2013”

http://www.semiconductors.org/news/2014/02/03/global_sales_report_2013/semiconductor_industry_posts_record_sales_in_2013/

[4] G. Moore, “Cramming more components onto integrated circuits," Electronics,

vol. 8, pp. 114{117, Apr. 1965.

[5] R. Chakraborty and S. Bhunia, “HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 28, no. 10, pp. 1493–1502, 2009.

[6] J. A. Roy, F. Koushanfar, and I. L. Markov, “EPIC: Ending Piracy of Integrated Circuits,” in *2008 Design, Automation and Test in Europe*, 2008, pp. 1069–1074.

[7] Semiconductor Industry Association, \Global billings report history (3-month moving average) 1976{March 2009." http://www.sia-online.org/galleries/Statistics/GSR1976-March09.xls, 2008.

[8] Global Semiconductor Alliance, “GSA members.” http://www.gsaglobal.org/membership/members/index.asp, 2009.

[9] P. Subramanyan, R. Sayak and S. Malik,“ Evaluating the Security of Logic Encryption Algorithms,” IEEE International Symposium on Hardware Oriented Security and Trust, 2015.

[10] IHS Technology Press Release: Top 5 most counterfeited parts represent a $169 billion potential challenge for global semiconductor industry. https://technology.ihs.com/405654/top-5-most-counterfeited-parts-represent-a-169-billion-potentialchallenge-for-global-semiconductor-market, 2012.

[11] C. Dunbar and G. Qu, “Designing trusted Embedded Systems from Finite State Machines”, ACM Transactions on Embedded Computing Systems (2014), Vol. 13, No. 5s, Article 153.

[12] S. Narasimhan, S. Bhunia and R. Chakraborty, “ Hardware IP Protection during Evaluation using Embedded Sequential Trojan” , co-published by IEEE CEDA, IEEE CASS, and IEEE SSCS, 2012.

[13] M. Gao, G. Qu, Q. Zhou and Y. Cui, “Circuit Obfuscation: Motivation, Assumptions and Techniques.” (White Paper, 2015)

[14] Counterfeit Integrated Circuits: Detection, Avoidance and the Challenges Ahead", published by Springer, Science + Business Media, New York, 2014.

[15] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security Analysis of Integrated Circuit Camouflaging," *ACM Conference on Computer Communications and Security, 2013.*

[16] Bao Liu; Wang, B., "Embedded reconfigurable logic for ASIC design obfuscation against supply chain attacks," *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014* , vol., no., pp.1,6, 24-28 March 2014.

[17] Ronald P. Cocchi, James P. Baukus, Bryan J. Wang, Lap Wai Chow, Paul Ouyang. "Building block for a secure cmos logic cell library," US Patent no. US20100301903, 2010.

[18] Report of the Defense Science Board Task Force on High Performance Microchip Supply, February 2005.

[19] G.E. Suh and S. Devadas. "Physical Unclonable Functions for Device Authentication and Secret Key Generation", ACM/IEEE Design Automation Conference, pp. 9-12, June 2007.

[20] C.Dunbar and G. Qu, "Cybersecurity for Intellectual Property: Developing Practical Fingerprinting Techniques for Integrated Circuitry", Doctoral Dissertation, University of Maryland, College Park, 2015.

[21] J. Rajendran, M. Sam, O. Sinanoglu, and R. Karri, "Security Analysis of Logic Obfuscation," *DAC, 2012.*

[22] "Open Source Digital Flow," 26 August 2013. [Online]. Available at: http://opencircuitdesign.com/verilog/. [Accessed 14 July 2015].

[23] ABC: A System for Sequential Synthesis and Verification. http://www.eecs.berkeley.edu/~alanmi/abc/

[24] E.M. Sentovich et.al., "SIS: A System for Sequential Circuit Synthesis," University of California, Berkeley, 1992.

[25] Product website of Concept Engineering, www. concept.de/GateVision.html

[26] "TRUST in Integrated Circuits (TIC) - Proposer Information Pamphlet." http://www.darpa.mil/MTO/solicitations/baa07-24/index.html, 2007.

[27] U. Guin, J. Carulli, Y. Makris, "Counterfeit Integrated Circuits: A Rising Threat in the Global Semiconductor Supply Chain", Invited Paper, Proceedings of the IEEE, Vol. 102, No.8, August 2014.