2017

Novel Computational Methods for Integrated Circuit Reverse Engineering

Travis Meade
University of Central Florida

NOVEL COMPUTATIONAL METHODS FOR INTEGRATED CIRCUIT REVERSE ENGINEERING

by

TRAVIS PAUL MEADE
B.S. University of Central Florida, 2012

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering & Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2017

Major Professors: Shaojie Zhang and Yier Jin

ABSTRACT

Production of Integrated Circuits (ICs) has been largely strengthened by globalization. System-on-chip providers are capable of utilizing many different providers which can be responsible for a single task. This horizontal structure drastically improves to time-to-market and reduces manufacturing cost. However, untrust of oversea foundries threatens to dismantle the complex economic model currently in place. Many Intellectual Property (IP) consumers become concerned over what potentially malicious or unspecified logic might reside within their application. This logic which is inserted with the intention of causing harm to a consumer has been referred to as a Hardware Trojan (HT). To help IP consumers, researchers have looked into methods for finding HTs. Such methods tend to rely on high-level information relating to the circuit, which might not be accessible. There is a high possibility that IP is delivered in the gate or layout level. Some services and image processing methods can be leveraged to convert layout level information to gate-level, but such formats are incompatible with detection schemes that require hardware description language. By leveraging standard graph and dynamic programming algorithms a set of tools is developed that can help bridge the gap between gate-level netlist access and HT detection. To help in this endeavor this dissertation focuses on several problems associated with reverse engineering ICs. Logic signal identification is used to find malicious signals, and logic desynthesis is used to extract high level details. Each of the proposed method have their results analyzed for accuracy and runtime. It is found that method for finding logic tends to be the most difficult task, in part due to the degree of heuristic's inaccuracy. With minor improvements moderate sized ICs could have their high-level function recovered within minutes, which would allow for a trained eye or automated methods to more easily detect discrepancies within a circuit's design.

I dedicate the following work to my parents.

ACKNOWLEDGMENTS

First I would like to thank my advisor and Chair Dr. Shaojie Zhang. I would also like to thank my co-advisor and co-Chair Dr. Yier Jin. Both of which have been a tremendous help in providing me a direction and a discerning eye that was critical in allowing me to be successful in my graduate studies. I cannot thank them enough.

I would like to thank my committee members Dr. Ali Orooji, Dr. Cliff Zou, and Dr. Mingjie Lin. Their support and feedback has enabled me to better convey the research presented in this document.

Lastly I would like to thank my friends, colleagues, and family that have had to tolerate my decision of striving for my PhD. I truly appreciate everyone's support.

In this dissertation Chapter 4, in part, is a reprint of "Gate-Level Netlist Reverse Engineering for Hardware Security: Control Logic Register Identification", co-authored with Yier Jin, Mark Tehranipoor, and Shaojie Zhang in IEEE International Symposium on Circuits and Systems (2016). The dissertation author was the primary investigator and author of the paper.

Chapter 5, in part, is a reprint of "The Old Frontier of Reverse Engineering: Netlist Partitioning", co-authored with Kaveh Shamsi, Shaojie Zhang, and Yier Jin submitted in IEEE International Symposium on Hardware Oriented Security and Trust (2018). The dissertation author was the primary investigator and author of the paper.

Chapter 6, in part, is a reprint of "The Old Frontier of Reverse Engineering: Netlist Partitioning", co-authored with Kaveh Shamsi, Shaojie Zhang, and Yier Jin submitted in IEEE International Symposium on Hardware Oriented Security and Trust (2018). The dissertation author was the primary investigator and author of the paper. Part of Chapter 6's material also comes from Gate-Level

Netlist Reverse Engineering Tool Set for Functionality Recovery and Malicious Logic Detection, co-authored with Shaojie Zhang, Zheng Zhao, David Pan, and Yier Jin submitted in International Symposium for Testing and Failure Analysis (2016). The dissertation author was the primary investigator and author of the paper.

Chapter 7, in part, is a reprint of "IP protection through gate-level netlist security enhancement", co-authored with Shaojie Zhang and Yier Jin in Integration the VLSI Journal (2016). The dissertation author was the primary investigator and author of the paper.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Integrated Circuits (ICs) are more prevalent in today's society than ever. Thanks to society's growing demand for smart devices, the periodic improvement in design/cost, and the ability to outsource parts of IC production, the amount of applications that rely on ICs has drastically increased. However, this large influx of smart devices has lead consumers to doubt the integrity of IC manufacturers. An in-depth analysis of the infrastructure surrounding IC manufacturing needs to be performed. This chapter presents a broad overview of the state of hardware manufacturing.

1.1 Chip Manufacturing

A major driving force behind the IC market today is globalization. Perhaps the biggest benefit is gobalization's allowance for a convenient opportunity to match IC developers with willing and capable IC manufacturers. As it so happens, globalization strongly supports the horizontal model for chip assembly; each step of IC development can be performed by a different group. The horizontal model then allows companies to focus their resources on their chosen specialization. Due to these factors globalization decreases a chip's cost and Time-To-Market (TTM). At present, from the economic point of view the amount of work that is required for modern chip production is not necessarily feasible for any one company to oversee. In fact, many IC providers rely on a contract based method for IC production. Even companies capable of partial in-house chip manufacturing tend to rely on third party resources at some stages of chip design [1]. Overall it is commonly accepted that globalization increase availability of ICs.

Globalization is a double-edged sword. As with any process that involves many steps, we expose our ICs to a higher possibility for accruing defects, and overall, the chance for error is increased

due to having differing contributors for each step. The complex contribution model involved in chip manufacturing also makes detecting such defects in a timely manner more difficult. Although techniques such as Automated Test Pattern Generation (ATPG) has helped improved quality control [2], rare triggering errors or defects that reduce a products life can be hard to adjust and test for. Multi-state chip designs significantly increases the chance of a compatibility issue, where Intellectual Property (IP) developed by different groups might not be able to work together properly or as intended. Companies might be hesitant to share potentially sensitive information even through secure communication channels. Such companies might not even disclose detrimental corner cases that need to be handled by other IP on a large System-on-Chip (SoC). Additionally if chip development on large systems does go wrong, determining the guilty party becomes much more difficult. The resulting finger pointing can be costly to small businesses that might not be at fault, which can deter more honest producers from entering the market in the first place. All of these issues could motivate governments to step in and subject companies to more regulations, thereby costing small businesses even more money and time. Further more all these detriments leads to the inevitable concern that globalization overall reduces the consumer trust in ICs.

As globalization increases the throughput of ICs, a new form of service has gradually been introduced to the IC community, that of Third-Party IP (3PIP) vendors [3]. Such vendors will supply customers with made-to-order IC designs. The quality and level of detail of such products varies greatly. 3PIPs can be given at a level as high as Verilog or as simple as layout. Due in part to globalization the bulk of SoC development relies heavily on these services. Benefits of third-party production includes reduction in design overhead, fabrication costs, and TTM, while the drawbacks include flawed cores, or worse, cores with inserted malicious logic triggerable by malicious users. The required vetting of products before use/fabrication falls to the IP consumer, which might not be feasible given their low-level understanding of the product.

The government has already taken an interest into the on goings of electronic part providers. Such

providers have been investigated for potential selling of counterfeit military-grade goods [4]. What was found was quite disturbing; not only would used parts be sold as if new, but quite a few sold parts were non-existant, fabrications created by requesting bogus part numbers. Luckily, an array of tests exist to help determine the quality of such products. This allows malicious distributors to be dealt with. However, there is a lack of methods for determining whether an IC was created maliciously, which leaves malicious IP providers free-reign to provide bad parts.

1.2 Hardware Trojans

As mentioned previously the reliance on third party facilities leads to the potential for many different groups involved in assembling even one IC, which is why a major threat to IC consumers is that of unspecified chip functionality. Due to lack of communication between parties, or that of the desire to sabotage another entity's IC, an IC can now more than ever be subjected to failure when in use. Even Field Programmable Gate Arrays (FPGA) are susceptible post-assembly, since FPGAs can be programmed with malicious pieces of code. During behavior specification and code synthesis the code is vulnerable; the tools used to generate such content may in fact be compromised themselves [5]. Specifications received by third party resources could also have something like a backdoor that can allow users to gain more control over a system than was originally anticipated.

In the research community any malignant circuitry that is incorporated into an IC to alter its behavior from specification in any way is referred to as a hardware Trojan. Hardware Trojans most often consist of two major characteristics: the trigger and the payload. Figure 1.1 gives a simple example of a hardware Trojan, assuming that propagating the signal z to z' without manipulation is the desirable behavior of the circuit.

Triggers, as the name implies, activate the potentially dormant payload. Activation requirements

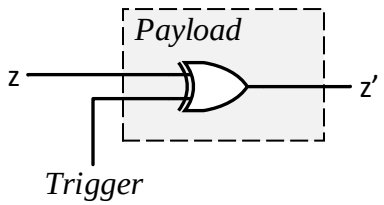


Figure 1.1: A simple Hardware Trojan.

can range from the input of a particular input, a sequence of particular input vectors, or even a required amount of clock cycles. Whatever the trigger, the event itself needs to be rare, otherwise the Trojan may be detected in functional testing of the chip. In the event of an IC user discovering a hardware Trojan, the blow back for the hardware Trojan developer would be devastating. After detection the best case for the Trojan developer is that the effort they put into developing the Trojan would be wasted since the chip would no longer be used. This spurs Trojan designers to develop "stealthy Trojans".

The payload, the most important part of a hardware Trojan, determines what adverse functionality the IC will perform. The payload can range from leaking sensitive on chip information to preventing use by stalling or outputting incorrect values. In the sample Trojan in Figure 1.1 the payload corrupts the z signal. After triggering the Trojan in this example can potentially cause IP consumers issues either at the current clock cycles output or potentially in later clock cycles.

1.3 Introduction to Reverse Engineering

To talk about circuit protection it is helpful to diverge and discuss a similar topic, reverse engineering. Reverse engineering refers to the process through which analysis of an object can allow one

to infer the process through which the object in question was created. Typically reverse engineering is looked down upon in industry due to the negative association with that of piracy. Although the way in which products are reverse engineered might be legal, how such information garnered from reverse engineering methods is leveraged typically is illegal. However, the development and results of reverse engineering can have a tremendous, beneficial impact on the IC community.

A large factor that positively impacted early development of hardware-level reverse engineering was the threat of hardware Trojans. Many early hardware Trojans were inserted into Finite State Machines (FSMs), which could allow, with certain transitions, behavior not within the original IC's specification or side-channel activity that could leak sensitive information. To help detect Trojans methods for finding such logic structures were proposed. An exemplary solution used the topology and the knowledge of control signals to help cluster signals into words [6]. Conceptually each word would form its own logical FSM. This idea stemmed from the concept that words within a netlist would have their datapath controlled by a similar, if not the same, set of signals. This method initially showed promise, in that many of the words generated were small, which allows users to quickly determine their functionality.

CHAPTER 2: LITERATURE REVIEW

As mentioned in the previous chapter restoring the trust in IP suppliers can be achieved by assisting IP consumers in determining and potentially verifying chip functionality. Although in some cases a straightforward design comparison method could fulfill this request, a more sophisticated design analysis tool is needed. The tool should be capable of using a recovered gate-level netlist as the input. The desired method should then recover the chips full functionality. Relying on these tools, users can verify the trustworthiness of purchased chips. Methods have been proposed to classify chips with malicious IP or to even fix the malicious sections of circuits [7]. However, typically these methods have been shown to be inaccurate [8].

Many works suggest that in order to find malicious components of a circuit, the user must first understand the function of the parts of the circuit. The process of extracting the high-level description of a circuit is the process called reverse engineering. To fulfill this need, proposed methods exist that map circuit components to a known set of components (i.e. a component table) by leveraging the technique of graph isomorphism [9]. Graph isomorphism has not been proven, as of the time of this thesis, to have a polynomial run time, so this method tends to be slow. Furthermore, some difficulties associated with this form of reverse engineering comes from the lack of knowledge of the processes used to generate the core. With unfamiliar processes the set of known components can be rendered useless. Fortunately, research has further developed to not rely on component tables [10, 11]. However, many methods for reverse engineering today still have drawbacks, either only partial recovery of chip functionality (e.g. detecting only data paths) or bad performance (e.g. poor runtime versus accuracy trade-off).

This chapter will focus on the previous efforts made in an attempt to restore foundry trust. It will emphasize the need for Trojan detection, while discussing proposed techniques that can help

prevent them. The section also covers the challenges associated with function recovery due in part to modern IP protection. This chapter will show the indirect link between the two problems, Trojan prevention and IP protection, and how they can, at times, be in direct conflict.

2.1 Split Manufacturing

In an early attempt to prevent information leakage using hardware Trojans, methods were proposed that implemented extra safety circuitry. Some of these extra security modules were referred to as guards or gates, and they were designed to limit the ability of inter-module communications. In [12] a hardware guard was designed to examine communication between the memory and CPU core. The guard was used to prevent sensitive data being extraneously written to the memory. The protection was done by forming a shadow set of memory operations that could be double checked with the original set of memory operations. However, their method did not prevent DoS, and the integrity of the guard was never taken into consideration.

To alleviate the concerns over potentially bad guards a method called SHADE [13] leveraged two guards that were presumably fabricated at non-colluding foundries that encrypted and decrypted the data that goes to memory such that activation of a Trojan becomes hard. Also to counteract DoS, the "outer guard" analyzed "heartbeats". If the "inner guard" takes too long to respond or was not encrypting the heartbeats correctly, the outer guard detects an attack. The concept of multiple non-colluding foundries is a tough requirement to meet and is not completely original. Split manufacturing [14] merges two pieces of circuitry to prevent foundries from understanding the intended IC. The cost associated with such a system could be quite high due to extra assembly required and the high potential detrimental affects on performance. Worse off, SHADE stops computation and provided no real solutions for handling a failed heartbeat. The authors also implied that the heartbeats needed to be adjusted as to prevent false triggers when the execution time lags for

non-malicious reasons. If the bound on the heartbeat is too loose, attacks can still be done on the IC. Lastly the authors have to assume that both the outer guard's heartbeat sensor was accurately produced and that the fabrication facilities were indeed non-colluding.

Other methods that rely on multiple non-colluding facilities have been proposed [15]. Such methods include fabricating full ICs and then compiling their output using a voting scheme. Such methods incur a substantial area and power overhead and require advanced knowledge of the IP as to provide differing ICs such that even if the foundries did decide to collude, the internal signals would be difficult to match.

Some methods take advantage of faster than homegrown ICs and perform verification by using trusted circuitry. In [16] the authors proposed a method to leverage 3PIP chips by proving that the 3PIP's results are accurate with the help of trusted, homegrown ICs. Several result-proving methods analyzed in the paper allow for a faster than homegrown IC computation. This method also creates a significant area and power overhead. Not only does this method require knowledge of the high-level functionality, but IP producers would have to disclose the IP pertaining to final IC, which many chip developers are hesitant to do. All of these methods have been classified as passive, pre-silicon prevention schemes and require access to the original design, which might not be available to the IC consumer.

2.2 From Silicon to Electrons

With the potential failure of hardware Trojan prevention many researchers looked to the possibility of checking for Trojans in designs post-silicon. Many users might only have access to very low level IC descriptions such as layout or just the fabricated chip. The fact that some ICs might not be given to a user in a computer readable format, spurred researchers to look for methods of

converting post-fabrication netlists into a machine readable low level description. Luckily several methods and services exist for acquiring such a netlist [17, 18, 19]. Many of these methods tend to be destructive (e.g. cross-sectioning, delayering, etc.). Such methods cause the IC to no longer function. Typically the recovered design descriptions can be in the form of transistor-layout or as high as a gate-level netlist containing the AND-OR gate information. Recovering higher level information from such low level descriptions requires a significant amount of extra effort.

One of the many tools at the disposal of IC users in the hopes of IC description recovery is Scanning Electron Microscopes (SEM). With many chips, preparation is needed to get a detailed image of the IC. In a process called delayering a Printed Circuit Board (PCB) can be analyzed completely. By alternating Hydrofluoric acid baths and Acetone baths a chip can be cleaned. The baths then allow for each layer's layout to be recovered with the use of a high resolution SEM [18]. Alternatively lasers, dremels, sandpaper, or Computer Numerical Control (CNC) mills can be used to remove scanned portions to reveal the next layer for analysis with varying amount of accuracy/cost associated with each method [20].

The biggest drawback to methods described above is their destructive nature. After the acid baths large portion of metals will have been eaten away which renders the circuit largely useless. However, the acids are relatively easy to acquire and renting a SEM is inexpensive. Another benefit to such a method is the short time required for recovery of small circuits. Notable Trojan detection methods have shown effective by utilizing the process of delayering through the use of chemical and imaging via a SEM alone [21].

Focused Ion Beams (FIBs) have been used to perform precise cross-sectional analysis [22]. This cross-section allows for tools such as SEMs to find potential chip defects. However, FIBs can also be used to mill part of the IC to probe internal signals of the IC. This microprobing allows one to determine functionality of potentially camouflaged or obfuscated ICs. Methods to help prevent the

recovery of sensitive on chip information have been proposed to counter FIB attacks. One example include overlaying a mesh of signal carrying wires [23]. These signal carrying wires when cut will corrupt parts of the circuit, making signal recovery impossible. Such methods have a high power and area overhead, and counter attacks utilizing the geometry of the FIB exist [24].

One approach for recovering chip information while maintaining the functionality of the chip uses X-ray to perform and micro computed tomography. Compared to SEM and microprobing the chip is left in tact and has the potential to be reused [22, 25, 26]. The price is also comparable to that of SEM [19]. Alternatively X-rays can be used to create a high resolution three dimensional structure [27].

However, a large problem lies in the resolution. To get a high resolution the IC needs to be close the x-ray emitter. The IC also needs to be able to rotate which might not be possible due to the dimensions of the IC [28]. Sample preparation can allow the IC to be scanned at the cost of the IC. A second problem is the IC design themselves. Materials resistant to x-rays (i.e. high-Z materials) could be present. A stronger x-ray can penetrate such materials, but prolonged exposure to high energy x-rays might have detrimental effects on the IC [26]. Also if the x-rays are too strong, information about the remaining material might be lost due to a lack of contrast, and thereby rendering the process useless. For this reason destructive IC recovery might be necessary.

2.3 IP Protection

With the efforts put into determining low-level functionality of chips a new problem came into the realization. The ability for one to recover any information from a netlist allows users to recover private or sensitive information that might be on the chip, which could include the layout of the IC itself. With knowledge of the design the IC could unlawfully be duplicated allowing for a

simple form of IP theft. Several approaches for preventing IP theft exist. The first of which is a passive approach where the function of the netlist remains unchanged, while the chip should be identifiable. Counterfeit ICs should be easily detected by this watermarking, but such methods do not prevent or impede IP thieves. More active methods for IP theft prevention have also been proposed. Both methods will be discussed in greater detail in this section.

2.3.1 Logic Locking

A more aggressive approach to preventing IP theft was that of Ending Piracy of Integrated Circuits (EPIC) [29]. A TRNG was used to corrupt wires in the IC. Using a secure communication channel the IP provider would communicate with the foundry and instruct them on programming the chip. After which the chip should function as normal, while the foundry has little to no information of the chip. Later methods would predict the optimal insertion spots to increase the difficulty of reverse engineering [30].

Some techniques leveraged the concept of don't care signals. The authors in [31] used analysis of the IC specifications to improve protection methods. Non-affecting control signals are inserted into the IC such that no extra functionality is included. This logic is made as complex as possible by leveraging the knowledge of the IC and the logical function that particular wires have. However methods such as these don't protect the circuit from automated techniques that can detect and remove such gates.

One tool used to help encrypt netlist is the use of a type of PUF called Random Unique Blocks (RUBs). A RUB can be used to make chip specific keys that are required for unlocking of the full functionality [32]. In this work RUBs were used with the intention of preventing a fabrication facility from stealing the netlist design. The encryption design duplicated states of the FSM and the RUB would be used to determine what key would be needed to leave the duplicated states. An

incorrect key entry would cause the FSM to lock and prevent further use, until the entire design was reset. The authors in [32] acknowledge the fact that a reverse engineering attack on the FSM would potentially be able to recover the design. The authors claimed that such a task would be intractable but provided no evidence to support their claim.

A highly influential sequential obfuscation method proposed was hardware Protection through obfuscation of Netlist (HARPOON) [1]. Their method broke up and locked parts of the netlist. HARPOON focuses on limiting access to the original logical FSM by requiring an unlocking sequence of inputs. To do so HARPOON expands the FSM's state space. HARPOON then uses the inserted states to corrupts parts of the circuit. The FSM is split into two main parts: the corrupting part (obfuscation mode) and the non-corrupting part (normal mode). For an example FSM see Figure 2.1. However, once the FSM's state is within a correct state, normal circuit execution will not cause circuit corruption. Fault-injection attacks [33, 34, 35] can be leveraged to prematurely transition to a normal mode state, bypassing the protection.

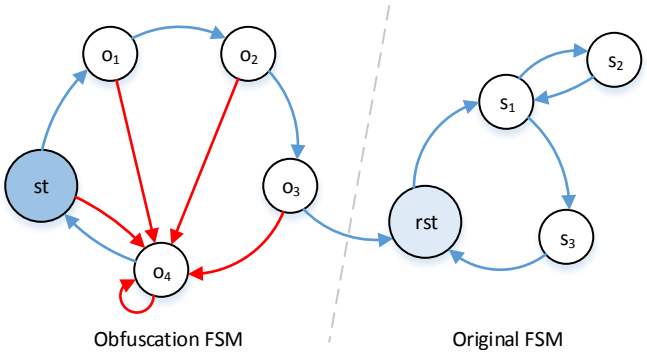


Figure 2.1: Simple example of a FSM generated by HARPOON. Red edges denote transitions that prevent access to the original FSM. Blue edges allow for the unlocking or the correct use of the original logical FSM.

Other IP Piracy prevention techniques leveraging temporal logic have been proposed [36, 37]. The authors in [36] focused on corrupting the logical FSM. The method inserts a few extra "State

Elements" (SEs) to expand the state space. The starting state of the modified FSM is within an unlocking FSM that accepts virtually any input and quickly moves the state to the starting state of the original functioning FSM. However, the input in the unlocking FSM determines an internal key that will be used for some transition in the original FSM. The wrong key will cause the FSM to transition to an incorrect state. This can have dire consequences for an IP pirate that rips off the IC with no regard to the correct sequence. However, the method is fairly easy to detect, and an adversary can leverage a SAT-based attack (which will be described in further detail later) and the scan chain to find the correct internal key thereby recovering the full unencrypted IP.

In [37] a static key is used to determine FSM traversal. An incorrect key would lead the FSM into one of many black hole states (i.e. a state designed to corrupt the IC functionality while providing no method to reach a non-corrupting state). To prevent a user from jumping to a non-black hole state via a fault (a common attack strategy) each non-black hole state required the key to function properly. The disadvantage to such an approach is that the method relies on a static key, so the use of a scan chain could provide a user the ability to determine the key, with little concern as to the actual temporal logic.

2.3.2 SAT-Based Attacks

Although methods for protecting circuits through obfuscation abound in modern research [29, 30, 38], such methods are completely or partially susceptible to combinatorial attacks proposed in research [39, 40]. A state-of-the art model, typically referred to as a SAT attack, consists of a unlocked black box that acts as an input output oracle that can be used to test potential solutions. A second part to such models is that of a locked model that describes partial functionality of the black box.

A SAT solver is used to generate a special input that distinguishes two keys by their resulting

outputs on the "discriminating input". The keys themselves both solve the current model based on the known input patterns. The whole process is achievable through the application of a series of conjoined miters one for each known input and one extra miter for the discriminating input. In [40] this SAT solver was shown to quickly, and with a perfect accuracy, decamouflage gates protected by what was considered state-of-the-art protection. Some "sequential" methods for protection rely on either a static key to ensure correct FSM transitions or an initial input sequence to program the FSM. Both of these methods are susceptible to the SAT attack, either through the use of scan chains or via a sequential to combinational circuit expansion referred to as an unrolling.

2.3.3 Watermarking

Watermarking, a well known IP protection scheme, passively protects circuits by allowing a user to verify an IP's reuse. Many watermarking based methods focus on the register assignment in the form of graph coloring, which embeds messages into the chip [41, 42]. These messages can identify the owner and fabricator of the IP, such that legal action can be taken, if piracy is suspected. Given that a sizable portion of the ICs are pirated, then only a couple hundred IDs need to be collected to determine it [43, 44].

Some watermarking techniques require a high amount of resources to redesign the IP such that each chip is uniquely identifiable. Such methods of protection are somewhat infeasible with volume of today's technology. Worse off, adversaries that are capable of recovering high-level function or even partial high-level function could redesign the chip such that the discussed watermarking methods would be bypassed.

2.4 Hardware Trojans

Since no Trojan has been reported publicly, many researchers rely on the hardware Trojans developed by other researchers to test detection methods. To assist in the development of unique, stealthy, and innovative Trojans, Polytechnic Institute of New York University designed and ran a contest in Computer Security Awareness Week (CSAW) 2008 [45, 46]. Each team developed an array of Trojans for a encrypted communications device. Each Trojan had to pass a few inspections (e.g. code inspection, functional testing, etc.) for it to be scored.

With the sheer volume of Trojan implementation a sizable portion of research has gone into classifying and bookkeeping Trojan designs [47]. Resources have been created to give access to possible Trojans based on research. The most commonly referred to benchmarks for hardware Trojans community is those provided by Trust-Hub [48].

Detection schemes aim to promote a particular Trojan design. Some authors claim that a linear-feedback shift register (LFSR) is the most difficult to detect Trojan trigger via testing methods [5, 49]. However, large LFSR are easy to find in a gate level netlist. Other authors try to emulate the effects of a Trojan (e.g. area, power consumption, etc.) [50].

2.4.1 Defenses

Many methods to prevent the insertion or in some cases the effects of hardware Trojans have peppered research. Some methods include the use of knowledge of the correct circuit structure. Using explicit knowledge of these "golden netlists" a variety of techniques have been developed and tested. One such method includes [51], which leveraged timing information to check when Trojans effect the circuits output. In the aforementioned work a circuit's process variations were simulated; statistical models of path delays over a set of challenge-response pairs were constructed.

Using this model circuits could be tested and if an IC's "signature" fell too far outside the model, which would cause the circuit to be labeled Trojan infected. This of course is prone to errors, especially since timing is sensitive to the temperature of the circuit at the time of testing. A more direct approach, which work primarily on the design level, was proposed by [52], which used a miter with a potentially Trojan Infected design and a golden model to check for any inconsistencies.

In general designers who have the full design information may compare the original design with the reverse engineered netlist to identify whether any additional logic has been inserted during the fabrication process [18, 21, 52]. However, this method may not apply to the commercial-of-the-shelf (COTS) ICs and SoCs with 3PIPs because of the lack of such golden models.

A broad categorization for hardware Trojan detection is that of destructive versus non-destructive. Methods that involve microprobing, cross-sectioning, and delayering are classified as destructive. Like methods that recover low level IP detail from chips, ICs subjected to destructive analysis become unusable. These methods have been used in fault analysis, hardware Trojan detection, and IC counterfeit detection. IC analysis tools exist which involve examining just the gate-level netlist [53]. While some destructive methods can produce ICs of this format, some gate-level extraction methods such as micro computed tomography using x-rays can leave a IC unfazed. For this reason such methods are classified as non-destructive. Non-destructive analysis is the preferable option since it can be used in tandem with destructive methods, if deemed necessary.

Some hardware Trojan defenses try to deter the insertion of Trojans. As it so happens split manufacturing [12, 13, 14, 15, 16] was an excellent attempt at prevention of Hardware Trojans, even though the original goal was, in most cases, to help prevent piracy. As mentioned earlier in the chapter these methods leveraged the fact that IC manufacturers do not necessarily need the complete access to the IP to assist designers in SoC fabrication. The major issue with such methods is that they rely on a weak attacker model. Split manufacturing either assumes that manufacturers are

non-colluding, can't infer from geometry the intended IC design, or don't have other side channels for leaking information.

Other prevention methods focus on structuring the IC such that Trojans cannot trigger or when a Trojan could trigger, would trigger often [54, 55]. These methods assume that Trojans will be inserted such that the trigger is based on some combination of internal wires. Then by leveraging Automatic Test Pattern Generation (ATPG) and knowledge of the netlist, the Trojan would be made to trigger during the testing phase, if it existed. This method leverages techniques that can be considered both prevention and detection.

The defenses that manipulate an IC at synthesis are identified as invasive methods. Other invasive methods hide the sensitive information on a chip by obfuscating the design. The obfuscation can inhibit but not prevent hardware Trojan insertion. Most Trojan detection/prevention techniques fall into the opposite category, typically referred to as non-invasive. These range from comparing side channel information to expected values to checking equivalence of gate structures. Very few methods, invasive or not, work without golden netlist information.

A popular detection scheme and one of the first attempts at classifying parts of netlists for the sake of hardware Trojan detection is Unused Circuit Identification (UCI) [7]. The method focused on finding wires that were unused in the testing phase and later omitting them. Each of the found signals is replaced with an exception throwing circuitry that triggers when the original signal would have been activated. The thrown exception is handled by the author's "BlueChip" software, which is a lightweight recovery tool that emulates the triggering instruction. The method is slightly unappealing since it requires access to the hardware in the design phase. To worsen UCI's effectiveness papers have been published that bypasses UCI [56, 8].

Broad classifications of wires and netlists have been used in many different ways. Wire categorizations have been created by representing their sub-net via a low dimensional vector defined by

their graph structure [53]. The component identification was then leveraged to help find similar components from known hardware Trojan infested designs. These flagged parts would then be verified which would require an in-depth examination. A major drawback to this method was that it mainly functions as a blacklist for known hardware Trojans, which would potentially have no effect on a 0-day Trojan design.

Logic clustering/classification via an array of tests was done in [6]. However, the accuracy of their method's results were never formally analyzed. In [5] wires pairs are blacklisted when particular states are unobserved. Aside from requiring a working golden model to test IO/state spaces with, a major drawback was the time required to simulate the circuit for finding such suspicious circuitry. Without netlist pruning the amount of time taken to determine all possible states is quite large.

Many classification methods rely on a side channel information of the netlist. Information such as area, power consumption, path delays, and temperature have been leveraged to detect hardware Trojans. Multimodular netlist classification [57] has also been shown to be effective. The biggest drawback to such methods is that they require insight as to how the original netlist golden model should behave. This might be even harder to acquire than the actual golden model, since a detailed knowledge of the cell library used is also required.

Path delay "fingerprinting" was used in [58]. The authors extracted the detailed information regarding their chips' path delay. These few chips were verified using other slower techniques. The extracted high-dimension path delay information was reduced to a lower dimension vector. These vectors were then turned into a convex hull. Any IC's path delay vector that was outside this hull was then classified as a Trojan. Their results proved to be very good. However, their method leaves to question how to verify. The process would be easy to use with access to the knowledge of the original IC design, but the focus of this thesis is on protection without golden models. Other side-channel based methods leveraged information such as delay and power consumption to de-

tect hardware Trojans [50]. However, their results are not as meaningful as could be given that the hardware Trojans were emulated by the insertion of extra ring oscillators, which can be much more taxing on a chip's resource than a simple stealthy hardware Trojan.

Methods for detecting hardware Trojans have been developed by taking advantage that hardware Trojan triggers are rare events [52, 54, 55]. Typical ATPG is not capable of detecting hardware Trojans, since they aim to find just hardware defects [59]. As mentioned earlier, the N-Detection [55] method generates a test pattern based on known rare internal wire signals. The test patterns are generated such that each rare signal in the netlist is triggered at least N (some pre-deteremined parameter) times. Trojans that trigger because of some combination of internal rare signals will have a high probability of triggering using the generated test pattern. The N-Detection method itself requires knowledge of the Golden Netlist which might not be available. These methods have been show to be effective on certain combinational netlists. However, Trojans that don't utilize internal signals (or Trojans that don't utilize trigger signals) can bypass such protection schemes. The perfect example of this would be an always on Trojan. In the defense of the above methods, detection of always on Trojans can be very difficult regardless of defense depending on the Trojan's implementation [5].

Some methods for hardware Trojan detection rely on computer vision and/or image processing techniques. By using information of the image garnered by the Golden Netlist a one class Support Vector Machine (SVM) was able to detect small defects in that could have been implemented maliciously [21]. To help prevent issues caused by process variations that can make the circuit outline noisy some leeway was given using a buffer zone around the expected layout, which was deemed to be reasonably within the realm of manufacturing. A ratio of the wrong material on either side of the extended boundary was used to judge the potential for Trojan insertion. Another method that used a SEM for layout imaging for Trojan detection was proposed in [18]. Their method was fairly straight forward as it simply used an overlay of a golden and infected netlist

to do classification. Both of these methods also require knowledge of the original design, which as mentioned before might not be available. Similarly these methods require the original design's layout, and any IC that is functionally equivalent but implemented differently would falsely trigger the Trojan detector since pixel differences would be almost everywhere.

In summary many Trojan detection schemes require access to the original netlists design. Currently many methods that don't leverage such resources tend to underestimate Trojan designers or work on a small subset of Trojans. Lastly many researchers feel based on the generation of the IC design a chip can be made impervious to Trojan insertion. All of these and more are reasons why the research community lacks the ability to protect hardware. However, a strong contender for IP verification has made its rounds recently in the hardware security domain, reverse engineering.

Reverse engineering approaches for hardware Trojan detection can use information from input output words to stitch together the data paths that make up the netlist [60]. Using techniques called forward propagation and backward propagation and by utilizing a modest signal comparison, signal pairs were checked and merged into large word sets. After which by leveraging function identification methods, a rough high-level IC design was constructed. The resulting data flow graph could then be verified by someone that the circuit contains no suspicious structures. The authors in [60] used the topology of the extracted word graph to estimate the tool's accuracy. The netlist set used to show the capabilities of their tool was a CMP router synthesized using different library and optimization settings. The topology comparison was done manually and although the number of words found between the different types of routers varied the authors claimed the topology was the same across each netlist. One drawback they mentioned was that in one of the different optimization only 4-bits of the 6-bit words were found to propagate by their heuristic, which the authors implied meant their method was susceptible to changes in optimization parameters.

Other approaches that were a little less elegant, used known structures to perform a matching of

IC parts to netlists. In [10] the authors examined netlist slices that contained 6 inputs. Using a permutation invariant function matcher, the found bit-slices are grouped into equivalence classes. Then using these classes, components were merged through two major methods either by common signals, such as ones that could control data flow or by signal propagation, like those found in adders. The authors also used a QBF solver and a set of known circuits to identify parts that behave in an equivalent manner. The author's methods were evaluated by considering circuit coverage. Perhaps the biggest critique of this method was the failure to show how much of the coverage was correct.

One notable reverse engineering approach work used hash of trees to determine if two signals are similar enough to belong to the same word [11]. The method was very simplistic; to their credit their method was much quicker than trying to match via a graph isomorphism. The authors also provided a way to measure their method's accuracy by examining the percentage of full words found and fragments found. This can be misleading if complete words are found but merged together, in a form of over grouping. Inappropriately merging two words could have a negative impact overall on a partitioning method. The authors did not directly address this type of problem. A more appropriate metric for analyzing clusters could have been used to evaluate the identification of words.

As discussed earlier computer vision techniques have also been implemented for the purpose of Trojan detection through reverse engineering netlists. The authors of [53] wanted to examine the structural properties of netlist slices to see if they could be used to potentially identify functionality. The authors used the topology of the netlist and normalized image segmentation to partition the netlist. These sub-nets, by use of their graph structure and graph dot products, were turned into low dimensional vectors. A large number of these vectors were assembled from varying IP, after which the vectors were clustered and analyzed. The resulting clusters primarily consisted of one type of vector coming from one netlist. Their method was a great example at an IP identification via

unsupervised learning. In their results it was found that some IP overlap between clusters existed, which meant that pieces or slices of netlists were identified across IPs or there were errors. Some of these ”rogue segments” were not thoroughly examined by the authors. The results, although not allowing necessarily for matching each part to the appropriate function, could help detect the presence of known hardware Trojan structures. However, the least appealing part of this method was that novel or unknown structures could be easily mislabeled, especially when learning is done on a small, misrepresented set of netlists.

Many reverse engineering methods either utilize high level information which is formed by assumptions regarding the process for IC synthesis. Other reverse engineering methods require a lack of novelty on the part of the attacker. The evaluation of these methods is abysmal, and there is lack of similarity of analysis between methods that could allow for comparison which is important since many of these methods work towards a similar goal. In short the need for these methods is just as big as the need for a more organized structure, such as problem definition and goals. To this end the following thesis presents a series of sub-problems pertaining to reverse engineering for IC validation and potential solutions for such problems.

CHAPTER 3: MOTIVATION AND ROAD MAP

In this chapter we disclose the goals, problems, and plans for accurately detecting Trojans. The chapter also helps establish a foundation upon which netlist function recovery methods can be discussed and numerically evaluated.

3.1 Remaining Problems

Although many papers have been proposed to assist users in the detection of hardware Trojans, without recovery of IC functionality it becomes near impossible to determine the existence of hardware Trojans. Highly successful methods exist that try to analyze the side channel information, but without access to an accurate golden model detection is impossible. Due to the lack of golden models in the general case the focus of hardware Trojan detection should fall to full function recovery. Even after function recovery analysis of the design should be carried out by an experienced IC designer, since high-level IC descriptions can still be quite complex.

Full function recovery is quite difficult and can be broken into many parts. We are going to focus on a small subset of problems in function recovery. The first problem being logic net versus data net classification. The logic classification problem is rarely directly tackled, but the benefits of such a method is great. Without a good logic detection scheme hardware Trojans can be overlooked. This is due to the fact that hardware Trojans tend to emulate control logic. As an example a Trojan that leaks an encryption key when a particular input is applied would need a complex comparator and the ability to allow input signal to be diverted.

A second problem that is required for full function recovery is determining how data flows through a netlist, and an important sub-problem related to data flow recovery is partitioning a netlists sig-

nals into words. The latter problem can be referred to as netlist partitioning, which should not be confused with the optimization problem presented in [61] or [62]. With knowledge of words and how their data flows through the netlist more information can be inferred such as word manipulation, and module structure. Techniques have been shown to recover module functionality based on gate-level module description alone [63].

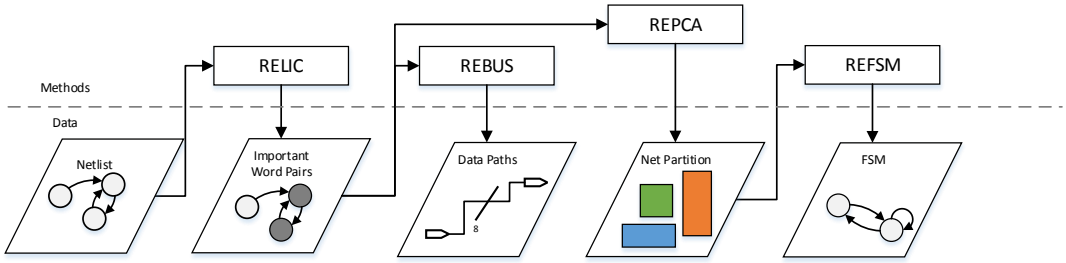


Figure 3.1: The proposed tool chain used for function recovery.

The last problem covered by the our proposed tool set is that of high-level logic description extraction. Once the logic has been identified, higher-level function might still be necessary to find potential logical flaws. Logic is capable of moving words and changing module behavior, so advanced netlist knowledge becomes highly desired. To meet such demands the use of FSMs can be employed. FSMs can simply and elegantly represent complex netlist structures that might take a long time to fully understand, if analyzed by hand and eye alone. Each state would be the state of the netlist's logic and transitions would be formed from the potential states achievable based on the current state and potential inputs. A similar problem stems from FSM extraction; it might be required to extract sequences that move from one state to another within a FSM. This sub-problem boils down to a simplified version of ATPG. A simple solution will be discussed in more detail later.

This list of problems is daunting to say the least. A solution to each problem will be provided. A

high-level mappings of problems can be seen in Figure 3.1. RELIC will be used to classify netlists as logic or data, and REBUS will be used to extract data paths. Partitioning netlists will be done by REPCA. While, High-level logic extraction is left to REFSM.

3.2 Logic Partitioning

Being a major goal of full function recovery, the problem of logic partitioning or classification becomes one of this thesis' key focal points. As mentioned in Chapter 2 and earlier in this Chapter, hardware Trojans tend to emulate logic, so detecting logic would at least allow IP consumers to detect the wires that are most-likely to compose the Trojan module. Previous solutions to logic classification is that of classifying registers, if they can directly or indirectly affect themselves, in later clock cycles [64]. This is a decent first approach, but when designs are structured to be area efficient such as an AES encryption core that uses a counter to determine the round, then the separation of logic and data becomes obscure. To prevent this misclassification a novel method is proposed that learns from the netlist.

The idea is based on an observation that register pairs from the same datapath will have a similar deep logical structure. Two data registers with the same word rely on the same control logic registers to propagate their signals along the same datapath from fan-in to output. The corresponding data signals on the datapath most-likely come from a similar logical structure from the previous clock cycles. A similar logic structure will be present all the way to the input signals. By taking advantage of the similarity of the data logic registers fan-in structure, we can accurately analyze a netlist that has a datapath controlled by potentially obfuscated signals.

3.3 Netlist Partitioning and Evaluation Motivation

Large SoCs can be easily composed of many different IPs, and even worse each IP could have many different modules creating very large, complex structures, which would be difficult to reverse engineer as a whole. To reduce the required effort for analyzing circuits a different approach can be taken. Rather than attempting to determine the IC's functionality all at once, researchers try to analyze pieces of the netlist and then after figuring out the components try to determine the full picture. To start this process of partial analysis it is required that accurately or at least meaningfully breaking a circuit into pieces can be done. As mentioned previously there are many abstractions and layers that can compose a large SoC, as can be seen by Figure 3.2. Partitioning can be attempted at each of these levels and function/module matching could be performed for each of these resulting partitions. This thesis will focus on word-level partitioning in gate-level netlists. To address the issues plaguing other word partitioning methods mentioned in Chapter 2 we develop a formal procedure based on well documented clustering techniques that leverages the exact word-level information of the original netlist. That is the resulting partition of our methods will be compared to the original design's intended word sets.

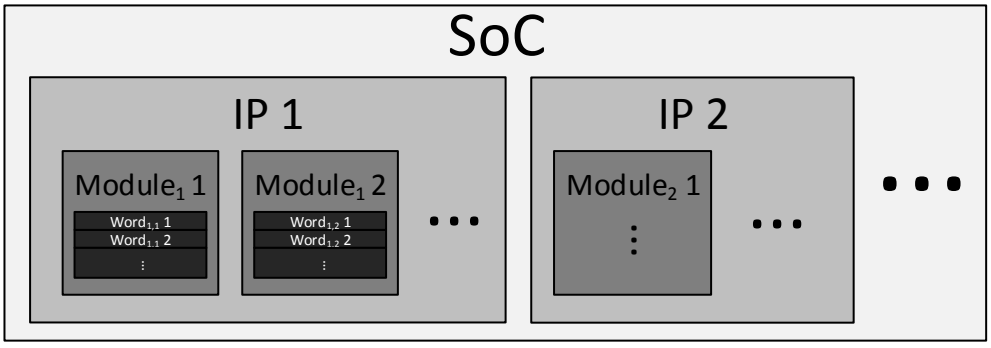


Figure 3.2: Simplified hierarchical of an SoC.

As shown in Chapter 2, there exist a number of ways to numerically evaluate a partitioning method.

However, many methods introduce bias that allows certain possibly flawed methods to appear to work well. For example, counting the number of complete words found can be very misleading. If words A and B are found by merging all bit of A and B, we could argue that A and B, were found. However, information of the words' separation gets lost by this partition, so although by the metric the method might look efficient, the partitioning method does not work perfectly. In a similar line of thought we need to be aware that the coarse and fineness of the ground truths structure should be taken into consideration when evaluating a method's capabilities of partitioning.

Due to the vast differences in hierarchical structure between ICs it becomes difficult to develop an evaluation that does not favor certain methods. To prevent this bias the method of Normalized Mutual Information (NMI) is leveraged, which although has roots deep within information theory domain, has been shown to be a modest method for evaluating clustering schemes regardless of the entropy of the ground-truth [65].

Another difficulty associated with evaluation of such partitioning methods is that of mutliple interpretation of ground truths. When working with the HDL, or a similar high-level language, the ground truth partitioning becomes even more difficult to infer due to a potential reliance on semantics that could change the ground truth without changing the observed netlist. Barring the potential for multi-interpretation, we can assume that there will be only one unique ground truth partition per gate-level netlist.

The last potential problem pertaining to netlist partitioning covered in this paper is that of multi-membership. Such a situation occurs when a signal is shared between words. A simple example could be circuit reduction found by the tool used to synthesize the circuit. There is always a possibility that in a large circuit redundant signals could be merged or removed to improve IC performance. Multiple membership could also occur when, based on certain control signals, a wire has different behavior. Probably the worst situation for multi-membership to appear is when

there is a mistake in the higher-level code that somehow gets propagated through to the resulting gate-level netlist.

A previous approach used for finding possible similarities between signals within words was comparison of signal graph information [10]. A major detriment to such methods is the redundancy of certain graph or structural information. This redundancy is caused by the fact that chip synthesis is typically performed by a deterministic protocol that optimizes a netlist structure. The computer has a high chance of incorporating similar structure types due to a user's desire to optimize for some design parameters, because of the described automation process, the variance in structure across several variables might be poor.

In [10] dimension/information reduction was simply done by leveraging graph dot products. This reduction could allow a more accurate matching by eliminating extraneous information that could lead to a misclassification while still preserving the potentially distinguishing information. However, for our work we plan to use a more commonly used method. One of the most common statistical method for dimension reduction is Principal Component Analysis (PCA). Due to it's wide usage we leverage it as a potential solution for netlist partitioning.

3.4 High-Level Netlist Description Extraction

Many early hardware Trojans in research were emulated using a FSM. The trigger condition could be a transition within the FSM that was rare via a large number of required input conditions, an exact sequence of state transitions, or some set number of clock cycles which would allow IP consumers to unknowingly avoid potential problems for potentially the lifetime of the IC. However, with a certain sequence of commands or a single input, a malicious IP provider could gain control over the IP in a way that could hurt the chip's consumer. To help detect such Trojans we seek to

extract FSMs in a netlist so that a trained IP consumer could find potential transitions or states that are suspicious, and by double checking IC behavior could determine the chip's integrity. The FSM extraction is done by simulating parts of the netlist in a somewhat abstract level. The states of the FSM become a composition of several registers at various clock cycles, and similarly the transitions are formed by considering the states the FSM can move between for differing sets of inputs.

When considering wires to simulate when extracting high-level details, the fewer the number of wires the better the run-time tends to be, even when states are determined using a subset of tracked states. To have the fastest overall runtime, only logic registers should be tracked when recovering a netlist's logic. Conversely, it might be the case that data registers can determine which states are visited and can change transition conditions. Even though data might not affect the state registers immediately, data registers have the potential to cause significant changes to state register values in future cycles. However, some of the registers might not be pertinent to what state the circuit is in or can visit. As an example, the value of a register which only affects output pins should probably not be store, and, unless considered a state register, it can be removed from the simulated netlist since it does not affect state registers. To this end we reduced the number of simulated registers, if when extracting high-level netlist descriptions, the runtime goes above some predetermined threshold.

CHAPTER 4: REVERSE ENGINEERING LOGIC IDENTIFICATION AND CLASSIFICATION

To assist with IC verification through high level function recovery, a novel wire classification algorithm is proposed. The proposed method, referred to as Reverse Engineering: Logic Identification and Classification (RELIC), will leverage the gate-level netlist to output a set of wires that are most likely to carry malicious signals.

4.1 Methods

RELIC determines the likelihood of a wire carrying logic by taking in an arbitrary netlist and producing a list of similarity scores for each register pair. These scores allow for a classification of registers that are important (e.g. a Trojan register or an intended chip control logic register). This Chapter covers a method for generating similarity scores through the use of a mixture of dynamic programming techniques and advanced graph algorithms, thereby creating a type of pseudo graph isomorphism. However, rather than comparing against a secondary graph structure, the structures of the netlists registers are compared against each other. This technique could also be adapted to comparing other netlists to an original one, which can allow for custom module library identification. RELIC also can identify intended chip logic registers by finding those registers that are not found to be part of a data signal on any datapath. Lastly the method can help find malicious logic inserted by others, since Trojan logic registers normally have logic different than any other original register fan-in structure, logic or otherwise.

RELIC was developed to replace the common graph isomorphism approaches with a faster heuristic by loosely comparing the topology of the fan-in logic. Due to its pseudo-isomorphisms fuzzy

logic, RELIC can match registers corresponding to the same word with an accuracy higher than traditional word checking methods that require the logic to meet a very specific structure. Figure 4.1 shows the work flow diagram for RELIC. It can identify registers, or even words, that are similar, but if there is a word that is improperly connected within the chip, RELIC might allow it to go undetected. This tool can be used in combination with other functional testing (or another lower level tool of the sort) to verify its findings.

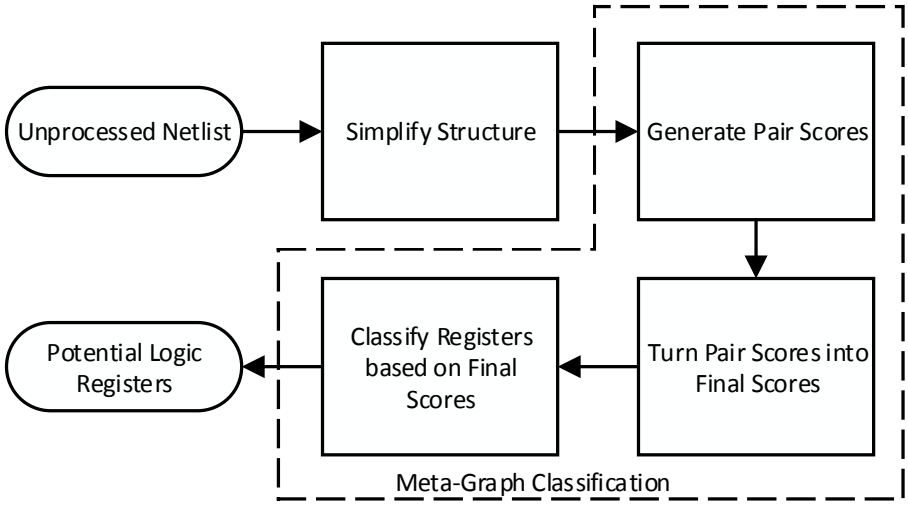
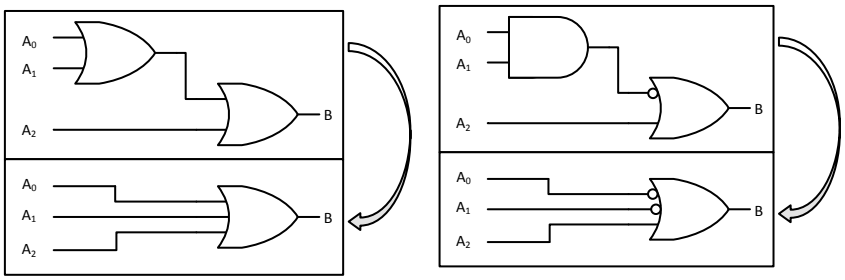


Figure 4.1: Flow Diagram for RELIC

4.1.1 Preprocessing

Taking an arbitrary pair of logic vertexes (registers or logical gates) RELIC will generate values that represent how similar their fan-in logic structure are to each other. The most obvious thing to do is to check if the logic function used by the logic vertexes are equivalent. If this preliminary check fails, a score of near zero is given to the pair. However, this check is too strict. As an example, NOR and AND have similar output types, and thus it would be desirable to match them to each other. Similarly XOR can be simulated by an OR and two AND gates. In these two

cases registers can easily have the same logic but have varying raw structures. Thus RELIC uses a preprocessing step to reduce the structure complexity. For simplicity all XOR gates will be reduced to AND-OR-INV logic.



(a) Similar gate types. (b) Differing gate types.

Figure 4.2: The two FSMs recovered from the RS232 netlist.

First, when designing a netlist some gate level obfuscation might occur, either purposefully or accidentally. Figures 4.2a and 4.2b show simple examples of functional obfuscation. There are two main scenarios. The first scenario is when the input logic vertex has a similar AND-OR logic to that of its parent logic vertex, and its output is not inverted (see Figure 4.2a). The second scenario is when the input logic vertex is inverted, and it has a differing AND-OR functionality (see Figure 4.2b), then the child logic vertex can be merged using DeMorgan's Law. The first part of preprocessing, the fan-in is checked for potential inputs that can be combined. If so, all wires can then be merged, and this process is repeated until the logic vertexes cannot merge with any of their un-merged children. Obviously no registers have their logic merged with a fan-in register, because we assume that registers update only on an edge of the clock cycle.

Second, a color is given to INPUT, AND, OR, register AND, and register OR logic vertexes. Additionally when checking two logic vertexes one might also want to check if the structure of the first is similar to the inverse of the second. This can be simulated by swapping AND color vertexes

with OR color vertexes and vice versa, in the inverted logic fan-in subgraph. This color/logic swap can then be used as a preliminary verification that within an inversion logic of the two vertexes have a high potential to be similar.

4.1.2 Scoring Function

RELIC generates similarity scores for an arbitrary pair of logic vertexes. Each score will fall in the rage 0 to 1, where scores of 1 will denote identical fan-in structure, 0 will be no common structure. These scores will be obtained by determining the similarity of all pairs of inputs between the logic vertexes in question. A connection will be added to a bipartite graph, if the score was above a predetermined threshold. A matching algorithm is then used to find the maximum disjoint children pairs that are similar between the logic vertexes under analysis in the constructed bipartite graph. Figure 4.3 shows an example of the bipartite matching of the two wires.

After finding the maximum matching of the bipartite graph, the similarity score (potentially weighted matching of the fan-in pairs) for the given wire pair is normalized by the maximum number of inputs between the two logic vertexes ($max(n, m)$, where n and m denote the sizes of the fan-in for each wire under comparison). Re-computing similarity scores can hurt run-time performance, especially if a logic vertex has a large fan-out set. To prevent re-computations, a dynamic programming technique of memoization is used.

The case of infinite recursion at this point needs to be addressed. Based on the current scoring functions, a pair of logic vertexes that each contain a fan-in path affected by their respective outputs might not halt on RELIC. To prevent this infinite loop from occurring a user defined depth, d, is also passed into each of these score queries. This depth is reduced by 1 in each recursive function call, and if the current depth is zero, then the return score is the smaller number of children ($min(n, m)$) over the larger number of children ($max(n, m)$) The pseudo-code for RELICs main procedure is

described in Algorithm 1.

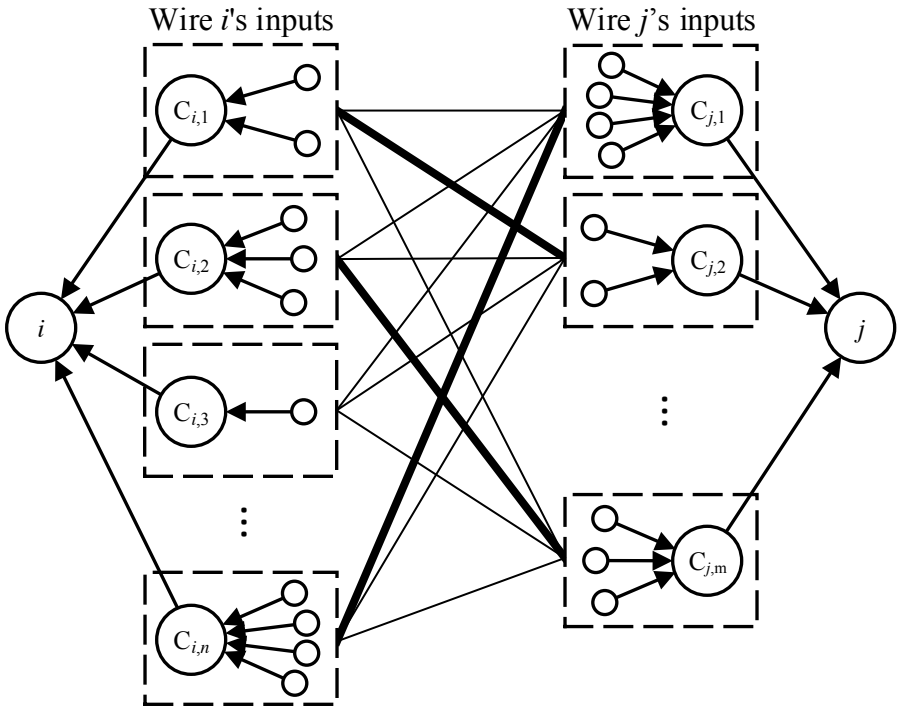


Figure 4.3: Conceptual weighted matching of two wires i and j, where the thicker lines represent the maximal weighted matching.

The run-time for this algorithm can be easily determined. Each pair of logic vertexes will be checked at most d times. If $Score(i, j, k)$ was already computed, the Dynamic Programming technique of memoization would return the previously computed value. The worst case run-time becomes $d \times N^2 \times O(MAXMATCHING)$, where N is the total number of logic vertexes in the netlist. Since maximum matchings run-time is polynomial, so RELICs run-time is polynomial, which is one additional advantage RELIC has over traditional graph isomorphism based approaches.

Once the similarity scores are obtained a simple classification is performed to identify logic registers. Each register has a counter initialized with zero, and for each similar pair of logic registers

(register pairs that have a similarity score above some pre-determined threshold), the logic registers respective counters are updated. Registers with high counters (above some pre-determined value, normally 0) are selected as non-logic affecting registers. In addition when scores were found to be one the number of comparisons were reduced by duplicating the scores across the board between the two values.

Algorithm 1 Compute similarity score between two logic vertexes in a $graph$, with indexes i and j, using a given depth, d, of their fan-in subgraphs.

```
1: function GetSimilarityScore(graph, i, j, d)
2:     max ← Max(graph[i].numChildren, graph[j].numChildren)
3:     min ← Min(graph[i].numChildren, graph[j].numChildren)
4:     if graph[i].color ≠ graph[j].color then
5:         return 0
6:     end if
7:     if d = 0 then
8:         return min / max
9:     end if
10:    Let G be a graph with a node for each child of i and j
11:    for a ∈ graph[i].children do
12:        for b ∈ graph[j].children do
13:            simScore ← GetSimilarityScore(graph, a, b, d − 1)
14:            if simScore ≥ Threshold then
15:                Add edge from a to b in G
16:            end if
17:        end for
18:    end for
19:    return MaxMatching(G) / max
20: end function
```

4.2 Results

A collection of netlists, including AES, MC8051, RS232, RSA, s349, and AES-128 were used to benchmark the performance of RELIC. For testing purposes RELIC used a depth of 7 on every netlist, except the MC8051 netlist which used a depth of 5. With 100% sensitivity of recovering the control logic registers, the overall accuracy was about 90% except in the instance of RS232. Detailed results can be found in Table 4.1 and the two low-accuracy cases of RS232 and MC8051 are discussed below. All simulations were run on a desktop of a 3.40 GHz Intel i7-4770 processor.

Table 4.1: RELIC Run-time and Accuracy Results

Netlist Name	Registers	Gates	Meta-graph Threshold	Accuracy %	Run-time
RS232 Transceiver	59	168	.8	79.6	2 s
32-bit RSA	555	2139	.8	95.3	3 s
MC8051 μP	578	6590	.9	89.1	10 s
AES-128	3968	12576	.8	100	240 s

The MC8051 proved to be very challenging. Due to the size of the fan-in trees a smaller depth was used. This small depth caused many pairs of registers to have higher scores than what they should have. To combat this shift towards a denser Meta-Graph a higher threshold was used. After the parameter changes RELIC using the Meta-Graph heuristic identified 63 registers with the potential to be logic registers. All intended logic registers were in this subset. Since the MC8051 is a micro-processor and much of the netlist is used for logic, which causes many unique structures. However, for the sake of high level instruction handling behavior recovery only 3 registers were selected as the ideal logic registers, which causes a low accuracy.

The RS232 core was an example of a netlist that can potentially have poor results when run on RELIC. The most notable problem this netlist had was that about one third of the registers were classified as being potential state registers. Only one tenth of the total registers are, semantically speaking, state registers. We can try to reduce error by changing the threshold but it still leaves about a fifth of the registers being false positive. The first problem when recovering the logic in the transceiver is the small size of the netlists. A large percentage portion of registers depend on the logic registers at varying depths. This caused many registers dependant on the logic to be falsely classified as logic.

A second issue is due to the structure of the RS232 chip itself. RS232 is the concatenation of two independent modules into one chip. This can cause the registers to be improperly identified as being similar to each other from different modules. These false positives can make the actual

similarity harder to detect. This false matching can be prevented by using methods similar to those used in other papers, such as WordRev [60], but using such methods could hinder RELICs ability to remove obfuscation.

4.3 Discussion

This Chapter presented a novel polynomial time method for classifying control logic registers and data registers from an arbitrary, and potentially obfuscated, netlist. The significant advantage RELIC has over previously proposed methods for register classification is its ability to use the given netlist as a reference when determining data words. This allows RELIC to bypass most obfuscation techniques and accurately determine and group word registers. By using max-cost-flow and dynamic programming RELIC was able to quickly classify medium size netlists. The preliminary results show that this method works on a large number of netlists with different structures and libraries. As a secondary function the given procedure can be capable of grouping together logic with similar functions, which can help a user when attempting to determine the full functionality of a chip.

CHAPTER 5: REVERSE ENGINEERING WORDS VIA PRINCIPAL COMPONENT ANALYSIS

For solving the problem of partitioning a netlist into a higher level structure we looked to leveraging common clustering techniques. This Chapter discusses a method that leveraged the structural information of each signal and the dimension reducing properties of PCA to estimate the original intended word sets of the netlist.

5.1 Methods

Each signals' initial numerical information for the PCA based netlist partitioner was derived via certain structural data, examples of which are fan-in set sizes and fan-out set sizes both at various depths in the gate-level netlist. It is possible that the same signal can belong to multiple sets, see Figure 5.1 for an example. Other fields consisted of gate type (OR, AND, XOR), and temporal logic type (flip-flop or not). We also kept information regarding the number of clock cycles for primary input to affect the gate and the number of clock cycles for the gate to affect a primary output. Other netlist distances leveraged include the closest flip-flop in the fan-in tree and closest flip-flop in the fan-out tree, both of which are measured by the number of non-buffer/inverter gates between the signal and selected flip-flop. Also leveraged was a small set of similarity scores, derived using the procedure in Chapter 4, (in our tests we selected five wires to compare against) extracted from the wires within the netlist under examination.

After generation of the principle components the first n_{pca} components are used for comparison of signals. For the comparison a simple distance metric is used to determine membership, where two points are within the same word, if the euclidean distance between the two points are less

than pre-determined cut-off. To determine the appropriate distance used for membership cut-off a random set of edges are selected. A ratio of edges that are included within words compared to the total number of edges is generated by an expected number of words in the final partition. Although it might not be the case, by assuming the inter-word signal distances are always smaller than the intra-word distances and that the number of signals per word are constant, then the ratio can be approximated by one divided by the expected number of words.

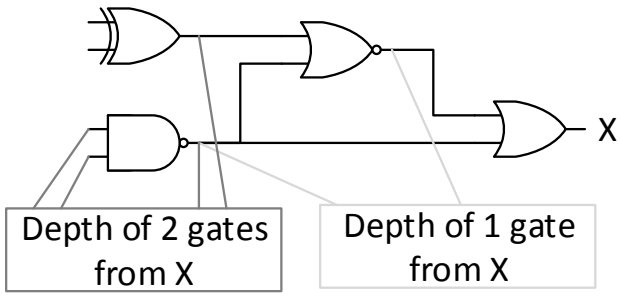


Figure 5.1: A simple example of the structural information that can be extracted from a gate-level netlist.

Based on this expectation and also on a the desired number of clusters a distance is selected from the sorted set, which acts as a good edge length cut-off. A sweep is then performed over the set of signals. Like the pairwise logic classification method, when a random signal is found to not be in a word, the signal is used to start a new word. Any signals found to be within a specified estimated distance is then joined to the word. A KD-tree can be leveraged to prune out signal pairs that are too far apart, reducing the overall runtime. Once each signal has been handled, the word sets are returned. This process can be seen by Algorithm 2.

As discussed in Chapter 3, the technique for evaluation of partitioning methods used in this is NMI. The method itself is quite simple and has been used frequently for clustering evaluation, which again makes it an obvious choice when selecting an evaluation method considering in some sense netlist partitioning is a form of clustering. As in [66] the formulation of NMI of a partition

P and ground-truth T can be expressed as

$$I_{norm}(T,P) = \frac{I(T,P)}{H(T)+H(P)} \tag{5.1}$$

where $I(T,P)$ is simply the mutual information and $H(X)$ is a type of entropy that normalizes I. Both of which can be calculated by the following equations,

$$I(T,P) = -2\sum_{i=1}^{C(T)}\sum_{j=1}^{C(P)} |T_i^c \cap P_j^c| log(\frac{|T_i^c \cap P_j^c||T|}{|T_i^c||P_j^c|}) \tag{5.2}$$

$$H(X) = \sum_{i=1}^{C(X)} |X_i^c| log(|X_i^c|) \tag{5.3}$$

where $C(X)$ is the number of classes in partition X, $|T|$ is the total number of element or nodes in the partition, and X_i^c is the set of the i-th class of partition X.

The value returned by the NMI is a real number in the range [0, 1]. The closer to 0 the worse off the partition is compared to the ground truth, while a NMI of 1 would be an exact match. As an example if a partition where all wires are in the same cluster is compared against a ground truth with at least two clusters, then the resulting NMI is 0 as no information is recovered from the partition.

5.2 Results

To show the significance of this method we compare the resulting NMI to that of another approach that attempted to partition the netlist into words. The approach compared against searched for logic word that could potentially behave like a Trojan [6]. The approach used information of control signals to partition the netlist. Signals that had a common input signal would be grouped

together, and conversely, signals with differing input signal sets would belong to different words.

Algorithm 2 Determine the word sets of a netlist with a set of signals S, given their principle components, pc, a desired number of words, n_w, a scaling factor α, and a distance metric, d.

```
1: function GetWordSet(S, pc, n_w, α)
2:     randDistances ← ∅
3:     i ← 0
4:     while i < α × |S| do
5:         a ←Random x ∈ S
6:         b ←Random x ∈ S
7:         randDistances ← randDistances.append(d(a, b))
8:         i ← i + 1
9:     end while
10:    sort(randDistances)
11:    index ← |randDistances| / n_w
12:    ϵ ← randDistances[⌊index⌋]
13:    words ← ∅
14:    seen ← ∅
15:    for Random x ∈ S ∧ x ∉ seen do
16:        seen ← seen ∪ {x}
17:        X ← ∅
18:        for y ∈ S ∧ y ∉ seen do
19:            if d(pc[x], pc[y]) < ϵ then
20:                seen ← seen ∪ {y}
21:                X ← X ∪ {y}
22:            end if
23:        end for
24:        words ← words ∪ X
25:    end for
26:    return words
27: end function
```

	Minimum	Maximum	Average
AES	0.09s	0.46s	0.21s
MSP430	0.18s	0.71s	0.53s
MC8051	0.28s	0.89s	0.49s
RSA	0.06s	0.17s	0.15s

Table 5.1: Time taken for various control signal based partitioning

However, this method then requires some form of high level information: control signals. Barring this a user would not be capable of recovering the functionality. The authors claimed that the control input of a MUX gate would be an example of such a signal, but with limited gate types,

IP consumers might not be privy to the information of what constitutes the control signal of the netlist. To allow such an approach to be leveraged we examined all the signals that could affect output or registers, and those with the highest fan-out set was selected as a control signal. Since there could be a number of signals that could be selected for control signals we simply choose the first n signals that had the highest output size, where n was some predefined number. Since each different value of n could create a different partition and a different NMI, we let n be anywhere from 1 to 400 and computed the NMI based on the result. Each of the minimum and maximum achieved NMIs are compared to the NMI that were computed over the different parameters used for the PCA based partitioning.

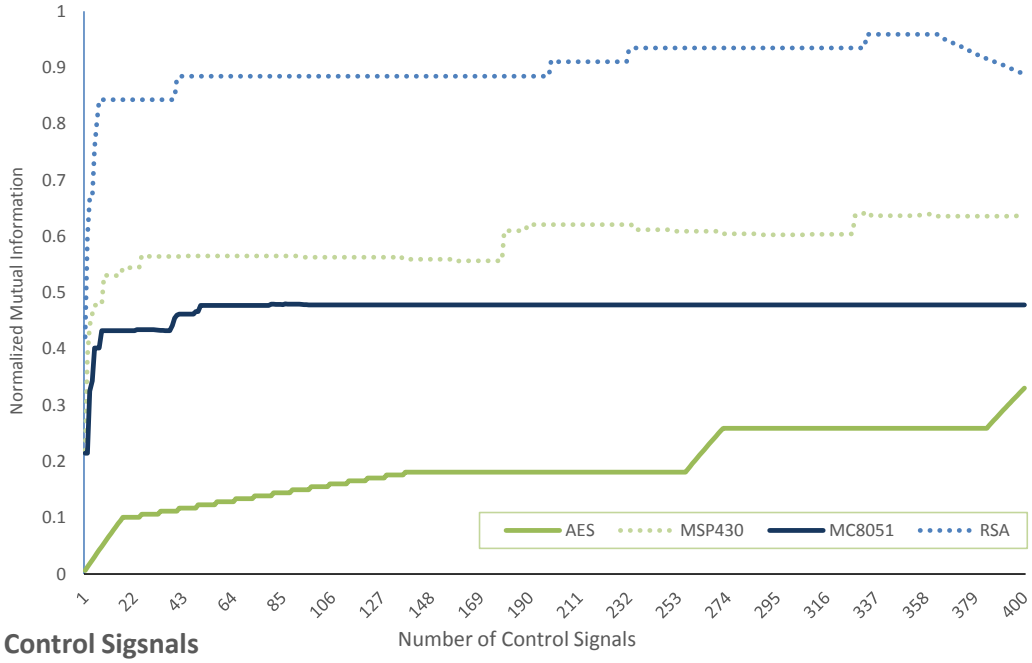


Figure 5.2: NMIs found using control signals on RSA, AES, and MC-8051 netlists.

The PCA based method was also, with a limited parameter set, capable of outperforming the control signal based method in terms of NMI. PCA, like logic classification and bus based methods, was unable to overtake control signal based partitioning as seen in Figure 5.6. As one would expect

PCA has its best accuracy when the expected number of words passed to the program is close to the ground truth's number of words. However, in practice guessing the correct number of words might be difficult, and PCA's performance suffers when the expected number is too low, as can be seen by Figures 5.3, 5.4, 5.5, and 5.6. By leveraging the distributions of distances the expected number of words might be better inferred, but this is left as a task for future works.

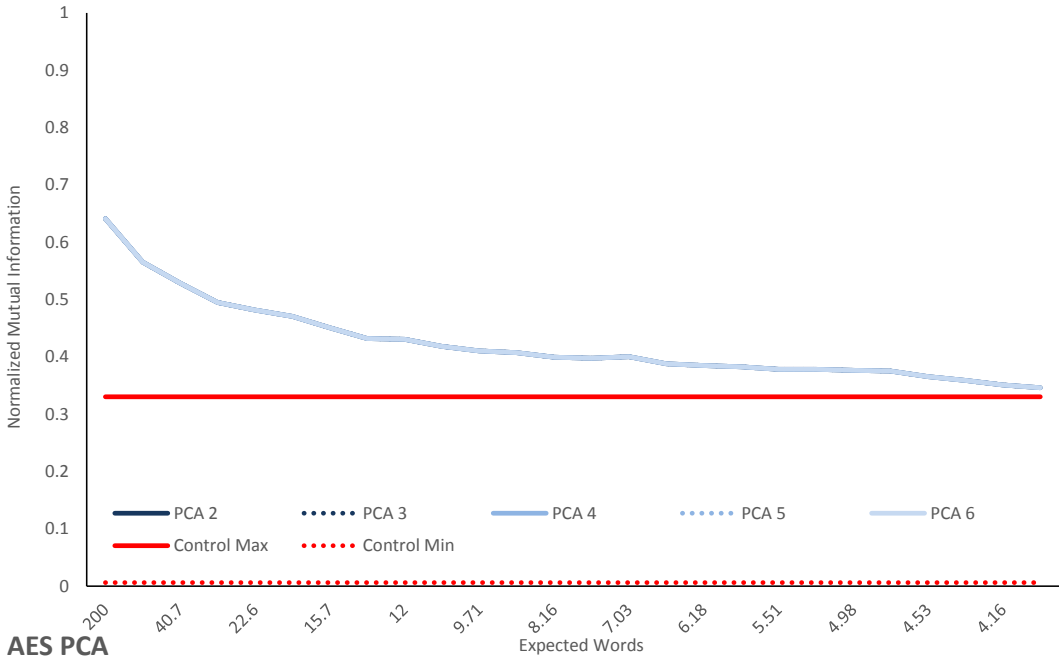


Figure 5.3: Cluster Scores found using PCA based partitioning methods compared to control signal based matching on the flattened AES-128 netlist.

Table 5.2: Average time taken for the PCA based partitioning method shown alongside the number of Flip Flops (FFs) and the Ground Truth's (GT's) entropy and word count

Depth	2	3	4	5	6	FF Pins	GT Entropy	GT Words
AES PCA	23.9s	23.8s	23.9s	23.8s	23.9s	6720	5.36	405
MSP430 PCA	0.90s	0.91s	0.89s	0.88s	0.93s	734	4.33	133
MC8051 PCA	1.44s	1.40s	1.53s	1.49s	1.50s	578	4.47	121
RSA PCA	0.83s	0.86s	0.87s	0.88s	0.83s	295	2.53	16

Aside from NMI, we also examined the difference of time taken for the control signal based heuris-

tic versus that of REPCA. The Control signal based method's run time can be seen in Table 5.1. The time taken for the PCA can be seen in Table 5.2.

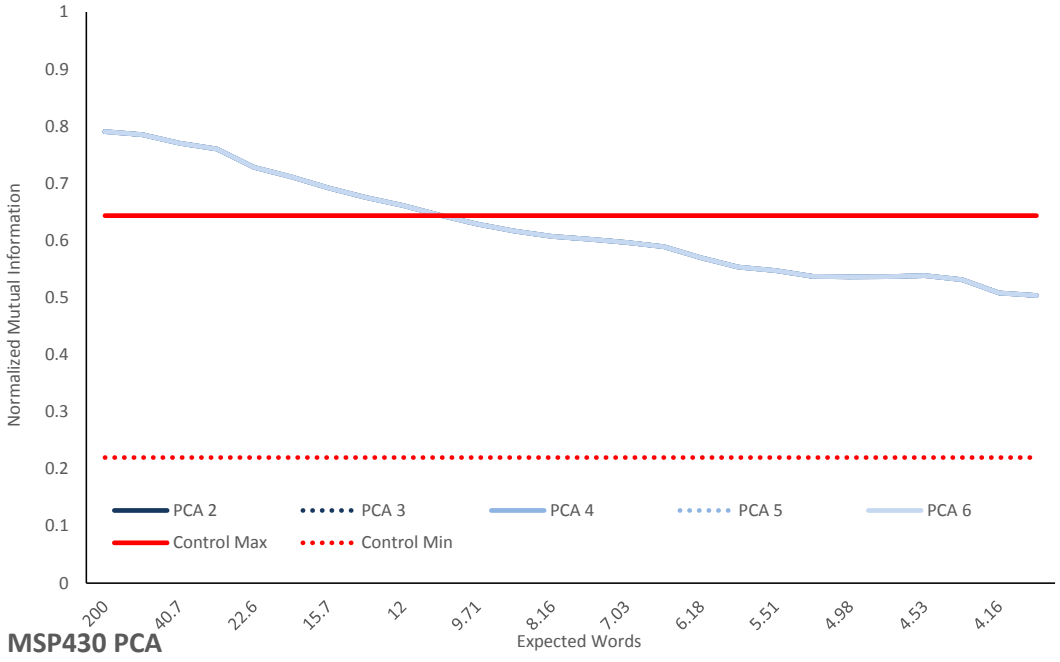


Figure 5.4: Cluster Scores found using PCA based partitioning methods compared to control signal based matching on the MSP430 netlist.

5.3 Discussion

In each netlist REPCA has its best result with the correct expected number of words. REPCA tends not to vary much with the depth used in the RELIC variable. However, the runtime does not vary much from the chosen parameter either. It appears that either the PCA or the comparison part has the larger overhead. In many cases REPCA outperforms the control signal based scheme even when the expected number of words is far from accurate. However, it should be noted that control based signals appears to work extremely well on smaller netlists, achieving a much higher possible accuracy than that of the PCA based method.

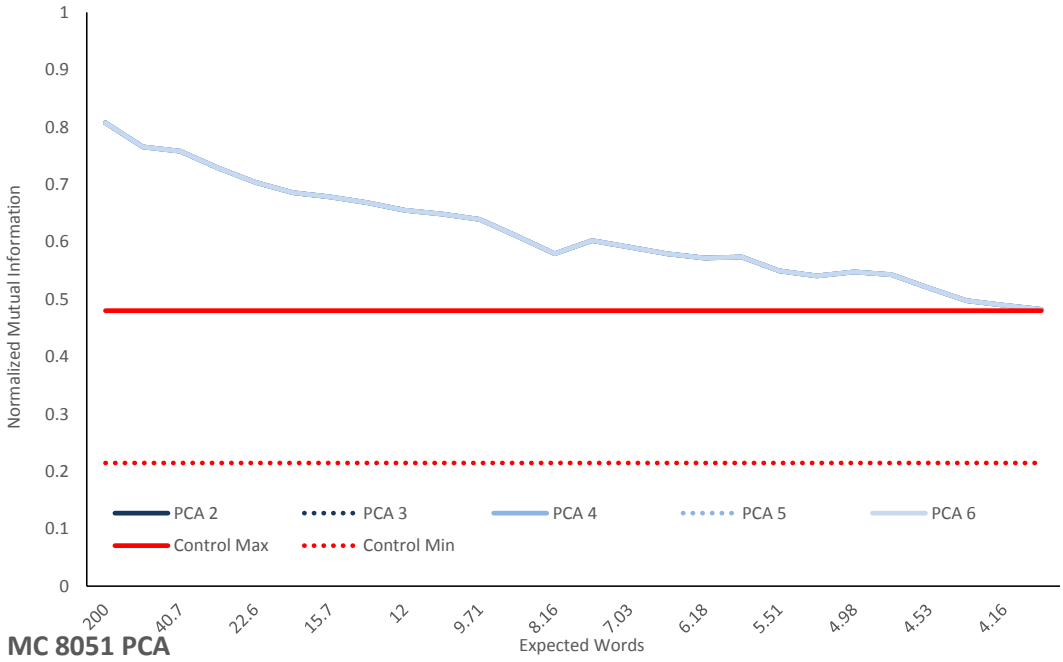


Figure 5.5: Cluster Scores found using PCA based partitioning methods compared to control signal based matching on the MC-8051 netlist.

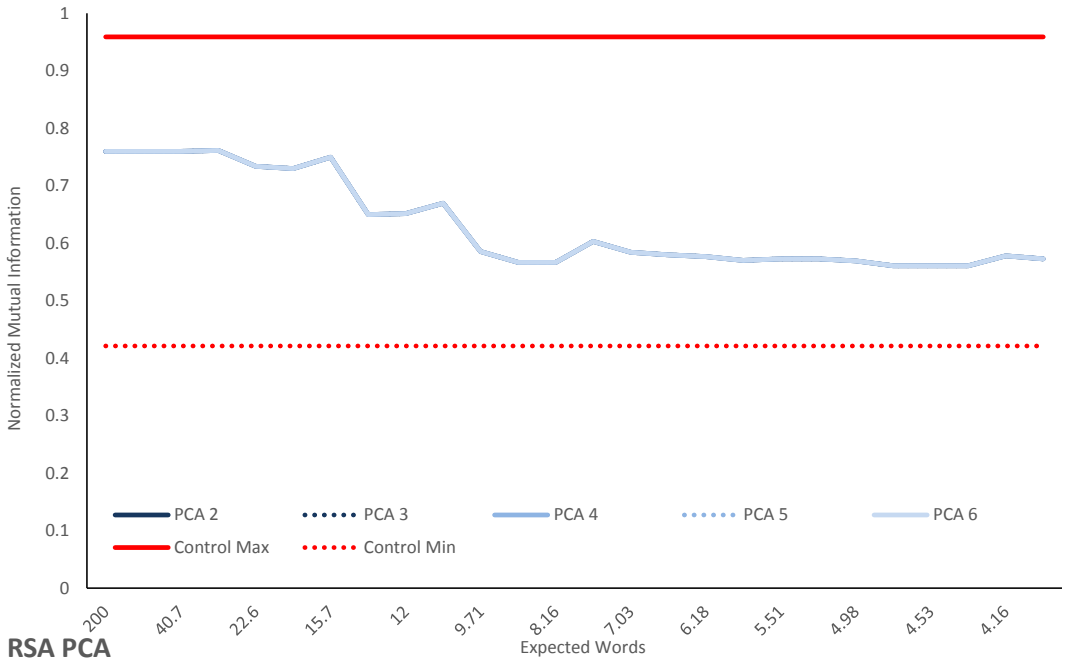


Figure 5.6: Cluster Scores found using PCA based partitioning methods compared to control signal based matching on the RSA netlist.

CHAPTER 6: REVERSE ENGINEERING DATABUSES

This Chapter focuses on presenting a novel method for recovering the data paths from a gate level netlist utilizing metrics proposed for logic classification in combination with forward propagation. The method used is commonly referred to as Reverse Engineering Databuses (REBUS). The proposed method will do so without the use of an external source that specifies component structure.

6.1 Methods

IC reverse engineering should not solely distinguish data from logic, and access to the expected number of words or the number of inter-word edges might not be available. For these reasons we shift focus to determining how data moves through the gate-level netlist. Reverse Engineering Data Buses (REBUS) finds datapaths, by utilizing the forward propagation method from WordRev [60] to assist in producing high level netlist descriptions. Prior method relied on a library of known function types to help identify the type of data transference. This identification also provided the ability to group wire together by the word data they contained.

REBUS uses the same scores generated by the logic classification method, and along with the concept of forward propagation in [60] tries to extract the data path in a netlist. The method could, due to the reduced number of comparisons, have a significantly better run time than the original logic method. This chapter will like the methods of Chapter 5 examine both the time and accuracy of the method on various types of netlists.

6.1.1 Signal Propagation

RELIC was designed to perform general classification of signals into either logic or data. RELIC did so by utilizing a pairwise comparison of signals to achieve a high confidence in the uniqueness of wire fan-in structures. The shear volume of the operations needed to completely classify just medium sized netlist causes RELIC to have poor a runtime performance for the simple task it performs. However, other approaches in research take advantage of a common theme across IC design that can reduce the potential number of function calls for comparing wires.

Algorithm 3 Determine the word sets of a netlist with a set of signals S, given a similarity score threshold, t, a similarity score depth d, a set of input words W, and a similarity score function f.

function BUSBASEDPARTITION(S, t, d, W)
  $words \leftarrow W \cup \{\{x\} | \forall w \in W(x \in S \wedge x \notin w)\}$
  $q \leftarrow \emptyset$
  for $w \in W$ **do**
    for $x, y \in w \wedge x \neq y$ **do**
      $q.append((x, y))$
    end for
  end for
  for $pair \in q$ **do**
    $x \leftarrow pair.first$
    $y \leftarrow pair.second$
    for $x_o \in fanout(x)$ **do**
      for $y_o \in fanout(y)$ **do**
        if $(f(x_o, y_o, d, \mathbb{T}) > t \vee f(x_o, y_o, d, \mathbb{F}) > t)$ **then**
          $X \leftarrow X \cup \{y\}$
          $w_x \leftarrow (w \in words \wedge x_o \in w)$
          $w_y \leftarrow (w \in words \wedge y_o \in w)$
          for $x_1 \in w_x$ **do**
            for $y_1 \in w_y$ **do**
              $q.append((x_1, y_1))$
            end for
          end for
          $words \leftarrow words \backslash \{w_x\}$
          $words \leftarrow words \backslash \{w_y\}$
          $words \leftarrow words \cup \{w_x \cup w_y\}$
        end if
      end for
    end for
  end for
  return $words$
end function

The comparison reduction comes from the abstract concept of a databus or datapath that moves information from one part of a netlist to another. Taking this concept into consideration REBUS is able to outperform the original RELIC in terms of speed. Aside from the standard inputs required from the original logic classification method, REBUS needs a set of input words to seed the word set. An extension to the method could leverage a set of output words to potentially find correct word pairs starting from the output with the use of backwards propagation. Regardless the method adds in the pairs of known word signals to a queue. While there are unresolved pairs within the queue the method will try to find new word pairs. For forward propagation the fan-outs are examined from the known word pair.

Algorithm 4 Determine the word sets of a netlist with set of signals S, given a similarity score threshold, t, a similarity score depth d, and a similarity score function F.

```
1:  function PairwiseSimScore(S, t, d)
2:      words ← ∅
3:      seen ← ∅
4:      for x ∈ S ∧ x ∉ seen do
5:          seen ← seen ∪ {x}
6:          X ← ∅
7:          for y ∈ S ∧ y ∉ seen do
8:              if (F(x, y, d, T) > t ∨ F(x, y, d, F) > t) then
9:                  seen ← seen ∪ {y}
10:                 X ← X ∪ {y}
11:             end if
12:         end for
13:         words ← words ∪ X
14:     end for
15:     return words
16: end function
```

6.1.2 Signal Comparison

When two wires are chosen for examination each pair of wires within the fan-out are compared, and if the signals under-inspection do not belong to the same word, the score between the two wires in question is evaluated. Since some registers could be storing the complement of the signal the comparison is done within a broad negation. If the resulting score is above some threshold, the

words of the correpsonding signals are merged. Any new signal pairs from the created word will be added to the queue, and evaluation will continue. The pseudo code for the process can be seen in Algorithm 3.

6.2 Results

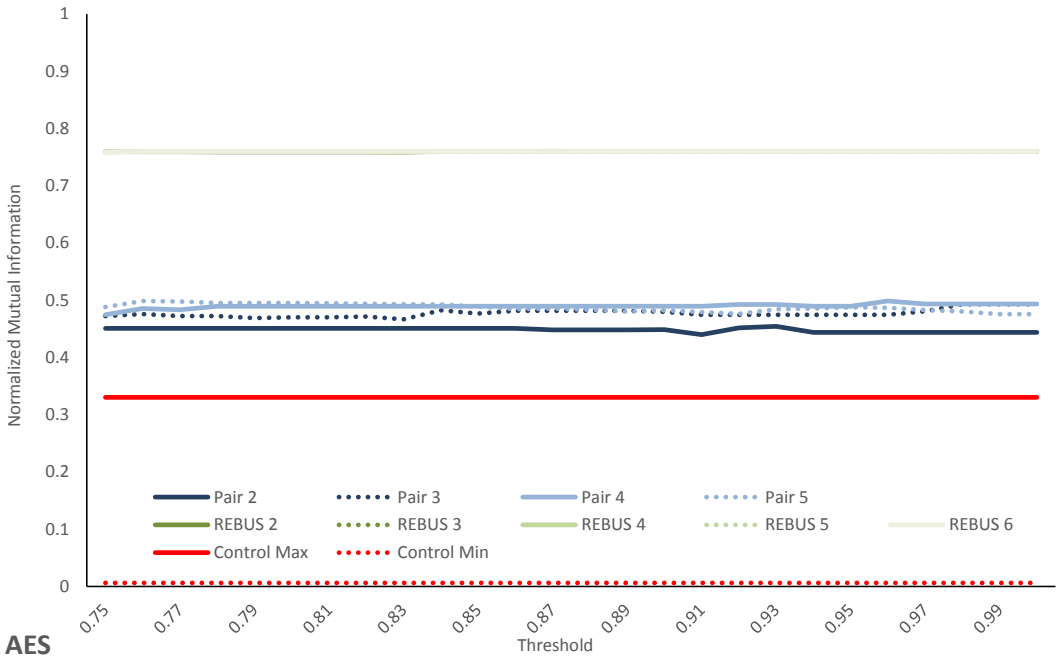


Figure 6.1: Cluster Scores found using pairwise method and REBUS compared to control signal based matching on the flattened AES-128 netlist.

To emphasize the difference of quality between REBUS and RELIC in terms of word extraction RELIC is extended to allow for word extraction. The RELIC extension will be referred to as the pairwise partition, or pairwise method, throughout this section. The pairwise method was created by using a near pairwise comparison of signals, while utilizing a representative signal to seed words. Direct pairwise comparisons can be quite slow, but they should be the most thorough. The work flow for the proposed partitioning scheme is as follows. A signal that has not been grouped

to a word yet will be selected at random as a seed signal. For each signal within some specified threshold of the seed signal, that has not already been added to a different word, we will add it to a current word. Then the current word is added to the set of known words. Since sometimes a signal might store the negation of the word and leverage the not pin of a register, we will allow comparison to be within some negation of the original signal. The process can be seen by algorithm 4.

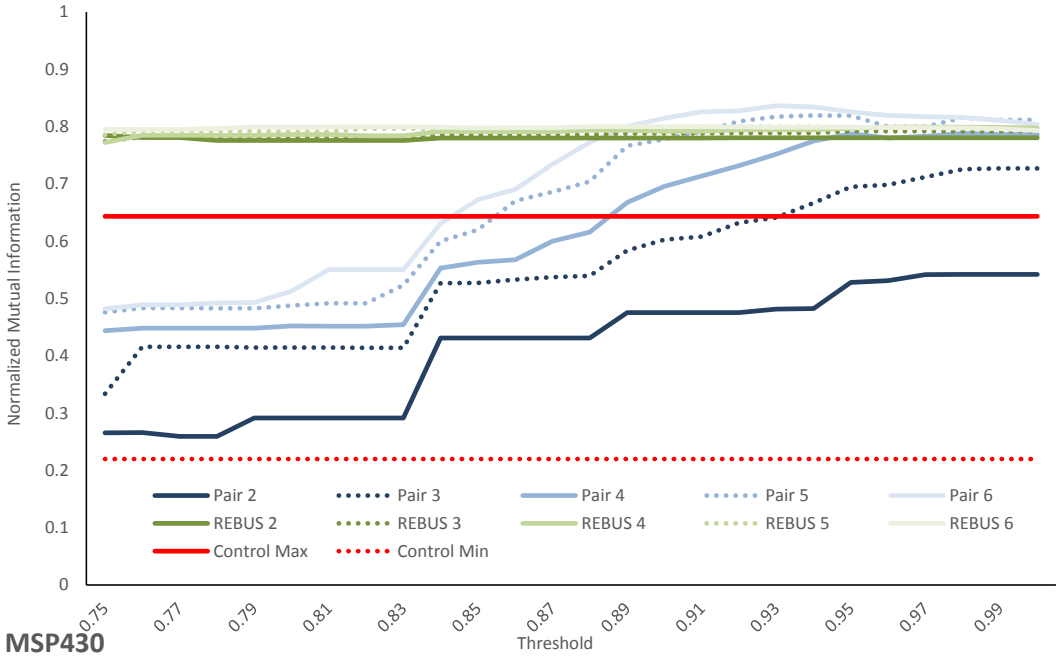


Figure 6.2: Cluster Scores found using pairwise method and REBUS compared to control signal based matching on the MSP430 netlist.

On the results for the AES core, seen in Figure 6.1, both the pairwise partition's and REBUS's NMI were highly stable; neither vary much with the given threshold. REBUS had a better result than the RELIC's pairwise comparison based scheme, and both methods were capable of outperforming the control signal based partitioning. The pairwise comparison method on MSP430 in Figure 6.2 had results that varied on both the threshold and the depth parameter. The higher the depth or the higher the threshold the better the performance, and although REBUS had a higher performance on average, the pairwise method had the best performance for a certain parameter combination.

Also seen in Figure 6.2 both the pairwise comparison method and REBUS were capable of outperforming the control signal based method. However, RELIC's pairwise comparison scheme only did so with certain parameter selection.

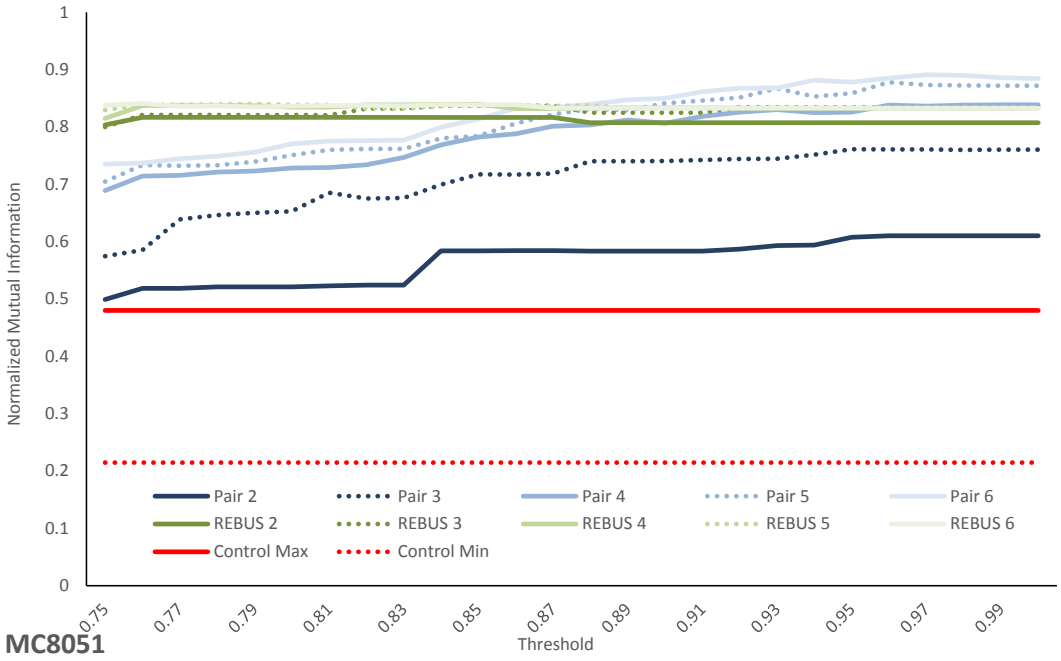


Figure 6.3: Cluster Scores found using pairwise method and REBUS compared to control signal based matching on the MC-8051 netlist.

The MC-8051 core also showed that REBUS and the simple pairwise comparison based partitioning schemes could outperform the control signal baseline (see Figure 6.3). It should be noted that once again REBUS was more consistent, but with certain parameters logic classification was able to overtake the bus scheme in terms of NMI. However, in the last netlist, the RSA core, both REBUS and pairwise RELIC had lower maximum NMIs than the control signal based method (see Figure 6.4). Not only was control signal based partitioning better, but unlike in the other three netlists REBUS always achieved a lower NMI than the pairwise RELIC method. In short, on the small RSA control signal based partitioning performed the best. This might be caused in part by a lack of repetitive structure due to the simplicity/size of the design.

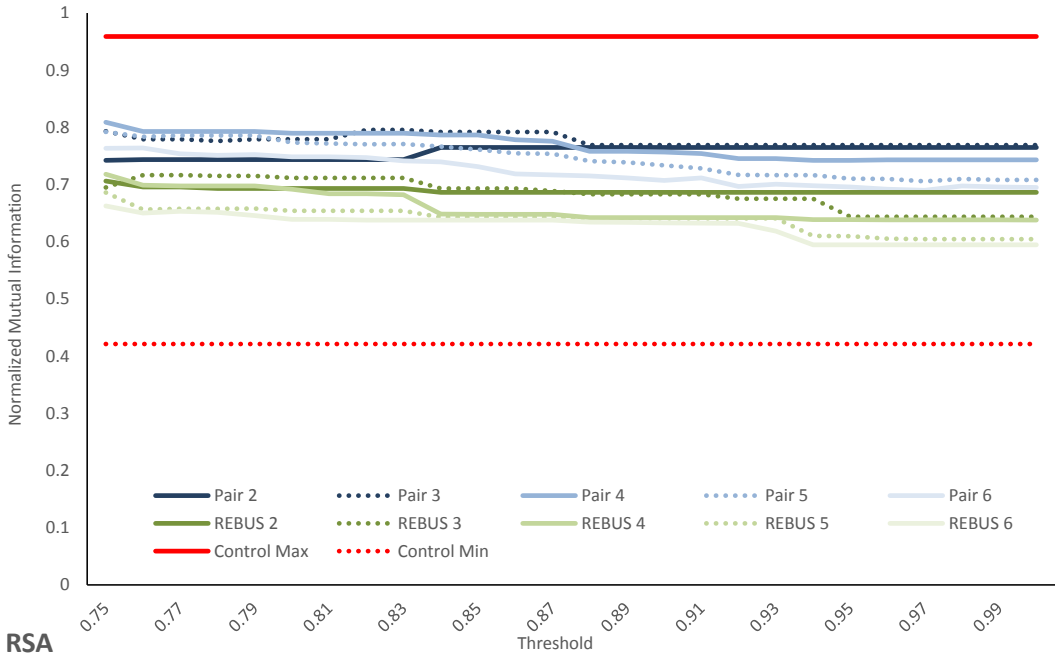


Figure 6.4: Cluster Scores found using pairwise method and REBUS compared to control signal based matching on the RSA netlist.

By far the fastest method found was to be the control based signal partitioning, see Table 5.1. It was capable of running over 100 times faster than any of the other methods on certain netlists and no netlist took longer than a second to execute. While the bus based method might not be as accurate, it is capable of running in a fraction of the time compared to slower methods such as the pairwise comparison, in part due to its limited comparisons, see Table 6.1. Due to memory issues the pairwise comparison based method had a very poor performance on the AES core.

6.3 Discussion

REBUS is a very efficient method in terms of runtime and accuracy. It is an unsupervised method capable of estimating the word-level partition. REBUS tends to work best for netlists that are of a moderate size. The smaller the netlist the more difficult it can be for unsupervised methods that

try to match. However, since REBUS utilizes a method that looks for words, it will not be able to partition netlists with varying degree of hierarchical structure. For higher levels of partition PCA based methods would probably be best due to the ability to change the expected number of clusters, which could help merge words from the same module.

Table 6.1: Average time taken for our netlist partitioning methods shown alongside the number of FFs and the GT's entropy and word count

Depth	2	3	4	5	6	FF Pins	Entropy	Words
AES Pair	146s	385s	1291s	4578s	-	6720	5.36	405
AES REBUS	11.4s	9.88s	11.5s	15.1s	19.6s	6720	5.36	405
AES PCA	23.9s	23.8s	23.9s	23.8s	23.9s	6720	5.36	405
MSP430 Pair	4.16s	5.49s	6.82s	7.91s	8.98s	734	4.33	133
MSP430 REBUS	0.30s	0.39s	0.48s	0.56s	0.649	734	4.33	133
MSP430 PCA	0.90s	0.91s	0.89s	0.88s	0.93s	734	4.33	133
MC8051 Pair	1.30s	1.97s	2.65s	3.19s	3.82s	578	4.47	121
MC8051 REBUS	0.33s	0.38s	0.42s	0.54s	0.68s	578	4.47	121
MC8051 PCA	1.44s	1.40s	1.53s	1.49s	1.50s	578	4.47	121
RSA Pair	1.29s	1.88s	2.45s	2.64s	2.84s	295	2.53	16
RSA REBUS	0.63s	0.70s	0.70s	0.63s	0.68s	295	2.53	16
RSA PCA	0.83s	0.86s	0.87s	0.88s	0.83s	295	2.53	16

CHAPTER 7: REVERSE ENGINEERING FINITE STATE MACHINES

Many sequential hardware Trojans can be represented as and FSM. Moreover a Trojan developer might use part of an already existing FSM to aid in the trigger. To help verify the integrity of a gate-level netlist we propose a tool to recover the full logic FSM. With the aid of an experienced IC user the netlist can be examined in a more meaningful manner than otherwise would be possible. To this end a software executable was created which is referred to as Reverse Engineering Finite State Machines abbreviated REFSM.

7.1 Methods

REFSM attempts to recover the control logic from a gate-level netlist and present to the user a higher-level description. A general outline of REFSM is shown in Figure 7.1. The netlist is first collected either from chip level reverse engineering or from the IP provider. The end user is then required to initiate the process and modify the recursion depth if run-time becomes an issue. Since designs can contain hundreds of thousands of gates or more, the first step is to reduce the number of gates to be analyzed by identifying and isolating FSM registers.

7.1.1 Logic Graph and State Registers

REFSM starts by creating the logical graph from a flattened netlist. The graph contains edges from inputs/registers to registers/outputs. Since REFSM determines the potential states of the registers, the outputs will not be considered. Any logic that is output exclusive is removed from the graph. What remains is logic from inputs and registers that can affect other registers either directly (register at time t can vary from register state/input at time $t-1$) or indirectly (register at

time t may vary based on register state/input at time $t - k$, where $k > 1$). Potential state registers can be determined using the heuristic algorithms proposed in any one of the following [67, 64, 6].

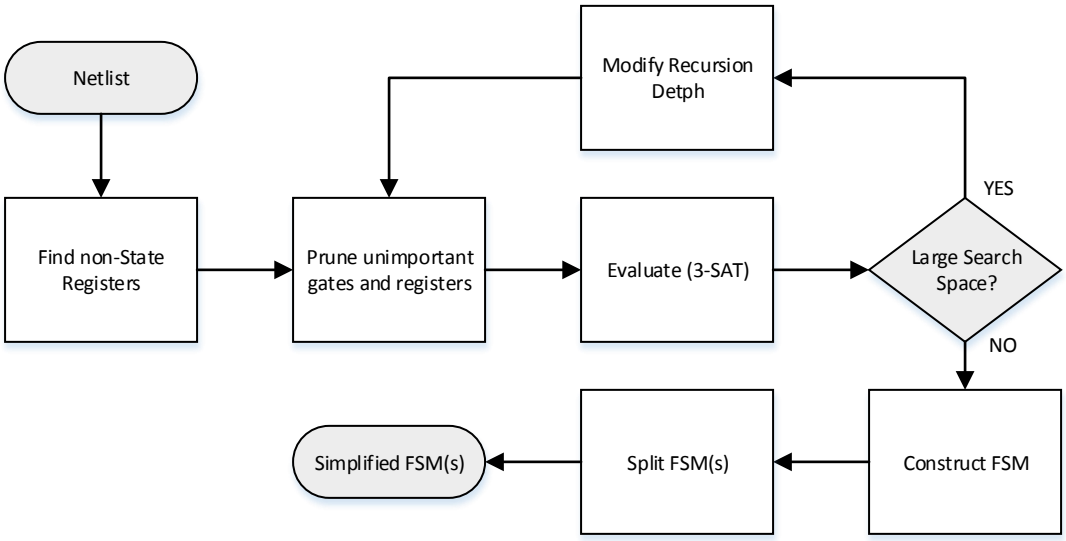


Figure 7.1: The flow of the REFSM method.

7.1.2 Prune Graph

As mentioned earlier pruning out registers helps reduce the time for FSM extraction. The call to remove registers is tough, so all registers that affect logic registers (directly or indirectly) are considered important. Only if the amount of possible states becomes too large, REFSM will prune some potentially less important registers. Our implementation considers both '0' and '1' (or Don't Care (DC) typically denoted as X) as potential values for each "unimportant" register. Checking and storing each one of these can take time, but certain assumptions about the graph can also reduce the number of states that need to be considered.

The aforementioned pruning process involves a Breadth First Search (BFS) through the netlist up to a maximum distance of δ from the set of state registers. This precomputation is used to produce a smaller subset of the netlist, which allows for an estimated register state graph in a reasonable amount of time and memory usage. However, in case that the current δ still causes program problems, δ will be decreased by user to run the algorithm again. The δ reduction process is performed until a state register graph is produced. Analysis can then be performed on the resulting graph to recover control flow and/or to detect malicious logic.

7.1.3 Evaluate State Space

After generating a pruned graph, REFSM searches for all possible states of registers that are achievable by using the function GetRegisterStates (see Algorithm 5 in the Appendix). The given netlist is represented by a set of Boolean logical expressions, $EXPS$, and a set of false and true values ('0' and '1') to represent each state that the registers can take on. The only registers that are listed in each state are those which were determined to be important in the prune step. The queue is initialized with the reset state ($resetState$). Meanwhile, the set of seen states (N) also contains the reset state to prevent reusing it again. By looping through all elements in the queue all possible register states are generated. A single iteration starts by pulling out the first element in the queue. A new set of expressions is generated by filling in all the values currently in the register state. As an example, if the register is set to be true (value '1') in the current state, then when making the new expressions from the netlist all variables relying on the register's output will be recalculated accordingly. This new expression is sent into the 3-SAT function, FETCH, for evaluation and returns the set of all achievable register states using the given expression. The GETREGISTERSTATES function constructs an FSM graph by searching for any states not included in the graph, and then evaluating which states they can reach. Each new state is added both into the queue and into N. The overall run-time is $O(|N|^2 + |N|2^{\#inputs})$.

Algorithm 5 Find an FSM graph given a set of expressions $EXPS$ from a flattened netlist and a starting expression set $resetState$

```
1: function GETREGISTERSTATES(EXPS, resetState)
2:     Let FSM be an empty graph G(N, E)
3:     Add the resetState to the Queue; Set N to {resetState}
4:     while Queue ≠ ∅ do
5:         Get a currentState from Queue
6:         currentExp ← EXPS.LastState(currentState)
7:         F ← FETCH(currentExp)
8:         for nextState ∈ F do
9:             if nextState ∉ N then
10:                Queue.add(nextState)
11:                N ← N ∪ {nextState}
12:            end if
13:            E ← E ∪ {(currentState, nextState)}
14:        end for
15:    end while
16:    return FSM
17: end function
18: function FETCH(exps)
19:    if exps contains no variables then
20:        return {exps}
21:    end if
22:    x ← first variable in exps
23:    newExps ← exps.set(x, false)
24:    F ← Fetch(newExps)
25:    newExps ← exps.set(x, true)
26:    F ← Fetch(newExps) ∪ F
27:    return F
28: end function
```

As a key part of the function GETREGISTERSTATES, the FETCH function starts by checking the expression for unassigned variables. If there is a variable that has yet to be assigned and the variable can affect the outcome of the expression, the FETCH function will need to decide what value to use. Otherwise, it will return the expression as it is. If there were unassigned variables, the FETCH function will randomly pick one of them, set its value to '0', check the outcome recursively and add it into the resulting expressions. The function will then set the variable to '1', check the outcome, and add the resulting expression into the output. After going through all variables, the function will then return all identified states.

The complexity of the FETCH function operation is $O(2^n)$ in the worst case, where n is the number of variables that can change. In practice, due to the structure that many netlists follow, there are few variables that have a significant effect on the outcome of the next state. Most of the states searches terminate at a depth of 8 or less in our experimentation. This makes the number of visited states less than 256. Further, many of the inputs perform a similar function so if one is set to '1', the others no longer need to be checked. For example given x variables AND-ed or OR-ed together, the number of decisions that need to be made becomes $x+1$. Although the computational complexity of the Fetch function appears daunting, it normally can be run in a reasonable amount of time such that the total run-time for REFSM becomes very low (See Table 7.1).

7.1.4 FSM Decomposition

After deriving the global FSM, some extra steps for further analysis of the recovered control logic may be required. Determining simple transition conditions is one task that REFSM performs. This enables users to find suspicious transitions. A more important task is separating local FSMs from the global FSM, which is referred to as FSM decomposition and is described below.

For demonstration purpose, we consider the case that two independent FSMs (F_1 and F_2) were merged (composed). This results in pairs of states (α, β) of the merged FSM, where α is from F_1 and β is from F_2. Each pair of transitions that originate from the individual states should be traversable. The edges leaving the state (α, β) will contain at least the Cartesian product of the reachable states from state α and β. More formally

$$\{\alpha, \beta | \alpha \in V(F_1) \wedge \beta \in V(F_2)\} \subseteq V(F_1 \times_F F_2)$$

$$\{((\alpha_i, \beta_i), (\alpha_j, \beta_j)) | (\alpha_i, \alpha_j) \in E(F_1) \wedge (\beta_i, \beta_j) \in E(F_2)\} \subseteq E(F_1 \times_F F_2)$$

where $\times_F$ denotes composition of FSMs. Since the FSMs are independent, it should also be the case that any node within the merged FSMs should correlate to two FSMs in the original graph. Otherwise the original FSMs would not contain all possible reachable states. Similarly Any edge from the merge FSM must correlate to transistions from the original independent FSMs. We can then say the following

$$V(F_1 \times_F F_2) \subseteq \{\alpha, \beta | \alpha \in V(F_1) \wedge \beta \in V(F_2)\}$$

$$E(F_1 \times_F F_2) \subseteq \{((\alpha_i, \beta_i), (\alpha_j, \beta_j)) | (\alpha_i, \alpha_j) \in E(F_1) \wedge (\beta_i, \beta_j) \in E(F_2)\}$$

Using these results it can be inferred that the merged FSM will be the tensor product of the original FSMs.

It should be noted that there have been algorithms which can decompose the tensor products on undirected, unlabeled, connected graphs into unique prime factor decompositions (UPFD) in polynomial time [68]. However, to decompose a merged FSM involves directed graphs and appears to be a harder problem. Therefore a heuristic-based approach is used to take advantage of the register labeling to split the graph into UPFD.

The basic idea is to assume that each pair of registers is originally independent. Then look for contradicting sets of independent registers (either by vertex label or transition topology) and merge the found sets together until all register sets can properly construct the original FSM using their tensor product. Algorithm 6 lists the detailed description of the used algorithm.

Algorithm 6 Returns a partition of an FSM given a set of registers, R, and an FSM graph $G(N, E)$

```
1: function SPLITFSM(R, G(N, E))
2:     Let ℙ = {P_i | P_i is the Partition containing register i}
3:     Assume no register depends on a register other than itself.
4:     for i, j ∈ R such that P_i ≠ P_j do
5:         Let G_i(N_i, E_i) be the FSM dependent on i
6:         Let G_j(N_j, E_j) be the FSM dependent on j
7:         Let G'(N', E') be the FSM dependent on i and j
8:         if there exists u ∈ N_i and v ∈ N_j and (u, v) ∉ N' then
9:             P_i ← P_i ∪ P_j; P_j ← P_i
10:        else
11:            if there exists e ∈ E_i and l ∈ E_j and (e, l) ∉ E' then
12:                P_i ← P_i ∪ P_j; P_j ← P_i
13:            end if
14:        end if
15:    end for
16:    return ℙ
17: end function
```

7.2 Results

REFSM was capable of fulfilling many roles for the sake of reverse engineering; extracting high level details of a netlist's logic, finding Hardware Trojans within gate level netlists, and detereming unlocking sequences for sequentially encrypted circuits was carried out by REFSM. The following section shows its use.

7.2.1 Extracting Logic

In order to verify the effectiveness and the scalability of the developed REFSM tool, we applied the tool on various circuit designs ranging from small-scale ASIC designs to medium and large-scale microprocessors. As we will demonstrate shortly, the control logic within all these testing circuits are recovered successfully in the format of finite state machines. The experimental tests are run on a desktop with Intel i7 quadcore and 16GB memory. The average run-time for different circuits are listed in Table 7.1.

Table 7.1: REFSM Run-time Results

Netlist Name	Registers	Gates	Run-time
RS232 Transceiver	59	168	1 s
32-bit RSA	555	2139	$<$ 1 s
MC8051 μP	578	6590	39 s
SPARC μP	119911	232978	600 s

For small-scale and medium-scale circuits, our algorithm can reconstruct the circuit control logic from a flattened netlist in less than 1 minute (less than 1 second in most cases). The run-time is below 10 minutes even for large-scale circuits. From Table 7.1, we can also find that in general the REFSM would have a larger computation time for larger circuits. However, the complexity of the control logic will affect the computation time. For example, 32-bit RSA Encryption [48] circuit finishes faster than the smaller RS232 transceiver due to the RSA circuits more regular circuit structure.

The RS232 transceiver includes two sub-modules for data transmitting and data receiving. The sub-modules including the transmitter and the receiver work independently without interfering with each other. In addition, they have their own input/output pins at the top module. However, the flattened netlist does not maintain the circuit hierarchical structure and there is no clear boundary between them. Therefore, the selection of an RS232 circuit is ideal for verifying the capability of REFSM in isolating different FSMs from a flattened netlist.

Using the flattened RS232 netlist as the input, our REFSM tools recover the control logic in the format of FSM of the entire circuit. Figure 7.2 shows the recovered global FSM which contains 25 unique states with quite complicated transmission conditions among these states. This FSM, although containing the entire functionality of the RS232 circuit control logic, is almost meaningless to users and testers due to its complexity. However, the FSM decomposition component of REFSM can help simplify the FSM structure.

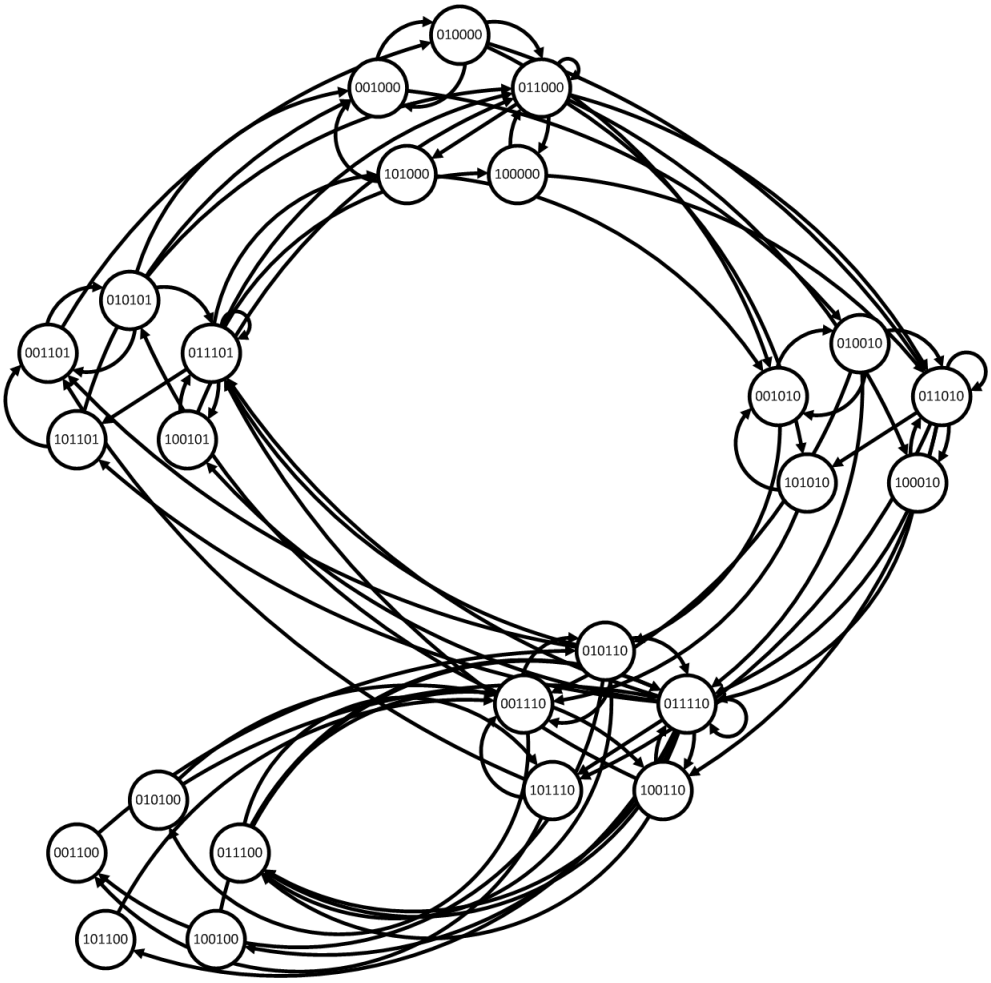
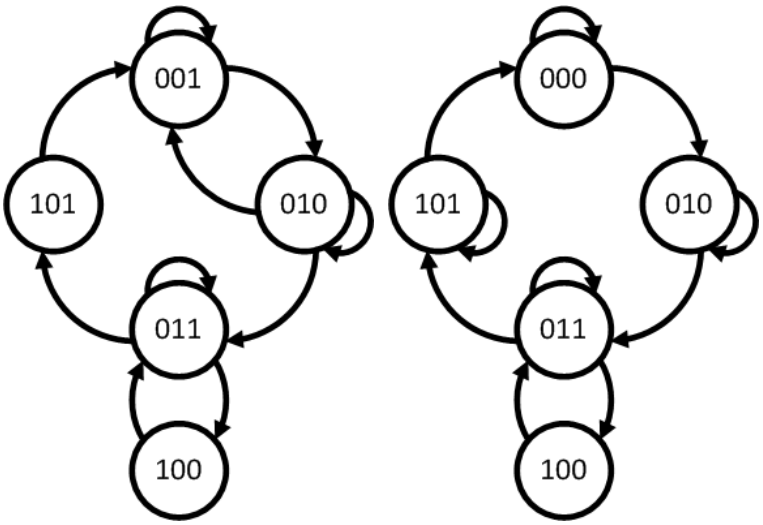


Figure 7.2: The extracted FSM from the RS232 transceiver.

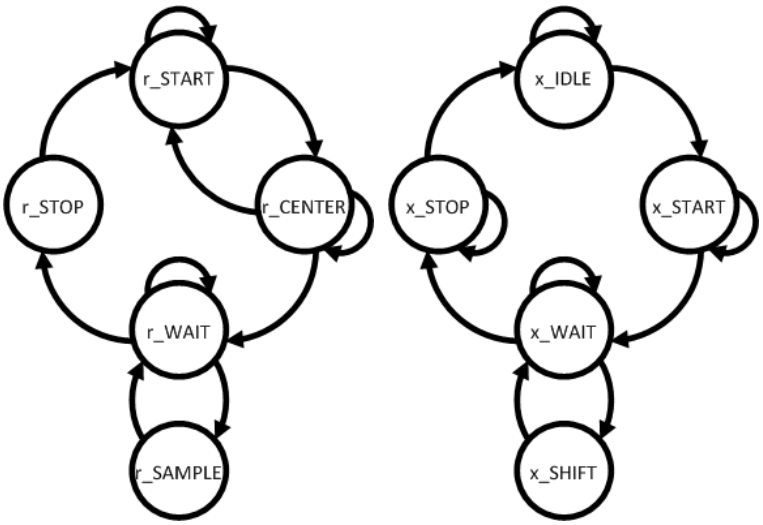
Using the recovered FSM in Figure 7.2, the developed FSM decomposition tool can isolate independent states from the entire FSM. In this case, two independent FSMs, Figure 7.3a and Figure 7.3b, are separated from the control logic in Figure 7.2. To validate the correctness of the FSM decomposition results, we build the real FSMs of the receiver and transmitter submodules in the RS232 circuit (see Figure 7.4a and Figure 7.4b) which are identical to the recovered FSMs both in available states and in all state transition conditions.



(a) First decomposed FSM. (b) Second decomposed FSM.

Figure 7.3: The two FSMs recovered from the RS232 netlist.

The reason we used the 8051 microprocessor is to show the potential of REFSM in dealing with a highly-complex circuit structure. The source code of the 8051 microprocessor is written in VHDL, where each instruction will take up to three clock cycles to complete [69]. Based on the RTL code, we first constructed the real FSM when dealing with different instructions (see Figure 7.5a). We then synthesize the circuit and generate the flattened netlist of the 8051 microprocessor. The flattened netlist is then used as the input of the REFSM, which then recovers the control logic from the netlist. The recovered netlist is shown in Figure 7.5b. A comparison between Figure 7.5a and Figure 7.5b shows us that these two FSMs are of the same structure. In fact, the transition conditions are also identical.



(a) The receiver FSM. (b) The transmitter FSM.

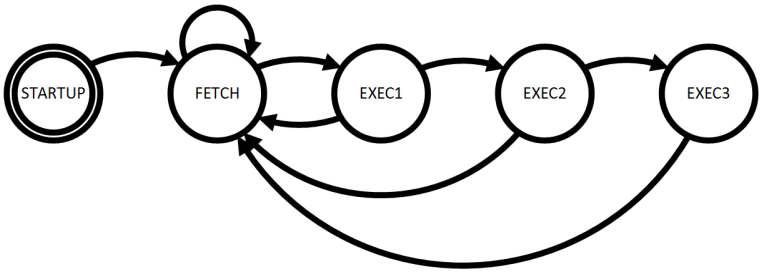
Figure 7.4: The two FSMs extracted from the RTL of the RS232 transceiver.

7.2.2 Trojan Detection

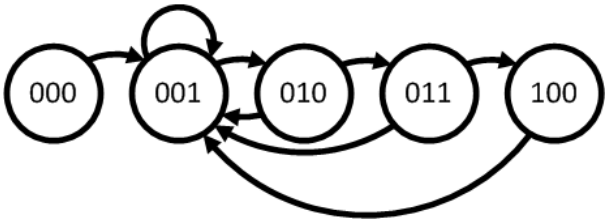
The capability of REFSM for control logic recovery can also help detect hardware Trojans which are triggered by a specific input sequence, so-called sequential Trojans. Compared to the hardware Trojans that rely on only combinational logic to be triggered, sequential Trojans are much more difficult to activate and can evade many hardware Trojan detection methods such as [70, 55, 54].

However, since the behavior of the sequential Trojan triggering mechanism can be modeled as an FSM with the specific input sequence serving as the transition conditions, REFSM can help rebuild and isolate the Trojan FSM. From this circuit users/testers can easily identify the Trojan logic as well as the Trojan triggering conditions.

For demonstration purposes, a Trojan-infected cryptographic platform is used [46]. The platform is an FPGA implementation designed to perform all necessary operations for cyphertext transmissions through public channels. The user inputs data via a keyboard attached to a PS2 interface.



(a) The RTL FSM.



(b) The REFSM FSM.

Figure 7.5: The FSM Recovered from MC8051 Netlist and RTL.

This text is displayed through a VGA port onto an attached monitor. The user then initiates the encryption of the data entered via a button on the FPGA board. The encryption used is an 128-bit AES encryption core; the user also has the ability to select up to 16 different encryption keys by changing a combination of four switches on the FPGA before initiating the encryption sequence. Once encryption is finished, the user can then send the encrypted data through an on-board serial port.

In this design a Trojan was inserted in the top level module that uses a finite state machine to read a specific input sequence from the user, via the keyboard. Once the sequence is entered, the activated hardware Trojan will leak the AES encryption key through the serial port. The Trojan trigger seems simple, but this hardware Trojan can evade many detection methods [70].

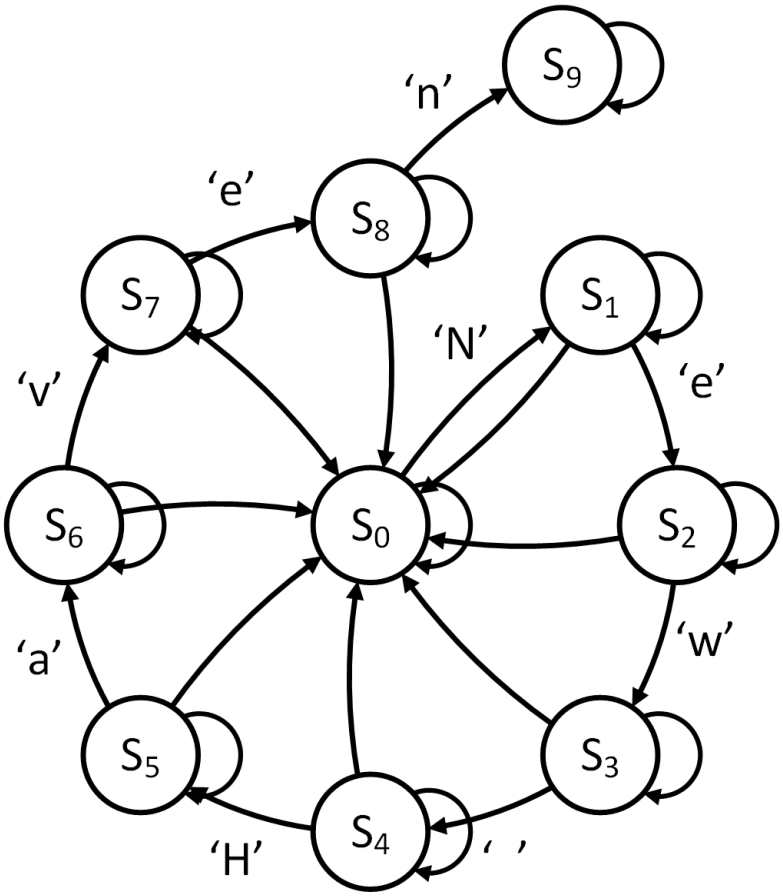


Figure 7.6: The extracted Trojan Logic from the case study.

However, if we can identify all states of the Trojan FSM, determining the the actual behavior of the Trojan becomes apparent. Using the state space exploration techniques presented, all FSM states and transitions were correctly identified by the REFSM, as well as the correct conditions of the inputs for each transition. State diagrams were constructed of the edge-lists for the recovered FSMs. Figure 7.6 shows the recovered FSM of the inserted hardware Trojan and its triggering conditions. The letter on each transition curve shows the keyboard input which will enable the transition among these states. While the REFSM tool will not tell us whether the recovered FSM is genuine or malicious, users/testers can easily identify the suspicious logic and conclude that the

special input sequence, ‘New Haven’ in this case, is outside the design specification and therefore potentially a hardware Trojan trigger. Users may validate their findings by triggering the suspicious circuit by inputting the special sequence.

Besides the elaborate example, we also applied our solutions to the hardware Trojan benchmarks from Trust-Hub [48]. Table 7.2 shows some of the testing results from which we can find that the REFSM tool can help detect hardware Trojans with sequential trigger and/or sequential payload in seconds.

Table 7.2: Run-Time and Trojan Detection Capability on Trust-Hub Benchmark

Benchmark	Trigger	Recovered?	Run-time
AES-T100	Always On	Recovered	18 s
AES-T400	Plaintext = 128’hffffffffffffffffffffffffffffffff	Recovered	$<$ 1 s
AES-T800	Plaintext = 1) 128’h3243f6a8885a308d313198a2e0370734 2) 128’h00112233445566778899aabbccddeeff 3) 128’h0 4) 128’h1	Recovered	$<$ 1 s
b15-T400	Address = 8’hFF	Recovered	$<$ 1 s
s38584-T100	Scan Enable Mode	Recovered	$<$ 1 s
MC8051-T200	pcon (control_mem) = 1’b1	Recovered	90 s

7.2.3 Unlocking FSMs

Sequential encryption schemes often focus their efforts on the implicit logical FSM. The straightforward approach increases the FSM’s state space thereby reducing access of the original FSM. In addition access to the circuit’s true logic can require a particular sequence of input vectors. Other methods incorporate special locking states in the updated FSM. These methods select a subset of states that can be accessed by the new reset state but cannot reach the states of the original FSM.

Two main methods for increasing the state space exist. The first method changes the logic of the registers to utilize previously unreachable states. The other method involves inserting additional registers that usually but not necessarily act as flags for the FSM's behavior. Additional registers tend to be very appealing since the state space increase exponentially with the number of inserted registers. The major detriments to a large number of register insertions is the time to unlock, area, and power overhead.

An example of sequential circuit encryption, HARPOON, inserts additional state elements (SE) and combinational logic that adversely affects the behavior of the netlist while the circuit is locked. The inserted SEs control the activation of the inserted combinational modules, that have the potential to corrupt parts of the netlist. Moreover HARPOON's FSM's state space is partitioned into three general sections (modes): obfuscation, authentication, and original. The obfuscation mode, the first part of the obfuscation mode, corrupts parts of the netlist. The authentication mode simply watermarks the netlist. The original mode, as it sounds, does not corrupt the netlist's internal signals and allows for normal execution. The authors assume that an attacker would randomly reverse engineer the netlist which gives the defender a large probability of protection, but a smarter solution exists based on their protection method.

Attacking the HARPOON protection requires first identifying the registers associated with the netlist's mode control. In general finding inserted registers partially reveals the function of the chip's logic. Several techniques can be used to extract these registers.

The first method that can be used was a register classification tool, RELIC [67]. RELIC itself is a tool used to separate parts of the netlist based on implicit features that are induced when including either extra logic or circuitry. RELIC finds repetitive wire patterns by examining the correlation of a wire's structural variables (e.g. fan in size, distance to input/output wires, etc.). The outlying wires tend to fall into the category of logic due to the nature of how netlists are synthesized (i.e. a

fixed protocol replicates structure within data words). RELIC might not be capable of finding all the inserted registers in one try. To compensate RELIC is used to find partial register sets, and the sets expand via register dependency.

The second method stems from the register set expansion technique mentioned in the first method. With the process RELIC itself is removed from the equation, and the register dependency becomes the sole method for "classification". This is done by way of Tarjan's Strongly Connected Components (SCC) algorithm [71]. The algorithm finds what is commonly referred to as the transitive closure of directed graphs. The algorithm and properties are well detailed in other resources. The graph returned contains a set of vertex sets that represent SCCs of the original graph, which is potentially connected by a set of directed edges that denote how the original graph's components interact. The graph itself is directed and acyclic (see Figure 7.7).

The Strongly Connected Component graph can also be used to attack more recent protection schemes such as DSD (Dynamic State-Deflection. DSD [37] relies on inserted, persistent logic that is unaffected by the original logic (or original data for that matter). Thus Tarjan's algorithm can detect these inserted state flip-flops. When observing the FSM and the transition probability generated by these inserted FFs the correct state becomes obvious. In general the components that are analyzed are those that contain no incoming edges (i.e. source SCCs). Source SCCs will exist because the graph is acyclic (and presumably non-empty).

Once found, the inserted registers are used by the REFSM tool to construct a partial FSM of the netlist. For protection schemes such as HARPOON the desired FSM section (i.e. original mode) is the authentication sequence's "end". The end is found using Tarjan's SCC algorithm. The FSM is broken down into its components, and the component(s) without outgoing edges (i.e. sink SCCs) are analyzed. If multiple sinks exist, the one selected is typically the component that has the lowest reachability probability, as the others are probably black-hole states (i.e. states that exist

to trap incorrect sequences). These black-hole states are typically included in other protection methods. REFSM then can be used for ATPG by generating the shortest input sequence to enter a state within the supposed normal mode FSM. This test pattern can be used to verify REFSM's capability to unlock FSM locking techniques.

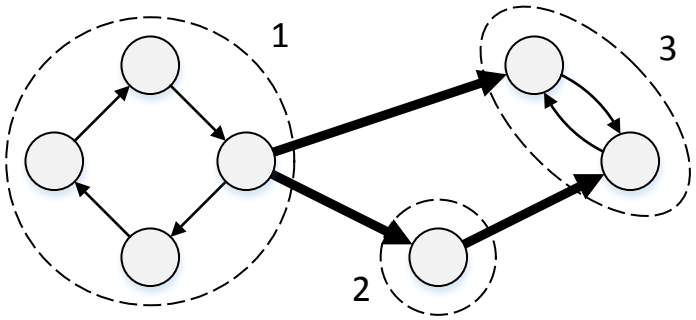


Figure 7.7: A graph which is partitioned into its three SCCs. The first being the only source SCC, and the third being the only sink SCC.

The best chance a user has at improving HARPOON without overhauling the method is to increase the complexity of the FSM. With a large enough FSM it becomes infeasible to extract the unlocking sequence. The major concern with this approach is the incurred overhead. Aside from power and area increase due to the increased number of SEs, the major drawback is the time the circuit takes to unlock from power-on.

Alternatively users can incorporate other defense techniques. Although this would also not necessarily ensure protection, it would definitely make reverse engineering even more difficult for adversaries. A typical defense that has been prevalent in current research is the use of gate camouflaging. Even though methods exist that can break standard gate camouflaging, the mixture of methods can slow down or even halt IP piracy.

7.3 Discussion

This paper proposed and evaluated a method for reverse engineering the control logic from a gate-level netlist. The algorithm designed and implemented showed promising results with reasonable run time on standard desktop computer hardware. For every test, all states were successfully identified along with their correct state transitions and conditions leading to near perfect FSM reconstruction. In addition, the developed tool helps identify sequential hardware Trojans which, otherwise, would be very difficult to detect through existing testing methods. We expect that the developed tool will be widely implemented in other hardware security areas. Nevertheless, one shortcoming of the developed tool stems from the fact that all tests were to compare the FSM implemented in RTL source code with the recovered FSM to verify the correctness. Manually analyzing the RTL source code to construct a FSM can lead to possible errors or incomplete state spaces. Therefore, more work can be done to automate the verification process for the REFSM tools.

CHAPTER 8: CONCLUSION

Due to the state of the IC production chain and its economy the threat of hardware Trojans has grown significantly over the the past few years. Many different methods for implementing and inserting hardware Trojans are available to malicious IC developers. The task of determining the purity of IP cores is left to the consumer. Prevention and detection methods exist, but many of them are lacking and detection has begun to rely more on reverse engineering techniques. Full function recovery has become a major goal for Trojan detection. However, the sub-problems associated with full function recovery has not been well defined, documented, or analyzed. To this end we propose several heuristic based methods to assist in furthering the pursuit of hardware Trojan detection and prevention by solving common sub-problems which can be associated with full function recovery. We also automate these method and test them against common gate-level netlist both with and without hardware Trojans.

Distinguishing logic from data can be considered a highly sought after task when searching for hardware Trojans. Since many Trojan designs involve manipulation of a FSM, determining the logic of a netlist can narrow down the search for the most common types of Trojans. To address this concern we proposed a tool called RELIC that used a recurrent comparison on gate-level fan-in structure to determine the similarity between different signals within a netlist. RELIC then sums a threshold function of the returned scores and uses these sums to pseudo classify logic versus data. By leveraging various dynamic programming practices RELIC was able to achieve a reasonable runtime.

A reverse engineering problem tangent to logic signal classification is that of signal partitioning. Methods that rely on word-level or module-level information to extract high level details motivate researchers to improve partitioning methods, but with limited research that analyzes the accuracy

of partitioning, the area has somewhat stagnated. We develop our own methods for partitioning netlists, REPCA, and by way of NMI, a common clustering evaluation technique, we are able to determine the efficiency of such methods. The resulting partitions were then compared against another notable method for partitioning by way of NMI. Along the same thread, REBUS, a method for extracting high-level features using databuses, was developed and similarly compared with known partitioning schemes. To more directly display the use of REBUS over similar schemes, we compared its result and time to that of a pairwise, brute-force RELIC solution. Both REPCA and REBUS were capable of outperforming the previously proposed method especially on larger netlists or netlists with a greater degree of data.

The last major reverse engineering problem addressed in this thesis was that of high-level logic extraction. Since many Trojans emulate the logic of a FSM, we developed a method, REFSM, that, when given a netlist, a library description, and a desired word, returns the netlist's behavior with respect to the word in the form of a FSM. To help reduce the complexity of the resulting FSM, as some FSMs might be the composition of multiple independent FSMs, REFSM utilizes a heuristic to decompose the FSM into smaller pieces, which can allow for a more accurate analysis. To show the effectiveness of the method we used it to extract several FSMs from Trojan infected netlists, and we found that REFSM was able to accurately extract the Trojan transitions. Further, in netlists without Trojans REFSM accurately extracted the full FSM even when the netlist was locked using simple FSM based obfuscation. The deobfuscation leads us to believe that hardware locking researchers needs to take into consideration stronger attack models when developing their solutions.

In conclusion this thesis has proposed four methods for gate-level netlist function recovery. The first method presented was RELIC a broad classification tool that leveraged the pairwise structural information to find logic. The second method REPCA used generic signal structure to create a partition of the netlist. The third method REBUS utilized forwarded propagation techniques and

combined it with the similarity score function from RELIC to extract data flow from the netlist. Logic was recovered in a high level description from the netlists which was of the form of a FSM in the last tool discussed. These methods will hopefully motivate researchers to produce better reverse engineering methods, whose effectiveness can be demonstrated through the use of the presented analysis techniques.

LIST OF REFERENCES

[1] R. S. Chakraborty, S. Narasimhan, and S. Bhunia, "Hardware trojan: Threats and emerging solutions," in *High Level Design Validation and Test Workshop, 2009. HLDVT 2009. IEEE International*. IEEE, 2009, pp. 166–171.

[2] J. Dworak, M. R. Grimaila, S. Lee, L.-C. Wang, and M. R. Mercer, "Modeling the probability of defect excitation for a commercial ic with implications for stuck-at fault-based atpg strategies," in *Test Conference, 1999. Proceedings. International*. IEEE, 1999, pp. 1031–1037.

[3] M. Tehranipoor, H. Salmani, and X. Zhang, "Hardware trojan detection: Untrusted third-party ip cores," in *Integrated Circuit Authentication*. Springer, 2014, pp. 19–30.

[4] R. J. Hillman, "Dod supply chain: Preliminary observations indicate that counterfeit electronic parts can be found on internet purchasing platforms," DTIC Document, Tech. Rep., 2011.

[5] S. K. Haider, C. Jin, M. Ahmad, D. M. Shila, O. Khan, and M. van Dijk, "Hatch: Hardware trojan catcher." *IACR Cryptology ePrint Archive*, vol. 2014, p. 943, 2014.

[6] Y. Shi, C. W. Ting, B.-H. Gwee, and Y. Ren, "A highly efficient method for extracting fsms from flattened gate-level netlist," in *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*. IEEE, 2010, pp. 2610–2613.

[7] M. Hicks, M. Finnicum, S. T. King, M. M. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE, 2010, pp. 159–172.

[8] C. Sturton, M. Hicks, D. Wagner, and S. T. King, "Defeating uci: Building stealthy and malicious hardware," in *Security and Privacy (SP), 2011 IEEE Symposium on*. IEEE, 2011, pp. 64–77.

[9] W. Li, Z. Wasson, and S. A. Seshia, "Reverse engineering circuits using behavioral pattern mining," in *Hardware-Oriented Security and Trust (HOST), 2012 IEEE International Symposium on*. IEEE, 2012, pp. 83–88.

[10] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascón, W. Y. Tan, A. Tiwari, N. Shankar, S. A. Seshia, and S. Malik, "Reverse engineering digital circuits using structural and functional analyses." *IEEE Trans. Emerging Topics Comput.*, vol. 2, no. 1, pp. 63–80, 2014.

[11] E. Tashjian and A. Davoodi, "On using control signals for word-level identification in a gate-level netlist," in *Proceedings of the 52nd Annual Design Automation Conference*. ACM, 2015, p. 78.

[12] A. Das, G. Memik, J. Zambreno, and A. Choudhary, "Detecting/preventing information leakage on the memory bus due to malicious hardware," in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2010*. IEEE, 2010, pp. 861–866.

[13] G. Bloom, B. Narahari, R. Simha, and J. Zambreno, "Providing secure execution environments with a last line of defense against trojan circuit attacks," *computers & security*, vol. 28, no. 7, pp. 660–669, 2009.

[14] R. W. Jarvis and M. G. McIntyre, "Split manufacturing method for advanced semiconductor circuits," Mar. 27 2007, uS Patent 7,195,931.

[15] M. Beaumont, B. Hopkins, and T. Newby, "Safer path: Security architecture using fragmented execution and replication for protection against trojaned hardware," in *Proceedings*

of the Conference on Design, Automation and Test in Europe. EDA Consortium, 2012, pp. 1000–1005.

[16] R. Kalayappan and S. R. Sarangi, "Seccheck: A trustworthy system with untrusted components," in *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*. IEEE, 2016, pp. 379–384.

[17] Http://www.chipworks.com/.

[18] F. Courbon, P. Loubet-Moundi, J. J. Fournier, and A. Tria, "Semba: A sem based acquisition technique for fast invasive hardware trojan detection," in *Circuit Theory and Design (ECCTD), 2015 European Conference on*. IEEE, 2015, pp. 1–4.

[19] Https://www.scienceexchange.com/.

[20] J. Grand, "Printed circuit board deconstruction techniques." in *WOOT*, 2014.

[21] C. Bao, D. Forte, and A. Srivastava, "On application of one-class svm to reverse engineering-based hardware trojan detection," in *Quality Electronic Design (ISQED), 2014 15th International Symposium on*. IEEE, 2014, pp. 47–54.

[22] M. Cason and R. Estrada, "Application of x-ray microct for non-destructive failure analysis and package construction characterization," in *Physical and Failure Analysis of Integrated Circuits (IPFA), 2011 18th IEEE International Symposium on the*. IEEE, 2011, pp. 1–6.

[23] A. Beit-Grogger and J. Riegebauer, "Integrated circuit having an active shield," Nov. 8 2005, uS Patent 6,962,294.

[24] Q. Shi, N. Asadizanjani, D. Forte, and M. M. Tehranipoor, "A layout-driven framework to assess vulnerability of ics to microprobing attacks," in *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 155–160.

[25] M. Alam, H. Shen, N. Asadizanjani, M. Tehranipoor, and D. Forte, "Impact of x-ray tomography on the reliability of integrated circuits," *IEEE Transactions on Device and Materials Reliability*, vol. 17, no. 1, pp. 59–68, 2017.

[26] H. Dogan, M. M. Alam, N. Asadizanjani, S. Shahbazmohamadi, D. Forte, and M. Tehranipoor, "Analyzing the impact of x-ray tomography on reliability of integrated circuits," in *ISTFA 2015 Proceedings from the 41st International Symposium for Testing and Failure Analysis*, 2015, pp. 1–10.

[27] M. Holler, M. Guizar-Sicairos, E. H. Tsai, R. Dinapoli, E. Müller, O. Bunk, J. Raabe, and G. Aeppli, "High-resolution non-destructive three-dimensional imaging of integrated circuits," *Nature*, vol. 543, no. 7645, pp. 402–406, 2017.

[28] M. Pacheco and D. Goyal, "X-ray computed tomography for non-destructive failure analysis in microelectronics," in *Reliability Physics Symposium (IRPS), 2010 IEEE International*. IEEE, 2010, pp. 252–258.

[29] J. A. Roy, F. Koushanfar, and I. L. Markov, "Epic: Ending piracy of integrated circuits," in *Proceedings of the conference on Design, automation and test in Europe*. ACM, 2008, pp. 1069–1074.

[30] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, "Security analysis of logic obfuscation," in *Proceedings of the 49th Annual Design Automation Conference*. ACM, 2012, pp. 83–89.

[31] S. M. Awan, S. Rashid, M. Gao, and G. Qu, "Security through obscurity: Integrated circuit obfuscation using don't care conditions," in *Control, Automation and Information Sciences (ICCAIS), 2016 International Conference on*. IEEE, 2016, pp. 64–69.

[32] Y. Alkabani, F. Koushanfar, and M. Potkonjak, "Remote activation of ics for piracy prevention and digital right management," in *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*. IEEE Press, 2007, pp. 674–677.

[33] A. Barenghi, L. Breveglieri, I. Koren, and D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," *Proceedings of the IEEE*, vol. 100, no. 11, pp. 3056–3076, 2012.

[34] C. H. Kim and J.-J. Quisquater, "Faults, injection methods, and fault attacks," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 544–545, 2007.

[35] K. Rothbart, U. Neffe, C. Steger, R. Weiss, E. Rieger, and A. Mühlberger, "High level fault injection for attack simulation in smart cards," in *Test Symposium, 2004. 13th Asian*. IEEE, 2004, pp. 118–121.

[36] A. R. Desai, M. S. Hsiao, C. Wang, L. Nazhandali, and S. Hall, "Interlocking obfuscation for anti-tamper hardware," in *Proceedings of the Eighth Annual Cyber Security and Information Intelligence Research Workshop*. ACM, 2013, p. 8.

[37] J. Dofe, Y. Zhang, and Q. Yu, "Dsd: a dynamic state-deflection method for gate-level netlist obfuscation," in *VLSI (ISVLSI), 2016 IEEE Computer Society Annual Symposium on*. IEEE, 2016, pp. 565–570.

[38] M. Yasin, B. Mazumdar, J. J. Rajendran, and O. Sinanoglu, "Sarlock: Sat attack resistant logic locking," in *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 236–241.

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

[40] M. El Massad, S. Garg, and M. V. Tripunitara, "Integrated circuit (ic) decamouflaging: Reverse engineering camouflaged ics within minutes." in *NDSS*, 2015.

[41] A. B. Kahng, S. Mantik, I. L. Markov, M. Potkonjak, P. Tucker, H. Wang, and G. Wolfe, "Robust ip watermarking methodologies for physical design," in *Proceedings of the 35th annual Design Automation Conference*. ACM, 1998, pp. 782–787.

[42] A. E. Caldwell, H.-J. Choi, A. B. Kahng, S. Mantik, M. Potkonjak, G. Qu, and J. L. Wong, "Effective iterative techniques for fingerprinting design ip," in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*. ACM, 1999, pp. 843–848.

[43] F. Koushanfar and G. Qu, "Hardware metering," in *Proceedings of the 38th annual design automation conference*. ACM, 2001, pp. 490–493.

[44] F. Koushanfar, "Integrated circuits metering for piracy protection and digital rights management: An overview," in *Proceedings of the 21st edition of the great lakes symposium on Great lakes symposium on VLSI*. ACM, 2011, pp. 449–454.

[45] A. Baumgarten, M. Steffen, M. Clausman, and J. Zambreno, "A case study in hardware trojan design and implementation," *International Journal of Information Security*, vol. 10, no. 1, pp. 1–14, 2011.

[46] Y. Jin, N. Kupp, and Y. Makris, "Experiences in hardware trojan design and implementation," in *Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on*. IEEE, 2009, pp. 50–57.

[47] M. Beaumont, B. Hopkins, and T. Newby, "Hardware trojans-prevention, detection, countermeasures (a literature review)," DTIC Document, Tech. Rep., 2011.

[48] *https://www.trust-hub.org/*.

[49] S. K. Haider, C. Jin, and M. van Dijk, "Advancing the state-of-the-art in hardware trojans design," *arXiv preprint arXiv:1605.08413*, 2016.

[50] C. Lamech, R. M. Rad, M. Tehranipoor, and J. Plusquellic, "An experimental analysis of power and delay signal-to-noise requirements for detecting trojans and methods for achieving the required detection sensitivities," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 3, pp. 1170–1179, 2011.

[51] J. Li and J. Lach, "At-speed delay characterization for ic authentication and trojan horse detection," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*. IEEE, 2008, pp. 8–14.

[52] M. Banga and M. S. Hsiao, "Trusted rtl: Trojan detection methodology in pre-silicon designs," in *Hardware-Oriented Security and Trust (HOST), 2010 IEEE International Symposium on*. IEEE, 2010, pp. 56–59.

[53] J. Couch, E. Reilly, M. Schuyler, and B. Barrett, "Functional block identification in circuit design recovery," in *Hardware Oriented Security and Trust (HOST), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 75–78.

[54] R. S. Chakraborty and S. Bhunia, "Security against hardware trojan through a novel application of design obfuscation," in *Proceedings of the 2009 International Conference on Computer-Aided Design*. ACM, 2009, pp. 113–116.

[55] R. S. Chakraborty, F. Wolff, S. Paul, C. Papachristou, and S. Bhunia, "Mero: A statistical approach for hardware trojan detection," in *Cryptographic Hardware and Embedded Systems-CHES 2009*. Springer, 2009, pp. 396–410.

[56] J. Zhang, F. Yuan, and Q. Xu, "Detrust: Defeating hardware trust verification with stealthy implicitly-triggered hardware trojans," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 153–166.

[57] F. Koushanfar and A. Mirhoseini, "A unified framework for multimodal submodular integrated circuits trojan detection," *IEEE Transactions on Information Forensics and Security*, vol. 6, no. 1, pp. 162–174, 2011.

[58] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*. IEEE, 2008, pp. 51–57.

[59] X. Wang, M. Tehranipoor, and J. Plusquellic, "Detecting malicious inclusions in secure hardware: Challenges and solutions," in *Hardware-Oriented Security and Trust, 2008. HOST 2008. IEEE International Workshop on*. IEEE, 2008, pp. 15–19.

[60] W. Li, A. Gascon, P. Subramanyan, W. Y. Tan, A. Tiwari, S. Malik, N. Shankar, and S. A. Seshia, "Wordrev: Finding word-level structures in a sea of bit-level gates," in *Hardware-Oriented Security and Trust (HOST), 2013 IEEE International Symposium on*. IEEE, 2013, pp. 67–74.

[61] S. Areibi and A. Vannelli, "Tabu search: A meta heuristic for netlist partitioning," *VLSI Design*, vol. 11, no. 3, pp. 259–283, 2000.

[62] W. L. Buntine, L. Su, A. R. Newton, and A. Mayer, "Adaptive methods for netlist partitioning," in *Proceedings of the 1997 IEEE/ACM international conference on Computer-aided design*. IEEE Computer Society, 1997, pp. 356–363.

[63] Y.-Y. Dai and R. K. Brayton, "Circuit recognition with deep learning."

[64] K. S. McElvain, "Methods and apparatuses for automatic extraction of finite state machines," Jan. 30 2001, uS Patent 6,182,268.

[65] A. Lancichinetti, S. Fortunato, and J. Kertész, "Detecting the overlapping and hierarchical community structure in complex networks," *New Journal of Physics*, vol. 11, no. 3, p. 033015, 2009.

[66] L. Danon, A. Diaz-Guilera, J. Duch, and A. Arenas, "Comparing community structure identification," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2005, no. 09, p. P09008, 2005.

[67] T. Meade, Y. Jin, M. Tehranipoor, and S. Zhang, "Gate-level netlist reverse engineering for hardware security: Control logic register identification," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 1334–1337.

[68] W. Imrich, "Factoring cardinal product graphs in polynomial time," *Discrete Mathematics*, vol. 192, no. 1-3, pp. 119–144, 1998.

[69] Oregano Systems, "8051 IP core," http://www.oreganosystems.at/?page_id=96.

[70] D. Sullivan, J. Biggers, G. Zhu, S. Zhang, and Y. Jin, "Fight-metric: Functional identification of gate-level hardware trustworthiness," in *Proceedings of the 51st Annual Design Automation Conference*. ACM, 2014, pp. 1–4.

[71] R. Tarjan, "Depth-first search and linear graph algorithms," *SIAM journal on computing*, vol. 1, no. 2, pp. 146–160, 1972.